UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Posture control of a low-cost commercially available hexapod robot for uneven terrain locomotion

Mayur Tikam
10055747

Dissertation submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Mechanical Engineering)

Department of Mechanical and Aeronautical Engineering
Faculty of Engineering, Built Environment and Information Technology
UNIVERSITY OF PRETORIA

Supervisor: Prof. Nico J. Theron
Co-supervisor: Dr Dan Withey

May 2018

Abstract

Legged robots hold the advantage on uneven and irregular terrain, where they exhibit superior mobility over other terrestrial, mobile robots. One of the fundamental ingredients in achieving this exceptional mobility on uneven terrain is posture control, also referred to as attitude control. Many approaches to posture control for multi-legged robots have been taken in the literature; however, the majority of this research has been conducted on custom designed platforms, with sophisticated hardware and, often, fully custom software. Commercially available robots hardly feature in research on uneven terrain locomotion of legged robots, despite the significant advantages they pose over custom designed robots, including drastically lower costs, reusability of parts, and reduced development time, giving them the serious potential to be employed as low-cost research and development platforms. Hence, the aim of this study was to design and implement a posture control system on a low-cost, commercially available hexapod robot – the PhantomX MK-II – overcoming the limitations presented by the lower cost hardware and open source software, while still achieving performance comparable to that exhibited by custom designed robots.

For the initial controller development, only the case of the robot standing on all six legs was considered, without accounting for walking motion. This Standing Posture Controller made use of the Virtual Model Control (VMC) strategy, along with a simple foot force distribution rule and a direct force control method for each of the legs, the joints of which can only be position controlled (i.e. they do not have torque control capabilities). The Standing Posture Controller was experimentally tested on level and uneven terrain, as well as on a dynamic balance board. Ground truth measurements of the posture during testing exhibited satisfactory performance, which compared favourably to results of similar tests performed on custom designed platforms.

Thereafter, the control system was modified for the more general case of walking. The Walking Posture Controller still made use of VMC for the high-level posture control, but the foot force distribution was expanded to also account for a tripod of ground contact legs during walking. Additionally, the foot force control structure was modified to achieve compliance control of the legs during the swing phase, while still providing direct force control during the stance phase, using the same overall control structure, with a simple switching strategy, all without the need for torque control or modification of the motion control system of the legs, resulting in a novel foot force control system for low-cost, legged robots. Experimental testing of the Walking Posture Controller, with ground truth measurements, revealed that it improved the robot's posture response by a small amount when walking on flat terrain, while on an uneven terrain setup the maximum roll and pitch

angle deviations were reduced by up to 28.6% and 28.1%, respectively, as compared to the uncompensated case. In addition to reducing the maximum deviations on uneven terrain, the overall posture response was significantly improved, resulting in a response much closer to that observed on flat terrain, throughout much of the uneven terrain locomotion.

Comparing these results to those obtained in similar tests performed with more sophisticated, custom designed robots, it is evident that the Walking Posture Controller exhibits favourable performance, thus fulfilling the aim of this study.

Keywords: Legged robots, hexapod robot, posture control, Virtual Model Control, force control, uneven terrain locomotion

Acknowledgements

I would like to thank my project supervisor, Prof. N. J. Theron, for being willing to take on a project outside of his regular field of research and for providing guidance and insights throughout the course of the project. In addition, I would like to express my gratitude towards my co-supervisor, Dr D. Withey, whose willingness to share his knowledge and expertise, as well as spending tireless hours sharing ideas and reviewing my documentation, are truly appreciated.

This research would not have been possible without funding from the TSO unit at the CSIR – I sincerely appreciate the financial support received from them, as well as the moral support from the management and technical assistance from many of the staff members, with various aspects of design and manufacturing required in this project. A noteworthy acknowledgement should also be made to Estelle Lubbe, for sharing the insights she gained throughout her research work with the PhantomX platform, as well as for converting her kinematic model algorithm from MATLAB to C++.

I am also grateful to the MIAS group at the CSIR, for allowing me to make use of their facility and resources, and to staff members at MIAS for their assistance with the Vicon motion capture system and gathering data during experimental testing.

Other individuals I would like to thank include Kevin Ochs, for being willing to share his hexapod locomotion software (before making it publicly available) and providing useful information for getting acquainted with the code base, as well as Corne Gouws at NMISA, for taking out the time to assist with the testing of force sensors. My thanks also go out to Malcolm Pandaram, for helping me to get started on working with Linux and coding in C++, and to Kriven Naidoo for his assistance during experimental testing.

Finally, this dissertation is dedicated to my mother, Daksha Tikam, without whose continuous inspiration, encouragement, and sacrifices, I would never have had the opportunity to pursue my studies as far as I have.

Table of Contents

Lists of Figures

List of Tables

Nomenclature

List of Acronyms

3D	Three-dimensional
AHRS	Attitude Heading Reference System
API	Application programming interface
BLDC	Brushless direct current
CAD	Computer-aided design
CMU	Carnegie-Mellon University
CNN	Cellular non-linear network
COM	Centre of mass
CPG	Central pattern generator
CR200	Crabster200
CSIR	Council for Scientific and Industrial Research (in South Africa)
DAQ	Data acquisition
DH	Denavit-Hartenberg
DOF	Degrees of freedom
DPSS	Defence, Peace, Safety and Security
EEPROM	Electrically erasable programmable read-only memory
FRF	Frequency response function
GPS	Global positioning system
GUI	Graphical user interface
HD	Harmonic Drive
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IK	Inverse kinematics
IMU	Inertial measurement unit
IR	Infrared
IROS	IEEE/RSJ International Conference on Intelligent Robots and Systems
LIDAR/LiDAR	Light Detection and Ranging, or Light Radar
LiPo	Lithium polymer
LQR	Linear quadratic regulator
LRF	Laser rangefinder
MIAS	Mobile Intelligent Autonomous Systems

MMC	Motor Map Controller
NMISA	National Metrology Institute of South Africa
NUKE	Nearly Universal Kinematics Engine
ODV	OptoForce Data Visualization
PCFDC	Posture Control strategy based on Force Distribution and Compensation
PD	Proportional-derivative
PI	Proportional-integral
PID	Proportional-integral-derivative
PS3	PlayStation® 3
PS4	PlayStation® 4
PSD	Power Spectral Density
RAM	Random-access memory
RMS	Root-mean-square
ROS	Robot Operating System
ROV	Remotely operated vehicle
RSJ	Robotics Society of Japan
SEA	Series elastic actuator
SHSC	Sky-Hook Suspension Control
SLAM	Simultaneous Localization and Mapping
TSO	Technology for Special Operations
TTL	Transistor-transistor logic
UAV	Unmanned aerial vehicle
USB	Universal Serial Bus
VMC	Virtual Model Control(ler)
YAML	YAML Ain’t Markup Language (recursive acronym)

List of Symbols

Symbol	Description	SI Unit
English Letters and Symbols		
$\mathbf{0}$	3 × 3 zero matrix	–
$\mathbf{0}_{m\times n}$	$m \times n$ zero matrix	–
$\boldsymbol{A}$	Coefficient matrix of state vector in state-space model	–
$\boldsymbol{B}$	Coefficient matrix of control input vector in state-space model	–
C	Damping coefficient	N·s/m
$\boldsymbol{C}$	Derivative gain matrix	–
D	Derivative control action	–
e	Error signal	–
$\boldsymbol{e}$	Error state vector	–
$e_{c,i}$	Compliance error of the ith foot	m
$e_{f,i}$	Force error of the ith foot	N
$e_{s,i}$	Tracking error of the ith foot	m
f_c	Filter cut-off frequency	Hz
$f_{i,z}$	z-component of contact force at the ith foot	N
$f_{m,i}$	z-component of the force measured on the ith foot	N
f_s	Static force	N
$f(t)$	Function of time, t	–
$\boldsymbol{F}_{i,z}$	n×1 vector of z-components of n foot contact forces	N
F_{ff}	Feedforward control action	–
$\boldsymbol{F}_s$	Feedforward term in virtual model controller	–
$F(s)$	Function of Laplace variable, s	–
F_{VMC}	Vertical force commanded by virtual model controller	N
$\boldsymbol{F_{VMC}}$	Virtual model controller output vector	–
F_z	Force in the z-direction	N
g	Gravitational acceleration constant (9.81)	m/s^2
G	Centre of mass of a rigid body	–
$G(s)$	Transfer function	–
$H(s)$	Transfer function of first-order filter	–

I	Moment of inertia of a rigid body	kg·m^2
$\boldsymbol{I}$	3 × 3 identity matrix	–
I_i	Integral control action for ith foot	m
k	Order of state-space system	–
K	Spring stiffness coefficient	N/m
$\boldsymbol{K}$	Proportional gain matrix	–
K_C	Compliance gain of Single Leg Force Controller	1/s
K_d	Derivative gain	–
K_I	Integral gain of Foot Force Controller	m/N·s
K_p	Proportional gain	–
K_P	Proportional gain of Foot Force Controller	m/N
$K_{S,i}$	Switching gain of Single Leg Force Controller for the ith leg	–
$\mathcal{L}\{f(t)\}$	Laplace transform of $f(t)$	–
M	Mass of rigid body	kg
$\boldsymbol{M}$	Rotation matrix returned by IMU	–
M_θ	Pitch moment commanded by virtual model controller	N·m
M_ϕ	Roll moment commanded by virtual model controller	N·m
n	Number of feet in contact with the ground	–
N	Number of legs on the robot	–
N_f	Parameter defining derivative filter cut-off frequency	rad
$\boldsymbol{p}$	Position vector	m
$\Delta p_{i,z}$	Adjustment value for z-position of the ith foot (single leg force controller output)	m
P	Proportional control action	–
$\boldsymbol{P}$	Matrix representation of contact foot positions for force-torque equilibrium equations	–
q_0	Scalar part of quaternion	–
$\boldsymbol{q}$	Quaternion vector	–
$\overline{\boldsymbol{q}}$	Imaginary part of quaternion	–
$\boldsymbol{Q}$	Controllability test matrix	–
r	Reference input signal for controller	–
$r(\cdot)$	Rank of a matrix	–
$\boldsymbol{R}$	Rotation matrix	–
$\mathbb{R}^3$	Three-dimensional space	–

s	Laplace variable	rad/s
t	(Continuous) Time variable	s
t^*	Time instant	s
t_k	Sampling instant (discrete time variable)	s
Δt_k	Duration between previous and current sampling instants	s
T	Cycle time	s
T_d	Derivative time constant	s
T_{st}	Stance phase duration	s
T_{sw}	Swing phase duration	s
T_t	Tracking time constant	s
u	Control signal	–
$\boldsymbol{u}$	Control input vector in state-space model	–
$\boldsymbol{u}_{IMU}$	Euler angle vector describing rotation of IMU	rad
$\boldsymbol{u}_{ZYX}$	Euler angle vector (using ZYX rotation sequence)	rad
$\boldsymbol{x}$	State vector in state-space model	–
$\boldsymbol{x}^*$	Arbitrary state	–
$x_{d,i}$	x-component of position commanded by gait engine for foot i	m
y	Measured input signal for controller	–
$y_{d,i}$	y-component of position commanded by gait engine for foot i	m
$Y(s)$	Laplace transform of controller input signal	–
$\boldsymbol{z}$	Vector in space	–
$z_{d,i}$	z-component of position commanded by gait engine for foot i	m
Δz_i	Actual, commanded adjustment value for z-position of the ith foot (after applying artificial saturation)	m
Δz_{lim}	Foot force controller output saturation limit	m
$z_{r,i}$	z-component of position reference for inverse kinematics engine, for foot i	m
$z_{t,i}$	z-component of temporary position reference, for foot i	m
$\mathbb{Z}$	Set of all integer numbers	–

Greek Letters and Symbols		
β	Duty factor	–
θ	Pitch angle	rad
μ	Mean	–
σ	Standard deviation	–
τ_c	Filter time constant	s/rad
ϕ	Roll angle	rad
ψ	Yaw angle	rad
ω_c	Filter cut-off frequency	rad/s
Superscripts		
$'$	Relative to body coordinate system	
-1	Inverse (of a matrix or vector)	
T	Transpose (of a matrix or vector)	
Subscripts		
avg	Average (divided among a certain number of legs)	
b	Body coordinate system	
d	Desired value (control setpoint)	
f	Filtered	
G	Of the Centre of Mass	
i	Of the ith foot, or general index	
j	General index	
IMU	Of the Inertial Measurement Unit	
w	World coordinate system	
x	In the x-direction or about the x-axis	
y	In the y-direction or about the y-axis	
z	In the z-direction or about the z-axis	
θ	In the pitch direction or about the pitch axis	
ϕ	In the roll direction or about the roll axis	
Accent Characters		
$\dot{}$	First time derivative (typically yielding velocity)	
$\ddot{}$	Second time derivative (typically yielding acceleration)	
$\hat{}$	Unit vector	

Chapter 1: Introduction

Legged robots constitute an important research field as they introduce the capability of uneven terrain mobility to robotic systems. Lowering the barriers to entry in this field has the potential to improve the progress of legged robot capabilities and real-world applicability. Therefore, this study deals with the development and implementation of a posture control system to improve uneven terrain locomotion of a hexapod robot, specifically on a low-cost, commercially available platform, as platforms such as this present promise for low-cost research and development platforms.

In the sections of this chapter which follow, brief background information will be given and the motivation for the study will be elaborated. In addition, an overview of the methodologies and scope of the study will be provided, along with validation and verification methods, contributions made by the study, and a glimpse into the content presented throughout the remainder of this dissertation.

1.1 Background

In the world of mobile robotics, legged robots hold the advantage on uneven and irregular terrain. Their ability to walk with discrete ground contact points grants them access to environments and terrain which most other vehicles, such as wheeled and tracked robots, are unable to traverse [1]–[4]. Various types of legged robots exist, ranging from those with two legs, known as bipeds, to ones that have six (hexapods) or more legs [3]. Quadrupeds (four-legged robots) and hexapods are commonly used because of their capacity for static stability, with hexapod robots exhibiting superior efficiency with regard to statically stable walking [5].

One of the key ingredients in achieving locomotion on uneven and irregular terrain, that has been identified in much of the literature [1], [6]–[10], is posture control (also referred to as attitude control), with particular emphasis on controlling the height of the robot's body above the ground, along with its orientation relative to the horizontal plane, in the form of roll and pitch angles [4], [6], [10]. A

selection of posture control implementations exist on quadruped and hexapod platforms, with one of the most widely used methods being Virtual Model Control (VMC) [4], [6], [7], [10]–[12].

However, most of this research takes place on custom designed platforms, with sophisticated (and consequently expensive) hardware, sensors, and actuators (often with torque control capabilities), e.g. [4], [6], [10], or even complex, over-actuated leg structures, e.g. [9], [13]. Additionally, many of these platforms run on custom software, written from the ground up.

Commercially available legged robot platforms, on the other hand, pose many advantages over their custom designed cousins, the foremost being radically lower costs. Other primary advantages of these platforms include reusability of parts and reduced development time. Commercial legged robots are also becoming more widely available, especially hexapods, which are popular among hobbyists because of their inherent stability [3]. One of the main shortcomings of such platforms is that most of the low-cost servo motors they use as actuators do not provide torque control capabilities, meaning they cannot be used with some of the control techniques commonly employed on more sophisticated platforms in the literature. Additionally, low-cost actuators, hardware, and sensing equipment generally provide lower accuracy and are manufactured to lower tolerances than their more expensive counterparts, which could contribute to larger control errors and, consequently, degraded performance.

Recently, the Technology for Special Operations (TSO) unit, at the Defence, Peace, Safety and Security (DPSS) division of the Council for Scientific and Industrial Research (CSIR) in South Africa, have sought to embark on research in the area of legged robotics, in order to investigate the use of such platforms for terrestrial locomotion in real-world, field use. An initial study in this respect has been conducted in [14], wherein the low-cost, PhantomX MK-II hexapod platform [15] was acquired and outfitted with a proprioceptive sensory system. This platform was also utilized to gather data that was used to test a full state estimation algorithm. The fundamental aim of [14] was to provide a starting point for future studies on the control of these types of robotic platforms, for uneven terrain locomotion.

1.2 Problem Statement

It is important to explore the viability of uneven terrain locomotion on commercially available platforms with lower costs, especially to enable wider spread applicability and adoption of legged robot technology outside of the lab. Despite the notable advantages offered by commercially available platforms, custom designed platforms are, by far, the most common in literature, with commercial platforms hardly featuring in posture control development. Furthermore, commercial platforms do not feature built-in posture control capabilities in their standard locomotion software.

With posture control being such a vital element for achieving useful uneven terrain locomotion, as discussed in the previous section, the development and testing of a posture control system on a commercially available legged robot, is required.

1.3 Aim

The aim of this study is to demonstrate the implementation of a posture control system on the low-cost, commercially available hexapod robot acquired by TSO. To this end, the design and development of the control system will need to be adapted to overcome the limitations presented by the commercially available robotic platform (especially the joint actuators which can only be position controlled, having no torque control capability) and the available locomotion software. Despite these adaptations, the design should attempt to still achieve performance comparable to that displayed on more sophisticated, custom designed platforms.

1.4 Methodology

Before commencing with development of a posture control system, existing methods utilized in the literature will be thoroughly investigated, while also identifying the relevant background and theory pertinent to this study. Thereafter, appropriate upgrades will be made to the existing hexapod robot platform, in terms of both hardware and software, while staying true to the philosophy of using low-cost components.

In addition, preliminary analyses required for controller development will be carried out, along with the derivation of methods for determining the actual posture of the robot, based on the available sensory information.

This will be followed by the development and implementation of a posture control system, for the simpler case of the hexapod standing on all six of its legs, rather than immediately dealing with the complexities involved in walking on uneven terrain. The performance of the implemented control system will be experimentally tested, to gain further insights and form a basis for implementation during walking.

Finally, the control system will be modified and expanded for the more general case of walking on uneven terrain. An artificial uneven terrain setup will be manufactured, for use in experimental tests which will be utilized to validate the performance and effectiveness of the implemented posture control method.

1.5 Scope

In carrying out the abovementioned methodology to fulfil the aim of this study, the following aspects will form part of the scope of the project:

- Incorporation of the necessary sensors, electronic equipment, and hardware on the hexapod platform, required for the purpose of posture control;
- Updating parameters and settings in the selected locomotion software to get the platform running on this software;
- Minor modifications to the locomotion software to incorporate the implemented control system;
- Design and development of a control system to regulate the robot body's posture, in the form of height and orientation;
- Digital implementation of the designed control system, by writing the relevant control software;
- Manufacturing of a simple test setup that emulates uneven terrain conditions on which the hexapod can walk; and
- Experimental testing of the implemented control systems, as well as processing and analysis of the test data.

This study is focussed specifically on the development and implementation of the first iteration of a posture control system for the hexapod platform being used, to serve as a proof of concept and a foundation for future studies to expand on this work, for more advanced implementations. Therefore, the following will **not** form part of the project scope:

- Major physical modifications to the hexapod platform, or detailed investigations on factors such as its kinematics, dynamics, and energy efficiency, among others;
- Development and incorporation of an advanced, exteroceptive sensory system;
- Major modifications to the locomotion software or implementation of more gaits in the same software, as well as writing of custom locomotion software or sensor drivers;
- Further development or implementation of a full state estimation algorithm for the platform;
- Development of higher level tasks of control, planning, or autonomy, beyond posture control;
- Optimization of the gains and parameters used in the designed control system; and
- Development of standardized terrain setups and testing procedures for legged robots, or methods for accurately characterizing and describing terrains so they can be reproduced in comparative studies.

1.6 Validation and Verification Methods

Accuracy and precision of the measurements provided by new foot force sensors, incorporated onto the hexapod, will be validated through simple dead weight tests performed with the sensors. The measured data will be processed, analysed, and compared to the actual loads induced on the sensors, to determine whether the force values they measure (calculated from the raw data using the manufacturer provided calibration factors) are within an acceptable tolerance range.

With regard to the designed posture controller, selected components of the control system will be mathematically analysed for a variety of scenarios, to verify that the proposed control algorithm will, in fact, produce the desired behaviour. An example of this is the analysis of the steady state behaviour of the designed Foot Force Controller, to verify that it does provide anti-windup and active compliance functionalities, as desired (see Chapters 6 and 7).

Furthermore, experimental testing will be used to validate the performance of the implemented control system, as well as its ability to improve the dynamic behaviour of the robot on both flat and uneven terrain conditions. In order to ensure that the posture measurements calculated from sensor data are valid, they will be compared to ground truth measurements taken by a Vicon motion capture system [16], during experimental testing.

1.7 Contributions of the Study

The following contributions will be made by the successful completion of this study:

1. Design of new feet (which can be easily replicated and manufactured) to incorporate OptoForce 3D force sensors onto the PhantomX MK-II hexapod robot platform, for research and development.
2. Upgrade of the open source `hexapod_ros` locomotion software [17] to repeatedly read the position feedback from the robot's servo motors during locomotion. This can potentially be contributed back to the Robot Operating System (ROS) community.
3. Implementation of a Virtual Model Controller on a low-cost, commercially available hexapod, taking into account the robot's dynamic behaviour. To the best of the author's knowledge, this has only been previously implemented on custom designed and manufactured platforms.
4. Presentation of a simple foot force distribution rule, that can be used on hexapod robots walking with a tripod gait, without requiring complex optimization problems to be repeatedly solved during locomotion.
5. Implementation of a direct force control method on the stance legs of a robot, to enable the control of contact forces (for posture control) using low-cost joint actuators, with only position control capabilities (no direct torque control). Furthermore, the method does not require

complex control structures with online terrain estimation features, such as the method in [18] which attempts to control contact forces through position-based control of the leg.

6. Development of a new force control system that achieves both direct force control (for stance legs) and active compliance control (for swing legs), i.e. both direct and indirect force control actions for interaction tasks, using the same general control structure and a simple switching technique when changing between leg phases, all without the need for torque control. To the best of the author's knowledge, such a force control formulation has not been previously developed or implemented on a legged robot.
7. Investigation into existing uneven terrain testing methods for legged robots, and identification of shortcomings in these methods, along with suggestions for future studies to be conducted in this area. Additionally, an uneven terrain setup will be created that can be used in later studies with the hexapod platform (or other small legged robot platforms), while also being comprehensively reported on to allow for easy replication by other researchers for their experimental tests (although this does not constitute a standardized terrain setup nor testing procedure for legged robots on uneven terrain).
8. Experimental testing of a fully implemented posture control system on the PhantomX MK-II platform, to formally demonstrate the effectiveness of the methods when used on a low-cost, commercially available legged robot platform.

Contributions 3 and 5, as well as part of 4, in the list above, are included in a paper that was published in [19] and presented at the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017), at Vancouver, Canada, in September 2017.

1.8 Dissertation Overview

Following this Introduction, Chapter 2 of the dissertation presents comprehensive background information on legged robots, in the greater context of the field of robotics, familiarizing the reader with important considerations and terminology germane to this study and the discussions presented in subsequent chapters. The chapter then delves further into the matter of control of legged robots for uneven terrain locomotion, providing a thorough review of the literature available on posture control of quadruped and hexapod robots.

For a more detailed coverage of specific background theory required in later discussions, Chapter 3 elaborates further on topics such as hexapod robots and interaction control methods. In addition, particular terminology and technical information, which are utilized throughout the dissertation, are provided, including that related to transformation mathematics, virtual modelling of a hexapod, and the meta-operating system used for implementing the control software developed in this study.

Chapter 4 then proceeds to present the PhantomX MK-II hexapod robot platform available for use in this study. An overview of the existing sensory system is given, while the chapter focusses more deeply on the new foot force sensors incorporated onto the platform, including the main findings of testing carried out with the sensors to validate the measurements they provide. Software utilized for locomotion and sensor data feedback are also discussed, serving as a backdrop for later discussions regarding the digital implementation of the designed control systems.

In Chapter 5, a simplified controllability analysis is carried out for the legged robot system, as a means of gaining useful insights into the conditions required for controlling the robot's body posture as desired, as well as scenarios which could lead to a loss of controllability during operation. Additionally, procedures for computing the robot's posture for the purpose of posture control are described, elaborating on the methods employed to obtain the necessary information from the data provided by the onboard sensors. Before proceeding with the discussions on controller development, some basic frequency domain concepts utilized in subsequent chapters are also introduced.

Thereafter, the development and implementation of the Standing Posture Control system, which only considers the case of the robot standing on all six legs, is covered in Chapter 6. Experimental tests, performed to assess the performance of this control system, are explained, followed by the presentation of the experimental results. These results are analysed and discussed in detail, with comparisons to results obtained on custom designed platforms in similar tests. The content from this chapter was published in [19].

Having gained valuable insights from the development and testing of the Standing Posture Control system, Chapter 7 examines the shortcomings of this controller, when walking is considered, and introduces modifications made to the control structure to overcome these inadequacies. For the purpose of testing the ensuing Walking Posture Control system, the development of an uneven terrain setup is briefly described, along with the experimental test procedure employed. Once again, the experimental results are presented and discussed, placing them in context with the literature by comparisons to results reported for tests conducted on custom designed hexapods.

In conclusion, Chapter 8 summarizes the methods utilized in the study and reiterates the main findings drawn from analysing the results. Suggestions and recommendations for improvements are also presented, opening the door for future development at the close of this study.

Chapter 2: Literature Review

To begin with, this chapter presents a review of the field of robotics and the place of legged robotics within the field. It also serves to introduce the reader to essential concepts and terminology pertinent to this study, which will be used throughout this dissertation. The review is then directed more specifically toward control considerations for legged robots walking on uneven terrain, ultimately investigating one of the central requirements for uneven terrain mobility – posture control. In particular, existing methods of posture control available in the literature, on quadruped and hexapod robots, are reviewed and compared, rationalizing the motivation for this study and divulging a clear choice for the control method to be implemented.

2.1 The Field of Robotics

Robotics is a rather wide field of engineering with various categories and areas of application. Autonomous robotic systems are generally classified into two major categories, namely *manipulation robotics* and *mobile robotics* [2]. Manipulators, also referred to as robot arms, are primarily used in the domain of industrial manufacturing, where they are fixed at the shoulder to a particular position in an assembly line and can perform repetitive tasks, such as spot welding and painting, with immense speed and accuracy [3] (see Figure 2.1). They are also extremely precise in positioning electronic components on circuit boards [3], particularly useful in the electronics industry where the use of robotic arms has made it possible to realize the production of devices such as the mobile telephone and laptop computer.

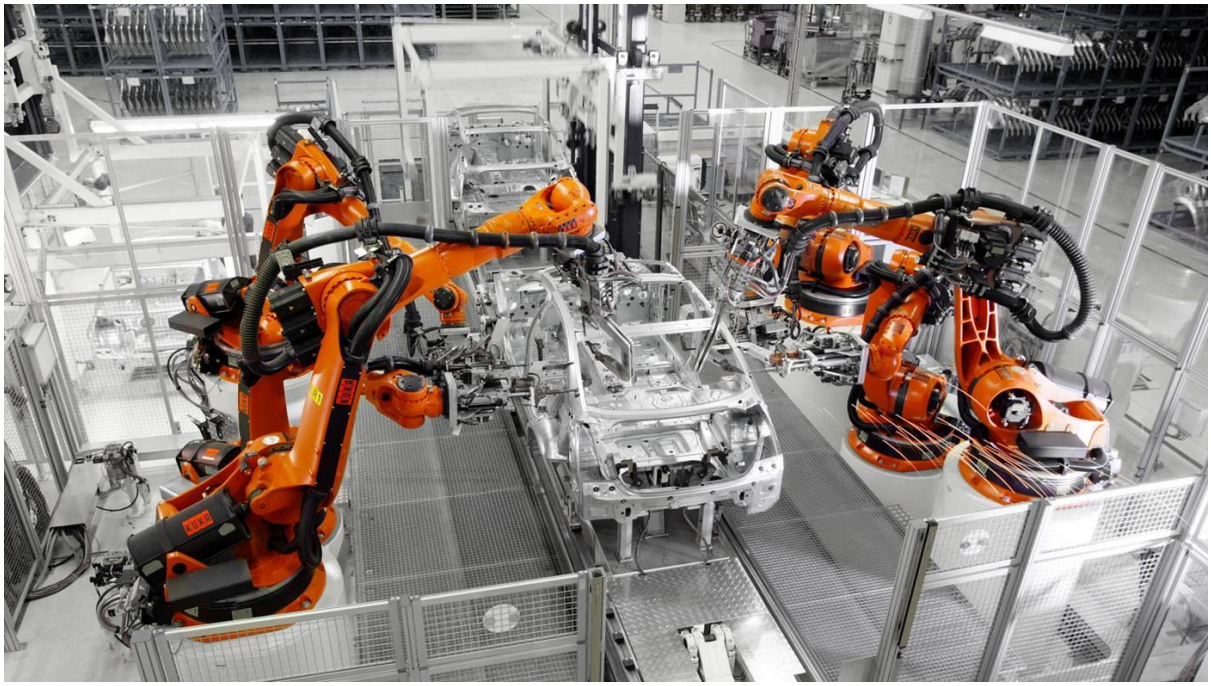


Figure 2.1: KUKA industrial robots in an automotive assembly line [20]

While manipulators are remarkably successful in these types of environments, they are fundamentally limited by their lack of mobility [3]. They have a restricted range of motion defined by their position and configuration and can thus only be used for very specific tasks in a specific location. Mobile robots, on the other hand, have the ability to travel within the environment they are in, "flexibly applying [their] talents wherever [they are] most effective" [3]. The following section delves further into the field of mobile robotics.

2.2 Mobile Robots

Mobility opens up a whole array of functionality and applications for robots. The ability to move around rather than carrying out tasks in one place allows mobile robots to be utilized in many different ways. Environments which are dangerous or inhospitable for humans (both on Earth and in space), for instance, can be accessed by mobile robots, in which case teleoperated systems are a popular choice, allowing a human operator to remotely perform localization and cognition activities, while the robot controls the low-level complexities associated with locomotion [3]. A fine example of this type of mobile robot application is NASA's Curiosity rover, sent to explore Mars and study the planet's habitability [21]. The self-portrait displayed in Figure 2.2 is a composite of dozens of component images taken by the rover using the camera at the end of its onboard robotic arm.

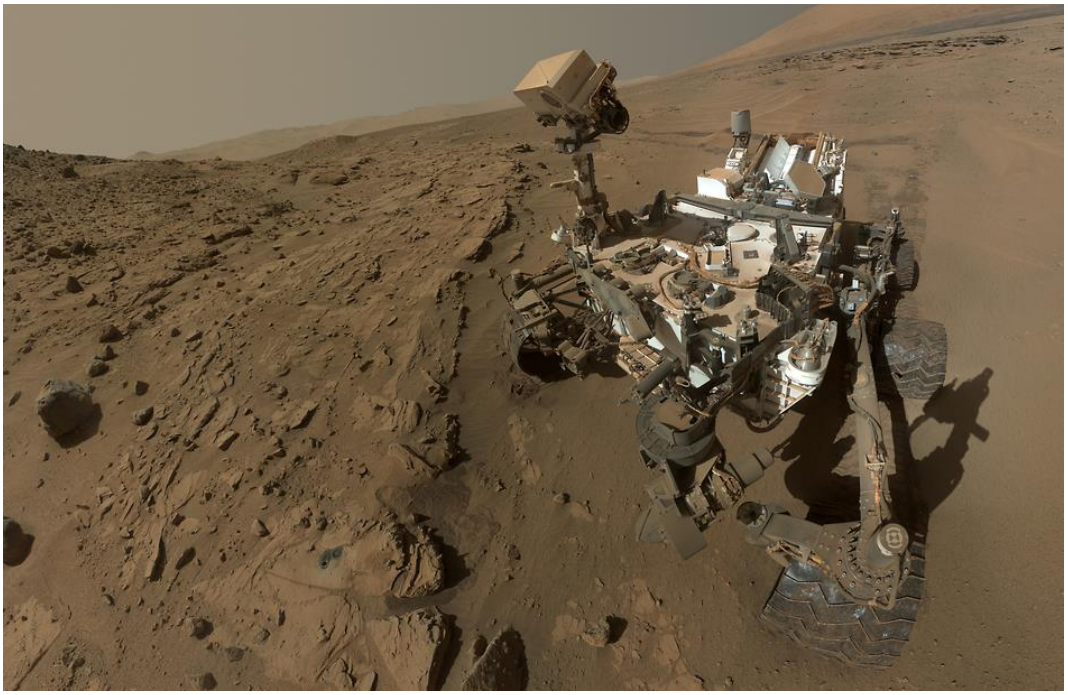


Figure 2.2: Self-portrait of NASA's Curiosity rover on the surface of Mars [21]

In more friendly environments, robots operate in the same space as humans and sometimes even interact with them. In these situations, their real advantage lies in their ability to function autonomously, in addition to their exceptional mobility, in which case it is vital that they are able to keep track of their position and navigate throughout their environment without human intervention [3]. Tour guide robots are one such example, as depicted in Figure 2.3.

Figure 2.3: A tour guide robot at the Museum of the Great War in Meaux, France [22]

Equipped with these fantastic abilities, mobile robots have found employment in a vast assortment of occupations (more examples of which can be found at [3]), and the many different implementations of robots in particular applications are more diverse still.

While a multitude of examples of commercial and research based mobile robots could be provided, this discussion is more focussed on the various methods in which mobility is achieved in robotics. Central to this idea is the topic of locomotion mechanisms, discussed in more detail in the following section.

2.2.1 Types of Locomotion

A robot's locomotion mechanism essentially furnishes it with the ability to move around through its environment [3], making it mobile. As one would imagine, movement can be achieved in a large variety of ways. Designs of mobile robots are thus wide ranging, with many locomotion mechanisms drawing inspiration from biological creatures, who seem to achieve locomotion with such elegance and efficiency. There are robots in existence, especially in the research environment, which can move by walking, jumping, running, sliding, skating, swimming, flying, or even rolling [3].

Furthermore, robotic locomotion cannot be discussed without mention of the wheel. A human invention which has not been developed in nature – at least not in the form of a fully rotating, actively powered joint [3] – the wheel is one of the most prominent forms of locomotion today, not only in mobile robotics.

In fact, the mechanical complexity associated with most biologically inspired locomotion mechanisms renders them quite impractical for use outside of the research world, at least at the current stage of technological development. Machado and Silva [2] point out that there are three fundamental configurations of mobile robot locomotion on ground:

- Rotational devices, such as wheels and tracks;
- Legs, resembling those of animals and insects;
- Articulated structures, similar to the body of a snake.

It is important, at this stage, to mention that other forms of mobile robots do exist, including those that fly or swim, as well as autonomous vehicles which navigate through the air or water, such as unmanned aerial vehicles (UAVs). However, these will not be reviewed further, given that the focus of this study is on land-based locomotion.

Articulated body robots do present certain advantages in particular situations [2], where wheeled or legged robots would not do so well, but this type of locomotion introduces its own complexities.

Robots of this sort are not the main focus of this study either. Therefore, the following sections go into further detail on the subjects of wheeled and legged robots.

2.2.2 Wheeled Robots

It is easy to see why the use of wheels for vehicles and robots has been so prolific in civilization up to this point when one considers the mechanical simplicity with which wheels can be implemented, along with the exceptional efficiencies [3] and speed [2] which can be achieved with them.

However, in order to attain these high efficiencies and be effective, wheeled vehicles require predominantly regular and, in most cases, prepared surfaces, such as tarred or paved [2]. Flat and hard surfaces, such as railway tracks, minimize rolling friction and can be one or two orders of magnitude more efficient than other forms of locomotion [3], as depicted in Figure 2.4. The softer and less uniform the terrain though, the greater the rolling friction, leading to a rapid accumulation of inefficiencies. This is clearly represented in Figure 2.4 by the high power consumption required by a tyre on soft ground.

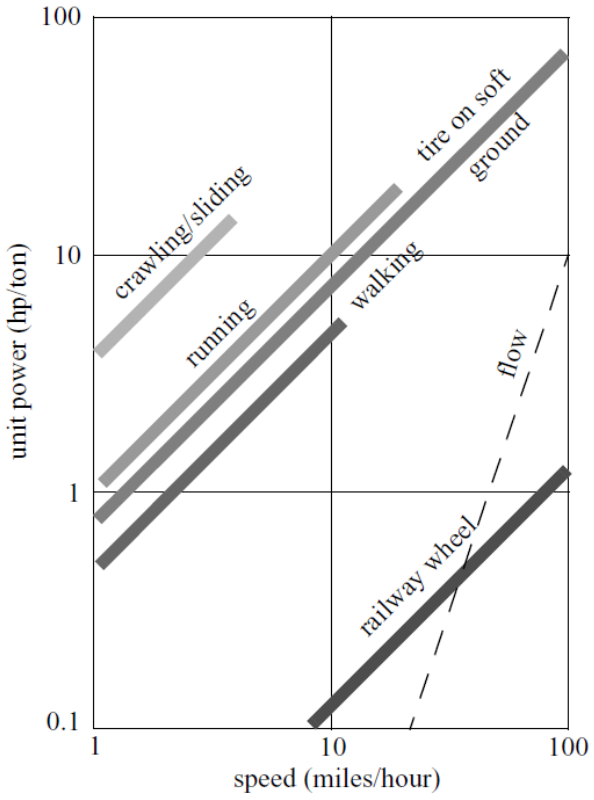


Figure 2.4: Specific power versus attainable speed for various methods of locomotion [3]

Another difficulty experienced by wheeled vehicles is the inability to traverse significantly sized obstacles and large surface irregularities [2]. All-terrain vehicles and tracked vehicles, such as tanks, present an alternative with increased mobility in irregular terrain. However, they are also only able to

surpass relatively small obstacles and still succumb to many of the challenges faced by conventional wheels, while consuming much larger amounts of energy as a result of the increased surface contact and rolling friction. In fact, Machado and Silva [2] found that research in 1967 estimated that more than 50% of the land surface of the Earth is inaccessible to traditional vehicles with wheels and tracks. It should also be noted that most wheeled vehicles require suspension systems for the wheels to maintain ground contact in the presence of terrain irregularities [3].

Why, then, are wheels utilized for locomotion in the majority of applications of ground-based vehicles and mobile robots? The answer to this question lies in the fact that the civilized world and, more specifically, the human environment in which most of these robots are designed to function consist largely of engineered and relatively smooth surfaces [3]. Some of them are even specifically designed for wheeled locomotion (asphalt roads, for example). As alluded to above, the wheel is yet to be surpassed, in terms of performance and efficiency, as a locomotion method in such environments.

Moreover, balance is generally not a significant problem with wheeled vehicles, as can often be the case with legged robots, since they are almost always designed to be statically stable (see Section 2.3.4) [3].

2.3 Legged Robots

It is on uneven terrain that legged robots really come into their own and display their true potential. In fact, Raibert [23] identified their increased mobility in rough terrain as one of the two most significant reasons for studying legged machines, the other being to improve on the understanding of human and animal locomotion.

The advantages exhibited by legged robots are wide-ranging. These will be elaborated below, followed by a discussion of some of their disadvantages.

2.3.1 Advantages of Legged Robots

Almost every source in the literature pertaining to research on legged robots mentions, even if at least briefly, the advantages they provide on uneven terrain. Siegwart and Nourbakhsh [3] concisely describe these key advantages as being increased adaptability and manoeuvrability on rough terrain.

Their superior mobility in natural terrains stems from the use of a discrete number of footholds or ground-contact points during locomotion, as opposed to requiring a continuous support surface, as is the case with wheeled and tracked vehicles [2], [3], [23], [24]. These ground-contact points can be selected and established in accordance with terrain conditions, while surface irregularities are adapted to by varying individual leg configurations [2].

Raibert [23] explains that legged systems can select their footholds in the terrain to be the best in their vicinity, while wheels are obliged to traverse the worst terrain. He illustrates this using a step ladder as an example, as a legged system can ascend using the rungs as footholds, while the ascent of a wheeled system is hindered by the spaces between the rungs. Siegwart and Nourbakhsh [3] allude to a similar point, stating that, as long as the robot maintains adequate ground clearance, the quality of the ground between the set of point contacts does not affect mobility. Raibert [23] further suggests that the individual footholds used by a legged robot can be selected with consideration for stability and traction.

In addition to providing improved mobility on uneven terrain, discrete footholds can also improve energy consumption on softer surfaces, such as sandy soil, as they deform the terrain less than wheels or tracks, consequently requiring less energy to get out of the depressions [2] (the reader is reminded of the elevated energy consumption of wheeled systems on soft ground, depicted in Figure 2.4 above). Moreover, [2] points out that the contact area between the foot and the ground can be constructed in a manner which reduces the ground support pressure.

Along with stepping over obstacles in rough terrain, legged robots can also cross holes or chasms, including those larger than their own body size, provided that their legs can achieve a reach greater than the width of the hole [3]. The reader will likely recognize that this is not usually possible with wheeled vehicles, which are often unable to cross significantly sized holes without falling or sinking into them. Even if only one section of the vehicle falls into a rather deep depression in the terrain, one or more of the wheels on the other side of the vehicle could lose contact with the ground, preventing sufficient traction to navigate back out of the chasm. Exhibiting mobility over natural obstacles or ditches, as well as man-made obstructions such as stairs, means that legged robots are able to operate in both structured and unstructured environments [24].

It was mentioned earlier that wheeled vehicles require suspension systems to traverse terrain irregularities. Legs, on the other hand, can provide an active suspension for the vehicle, decoupling the path of the body from the paths of the individual feet, as well as varying body height to dampen the effects of terrain irregularities on the body, thus allowing the payload to travel smoothly in spite of prominent terrain variations [2], [23]. Raibert [23] also asserts that, “in principle, the performance of legged vehicles can, to a great extent, be independent of the detailed roughness of the ground.” Gonzalez de Santos et al. [24] go further to explain that the ability of legged robots to keep their bodies levelled and maintain static stability while negotiating uneven terrain is essential for carrying equipment and sensors which would need to remain levelled whilst taking measurements.

Along with keeping onboard equipment and sensors levelled, the robot can impart additional translation and rotation motions to its body without changing its footholds, consequently providing additional degrees of freedom (DOF) for the onboard equipment when necessary [24]. For instance, when immobilized, the robot body can be actuated to act as an active support base for a manipulator arm or tool mounted onto it [2].

Alternatively, a legged robot could also utilize one (or more) of its own legs for manipulation tasks, as can be observed in many animals [2]. Many legged robots have multiple degrees of freedom in each leg, allowing them to manipulate objects in the environment with incredible dexterity [3].

Another benefit of having multiple degrees of freedom in the leg joints is that legged robots can change their direction of motion without any slippage [2], equipping them with inherent omni-directionality [24]. Their legs could also be used to hug themselves to the terrain or objects that they move on, as in the case, for example, of improving their balance when moving along the outside surface of pipes [2].

Having many legs also introduces the potential for failure tolerance. Unlike wheeled locomotion, in which the failure of one wheel usually results in a critical loss of mobility, legged robots may possess sufficient legs for redundancy, thus allowing them to maintain static balance and continue locomotion in the event of one or more of the legs incurring severe damage [2]. This is particularly true for robots with a large number of legs, such as six-legged [25] or eight-legged walking robots.

The topic of failure tolerance has started receiving more attention in robotics research relatively recently [2]. A fascinating demonstration of such research is presented by Cully et al. [26], who utilize learning algorithms to bestow robots with the ability to adapt like animals. Their research was carried out on both a robotic arm and a six-legged walking robot. Employing an approach which they refer to as “Intelligent Trial and Error,” their robots are able to conduct experiments using estimates of “learnt” behaviours and discover ways in which to compensate for damage (in the form of damaged legs or joints, for example). In many cases this method results in the robot realizing a behaviour with superior performance and efficiency compared to the default behaviour, in both damaged and undamaged conditions.

2.3.2 Disadvantages of Legged Robots

Despite their incredible array of abilities and advantages, legged robots are not infallible. The main disadvantages they are faced with include the need for complex control algorithms [2], as well as issues related to power consumption and mechanical complexity [3].

The point on the complexity of the control algorithms involved in achieving stable locomotion with skilful manoeuvrability is of vital importance and can be clearly observed in much of the literature pertaining to legged robots, where multi-layered and interlinked control schemes and algorithms are typically utilized. Just a few of the tasks addressed by the control systems of many research robots include terrain modelling, path planning, foothold selection, gait generation, feet trajectory planning, joint position control, posture and attitude control, stabilization, and more [8], [9], [25], [27], [28].

As indicated in [3], mechanical complexity arises because, in order to achieve the high manoeuvrability which legged robots are famous for, they require their legs to have sufficient degrees of freedom to impart forces in several different directions. These legs ought to also have the capability of supporting some of the total weight of the robot, as well as being able to lift and lower the robot in many cases.

Legs with multiple degrees of freedom usually require many actuators, and the mechanisms used in leg construction can be relatively heavy, both of which contribute to large energy consumption to operate satisfactorily [2]. Machado and Silva [2] also identified another notable limitation of legged robots as operating at comparatively lower speeds than other locomotion mechanisms.

2.3.3 Types of Legged Robots

Legged robots require a minimum of one leg to achieve locomotion. This result may be slightly surprising for the reader; however, one-legged robots have been developed and studied, with a certain amount of success. Siegwart and Nourbakhsh [3] highlight that having only one leg reduces the mass of the robot, eliminates the need for multi-leg coordination, and results in only one point contact with the ground, giving it the potential for use on the roughest terrain.

It was already mentioned that legged robots can cross gaps as large as their reach. However, Siegwart and Nourbakhsh [3] profess that one-legged robots hold an advantage over multi-legged robots in this respect, as they are able to dynamically cross gaps larger than their own stride “by taking a running start.” This distinction has perhaps become less pronounced in recent times with the advent of multi-legged robots which can run, bound, and jump much more successfully, and at higher speeds, than before. A great example of this is Boston Dynamics' WildCat robot, which can gallop outdoors, untethered, at speeds up to 32 km/h [29].

Although one-legged robots pose certain benefits, balance remains the biggest challenge [3], as one would expect. Raibert [23] and his group at the Carnegie-Mellon University (CMU) mainly used their one-legged hopping machines, such as the one shown in Figure 2.5, for research purposes. The aim of their research was to identify the fundamental concerns to be addressed when controlling legged robots, develop primary control algorithms, and study running.

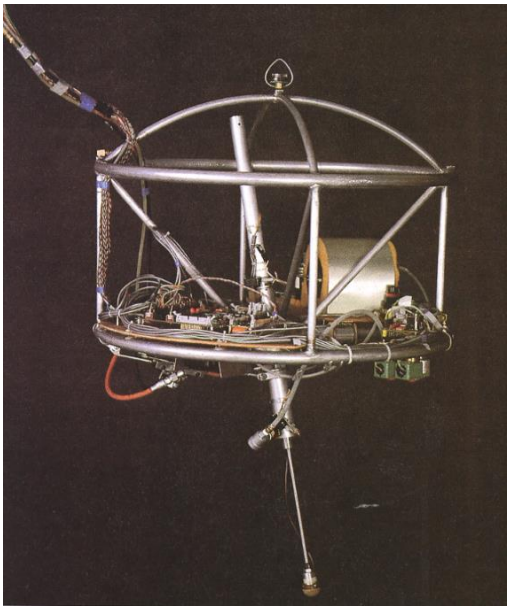


Figure 2.5: One-legged, three-dimensional hopping machine developed at CMU [23]

Two-legged robots, also referred to as bipeds, are much more common. They are predominantly constructed with a humanoid structure [2], [3], and their anthropomorphic form is ideal for researching human-robot interaction [3]. Bipeds have been developed which are able to run, jump, climb up and down stairs, and even perform acrobatic manoeuvres like somersaults [3]. One of the most well-known bipedal, humanoid robots is Honda's ASIMO (Advanced Step In Innovative Mobility) [30], shown in Figure 2.6. Similar to one-legged robots, balance and stability are the main concerns regarding bipeds, as they continuously require active control to prevent them from falling over [2], [3].

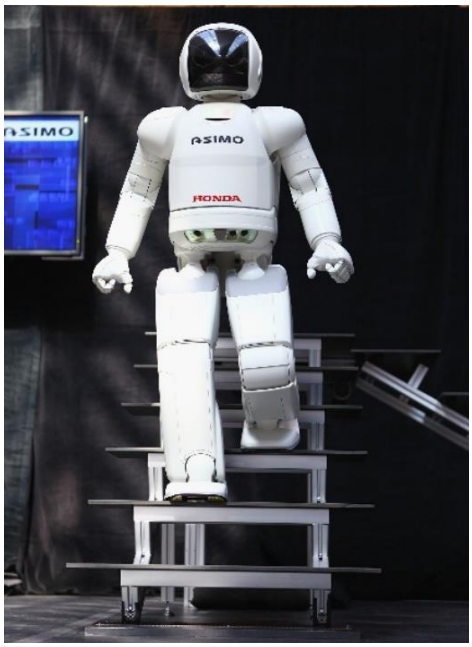


Figure 2.6: Honda's ASIMO humanoid robot climbing down stairs [30]

Robots with three legs or more are referred to as multi-legged robots [2]. Since three-legged robots, or tripods, are not very commonly used, they will not be discussed further. Quadruped robots, on the other hand, which are rather popular, have four legs and are usually of mammalian form [3]. Even though they are passively stable, maintaining stability during walking is still tricky as the gait is required to actively shift the centre of mass (COM) during locomotion [3]. BigDog by Boston Dynamics is one of the most well-known quadruped robots. Its capabilities demonstrate the practical viability of legged robots, proving that they are not limited to the research world [1]. Figure 2.7 demonstrates BigDog operating in the "real world."

Figure 2.7: Boston Dynamics' BigDog quadruped climbing a snow-covered hill [1]

Another popular variety of legged robot is the six-legged, or hexapod, robot. Hexapod robot construction is often inspired by insects, having legs with three degrees of freedom [3]. Renowned for their static stability, hexapods reduce control complexity slightly and, as a result, can be readily built and used by hobbyists [3]. A prevalent hexapod platform in hobby robotics, the PhantomX MK-II, is displayed in Figure 2.8.

Figure 2.8: PhantomX MK-II hexapod robot, popular with hobbyists [15]

Ding et al. [5] state that hexapods present the greatest efficiency for statically stable walking, and Gonzalez de Santos et al. [24] found that, while statically stable legged robots are inherently slower than those with dynamic stability, hexapods are able to achieve higher speeds than quadrupeds when walking with statically stable gaits. Furthermore, Preumont et al. [31] discovered that statically stable robots with more than six legs pose no advantage with regards to walking speed.

Eight-legged robots also exist [2], but are not as common. An example is the SCORPION robot shown in Figure 2.9, which was developed for space robotics explorations and is being evaluated by NASA for extraterrestrial exploration [32].

In addition, application specific legged robots, such as pipe inspection and climbing robots [2], as well as wheel-leg hybrid robots have been developed [2], [3], but these will not be discussed further here.

Figure 2.9: SCORPION eight-legged robot designed for space robotics applications [32]

2.3.4 Gaits and Stability

Legged robot locomotion is achieved by coordinating the leg motions, in a sequence of lift and release events for each leg, referred to as a gait [3]. The determination of the best sequence of lift-off and placement of the feet on the ground is referred to as gait, or locomotion, planning and is, according to [5], the most frequently studied problem in the field of legged robotics. A legged robot's leg count determines the amount of gaits it can possibly realize, with a one-legged robot technically not using a gait as it does not have multiple legs which require coordination [3].

As mentioned earlier, stability is usually more of a concern with legged robots than for those with wheels, as they have a propensity for falling over during locomotion and also require active balancing if they have too few legs. Accordingly, robotic locomotion is categorized as being either statically stable locomotion or dynamic locomotion [5].

At this point it is enlightening to present a formal definition of the term “statically stable.” During statically stable locomotion the moving body is required to be stable at all times, thus the vertical projection (along the gravitation vector) of the centre of mass must always remain within the interior of the supporting polygon created by the supporting feet (i.e. the feet in contact with the ground) [4], [5], [27]. When this condition is not met, the robot is said to walk with a dynamically balanced gait [4].

Siegwart and Nourbakhsh [3] explain that a statically stable pose can be achieved with three legs, as long as the COM remains within the ground contact tripod. They illustrate this with the example of a three-legged stool, which maintains balance without requiring motion or active control. If the stool is gently pushed, the small deviation from stability is passively corrected back to the stable pose upon removal of the disturbing force. During walking, however, a statically stable gait requires that, at the very least, a tripod of supporting legs is constantly in contact with the ground [3], implying that a robot would require more than three legs to achieve statically stable locomotion.

A quantitative measure of the stability of a walking robot at any given time can be evaluated in the form of a stability margin. The stability margin is defined as the minimum distance from the vertical projection of the COM onto the ground to the edges of the polygon of support created by the ground contact feet [33].

Examples of static gaits include walking and crawling, while dynamic gaits include running, hopping, trotting, and galloping [5], [27]. According to [27], dynamic gaits allow for faster locomotion with the drawback of being far more demanding on the balancing task. Thus, for highly irregular terrain having significantly sized obstacles, statically stable gaits are preferable, especially from a control perspective.

The extent of the control demand related to dynamically balanced locomotion is brought to light in [3] with an intriguing analogy to organisms found in nature with various successful leg configurations. Creatures with six or more legs, such as insects and arachnids, are able to walk immediately from the time of birth, as balance is a relatively simple problem for them (having inherent static stability). Larger animals with four legs, such as mammals and reptiles, generally do not walk with a statically balanced gait but are able to stand easily with a stable pose on their four legs. Such animals usually spend a few minutes after birth learning how to stand, after which it takes them yet another few minutes to learn how to walk without falling over [3]. Some mammals have perfected the ability to walk on two legs,

and in some cases, such as humans, balance has evolved to the stage where jumping on one leg is even possible. Since standing on two legs is not a statically stable pose, it takes infants several months to learn and develop the ability to stand and walk without falling and even more time to be able to confidently run, jump, and stand on a single leg [3].

An alternative to using a pre-planned and fixed gait is to plan the footholds, leg sequences, and body motions in a flexible manner, adapting to the terrain conditions and robot's state as it walks. This is referred to as a free gait and is especially beneficial when walking on highly irregular terrain [5]. Adaptive foothold, body path, and leg trajectory planners are implemented on a variety of legged robots found in the literature, e.g. [27], [28], [34].

Gaits are characterized in terms of the concepts of cycle time, T, duty factor, β, and relative phase [25]. The cycle time is essentially the time taken to complete a single cycle of the gait and is broken down into the stance phases, T_{st}, and swing phases, T_{sw}, of the various legs [25].

The duty factor for a leg is defined as the ratio (thus $\beta \in [0,1]$) between the duration of the leg stance phase, or support period, and the cycle time [25], [33]:

$$\beta = \frac{T_{st}}{T_{st} + T_{sw}}. \tag{2.1}$$

A gait can be distinguished as a walking or running gait based on its duty factor, as $\beta \geq 0.5$ for walking and $\beta < 0.5$ for running gaits [33].

Relative phase is a similar concept to duty factor; however, the relative phase of a leg i is defined as the fraction of the cycle time elapsed from the instant at which a reference foot is set down to the instant that foot i sets down [25]. The right rear limb of the robot is usually designated as the reference leg, with the setting down of this foot being considered as the start of a stride [25].

2.3.5 Legged Robot Sensory Systems

An important topic worth considering when discussing legged robots, or any form of robot or autonomous system for that matter, is that of sensors and sensory systems. According to [3], one of the most significant tasks which an autonomous system is required to perform is to acquire information about the environment around it. This is accomplished by extracting meaningful information from measurements taken using a variety of sensors.

2.3.5.1 Types of Sensors

Sensors are usually classified based on two functional categories [3]:

- *Proprioceptive* or *exteroceptive* sensors
- *Active* or *passive* sensors

Siegwart and Nourbakhsh [3] explain that proprioceptive sensors measure aspects which are internal to the robotic system, such as motor speed, wheel load, joint angles, and battery voltage, to name a few. Exteroceptive sensors, on the other hand, are used to obtain information from the surroundings of the robot, in order to identify environmental features. Examples of exteroceptive sensory measurements include measurements of distance, light intensity, and sound amplitude. Proprioceptive and exteroceptive sensors are often used in combination on mobile robots.

As described in [3], passive sensors rely on measurements of ambient energy entering into the sensor from the environment, as in the case of temperature probes, microphones, and cameras. Active sensors, however, emit energy into the environment and make measurements of the reaction of the environment. Their interactions with the environment are thus more controlled than those of passive sensors, usually resulting in superior performance. The downsides of this method of measurement include the risk of the emitted energy potentially having an effect on the very characteristics attempted to be measured by the sensor, as well as the possibility of interference with other signals. Some examples of active sensors are ultrasonic sensors and laser rangefinders.

Some of the sensors commonly used on legged robots are described in Table 2.1, along with examples of robots using these sensors.

Table 2.1: Sensors commonly used on legged robots, including examples of robots using them

Sensor Type	Description	Proprioceptive/ Exteroceptive
Joint-position sensors	Measure angles of the joints in the legs. Robot examples: • SILO6 – 2000 pulse/turn encoder [24] • Warp1 – joint and motor encoders [4] • LittleDog [34]	Proprioceptive
Heading sensors	Determine direction in which robot is moving. Examples of sensors used to measure heading include gyroscopes, inclinometers, and compasses, the two most common types of compasses being Hall effect and flux gate compasses [3]. Robot example: • SILO6 – electromagnetic compass [24]	Can be proprioceptive (such as gyroscopes and inclinometers) or exteroceptive (such as compasses) [3]

Inertial Measurement Unit (IMU)	Consists of three accelerometers aligned with the axes of the local coordinate frame, and three gyroscopes measuring angular velocities about each axis [35]. Generally used to determine robot pose, or more specifically body orientation in legged robots. Robot examples: • Messor [35] • LAURON V [9] • Warp1 [4] • StarlETH [6] • LittleDog [27], [34]	Proprioceptive
Foot force/load sensors	Measure force/load exerted on each foot; sometimes only contact sensors are used to determine whether feet are in contact with ground [4]. Robot examples: • Warp1 – contact/distance sensors and load sensors [4] • Messor – resistive sensors (resistance inversely proportional to force) [35] • LAURON IV – Three-dimensional (3D) foot force and spring force sensors [36] • LittleDog – 3-axis force sensors [34] • StarlETH [6]	Proprioceptive
Joint torque sensors	Measure torque on leg joints. In many cases, measurements of electrical current in joint actuators are used to estimate joint torques. Robot examples: • Warp1 [4] • Messor [35] • LAURON series of hexapods [9], [36] • Weaver [13]	Proprioceptive
Cameras	Obtain images of robot's surroundings and environment. Used for tasks such as visual odometry and Simultaneous Localization and Mapping (SLAM) [35]. Robot examples: • Messor – video camera [35] • BigDog – stereo cameras [1] • LAURON V – stereo cameras and infrared (IR) camera [9]	Exteroceptive

Laser rangefinder (LRF) and LIDAR[1]	Measures distance using a laser beam. LIDAR generally uses multiple laser scanning to obtain three-dimensional range information [37]. Robot example: • Messor – uses LRF for map building [35] • LAURON V – rotating LIDAR [9] • BigDog [1]	Exteroceptive
Structured light system	A vertical laser stripe, usually projected onto stairs to measure stair height and depth. Robot example: • Messor [35]	Exteroceptive
Global positioning system (GPS)	Outdoor beacon system using a collection of satellites to determine global position [3]. Robot examples: • SILO6 [24] • LAURON IV [36] • BigDog [1]	Exteroceptive

The sensors described above are a mere glimpse into the topic of sensory systems and perception. Especially when considering a legged robot which is required to function robustly outside the laboratory environment, on uneven terrain and with autonomy, the appropriate and sufficient use of sensors is crucial. BigDog, for example, boasts approximately fifty sensors on board, ranging from an IMU and joint-position sensors to sensors monitoring homeostasis, such as hydraulic pressure and temperature, engine speed and temperature, and many others [1].

However, before outfitting a robot with a huge number of sensors, of every variety imaginable, it is vital to bear in mind a few key aspects of different sensor types, particularly pertaining to the intended application of the robot in question. This is especially true when considering proprioceptive and exteroceptive sensor types, which are further elaborated in the following section.

2.3.5.2 Proprioceptive versus Exteroceptive Sensors

When using exteroceptive sensors the robot's surroundings are perceived from the viewpoint of the robot's local reference frame [3]. Since walking robots are mobile machines, their motion and continually changing position can have a notable effect on the overall behaviour of the exteroceptive sensors.

[1] The term LIDAR, also written as LiDAR, is sometimes taken to be an acronym of “Light Detection and Ranging” but can also be considered a portmanteau of the words “light” and “radar” [37].

Furthermore, exteroceptive sensors are vulnerable to erroneous performance or a lack of functionality depending on environmental conditions [3]. An example which illustrates this clearly is that of GPS, in which the receiver is required to read data, in a line-of-sight manner, from four satellites simultaneously. This is unlikely to be possible in locations such as indoors, in confined spaces like city blocks with a large number of tall buildings and even in dense forests, causing the GPS sensor to not function effectively in such areas. The specific orbital paths of the various satellites which constitute the GPS system also result in the coverage in different parts of the Earth not being geometrically identical. Compasses are another example, as the magnetic fields of man-made structures and other magnetic objects in their vicinity can affect these sensors' performance. Many other examples may come to the reader's mind, such as cameras which would not be very useful in conditions with low or no light, or laser and infrared which would not effectively detect objects such as glass.

It is also easy to conceive that sensors which rely on data being sent out and received from external sources could be susceptible to hacking, jamming, or being overridden by external, and possibly malicious, entities wishing to obtain information about the robot's state and location, as well as preventing it from carrying out its intended mission. This issue is of grave concern especially in defence and security-based applications of mobile robots, as in the case of a hexapod robot being developed by the CSIR for military operations.

Needless to say, proprioceptive sensors are not plagued by these problems as they are not dependent on signals coming in from sources external to the robot, or on the conditions in the robot's surroundings and environment. Rotella et al. [38] go as far as declaring that exteroceptive sensors, such as cameras or GPS units, are unfit for use in unstructured environments on their own, asserting that the activities of control and localization should be accomplished through proprioceptive sensing.

Naturally, these sensors do present their own problems. While proprioceptive sensors are extremely accurate at measuring the characteristics which they are primarily designed to measure [3], using the data obtained from them to estimate other factors can quickly reduce their accuracy and even effectiveness. For instance, Gonzalez de Santos et al. [24] indicate that using even extremely accurate joint-position sensors for the purposes of odometry and dead-reckoning builds up cumulative position errors, or drift, which increases with the distance walked. This is mainly attributed to flexion in the leg links and joints, as well as foot slippage. It is intuitive to reason that estimating characteristics such as position, rather than measuring them against a fixed reference point (with exteroceptive sensors such as land-based beacon systems [3]), would give rise to inaccuracies. Many research groups counteract this by using data from multiple sensors or combining both proprioceptive and exteroceptive sensors, obtaining the "best of both worlds," so to speak. In the case of SILO6, Gonzalez de Santos et al. [24]

fuse data from the joint-position sensors with data from heading sensors, such as a compass, in order to improve the accuracy of their measurements, along with combining this information with GPS data when satellite reception is available. Another example is that of fusing foot force sensor data with joint current data in order to improve the accuracy of limb load measurements, a technique used by robots such as Messor [35].

It is thus clear that neither proprioceptive nor exteroceptive sensors, on their own, are the quintessential solution, or "silver bullet," for legged robot sensory systems. These two categories present their own advantages and disadvantages, with each one being necessary for a number of different applications. However, for the task of posture control on uneven terrain, it seems more appropriate to rely on a proprioceptive sensory system, allowing the robot to stabilize itself and navigate the terrain regardless of environmental conditions and reception from external sources. This is especially true when considering that the application area focussed on in this study involves operations in the defence domain.

2.3.6 Applications of Legged Robots

Legged robot applications are exceedingly wide-ranging. It was pointed out earlier that one of the most common appeals of using legged robots lies in their ability to replace humans in environments which are difficult for humans to access or would endanger their lives [2]. This is not the only use for legged robots, however. Machado and Silva [2] have put together a fairly comprehensive list of legged robot applications from a multitude of sources in their review of the available literature (up to the point of writing). The following list of applications is chiefly compiled using the work of [2], with additional citations indicated for applications also mentioned in various other sources:

- Exploration of remote locations, such as in:
    - Volcanoes,
    - Space [28], [32],
    - The sea bottom [39];
- Working in hostile or dangerous environments, such as:
    - Places with high radiation levels, or chemical or biological pollution, such as nuclear power plants [28],
    - Mining prospecting and mine exploration,
    - Demining and recovery of unexploded munitions [7], [8], [24],
    - Disaster areas [28],
    - Search and rescue missions [28],
    - Military operations [1];

- Excavation or construction works;
- Forestry [3];
- Assisting humans with payload transportation operations;
- Medical applications, such as endoscopies or a substitute for wheelchairs;
- Entertainment [3] and home use (predicted to be utilized for domestic tasks or as simple companions);
- Education;
- Hobby use [3].

While this list is comprehensive, it is not exhaustive, and it is also likely that even more applications will be discovered as research into legged robots progresses, making the technology increasingly feasible and capable.

2.4 Control Considerations for Uneven Terrain Walking

It has already been stated that stability becomes an important concern when dealing with legged robots, especially those with fewer legs, and that the control requirements for walking, particularly on rough terrain, are quite intensive. As a natural consequence, legged robot control systems have been in development since the early days of research into legged robotics.

In the 1980s Raibert [23] and his team at CMU's Leg Laboratory began experimenting with control algorithms for simple legged robots, in an attempt to determine the fundamental components which would be required in order to effectively control these robots. They worked on an assortment of one-, two-, and four-legged machines and accomplished a wide range of manoeuvres including hopping, running (with trotting, pacing, and bounding gaits), climbing simple stairs, jumping over obstacles, and even simple gymnastic manoeuvres. They found that they were able to control these dynamic motions using a relatively simple control approach, breaking down the behaviour into three primary activities:

- Supporting the body with vertical bouncing or hopping motions;
- Controlling the posture, or attitude, of the body through torques at the hip during the stance phase of each leg; and
- Placing the feet in strategic locations to maintain balance and achieve the desired velocity.

Raibert et al. [1] refer to these three activities as the "essential ingredients" of the control systems. However, they acknowledged that the initial research was limited, in view of the fact that these robots still lacked onboard power and control algorithms which provided locomotion and stability on rough terrain. These obstacles would need to be overcome in order to build practical legged robots.

This is where the research described in [1], on the BigDog quadruped robot, comes in. BigDog is an outstanding demonstration of the versatility and practical applicability of legged robots. It is a self-contained robot with onboard power and sensors, which allow it to travel over a multitude of terrain types, including mud and snow. It is able to stand, squat, walk (with crawling, trotting, running, bounding, and galloping gaits), jump, carry large loads, and even travel on inclines with various surfaces, including rutted and rocky trails.

The control system coordinates the leg and body kinematics, along with the ground reaction forces, while responding to basic postural commands [1]. Load carrying ability is optimized by distributing the load among the legs. Terrain changes are sensed and accordingly adapted to by posture control. The posture control algorithm essentially controls the body position and attitude by utilizing the leg kinematics and ground reaction forces, implementing computed leg compliance on terrain irregularities. The body pitch and roll angles, along with the height, relative to the ground can thus be controlled, resulting in adaptation to local terrain variations without the need for higher-level terrain sensing. Figure 2.10 illustrates an extremely simplified, high-level block diagram of the main components which make up the control structure of BigDog.

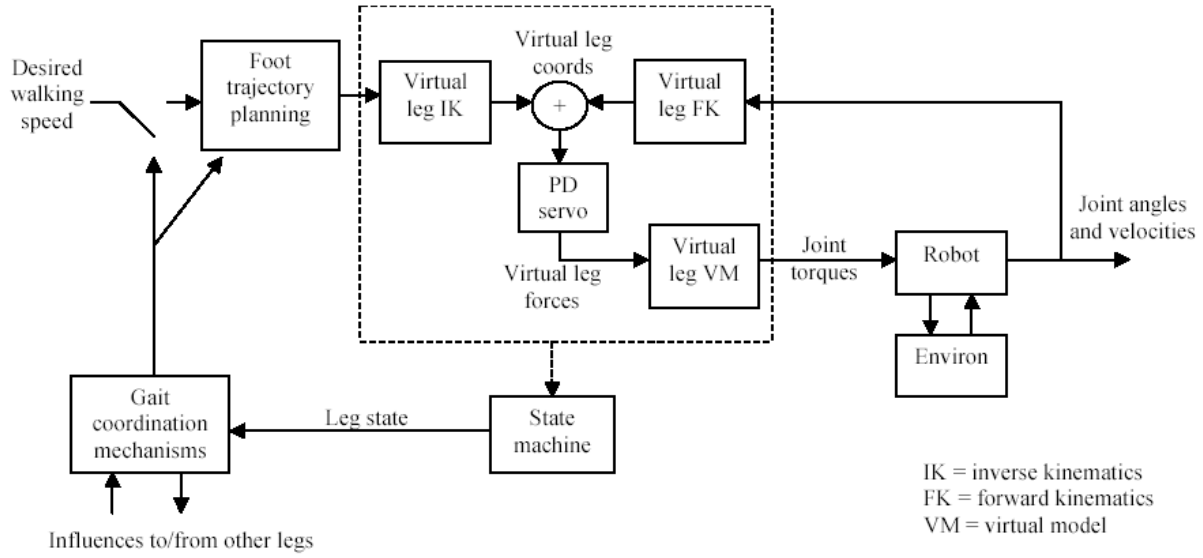


Figure 2.10: Simplified, high-level block diagram of BigDog's control structure [1]

In addition, footfall placement is adjusted to compensate for the orientation of the body and the ground plane relative to gravity [1]. Accordingly, the control system makes the robot lean forward on inclines, backward on declines, and sideways while walking along a contour line. Shallow to moderate inclines can be handled by merely making minor postural adjustments, while inclines steeper than 45° also require that the walking gait pattern be modified and smaller steps are taken.

Many similarities can be observed in the control systems of other robots in the literature. For example, the quadruped StarlETH (see Figure 2.11), used in the Autonomous Systems Lab of the Institute of Robotics and Intelligent Systems at ETH Zurich, requires that its body roll, pitch, and height are kept constant in most of its gaits [6]. Hutter et al. [6] point out the following as the key elements of their control system, in order of decreasing priority:

1. Maintaining support and stability;
2. Controlling the height of the robot body;
3. Controlling the pitch and roll angles of the robot body;
4. Controlling the main body turning rate; and
5. Controlling the main body velocity.

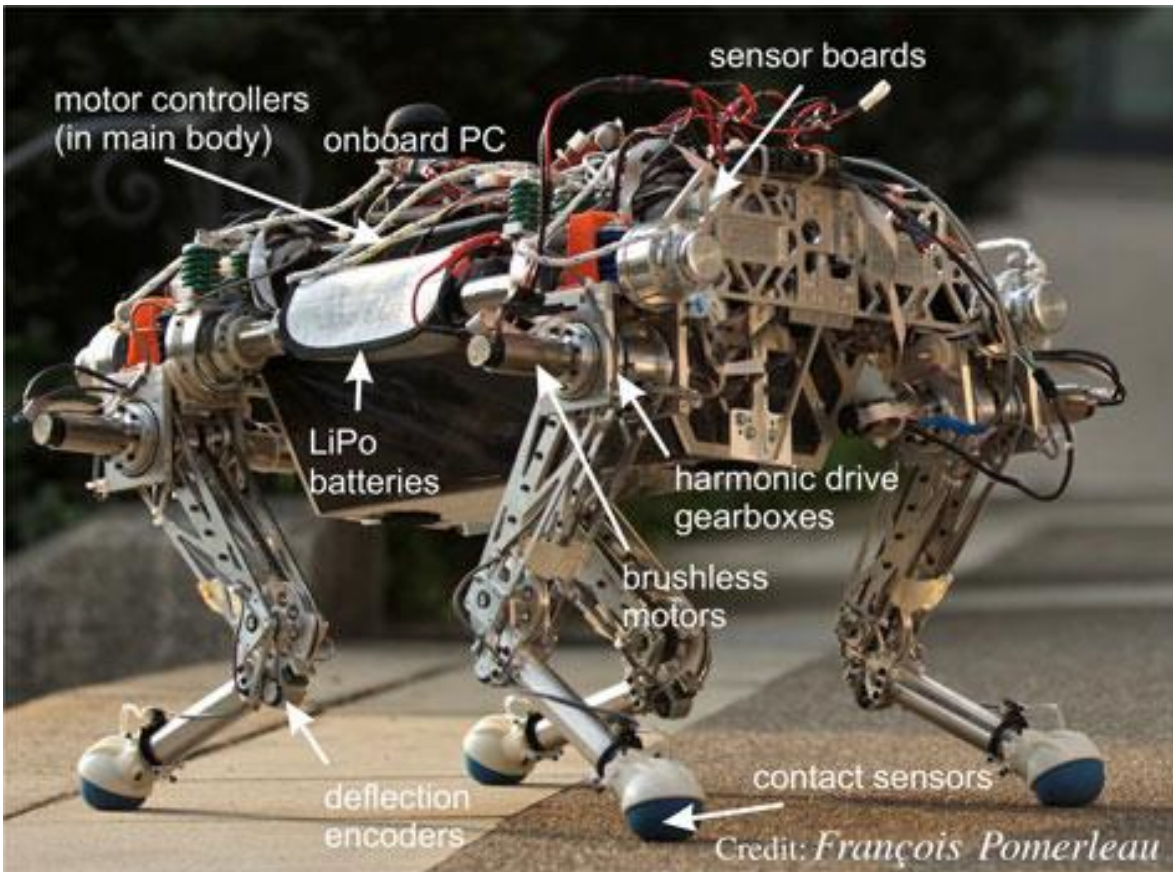


Figure 2.11: StarlETH quadruped robot developed by ETH Zurich [6]

Points 2 and 3 essentially constitute posture control. The importance of posture, or attitude, control is exceptionally evident in the above discussion, especially for locomotion on uneven terrain. This is not confined to robots which primarily execute dynamic motions and balance dynamically. Even on hexapod robots, which predominantly walk with statically stable gaits, posture control is deemed by many researchers to be imperative for locomotion on rough terrain.

Arena et al. [40] insist that the issue of attitude control is fundamental for hexapods during locomotion on uneven terrain. Moreover, the impressive manoeuvrability displayed by multi-legged robots was realized by Roennau et al. [9] to be a result of kinematics that allows high flexibility and continuous terrain adaptation. They accomplish terrain adaptation on their hexapod robot, LAURON V, by means of a reactive control system which adjusts body position and posture. Furthermore, Zhang et al. [10] argue that posture adjustment can be effectively used to achieve a larger stability margin, a necessity for broad terrain adaptation to unstructured terrain which, according to them, is "the purpose of research on hexapod robot[s]."

There are also more specific applications of hexapod robots in which posture control is vital. Demining work is one such application. Uchida and Shimoi [7] identified that for demining it is important that each leg is placed precisely to avoid mines and that a stable attitude is maintained. In view of the fact that the terrains of the minefields which they considered were uneven, Uchida and Shimoi [7] indicated that active attitude control would indeed be required to maintain a stable attitude. Irawan and Nonami [8], whose work involved a later iteration of the hexapod robot used in [7] for demining tasks, also identified that their robot, COMET-IV, is required to maintain a horizontally stable attitude, especially on uneven ground with soft surfaces.

Another illustrative example is the Crabster200 (CR200) six-legged, undersea, remotely operated vehicle (ROV), which can overcome high tidal currents by being operated to adjusted its body posture against the current direction [39].

It is apparent from this analysis that posture, or attitude, control is unquestionably an essential requirement for the locomotion of multi-legged robots, of all types, on uneven terrain. Therefore, the following section focusses specifically on posture control work and approaches carried out in the literature.

2.5 Previous Work on Posture Control for Hexapod and Quadruped Robots

Since a hexapod robot is being used in this study, the focus from this point on will predominantly be on hexapods, while also considering work done on quadrupeds, owing to the fact very similar control techniques are applied to these types of robots. Given that bipedal robots continuously require active stabilization and control, as mentioned previously, and generally have a humanoid structure, they utilize fairly different walking motions and control strategies compared to quadrupeds and hexapods. As a result, bipeds are not investigated further in the sections that follow.

2.5.1 Virtual Model Control Strategies

Research into posture control of legged robots has been ongoing for some time. One of the earlier posture controllers developed for a hexapod was based upon the Virtual Model Control (VMC) scheme [12]. Nelson and Quinn [12] implemented the control method on a pneumatically actuated, cockroach-like robot, named Robot III, using an optimization approach to solve the force distribution problem. The resulting algorithm was able to control the posture of the robot in conformance with commanded body postures and orientations, as well as reject disturbances such as the case of the robot being pushed sideward. Nelson and Quinn [12] claim that the algorithm could easily be modified to be used during locomotion.

A few years earlier, Yoneda et al. [41] implemented a variation of Sky-Hook Suspension Control (SHSC), which is traditionally used for wheeled vehicles, on the TITAN VI quadruped robot. The method is based on equilibrium equations involving the ground reaction forces of the robot's support legs and is, in fact, very similar to VMC. They also included a passive suspension element, made of elastic material and having a stroke of 30 mm, to each foot of the robot, in order to absorb the impact when the foot touches down, thus reducing the disturbance induced on the robot body. As a result, posture was able to be controlled quite well, with dynamic walking on uneven terrain conditions exhibiting maximum pitch and roll angles of the body of slightly less than 3°. It should be noted, however, that the uneven terrain used was simply a 40 mm thick plate, which is quite small compared to the size of TITAN VI, a 195 kg robot with leg lengths of up to 1.289 m [41].

Taking inspiration from this work, Uchida and Shimoi [7] also utilized SHSC on a simulation of the COMET-I mine detecting hexapod robot (mentioned earlier), for the cases of both even and uneven terrain, with uneven terrain being represented by a 10 mm high step-obstacle in the robot's path of motion. Based upon their simulated results, they conclude that the proposed control scheme would be useful for uneven terrain locomotion [7].

This idea of basing the posture control on a virtual model of the body and ground reaction forces has been taken forward through later iterations of COMET. A newer version of the large, hydraulically driven hexapod, COMET-IV, is used by Irawan and Nonami [8] for walking on uneven and extremely soft terrain, as can be seen in Figure 2.12. Their attitude control method attempts to optimize the virtual forces from the moment of inertia of the robot's body with linear quadratic regulator (LQR) control [8].

Figure 2.12: COMET-IV hexapod on uneven and soft terrain [8]

Qingjiu and Fukuhara [42] went a step further with the idea of employing a virtual suspension model for a robot, utilizing the method to control both the posture and vibrations of the body of a hexapod robot by means of sliding mode control. They performed experiments with the control algorithm on a rather large, crab type, six-legged robot and were able to realize both a stable posture and decreased vibrations with three kinds of gaits.

Yet another variation of virtual model based control was implemented in the form of a conventional proportional-integral-derivative (PID) controller on the Warp1 quadruped robot [4]. Ridderström and Ingvast [4] explored the case of the robot standing on all four legs and were thus able to use a simple distribution rule for the vertical leg forces, rather than an optimization algorithm. The resulting controller was able to track step changes in the posture reference values, as well as regulate the body posture while the robot stood on a balance board which was manually tilted back and forth, as depicted in Figure 2.13.

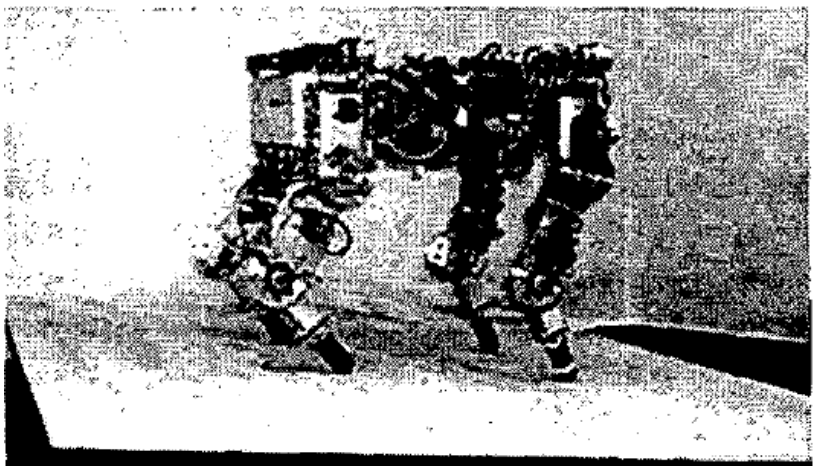


Figure 2.13: Warp1 quadruped regulating its posture on a balance board [4]

Virtual Model Control has remained a popular posture control method even in more recent times, owing to its simplicity and effectiveness. One of the well-known legged robot platforms using the VMC technique is the StarlETH quadruped from ETH Zurich [6]. They distribute the required foot forces by optimization and use joint torque controllers to achieve the desired forces. Incorporating this posture control system into their greater, hierarchical locomotion control structure allows StarlETH to walk with a variety of dynamic gaits and carry out dynamic manoeuvres on an assortment of terrain conditions, as in the experiment shown in Figure 2.14.

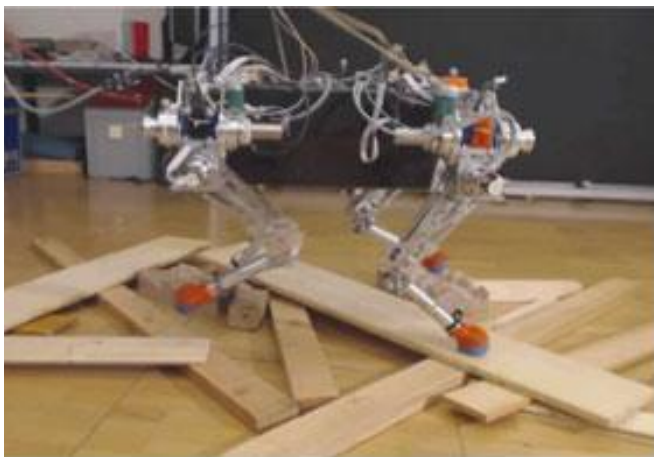


Figure 2.14: StarlETH quadruped trotting over terrain with loose obstacles [6]

Zhang et al. [10] also implemented VMC, with optimization methods for foot force distribution, on their "bionic" hexapod robot, HITCR-II. They termed their algorithm the *Posture Control strategy based on Force Distribution and Compensation*, or PCFDC. Interestingly, this is one of the few implementations which essentially plugs the posture control module into the inverse kinematics (IK) based foot position control system. Thus, to adapt to the terrain, foot forces are adjusted by providing corrections to the desired foot-end positions of each leg, rather than by controlling leg joint torques. This is portrayed more clearly in the block diagram of Figure 2.15.

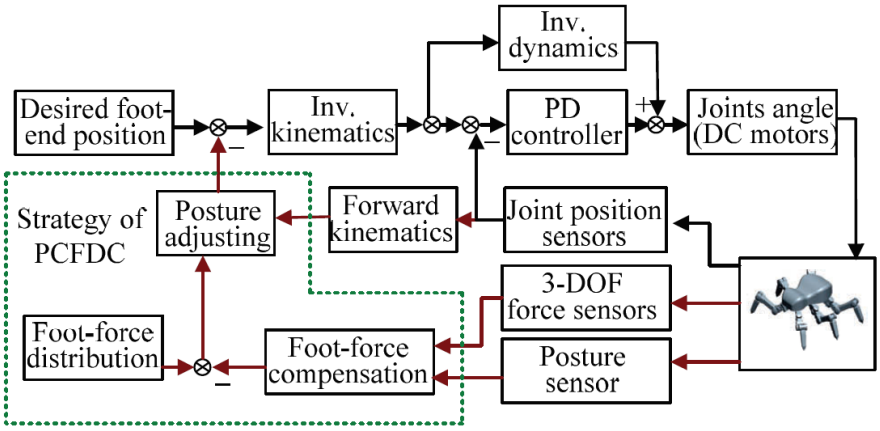


Figure 2.15: High-level block diagram illustrating control structure of HITCR-II [10]

Both simulations and experiments with HITCR-II were used by Zhang et al. [10] to test their control strategy, imitating uneven terrain conditions with obstacles in the form of rectangular blocks (see Figure 2.16). They found significant improvements in posture and stability margin while the robot walked over the obstacles, and the authors proclaim that the PCFDC controller effectively improves walking stability.

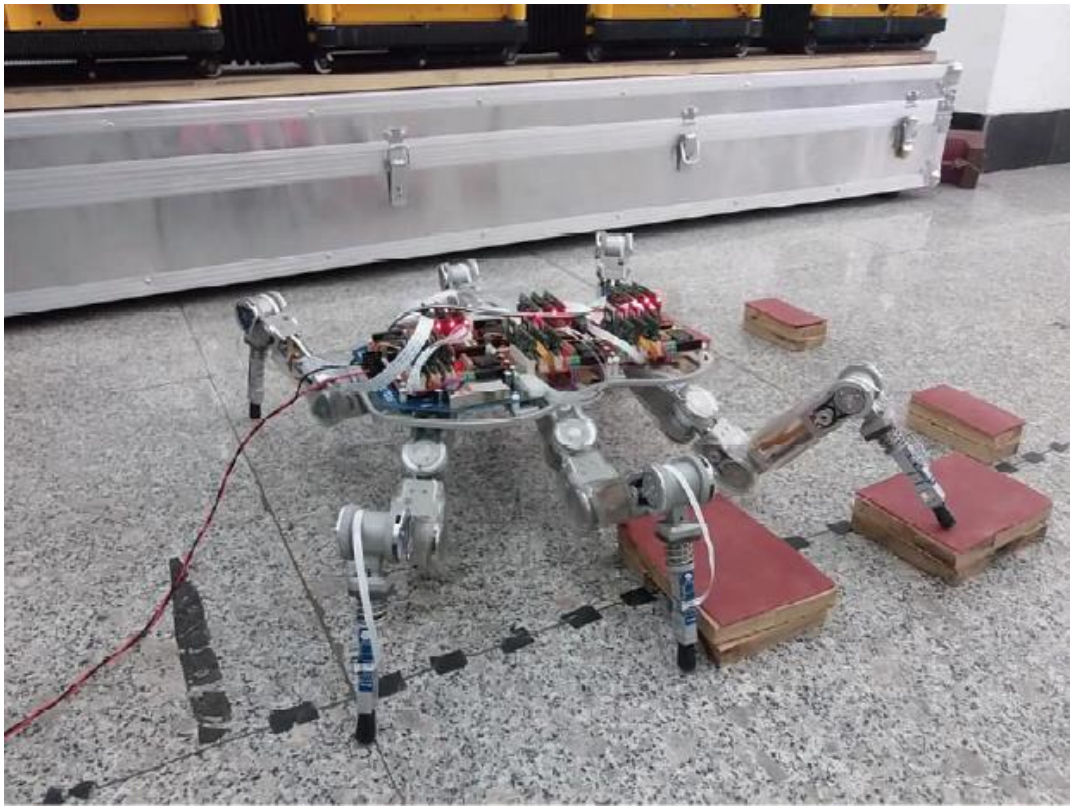
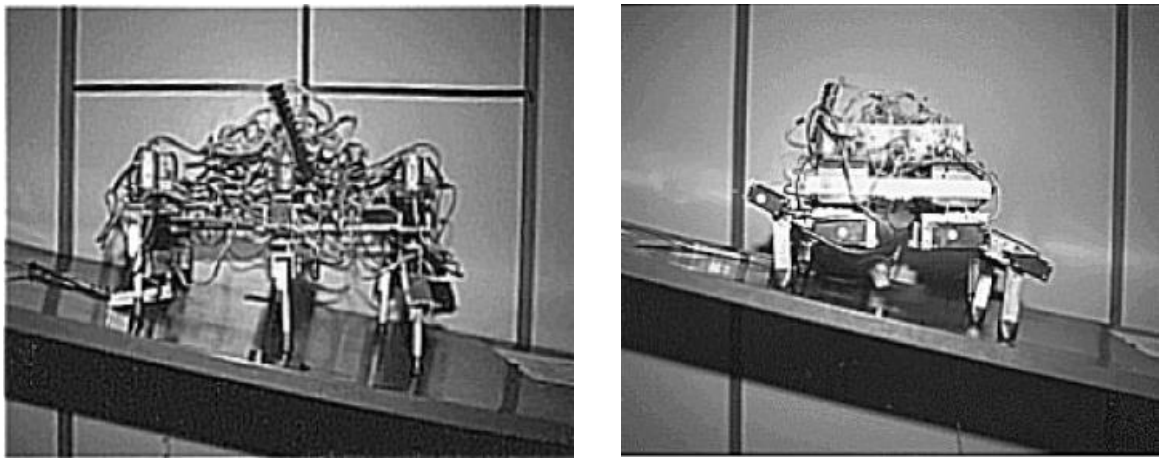


Figure 2.16: HITCR-II hexapod walking on uneven terrain in the form of rectangular blocks [10]

2.5.2 Other Posture Control Strategies

Although Virtual Model Control, and similar techniques, have been commonly used and shown to be effective, other approaches to posture control have been taken. One such case is a study by Arena et al. [40], in which they took an analogic spatio-temporal approach over digital control. They implemented a cellular non-linear network (CNN) as a central pattern generator (CPG) for the locomotion control, and achieved attitude control with proportional-integral (PI) controllers on each of the legs, which were also implemented as non-linear controllers based on a CNN. They tested the control system on a hexapod robot platform, where the attitude controller was used to maintain a horizontal orientation of the platform while walking on sloping planes, as demonstrated in Figure 2.17. Walking on uneven or irregular terrain was not considered by them in this study.



(a) (b)

Figure 2.17: Experiments of a hexapod walking on a slope in the (a) pitch and (b) roll direction [40]

In a later study, Arena et al. [43] employed a similar approach in order to control a cockroach-like hexapod robot to climb over obstacles. Locomotion is still achieved with a CNN-based CPG, but attitude is now controlled using PID controllers for pitch and roll, with a Motor Map Controller (MMC) adaptively varying their references depending on which phase of walking or climbing the robot is in. The phase is determined based on exteroceptive sensing to detect upcoming obstacles. A simplified block diagram of the control scheme is illustrated in Figure 2.18. In the study, the control system was implemented only in simulation, using fictitious antennae to evaluate the distance to, as well as height of, the obstacle in front of the robot.

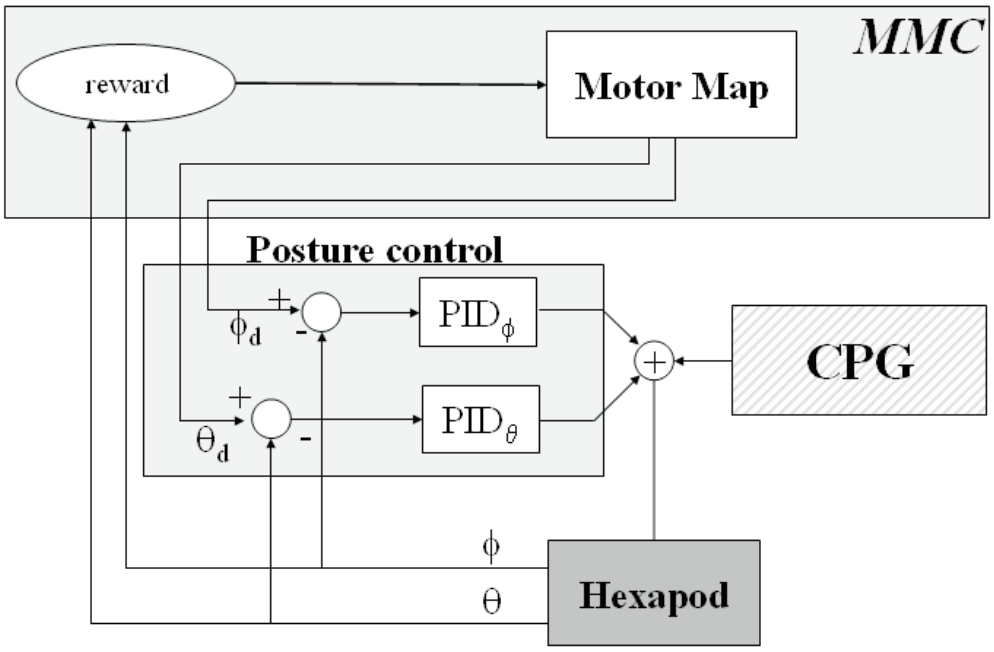


Figure 2.18: Simplified block diagram of adaptive posture control system implemented in [43]

Similarly, Campos et al. [25] make use of a nonlinear dynamical system to achieve lateral postural control (only roll angle) of a hexapod robot, with the gait being generated using a CPG. To test the method, the robot was simulated to walk on a platform which was continuously subjected to changes in its lateral inclination, and the attitude controller was found to be able to keep the robot's roll angle to a minimum [25].

In contrast, Belter and Skrzypczynski [28] implemented a posture optimization strategy, based on the Particle Swarm Optimization method, on the Messor robot, for traversing rough terrain. Their results showed that the approach allows the robot to determine the optimal posture, in terms of stability, while climbing obstacles and walking on irregular terrain. However, it should be noted that this does not necessarily allow for a specific posture to be held while walking and that the level of autonomy displayed in their study requires exteroceptive sensors for dynamic path and foothold planning.

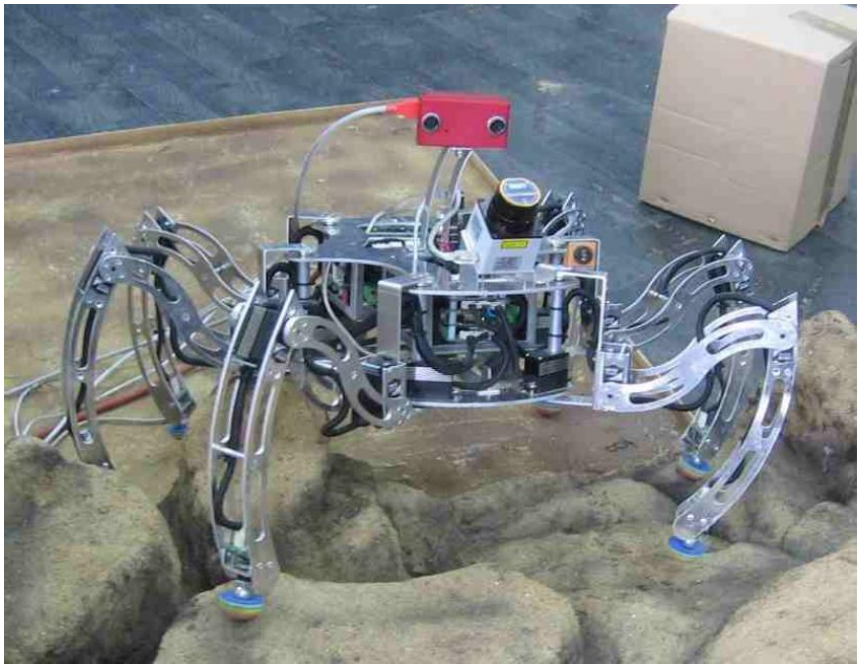


Figure 2.19: Messor hexapod on rocky terrain [28]

As mentioned earlier, another robot which utilizes posture adjustment is the CR200 ROV developed for subsea exploration (see Figure 2.30). In order to address the problem of the robot overturning in high tidal currents, Gyeong-Mok et al. [39] implemented a controller to adjust the body posture against the current. The implemented algorithm is based fully on the kinematics (forward and inverse) of the robot, not taking forces into account. While the controller manages to successfully adjust posture using the kinematics, it should be noted that, since this robot is an ROV, it is only semi-autonomous, at least in the implementation discussed in [39]. Therefore, it does not continuously adjust its posture depending on the terrain while walking, rather requiring a desired posture as a user input to the controller.

A more autonomous posture control approach based primarily on kinematics has been taken by Roennau et al. [9], to enable the LAURON V hexapod robot, shown in Figure 2.20, to surmount obstacles and steep slopes without having knowledge about its environment.

Figure 2.20: LAURON V hexapod on rough terrain [9]

Their controller has three main "behaviours" which contribute to the posture control: position, inclination, and height behaviour. Each behaviour makes adjustments to the foot positions, to effectively adjust posture [9]. These three behaviours are scalable and can be independently switched on and off. The control algorithms for each behaviour utilize a proportional gain type of control strategy and take into account the foot positions and ground contact status of each foot. The COM of the robot is also dynamically calculated based on the centres of mass of the main robot body and each individual leg, by following the Denavit-Hartenberg (DH) notation. Based on the foot positions and IMU data, the slope of the terrain is estimated and the desired robot pose is dynamically set to half of the steepness of this virtual slope.

In addition to these main behaviours, a vital component in improving uneven terrain manoeuvrability is the incorporation of a fourth rotational joint on each leg of LAURON V [9]. This improves kinematic ranges and adjusts the leg shanks to always be parallel to the gravity vector (see Figure 2.21), decreasing large joint torques resulting from the shanks being perpendicular to the ground inclination (which in turn reduces energy consumption). In fact, the adjustment of the angle of this fourth joint is one of the primary functions of the posture controller's inclination behaviour.

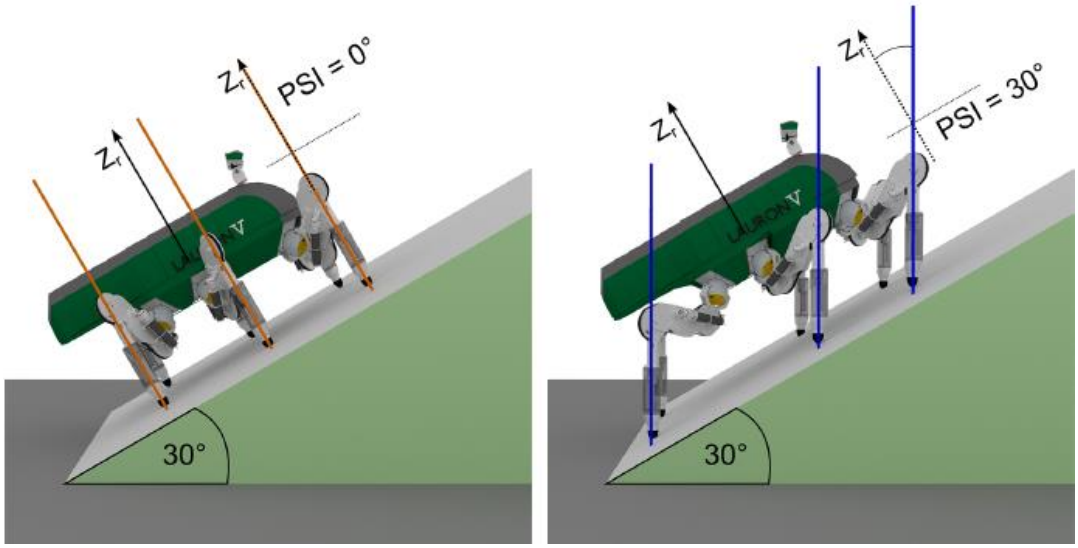


Figure 2.21: Illustration of LAURON V on slope, orientating its feet relative to the gravitational vector using the fourth rotational joints in its legs [9]

Experimental results revealed that LAURON V was able to climb over large obstacles (relative to its own height), as well as stand on slopes up to 42° and walk on slopes of up to 25°, with all three behaviours and the fourth rotational joints turned on [9] (examples of experimental tests can be seen in Figures 2.22 and 2.23). Roennau et al. [9] suggest that foot slippage was the main problem when walking on slopes. LAURON V is also able to remain stable in the presence of impulse disturbances, such as being pushed or kicked sideward [9]. While the control system's performance is certainly impressive, it is worthwhile to take note of the large computational requirements of the controller and how slowly LAURON V is able to carry out the above manoeuvres even in lab conditions, as can be viewed in the video at [44], in which many of the motions are sped up significantly.

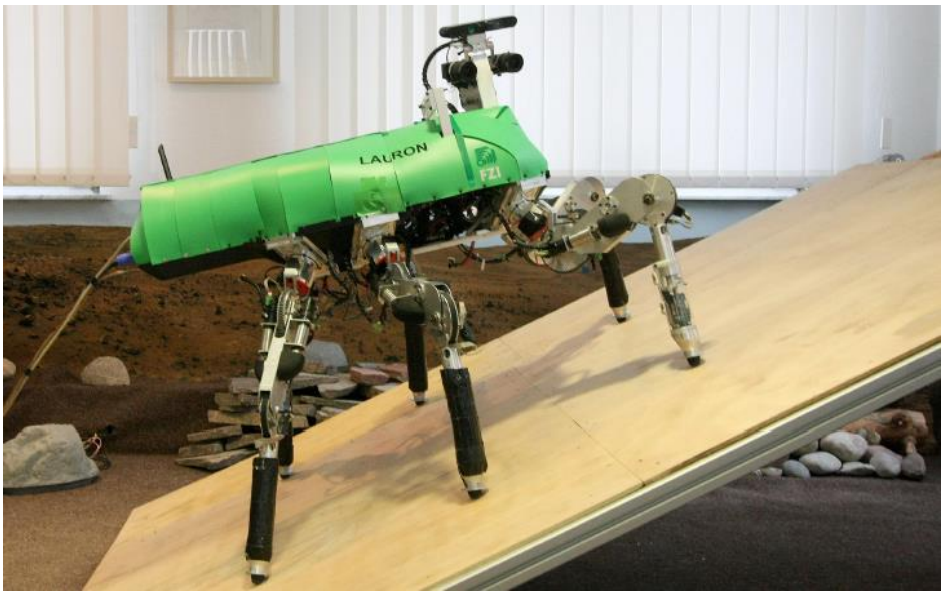


Figure 2.22: Experimental test of LAURON V walking on a slope [9]

Figure 2.23: Experimental test of LAURON V climbing over obstacle approximately 40 cm high [9]

Another hexapod which uses additional leg joints and a kinematic approach to controlling posture on uneven terrain is Weaver. However, Bjelonic et al. [13] took it a step further than LAURON V by equipping Weaver with five joints per leg, for a total of 30 DOF. The authors propose that this allows Weaver to cope with large slopes in any orientation, rather than only in the longitudinal direction like LAURON V. They also use DH notation for the kinematics and proprioceptive sensing to determine the orientation of the gravity vector relative to the robot's body. Their inclination controller increases stability by shifting Weaver's COM and adjusting the additional leg joints to align the force ellipsoids of the foot tips with the gravity vector, improving both manoeuvrability and energy efficiency. Furthermore, Weaver utilizes impedance controllers to provide compliant behaviour of the legs, which, according to Bjelonic et al. [13], realizes self-stabilizing behaviour by reacting to unstructured terrain. A diagram of this hierarchical control architecture is presented in Figure 2.24.

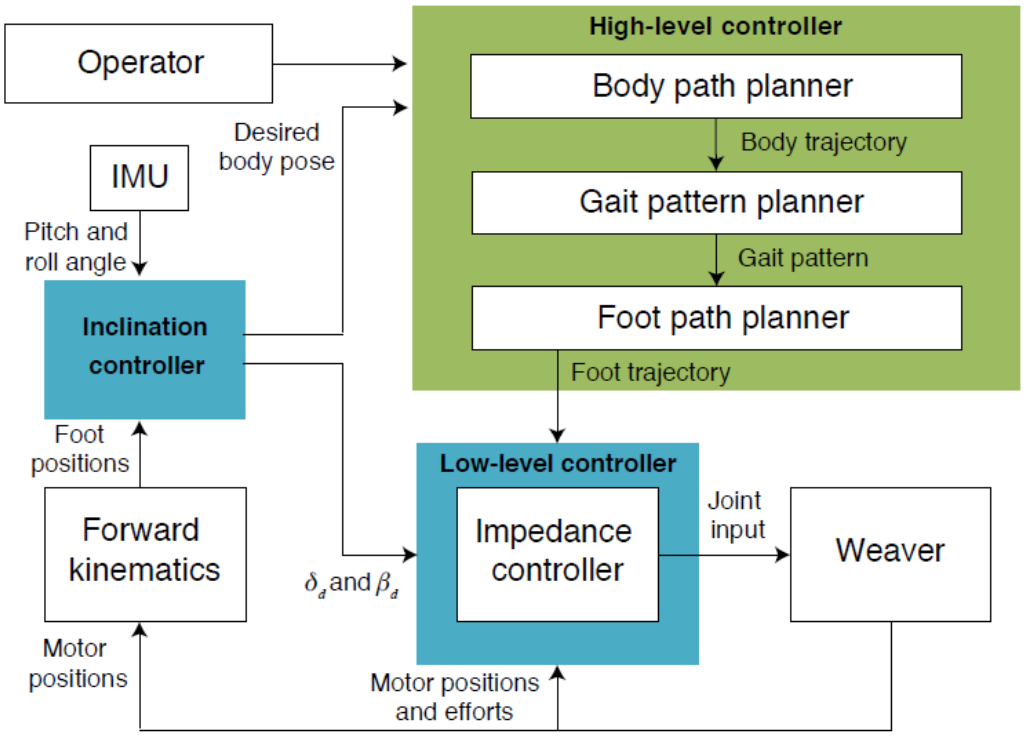


Figure 2.24: Diagram of hierarchical control architecture used on Weaver hexapod [13]

Based on the results of experimental tests, Bjelonic et al. [13] report that, with their reactive posture control system, Weaver was able to walk up slopes of up to 30° and stand with static stability on inclines of up to 50° (see Figure 2.25). They also found that mobility was improved and angular motions reduced while walking on a variety of uneven terrains (see Figure 2.26).

Figure 2.25: Inclination control test of Weaver hexapod [13]

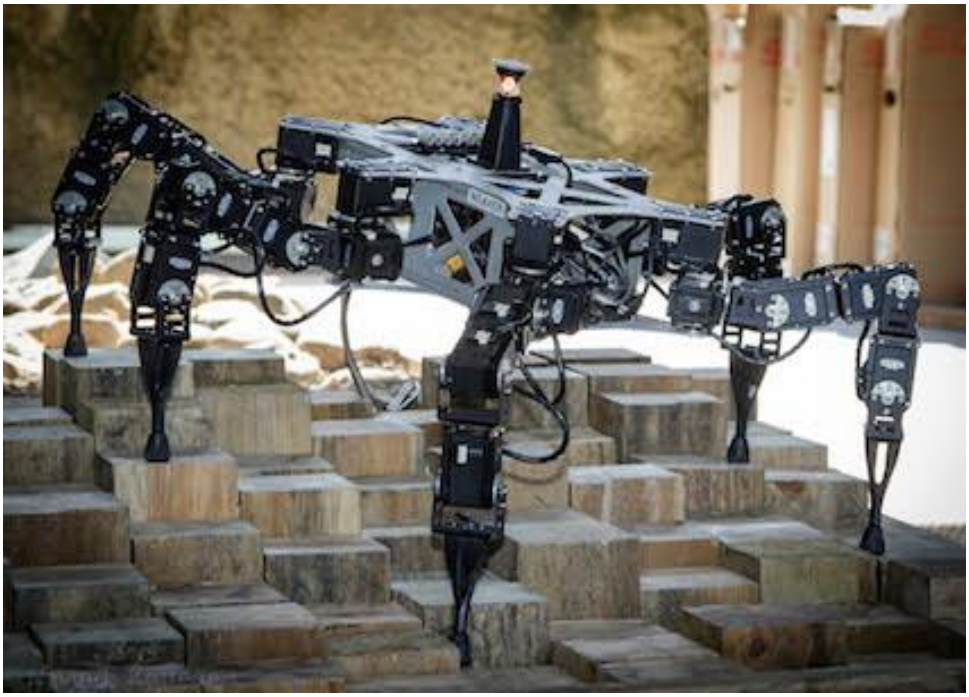


Figure 2.26: Weaver hexapod on a section of the multi-terrain testbed used in [13]

Most of the abovementioned robots are custom designed platforms with custom software written from the ground up. Robots like BigDog [1], COMET [7], [8], TITAN VI [41], and Warp1 [4], among others, are large, heavy platforms, the manufacturing and construction of which is no trivial undertaking. Smaller robots like StarlETH [6] and HITCR-II [10] boast highly specialized actuators and sensing equipment, much of which is custom designed and manufactured. Figure 2.27 shows a detailed representation of the three different series elastic actuators (SEAs) used on StarlETH, consisting of brushless direct current (BLDC) motors, Harmonic Drive (HD) gearboxes, springs, chain drives, and deflection encoders, among other specialized components [6]. HITCR-II's legs, the structure of which is illustrated in Figure 2.28, also make use of BLDC motors and harmonic reducers, along with custom designed 3 DOF foot force sensors [10].

(a) (b) (c)

Figure 2.27: SEAs used on StarlETH for (a) hip abduction/adduction, (b) hip flexion/extension and (c) knee flexion/extension joints [6]

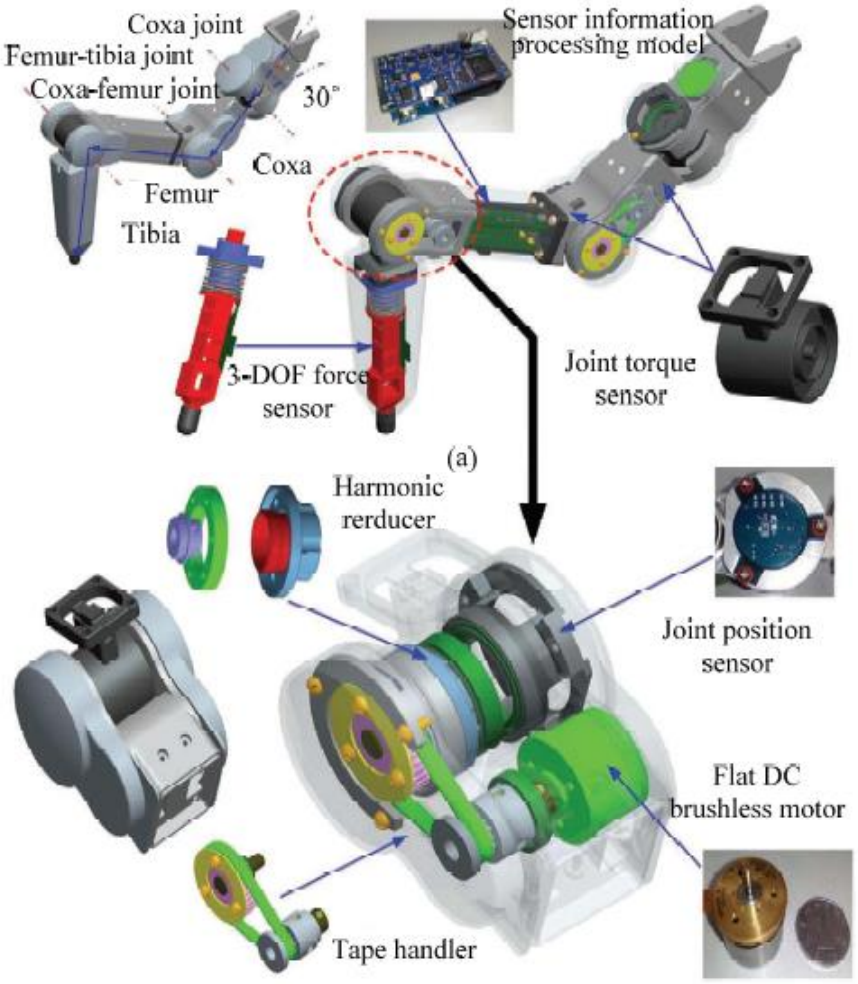


Figure 2.28: HITCR-II leg structure, showing joint actuator construction and 3 DOF force sensors [10]

More complex still are robots like LAURON V [9], [36] and Weaver [13], which possess additional leg joints and have redundant degrees of freedom (see Figure 2.29), or CR200 [39] which is designed to have multi-function legs which can act as manipulators (see Figure 2.30).

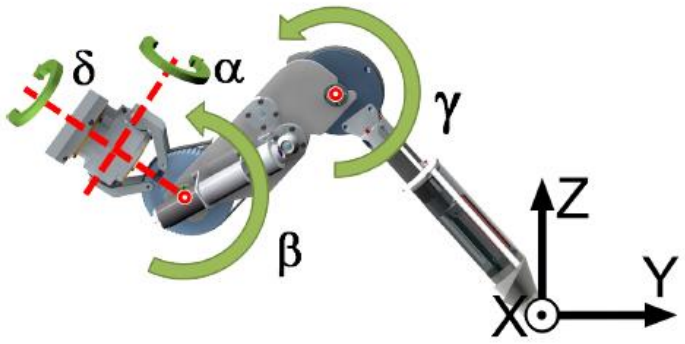


Figure 2.29: Leg structure of LAURON V with fourth rotational joint [9]

Figure 2.30: Conceptual illustration of CR200 using its front legs as manipulators [39]

2.5.3 Work on Commercially Available Platforms

Despite their increased costs and development time, platforms such as those described above are, by far, the most common in literature, though commercially available platforms are beginning to gain more popularity in legged robotics studies. However, much of the time, commercial platforms like the PhantomX hexapod are used for research on higher level tasks such as localization and mapping [45], [46], terrain classification [47], [48], and road following [49]. Even when stabilization or uneven terrain mobility are considered with these platforms, they are often modified physically and controlled with fully custom software, as in [50], or controlled primarily using kinematics, e.g. [51].

Some work is being done on attitude control and uneven terrain walking with low-cost hexapod platforms in the hobby robotics community. One project of particular interest, utilizing the PhantomX MK-II, was the work done by Renée Love [52] on hexapod locomotion software which included an attitude adjustment feature. The locomotion software code base was originally written by Kevin Ochs [53] for his custom designed Golem hexapod, primarily as a means to learn how to use the Robot Operating System (ROS) [54] (see Section 3.5). Renée Love's fork of the project aimed to modify the code base for use on the PhantomX MK-II, as chronicled on her forum thread at [52]. The two projects, along with some contributions by other members, were eventually incorporated into a single code

base and refactored as a more generic hexapod locomotion stack for ROS. This has now been made available on GitHub at [17], with the idea that it can be utilized for any hexapod with legs that have 3 or 4 DOF (with the fourth joint being for a tarsus leg segment, like that on Golem).

The attitude control feature incorporated into this locomotion stack uses the orientation information provided by an onboard IMU to raise or lower specific legs, with the aim of maintaining a level orientation of the robot body, and is thus aptly termed *auto-levelling* [17]. Adjustment of the leg positions is based purely on inverse kinematic calculations, assuming the robot to be standing on a planar surface. An example of the PhantomX MK-II, running on this locomotion software, can be observed in the video at [55] as it adjusts its attitude in response to the gradual tilting of the board on which it is standing.

A more recent hobby project showcasing impressive uneven terrain mobility is the MX-Phoenix hexapod created by prolific hobby roboticist Kåre Halvorsen, who goes by the pseudonym Zenta [56]. Although the MX-Phoenix is a custom designed platform, it makes use of commercially available actuators and electronic equipment, with the body parts being manufactured using a low-cost 3D printer [57], thus keeping the costs lower than those of many sophisticated and high-tech research platforms. The hexapod is demonstrated in the video at [58] climbing stairs and scaling significantly uneven terrain, such as large rocks (as can be seen in Figure 2.31). This level of mobility is made possible, in part, by the long tibia segments designed for the legs, along with the dynamic gait algorithm created by Zenta [57]. It is apparent from Zenta's forum posts [59] and the video [58] that the reactive gait is designed to lower the robot's swing legs until they contact the terrain, before changing them to stance legs. Simple switch sensors are integrated into the feet to determine their contact status [59].

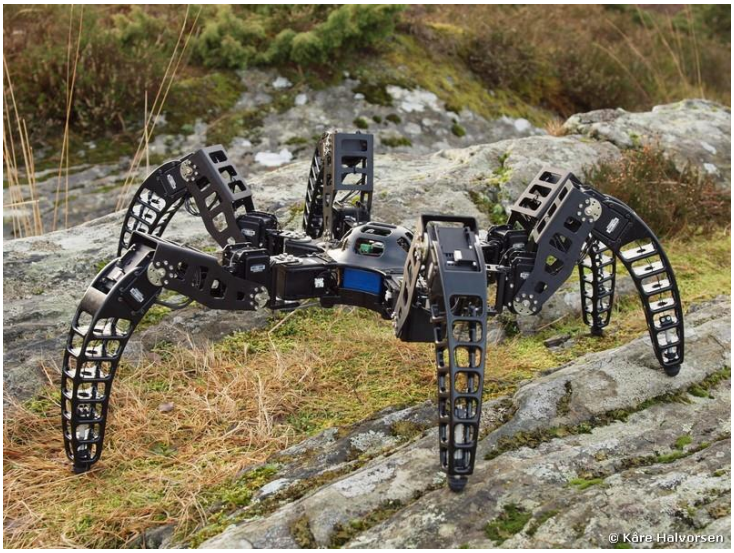


Figure 2.31: MX-Phoenix hexapod on rocky terrain [59]

Though the body orientation of the MX-Phoenix robot can be observed at [58] being adjusted freely while standing on uneven terrain, it should be noted that this follows user inputs from the remote control and is achieved with kinematic calculations [59], like with most other locomotion packages for hobby platforms. Zenta [59] explains that the gait engine also adjusts the body level so as to equalize the leg heights when all six legs are simultaneously in contact with the terrain, thereby roughly aligning the body with the terrain slope. This is accomplished entirely using foot position information and, at the time of writing this dissertation, no IMU has been incorporated into the MX-Phoenix platform, so it has no information about its orientation relative to the gravity vector or world frame. Zenta [59] has expressed that incorporating an IMU is planned to be the next step in the development of MX-Phoenix.

However, even with this added sensory information, the control of the body level would still be based wholly on kinematics, and there is no evidence, at least at this stage, that any of the robot's kinetic dynamics are planned to be taken into account in the locomotion and control software. This limits the robot's ability to dynamically react to disturbances from the environment, especially if no contact force information is taken into account. On highly uneven, or even dynamically varying, terrain conditions, the assumptions that purely kinematic control strategies are based on quickly become irrelevant, resulting in inadequate posture regulation. In fact, Schneider and Schmucker [60] assert that, in addition to increasing their reliability and functionality, the use of force information on walking robots simplifies many of their control algorithms.

While the above examples demonstrate a promising start in uneven terrain mobility with commercially available or low-cost platforms, they still lack the manoeuvrability and adaptability on uneven terrain exhibited by some of the more complex and capable control systems implemented on the custom designed platforms discussed above.

2.5.4 Review of Strategies Used in the Literature

It is evident from the preceding discussions that Virtual Model Control is one of the most commonly employed posture control techniques. The strategy relies on simple, linear control laws and has been shown to be effective in controlling the posture of a variety of legged robots.

Many of the other techniques utilize much more complicated control strategies, either involving non-linear networks or higher levels of control, such as dynamic foothold planning, with exteroceptive sensing capabilities. Furthermore, methods that are based primarily on kinematics generally seem to rely on additional leg joints to improve the robot's manoeuvrability and terrain adaptability.

Considering that low-cost, commercially available platforms have simpler kinematic structures and less sophisticated hardware, it would be preferable to keep the control strategy as simple as possible. A further advantage of this is that such strategies could eventually be implemented on less powerful

or costly computational hardware, which could potentially be mounted onto the robot itself. With these ideas in mind, it is apparent that VMC-based posture control techniques appear to be the "stand-out" choice for implementation on a commercial legged robot platform.

2.6 Conclusion

An overview of fundamental concepts regarding legged robots, in the context of the field of robotics, has been provided in this chapter. A more specific investigation on uneven terrain locomotion has also been carried out, reviewing literature pertaining to the control tasks associated with such locomotion.

This literature review has revealed that, while uneven terrain locomotion with hexapods has been quite successfully demonstrated in the lab on specialized robotic platforms, achieving such locomotion on lower cost, commercially available platforms would be highly advantageous, making the technology more feasible to implement in the real world. Furthermore, posture control has been clearly shown to be a vital component in realizing uneven terrain locomotion, with a great deal of development still required to effectively implement it on low-cost, commercial robots. This provides a strong motivation for the implementation of a posture control system on such a robotic platform in this study.

Additionally, the Virtual Model Control (VMC) strategy has been identified as a suitable technique to explore for implementation on a low-cost robotic platform, owing to its widespread usage and utilization of simple, linear control laws to regulate body posture. Therefore, the posture control system designed in this study will be based on the VMC strategy.

Chapter 3: Background Theory

Chapter 2 provided an overview of legged robotics and the research conducted in the literature on posture control for quadruped and hexapod robots, through which the motivation for this study was identified. In this chapter, specific topics pertaining to this study will be discussed in more detail, including hexapod robots and interaction control methods. Additionally, particular background theory and terminology required to facilitate the understanding of technical discussions in the remainder of this dissertation will be introduced and explained, such as transformation mathematics, virtual modelling of a hexapod, and the software system used to implement the control systems designed in this study.

3.1 Hexapod Robots

In view of the fact that a hexapod robot is used in this study, this section expands upon this type of legged robot in more detail than Section 2.3.3, with specific emphasis on mechanical design and construction, as well as gaits used for locomotion.

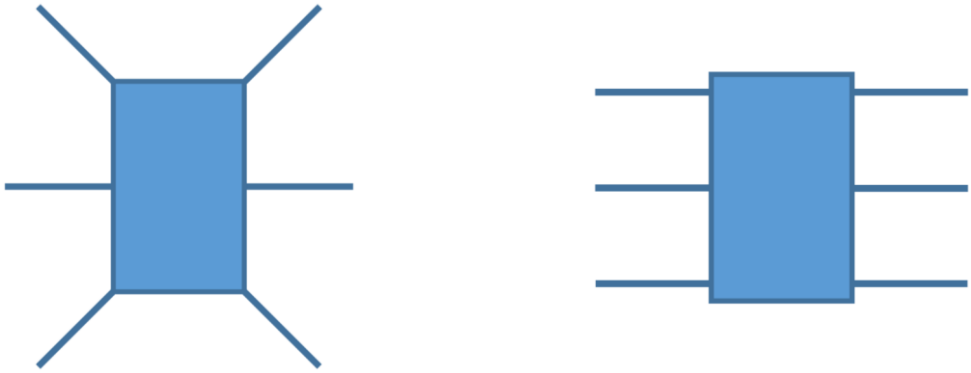
3.1.1 Body Structures

The main body of a legged robot is essentially a frame which acts as an attachment point for the legs and houses the robot subsystems, such as the onboard computer, sensors and power system [24], as well as carrying a payload, if present. Therefore, the body is required to be large enough while also being rigid enough to withstand deformation, without adding excessive mass to the robot. Though this may make the body construction seem like a fairly simple matter, it is not such a straightforward design consideration. In addition to being required to perform the abovementioned functions, the body shape determines key features of the robot, such as static stability. Likewise, the distribution of the legs around the body determines the size of actuators required, which in turn affects the total mass of the robot and, consequently, the speed at which it can travel.

Hexapod robot body structures, or shapes, are primarily grouped into two basic categories: *rectangular* and *hexagonal* (also referred to as *circular*) body structures [5], [31]. Additionally, a

hexapod structure exists that incorporates features of both the rectangular and hexagonal body shapes. These three body structures are described in more detail below.

On hexapods with rectangular body structures, the six legs are divided into two groups of three, distributed symmetrically, about the longitudinal mid-plane, on the two sides of the body [5]. This body type has traditionally been the most common on hexapods [24], [31], with a large amount of literature available on rectangular hexapods [33], some examples of which include [7], [8], [31], [42], [61]. While it is common for the front and back legs to be mounted at 45° angles to the body, predominantly for the purpose of increasing kinematic range, as depicted in the schematic diagram in Figure 3.1 (a), some rectangular hexapods in the examples above have all six legs mounted perpendicular to the main body, as can be seen in Figure 3.1 (b).



(a) (b)

Figure 3.1: Rectangular hexapod body structure, with legs mounted (a) at 45°, and (b) perpendicular to main body

Hexagonal, or circular, hexapods have round or hexagonally shaped bodies with the six legs distributed axisymmetrically around the body [5]. The main advantage of this body type is that it does not require a special gait for turning or changing direction, unlike rectangular hexapods [31]. A significant drawback of this configuration, however, is that it demands more complex control, as the trajectories of the various legs are different, even in a straight line, whereas the leg trajectories of rectangular hexapods are all the same when moving in a straight line [31]. A schematic diagram of this body structure is illustrated in Figure 3.2 below.

Figure 3.2: Hexagonal, or circular, hexapod body structure, with legs mounted axisymmetrically around body

Another body configuration which has been gaining popularity is one in which the attachment points for the central legs, on both sides, are shifted outward slightly (away from the longitudinal axis of the body) [5], [24]. This can, in essence, be seen as a transition between the rectangular and hexagonal body types [5]. The rationale behind using this configuration is related to its energy saving ability [5]. Gonzales de Santos et al. [24] explain that this energy saving property arises from the fact that satisfactory force distribution can be obtained when the central legs are shifted outward slightly, resulting in these legs supporting less weight and increasing the contribution of the corner legs to supporting the body. To illustrate the effectiveness of this configuration, they show that moving the central legs of their SILO6 hexapod robot slightly outward reduces the maximum joint torque by approximately 15.06%.

Walas and Belter [35] also use this rectangular body structure with an “extended middle” on the Messor robot to improve force distribution, as well as for realizing suitable dimensions to enable the hexapod to climb stairs. Figure 3.3 depicts a schematic of Messor’s main body, or trunk, clearly showing the central leg attachment points extended outward. In addition to the main dimensions of the trunk, this diagram also indicates the angular workspace for the first joint (coxa) of each leg.

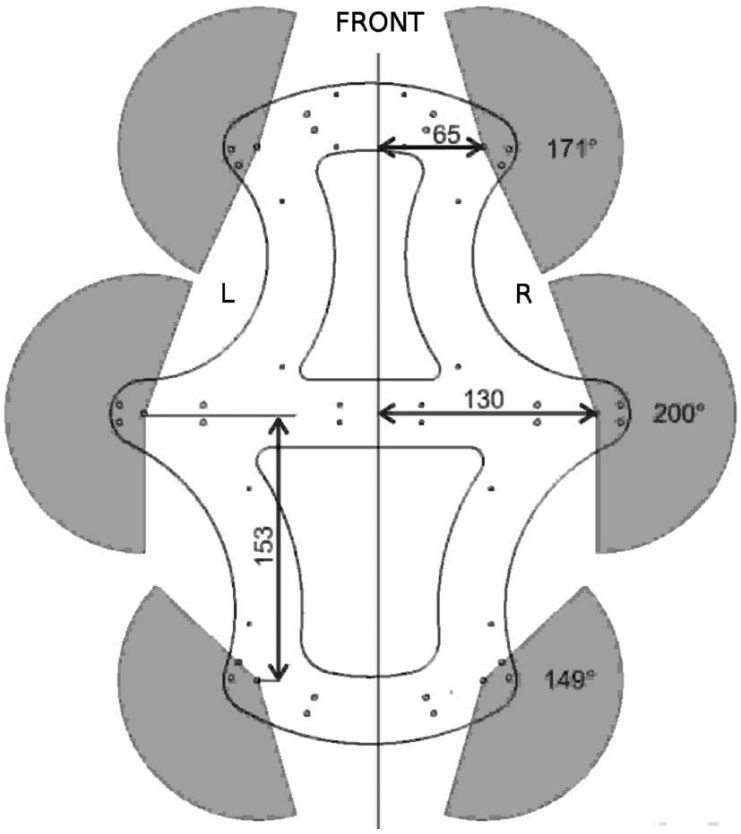


Figure 3.3: Schematic depicting trunk of Messor hexapod with central leg attachment points extended outward, as well as coxa joint workspaces [35]

3.1.2 Leg Configurations

Leg designs of walking robots are often biologically inspired, with *mammal* and *insect-like* legs typically being used [24]. Tedeschi and Carbone [33] elaborate that the major distinction between the two configurations is that insect type legs, also referred to as reptilian type legs, are configured such that the legs protrude out from the two sides of the body with the knee joints positioned to the sides of the base. On the other hand, the bodies of mammals are located above the legs, with the knee joints positioned below the hip.

Another leg configuration, which is essentially a variation of the insect type, is the *arachnid* configuration, wherein the leg extremities are still situated to the side of the body but the knees protrude out above the robot body [33]. Schematic representations of these three leg configurations are illustrated in Figure 3.4.

(a) (b) (c)

Figure 3.4: Schematic representations of (a) mammal, (b) insect (a.k.a. reptile), and (c) arachnid type hexapod leg configurations (extracted from [33]*)*

Walking robots with mammal type legs are more energy efficient, as this configuration requires lower joint torques to support the body [24], [33]. However, they are less efficient with respect to stability, owing to a narrower support polygon and higher centre of mass of the robot [24].

Insect-like legs provide greater efficiencies for both static and dynamic stability measures as they offer a wider support polygon, while keeping the COM lower [24]. However, according to [24] the joint torques and power consumption in this configuration increase “extraordinarily.”

Mammalian legs are usually found on quadruped robots, some examples being BigDog [1], LittleDog [27], [34] and Warp1 [4]. Hexapods, on the other hand, are much more commonly equipped with insect-like leg configurations, often drawing inspiration from the cockroach (examples include the robots used in [12] and [43]) or stick insect (as in the case of the LAURON series of hexapods [9], [36]). Insect type legs will accordingly be the focus of the remainder of this discussion, which is also equally applicable to arachnid type legs, as they only differ from insect-like legs in the positioning of the tibia, or knee, joint.

The majority of the insects which inspire hexapod leg design have at least four leg joints [5], [36], with the main segments of the insect leg being the *coxa*, *femur*, *tibia*, and *tarsus* [5]. Figure 3.5 illustrates a typical insect leg with six different segments indicated.

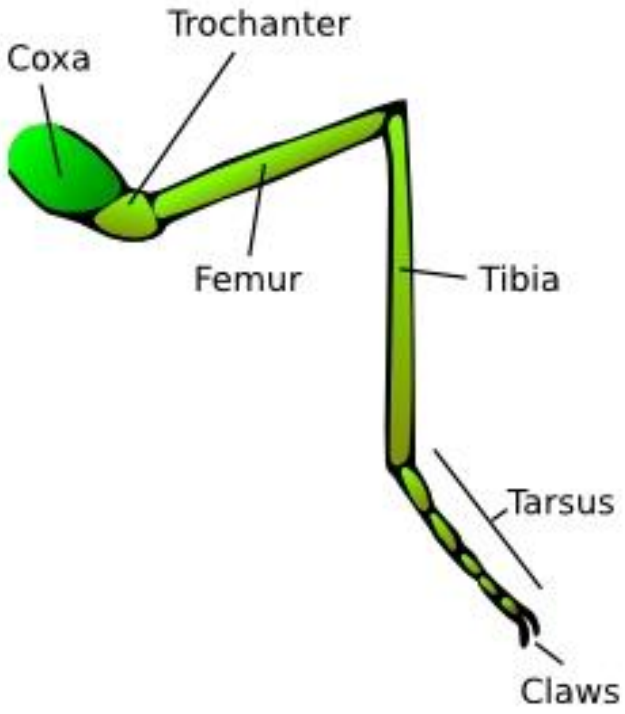


Figure 3.5: Illustration of typical insect leg, with various segments labelled accordingly [62]

Hexapod robot legs sometimes have up to six degrees of freedom [5], but most hexapods have legs with three degrees of freedom (provided by three independent leg joints) [36], which are often referred to as 3 DOF legs. Almost all hexapod legs have coxa, femur, and tibia segments (and associated joints), as depicted in Figure 3.6 below, which shows the three distinct segments of a 3 DOF hexapod leg, along with the rotational axes of the corresponding joints (at the base of each segment).

In addition to these three segments, some hexapods incorporate a fourth, or even fifth joint. Some examples of such robots were given in Chapter 2. Often, the fourth segment is a tarsus, or foot, as in the case of Golem [53], but some hexapods, such as the LAURON series [9], [36], include a trochanter joint for the fourth degree of freedom.

Three degrees of freedom are sufficient to accomplish motion by specifying the desired ground contact points for each foot [24]. Therefore, the fourth joint is often neglected in view of weight reduction and simplification of the mechanism. As expressed in [24], it is important that the legs be kept lightweight, as they contribute to the total mass of the robot, which is ultimately supported by the legs themselves. Consequently, the legs must have a high payload-to-weight ratio.

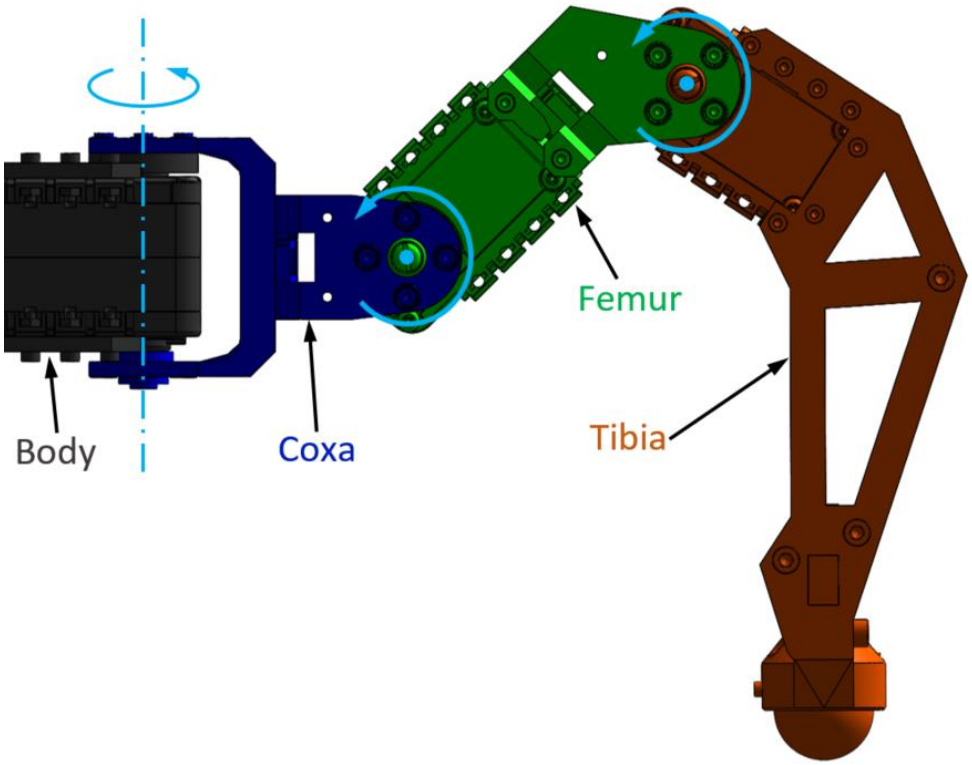


Figure 3.6: Schematic of 3 DOF hexapod leg, indicating the individual segments and the rotational axes of their joints

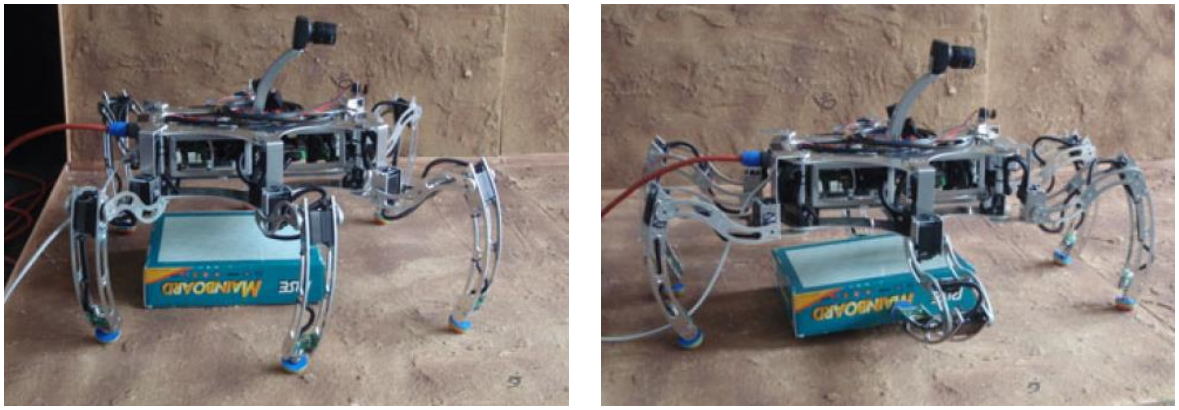
Interestingly, to address the large power consumption of the insect-like leg configuration, Gonzalez de Santos et al. [24] developed a leg which can be configured in either the insect or pseudo-mammal form to achieve suitable stability along with energy efficiency. Their SILO6 hexapod can be seen in Figure 3.7 with its legs in these two configurations.

(a) (b)

Figure 3.7: SILO6 hexapod with legs in (a) insect-like and (b) pseudo-mammal configuration [24]

Messor can also use a similar leg configuration, walking on its four corner legs, like a quadruped, while using the central legs to carry a payload underneath its trunk [35], as can be seen in Figure 3.8. During this "quadruped walking," Messor's legs get positioned towards the front and back, rather than out

to the side. However, the tibia joints, or knees, still protrude outward, and actually slightly upward like the arachnid configuration, instead of being positioned below the body as with traditional mammal leg configurations.



(a) (b)

Figure 3.8: Messor robot in (a) hexapod mode, with arachnid leg configuration, and (b) quadruped mode, with corner legs positioned to front and back of body, while central legs carry a payload [35]

3.1.3 Gaits

Having a relatively large number of legs, hexapod robots generally walk with statically stable gaits, as mentioned previously. In addition, many of the commonly used hexapod gaits possess a certain degree of symmetry [25]. This does not, however, imply that there is a shortage of gaits which can be used on a hexapod robot. Some of the most common hexapod gaits are described below. The leg numbering notation defined in Figure 3.9 is used for the gait descriptions in the remainder of this section.

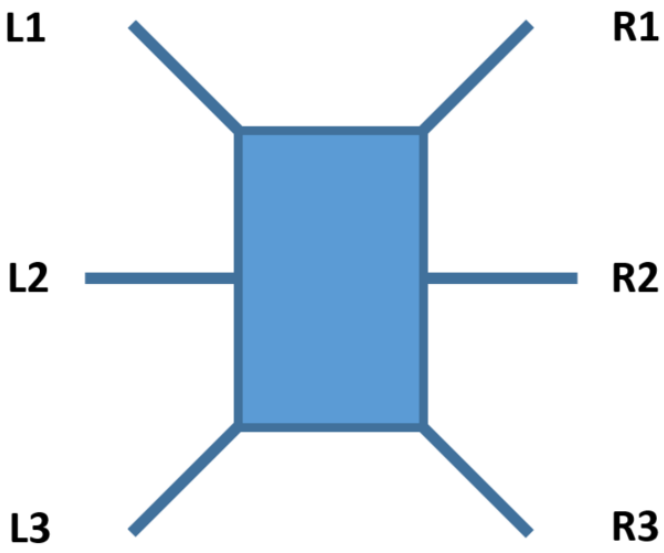


Figure 3.9: Leg numbering notation used for hexapod gait descriptions

Campos et al. [25] identified the three most frequently used hexapod gaits to be:

- **Metachronal gait** – a low speed gait which "can be described as a back-to-front propagating wave" [25] (the gait is thus also referred to as a wave gait), as the legs on either side of the robot move one-by-one, from the back to the front, one side at a time; duty factor $\beta = 3/4$.
- **Ripple gait** – a medium speed gait in which the pairs of corner legs diagonally opposite each other (L1-R3 and L3-R1 pairs) move together in phase, while the two central legs (L2 and R2) move alone; duty factor $\beta = 5/8$.
- **Tripod gait** – a high speed gait in which a tripod of legs (L1-L3-R2 for example) remains in contact with the ground while the remaining three legs move together in phase (technically described by Campos et al. [25] as the "ipsilateral anterior and posterior legs, and the contralateral middle leg," moving together in phase at each step); duty factor $\beta = 1/2$.

Another gait worth discussing is the quadruped gait [5], also known as the tetrapod gait [63]. This gait involves four legs acting to support the body while the remaining two move together in phase [5], in a sequence similar to the gaits used by quadruped robots. A variety of sequences can be utilized, one example being that legs L1-R3, R1-L3, L2-R2 are paired to move together [5]. The reader will likely have noticed a similarity to the ripple gait with this sequence of steps, the only difference being that central legs move together rather than one after the other. As a result, the gait is expected to achieve a similar speed to the ripple gait. This can also be perceived from the duty factor of $\beta = 2/3$ [5] for the quadruped gait, which differs from the ripple gait by only $1/24$.

In their study of the literature, Ding et al. [5] found that the quadruped gait shows favourable fault tolerant abilities in certain conditions as a result of three of the four support legs being able to support the robot body in the event that one is damaged or broken.

Figure 3.10 illustrates the leg sequences of the gaits described above, with the dark bars indicating swing legs. It should be noted that these diagrams simply depict the pairing of the legs during swing as time progresses (along the horizontal axis), not accounting for potential overlaps and phase shifts in the swing phases of the individual legs.

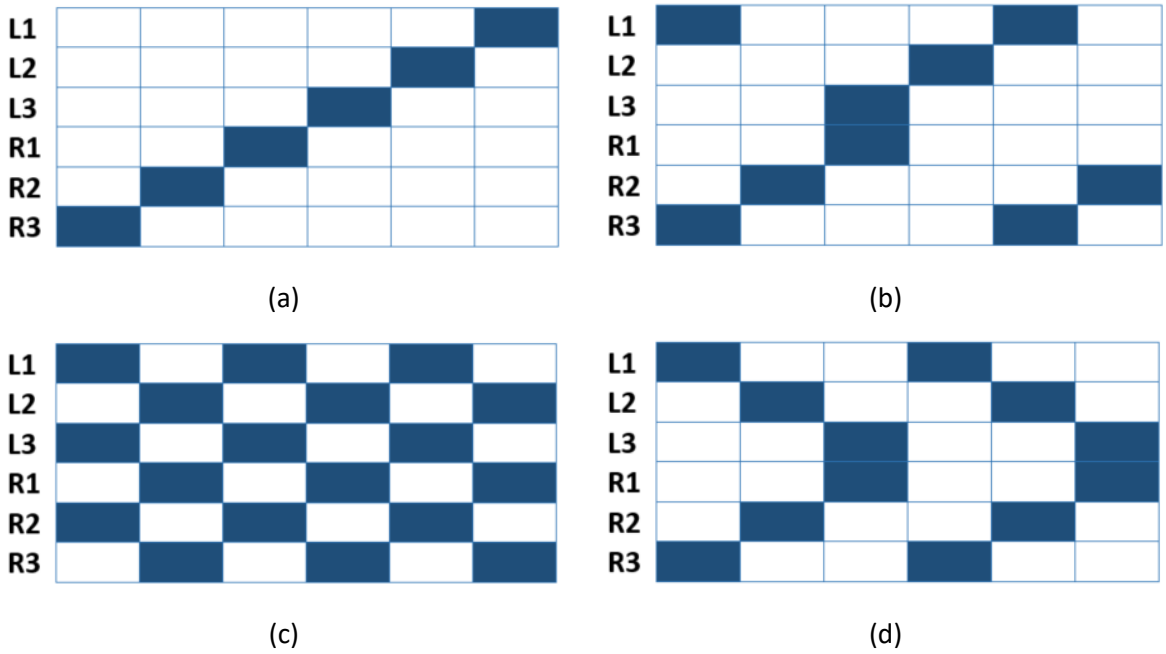


Figure 3.10: Hexapod gait diagrams depicting leg sequencing through time (along horizontal axis) for (a) metachronal, (b) ripple, (c) tripod, and (d) typical tetrapod (a.k.a. quadruped) gaits

This section discussed factors related to the design, construction, and locomotion of hexapods at large, in order to assist in the understanding of the explanations of the robotic system used, as well as design considerations for the control system, which are presented in later chapters.

The sections which follow focus more specifically on theory related to the modelling and control of robotic systems, with particular regard to aspects applicable to the current study.

3.2 Transformation Mathematics and Kinematics

Of course, before being able to study the motions associated with robotic systems, it is necessary to be able to describe the locations of the various bodies and objects of interest. These could include not only parts of the robot itself, but also items which it uses and interacts with, or which are present in its environment [64]. At the most fundamental level, objects, specifically rigid bodies, can be described in space by just their *position* and *orientation*. Consequently, methods of representing and manipulating these properties mathematically are of great interest and importance in robotics.

In order to facilitate the description of how objects are related to each other, in three-dimensional space, a coordinate system, or *frame*, is rigidly attached to each object [64]. This way, an object can be described by the position and orientation of its frame relative to a particular coordinate system.

Since frames, themselves, are coordinate systems, any frame can act as the reference coordinate system within which to describe the position and orientation of a body [64]. Therefore, it is frequently necessary to be able to *transform*, or change the description of bodies, between various frames. As a result, the mathematics related to describing bodies in space involves a variety of representations and

operations, for manipulating their position and orientation quantities, with respect to different coordinate systems.

Excellent and detailed treatments of this mathematics can be found in numerous texts on the subject. One of the quintessential resources which includes foundations of spatial descriptions and transformations (along with a range of other fundamental topics in robotics) is *Introduction to Robotics: Mechanics and Control* by Craig [64]. For a more summarized presentation of the mathematics pertaining to transformations, with specific emphasis on representing orientation, the reader is pointed to the reference text by Diebel [65].

Hence, the purpose of this discussion is not to provide an in-depth coverage of the mathematics referred to above. Instead, the focus is on bringing the key aspects of interest to the reader's attention, as well as highlighting some of the imperative considerations regarding different conventions currently used to define orientation. The information presented in the remainder of this section is based on the resources mentioned above, i.e. [64], [65].

3.2.1 Frame Descriptions

As mentioned above, frames require both a position and orientation, relative to a chosen coordinate system, to be fully described in space. The two primary methods of representing these quantities will be discussed in more detail below, followed by an overview of general coordinate frame transformations, as well as other methods of parameterizing orientation, specifically when describing the attitude of a robot body.

3.2.1.1 Position Vectors

Of the two attributes used to describe a frame, position is quite straightforward. As is generally the case in mechanics, the position of a point in three dimensional space is represented by a 3 × 1 vector [64]. In a Cartesian coordinate system, the three components of the vector denote the x-, y-, and z-positions of the point, with the position vector, $\boldsymbol{p}$, being expressed mathematically as

$$\boldsymbol{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}. \tag{3.1}$$

When considering the representation of attitude for rigid bodies such as robots, two coordinate systems are of particular interest [65]:

- The *world coordinate system*, which is fixed in inertial space and is generally used as the reference frame. A subscript w is used to identify the position of this coordinate system, denoting its origin as $\boldsymbol{p}_w$.

- The *body coordinate system*, which is rigidly attached to the body whose orientation, or *attitude*, is required to be described. A subscript b is used to identify the position of this coordinate system, denoting its origin as $\boldsymbol{p}_b$.

If just these two coordinate systems are used, the expression of points and vectors in one of the two frames can be distinguished by simply using a prime symbol. Thus, if a vector $\boldsymbol{p}$ represents a point expressed in the world coordinates, then $\boldsymbol{p}'$ can be used to express the same point in body coordinates.

Accordingly, the origins of the two frames expressed in their own coordinates, i.e. $\boldsymbol{p}_w$ and $\boldsymbol{p}'_b$, are both zero. However, the origin of the world frame expressed in body coordinates, $\boldsymbol{p}'_w$, and the origin of the body frame expressed in world coordinates, $\boldsymbol{p}_b$, are non-zero, in general.

These fundamentals are sufficient to be able to represent the position of a rigid body in the two main coordinate systems of interest. This leads to the second attribute used to describe frames: orientation, which is the subject of discussion in the following sub-section.

3.2.1.2 Rotation Matrices

The orientation of a frame relative to some other frame is denoted by a 3 × 3 rotation matrix. When multiplied with a vector, a rotation matrix rotates the vector while preserving its length. More specifically, the rotation matrix, $\boldsymbol{R}$, associated with the attitude of a rigid body transforms a vector represented in world coordinates into one represented in body coordinates [65]. Thus, for $\boldsymbol{z} \in \mathbb{R}^3$, a vector expressed in the world coordinates, and $\boldsymbol{z}' \in \mathbb{R}^3$, the same vector expressed in the body coordinates, the following relations apply:

$$\boldsymbol{z}' = \boldsymbol{R}\boldsymbol{z} \tag{3.2}$$

$$\boldsymbol{z} = \boldsymbol{R}^T\boldsymbol{z}', \tag{3.3}$$

where $\boldsymbol{R}^T$ is the transpose of $\boldsymbol{R}$.

The mapping of vectors between coordinate frames, as described in Equations (3.2) and (3.3), effectively involves projecting the vector components onto the unit vectors of the frame being rotated into, i.e. the destination frame. Thus, the rotation matrix, $\boldsymbol{R}$, is simply a stack of the unit vectors that describe the world and body coordinate systems. In particular, the rows of $\boldsymbol{R}$ are the unit vectors of the body frame expressed in the world coordinates, while the columns of $\boldsymbol{R}$ are the unit vectors of the world frame as expressed in the body coordinates [65]. Mathematically, this is represented as

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{x}}_b^T \\ \hat{\boldsymbol{y}}_b^T \\ \hat{\boldsymbol{z}}_b^T \end{bmatrix} = [\hat{\boldsymbol{x}}_w' \quad \hat{\boldsymbol{y}}_w' \quad \hat{\boldsymbol{z}}_w'].$$ (3.4)

Care should be taken not to confuse the prime symbols in Equation (3.4), which indicate the vectors being expressed in the body coordinates, with the superscripted T symbol, which is used to represent the transpose of a vector or matrix. This notation will be used consistently throughout this dissertation, with prime symbols never being used to express a transpose.

Since unit vectors are orthonormal, it can be seen in Equation (3.4) that so are the rows and columns of the rotation matrix. Hence, from linear algebra it is known that

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^T,$$ (3.5)

where $\boldsymbol{R}^{-1}$ is the inverse of $\boldsymbol{R}$. The physical interpretation of this property, with respect to rotation matrices, is that, with $\boldsymbol{R}$ representing the rotation of the body frame relative to the world frame, the rotation of the world frame relative to the body frame is simply given by the transpose of $\boldsymbol{R}$. This is exactly what is represented mathematically in Equations (3.2) and (3.3).

While the elements of the rotation matrix can also be interpreted and calculated in a few other ways, details of this can be found by the reader at the sources provided above. At this stage, it is more pertinent to bring to light an important factor related to the definition of rotation matrices: two possible conventions exist, and are currently used, for defining rotation matrices.

The definitions in Equations (3.2) – (3.4) are based on the convention used in [65], where the rotation matrix describes a rotation from the world coordinates to the body coordinates. Conversely, the other convention defines the rotation matrix as that which maps from the body coordinates to the world coordinates, and this is the convention followed by Craig [64].

Using $\boldsymbol{S}$ to denote a rotation matrix defined by the latter convention, the vector mapping relations equivalent to Equations (3.2) and (3.3) become,

$$\boldsymbol{z} = \boldsymbol{S}\boldsymbol{z}'$$ (3.6)

$$\boldsymbol{z}' = \boldsymbol{S}^T\boldsymbol{z}.$$ (3.7)

Comparing Equations (3.2) and (3.3) to Equations (3.6) and (3.7), it is clear that,

$$\begin{aligned} \boldsymbol{R} &= \boldsymbol{S}^T \\ \boldsymbol{S} &= \boldsymbol{R}^T. \end{aligned}$$ (3.8)

Hence, the relation between the rotation matrices of the two conventions is that they are the transposes of each other. Consequently, the individual elements of the two rotation matrices are linked by the following relation:

$$r_{ij} = s_{ji}, \tag{3.9}$$

where r_{ij} is the element in the ith row and jth column of $\boldsymbol{R}$, and s_{ji} is the element in the jth row and ith column of $\boldsymbol{S}$.

Although converting between rotation matrix descriptions in the two conventions is a simple task of taking the transpose of the matrix, it is crucial to ensure that the rotation matrix being used is in the correct convention before performing the associated mathematical operations on it, a fact that will become more evident in Chapter 5. The remainder of this section will persist with the convention of [65] for presenting operations related to rotations and frame descriptions.

In many instances, it is beneficial to describe the rotation of a frame as a series of sequential rotations performed on the frame. To this end, a useful property of rotation matrices is that they can be multiplied together to yield an equivalent rotation matrix, which applies the same overall rotation as the final rotation obtained by sequentially applying the original rotation matrices. This can be illustrated more coherently by an example of rotating a vector $\boldsymbol{z}$ between arbitrary coordinate systems, denoted by prime and double prime symbols, as follows:

Let

$$\boldsymbol{z}' = \boldsymbol{R}_a \boldsymbol{z} \tag{3.10}$$

$$\boldsymbol{z}'' = \boldsymbol{R}_{b/a} \boldsymbol{z}'. \tag{3.11}$$

Then,

$$\boldsymbol{z}'' = \boldsymbol{R}_{b/a} \boldsymbol{R}_a \boldsymbol{z} = \boldsymbol{R}_b \boldsymbol{z}, \tag{3.12}$$

where

$$\boldsymbol{R}_b = \boldsymbol{R}_{b/a} \boldsymbol{R}_a. \tag{3.13}$$

It should be noted from the above equations "that the rotations are applied in the reverse order" [65]. Thus, in Equation (3.13) $\boldsymbol{R}_b$ represents a rotation sequence in which $\boldsymbol{R}_a$ is applied first, followed by the rotation $\boldsymbol{R}_{b/a}$. Since the rotation matrix descriptions in the convention of [64] are the transposes of those used here, it is natural that the multiplication order of sequential rotations, in that convention, is the opposite of that observed in Equation (3.13). That is, the convention in [64] applies sequential rotations by post-multiplying them.

3.2.1.3 General Coordinate Frame Transformations

The expressions in Equations (3.2) and (3.3) apply to general vectors, which are relative quantities that lack a position in space. On the other hand, transforming a *point* between two coordinate systems requires that the offset to the origin of the target frame is first subtracted (yielding a vector quantity), before the rotation matrix is applied. Thus, using the definitions of points and coordinate frame origins in Section 3.2.1.1, general transformations can be written as

$$\boldsymbol{p}' = \boldsymbol{R}(\boldsymbol{p} - \boldsymbol{p}_b) = \boldsymbol{R}\boldsymbol{p} + \boldsymbol{p}'_w \tag{3.14}$$

$$\boldsymbol{p} = \boldsymbol{R}^T(\boldsymbol{p}' - \boldsymbol{p}'_w) = \boldsymbol{R}\boldsymbol{p}' + \boldsymbol{p}_b. \tag{3.15}$$

Therefore, it can be seen that general transformations of points from one frame to another involve both a translation and a rotation. This result illuminates the need for both *position* and *orientation*, or *attitude*, information to describe rigid bodies in space. The combination of position and attitude information represents the *pose* of a rigid body.

Although position is most intuitively denoted by $\boldsymbol{p}_b$, which expresses the position of the origin of the body frame in world coordinates, it is just as valid to utilize $\boldsymbol{p}'_w$, which represents the origin of the world frame with respect to the body coordinates. From Equations (3.14) and (3.15), it is evident that these two representations of position are related to one another as follows:

$$\boldsymbol{p}'_w = -\boldsymbol{R}\boldsymbol{p}_b \tag{3.16}$$

$$\boldsymbol{p}_b = -\boldsymbol{R}^T\boldsymbol{p}'_w. \tag{3.17}$$

While rotation matrices are one form of characterizing attitude, other parameterizations of attitude do also exist, as will be discussed in the section which follows.

3.2.1.4 Other Parameterizations of Attitude

Using rotation matrices to represent and manipulate attitude can be cumbersome, while also presenting the possibility of being computationally expensive to store and perform operations with. Two other popular methods for parameterizing the attitude, or orientation, of rigid bodies are Euler angles and quaternions.

3.2.1.4.1 Euler Angles

One of the most common ways of encoding attitude information is with a set of three Euler angles, which offers an intuitive and easy to use method of working with orientations. These three angles represent three sequential coordinate rotations and can describe any rotation. A wide variety of rotation sequences and associated Euler angle triples are possible [65], but these will not be elaborated in this discussion. The focus here is on one specific rotation sequence which is prevalent in the fields of aerospace engineering and computer graphics, as well as robotics. While a number of

names are given to this set of rotation angles, such as *Cardan angles* or *Tait-Bryan angles*, it is customary in aeronautics and robotics to refer to them simply as *Euler angles*. More specifically, the three individual angles, ϕ, θ, and ψ, are known as *roll*, *pitch*, and *yaw*, respectively.

These three angles are applied in a ZYX rotation sequence; that is, first the frame is rotated about the z-axis through an angle of ψ, then this new frame is rotated about its y-axis by θ, finally being followed by a rotation about the x-axis of the latest frame through an angle of ϕ. Mathematically, this is expressed with rotation matrices as

$$\boldsymbol{R}_{ZYX}(\phi,\theta,\psi) = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi), \tag{3.18}$$

where $\boldsymbol{R}_{ZYX}(\phi,\theta,\psi)$ is the rotation matrix representation of the Euler angle rotation sequence and $\boldsymbol{R}_i(\alpha)$, with $i \in [x,y,z]$, is a rotation matrix denoting a coordinate rotation, or rotation about the i-axis by an arbitrary angle α.

This sequence of rotation is illustrated in Figure 3.11, where the coordinate frame described by (x,y,z) coordinates is rotated into the (x',y',z') frame, with intermediate rotations into the (x''',y''',z''') and (x'',y'',z'') coordinates.

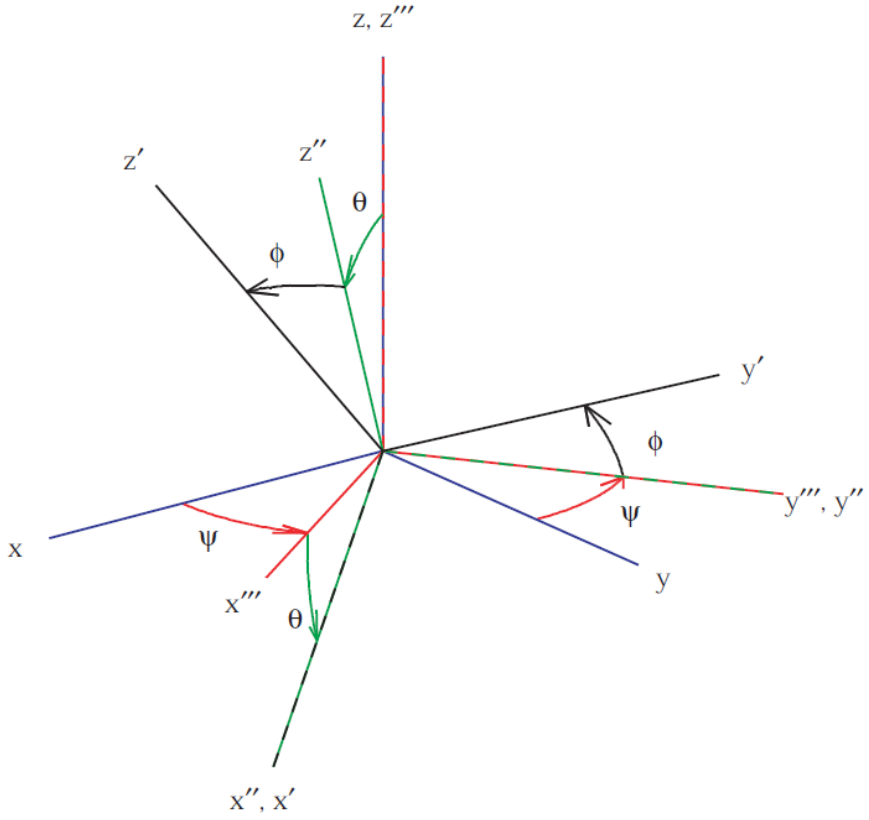


Figure 3.11: Rotation of a frame using the ZYX Euler angle sequence [65]

The multitude of operations and calculations that can be performed with Euler angles will not be discussed here. However, the one important operation that the reader's attention will be drawn to is the conversion from a rotation matrix to the ZYX Euler angles. Letting $\boldsymbol{u}_{ZYX}(\boldsymbol{R})$ be the vector function of $\boldsymbol{R}$ that yields the associated roll, pitch and yaw angles, the following mapping holds:

$$\boldsymbol{u}_{ZYX}(\boldsymbol{R}) = \begin{bmatrix} \phi(\boldsymbol{R}) \\ \theta(\boldsymbol{R}) \\ \psi(\boldsymbol{R}) \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(r_{23}, r_{33}) \\ -\operatorname{asin}(r_{13}) \\ \operatorname{atan2}(r_{12}, r_{11}) \end{bmatrix}, \tag{3.19}$$

where $\operatorname{atan2}(y, x)$ is used to compute the arc tangent, taking into account the quadrant in which the arguments lie.

A drawback of using Euler angles is that, for some rotation sequences, they can have singular configurations. For the ZYX sequence, this singularity occurs at pitch angles of $\theta = \frac{\pi}{2} + n\pi$, for $n \in \mathbb{Z}$ (the set of all integers). Physically, this corresponds to a vehicle pitched vertically upward, or inverted with the nose pointing straight down. In such singular configurations, the roll and yaw angles cannot be distinguished. This problem of singularities can be overcome by using quaternions.

3.2.1.4.2 Quaternions

Along with providing the advantage of not having mathematical singularities, attitude parameterizations using quaternions are also more accurate than Euler angles when integrating incremental changes in angular velocity over time. An extensive compilation of quaternion mathematics is available (many fundamentals of which can be found at [66]), but this does not pertain to the current study. This section simply serves to present the mathematical definition of a quaternion, along with the necessary conversion equations.

A quaternion, $\boldsymbol{q}$, is comprised of four quantities and defined as [66]

$$\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}, \tag{3.20}$$

where $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ are hyperimaginary numbers.

In Equation (3.20), the quantity q_0 is the real, or scalar, part of the quaternion, while $q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$ makes up the imaginary, or vector, part. It is also customary to represent the quaternion by a 4 × 1 vector, as

$$\boldsymbol{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ \overline{\boldsymbol{q}} \end{bmatrix}, \tag{3.21}$$

where $\overline{\boldsymbol{q}}$ denotes the imaginary part of $\boldsymbol{q}$.

Quaternions are usually normalized to have a unity norm (in which case they can also be referred to as unit quaternions), that is

$$\|\boldsymbol{q}\| = \sqrt{q_0^2 + q_1^2+q_2^2+q_3^2} = 1. \tag{3.22}$$

Two important conversions related to quaternion representations of attitude are those from a unit quaternion to a rotation matrix and from a unit quaternion to ZYX Euler angles. The mapping function, $\boldsymbol{R}_q(\boldsymbol{q})$, which calculates a rotation matrix from the elements of a unit quaternion is defined as

$$\boldsymbol{R}_q(\boldsymbol{q}) = \begin{bmatrix} q_0^2 + q_1^2-q_2^2-q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2+q_2^2-q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2-q_2^2+q_3^2 \end{bmatrix}. \tag{3.23}$$

Consequently,

$$\boldsymbol{z}' = \boldsymbol{R}_q(\boldsymbol{q})\boldsymbol{z} \tag{3.24}$$

$$\boldsymbol{z} = \boldsymbol{R}_q(\boldsymbol{q})^T\boldsymbol{z}'. \tag{3.25}$$

From Equations (3.19) and (3.23) it follows that the mapping function which converts unit quaternion quantities into roll, pitch, and yaw angles is defined as

$$\boldsymbol{u}_{ZYX}\left(\boldsymbol{R}_q(\boldsymbol{q})\right) = \begin{bmatrix} \phi(\boldsymbol{R}_q(\boldsymbol{q})) \\ \theta(\boldsymbol{R}_q(\boldsymbol{q})) \\ \psi(\boldsymbol{R}_q(\boldsymbol{q})) \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(2q_2q_3 + 2q_0q_1, q_0^2 - q_1^2-q_2^2+q_3^2) \\ -\operatorname{asin}(2q_1q_3 - 2q_0q_2) \\ \operatorname{atan2}(2q_1q_2 + 2q_0q_3, q_0^2 + q_1^2-q_2^2-q_3^2) \end{bmatrix}. \tag{3.26}$$

3.2.1.5 Final Note on Conventions

As was mentioned earlier, all the operations and relations presented in Section 3.2.1.4 are based on the rotation matrix convention adopted by Diebel [65]. Naturally, these functions are formulated differently for rotation matrices defined in the other convention, owing to their elements being in diagonally opposite locations to those used here, as well as the different order of multiplication used when applying these rotations sequentially. Therefore, the importance of using the appropriate functions, for converting between attitude representations with rotation matrices defined in a particular convention, cannot be stressed enough. If the functions defined for one convention are used on rotation matrices in the other convention, inadvertent transposing of the rotations will effectively take place, switching the attitude representation from the intended frame to the other frame along the way. Furthermore, rotation sequences may be applied in the wrong order, resulting in a very different orientation description than the one intended. Clearly, the potential for errors is great if care is not taken to adhere to a single convention when working with rotation matrices.

A further caveat to keep in mind is that the vector descriptions of quaternions may be defined with the scalar part of the quaternion, q_0, appearing as the last element of the vector, rather than the first. That is, the quaternion, $\boldsymbol{q}$, may be represented, just as validly as in Equation (3.21), by [66]

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_0 \end{bmatrix} = \begin{bmatrix} \overline{\boldsymbol{q}} \\ q_0 \end{bmatrix}. \tag{3.27}$$

Accordingly, when using the quantities of a quaternion parameterization of attitude, it is necessary to be mindful of the order in which those quantities are provided.

3.2.2 Kinematics

In addition to frame descriptions and transformation mathematics, another important topic regarding the study of robotics is that of kinematics, which is the science of motion without regarding the forces that cause it [64]. A thorough treatment of the kinematics related to the PhantomX is available at [14], [67]. The purpose of this section is merely to highlight some of the considerations of kinematics germane to this study, as well as defining terminology utilized in this regard.

3.2.2.1 Link Descriptions

As described in Section 3.1.2, the legs of a walking robot are made up of various rigid body segments connected in a chain by joints. Each of these segments is referred to as a *link* [64]. From a mathematical perspective, Craig [64] describes a link to be considered only as a rigid body defining "the relationship between two neighbouring joint axes" in a kinematic chain. Each joint axis is defined by a line in space, and, for any two axes in three-dimensional space, there exists a line that is mutually perpendicular to both axes. This mutually perpendicular line always exists and is unique, except in the case of the two axes being parallel. For the mathematical model of a kinematic chain, the links are defined by these mutual perpendiculars between the joint axes.

Thus, the link length is determined as the length of this line connecting the joint axes, regardless of the geometry of the actual body which makes up a link. This is illustrated in Figure 3.12, where a schematic of an arbitrary link between two joints is shown, along with the mutually perpendicular line between them defining the link length. Clearly, the link line does not necessarily coincide with the physical geometry of the link in space. It is important to keep this observation in mind for the discussion on the kinematic parameters of the PhantomX, in Chapter 4. In the case of the last link of a chain, the link length is generally defined from the last joint to a point on the end-effector. On the PhantomX leg, the end point of interest is the tip of the foot, hence the last link line is projected out to this point, which generally makes contact with the ground.

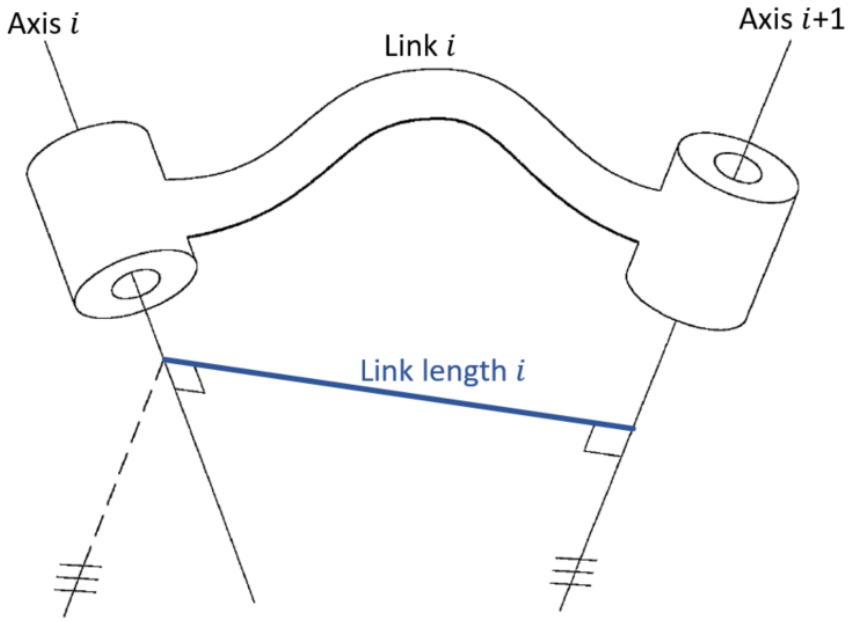


Figure 3.12: Schematic representation of an arbitrary link, showing the length of the link line mutually perpendicular to the joint axes (adapted from [64]*)*

The Denavit-Hartenberg (DH) notation, mentioned in Section 2.5.2, utilizes the link definition introduced above, along with a few other parameters describing the relationships between consecutive links and joints, to obtain mathematical descriptions, or models, of mechanisms. The derivation of such a kinematic model, using DH notation, for the PhantomX can be found at [14], [67].

3.2.2.2 Kinematic Description Mappings

For a mechanism forming a kinematic chain, the position of its end-effector can be represented in three different ways. One method of specifying the positions of all the links, for a mechanism having n degrees of freedom, is with a set of n joint position variables, in the form of an $n \times 1$ joint vector [64]. The space encompassing all such joint vectors is referred to as *joint space*.

By using a kinematic model of the chain, such as that touched on above, the position of the end-effector, specifically the position of the foot in the case of a robot leg, can be computed from the joint space description relative to an orthogonal, or Cartesian, coordinate system. Accordingly, such a description of the kinematic chain is said to be in *Cartesian space*, sometimes referred to operational space, task-orientated space, or simply task space [64].

Before being able to use the joint positions to determine the Cartesian space description, however, it is necessary to consider how those joints are actuated. Kinematic joints are hardly ever actuated directly. In some cases, two actuators could work together in a differential pair to impart motion to a single joint, while another possibility is to use a linear actuator to rotate a revolute joint through the motion of a linkage [64]. Even if electric motors are used as rotational joints, the angular position

values of the motor outputs do not necessarily coincide with those of the joints, as defined in the kinematic model. Since sensory equipment generally measures the positions of actuators, this set of values in *actuator space* makes up the actuator vector, from which the joint vector can be computed.

Figure 3.13 portrays the three different descriptions of a kinematic chain, with arrows indicating mappings between descriptions in different spaces. The solid arrows represent forward mappings, while dashed arrows indicate inverse mappings. Thus, in robotics the term *forward kinematics* (mentioned a few times in Chapter 2) refers to the computation of the end-effector position, in Cartesian coordinates, from joint position information (which usually also involves mapping from the actuator vector to the joint vector). On the other hand, *inverse kinematics*, or IK, is related to determining the joint positions, and subsequently the actuator positions, required to achieve a desired end-effector position.

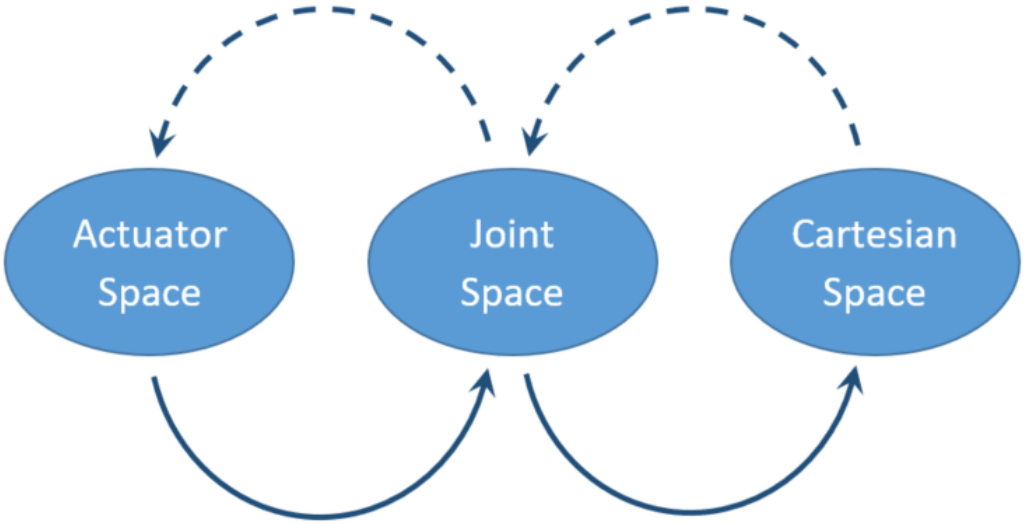


Figure 3.13: Mappings between kinematic descriptions – solid arrows indicate forward mappings, while dashed arrows indicate inverse mappings (recreated from [64]*)*

This background information (Section 3.2.2) facilitates the discussions in Section 4.3.1, which involve some of the parameters required for the kinematic mappings used in controlling the motion of the PhantomX.

3.3 Virtual Model of a Hexapod Robot

Based on the analysis of posture control techniques in Section 2.5, it was concluded that the Virtual Model Control (VMC) strategy was the obvious choice to be implemented in this study, owing to its effectiveness, simplicity and use of well understood, linear control laws.

Before proceeding with detailed discussions about the robotic platform and control related matters, it will be valuable to introduce the basic theory of creating a virtual mathematical model for the

hexapod robot. An idea of the rationale behind VMC was given throughout Section 2.5, but it will be explained more coherently here.

The basic premise of VMC is to treat the main body of the robot as a mass in free space and create a model with virtual elements attached to it [11]. These virtual elements passively act to regulate the body posture relative to a desired point, by applying the necessary restorative forces to it. As a result, the mechanical characteristics of the elements can be freely chosen to obtain the desired dynamic behaviour of the robot body. Then, to achieve this behaviour, the legs are controlled in a manner which emulates the effects of the virtual elements on the robot body [6].

A vital assumption which underlies this approach is that the dynamics of the legs are insignificant enough, when compared to that of the body, to allow them to be neglected in the model (but still be used to apply control actions on the body). While, strictly speaking, this is not generally the case and the leg motions can have a substantial effect on the body's motion, in practice it is usually adequate to cast these unmodelled dynamics off as disturbances to the system and have the control system compensate for them. Considering the application, in most cases the simplicity and ease of implementation resulting from this assumption far outweigh the small gains in accuracy that could be obtained by exploring much more complicated methods, which do take the additional dynamics and nonlinearities into account. This is evidenced by the numerous, successful implementations of the technique presented in the literature.

Given that the virtual elements, and their mechanical characteristics, can be freely chosen in the model, the question of what virtual elements to select arises. Although it is certainly possible to make use of any mathematical model of an element which one desires, it is usually best to make use of elements which represent physical components, as they provide a physical analogue to the virtual model and often have well understood mechanical characteristics.

To this end, it is customary to make use of linear springs and dampers in the model. In particular, translational spring-dampers are used for linear motion along axes, while torsional spring-damper elements are employed for angular motion about axes. Since the robot body's height above the ground and orientation relative to the horizontal plane are commonly of interest in posture control tasks, the translational spring-dampers are generally applied in the vertical direction, while torsional spring-dampers are typically applied about the body's roll and pitch axes.

The spring-damper elements are attached to the robot's COM at one end, while the other end is grounded such that, at the steady state, or static, position, the linear and angular displacements are at the posture reference values, or setpoints. That is, the virtual elements impart the necessary forces

to restore deviations in the body posture from the setpoints, in accordance with the dynamic behaviour they are designed to achieve.

A schematic, free-body diagram of a virtual model, as described above, is depicted in Figure 3.14. The legs are indicated by dashed lines, to make it explicit that they are ignored in the model. Furthermore, the full mass of the robot, M (including the masses of the legs), is assumed to act at the COM, G, of the main body, indicated in the diagram as a weight, Mg, where $g = 9.81\ \mathrm{m/s^2}$ is the gravitational acceleration.

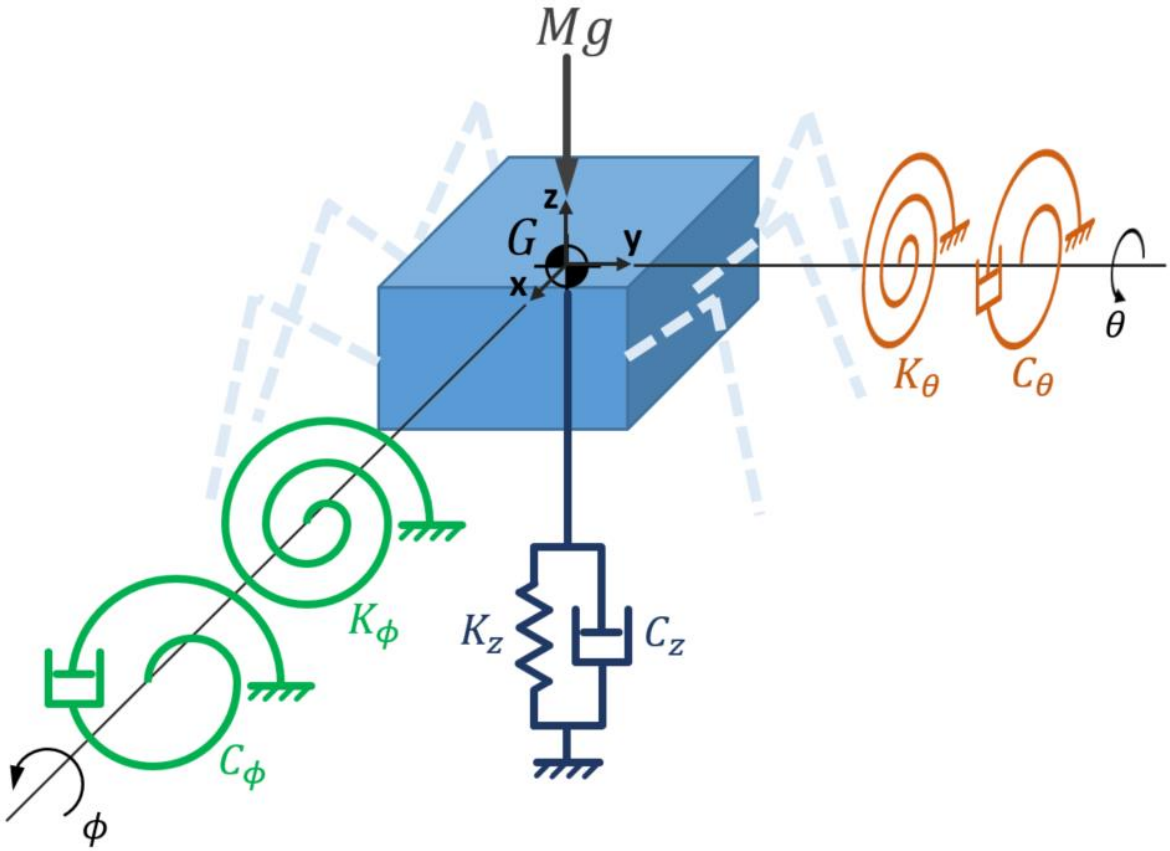


Figure 3.14: Free-body diagram of the virtual model of a hexapod robot

By taking the sums of force in the z-direction (ΣF_z) and the moments about the x- and y-axes (ΣM_x and ΣM_y), the following equations of motion describing the virtual model of the robot body are obtained:

$$\begin{aligned} M\ddot{p}_{G,z} = \Sigma F_z &= -K_z \Delta p_{G,z} - C_z \Delta \dot{p}_{G,z} \\ I_\phi \ddot{\phi} &= \Sigma M_x = -K_\phi \Delta\phi - C_\phi \Delta\dot{\phi} \\ I_\theta \ddot{\theta} &= \Sigma M_y = -K_\theta \Delta\theta - C_\theta \Delta\dot{\theta}, \end{aligned} \tag{3.28}$$

where $M \equiv$ Mass of the robot,

$I_\phi \equiv$ Moment of inertia about the x-axis (roll moment of inertia),

$I_\theta \equiv$ Moment of inertia about the y-axis (pitch moment of inertia),

$\ddot{p}_{G,z} \equiv$ Vertical acceleration of robot body,

$\ddot{\phi} \equiv$ Angular acceleration about x-axis (roll acceleration),

$\ddot{\theta} \equiv$ Angular acceleration about y-axis (pitch acceleration),

$\Delta p_{G,z} \equiv$ Vertical displacement of robot body from the static position,

$\Delta\phi \equiv$ Angular displacement about x-axis (roll angle) from the static position,

$\Delta\theta \equiv$ Angular displacement about y-axis (pitch angle) from the static position,

$K_z,\ K_\phi, K_\theta \equiv$ Virtual spring stiffness coefficients,

$C_z,\ C_\phi, C_\theta \equiv$ Virtual damping coefficients,

and, as is commonly used throughout this dissertation, single and double overdots denote the first and second time derivatives, respectively, of the symbols they modify.

A point to be noted (which will become important when using this model for control) is that since the above equations have been derived for displacement relative to the static position, the robot's weight is compensated by the static load on the vertical spring. Hence, the gravitational force does not appear in the mathematical model.

A further observation which can be made from Equations (3.28) is that by assuming the virtual elements to be acting on the centre of mass of the robot, rather than some other point on its body, the equations of motion of the three posture variables are decoupled, greatly simplifying the dynamics for control. More specifically, each direction of motion can be treated as an independent, single degree of freedom system, allowing for each control variable to be controlled individually.

Thus, these equations, describing the virtual model of the robot, provide the basis for determining the forces which the controller will eventually be required to apply to the robot, to regulate its posture with the desired dynamic behaviour. This will become more apparent in Chapter 6. In fact, it will be seen that selecting the controller gains effectively boils down to specifying the desired mechanical characteristics of the virtual elements used in the model derived above.

3.4 Interaction Control Methods in Manipulator Robotics

In Section 2.5 it was alluded to that most of the implementations of posture control in the literature make use of joint torque control to achieve the desired foot forces, but that joint actuators do not always offer the capability for torque control. This is especially prominent in commercially available platforms, and it will become more apparent in Section 4.2 that this is indeed the case for the platform used in this study.

Nevertheless, it is still necessary to be able to control the foot forces, and so other force control methods are required to be explored, especially those that can attain force control by appropriately controlling the motion of the leg. Given that a robot leg is kinematically equivalent to a manipulator, it is applicable to look into motion-based, force-control methods used for manipulators. The conceptual overview of such control methods in the remainder of the section is drawn from [68].

3.4.1 Interaction Tasks with Manipulators

Manipulator robots are increasingly required to operate autonomously in unstructured environments, rather than just in extremely controlled environments which can be characterized and modelled very well beforehand. It is thus quite discernible that utilizing purely motion control strategies, to handle interactions between the robot and its environment, quickly becomes inadequate. During such interactions, physical constraints are imposed by the environment on the geometric paths which the end-effector may follow, a condition referred to as *constrained motion*. Inevitable modelling errors and uncertainties may lead to escalation of the contact forces, ultimately resulting in potentially unstable behaviour during the interaction or, eventually, even saturation of the joint actuators or breakage of the object which the robot is handling.

Therefore, appropriate control strategies are required for the task of interaction of the robot with the environment. Interaction control strategies can be classified into one of two categories: *indirect force control* and *direct force control* [68]. The major distinction between the two techniques is that the former modifies motion control based on measured forces, without closing a force feedback loop, while the latter can achieve a desired contact force value, owing to the closure of a force loop [68].

Indirect force control schemes include *compliance control* (also referred to as *stiffness control*) and *impedance control*, wherein the contact force is related to the end-effector position error through a mechanical stiffness or impedance, the parameters of which can be adjusted as desired. A manipulator operated with impedance control can be described by the dynamics of an equivalent mass-spring-damper system, for which the contact force acts as an input.

On the other hand, direct force control strategies involve explicitly regulating the contact force to a desired value, rather than managing the motion of the manipulator under the influence of external

forces. When the environment model is known in detail, a commonly employed strategy in manipulators is that of *hybrid position/force control*, which attempts to control position in the unconstrained task directions while purely controlling force in the constrained directions. However, detailed models of the environment are usually unavailable, and in such cases *inner/outer motion/force control* strategies can be useful, in which a force control loop encloses the inner motion control loop of the manipulator [68]. Thus, in addition to feeding the desired contact forces in the constrained directions to the outer force control loop, it is also possible to provide the desired end-effector motion as input to the inner loop. This results in a *parallel control* composition of both force control and motion control actions, with force control dominating motion control along constrained task directions.

At this stage, it is worth mentioning that a comprehensive examination of the modelling of manipulator dynamics, as well as detailed derivations of motion and force control strategies, is not really relevant to this study and will thus not be presented. Instead, a brief, conceptual overview of the various control techniques mentioned above will be given, to provide an idea of the methodologies followed for solving interaction control problems. Details of the applied solutions will be elaborated when discussing their implementation on the robot used for this study in Chapters 6 and 7.

3.4.2 Indirect Force Control

When a manipulator is controlled to achieve a desired end-effector position, by means of proportional control action on the position error, the presence of a contact force results in the manipulator not attaining its desired position. At steady state, this type of proportional, feedback control results in the manipulator behaving like a generalized spring [68], as it exhibits a position offset proportional to the applied force. Thus, such a control approach offers active compliance (hence the name compliance control), or compliant behaviour during interaction, rather than completely rigid motion.

Compliance control is designed only to deliver a specified static behaviour during interaction, with no ability to specify or influence the transient motion of the manipulator. For dynamic behaviour, mass and damping must also be included besides stiffness, resulting in impedance control. In manipulators, such active impedance is usually achieved by using inverse dynamics based motion control, which determines the required actuator control inputs by taking into account the full dynamics of the manipulator, including the effects of contact forces on the end-effector's acceleration. At steady state, the resultant behaviour of the manipulator is analogous to that obtained with the compliance controller, with the incorporation of mass and damping into the impedance model only affecting the *dynamic* behaviour of the end-effector during interaction with the environment.

However, merely using impedance control in this manner, for the motion control, typically results in inadequate trajectory tracking in free space, as the presence of even small disturbances to the end-effector's acceleration could result in undesirably large deviations from the desired trajectory. A potential remedy to this problem is to specify the impedance properties such that a low mass-to-stiffness ratio is obtained, resulting in relatively stiff behaviour of the manipulator. Unfortunately, the drawback of this is often that it directly conflicts with the desire of compliant behaviour from the impedance controller, especially when the manipulator interacts with a rather stiff environment.

For this reason, the actions of motion control and impedance control are usually separated. The motion control action is deliberately made "stiff" to ensure satisfactory disturbance rejection, but instead of tracking the desired end-effector position it acts to track a reference position provided by the impedance controller, without accounting for how motion will be affected by contact forces. In effect, the impedance controller uses the desired position as input, with feedback of the contact forces and suitable integration of the acceleration and velocity components, to generate a position reference to be used by the motion controller.

The resulting control structure consists of an inner motion control loop and an outer impedance control loop, as illustrated in the high-level, conceptual block diagram in Figure 3.15. The impedance properties are selected to ensure an equivalent mass-spring-damper behaviour is obtained by the *position displacement* between the desired position and reference position for the motion controller, rather than by the actual motion of the end-effector.

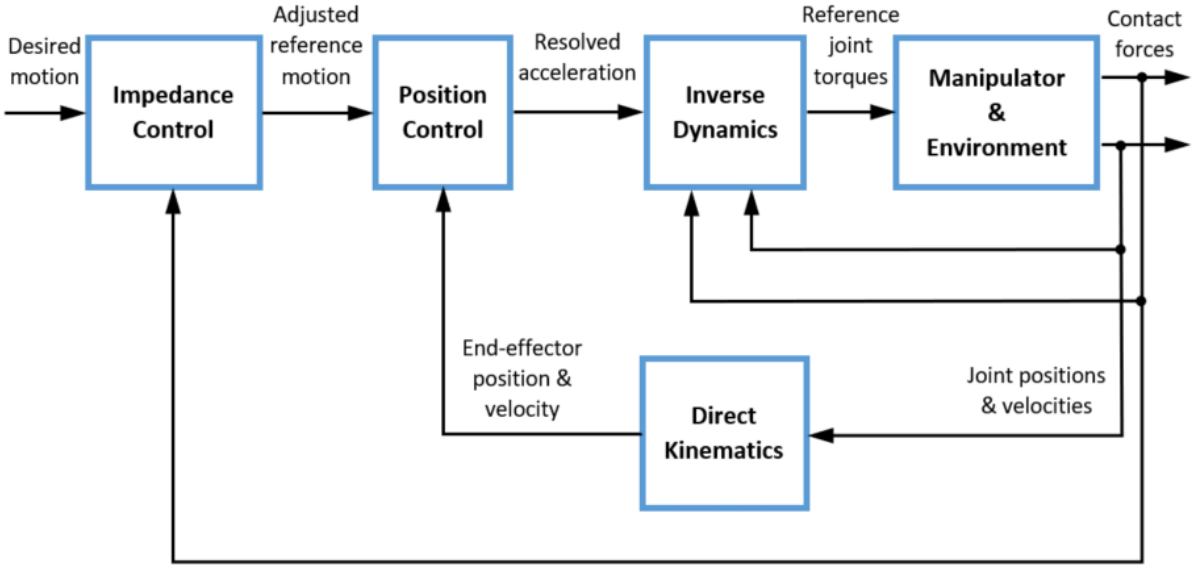


Figure 3.15: High-level block diagram depicting typical impedance control structure in manipulators (recreated from [68]*)*

3.4.3 Direct Force Control

The methods of indirect force control through motion control described above limit the contact force values during interaction with the environment, rather than ensuring precise control of desired contact forces, despite the need for force feedback. However, certain interaction tasks do require that precise contact forces are obtained. For such cases, a different approach can be taken to design a *direct force control* scheme, which acts to diminish the error between the desired and measured forces [68]. This is accomplished by closing an outer force control loop which generates the reference input for the manipulator's motion control system, thus regulating the contact force to a desired value.

It should be noted that while indirect force control methods, such as impedance control, generally operate in the manipulator's unconstrained directions, simply reacting to forces resulting from interaction, precise force regulation is usually required in constrained directions of motion.

Similar to the case of impedance control, the task of the outer force control loop is to provide a position displacement, or offset, by which the reference position for the inner motion control loop can be adjusted. To this end, the position offset can be computed through proportional-integral (PI) control action on the force error. The integral action ensures that, at steady state, the contact force is equal to the desired force, while the proportional action is utilized to improve the transient behaviour during interaction.

Since regulation of the contact force takes place in the manipulator's constrained directions, it overrides the control of the end-effector's motion. As described earlier, in order to retain motion control along the unconstrained task directions, a parallel control approach is generally taken, such that position adjustments are only made for directions in which the force error acts (constrained directions), while position tracking in unconstrained directions remains unaffected. Although, in manipulators this usually requires some redefinition of the integral control action, to provide the inverse dynamics based motion controller with appropriate velocity and acceleration adjustment values in the constrained directions as well. Further details of this composition will not be discussed herein.

A high-level, conceptual block diagram of a typical direct force control structure for manipulators is presented in Figure 3.16. This diagram depicts a parallel composition of motion and force control for the manipulator. Signals without labels are identical to those in Figure 3.15.

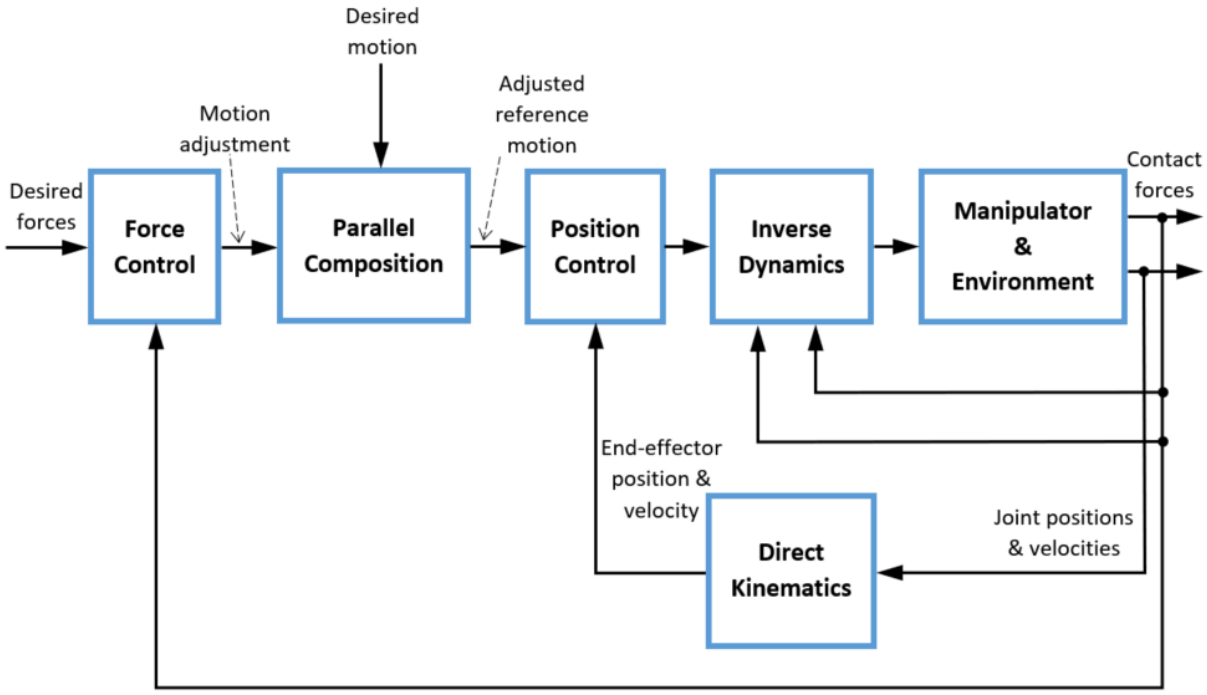


Figure 3.16: High-level block diagram depicting typical direct force control structure in manipulators (recreated from [68]*)*

3.4.4 Final Note on Motion-Based Force Control

It is worthwhile, at this point, to mention that generally the inverse dynamics based motion controllers used on manipulators generate torque reference points as input to the actuators. This might suggest that the methods discussed in the preceding sections are no more useful to a robot with position controllable actuators than the torque control methods utilized in most legged robot literature.

However, it is important to realize that for purely torque control based methods usually found in legged robot literature, the desired outputs of the robot, in the task space, are directly converted into the joint space, using the leg Jacobian, and used to calculate the actuator inputs (joint torque reference values). Thus, torque controllable joints are necessary to achieve the desired motion and forces.

On the other hand, the methods discussed above do not directly calculate torque inputs, but rather focus on relating the desired interaction behaviour to the necessary motion which would achieve this. Thus, the force control loops operate purely in the task space, generating the necessary inputs to the robot's motion control system, which then attempts to achieve this desired motion, producing the desired force or dynamic behaviour as an effect of the motion. In theory, this means that the errand of interaction control can be separated from the method of motion control utilized, regardless of whether that is based on controlling joint torques or positions. Of course, in practical application some additional considerations are required to be made to implement such methods together with different motion control techniques.

Nonetheless, the deduction to be made from this discussion is that the interaction control techniques described above are indeed important to consider, and they do indeed provide meaningful insights for attaining force, or interaction, control with robots having purely position-based motion control capabilities.

3.5 The Robot Operating System (ROS)

Chapter 2 provided a flavour of the complexities involved in robotic control and tasks involving autonomous behaviour. These complexities extend further out across the entire spectrum of robotic and autonomous systems development. Tasks required to create an autonomous, robotic system range from the lowest level of hardware-software communication, through various layers of sensory feedback and motion control hierarchies, all the way up to high-level challenges such as environmental perception, planning and navigation, each of which presents its own set of difficulties. Needless to say, the creation of software to address these problems and achieve the level of autonomy required of robots is no trivial undertaking. This is where the Robot Operating System, or ROS, comes into the picture.

3.5.1 A Brief History of ROS

ROS was designed to bridge the gaps between individual solutions created to solve the large multitude of problems faced in robotics. While various laboratories or research groups are able to focus on specific aspects of robotic development, it goes without saying that no single individual, laboratory, or institution can tackle the entire range of issues on their own [69]. For this reason, ROS was created, from the ground up, to facilitate collaboration in the development of robotics software. The resulting system is a flexible framework for writing complex robot software, which includes conventions and software utilities that assist in the creation of robust capabilities required on a multitude of robotic systems [69].

Before ROS was formally originated, the idea behind such a system began with numerous efforts at Standford University in the mid-2000s, to create prototypes of flexible and dynamic robotics software systems, primarily for application to their in-house robotics research programmes [70]. Willow Garage, which was a California-based robotics incubator, took on the task of extending these concepts further and creating well-tested implementations, as part of their Personal Robotics project, and so the entity of ROS was born in 2007.

Over the next few years, the framework went through many iterations of design and implementation concepts at both Willow Garage and Stanford [70]. The core ideas and fundamental software packages of ROS gradually took shape through development by a large number of contributors, until eventually the first distribution of ROS was released in 2010, with many of the components and application

programming interfaces (APIs) that still make up the current framework today. From the beginning, ROS software development took place under the permissive BSD open source license, at multiple institutions, and for a variety of robots. Over time, ROS has been adopted by much of the robotics research community, growing into a widely-used platform with tens of thousands of users around the world, utilizing ROS in a range of domains from hobby projects to large, industrial automation systems.

With a core philosophy in ROS being to promote collaborative development of generic software components, the entire framework has been designed to be as modular and distributed as possible [71]. The modularity gives users the option to utilize as many or as little of the components of ROS they require, while easily being able to integrate the framework into their own projects. A distributed nature also means that users can leverage the enormous number of packages, made available by the huge community, as well as being able to easily share their own packages for others to use. The integration point of solutions built by the ROS community on top of a common infrastructure offers users many useful features, ranging from drivers for hardware to functionalities universally required on robots, and even tools that assist software development [71].

3.5.2 Fundamental Features and Core Components

Although referred to as the Robot Operating System, ROS is in fact a *meta-operating system* for the robot, which runs in the operating system of the computer or machine on which it is being used. Currently, only Unix-based platforms are supported, with ROS software being primarily developed and tested on Ubuntu and Mac OS X systems, although the community does also provide support for some other Linux platforms [72].

For a robotic system, ROS provides the services expected of an operating system, including hardware abstraction, low-level device control, message-passing and communication between processes, and implementation of commonly used functionality, among others [72], [73]. These services are carried out across a peer-to-peer network of ROS processes, referred to as the *Computation Graph*. This Computation Graph, sometimes referred to simply as the Graph, could run on a single machine or be distributed across multiple machines, in a network of interconnected computers, robots, and other applicable devices.

Processes or programs made to run in ROS can be written in a variety of programming languages, through the use of the related *client libraries* [74]. The ROS client libraries are collections of code which make many of the ROS concepts and functionalities available to the programmer, in their language of choice. Currently, the main languages for which robust support for ROS is focussed on are C++ and Python, with the associated client libraries for these languages being *roscpp* and *rospy*, respectively. Of these two, roscpp is the most widely used and acts as the high-performance library for ROS. On the

other hand, rospy is geared towards taking advantage of the object-oriented, scripting nature of Python, promoting implementation and development speed over runtime performance.

Detailed, technical information about the components which make up ROS programs can be found at the ROS Wiki [75], along with comprehensive examples and tutorials for writing software to run in ROS. In the remainder of this section, the basic features and concepts will be introduced and briefly described, in order to facilitate the explanations about ROS-specific implementations of the software written in this study. In essence, the overview below acts as a glossary of ROS-related terminology that is used throughout the remainder of this dissertation.

Firstly, software in ROS is organized into *packages* [76]. Packages are the most elementary items which can be built and released in ROS, with the idea being that they provide a usefully organized collection of resources that deliver a particular function or solution. Some examples of resources that can be contained within packages include ROS runtime processes, ROS-dependent libraries, datasets, and configuration files.

As discussed above, the actual runtime processes in ROS run as part of the Computation Graph. Processes and data are provided to the Graph and communicated across it in various ways. These various concepts, which make up the Computation Graph, are the most pertinent to the discussions of software written for ROS in this study, and are thus elaborated below [76]:

- **Nodes** – These are the actual runtime processes referred to above, which perform computations and are written using the client libraries. Since ROS is designed to be highly modular, nodes are generally written to carry out very specific tasks and be as "lightweight" as possible, with robotic control systems typically comprising of many nodes.
- **Master** – The ROS Master acts as a "nameservice" in the Computation Graph, providing name registration and lookup to the rest of the elements of the Graph. Nodes communicate with the Master, reporting registration information for their topics and services (see below), allowing for nodes to find, and communicate with, each other.

  It should be noted that nodes only communicate with the Master for lookup information, connecting directly to other nodes when sending, or receiving, information to, or from, them.
- **Parameters** and the **Parameter Server** – The Parameter Server, currently part of the Master, provides a central storage point for data, known as parameters, to be stored during runtime. Parameters can be accessed by nodes, or even users via the command line, at any time and can include any useful data such as robot platform information, configuration properties, hardware and software settings, and controller properties, to give just a few examples.

- **Messages** – Communication between nodes is accomplished by passing messages to each other. A message is effectively a data structure, comprising of fields of specific data types. The most basic messages represent standard, primitive data types, such as integers and Booleans, as well as arrays of these types. More complex messages are created as arbitrarily nested structures of these basic types and arrays.
- **Topics** and the **Publish / Subscribe Model** – A topic is a unique name used to identify the context and relevance of the content of a message. Messages are routed across the Graph using publish / subscribe semantics, in which a node sending out a message *publishes* it to a given topic, while nodes requiring particular kinds of data *subscribe* to the appropriate topics. A single topic may have multiple, concurrent publishers or subscribers, while a given node may publish and subscribe to multiple topics. The model is designed to separate entities that generate information from those that utilize the information, and so publishers and subscribers are generally unaware of each other's existence. This enhances the modular architecture of ROS and allows for complex systems to be built up more easily.

  As an example, one could consider the case of a sensor driver, which would simply publish the relevant sensory information over a particular topic, while any nodes which require this information would subscribe to the topic. This way, the driver is not dependent on how the nodes use the information, or even, in fact, on the presence or existence of nodes in the system which utilize this information. Furthermore, the nodes could be written such that they are independent of the method in which sensory information is obtained from hardware, or possibly even independent of the particular hardware or sensors used, as the nodes are only interested in the resulting information which comes through on the topic.
- **Callbacks** – In order for nodes to be able to use the messages published on topics they are subscribed to, callback functions are required to be written and linked to the appropriate subscribers in the node's source code. When a message is published over a topic, ROS calls the associated callback functions of the subscribed nodes, passing the message into the callbacks as an input, and loading the callbacks onto their nodes' callback queues, for processing. A callback defines how the contents of a message should be handled, with most callbacks often simply copying the message contents over to local data structures in the node, to be used at a later stage of computation within the node.

  In roscpp, callback queues are processed by *spinners*, with a variety of spinning models available that offer the user with options between single- or multi-threaded spinning, as well as blocking or non-blocking spinning. More details can be found at [77].

- **Services** – Although the publish / subscribe model offers many advantages, its many-to-many, one-way transport methodology is not suitable for request / reply interactions, which can often be required in distributed systems. In ROS, these types of interactions are facilitated via services, which are similar to messages but are defined as a pair of message structures, specifying the data for both the request and the reply. Similar to the case of topics, a service is offered by a node under a unique name, and client nodes make use of the service by sending out a request message and awaiting the providing node's reply.

  From a coding perspective, this functionality is usually exposed to the developer by the client libraries in a manner that emulates a remote procedure call.
- **Bags** – ROS bags represent collections of stored message data, in a ROS-readable format. Message data can be saved in bags by recording the necessary topics. These stored data can then be inspected, analysed, and even played back in ROS to simulate the presence of the publishing nodes during development and testing.

A schematic diagram, illustrating the basic concepts of inter-node communication in ROS, is presented in Figure 3.17.

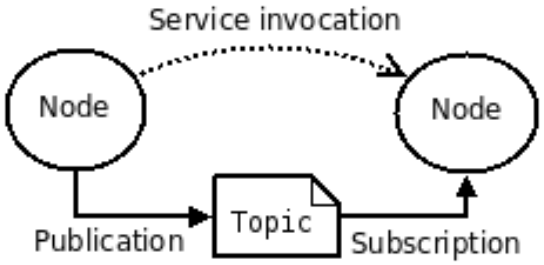


Figure 3.17: Schematic diagram illustrating basic concepts of inter-node communication in ROS [76]

Chapter 4: Description of Robot Platform

In this chapter the specific hexapod robot platform used in this study will be introduced and discussed, in detail. Existing sensors will be given an overview, while the new foot force sensors incorporated onto the platform will be presented. Furthermore, the ROS software utilized for locomotion and sensor feedback will be elaborated upon, serving as a basis for the discussions on digital implementation of the designed control systems, which will follow in later chapters.

4.1 PhantomX MK-II Hexapod Robot

In Chapter 1 it was stated that the PhantomX MK-II hexapod robot [15], introduced in Section 2.3.3 (see Figures 2.8 and 4.1), forms the basis of the robotic platform utilized in this study. This is the second generation of PhantomX hexapods distributed by Trossen Robotics. The PhantomX MK-II was originally chosen by Lubbe [14], based on a trade-off study comparing it to other commercially available hexapods, and acquired by the TSO unit of the CSIR for research purposes.

Figure 4.1: PhantomX MK-II hexapod robot - front view [15]

4.1.1 Original Hardware

It can be seen in Figure 4.1 that the PhantomX MK-II, constructed primarily from Plexiglas [15], has a rectangular body structure with an extended middle. In addition to the central legs being shifted outward, the front and back legs are mounted at 45° angles to the longitudinal plane of the body. An arachnid type leg configuration is used for the legs, each of which have three DOF, resulting in a total of eighteen DOF for the robot.

Each leg joint is actuated by Dynamixel AX-12A hobby-class servo motors [78], which have a nominal operating voltage of 11.1 V and are able to provide feedback of angular position, angular speed, temperature, load, and input voltage. While the AX-12A servos can be controlled with respect to either position or speed, offering an optional "wheel mode" which allows the servos to spin freely, it should be noted that these actuators do not provide any direct torque control functionality. On the hexapod, the actuators are powered by an 11.1 V Lithium polymer (LiPo) battery, mounted in between the lower and upper plates of the main body.

Compared to most hexapod platforms used for research, for example [7]–[10], [13], [24], [35], [36], [42], this robot is relatively small, with the main body being approximately 260 mm long and 200 mm wide. The platform also has a relatively low mass of approximately 2 kg.

Three kit options for the PhantomX MK-II were offered by Trossen Robotics during its production run, namely *Barebones*, *No Servos*, and *Comprehensive* [15]. All but the *Barebones* kit included the necessary electronic components to operate the robot. The *Comprehensive* kit was procured by TSO for research.

A central component of the electronic system is the ArbotiX-M Robocontroller from Vanadium Labs [79], an Arduino compatible microcontroller developed specifically for controlling Dynamixel servo motors and running locomotion software for hobby-class legged robots. The ArbotiX-M Robocontroller is displayed in Figure 4.2.

Figure 4.2: ArbotiX-M Robocontroller from Vanadium Labs [79]

Another principal piece of hardware included with the electronics is the ArbotiX Commander, a hand-held, gamepad style controller which uses an XBee radio communication module to wirelessly control ArbotiX powered walking robots [80]. Figure 4.3 shows the ArbotiX Commander.

Figure 4.3: ArbotiX Commander – wireless controller for ArbotiX powered walking robots [80]

Updates to the hardware system used in this study are discussed in Sections 4.3.1 and 4.4.

4.1.2 Original Software

Vanadium Labs have developed open source locomotion software called NUKE (an acronym for Nearly Universal Kinematics Engine), which generates inverse kinematics and gait engines for a variety of legged robot configurations [15]. While NUKE is the official software offered, maintained, and supported by Trossen Robotics for the PhantomX MK-II, an alternative, open source firmware package, referred to as the Phoenix Code, is also available and can be used with the PhantomX hexapods, among other legged robots [81]. The Phoenix Code is a more advanced kinematics engine, providing an assortment of walking gaits and robot motion features. In fact, most of the original software which made up the Phoenix Code was based on kinematics work done by Zenta [81], who was introduced in Chapter 2. Lubbe [14] made use of this locomotion software, on the ArbotiX-M Robocontroller, for the PhantomX.

However, in this study, the software system was overhauled completely. Details about the upgraded software are discussed in Section 4.3 below.

4.2 Sensory System

As discussed in Section 2.3.5, the sensory system of a mobile robot platform is of great significance and essentially determines its capabilities. More specifically, it was concluded in Section 2.3.5.2 that the task of posture control of a hexapod requires a suitable proprioceptive sensory system. Lubbe [14] initially outfitted the PhantomX with such a system of sensors for the purpose of testing a state

estimation algorithm with the hexapod. In addition to the position feedback built into the joint actuators, an IMU and foot contact sensors were incorporated into the robotic platform.

While the majority of this sensory system was suitable for use of posture control in this study, the foot contact sensors were required to be replaced by more capable force sensors. Hence, only a brief description of the joint position sensors and IMU will be given below, followed by a discussion on the integration of new force sensors on the robot's feet.

4.2.1 Joint Position Sensors

The angular positions of the leg joints are determined from the position feedback provided by the Dynamixel AX-12A joint actuators (shown in Figure 4.4). In "joint mode" these servo motors have an operating range of 300° at a resolution of 0.29° [78]. The angular position is represented digitally by a 2-byte integer and can take on values in the range of 0 – 1023 (mapping to 0° – 300°), for both goal and feedback positions, as illustrated by the diagram in Figure 4.5.

Data such as goal and feedback positions are stored in registers of the Control Table of the built-in microcontroller inside the Dynamixel servo [78]. This Control Table is made up of an EEPROM (electrically erasable programmable read-only memory) area and a RAM (random-access memory) area. Data in the EEPROM area remains stored even when the servo is powered off and includes the servo's identifier (ID) number, along with other servo information and settings. On the other hand, data registers in the RAM area are reset to an initial value when powering on the servo. Examples of information stored in the RAM area include position or speed instructions, as well as feedback measurement values.

A noteworthy advantage of the AX-12A's angular position feedback, pointed out by Lubbe [14], is that the resistive potentiometer which measures angular displacement is mounted on the output shaft of the actuator, rather than on that of the motor (before the gearbox). The result is that the actual output of the actuator is used for feedback, accounting for factors such as play in the gearbox, which affect the motion of the actuator.

Figure 4.4: Dynamixel AX-12A servo motor used as joint actuators for the PhantomX MK-II [78]

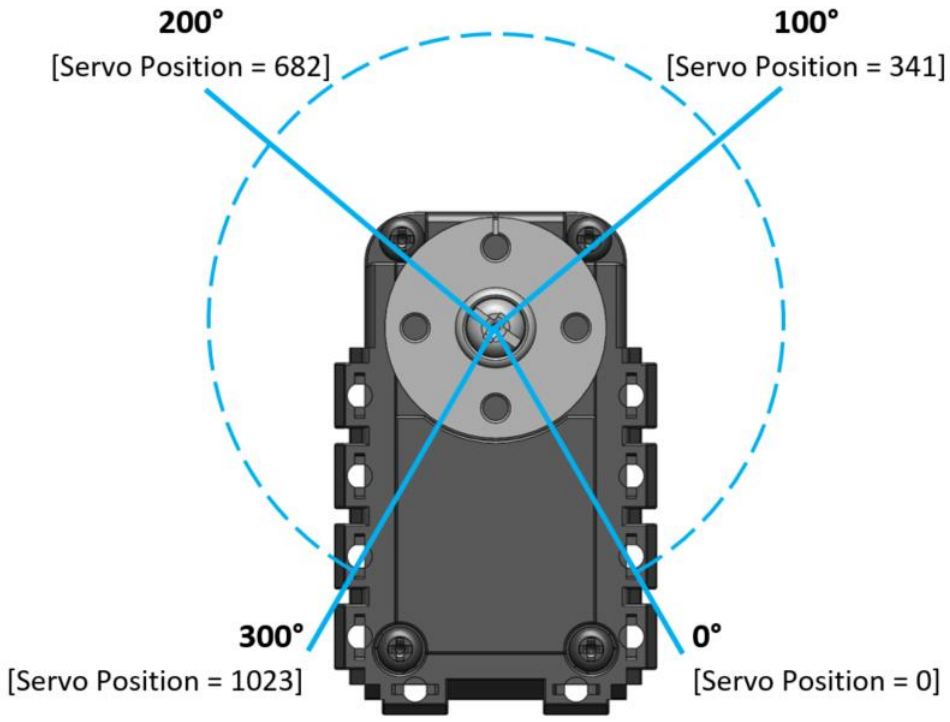


Figure 4.5: Mapping between angular output and digital position values of AX-12A servo motor

4.2.2 Inertial Measurement Unit

For inertial sensing, the PhantomX platform was equipped by Lubbe [14] with a LORD MicroStrain® 3DM-GX3® -25 miniature Attitude Heading Reference System (AHRS) (see Figure 4.6). The 3DM-GX3® -25 combines a triaxial accelerometer, a triaxial gyroscope, a triaxial magnetometer, and temperature sensors, along with an onboard processor running a sensor fusion algorithm to provide orientation and inertial measurements [82]. The inertial measurements are fully calibrated and include acceleration, angular rate, magnetic field, deltaTheta, and deltaVelocity vectors, while the computed orientation estimates can be output as Euler angles, a rotation matrix, or a quaternion, at rates of up to 1000 Hz [82].

Figure 4.6: LORD MicroStrain® 3DM-GX3® -25 miniature Attitude Heading Reference System [82]

Having such extensive measurement capabilities, the 3DM-GX3® -25 AHRS is effectively a combination of an IMU and a heading sensor. At this point, the astute reader may point out that heading sensors such as the magnetometer in the 3DM-GX3® -25 are, in fact, exteroceptive sensors, as they depend on the Earth's magnetic field for heading measurements. However, the magnetometer measurements were neither used by Lubbe [14], nor will they be used in this study, relying only on measurements made by the IMU and thus adhering to the choice of proprioceptive sensing.

4.2.3 Foot Force Sensors

In the study by Lubbe [14], FlexiForce™ A201 force sensors were used on the feet to determine their ground contact status. Although they were used only in a binary, on-off fashion, to determine whether or not there was contact, these resistive sensors have the ability to measure forces up to 100 N, as their resistance decreases with increasing force applied to them. However, the accuracy and robustness of these sensors were simply insufficient for continued use on the hexapod, especially for the task of posture control. In fact, the sensors on most of the feet had already incurred significant damage and were splitting apart at the onset of the current study, having not managed to hold up to the impact between the robot's feet and ground, during walking motions in the study by Lubbe [14]. For these reasons, it was necessary to equip the robot with new force sensors which would be more appropriate for the application.

4.2.3.1 OptoForce 3D Force Sensors

Accurate and sensitive foot force measurements are required for posture control, while the sensing equipment used is required to withstand the cyclical loading and striking impacts experienced by the feet throughout walking motion. Taking these factors into account, 3D optical force sensors from OptoForce were selected for use on the PhantomX.

These OptoForce 3D sensors measure force magnitudes in three axes and are developed primarily with robotics applications in mind, especially geared toward low budget research programmes [83]. While traditional force sensing technology, such as strain gauges, can be limiting for robotics use, due to bulky construction, inadequate robustness, and high costs, among other factors, the unique design of OptoForce's optical, silicon based force sensors addresses just these issues [84].

OptoForce sensors make use of a single deformable structure, made of silicon, to measure forces in all three axes [84]. A photodiode is located within the structure, emitting light onto a reflective layer on the underside of the sensing surface. By measuring the reflection of the light as the silicon surface deforms, both the magnitudes and directions of the imparted forces can be reconstructed. This effectively decouples the sensing elements from the contact surface, greatly improving robustness and reliability of the sensor, which is able to withstand overloading as high as 200% of the nominal

capacity. Furthermore, the silicon materials used by OptoForce have Shore A hardness ratings between 50 and 87, making them as abrasion resistant as typical shoe heels[2].

The design also results in a compact sensor, which measures the necessary forces without adding unnecessary bulk and weight to the robotic platform. Figure 4.7 illustrates the construction of an OptoForce 3D sensor with a semi-spherical contact surface, like the ones used on the PhantomX. OptoForce also offers flat-top sensors, but these are only available for nominal capacities of 100 N and larger [85].

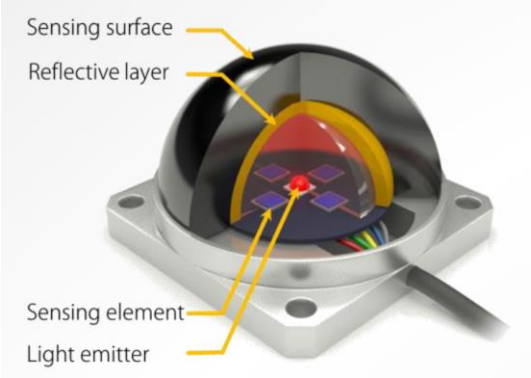


Figure 4.7: Construction of semi-spherical OptoForce 3D force sensor [84]

Semi-spherical 3D force sensors are available from OptoForce, as standard, in nominal capacities of 10 N, 40 N, and 100 N [85] (the 40 N sensor is used in this study). Considering that only statically stable walking will be utilized in this study, the robot's weight is expected to be supported by at least three legs at all times. Since the PhantomX platform weighs approximately 20 N, if the weight is distributed more or less evenly among at least three legs, the load on each foot is not expected to exceed about 7 N, which might suggest that 10 N force sensors would be appropriate for the application. However, it is important to take into account that, especially on uneven terrain, a situation may arise in which one or more feet do not have a foothold on the ground, resulting in the entire weight being briefly loaded onto only one or two of the legs. Even in the case of three legs supporting the robot, the weight may certainly not be distributed evenly, possibly resulting in more than 10 N of load being supported by a single foot.

Another factor to consider, which is perhaps even more pertinent, is that the robot's weight is hardly ever supported in a steady state manner. As the robot walks, the legs constantly switch between swing and stance phases, resulting in large forces being experienced by the feet when touching down onto the ground. This is particularly pronounced during uneven terrain walking, as a foot may strike into an obstacle, or slightly elevated section of terrain, earlier in the swing phase than expected, inducing a

[2] More information on the Shore hardness scale can be found at [124].

high-speed impact (and thus significantly larger force) onto the foot than in the case of just supporting a portion of the robot's weight while standing. Not only should the force sensor be able to measure these larger forces, but it should also be able to withstand these large impacts and overloads without incurring damage. Although the 10 N sensor would be able to handle up to 20 N of overload, the considerations above suggest that in certain situations a foot may instantaneously experience even larger forces, and the sensor should be sufficiently robust to withstand them. It is thus apparent that the 40 N 3D force sensor is the appropriate choice for use on the PhantomX, in terms of both sensing capacity and robustness.

An option which was offered by OptoForce during personal e-mail communication with a Product Consultant at OptoForce, in February 2016, was to custom manufacture semi-spherical, 3D force sensors with a nominal capacity of 20 N, if required. However, it was advised that the cost per sensor would be significantly higher and assured that that the accuracy and reliability of the 40 N sensor measurements would be more than sufficient for the application, even at loads much smaller than the nominal capacity, owing to the full-scale nonlinearity not exceeding 2 – 3%. This discussion affirmed the choice of the 40 N nominal capacity force sensors, and they were thus procured by TSO, not only for this study, but also for future research to be conducted using the PhantomX robotic platform.

The OptoForce 3D force sensor, model OMD-20-SE-40N, which is utilized in this study, is depicted in Figure 4.8 with the coordinate frame in which forces are measured. Having a semi-spherical contact surface means that the sensor can only measure compressive forces in the z-direction, with a nominal capacity of 40 N, while both magnitudes and directions of forces in the xy-plane can be measured with nominal capacities of ± 10 N (i.e. 10 N in either direction of applied force) [83]. Furthermore, typical deformations of the contact surface at nominal capacity are 2 mm in the z-direction and ± 1.5 mm in the xy-plane, with the diameter of the semi-sphere being 20 mm.

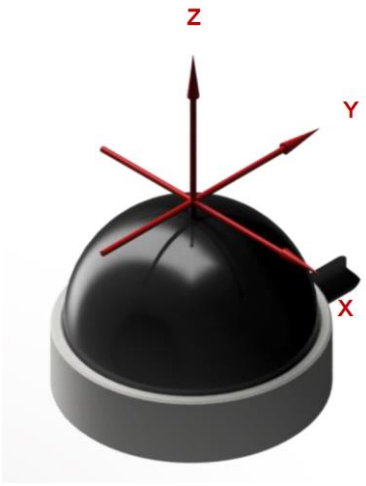


Figure 4.8: Model of OptoForce 3D force sensor (OMD-20-SE-40N), indicating coordinate frame of measured forces [83]

According to the product datasheet [83], the OMD-20-SE-40N sensor exhibits a full-scale nonlinearity value of 2%, in both the z- and xy-directions, and has measurement resolutions of 2.5 mN and 2 mN in the z- and xy-directions, respectively. The sensors also feature low crosstalk and hysteresis values of <5% and <2%, respectively, and can operate in temperatures between -40 °C and +80 °C. In addition, the sensors have an IP65 rating for being dust and water proof, further increasing their robustness.

Data acquisition (DAQ) units are supplied with the sensors, with each DAQ having the capacity to connect up to four sensors, which cannot be detached from the DAQ [83]. A variety of interface types can be used to communicate with the DAQ, including Universal Serial Bus (USB) and Ethernet (multiple protocols supported), at sampling frequencies of up to 1000 Hz. More detailed information on the OMD-20-SE-40N sensors can be found in the datasheet [83], attached in Appendix A-1.

While only six force sensors are required for the six feet of the PhantomX, a total of eight sensors were acquired, for redundancy and future use. These eight sensors were supplied connected to two DAQs (four sensors per DAQ), along with two USB cables (Mini Type B male to Type A male) and sensitivity reports from the calibration process of each sensor (in the compressive z-direction only), which can be found in Appendix A-2.

4.2.3.2 Force Sensor Testing

The force sensors are calibrated by OptoForce, at nominal capacity in the z-direction, before supplying them. However, it was deemed appropriate to carry out a basic test of the sensors to evaluate their accuracy across their measurement range and ensure that the sensitivity values provided by the manufacturer are suitable for use.

This testing is reported comprehensively in Appendix A-3 and all test data are provided in Appendix A-4. Based on the analysis of these results, it was concluded that the force measurements provided by the OptoForce 3D force sensors, using the manufacturer provided sensitivity values for each sensor, are repeatable, reproducible, and sufficiently accurate for use on the PhantomX robot platform.

4.2.3.3 Integration of Force Sensors onto Robotic Platform

Having determined the suitability of the OptoForce sensors for the PhantomX, it was necessary to mount them to the robot's feet. Since they could not be mounted to the original feet, new feet were required to be designed and manufactured for the PhantomX MK-II, as discussed in Appendix B.

The newly designed foot can be seen attached to the PhantomX leg in Figure 4.9, with the OptoForce sensor mounted. The two DAQs were mounted onto the main body of the hexapod (these can just be made out in Figure 4.15).

Figure 4.9: Tibia segment of PhantomX with new foot and OptoForce sensor in contact with ground

4.3 Software

While the ArbotiX-M Robocontroller works well to control servos for locomotion of legged robots, it becomes limiting when performing more advanced or complex tasks, such as hierarchical, closed-loop control of the robot, which requires feedback of the servo positions (among other sensor data).

In the study by Lubbe [14], servo position feedback was also required for the state estimation. Given that the servos were being controlled with the ArbotiX-M, it was necessary to modify the Phoenix Code to read the "Present Position" RAM registers of the built-in microcontrollers in each of the Dynamixel servo motors, at a set frequency. This array of servo positions was also required to be sent via a serial communication link to a single board computer on the robot, which was used for data acquisition of all the sensory feedback.

A significant issue which was encountered by Lubbe [14] was that the highest frequency at which the servo positions could be read was 20 Hz. Attempting to request servo positions at any higher frequency would start interrupting the robot's motion, causing it to become intermittent and "jerky."

Another factor contributing to the intermittent motion observed by Lubbe [14] was the amount of data being transferred through the Arbotix-M to retrieve the servo positions. As a remedy, before publishing them over the serial port, the servo position values were divided by four to reduce the measurement range to 0 – 255, resulting in 256 possible values which could then be represented by a 1-byte integer, rather than the standard 2-byte integer used by the servos. Consequently, the position feedback resolution was reduced by a factor of four (despite only reducing the size of the data being transferred by a factor of two), while the accuracy of feedback was also affected by rounding the result of the division, as an integer value was required.

For more advanced control tasks, these factors would undoubtedly be problematic. In addition to reading servo positions, control commands would need to be communicated to the ArbotiX-M and executed by the Phoenix Code. Having all this additional servo communication hamper the robot's motion would be of no use, while being confined to use small data types would limit the control capability which could be achieved.

Furthermore, Trossen Robotics themselves declare that the Phoenix Code base is quite complex and can be difficult to customize and integrate into other projects [81]. In personal communication, over the Trossen Robotics Forum portal's private messenger, in April 2016, Kevin Ochs also highlighted some inadequacies with the Phoenix Code firmware, including a lack of consistent coding standards and a poorly structured code base, with insufficient comments or algorithmic clarity.

It was evident from the above considerations that the ArbotiX-M Robocontroller and Phoenix Code would not be suitable for this study. It was, therefore, decided to port the platform over to ROS for all the locomotion and control. Some of the advantages of ROS were highlighted in Section 3.5, while Section 2.5.3 introduced the locomotion software stack for ROS, available for hexapods such as the PhantomX. ROS was thus a clear choice for bringing together the tasks of locomotion, sensor data acquisition, posture estimation, and posture control for the robotic platform in this study.

4.3.1 ROS Locomotion Software Stack: `hexapod_ros`

The ROS hexapod locomotion software stack introduced in Section 2.5.3 is referred to as `hexapod_ros` and can be found on GitHub at [17]. This software stack, coded in C++, is a combination of various ROS packages, each dedicated to a particular function of the robot or aspect of the firmware, which includes nodes that execute the main locomotion control and handle communication with a wireless controller for teleoperation.

The main node, `hexapod_controller`, implements the gait and IK engines, as well as managing servo communication for the joint actuators [17]. Currently, the gait engine only offers a sinusoidal tripod gait (i.e. during walking, the trajectories of the swing legs are defined by sinusoidal curves)

which was chosen for its simplicity and smooth transitions between steps. Although it is unfortunate that more gaits are not available, the tripod gait is sufficient for use in this study.

All the above functions were clustered into a single ROS node in order to achieve minimal servo communication latency, keeping the gait smooth [17]. During the personal communication mentioned above, Kevin Ochs also pointed out the low latency realized when communicating with the servos, especially when using a USB2AX [86] serial communication device, which acts as a Dynamixel servo interface over USB (see Section 4.4).

With regard to teleoperation, the `hexapod_teleop_joystick` node handles communication between the remote controller and the `hexapod_controller` node via Bluetooth, translating joystick commands into ROS messages, which act as control instructions for the `hexapod_controller` node. This teleoperation node is primarily set up to make use of a PlayStation® 3 (PS3) controller, but it can also be used with a PlayStation® 4 (PS4) controller.

As mentioned in Section 2.5.2, the `hexapod_ros` stack is agnostic to 3 DOF and 4 DOF hexapods. This is achieved by defining the properties and settings for the hexapod being used in configuration files and only loading these parameters into the locomotion node at runtime (via the ROS Parameter Server). These parameter files, saved in the `hexapod_description` package, are defined in YAML[3] syntax and include properties related to the robot's dimensions and kinematics, servo settings and configuration, and gait engine settings such as walking speed and leg lift height, among others.

Although a parameter file is included as standard for the PhantomX MK-II, based on the work by Renée Love (see Section 2.5.2), small measurement errors of the hexapod's kinematics were noticed. In addition to these dimensions which required correction, the tibia length (see link length description in Section 3.2.2.1) of the PhantomX leg was required to be revised, taking into account the new feet and force sensors. The dimensions in the PhantomX parameter file were accordingly updated for this study.

Another property required to be updated was the "servo offset" value for each joint actuator. The servo offset represents the angular position of the servo motor when the joint angles, relative to the coordinate frames used in the kinematic model, are at 0° (see discussion on kinematic description mappings in Section 3.2.2.2). Since the IK engine does not use DH notation to define the leg's kinematic model, the servo offset values used by Lubbe [14] could not be used directly with the `hexapod_ros` stack. Figure 4.10 shows the configuration of the leg when the joint angles are 0° in the IK engine.

[3] The acronym YAML originally stood for "Yet Another Markup Language," but was later redefined as a recursive acronym "YAML Ain't Markup Language."

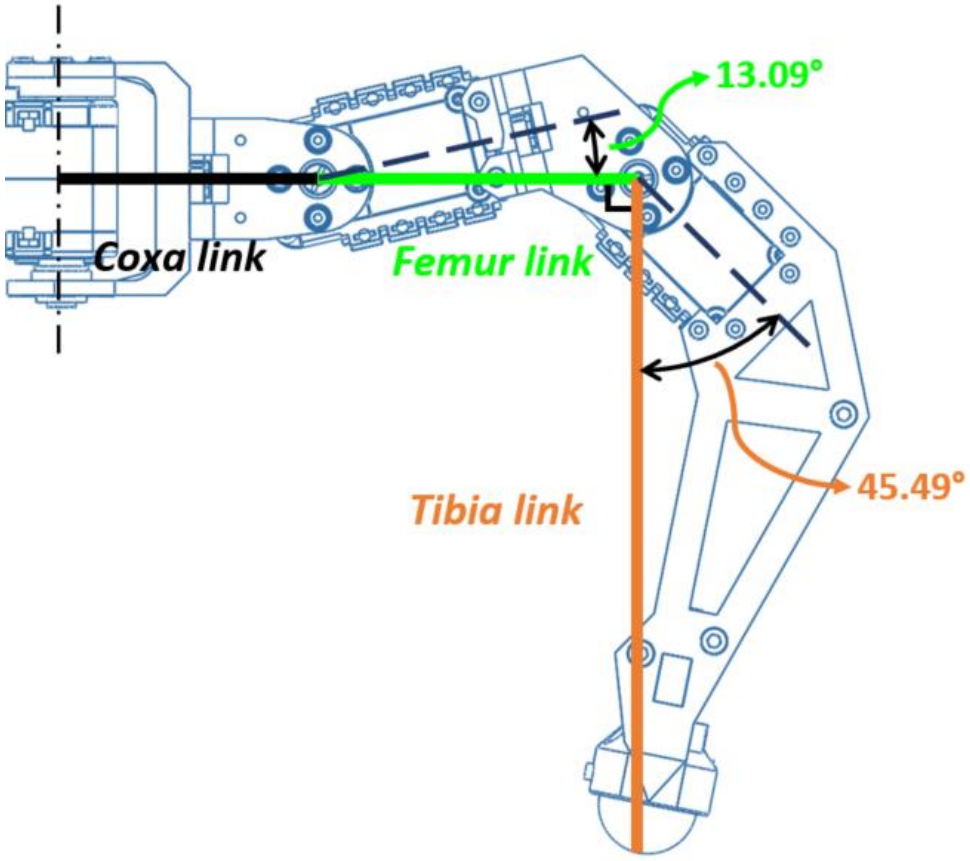


Figure 4.10: Default PhantomX leg configuration for `hexapod_ros` IK engine, with offsets between link lines and servo centre planes indicated for femur and tibia leg segments

It should be noted that the `hexapod_controller` node treats the centre position of the servo as the reference, hence using a range of -150° – +150°, rather than 0° – 300°. For this reason, the coxa servos have a servo offset parameter of 0°. Based on the figure above and the mounting of the servos on the coxa and femur brackets, the servo offsets for the femur and tibia joints, relative to the centre position of the servos, were calculated as follows:

$$\begin{aligned} Femur\ servo\ offset &= -13.09° \\ Tibia\ servo\ offset &= 13.09° + (90° - 45.49°) = 57.6°. \end{aligned} \tag{4.1}$$

Furthermore, the walking parameters, such as initial foot positions, standing height, and leg lift height, were updated to be more consistent with the default gait behaviour when using the Phoenix Code, as the original `hexapod_ros` parameters resulted in a less natural walking motion. These settings were determined through manual measurements made of the robot, when running on the Phoenix Code (with the ArbotiX-M Robocontroller). Updating the initial foot positions also aligned the tibias to be more perpendicular to the ground while the robot is standing up, allowing for virtually all of the contact force to act along the z-direction of the force sensors (on flat terrain). The updated values of all the parameters and settings discussed above are presented in Appendix C.

In order to control the robot's locomotion with the `hexapod_ros` software, the necessary nodes are required to be run in a ROS core. The Indigo Igloo distribution of ROS was used for this study, as `hexapod_ros` was developed in this distribution and has full compatibility with it. ROS and the necessary packages are installed and operated on a laptop computer running the Ubuntu 14.04 ("Trusty Tahr"), Linux-based operating system.

Having set up the robot platform for locomotion with ROS, it was necessary to also establish sensor communication in ROS. The necessary drivers used for sensor feedback are outlined in the following section.

4.3.2 Sensor Feedback in ROS

Minor modifications made to the `hexapod_ros` software to obtain servo position readings, along with the ROS drivers used for IMU and force sensor feedback, are discussed separately in the three sections that follow.

4.3.2.1 Dynamixel Servo Position Feedback

With servo communication being managed by the `hexapod_controller` node, servo position feedback would be required to be established through this node. Originally, the node only reads servo positions before making the hexapod stand up, in order to determine the current leg positions so they can be brought to the correct initial positions (as they may have been manually moved since the robot was last made to sit down, for example if the robot was lifted up and carried to a different location). During walking, however, the current servo positions at any given iteration of the locomotion loop are assumed to be equal to the position commands sent out in the previous loop iteration. Thus, rather than feeding back the actual positions and controlling the legs in a closed-loop manner, the gait steps are simply executed in an open-loop sequence. Furthermore, when the actual servo positions are read in the beginning, they are requested sequentially over the serial port, one servo at a time, resulting in a relatively slow read process when querying eighteen servos for their positions.

While not originally implemented in `hexapod_ros`, the USB2AX device does offer a synchronized reading feature, referred to as "SYNC_READ" [87]. This function can be used to read data from multiple Dynamixel servos at once, with one command, resulting in a significant boost in performance when reading the same data values from a large number of devices. The USB2AX controller achieves this by converting SYNC_READ commands into multiple, separate read commands to retrieve the necessary data from each servo, then packing this data into a single packet which is sent back to the computer. As a result, the effect of USB latency is decreased substantially.

Therefore, the `ServoDriver` class of the `hexapod_controller` package was modified to include a ROS Timer object [88], which is set to execute its callback function every 0.01 s, in which a

`readServoPositions` function is called. The `readServoPositions` function was written to set up the SYNC_READ instruction packet to read the "Present Position" RAM registers from the servos. Once the result is received from the serial port, the individual servo position values are extracted from the status, or return, packet and published as ROS messages (by calling another custom function called `publishServoPositions`), for use by other nodes.

It should be noted that the locomotion engine was not updated to use the actual position feedback for controlling the leg motions, as it was not originally coded to operate this way. Therefore, considering how the gait engine was originally designed, simply feeding it the actual servo positions would not even be very useful. In fact, attempting to use this feedback without making fundamental changes to the motion control algorithm would result in undesirable behaviour.

In addition to modifying the locomotion code to execute timed SYNC_READs of the servo positions, it was required to update the "Return Delay Time" EEPROM register of each servo, to ensure that the data can be read fast enough from all the servos. This setting defines the delay time taken from the transmission of an instruction packet until the return of a status packet [78]. It is represented by a 1-byte integer which can take on values from 0 to 254, with each data value corresponding to a 2 µs delay time. The default value is 250, resulting in a 0.5 ms delay. If a SYNC_READ is attempted without changing this value, each servo has a 0.5 ms delay before responding. In addition to blocking the serial port and affecting the robot's motion, this results in the SYNC_READ command timing out and no servo position readings being received.

Therefore, to minimize the delay and ensure that the servo positions could be read at a frequency of 100 Hz, without affecting locomotion, the "Return Delay Time" registers were set to a value of 0 for all eighteen Dynamixel actuators. While this should, in theory, result in no delay at all, Kevin Ochs mentions (in a comment in the source code of the `ServoDriver` class [17]) that a value of 0 actually results in a 1 µs delay time. However, this is still incredibly small and it was found that, using this setting, the servo positions can be consistently read without negatively affecting the PhantomX's locomotion.

An important point to be noted is that the use of the USB2AX, with the above software modifications, has allowed for servo positions to be fed back at a much faster rate and a considerably higher resolution than when the ArbotiX-M Robocontroller was used. Specifically, the feedback rate has been increased to 100 Hz, from just 20 Hz, while the full position measurement range of 0 – 1023 counts is now usable, rather than only values in the range 0 – 255.

4.3.2.2 MicroStrain® Inertial Measurement Unit Feedback

The advantages of using ROS really start to shine through when looking at adding functionality to a project. If the MicroStrain® 3DM-GX3® -25 IMU were to be used with another software framework, a custom program would likely need to be written to establish serial communication with the sensor, extract the necessary information and pass it onto the relevant control software. With ROS, however, a driver is available for IMUs compatible with the MicroStrain® 3DM-GX2® and 3DM-GX3® communication protocols.

This ROS driver is distributed as the `microstrain_3dmgx2_imu` package, which includes a standalone driver and a ROS node [89], as well as a C++ application programming interface (API) for advanced users who require lower-level access to the IMU [89]. The ROS node, `imu_node`, publishes IMU data, including acceleration, angular rates, and orientation (in the form of a quaternion), over a ROS topic, with additional topics also publishing diagnostic information and calibration status. ROS services for IMU self-testing and gyroscope bias calibration are also provided, while a variety of IMU settings can be adjusted via ROS parameters.

For this study, the `microstrain_3dmgx2_imu` package was installed on the Ubuntu system and used to obtain orientation estimates from the IMU's sensor fusion algorithm at the default driver rate of 100 Hz.

4.3.2.3 OptoForce Foot Force Sensors Feedback

The OptoForce Data Visualization (ODV) software used for testing the force sensors provides a great interface for visualizing and recording the force data, but offers no functionality for continuously streaming that data to other programs for "live" use. Furthermore, while a variety of communication interfaces and operating systems are supported by OptoForce for their 3D force sensors, the only official ROS driver made available by them makes use of Ethernet communication with the DAQ. Since it was preferred to utilize the USB interface, an alternative was required.

Fortunately, with ROS becoming so prevalent in the robotics industry, a number of third-party ROS drivers are available for OptoForce sensors. One of the most prominent of these is the `optoforce` package created by the Shadow Robot Company, which is made available by them on GitHub [90].

The driver is coded in Python, using the rospy client library for ROS integration, and communicates with a variety of OptoForce sensors through USB by means of the "pySerial" library. YAML parameter files are used to define the sensitivity values of the particular sensors in use, and the resulting force measurements are published on individual ROS topics for each sensor. Other parameters, such as sampling and DAQ filter frequencies, among others, can also be specified when launching the `optoforce` node.

For the PhantomX, two separate `optoforce` nodes are launched, one for each of the two DAQs, with the sampling and filter frequencies set to 1000 Hz and 15 Hz, respectively. Reasons for using this sampling frequency will be elaborated in Chapter 6.

4.4 Other Hardware Required for ROS Integration

In Section 4.1.1 it was mentioned that updates were made to the hardware system of the PhantomX in this study. The main reasons for this were related to the inadequacies of the Arbotix-M Robocontroller and the use of ROS in this study, as discussed in the previous section. The primary components incorporated into the hexapod platform are described below. In addition to these components, the Arbotix Commander was replaced with a PS3 controller to provide user inputs to the locomotion engine.

4.4.1 USB2AX Servo Communication Interface

As discussed above, servo communication is now established through USB using the USB2AX device. The USB2AX has an extremely small form factor (16 mm × 36 mm) and plugs directly into a USB port, connecting to the servo chain with a standard, 3-pin Dynamixel connector [86]. It can communicate with a range of Dynamixel servos using the half-duplex, transistor-transistor logic (TTL) [91] interface, at baud rates of up to 1 Mbps [86]. Other features offered by the USB2AX include surge protection and compatibility with an assortment of operating systems and programming languages. Figure 4.11 shows the USB2AX device, with the major components of interest labelled.

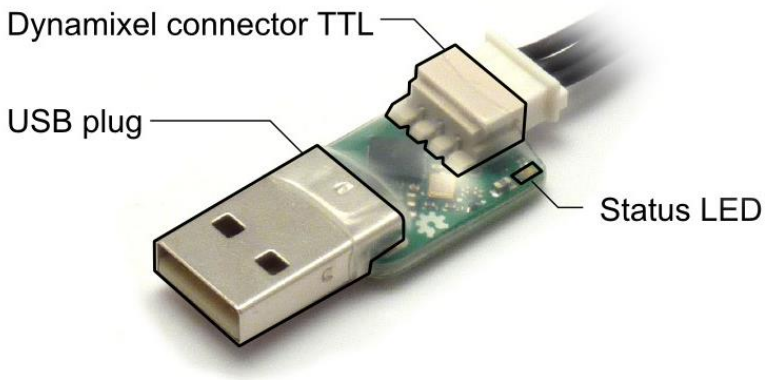


Figure 4.11: USB2AX – Dynamixel servo communication interface for USB [86]

4.4.2 AX/MX Power Hub

On the original PhantomX platform, both servo communication and power distribution were handled by the ArbotiX-M, which is no longer being used. Servo communication has been accounted for by means of the USB2AX, but an additional electronic component is required to power the Dynamixel actuators. For this purpose, the 6 Port AX/MX Power Hub [92], displayed in Figure 4.12, was purchased for the PhantomX. The AX/MX Power Hub allows for multiple Dynamixel servos, using the 3-pin TTL connector, to be connected and powered up. Power can be supplied to the board via the screw

terminal block or the DC barrel jack, allowing for an array of power supplies or batteries to be used. On the PhantomX, the onboard battery is connected to the AX/MX Power Hub through the barrel jack connector, using the wiring harness originally supplied with the robot.

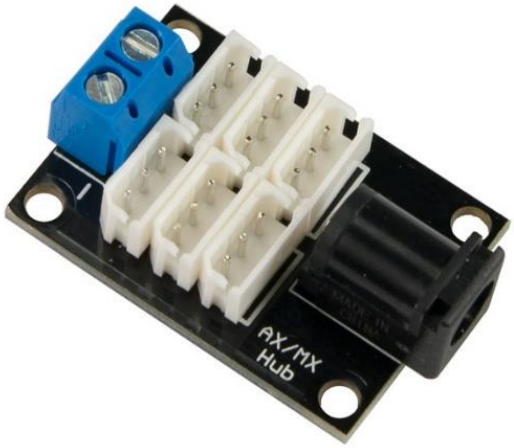


Figure 4.12: 6 Port AX/MX Power Hub used to power Dynamixel servos [92]

One port of the hub is connected to the USB2AX controller with a 3-pin Dynamixel cable, leaving the other five ports open for connecting servos. This, however, does not limit the setup to only include five actuators, since Dynamixel servos have the ability to be "daisy chained" on a single communication bus, with the power hub simply facilitating better management of the cables and wiring. A second 6 port hub (without power connectors), which was originally supplied with the PhantomX kit, is also utilized to split the wiring between the rear legs of the robot.

4.4.3 USB Hub

Along with the USB2AX, the IMU and two force sensor DAQs are also connected to the laptop computer through USB. Therefore, a Trascend® 4 port, USB 3.0 hub (model TS-HUB3K) [93] is mounted on the robot. This hub supports high speed data transfer and most operating systems, including Linux. While the hub can be used either powered or unpowered, it was found that the USB2AX and all sensors could be powered using just the laptop's USB port, so no power was required to be connected to the hub on board the robot. An image of the TS-HUB3K USB hub can be seen in Figure 4.13. A USB 3.0 extension cable connects the USB hub to the laptop's USB port, acting as a "tether" for the PhantomX robotic platform.

Figure 4.13: Transcend® 4 Port USB 3.0 Hub (TS-HUB3K) [93]

4.5 Fully Equipped Platform

An image of the fully equipped PhantomX hexapod robot platform, utilized in this study, is presented in Figure 4.14 below. Figure 4.15 also provides a closer view of the electronic equipment mounted on the robot's body, with the USB hub and USB2AX module clearly visible, as well as a voltage monitor for the onboard battery. Reflective, spherical markers can be seen mounted on the robot body in both of these figures. These are utilized for motion capture, discussed in more detail in Section 6.2.4.

Figure 4.14: Fully equipped PhantomX hexapod robot platform utilized in this study

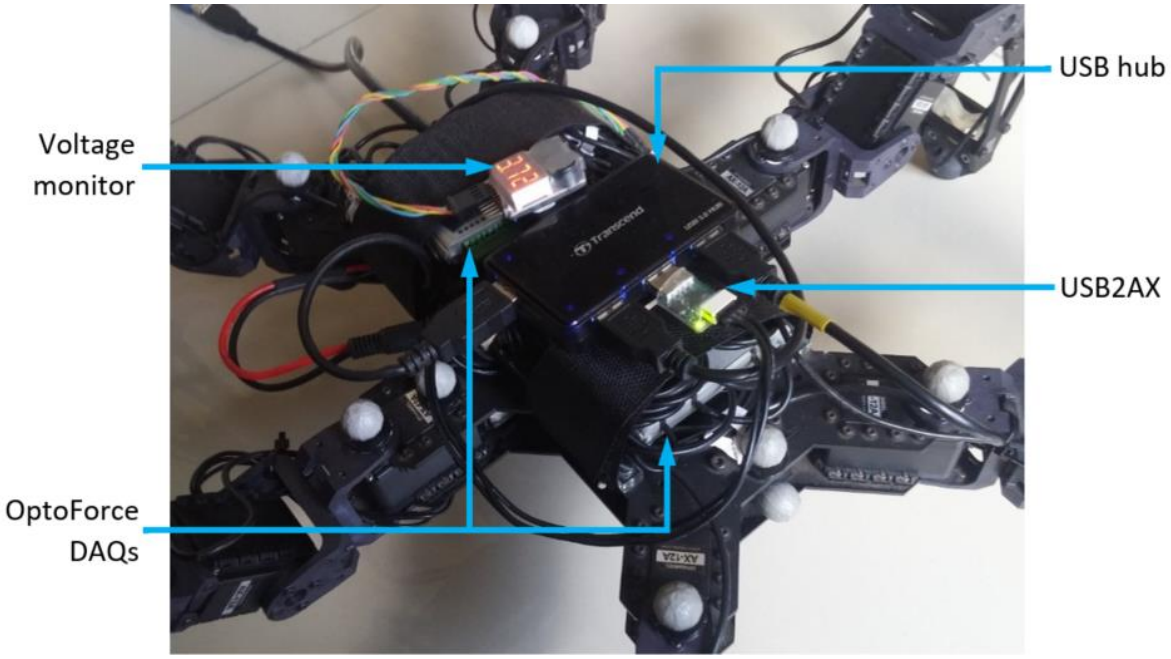


Figure 4.15: Electronic equipment mounted onboard the PhantomX platform

4.6 Conclusion

This chapter introduced the legged robotic platform utilized in this study, including the hexapod robot and a proprioceptive sensory system. The discussion on the sensory system was especially focussed on the new foot force sensors. Through experimental tests performed with the sensors, the force measurements provided by them were deemed to be sufficiently accurate for the application at hand. Newly designed feet for the PhantomX were also presented, which allow for the new force sensors to be mounted on the platform.

Furthermore, modifications and updates made to the platform to utilize ROS-based software for locomotion and control were also discussed. These included additional hardware components and appropriate sensor driver software, along with modifications to the ROS locomotion engine settings for this platform. The final platform provides a foundation for implementing and testing the control algorithms developed in this study, which are the subject of the chapters that follow.

Chapter 5: Preliminaries for Control

Before proceeding with the development of a posture control system for the hexapod, some preliminary analyses and derivations are necessary. In particular, the controllability of the system will be analysed in Section 5.1 below, after which Section 5.2 discusses the calculation of the robot's posture components, based on the data obtained from sensor feedback. Finally, Section 5.3 provides a brief overview of frequency domain concepts, related to the mathematical representation of linear systems, that are used for controller design and analysis in the next chapter.

5.1 Controllability Analysis

Prior to designing and implementing a control strategy for a dynamic system, it is instructive to analyse whether all the output modes of the system are capable of being affected by the inputs as required. For this reason, it is necessary to perform a controllability analysis on the system to be controlled.

It should be declared that a full, nonlinear controllability analysis of the entire hexapod robot is outside of the scope of this study. However, a linear controllability analysis of the simplified system to be used for posture control (comprised of the height, roll, and pitch posture components as states to be controlled, with foot contact forces acting as the inputs to the system) is still valid and provides useful insights into the conditions required for controllability. The following sections describe just such an analysis.

5.1.1 Simplified Dynamic Model of Hexapod

In Section 3.3, a virtual model of a hexapod robot body was introduced, for the purposes of posture control. However, given that the elements imparting forces on the body in this model are virtual, it is important to realize that, on the actual robot, the full posture control system will utilize contact forces at the feet to affect the desired motion. Accordingly, to analyse controllability it is necessary to model

the robot body in space with the foot contact forces as inputs, while the motions of the posture components are the modes to be controlled.

Figure 5.1 below illustrates a simplified model of the hexapod body in free space, acted upon by gravity and the foot contact forces from the ground. A world-fixed coordinate frame is also indicated in the top left of the figure. Since only the vertical translation, as well as roll and pitch rotations, are to be controlled, only the z-components of the contact forces will be assumed to act as inputs, for simplicity (a common assumption used in literature, e.g. [10], [13]). In fact, even though there are also x- and y-components of force which arise from friction at the foot while walking, these are used to attain locomotion and should not be tampered with for posture control. Adjusting the x- and y-positions of the feet to control posture would inadvertently result in undesired x- and y-direction (and yawing) motions of the robot!

The robot's legs are indicated in Figure 5.1 (by dashed lines) to show the relationship between the body and the contact force positions (also indicated in the figure as labelled dots at the ends of the legs). However, it should be noted that the dynamic model does not account for these as separate bodies, considering only the main body of the robot as a rigid body, with the COM, G, acted upon by the contact forces.

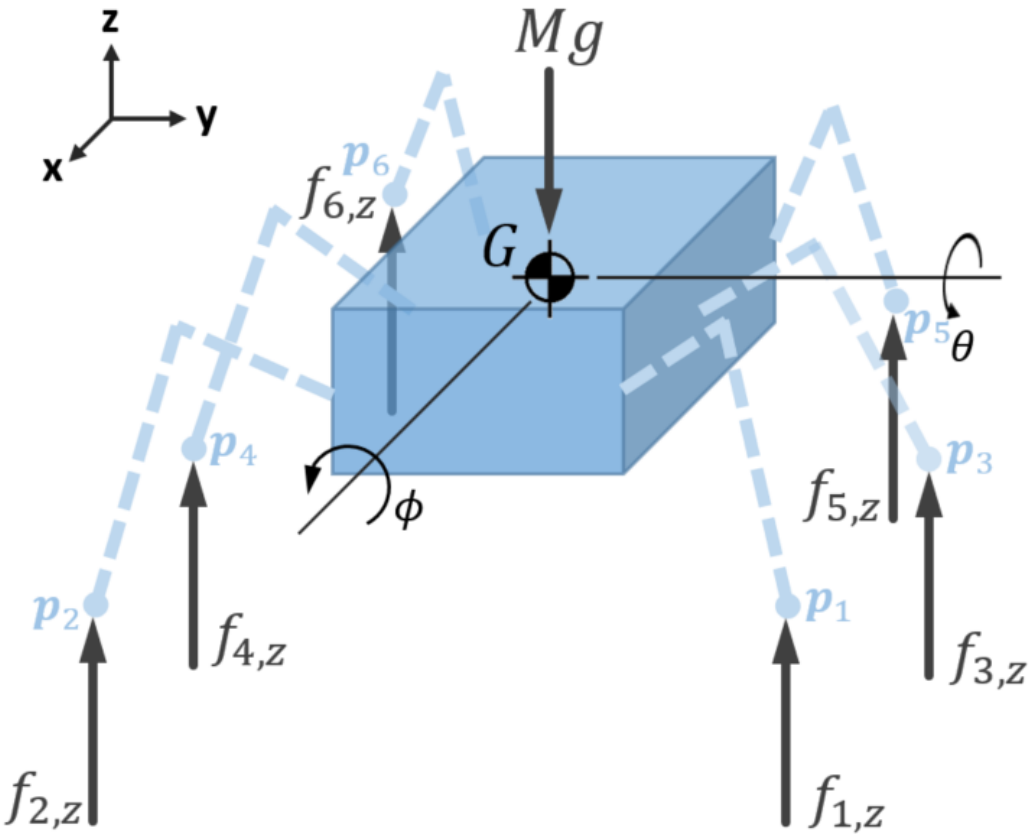


Figure 5.1: Simplified model of hexapod robot body, acted on by gravitational force and foot contact forces (a world-fixed coordinate frame is indicated in the top left)

Summing the forces in the z-direction (ΣF_z) and the moments about the x- and y-axes (ΣM_x and ΣM_y), for n arbitrary feet in contact with the ground ($1 \leq n \leq N$, where N is the number of legs on the robot), results in the following equations of motion describing the simplified body model:

$$\begin{aligned} M\ddot{p}_{G,z} = \Sigma F_z &= \sum_{i=1}^{n} f_{i,z} - Mg \\ I_\phi \ddot{\phi} &= \Sigma M_x = \sum_{i=1}^{n} -(p_{G,y} - p_{i,y}) f_{i,z} \\ I_\theta \ddot{\theta} &= \Sigma M_y = \sum_{i=1}^{n} (p_{G,x} - p_{i,x}) f_{i,z} \ , \end{aligned} \tag{5.1}$$

where $M \equiv$ Mass of the robot,

$I_\phi \equiv$ Moment of inertia about the body's local x-axis (as in Figure 3.14),

$I_\theta \equiv$ Moment of inertia about the body's local y-axis (as in Figure 3.14),

$\boldsymbol{p}_G \equiv$ Position vector of G (relative to the world frame indicated in Figure 5.1),

$\phi \equiv$ Angular displacement about the body's local x-axis (as in Figure 3.14),

$\theta \equiv$ Angular displacement about the body's local y-axis (as in Figure 3.14),

$f_{i,z} \equiv$ z-component of contact force at foot i,

$\boldsymbol{p}_i \equiv$ Position vector of foot i (relative to the world frame indicated in Figure 5.1),

$g \equiv$ Gravitational acceleration (9.81 m/s^2),

and as is commonly used throughout this dissertation, single and double overdots denote the first and second time derivatives, respectively, of the symbols they modify.

In order to analyse controllability, it is necessary to write the dynamic equations in the standard state-space form [94],

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{5.2}$$

where $\boldsymbol{x}$ represents the state vector, $\dot{\boldsymbol{x}}$ indicates the first time derivative of $\boldsymbol{x}$, $\boldsymbol{u}$ represents the control input vector, and $\boldsymbol{A}$ and $\boldsymbol{B}$ are the coefficient matrices of $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively.

Then, the algebraic controllability theorem states that the time-invariant system, as described by Equation (5.2), is controllable if and only if the rank, $r(\boldsymbol{Q})$, of the controllability test matrix,

$$\boldsymbol{Q} = [\boldsymbol{B} \quad \boldsymbol{AB} \quad \cdots \quad \boldsymbol{A}^{k-1}\boldsymbol{B}], \tag{5.3}$$

is equal to the order of the system, k [94].

Clearly, the matrix Equation (5.2) represents a system of first order differential equations, while Equations (3.28) are all second order differential equations. Therefore, by introducing the following relations,

$$\begin{aligned} \ddot{p}_{G,z} &= \dot{p}_{G,z} \\ \ddot{\phi} &= \dot{\phi} \\ \ddot{\theta} &= \dot{\theta}, \end{aligned} \tag{5.4}$$

and defining the state vector as

$$\boldsymbol{x} = \begin{bmatrix} p_{G,z} & \phi & \theta & \dot{p}_{G,z} & \dot{\phi} & \dot{\theta} \end{bmatrix}^T, \tag{5.5}$$

with corresponding derivative vector

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{p}_{G,z} & \dot{\phi} & \dot{\theta} & \ddot{p}_{G,z} & \ddot{\phi} & \ddot{\theta} \end{bmatrix}^T, \tag{5.6}$$

the dynamic equations can be put into the standard, state-space matrix form:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ \frac{1}{M} & \frac{1}{M} & \cdots & \frac{1}{M} \\ -\frac{(p_{G,y} - p_{1,y})}{I_\phi} & -\frac{(p_{G,y} - p_{2,y})}{I_\phi} & \cdots & -\frac{(p_{G,y} - p_{n,y})}{I_\phi} \\ \frac{(p_{G,x} - p_{1,x})}{I_\theta} & \frac{(p_{G,x} - p_{2,x})}{I_\theta} & \cdots & \frac{(p_{G,x} - p_{n,x})}{I_\theta} \end{bmatrix} \begin{bmatrix} f_{1,z} \\ f_{2,z} \\ \vdots \\ f_{n,z} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \\ 0 \\ 0 \end{bmatrix}, \tag{5.7}$$

where $\boldsymbol{I}$ is the 3 × 3 identity matrix, $\boldsymbol{0}$ is the 3 × 3 zero matrix,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \tag{5.8}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ \frac{1}{M} & \frac{1}{M} & \cdots & \frac{1}{M} \\ -\frac{(p_{G,y} - p_{1,y})}{I_\phi} & -\frac{(p_{G,y} - p_{2,y})}{I_\phi} & \cdots & -\frac{(p_{G,y} - p_{n,y})}{I_\phi} \\ \frac{(p_{G,x} - p_{1,x})}{I_\theta} & \frac{(p_{G,x} - p_{2,x})}{I_\theta} & \cdots & \frac{(p_{G,x} - p_{n,x})}{I_\theta} \end{bmatrix} \tag{5.9}$$

$$\boldsymbol{u} = \begin{bmatrix} f_{1,z} \\ f_{2,z} \\ \vdots \\ f_{n,z} \end{bmatrix}, \tag{5.10}$$

and $[0\ 0\ 0\ -g\ 0\ 0]^T$ represents an exogenous input.

5.1.2 A Note on Using Algebraic Controllability Theorem

In order to make use of the algebraic controllability theorem, it is necessary that the system being analysed is both linear and time-invariant [94]. While the system described above is linear, it should be noted that the number of contact legs, as well as the foot positions, do vary with time as the robot walks. However, the algebraic controllability theorem may still be able to be used to gain the insights required of this analysis.

To aid this idea, it is useful to consider the following conservative statements about the controllability of nonlinear systems made by Blösch [11]:

- If the controllability matrix, evaluated for a system linearized about some state $\boldsymbol{x}^*$, has full rank, then the system can be said to be locally controllable at $\boldsymbol{x}^*$.
- If a system is locally controllable for every $\boldsymbol{x}^*$, then the system can be deemed globally controllable.

A similar conjecture is made in this study about time-varying systems as follows:

> *If the controllability matrix of a time-variant system has full rank for every time instant, t^*, the system can be considered controllable at all times.*

This is, by no means, stated as a formal theorem, and will thus not be supported by a proof. Although, the philosophy will be carried forward in order to utilize the algebraic controllability theorem, to determine the conditions which are required for the inputs to affect the state as desired, as well as assess what situations may lead to a loss of controllability.

5.1.3 Algebraic Controllability Analysis at an Arbitrary Time Instant

Now, considering that the system described in Equation (5.7) is of order $k = 6$, the controllability matrix is calculated as

$$\boldsymbol{Q} = [\boldsymbol{B} \;\; \boldsymbol{AB} \;\; \boldsymbol{A}^2\boldsymbol{B} \;\; \boldsymbol{A}^3\boldsymbol{B} \;\; \boldsymbol{A}^4\boldsymbol{B} \;\; \boldsymbol{A}^5\boldsymbol{B}]. \tag{5.11}$$

It should be noted that the exogenous input in Equation (5.7) does not affect the controllability matrix, and will be compensated for directly using a feedforward term in the controller (see Chapter 6).

Looking at $\boldsymbol{A}$ in Equation (5.8), the following holds true:

$$\begin{aligned} \boldsymbol{A}^2 &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}\begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} = \boldsymbol{0}_{6\times 6} \\ \therefore \boldsymbol{A}^m &= \boldsymbol{0}_{6\times 6}, \qquad \forall \; m \geq 2. \end{aligned} \tag{5.12}$$

As a result,

$$\boldsymbol{A}^m\boldsymbol{B} = \boldsymbol{0}_{6\times n}, \qquad \forall \;\; m \geq 2. \tag{5.13}$$

Hence, the last four terms in Equation (5.11) all evaluate to zero matrices, which leaves $\boldsymbol{AB}$. At a particular time instant, t^*, with n legs in contact with the ground, the multiplication of $\boldsymbol{A}$ and $\boldsymbol{B}$ is evaluated as follows:

$$\begin{aligned} \boldsymbol{AB} &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ \frac{1}{M} & \frac{1}{M} & \cdots \;\; \frac{1}{M} \\ -\frac{(p_{G,y} - p_{1,y})}{I_\phi} & -\frac{(p_{G,y} - p_{2,y})}{I_\phi} & \cdots \;\; -\frac{(p_{G,y} - p_{n,y})}{I_\phi} \\ \frac{(p_{G,x} - p_{1,x})}{I_\theta} & \frac{(p_{G,x} - p_{2,x})}{I_\theta} & \cdots \;\; \frac{(p_{G,x} - p_{n,x})}{I_\theta} \end{bmatrix} \\ \therefore \boldsymbol{AB} &= \begin{bmatrix} \frac{1}{M} & \frac{1}{M} & \cdots & \frac{1}{M} \\ -\frac{(p_{G,y} - p_{1,y})}{I_\phi} & -\frac{(p_{G,y} - p_{2,y})}{I_\phi} & \cdots & -\frac{(p_{G,y} - p_{n,y})}{I_\phi} \\ \frac{(p_{G,x} - p_{1,x})}{I_\theta} & \frac{(p_{G,x} - p_{2,x})}{I_\theta} & \cdots & \frac{(p_{G,x} - p_{n,x})}{I_\theta} \\ 0 & \cdots & & 0 \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & & 0 \end{bmatrix}. \end{aligned} \tag{5.14}$$

Thus, the rank of $\boldsymbol{Q}$ is comprised of the ranks of both $\boldsymbol{B}$ and $\boldsymbol{AB}$, such that

$$r(\boldsymbol{Q}) = r(\boldsymbol{B}) + r(\boldsymbol{AB}). \tag{5.15}$$

Comparing Equations (5.9) and (5.14), it is apparent that

$$r(\boldsymbol{B}) = r(\boldsymbol{AB}). \tag{5.16}$$

Substituting Equation (5.16) into Equation (5.15) leads to

$$r(\boldsymbol{Q}) = 2r(\boldsymbol{B}). \tag{5.17}$$

Considering that $k = 6$, for full controllability it is required that $r(\boldsymbol{Q}) = 6$. From Equation (5.17), this is equivalent, in this case, to requiring that $r(\boldsymbol{B}) = 3$. Looking at $\boldsymbol{B}$ in Equation (5.9), it is clear that it can only have at most three linearly independent rows, regardless of the number of columns, n, but that the rank would be limited by the number of columns if $n < 3$, i.e.

$$\begin{aligned} r(\boldsymbol{B}) &\leq 3, \qquad n \geq 3 \\ r(\boldsymbol{B}) &< 3, \qquad n < 3. \end{aligned} \tag{5.18}$$

This leads to the first, and possibly most, important observation from the controllability analysis*: the hexapod is required to have at least three feet in contact with the ground to ensure full controllability of the three posture components (height, roll, and pitch), when foot contact forces are used as control inputs*.

In fact, since the analysis has been based on the number of contact legs, making no assumption about the total number of legs on the robot, it is equally valid for robots with more or less than six legs. The statement above can thus be generalized to any N-legged, statically stable robot, for which the three posture components examined here are to be controlled through foot force control inputs.

While the above condition is necessary for static controllability, it is certainly not sufficient, as even with three or more contact legs, singular configurations may arise. Inspection of $\boldsymbol{B}$ makes it evident that if all the feet were to lie directly beneath the COM (if physically possible), the numerators in the terms appearing in the last two rows of $\boldsymbol{B}$ would all evaluate to 0, resulting in a loss of row rank. In fact, if some, or all, the feet were to merely lie on the same point (again, if possible), not necessarily beneath the COM, then multiple columns of $\boldsymbol{B}$ would become equivalent, which would result in a loss of column rank.

Further scrutiny of the elements of $\boldsymbol{B}$ reveals that if all the contact legs were to be collinear, or lie along the same straight line, the last two rows of $\boldsymbol{B}$ would become linear combinations of each other, resulting in a loss of rank. The physical reason for the loss of controllability in this situation is that the robot's base of support is reduced to a line. Consequently, the contact forces are able to impart moments not about this line, but only about an axis perpendicular to it, resulting in the loss of one rotational degree of freedom in the control.

Therefore, a more general condition for posture controllability, at any point in time, can be stated:

> *For the posture of an N-legged robot (as described by its body height, roll, and pitch) to be fully controllable with statically stable motion, using foot contact forces as control inputs, at least three, unique, non-collinear ground contact points are required*.

5.1.4 A Note on Controllability with Dynamic Motion

In addition to the above singular configurations, another peculiar situation is prudent to contemplate. If the contact feet are positioned such that the vertical projection of the COM lies outside of the base of support, a situation of instability occurs. Theoretically, however, this does not show up as a loss of controllability in the form of reduced rank of the controllability matrix. The reason for this is that, from a mathematical perspective, the posture could still be controlled by applying negative vertical forces at certain contact points. Of course, in practice this is not possible with the robot as forces can only be imparted by pushing the legs against the terrain, and so situations such as these are not practically controllable, at least with leg configurations such as that on the PhantomX used in this study.

However, it is known (and some examples were provided in Chapter 2) that walking robots with less than three legs can achieve locomotion without falling over, despite the lack of a base of support, and that robots with more legs can also recover from situations of instability. Not surprisingly, this is attributed to the use of dynamically stable locomotion and dynamic manoeuvres, in which the contact leg positions are continually readjusted to attain the desired motion without allowing the robot to fall over.

While dynamic manoeuvres do prevent total instability, it is important to keep in mind that this only results in a stabilizable system. In other words, the modes of a dynamically stable robot are still not all controllable. This fact is made vividly clear when considering that a robot executing a dynamic gait or motion (for example, hopping) cannot achieve or maintain a particular, desired posture, due to the very nature of its continuously dynamic motion.

5.1.5 Concluding Remark

The simplified controllability analysis presented in this section has revealed conditions which would result in a loss of posture controllability for a multi-legged robot. While it would be preferable for these situations to be avoided completely during locomotion, this is not always possible, especially when utilizing a purely proprioceptive sensory system while walking on uneven terrain, as will be exposed in Chapter 7.

5.2 Posture Calculation

In Section 4.3, mention was made of the work done by Lubbe [14] on a state estimation algorithm for the PhantomX hexapod. It would have been ideal to utilize this state estimation to determine the robot's posture for the task of control in the present study. However, that particular implementation was coded in MATLAB to run offline, as a post processing step on sensor data recorded from the robot during motion, with the idea being simply to test and validate the algorithm. Of course, this would not be useful for the task of control, as state estimates are required in real-time, or "on-the-fly," in order to be able to compensate for deviations in the posture. While some work was done, at a later stage, by the original author, on porting the state estimation code over to C++ for an online, real-time implementation, this was never fully brought to fruition or tested.

Nevertheless, the current study only deals with controlling the posture and is thus only focussed on the height of the body above the ground, along with its roll and pitch angles. Thus, simpler methods of calculating the posture can be employed, without warranting an investigation into implementing a full state estimation algorithm to run online. These posture calculations will be elaborated in more detail in the following sections.

5.2.1 Body Coordinate Frame

In this study, the coordinate frame used for controlling the robot's posture is fixed to the main body and rotates with it. This body coordinate frame is denoted as $\{\boldsymbol{B}\}$ and is located such that its origin is coincident with that of the IMU's internal coordinate frame, having its z-axis aligned with the centre of the body's horizontal plane [19] (specific details regarding the mounting of the IMU on the body can be found at [14]). A schematic diagram of the PhantomX hexapod is shown in Figure 5.2, where the body coordinate frame can be seen in the described location, with the x-axis pointing forward and z-axis pointing upward. It should be noted that while the origins of $\{\boldsymbol{B}\}$ and the IMU frame coincide, the directions of the various axes do not, as explained in more detail in Section 5.2.3 below. Figure 5.2 also indicates the leg (and foot) numbering convention that will be used throughout this dissertation.

Furthermore, since the z-axis points upward, the y-axis points to the left of the robot, resulting in positive pitch angles corresponding to pitching the front, or "nose," of the robot downward. This is contrary to the case of aircrafts, in which it is customary to make use of a downward-pointing z-axis so that positive pitch angles (using the ZYX Euler angle sequence) correspond to pitching the vehicle's nose upward [65].

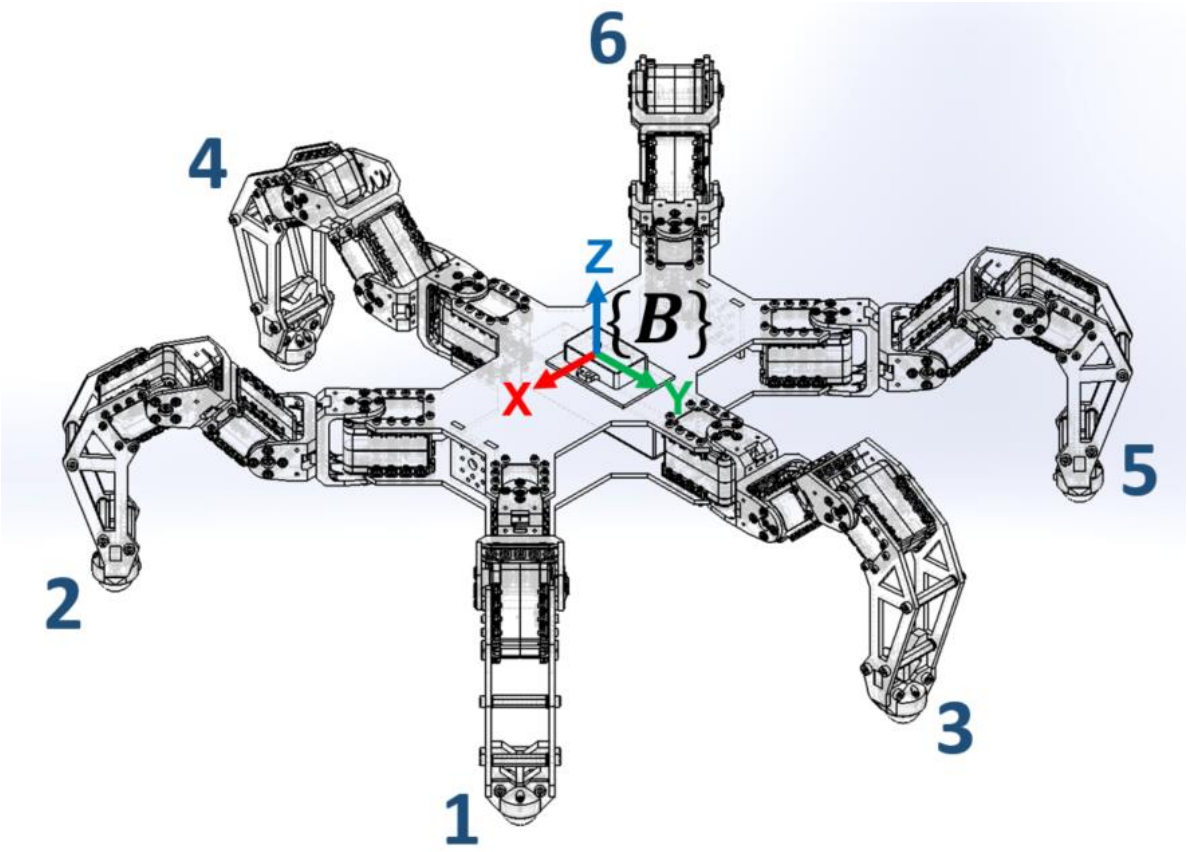


Figure 5.2: Schematic of PhantomX hexapod with body coordinate frame, $\{\boldsymbol{B}\}$, and leg numbers [19]

With regards to this body coordinate frame, the posture of the robot is defined as the z-position of $\{\boldsymbol{B}\}$ above the ground, i.e. height, as well as the orientation of its xy-plane relative to the horizontal plane of the Earth (perpendicular to the gravity vector), i.e. roll and pitch angles about the x- and y-axes, respectively [19]. The sections that follow describe the calculations of these specific quantities, based on the data obtained from the robot's sensory system.

5.2.2 Height Calculation

Estimating the height of the robot above the terrain on which it stands is not as straightforward as it may initially seem. This is especially true when the terrain beneath the robot is not flat and level, as there is no single point which can be used as a definite reference against which to measure the height. In fact, even if there was a well-defined reference point, it may not be possible to measure the height relative to this point without exteroceptive sensing equipment.

For these reasons, and considering that the use of proprioceptive sensing is generally preferable for posture measurements (as discussed in Section 2.3.5.2), it is common to use the individual positions of the feet to estimate the robot's height. Some studies in the literature, covered in Chapter 2, employed this type of approach, either by calculating an average of the foot heights, e.g. [4], or by using a least squares fit of the individual heights, to obtain an equivalent "ground plane" off which

measurements can be made, e.g. [9]. While fitting a plane can be useful for estimating the ground slope and utilizing that information in higher levels of planning and control, it is not really necessary for the posture calculation required in this study. Therefore, the average of the individual heights of the n feet in contact with the ground is used as a measure of the robot height, h:

$$h = -\frac{\sum_{i=1}^{n} p'_{i,z}}{n}, \qquad 1 \leq n \leq N, \tag{5.19}$$

with $N = 6$ for a hexapod robot. It should also be noted that the range of n above assumes at least one foot in contact with the ground, since without any contact feet there would be no way of proprioceptively estimating the height of a legged robot.

In Equation (5.19), $p'_{i,z}$ refers to the z-position of the ith foot, as calculated in $\{\boldsymbol{B}\}$. The negative of the mean value of these vertical positions is taken in the above equation because the z-axis of $\{\boldsymbol{B}\}$ points upward, resulting in negative z-position values when the feet are below the body.

Of course, in order to actually use the above equation to estimate the height, it is necessary to calculate the foot positions. Fortunately, the functions written to implement the PhantomX's kinematic model in the C++ port of the state estimation code (discussed above) can be used independently of the state estimation algorithm. Thus, the individual foot positions can be calculated by feeding the servo position measurements into the robot's forward kinematic model, the detailed derivation of which can be found at [67].

Initially, the kinematic model mapped the servo position values to joint position displacements based on a servo position value range of 0 – 255. Since then, the servo communication hardware has been upgraded for this study, allowing for the full measurement range of 0 – 1023 counts to be retrieved from the actuators (as discussed in Section 4.3.2.1). Therefore, the kinematic model was updated to accordingly make use of the full measurement range, allowing for increased measurement resolution and hence improved calculation accuracy. In addition, the link length and servo offset values for the tibia were updated to account for the new foot and force sensor, as was done for the locomotion engine (see Section 4.3.1).

In order to utilize this kinematic model with the control software, the C++ functions were wrapped into a ROS node. This was done in the main `posture_control` node which implements the VMC algorithm (discussed in more detail in Chapter 6). A subscriber was created to receive the servo position readings published by the `hexapod_controller` node (see Section 4.3.2.1). In this subscriber's callback function, the servo position measurement values are extracted from the received message and the kinematic model functions are called to calculate each foot position. Furthermore,

Equation (5.19) is employed in the abovementioned callback function to calculate the mean height of $\{\boldsymbol{B}\}$, which is then stored for use in the VMC controller, while also being published out so it can be viewed or recorded for later analysis.

5.2.3 Orientation Calculation

In addition to joint position and foot force sensors, the robot was equipped with an IMU, for the exact purpose of measuring orientation. The MicroStrain® 3DM-GX3® -25 was presented in Section 4.2.2, where it was stated that the IMU provides orientation estimates calculated by its onboard sensor fusion algorithm.

Therefore, in the `posture_control` node, a subscriber was created to receive IMU data messages, published by `imu_node` in the ROS MicroStrain® driver (see Section 4.3.2.2), and use the included quaternion to determine the robot's roll and pitch angles. However, extracting the robot's orientation was not as simple as converting the quaternion values, provided by the ROS driver, to Euler angles using the mapping functions presented in Equation (3.26) of Chapter 3. These complications arose from transformations performed by the ROS driver on the IMU rotation, as well as the use of different rotation matrix conventions in the IMU hardware and the ROS driver.

Accounting for the differences between the orientations provided the IMU and ROS driver, and letting the quaternion provided by the ROS driver be denoted by $\boldsymbol{q}$, as described in Equation (3.21), the Euler angles describing the rotation of the IMU can be computed as

$$\boldsymbol{u}_{IMU}(\boldsymbol{q}) = \begin{bmatrix} \phi_{IMU}(\boldsymbol{q}) \\ \theta_{IMU}(\boldsymbol{q}) \\ \psi_{IMU}(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} \text{atan2}(-2q_2q_3 - 2q_0q_1, -q_0^2 + q_1^2 + q_2^2 - q_3^2) \\ \text{asin}(2q_1q_3 - 2q_0q_2) \\ \text{atan2}(2q_1q_2 + 2q_0q_3, -q_0^2 - q_1^2 + q_2^2 + q_3^2) \end{bmatrix}. \tag{5.20}$$

Details of the transformations applied to derive this mapping can be found in Appendix D.

Although the necessary angles were computed slightly differently in the `posture_control` node, using the `tf`[4] package to simplify the coding, the mapping of Equation (5.20) is suggested as it adheres to a single convention (that of [65]), eliminating any confusion or ambiguity.

Another mapping was also required to be made to obtain the roll and pitch angles of the robot's body frame, $\{\boldsymbol{B}\}$, in view of the fact that the IMU is actually mounted on the robot with its y-axis pointing forward and x-axis to the left. Consequently, the x-axis of $\{\boldsymbol{B}\}$ aligns with the y-axis of the IMU and vice versa. Therefore, the roll angle, ϕ, of the robot about the x-axis and the pitch angle, θ, about the

[4] The ROS `tf` package allows users to keep track of multiple coordinate frames over time, offering them the relevant functions to transform points and vectors between coordinate frames. More information about this package can be found at [125].

y-axis can be obtained by simply swapping the roll and pitch angle values of the IMU calculated above. That is

$$\phi = \theta_{IMU} \tag{5.21}$$

$$\theta = \phi_{IMU}. \tag{5.22}$$

While this is sufficient for obtaining the roll and pitch of $\{\boldsymbol{B}\}$ for posture control, it is important that this swap is kept in mind, if the Euler angles are to be used to reconstruct the rotation matrix or quaternion using the ZYX rotation sequence. In such a case, the two angles must be swapped back before using the reverse mappings provided in [65][5].

Furthermore, it should be noted that the yaw angle of the IMU, ψ_{IMU}, does not correspond with the heading of the actual robot body. This is because yaw angle provides the heading in the x-direction of the frame it is calculated for. Since the roll angle is the last rotation in the sequence and is actually applied about the IMU's x-axis, it acts to "roll out" the IMU's y-axis (see Figure 3.11), which corresponds to the x-axis of $\{\boldsymbol{B}\}$. This additional yawing action is not accounted for in ψ_{IMU}.

Of course, this is of no consequence to the present study on posture control, as yaw is not used. However, in the event that the yaw of the robot is also required, the most straightforward way to determine all the Euler angles would be to use the ZXY rotation sequence mappings, based on the rotation matrix of the IMU[6]. The relevant mapping equations for the ZXY rotation sequence can be found at [65][7]. This mapping would yield the same ϕ and θ values as Equations (5.21) and (5.22), while the resulting yaw angle, ψ, would correspond to the actual heading of the robot. The reason this sequence was not used in this study is that it is preferable to maintain use of the standard convention of the field (mobile robotics) as far as possible, to avoid any further confusion.

5.2.4 Concluding Remark

This section presented the computations necessary to calculate the three posture components of the robot body using the available sensor data. The particular implementation of these computations, in ROS, was also briefly discussed. It should be noted that, while the posture calculations were incorporated into the `posture_control` node in this study for the sake of simplicity, in future it would likely be preferable to implement this functionality as a node on its own. Especially if the posture calculation becomes more complex, or if full state estimation is utilized, this task should be

[5] When referring to [65] for the necessary Euler angle mappings, the reader should note that the ZYX rotation sequence is denoted as the (1,2,3) Euler angle sequence in [65].
[6] See Appendix D for the computation of the IMU rotation matrix, denoted as $\boldsymbol{M}$, from the rotation provided by the ROS driver (specifically Equation (D.7)).
[7] The ZXY rotation sequence is referred to as Euler angle sequence (2,1,3) in [65].

separated from the control tasks, both for an improved, modular software architecture and to prevent either of these processes from holding the other up during computation. However, for this study the computation of posture is not very intensive and was found to not have a significant effect on the frequency of execution of the `posture_control` node (discussed in more detail in the next chapter).

5.3 Frequency Domain Representations of Linear Systems

This section provides a brief overview of frequency domain concepts used in Chapters 6 and 7. This is meant only to aid the discussions in the next chapters. For a more comprehensive coverage of these concepts, as well as many of the fundamental concepts related to control theory, the reader is referred to the informative textbook *Modern Control Systems* by Dorf and Bishop [95], from which most of the information presented below was obtained.

The mathematical description of a linear system can be converted from a time domain representation, to a frequency domain representation, by means of a Laplace transformation. The primary advantage of this conversion is that differential equations, which are not as easy to solve, are substituted by relatively easily solved algebraic equations.

Mathematically, the Laplace transform of a function of time, $f(t)$, is described as

$$\mathcal{L}\{f(t)\} = F(s) = \int_{0^-}^{\infty} f(t)e^{-st}dt\,, \tag{5.23}$$

where $F(s)$ is the frequency domain representation of $f(t)$, in the form of a function of the Laplace variable, s, and 0^- indicates that the integral should include any discontinuity at $t = 0$. A list of important Laplace transform pairs ($f(t)$ and the associated transform $F(s)$) are provided in [95].

Another important frequency domain concept is that of a transfer function, detailed information on which can be found at [95], if necessary. A useful example of a transfer function pertinent to this study is that of a first-order system, with time constant τ_c, given by [96]

$$H(s) = \frac{1}{1 + s\tau_c}. \tag{5.24}$$

The use of such a system as a filter will be discussed in the next chapter, while the relationship between τ_c and the filter frequency will be illuminated in Chapter 7.

5.4 Conclusion

This chapter presented preliminary analyses and derivations required for developing a posture control system for the hexapod robot. In particular, a simplified, linear controllability analysis was conducted for the posture of the hexapod robot at any given time, t^*. While this is not a substitute for a full,

nonlinear, and time-varying controllability analysis of a hexapod robot system, the exercise has been a fruitful one, as it yielded meaningful insights about the situations and configurations of the robot which would result in a loss of controllability. In particular, the main finding of the controllability analysis was that a general condition for the posture of an N-legged robot (as described by its body height, roll, and pitch) to be fully controllable, with foot contact forces as control inputs, is that at least three, unique, non-collinear ground contact points are required.

Furthermore, the procedures used for calculating the height and orientation of the robot body, for the purpose of posture control, were described. Methods for obtaining the necessary information from the sensor data, as well as computations carried out on these data, were discussed, while also touching on the particular implementations of these methods in ROS.

Additionally, a brief overview of frequency domain concepts used in this study was provided, along with an example of a transfer function relevant to this study. These concepts will used in the discussions on the control system design, in upcoming chapters.

Having fulfilled these prerequisites, the posture control system for the hexapod can be developed and tested. Hence, the chapters that follow present the detailed development and implementation of the posture control system designed in this study.

Chapter 6: Standing Posture Control

While controlling the posture of the robot is no trivial task, the entire undertaking is made substantially more complicated by the constant motion and continually changing kinematic configurations experienced by the robot during walking. Therefore, it was decided to first consider only the case of the robot standing on all six legs, without walking, to be able to establish the foundations of the posture control system, before having to also deal with the added complexities introduced by walking.

This chapter is dedicated to presenting the methods used in designing and implementing the "Standing Posture Controller," along with results of experimental tests carried out to assess this controller's performance on the PhantomX platform. The methods and results discussed in this chapter were published in [19] and presented at the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017), at Vancouver, Canada, in September 2017. This work demonstrated the implementation of a Standing Posture Control system, in the form of VMC, on a low-cost, commercially available hexapod robot (the PhantomX). A further contribution of this study was the implementation of a direct force control method to control foot forces, using low-cost actuators which can only be position controlled. To the best of the authors' knowledge, this combination of control techniques, implemented on a low-cost, commercially available platform, has not been previously tested.

6.1 Control Structure

It has already been stated that the Virtual Model Control strategy will be employed in this study. Therefore, this section presents the implemented control system, consisting of the higher level VMC controller and foot force distribution, as well as the lower level single leg force controller, used to achieve the contact forces commanded by the higher-level posture control elements.

6.1.1 A Note on Posture Calculation

In Section 5.2.2, the generalized formula for calculating the height of the robot body was provided (see Equation (5.19)). The reader will likely recall that this calculation relied on information regarding the number of legs in contact with the terrain. However, considering that, for the Standing Posture Controller implementation, the robot is assumed to be standing on all six legs at all times, all of these can be assumed to be contact legs. Hence, the height, h, is calculated as follows, for the specific case of Standing Posture Control:

$$h = -\frac{\sum_{i=1}^{N} p'_{i,z}}{N}, \tag{6.1}$$

where, once again, $N = 6$ for the hexapod.

6.1.2 Virtual Model Control Strategy

The basic premise behind VMC was explained in Section 3.3, along with presenting the equations of motion for a model of the hexapod body with virtual springs and dampers attached to it. These equations of motion are restated here to more easily facilitate the discussion which follows:

$$\begin{aligned} M\ddot{p}_{G,z} = \Sigma F_z &= -K_z \Delta p_{g,z} - C_z \Delta \dot{p}_{G,z} \\ I_\phi \ddot{\phi} &= \Sigma M_x = -K_\phi \Delta\phi - C_\phi \Delta\dot{\phi} \\ I_\theta \ddot{\theta} &= \Sigma M_y = -K_\theta \Delta\theta - C_\theta \Delta\dot{\theta}. \end{aligned} \tag{6.2}$$

6.1.2.1 Virtual Model Controller Design

For the virtual model controller to regulate body posture with the dynamics described by Equations (6.2), it needs to be set up to apply control actions, in the form of forces, equivalent to those applied by the virtual elements, i.e. the terms on the right-hand sides of the above equations.

Thus, letting the control outputs be defined by the vertical (z-direction) force, F_{VMC}, roll moment, M_ϕ, and pitch moment, M_θ, to be applied to the robot, then theoretically the control actions should be equivalent to

$$\begin{aligned} F_{VMC} &\equiv -K_z \Delta p_{G,z} - C_z \Delta \dot{p}_{G,z} \\ M_\phi &\equiv -K_\phi \Delta\phi - C_\phi \Delta\dot{\phi} \\ M_\theta &\equiv -K_\theta \Delta\theta - C_\theta \Delta\dot{\theta}, \end{aligned} \tag{6.3}$$

to achieve the desired dynamic response.

However, the virtual model was derived for displacement deviations from the static position, but the posture components on the actual robot are measured relative to different reference values. Therefore, the deviation values in the above model must be replaced by error values computed between the desired and actual positions.

That is,

$$\begin{aligned} -\Delta p_{G,z} &\equiv e_z = h_d - h \\ -\Delta\phi \quad &\equiv e_\phi = \phi_d - \phi \\ -\Delta\theta \quad &\equiv e_\theta = \theta_d - \theta, \end{aligned} \tag{6.4}$$

where e_z, e_ϕ, and e_θ are the height, roll, and pitch errors, respectively, while h_d, ϕ_d, and θ_d represent the height, roll, and pitch reference values, or setpoints, respectively. The quantities h, ϕ, and θ, are the posture components, or control variables, as defined in Section 5.2. It is important to note that, for the equivalence relations in Equations (6.4) to be applicable, it is necessary to make the assumption that the position of the robot's COM coincides with the origin of the body frame, $\{\boldsymbol{B}\}$. The reason for this is that the posture components are calculated with respect to $\{\boldsymbol{B}\}$, while $\Delta p_{G,z}$ represents the vertical deviation of the COM. This assumption will be further elaborated in Section 6.1.3 below.

It may be appealing to subsequently assign the velocity terms in the controller to the derivatives of these error values, given by

$$\begin{aligned} \dot{e}_z &= \dot{h}_d - \dot{h} \\ \dot{e}_\phi &= \dot{\phi}_d - \dot{\phi} \\ \dot{e}_\theta &= \dot{\theta}_d - \dot{\theta}, \end{aligned} \tag{6.5}$$

where, as usual, single overdots indicate the first time derivatives of the symbols they modify.

Taking into account the terms appearing on the right-hand sides of Equations (6.5), it should be realized that these derivatives should not naively be used as the velocity terms in the controller. More specifically, the derivatives of the reference values are subject to spiking, to extremely large magnitudes, when step changes are applied to the references, causing the phenomenon of "derivative kick" [97]. In order to avoid this, only the derivatives of the position values should be utilized. This is, in any case, more consistent with the behaviour of a physical damper, which applies retarding forces to an object at any non-zero velocity, as opposed to being based on an error from a certain reference velocity.

Furthermore, Åström [98] explains that differentiation is always sensitive to noise, especially high-frequency noise in a signal, which can be significantly amplified in the derivative of the signal. This is distinctly illustrated by the transfer function of a differentiator, $G(s) = s$, which grows infinitely with increasing values of s:

$$\lim_{s\to\infty} G(s) = \lim_{s\to\infty} s = \infty. \tag{6.6}$$

Therefore, in the practical implementation of a controller with derivative action, it is necessary to limit the high frequency gain of the derivative term, by filtering the signal. Traditionally, the derivative term, $D = D(s)$ (represented in the frequency domain), in a controller is computed by simply differentiating the input signal, $Y = Y(s)$, and multiplying it by the derivative gain, K_d. This is mathematically defined as

$$D = -sK_dY, \tag{6.7}$$

where the negative of the derivative is used so that the control action acts to dampen or retard motion.

In order to limit the high frequency gain, the derivative term can be implemented as [98]

$$D = -\frac{sK_d}{\left(1 + s\frac{K_d}{K_pN_f}\right)}Y = -\frac{sK_pT_d}{\left(1 + s\frac{T_d}{N_f}\right)}Y, \tag{6.8}$$

where K_p is the proportional gain of the controller, $T_d = K_d/K_p$ is defined as the derivative time constant, and N_f is a parameter related to the filter cut-off frequency.

The term in Equation (6.8) can be interpreted as the ideal derivative of Equation (6.7) filtered by a first-order system, $H(s)$, with the time constant $\tau_c = T_d/N_f$, the transfer function of which is given by [96]

$$H(s) = \frac{1}{1 + s\tau_c} = \frac{1}{1 + s\frac{T_d}{N_f}}. \tag{6.9}$$

This approximation (Equation (6.8)) acts as a derivative for low-frequency signal components, but the gain is limited to K_pN_f. As a result, high-frequency measurement noise in the signal is amplified, at most, by a factor K_pN_f [98], [99]. This can be demonstrated mathematically by individually considering the derivative action gains at the limits of low- and high-frequency signal components.

For low frequencies, $s \ll 1$; thus, considering that the ratio $K_d/(K_pN_f)$ would generally not be many orders of magnitude larger than 1, especially for practical controller gains, it follows that

$$\begin{aligned} s\frac{K_d}{K_pN_f} &\ll 1, \qquad s \ll 1 \\ \therefore 1 + s\frac{K_d}{K_pN_f} &\approx 1, \qquad s \ll 1. \end{aligned} \tag{6.10}$$

Subsequently,

$$\begin{aligned} \frac{sK_d}{\left(1+s\frac{K_d}{K_pN_f}\right)} &\approx \frac{sK_d}{1}, \qquad s \ll 1 \\ \therefore D = -\frac{sK_d}{\left(1+s\frac{K_d}{K_pN_f}\right)}Y &\approx -sK_dY, \qquad s \ll 1, \end{aligned} \tag{6.11}$$

showing that the derivative term of Equation (6.8) closely approximates the ideal derivative in Equation (6.7), for low-frequency signal components.

The high-frequency case can be assessed by computing the limit of the derivative gain when $s \to \infty$:

$$\begin{aligned} \frac{sK_d}{\left(1+s\frac{K_d}{K_pN_f}\right)} &= \frac{sK_d}{s\left(\frac{1}{s}+\frac{K_d}{K_pN_f}\right)} \\ &= \frac{K_d}{\left(\frac{1}{s}+\frac{K_d}{K_pN_f}\right)}. \end{aligned} \tag{6.12}$$

Hence,

$$\begin{aligned} \lim_{s\to\infty}\frac{sK_d}{\left(1+s\frac{K_d}{K_pN_f}\right)} &= \lim_{s\to\infty}\frac{K_d}{\left(\frac{1}{s}+\frac{K_d}{K_pN_f}\right)} \\ &= \frac{K_d}{K_d\frac{1}{K_pN_f}} \\ &= K_pN_f \\ \therefore \lim_{s\to\infty}\frac{sK_d}{\left(1+s\frac{K_d}{K_pN_f}\right)} &= K_pN_f, \end{aligned} \tag{6.13}$$

clearly demonstrating that high-frequency noise is amplified, in the derivative action, by a factor of no more than K_pN_f.

On account of the reasons provided above, the derivative terms in the virtual model controller are to be set to use the derivatives of filtered position signals, rather than simply the derivatives of the position values. Additionally, one more factor needs to be accounted for in implementing a controller to emulate the behaviour of the desired virtual model.

In the z-direction equation of motion of (6.2), the virtual spring is specified to support the robot's weight at the static position, thus resulting in the presence of only restorative forces for deviations from this static position (i.e. the terms in the equation arising from the robot's weight and the static

force of the spring "cancel" each other out). When emulating the virtual spring with a control system, it is important to remember to also add this static force, allowing the robot's weight to be supported when the height is kept at its reference value.

Taking all of the above into consideration, the three individual control equations become

$$\begin{aligned} F_{VMC} &= K_z e_z - C_z \dot{h}_f + f_s \\ M_\phi &= K_\phi e_\phi - C_\phi \dot{\phi}_f \\ M_\theta &= K_\theta e_\theta - C_\theta \dot{\theta}_f, \end{aligned} \tag{6.14}$$

where $\dot{h}_f$, $\dot{\phi}_f$, and $\dot{\theta}_f$ are the filtered derivatives of the measured height, roll, and pitch signals, respectively, and f_s is the static force employed to counteract the robot's weight.

Equations (6.14) can be written in matrix form as

$$\boldsymbol{F_{VMC}} = \boldsymbol{K e} - \boldsymbol{C \dot{e}} + \boldsymbol{F}_s, \tag{6.15}$$

where

$$\boldsymbol{F_{VMC}} = \begin{bmatrix} F_{VMC} \\ M_\phi \\ M_\theta \end{bmatrix} \tag{6.16}$$

is the virtual model controller output, and the position and velocity error states, $\boldsymbol{e}$ and $\dot{\boldsymbol{e}}$, respectively, are defined as

$$\boldsymbol{e} = \begin{bmatrix} e_z \\ e_\phi \\ e_\theta \end{bmatrix} \tag{6.17}$$

$$\dot{\boldsymbol{e}} = \begin{bmatrix} \dot{h}_f \\ \dot{\phi}_f \\ \dot{\theta}_f \end{bmatrix}. \tag{6.18}$$

The proportional and derivative gain matrices are denoted by $\boldsymbol{K}$ and $\boldsymbol{C}$, respectively, such that

$$\boldsymbol{K} = \begin{bmatrix} K_z & 0 & 0 \\ 0 & K_\phi & 0 \\ 0 & 0 & K_\theta \end{bmatrix} \tag{6.19}$$

$$\boldsymbol{C} = \begin{bmatrix} C_z & 0 & 0 \\ 0 & C_\phi & 0 \\ 0 & 0 & C_\theta \end{bmatrix}, \tag{6.20}$$

while

$$\boldsymbol{F}_S = \begin{bmatrix} f_S \\ 0 \\ 0 \end{bmatrix} \tag{6.21}$$

is a feedforward term in the controller representing the static force for gravity compensation.

As suggested in Section 3.3, it can be seen that controller gains, $\boldsymbol{K}$ and $\boldsymbol{C}$, are equivalent to the mechanical characteristics of the virtual elements chosen for the virtual model. The diagonal structures of the gain matrices explicitly indicate that the three control variables are decoupled (this decoupling effect is demonstrated in Section 6.3.1). Furthermore, from the structure of the control Equation (6.15) it can be recognized that the use of springs and dampers in the virtual model effectively equates to proportional-derivative, or PD, control, which is a simple and well known, linear control strategy[8].

Based on measurements of the contact forces when the robot is standing up, obtained using the OptoForce sensors, a static force value of $f_S = 20.9$ N was found to be appropriate to compensate for the weight of the robot. It should be noted that the static force is sensitive to changes in the configuration of the robot platform, such as, for instance, the addition or removal of equipment and hardware from the body. Even changing the onboard battery results in changes in weight, as the exact mass of the battery varies from one battery to another. However, it is preferable to design the controller to be fairly robust to this type of change in configuration, rather than having it rely on an extremely precise value of static force to operate satisfactorily. This is of particular significance when considering that during the employment of a legged robot in field-use, it would be expected that the robot be able to carry payloads or have onboard equipment changed as desired. Thus, the static force of $f_S = 20.9$ N was found to be a good overall value for general use on the robot, with small changes in its weight being compensated for by specifying a sufficiently "stiff" proportional gain in the z-direction.

Of course, the control equations above describe a continuous system. To be implemented digitally, it is necessary for them to be discretized. Since the control equations for the three posture components are decoupled, as described in Section 3.3, they can be implemented as three individual controllers. Hence, the discretization process for a generic PD control equation, representative of those in Equations (6.14), will be presented below.

[8] More information on PD control can be found at [95].

The continuous, generic control equation, in the time domain, takes the form

$$u(t) = P(t) + D(t) + F_{ff}(t), \tag{6.22}$$

where $u(t)$ is the control signal (or controller output), $P(t)$ and $D(t)$ are the proportional and derivative parts, respectively, and $F_{ff}(t)$ is a feedforward signal.

The proportional part of the controller acts on the error, $e(t)$, between the reference and measured values of the input signal, $r(t)$ and $y(t)$, respectively. Thus,

$$P(t) = K_p e(t) = K_p\big(r(t) - y(t)\big). \tag{6.23}$$

This term can be discretely implemented by simply replacing the continuous signals with their sampled equivalents. Thus, for a particular sampling instant, t_k,

$$P(t_k) = K_p\big(r(t_k) - y(t_k)\big). \tag{6.24}$$

On the other hand, the derivative part is not as straightforward. In the frequency domain, the derivative part of the controller is described by Equation (6.8), which can be rearranged as follows:

$$\begin{aligned} D(s) &= -\frac{sK_pT_d}{\left(1 + s\frac{T_d}{N_f}\right)}Y(s) \\ D(s)\left(1 + s\frac{T_d}{N_f}\right) &= -sK_pT_dY(s) \\ D(s) + sD(s)\frac{T_d}{N_f} &= -sK_pT_dY(s). \end{aligned} \tag{6.25}$$

Therefore, in the time domain, the derivative part can be written as the following differential equation:

$$D(t) + \frac{T_d}{N_f}\frac{dD(t)}{dt} = -K_pT_d\frac{dy(t)}{dt}. \tag{6.26}$$

A discrete version of this equation can be obtained by approximating the derivatives with a backward difference, which gives [98], [99]

$$D(t_k) + \frac{T_d}{N_f}\frac{D(t_k) - D(t_{k-1})}{\Delta t_k} = -K_pT_d\frac{y(t_k) - y(t_{k-1})}{\Delta t_k}, \tag{6.27}$$

where $\Delta t_k = (t_k - t_{k-1})$ is the duration between the previous and current sampling instants.

Solving Equation (6.27) for $D(t_k)$ yields the derivative part of the discretized controller, for sampling instant t_k, as

$$D(t_k) = \frac{T_d}{T_d + N_f \Delta t_k} D(t_{k-1}) - \frac{T_d}{T_d + N_f \Delta t_k} K_p N_f \big(y(t_k) - y(t_{k-1})\big). \tag{6.28}$$

Åström [98] highlights that using a backward difference to approximate the derivatives is advantageous in that the parameter $T_d/(T_d + N_f \Delta t_k)$, in Equation (6.28), is in the range of 0 to 1 for all values of the constituent parameters, thus guaranteeing that the difference equation is stable.

As a result, the discrete control signal, $u(t_k)$, is digitally computed at every sampling instant t_k by the following mathematical algorithm:

$$\begin{aligned} P(t_k) &= K_p\big(r(t_k) - y(t_k)\big); \\ D(t_k) &= \frac{T_d}{T_d + N_f \Delta t_k}\Big(D(t_{k-1}) - K_p N_f \big(y(t_k) - y(t_{k-1})\big)\Big); \\ u(t_k) &= P(t_k) + D(t_k) + F_{ff}(t_k); \end{aligned} \tag{6.29}$$

where $F_{ff}(t_k) = f_s$ for the height controller, $F_{ff}(t_k) = 0$ for both the roll and pitch controllers, and K_p and K_d are replaced by the relevant proportional and derivative gain terms from $\boldsymbol{K}$ and $\boldsymbol{C}$, respectively.

6.1.2.2 Virtual Model Controller Implementation

The discrete control equations in (6.29) were implemented for each posture control variable in ROS as a C++ function (`updateController`), as part of the `posture_control` node. A ROS Timer object was used to execute the posture control algorithm at a set frequency of 20 Hz, by calling the `updateController` function in its callback (which was set to trigger every 0.05 s). A fixed timestep value of 0.05 s was used in place of Δt_k in the code implementing Equation (6.29), with satisfactory results. The standard, single threaded ROS spinner was utilized for the `posture_control` node, to continuously pump callbacks until the node is shutdown.

Merely calculating net force and torque components to apply on the robot does not induce any physical control action on it. It is subsequently required to determine the forces needed to be applied by the legs, to achieve the desired force and torques on the body as calculated above, which is the focus of the section that follows.

6.1.3 Foot Force Distribution

Based on the contact and position information of each foot, the force and torques calculated above can be distributed among the individual feet to attain the same resultant force and torque components on the body.

6.1.3.1 Foot Force Distribution Theory

For n legs in contact with the ground ($1 \leq n \leq N$), the force and torque equilibrium equations are

$$\begin{aligned} F_{VMC} &= \sum_{i=1}^{n} f_{i,z} \\ M_{\phi} &= \sum_{i=1}^{n} p'_{i,y} \cdot f_{i,z} \\ M_{\theta} &= \sum_{i=1}^{n} -p'_{i,x} \cdot f_{i,z}. \end{aligned} \tag{6.30}$$

As with Equations (5.1) of the controllability analysis, $f_{i,z}$ represents the z-component of the contact force at foot i. Unlike those equations though, the x- and y-direction position values for foot i in Equations (6.30) above, $p'_{i,x}$ and $p'_{i,y}$, respectively, are those calculated in the body-frame $\{\boldsymbol{B}\}$ by the kinematic model (indicated explicitly by the prime symbols), rather than being calculated relative to the world frame (hence the negative in the pitch moment equation of (6.30), instead of in the roll moment equation as in (5.1)).

The system of Equations (6.30) represented as a matrix equation becomes

$$\boldsymbol{F_{VMC}} = \boldsymbol{P}\boldsymbol{F}_{i,z}, \tag{6.31}$$

with $\boldsymbol{P}$ and $\boldsymbol{F}_{i,z}$ being defined as

$$\boldsymbol{P} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ p'_{1,y} & p'_{2,y} & \cdots & p'_{n,y} \\ -p'_{1,x} & -p'_{2,x} & \cdots & -p'_{n,x} \end{bmatrix}, \qquad 1 \leq n \leq N \tag{6.32}$$

$$\boldsymbol{F}_{i,z} = \begin{bmatrix} f_{1,z} \\ f_{2,z} \\ \vdots \\ f_{n,z} \end{bmatrix}, \qquad 1 \leq n \leq N. \tag{6.33}$$

Since this system has three equations, when $n = 3$, i.e. three legs are in contact with the ground, the system also has three unknowns (the required forces for the three contact legs). Thus, Equation (6.31) has a unique solution, and the three individual foot forces can be computed directly as

$$\boldsymbol{F}_{i,z} = \boldsymbol{P}^{-1}\boldsymbol{F_{VMC}}. \tag{6.34}$$

However, when there are more than three contact feet ($n > 3$), the solution is not unique and the equilibrium equations in (6.30) are typically used as equality constraints for an optimization problem, which minimizes some objective function involving the foot forces, for example [6], [10], [11]. In the

case of $n < 3$, there is no solution to Equation (6.31), but this scenario is generally avoided when using statically stable gaits.

In the present application of Standing Posture Control, the hexapod is always standing on all six legs, so a much simpler force distribution method can be employed, simply to divide the virtual forces evenly amongst all the feet, rather than employing an optimization algorithm. Along with reducing complexity, this approach also significantly reduces the computational expense of the controller, leaving open the potential for implementation on less powerful computer hardware, which could run onboard the robot.

In fact, a similar technique was exploited by Ridderström and Ingvast [4] to distribute the vertical leg forces for the Warp1 quadruped platform. This decision was informed by their review of literature on the Spring Flamingo biped [100], where the authors argued that it is not necessary to solve the force distribution problem exactly, in practice. Ridderström and Ingvast [4] also then found that their posture controller worked quite well without a more advanced implementation.

Proceeding with this simple force distribution approach for the Standing Posture Controller on the PhantomX, the vertical force, F_{VMC}, and roll moment, M_{ϕ}, are evenly divided between all six legs. The pitch moment, M_{θ}, on the other hand, is only distributed between the front and back leg pairs, as the central legs, numbered 3 and 4 (see Figure 5.2), are collinear with the y-axis of $\{\boldsymbol{B}\}$ (in the case of standing) and thus cannot apply a torque about this axis. The resulting force distribution rule calculates the contact forces of each foot of the robot, for the standing case, as follows [19]:

$$f_{i,z} = \begin{cases} F_{avg} + \dfrac{M_{\phi,avg}}{p'_{i,y}} - \dfrac{M_{\theta,avg}}{p'_{i,x}}, & i \in [1,2,5,6] \\ F_{avg} + \dfrac{M_{\phi,avg}}{p'_{i,y}}, & i \in [3,4] \end{cases}, \tag{6.35}$$

where

$$\begin{aligned} F_{avg} &= \frac{F_{VMC}}{6} \\ M_{\phi,avg} &= \frac{M_{\phi}}{6} \\ M_{\theta,avg} &= \frac{M_{\theta}}{4}. \end{aligned} \tag{6.36}$$

It should be noted that this calculation is performed in the body coordinate frame, $\{\boldsymbol{B}\}$. It is, therefore, accurate when the body is level, but when the orientation is not horizontal the gravitational force of the robot (which was added directly into the vertical component, F_{VMC}) actually acts at an angle relative to the body, resulting in the z-direction force equilibrium not being exact. However, the

controller usually acts to restore the body to a horizontal attitude in any case, and when non-zero pitch or roll angles are commanded they are generally kept rather small, so the errors in the above calculations should, for the most part, remain negligible.

A further simplification which is made, as mentioned in the previous section, is that the COM is assumed to be coincident with the origin of $\{\boldsymbol{B}\}$, rather than taking into account its offset in the xy-plane. In principle, this should be included in the equilibrium equations as the offset causes additional moments about the x- and y-axes. However, practically speaking, it is more prudent to consider these additional moments as external disturbances, to be counteracted by the controller. Similar to the case of the mass, discussed earlier, in a real-world situation the position of the COM is not straightforward to measure precisely. It is also sensitive to small changes in the configuration of the robot, such as changing the battery and adjusting or mounting different components on the body. Furthermore, if the robot is to be used to carry a payload, the COM position is virtually guaranteed to be affected. For these reasons it is not desirable for the controller to rely on knowledge of the COM position in order to perform well; it should be able to exhibit satisfactory performance in the presence of any general disturbance forces. Moreover, inclusion of the COM offset introduces both nonlinear terms and coupling between control variables to the system, monumentally increasing the complexity for control.

6.1.3.2 Foot Force Distribution Implementation

The foot force distribution rule described by Equations (6.35) and (6.36) was implemented in ROS as a function, named `distributeStandingFootForces`, in the `posture_control` node. This function is called in the main timer callback, immediately after calling the `updateController` function. As such, the VMC outputs are distributed among the feet at every iteration of the posture controller update. The resulting force setpoints for each leg are published over a relevant topic.

Having determined the desired foot contact forces for each leg, the next step is to devise a leg control scheme to achieve these forces.

6.1.4 Single Leg Force Controller

When the Dynamixel AX-12A joint actuators were introduced in Section 4.1.1, it was stated that they do not provide the capability to directly control output torque and are thus only used to track desired positions. Other ranges of Dynamixel servo motors are available which can be torque controlled [101]; however, these cost more than the AX-12A series and are also incompatible with the PhantomX leg brackets. Consequently, achieving the desired foot forces by controlling the joints to output equivalent torques, calculated using the leg Jacobian, is not an option on the robotic platform being used in this study.

Therefore, a force control strategy is required to be implemented for each leg to regulate the foot contact forces to the desired values by appropriately modifying the motion of the leg. Given that the desired leg positions can be fed into the IK engine, it would be ideal to influence the foot forces through appropriate adjustments to the leg position commands.

Methods such as these were presented in Section 3.4, for the task of interaction control in manipulator robots. Two main categories were discussed, namely indirect and direct force control. It was indicated that the former is useful for controlling the static and dynamic behaviour of the manipulator when experiencing external forces, while the latter category allows for the regulation of contact forces to desired values, in the constrained directions of the manipulator's motion.

Specifically considering the stance legs of the robot, to be used to invoke the necessary forces on the body to control its posture, it becomes apparent that this case corresponds to regulation of force in a constrained direction, pointing to the use of direct force control being appropriate. It is true that, technically speaking, the legs are not completely constrained as their movement does cause motion on the body that they are attached to. However, the feet can be seen as being constrained to stay in contact at a particular point on the ground, unlike the stance legs which move around in free space.

6.1.4.1 Single Leg Force Controller Design

With the above discussion in mind, a direct force control strategy, in the form of a simple PI controller, can be implemented for a single leg, i, according to the following control law:

$$\Delta p_{i,z} = -K_P e_{f,i} - K_I \int_0^t e_{f,i}\, d\tau, \tag{6.37}$$

where

$$e_{f,i} = \left(f_{i,z} - f_{m,i}\right), \tag{6.38}$$

and $\Delta p_{i,z} \equiv$ Adjustment value for z-position of the ith foot (controller output),

$e_{f,i} \equiv$ Force error of the ith foot,

$f_{i,z} \equiv$ Commanded vertical force on the ith foot, as calculated in Equation (6.35),

$f_{m,i} \equiv$ z-component of the force measured on the ith foot (by the force sensor),

$K_P \equiv$ Proportional gain of force controller, and

$K_I \equiv$ Integral gain of force controller.

Effectively, this results in the leg being pushed downward (against the terrain) if an increase in force is required, while a reduction in the contact force is achieved by retracting the leg upward.

In Equation (6.37) the integration of the force error acts to adjust the foot position until the desired force is met, contrary to its use in physically constrained directions for robotic manipulators, where the output creates a position error, to adjust forces with no actual, resultant motion [68]. On the hexapod robot, although the vertical leg motions are not constrained, the VMC adjusts the force reference values based on the motion of the robot. The result is not only force control of the feet, but also control of the desired dynamic response of the robot, without the need for impedance control on the stance legs.

Although the control law described above provides the foundation of the force controller, for practical implementation on the robot some modifications need to be made. Firstly, to prevent the controller from pulling the leg up until the femur servo hits the coxa bracket or attempting to push the leg down, outside of its kinematic range, artificial saturation limits were required to be introduced to the control algorithm. Based on the standing height of the robot, saturation limits of +5 cm and -5 cm were found to be appropriate for use on the PhantomX.

However, integral controllers have the drawback of "winding up" when actuators are saturated, as the resultant, non-zero error signal is continually integrated [98], [99]. The error is then required to have the opposite sign for a long period of time before normal operation of the controller can be restored, which is highly undesirable as it effectively renders the controller useless for that time period. As a remedy to this problem, an anti-windup mechanism, based on back-calculation and tracking, is also implemented in the single leg force control structure.

The principal of this mechanism's operation is to create an extra feedback path around the integrator, in order to drive its input to zero, when the normal feedback path around the process is broken by saturation of the controller output [98], [99]. In particular, when the controller output in Equation (6.37), $\Delta p_{i,z}$, exceeds the saturation limits, the actual output adjustment value, Δz_i, is set to the limiting value. This breaks the feedback path around the leg's motion control process, as the process input remains constant (at the saturation value).

To create the additional feedback path required around the integrator, a tracking error signal, $e_{s,i}$, is formed as the difference between the controller output and actual output. That is,

$$e_{s,i} = \left(\Delta z_i - \Delta p_{i,z}\right). \tag{6.39}$$

This error signal is then fed back into the input of the integrator through a gain $1/T_t$ (see the block diagram in Figure 6.1). When the controller output is within the saturation limits, then $\Delta z_i = \Delta p_{i,z}$ and the tracking error, $e_{s,i}$, is accordingly zero, leaving the normal operation of the controller unaffected. However, when the leg position saturates, Δz_i is set to the limit value while $\Delta p_{i,z}$

continues to grow in magnitude, resulting in a non-zero value of $e_{s,i}$. This creates the additional feedback to the integrator, which acts to drive the integrator's output to a value such that the integrator input becomes zero.

Thus, the input to the integrator takes the form $-K_I e_{f,i} + (1/T_t)e_{s,i}$. During conditions of saturation, the aim is to drive this input to zero at steady state. Hence,

$$\begin{aligned} -K_I e_{f,i} + \frac{1}{T_t} e_{s,i} &= 0 \\ \therefore e_{s,i} &= K_I T_t e_{f,i}. \end{aligned} \tag{6.40}$$

Given that $e_{s,i} = \left(\Delta z_i - \Delta p_{i,z}\right)$, it follows that, at steady state

$$\begin{aligned} \Delta z_{lim} - \Delta p_{i,z} &= K_I T_t e_{f,i} \\ \therefore \Delta p_{i,z} &= \Delta z_{lim} - K_I T_t e_{f,i}, \end{aligned} \tag{6.41}$$

where Δz_{lim} represents the saturation limit value of the output.

The sign of $e_{f,i}$ acts in the opposite direction of $\Delta p_{i,z}$, i.e. negative force errors correspond to positive foot position adjustments (to retract the foot upward) and vice versa. Thus, the result of Equation (6.41) implies that, when the leg motion is saturated, the controller output signal, $\Delta p_{i,z}$, settles on a value just outside of the saturation limit, allowing the control signal to react as soon as the error changes sign, rather than having to first wind down over a long time.

How far outside the saturation limit the controller output settles, as well as how quickly it is reset, are dictated by the feedback gain $1/T_t$. Suitably, the parameter T_t can be interpreted as the "time constant" related to the rate at which the integral is reset, and it is thus aptly referred to as the *tracking time constant*.

Incorporating the abovementioned saturation and anti-windup mechanisms into the force controller leads to a control law, for a single leg i, which is described mathematically as

$$\Delta p_{i,z} = -K_P e_{f,i} + \int_0^t \left(-K_I e_{f,i} + \frac{1}{T_t} e_{s,i}\right) d\tau. \tag{6.42}$$

The controller output is fed through an artificial saturation block to yield the commanded position adjustment, Δz_i, which is required to modify the z-component of the foot position commanded by the gait engine (denoted $z_{d,i}$), creating the adjusted reference position, $z_{r,i}$, for the IK engine. As such,

$$z_{r,i} = z_{d,i} + \Delta z_i. \tag{6.43}$$

This control structure is illustrated more clearly in the block diagram in Figure 6.1 below, where the Foot Force Controller for a single leg i (indicated in blue) can be seen integrated into the greater motion control system for that leg.

For the sake of simplicity, only the z-component of the force sensor measurement (relative to the force sensor's coordinate frame) is used, and the force controller acts only in the body frame, adjusting the z-positions of the feet relative to $\{\boldsymbol{B}\}$. It is for this reason that the x- and y-position commands from the gait engine can be seen, in Figure 6.1, being fed directly into the gait engine, without any adjustment.

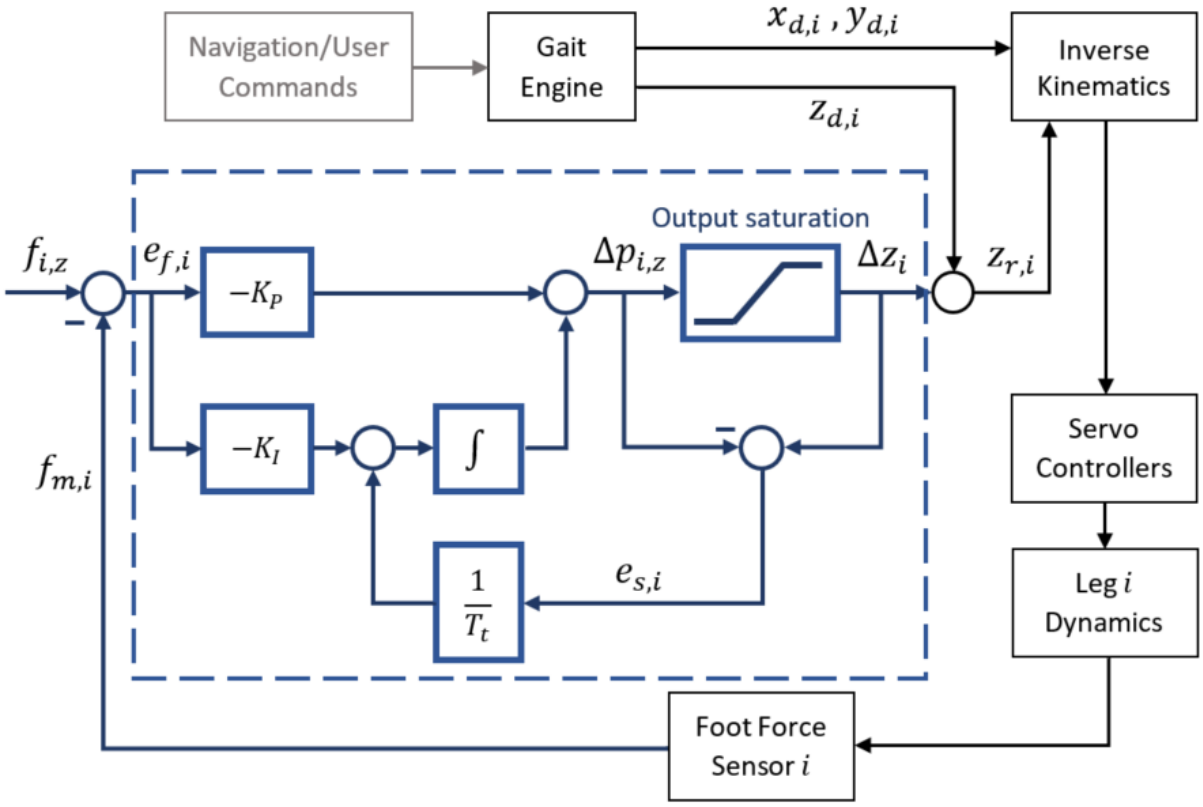


Figure 6.1: Block diagram of Foot Force Controller for a single leg i, indicated in blue (adapted from [19]*)*

When the robot is simply standing, the gait engine outputs are constant. As a result, all the vertical motion of the legs occurs as a result of the force controller. Hence, the control structure described above, can be viewed as a hybrid position/force controller (see Section 3.4.1), in which the x- and y-directions of motion are controlled purely for position, while the z-direction motion is exclusively governed by force control. However, for the z-direction, the command from the position control (gait engine) is still factored in, similar to the parallel control composition described in Section 3.4.3, in which both force and motion control influence the motion (with the former dominating the latter in constrained directions). The reason for this composition of the control structure will become more transparent in Chapter 7, when the situation of walking is also to be handled.

As with the virtual model controller, it is required to discretize the continuous force control Equation (6.42), for digital implementation on a computer. The proportional part of the controller is discretized in the same way as for VMC and is quite straightforward. On the other hand, from Equation (6.42) it can be deduced that the integral action, $I_i(t)$, is given by

$$I_i(t) = \int_0^t \left(-K_I e_{f,i}(\tau) + \frac{1}{T_t} e_{s,i}(\tau)\right) d\tau. \tag{6.44}$$

Differentiating Equation (6.44) produces

$$\frac{dI_i(t)}{dt} = -K_I e_{f,i}(t) + \frac{1}{T_t} e_{s,i}(t). \tag{6.45}$$

Approximating the derivative by a forward difference [98] at a particular sampling instant, t_k, results in

$$\frac{I_i(t_{k+1}) - I_i(t_k)}{t_{k+1} - t_k} = -K_I e_{f,i}(t_k) + \frac{1}{T_t} e_{s,i}(t_k), \tag{6.46}$$

which can be rearranged to form a recursive equation for digitally computing the integral action:

$$I_i(t_{k+1}) = I_i(t_k) + \Delta t_{k+1}\left(-K_I e_{f,i}(t_k) + \frac{1}{T_t} e_{s,i}(t_k)\right), \tag{6.47}$$

where $\Delta t_{k+1} = (t_{k+1} - t_k)$.

The resulting mathematical algorithm for computing the force control action, for a single leg i, looks as follows:

$$\begin{aligned}
&\Delta t_k = t_k - t_{k-1};\\
&I_i(t_k) = I_i(t_{k-1}) + \Delta t_k\left(-K_I e_{f,i}(t_{k-1}) + \frac{1}{T_t} e_{s,i}(t_{k-1})\right);\\
&e_{f,i}(t_k) = \left(f_{i,z}(t_k) - f_{m,i}(t_k)\right);\\
&P_i(t_k) = -K_I e_{f,i}(t_k);\\
&\Delta p_{i,z}(t_k) = P_i(t_k) + I_i(t_k);\\
&\Delta z_i(t_k) = \text{outputSaturation}\left(\Delta p_{i,z}(t_k)\right);\\
&z_{r,i}(t_k) = z_{d,i}(t_k) + \Delta z_i(t_k);\\
&e_{s,i}(t_k) = \Delta z_i(t_k) - \Delta p_{i,z}(t_k);
\end{aligned} \tag{6.48}$$

where $\text{outputSaturation}()$ represents a function that applies the artificial saturation limits to the controller output.

6.1.4.2 Single Leg Force Controller Implementation

In the ROS locomotion stack, the gait and IK engines are both encompassed in the `hexapod_controller` node, where they are updated at every iteration of this node's main loop, sending the relevant commands to the joint actuators. Therefore, in order for the force controller to be able to intercept the gait commands and adjust them at every iteration, it was necessary to incorporate the force control system into the same node.

Thus, a C++ class (`ForceController`) was written and included into the `hexapod_controller` package. This class implements a stack of N identical, single leg force controllers, one for each of the N legs of the robot (six in the case of the hexapod). The main function of this class, named `updateController`, executes the algorithm described in Equations (6.48) for each leg, generating the set of six position adjustment values. Subscribers are also included to receive force sensor data, as well as the desired contact force values, or setpoints, from the force distribution module. All relevant controller parameters are loaded at runtime from the YAML configuration file for the PhantomX.

The source code of the main `hexapod_controller` node was modified to instantiate an object of the `ForceController` type. This object's update function is called at every loop iteration, immediately after the gait positions have been calculated. Once the adjusted reference positions are received from the force controller stack, they are fed into the IK engine (instead of the gait commands) to determine the required joint positions.

While embedding the contact force control system into the `hexapod_controller` node allows for it to update the foot position commands at every iteration, it does subject the controller to inconsistent execution frequencies. This is not only caused by the fact that the locomotion package was not coded to follow strict "real-time" constraints, but also by the presence of an inner loop in the servo communication object, which executes within the main loop of the node in a blocking fashion.

Perusal of the source code of the classes, making up the various functionalities of the `hexapod_controller` node, revealed this anomaly in the `ServoDriver` class (mentioned briefly in Section 4.3.2.1). This class handles the mapping between joint angles and servo positions, along with all the serial communication with the joint actuators. In this class, the main function which transmits the position commands to the servos (`transmitServoPositions`) contains a loop that, at each iteration, increments the relevant servo position commands by a single integer value at a time and transmits these to the servos. Thus, the loop is executed until all the servo position commands reach the goal positions associated with the joint angles requested by the IK engine. The purpose behind using this interpolation loop was to smooth the motions of the legs.

This inner loop is set to run at a particular frequency, referred to as the *interpolation loop rate*, by using the `sleep` function of a ROS Rate object [102]. The Rate object in ROS does not merely execute a sleep by pausing the process for a set amount of time, like most implementations of these functions do. Instead, the Rate instance attempts to keep a loop at a particular frequency, by accounting for the time already spent on the processes in the loop and pausing the process for the remainder of the time in the desired loop period. If the time already used by the loop exceeds the desired period, the sleep function simply does not pause the process at all. Thus, no real-time constraints are imposed on the program, but an attempt is made to achieve a desired execution frequency.

In the context of the `hexapod_controller` node, such a Rate instance is also used to limit the maximum loop execution rate to 500 Hz for the PhantomX. During this main loop, the `transmitServoPositions` function of the `ServoDriver` instance is called after the IK engine calculates the desired joint angles. This function call executes the interpolation loop, at a maximum rate of 550 Hz, until the desired joint angles have been commanded from the actuators, thus blocking the execution of the main loop while interpolating. As a result, if very small changes in foot positions are required, corresponding to small changes in the servo positions, then the interpolation loop only executes a few times and the main loop is not blocked for too long, allowing it to run at rates relatively close to the maximum of 500 Hz. However, considering that a single integer value increment to the servo position command corresponds to an angular increment of only 0.29° (see Section 4.2.1), it rarely occurs that only a few iterations of the interpolation loop are required. If a significant change in position is required for even one servo, the interpolation loop blocks the main loop for a considerable amount to time, dropping its execution rate to well below 500 Hz.

Some measurements of the time taken between main loop iterations, while making the robot walk around (using just the gait engine, without posture control), revealed that the necessary leg motions resulted in the loop durations often being as large as 15 – 20 ms, sometimes larger. This indicates that the main loop frequency can even drop below 50 Hz, on occasion.

As a side note, it should be mentioned that although the main loop of the node can become this slow, the frequency of execution of SYNC_READs (to obtain servo position feedback) is not generally affected by this drop in the main loop rate, despite the `readServoPositions` function being coded as a part of the `ServoDriver` (see Section 4.3.2.1). The reason for this is that the `readServoPositions` function is called inside of a timed callback, and the `hexapod_controller` makes use of an asynchronous, non-blocking spinner [77]. Two threads are utilized by the spinner for pumping callbacks, without any interruption from the processes running in the node's main loop.

While the software architecture described above (with a main control loop and inner interpolation loop) could be adequate for the open-loop control of gaits (which the `hexapod_controller` node manages to achieve quite effectively), the inconsistent execution frequency can certainly become problematic for feedback control systems, such as the force controller being implemented here. For the standing case, however, the only motion demanded of the legs is from the force controllers themselves, which do not generally command enormous position changes in a single time step. Therefore, the variable loop frequency did not pose a significant problem, with the execution rate being considerably faster than 50 Hz, in general. However, this issue becomes more significant in the walking case, as will be discussed in more detail in Chapter 7.

Another point to be drawn out of this discussion is that, since the force controller frequency mostly stays above 50 Hz, but can sometimes drop lower, the `posture_control` node frequency was kept to 20 Hz. There would be no point executing the VMC, with force distribution, and transmitting the foot force commands faster than the force controller can act on them.

Moreover, considering that the theoretical maximum frequency for this loop is 500 Hz, it was necessary to receive force sensor measurements at a higher rate, so as to not use “stale” measurements for the task of control. The OptoForce DAQs offer four sampling rates of 30 Hz, 100 Hz, 333 Hz, and 1000 Hz. Evidently, the only option faster than 500 Hz is 1000 Hz, hence the use of this sampling frequency for force measurements, as expressed in Section 4.3.2.3.

The full stack of N single leg Force Controllers are shown integrated into the greater control structure of the hexapod platform in the following section.

6.1.5 Overall Control Structure

A high-level block diagram illustrating the overall control structure is presented in Figure 6.2 below. The diagram shows how the implemented posture control system, comprised of Posture Calculation, VMC, Foot Force Distribution, and six, individual Foot Force Controllers, fits into the larger robotic system, which includes the locomotion generation software stack.

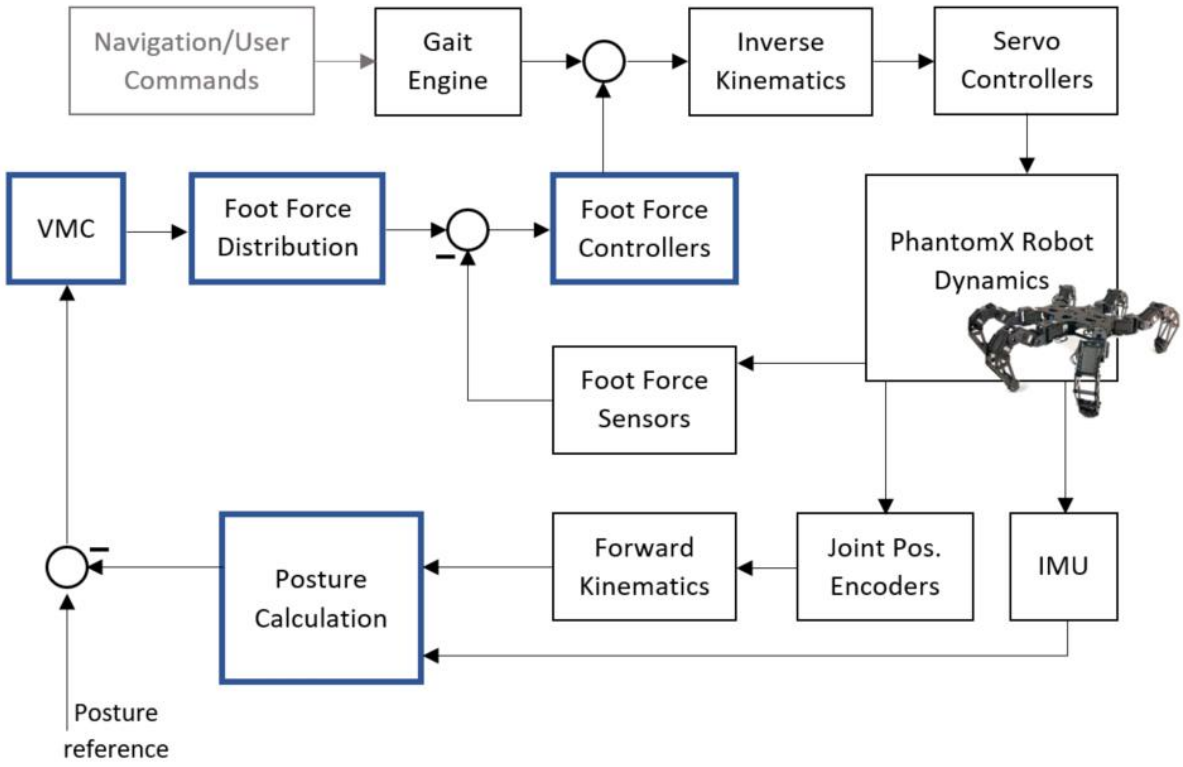


Figure 6.2: High-level block diagram depicting overall control structure on hexapod robot; posture control system is indicated by blue blocks (adapted from [19]*)*

In the block diagram above, the posture control system, indicated by blue blocks, can be seen closing a feedback loop around the robotic system, or process, by feeding back measurements from the sensory system and making appropriate adjustments to the foot position commands used for motion control. The entire posture control system has essentially been "bolted on" to the existing control software, with minimal changes to the locomotion engine being required in order to incorporate closed-loop posture control functionality. This is an important feature for developing control systems which can easily be used with a wide variety of commercial robotic platforms and commonly available locomotion software suites.

For this study, all the necessary controller gains and parameters were determined through manual tuning. It would be possible to select the gains by, for example, utilizing techniques to place the closed-loop poles at desired locations and accordingly shape the dynamic response, such as those described in [11], or making use of optimal control theory to determine the gains which yield a linear quadratic regulator (LQR) [94]. However, further investigation into the dynamic process would be required to make effective use of such methods. For these particular methods, the former would warrant a more thorough analysis of the physical and mechanical properties of the robot so that appropriate closed-loop pole locations could be chosen. Similarly, the latter method of LQR design would only

really return a superior controller if the designer has a good idea of how to suitably select the *state*- and *control-weighting matrices*.

It must be kept in mind that this study is the first iteration of controller design for the robotic platform in use, and the focus is geared towards developing, and investigating the viability of, these control strategies on the platform, rather than on optimizing the design or obtaining the perfect parameters. By forming a foundation of the control systems on the robot being used, this study creates a platform for subsequent investigations to look deeper into the matters related to designing said control systems. In fact, this type of iterative controller design is often observed in the literature, such as the example of the virtual model control system used on the legged robots at ETH Zurich's Autonomous Systems Lab. In this case, the study in [103] aimed to initially implement a simple, manually tuned VMC strategy on the ALoF quadruped, forming the basis for progressively more advanced implementations on StarlETH, in studies such as [6], [11].

Furthermore, Ridderström and Ingvast [4] maintain that excessive parameter tweaking is not imperative, as a useful posture controller should be able to operate with a wide range of parameters for robustness.

For the reasons mentioned above, manual parameter selection was deemed sufficient for this study.

A fairly standard parameter tuning method was carried out. Firstly, initial parameters were obtained by considering the orders of magnitude of the errors, and other terms, in the controller equations and selecting the orders of magnitude of the gains to yield realistic controller output magnitudes. Gains were subsequently increased until the dynamic behaviour of the system, in response to step changes in the reference values, approached unstable conditions, thus determining maximum values of the gains. These were then decreased, or "detuned," by a fair amount, to keep the system well within the stable region. Finally, adjustments were made by testing the step responses as parameters were tweaked, until a satisfactory compromise between accuracy, speed, and smoothness of the response was achieved.

Tables 6.1 and 6.2 provide the final parameter values used for the Standing Posture Control system.

Table 6.1: Parameter values used for VMC in Standing Posture Control system

Posture Component	Symbol (X)	Parameter Values		
		K_X	C_X	$N_{f,X}$
Height	z	1500.0 N/m	15.0 N·s/m	2.0
Roll	ϕ	90.0 Nm/rad	1.5 Nm·s/rad	2.0
Pitch	θ	90.0 Nm/rad	1.5 Nm·s/rad	2.0

Table 6.2: Parameter values used for Single Leg Force Controller in Standing Posture Control system

Parameter	Value
$\boldsymbol{K_P}$	5.0e-3 m/N
$\boldsymbol{K_I}$	7.0e-3 m/(N·s)
$\boldsymbol{T_t}$	1.0 s

While contact force setpoints are continuously updated by the VMC, default posture setpoints are required for when the robot is simply standing in its normal configuration. The default height setpoint of the body frame, $\{\boldsymbol{B}\}$, was calculated based on the standing height which was set for the robot, as well as the vertical position of $\{\boldsymbol{B}\}$ relative to the robot's body. Roll and pitch angle setpoints of 0° were selected to maintain a horizontal orientation of the robot's body, as standard. These default posture setpoints are summarized in Table 6.3.

Table 6.3: Default posture setpoint values for Standing Posture Controller

Setpoint Component	Value
Height, $\boldsymbol{h_d}$	0.123 m
Roll angle, $\boldsymbol{\phi_d}$	0°
Pitch angle, $\boldsymbol{\theta_d}$	0°

6.2 Experimental Setup and Tests

A variety of experimental tests were carried out to evaluate the effectiveness and performance of the implemented posture control system. Controller speed and accuracy were tested through a series of escalating step tests, on flat and level terrain, in which step changes of increasing magnitude were applied independently to the height, pitch, and roll references. Each step test was repeated at least three times, and the results compared, in order to assess the precision of the control system. These various step tests are described in more detail in Section 6.2.1 below.

While step tests on a level surface are useful for evaluating the ability of the controller to track the desired reference, it is also important to investigate whether the control system can regulate the

robot's posture on uneven terrain and in the presence of disturbances, since these are the key situations which necessitate posture control in walking robots in the first place. Therefore, the control system was tested with the robot standing on artificial rocks (uneven terrain) as well as on a dynamic balance board, described in more detail in Sections 6.2.2 and 6.2.3, respectively.

A Vicon motion capture system [16] was used to track the robot and balance board during the experimental tests, thereby allowing the posture estimates used by the controller to be compared to ground truth measurements, as well as providing an indication of the magnitudes of the disturbances induced by tilting the balance board. Specific details regarding the Vicon object models and processing of Vicon data are discussed in Section 6.2.4 below.

6.2.1 Step Tests on Flat, Level Terrain

Individual step tests were carried out for the three control variables, namely height, pitch, and roll, so that their responses could be analysed independently, while also inspecting how significantly the "unstepped" control variables are disturbed by adjustment of a single control variable. Escalating steps, of positive and negative magnitudes, were used so that a single test procedure would evaluate controller performance for various step magnitudes, setpoint values at various distances away from the nominal value, and both positive and negative steps.

In view of the fact that the force controller saturates the leg motion at ±5 cm, the height setpoint was limited to 4.5 cm above and below the nominal value of 12.3 cm, i.e. the maximum and minimum values used for the height setpoint were 16.8 cm and 7.8 cm, respectively. Successive step magnitudes were incremented by 1 cm at a time, changing step direction each time.

Saturating the vertical leg motion also limits the orientation that the robot body can achieve. The maximum roll and pitch angles achievable by the control system before saturating were calculated to be approximately 11.65° and 12.75°, respectively. Maximum and minimum setpoints of 0.15 rad (approximately 8.6°) and -0.15 rad, respectively, were thus used for both roll and pitch, ensuring that the robot stayed well clear of the saturation limits. In this case, the magnitudes were increased in increments of 0.03 rad, starting at the nominal values of 0 rad.

6.2.2 Uneven Terrain Tests

Artificial rocks were arranged in a manner which emulated uneven terrain conditions (see Figure 6.3 below). The robot was made to stand up on these rocks with the controller disabled and, alternately, enabled, so that the uncompensated and compensated postures could be compared. This test was vital to determine whether the controller can handle general uneven terrain conditions, where the footholds will not necessarily be on a planar surface.

A step test of the height controller was also carried out while standing on the rocks, to assess the performance with the legs being at various heights and the contact planes of some feet being at slight angles to the horizontal. Since the height difference between the highest and lowest footholds on this rock configuration was in the region of 0.07 m, some of the legs were already quite close to their saturation limits. Thus, the step size was limited to 0.03 m for this test.

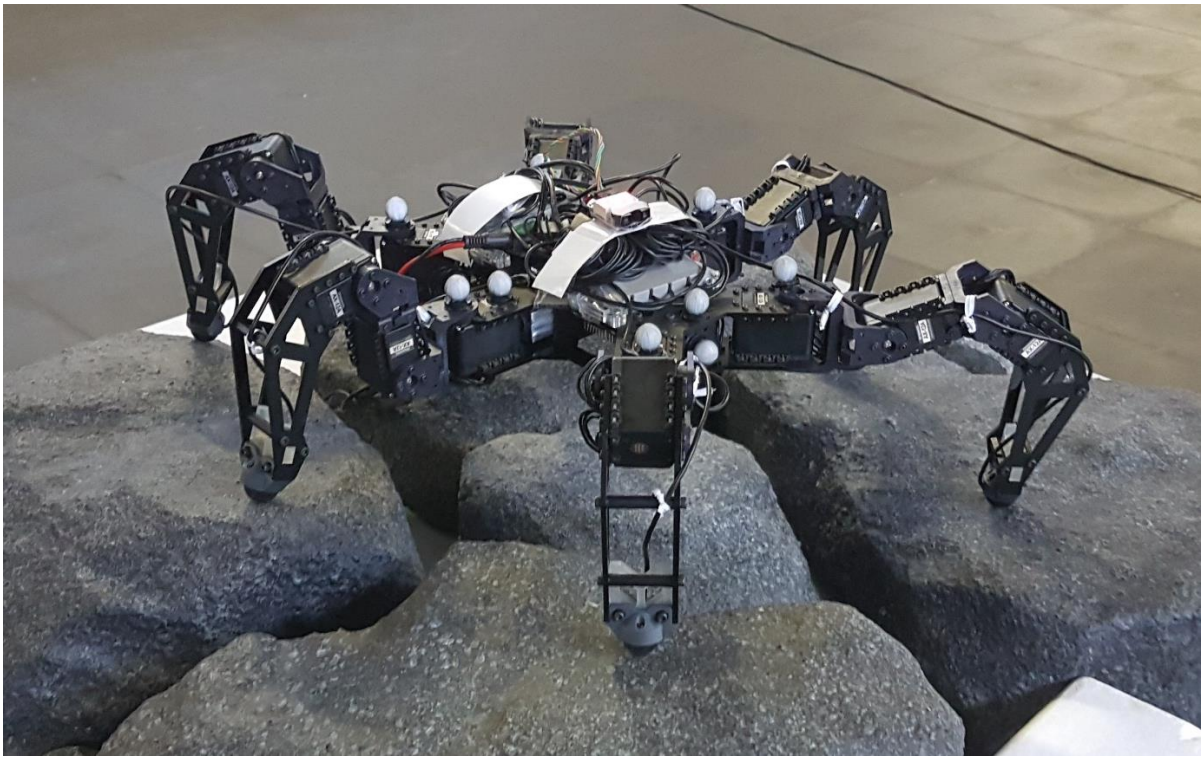


Figure 6.3: PhantomX hexapod robot platform standing on artificial rocks

6.2.3 Balance Board Tests

Tilting a balance board on which the robot is standing is an effective means of inducing force disturbances on the legs and, thus, disturbances to the robot's posture. Consequently, the controller's effectiveness to regulate the robot's posture can be tested both in the presence of disturbances and under dynamic terrain conditions, by conducting balance board tests.

For this purpose, a tripod mount plate was attached to the underside of a wooden board, allowing it to be mounted onto a tripod. A ball and socket joint on the tripod provided sufficient rotational freedom to tilt the board as desired.

The balance board was gradually tilted back and forth manually, first in the roll and pitch directions independently, then in a circular motion, both clockwise and anticlockwise, to create coupled roll and pitch disturbances. Taking into account the controller saturation limits, the magnitudes of the board roll and pitch angles were limited to approximately 10.5° and 11.5°, respectively. Tilting rates of up to roughly 10.5°/s were applied to the board.

In Figure 6.4 below, the PhantomX can be seen standing on the balance board, mounted atop the tripod.

Figure 6.4: PhantomX hexapod robot platform standing on balance board

6.2.4 Vicon Setup and Data Processing

This study made use of a Vicon motion capture system belonging to the Mobile Intelligent Autonomous Systems (MIAS) research group at the CSIR. It consists of twelve cameras that track objects, to provide ground truth measurements of their position and orientation. The cameras are attached to two data acquisition machines, connected to a desktop computer on which the Vicon Tracker software package is installed. Vicon Tracker provides a graphical user interface (GUI) which is used to calibrate the Vicon system, set the location and orientation of its world frame, create masks to remove noise detected by the cameras, and create object models, which are discussed below.

Both the PhantomX and the balance board were tracked using the Vicon. Not only was the balance board tracked to obtain its roll and pitch information during the dynamic balancing tests, but it was also used as a reference against which the height of the robot could be measured during the flat terrain step tests, as can be seen in Figure 6.5.

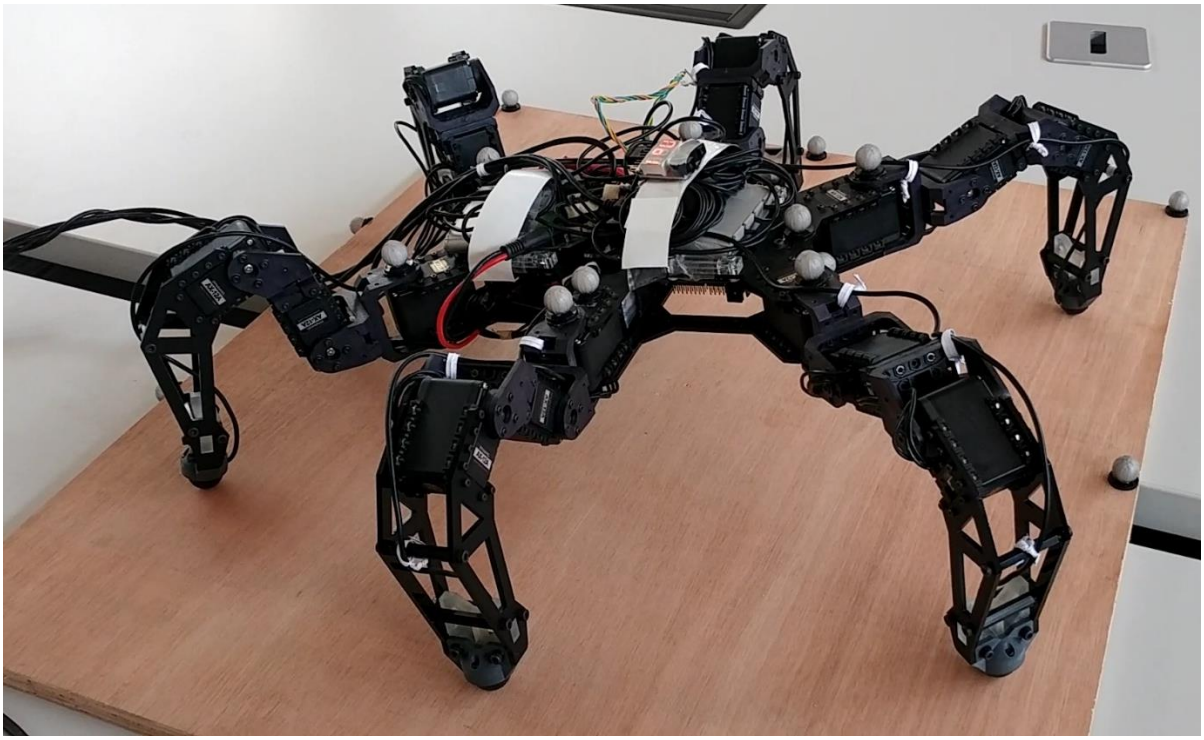


Figure 6.5: PhantomX standing on level balance board for height reference in Vicon measurements of flat terrain step tests

In order for the Vicon to be able to track the robot and balance board, it was necessary to place reflective markers on them (using the double-sided tape provided with the markers). On the PhantomX, markers were attached only to the main body, as this is the rigid body whose motion is of interest, while markers on the balance board were only attached along the edges, leaving sufficient room on the board for the PhantomX. To avoid ambiguity in the orientations of the bodies, the markers were required to be arranged in a non-symmetric and non-planar configuration around each body. The markers can be seen attached to the robot and balance board in the above figures, as well as in Figures 4.14 and 4.15 of Chapter 4 in the case of the PhantomX.

For the Vicon system to be able to recognize these marker configurations as rigid bodies, object models had to be created defining the relationships between the markers. This was accomplished with the Vicon Tracker software package. In Vicon Tracker, the marker configurations for the PhantomX and balance board were created as separate objects, each with their own coordinate frames. The PhantomX object frame was orientated the same way as the body frame, $\{\boldsymbol{B}\}$, i.e. with the x-axis pointing forward and z-axis upward. Since the software only allows for the frames to be "snapped" to coincide or align with specific markers, the origin of the PhantomX object frame was able to be located such that it is coincident with the z-axis of $\{\boldsymbol{B}\}$, but slightly higher than the origin of $\{\boldsymbol{B}\}$. Furthermore, the limited ability to align the axes of the object frame meant that a small deviation between the orientations of this object frame and $\{\boldsymbol{B}\}$ were indeed present. Figure 6.6 shows a screenshot of the GUI of Vicon Tracker, in which the PhantomX object model and its coordinate frame can be seen. As

in Figure 5.2 which showed the body coordinate frame, $\{\boldsymbol{B}\}$, the x-, y-, and z-axes of the Vicon object coordinate frame are indicated in red, green, and blue, respectively, in the figure below.

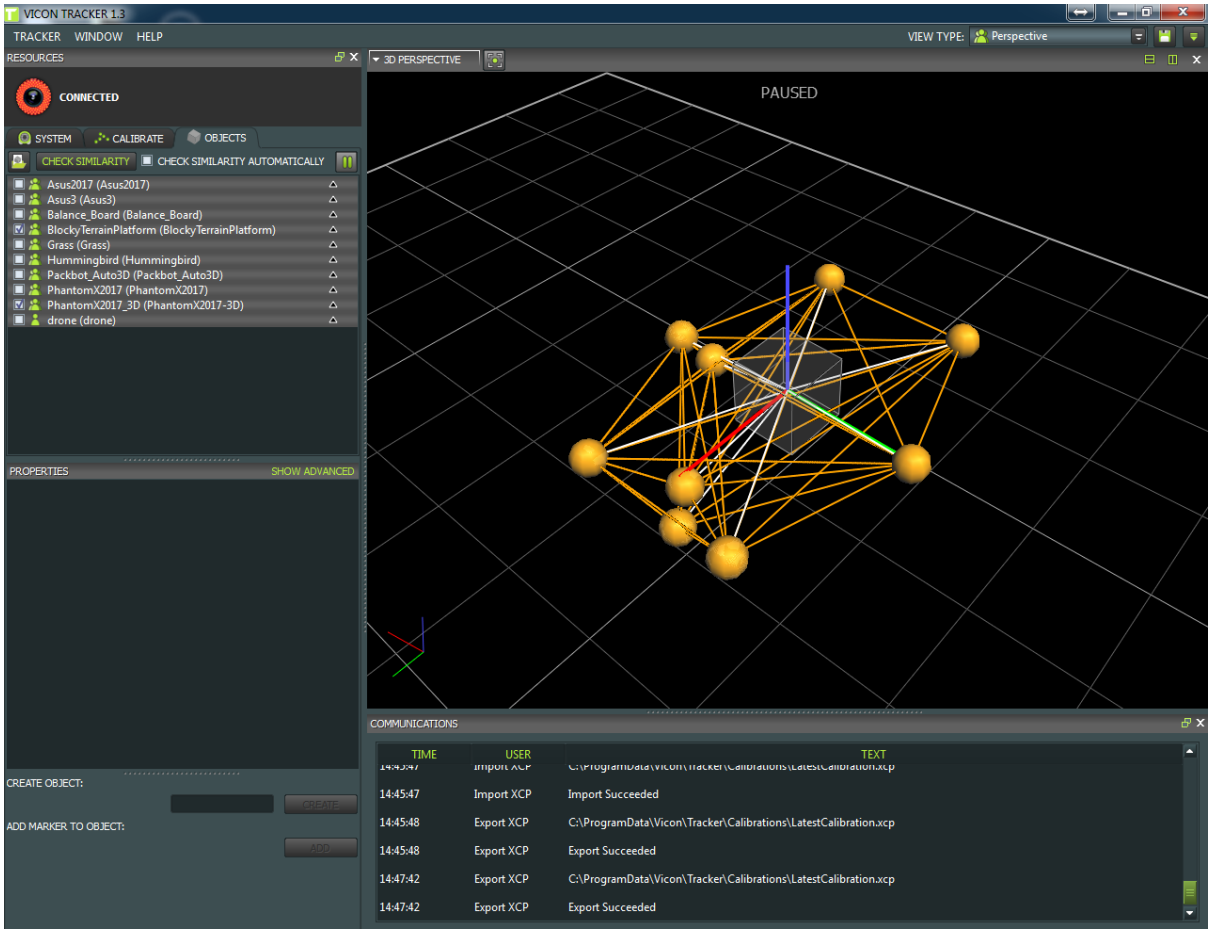


Figure 6.6: Screenshot of Vicon Tracker GUI, showing PhantomX object model and coordinate frame (x-, y-, and z-axes are indicated in red, green, and blue, respectively)

Similarly, the coordinate frame for the balance board object was located in the middle of the board, slightly above its surface (in line with the centres of the Vicon markers placed on the board). It was orientated to have its xy-plane as close to being parallel to the board surface as possible, with the z-axis pointing upward. When the PhantomX was made to stand on the board, it was orientated such that its x- and y-axes pointed in the same directions as those of the balance board, so that the roll and pitch directions for the two bodies would correspond.

Ground truth measurements of these two objects were recorded as rosbags, through the use of the `vicon_bridge` driver package in ROS [104]. For each object, the Vicon provides a translation, which represents the position of its frame's origin, along with a rotation, in the form of a quaternion vector. MATLAB scripts were written to convert the various rosbags into MATLAB-readable data, using the open source `matlab_rosbag` [105] library, in a MATLAB environment installed on an Ubuntu machine. This allowed for all the post-processing of the Vicon data to be performed in MATLAB.

In view of the discrepancies between the orientations of the Vicon object frames and the actual bodies at their null positions, described above, post-processing of the Vicon data included the removal of these small, constant offsets. In addition, the height offsets of the object frames meant that the points in space corresponding to the origin of $\{\boldsymbol{B}\}$ and the centre of the balance board surface were required to be transformed into the world frame description, according to the transformation Equation (3.15) given in Section 3.2.1.3. This transformation equation is repeated here for convenience:

$$\boldsymbol{p} = \boldsymbol{R}\boldsymbol{p}' + \boldsymbol{p}_b. \tag{6.49}$$

In the context of a Vicon object, $\boldsymbol{p}_b$ and $\boldsymbol{R}$ refer to the position and orientation, respectively, of the object frame, relative to the world frame, as measured by the Vicon. With $\boldsymbol{p}$ being the position of the desired point described in the world frame, $\boldsymbol{p}'$ describes the offset between the object frame and the desired point, relative to the object frame. Hence, for the two Vicon objects which only have height offsets between the frame origins and the desired points,

$$\boldsymbol{p}' = \begin{bmatrix} 0 \\ 0 \\ p_z' \end{bmatrix}, \tag{6.50}$$

where p_z' represents this height offset. Based on the size of the Vicon markers and their distances to $\{\boldsymbol{B}\}$ and the balance board surface, specific height offset values of -7.539 mm and -9 mm were required to be used for the PhantomX and balance board, respectively.

While the transformations above yield the position measurements of the two bodies being tracked, the Euler angles corresponding to the orientation were calculated using the mapping from a quaternion, provided in Equation (3.26) of Section 3.2.1.4.2. In MATLAB, the functions `quatinv`, `quatrotate`, `quat2angle`, and `angle2quat`, which are included in the Aerospace Toolbox, were utilized to manipulate the quaternion provided by the Vicon, as well as to convert between quaternion and Euler angle representations. Thus, the computations above yield the ground truth values of the posture components of the robot, as measured by the Vicon system.

6.3 Results and Discussion

The results of the experimental tests described above are presented and discussed in the sections that follow. It should be noted that, while multiple runs of the tests were conducted, the results were found to compare very well between runs, suggesting that the controller performance is remarkably consistent. Thus, for the sake of brevity, only representative plots of the typical response are presented below, rather than the results of all the runs.

6.3.1 Step Tests on Flat, Level Terrain

Results of the three individual step tests, corresponding to the three posture control variables, are presented and discussed separately below.

6.3.1.1 Height Step Response

Figure 6.7 shows the response of the robot to step changes in the reference height. In the figure, the height, roll, and pitch responses are plotted separately, with ground truth measurements from the Vicon indicated in blue, the posture estimates calculated from sensor measurements indicated in red, and the posture reference values, or setpoints, indicated by black dashed lines.

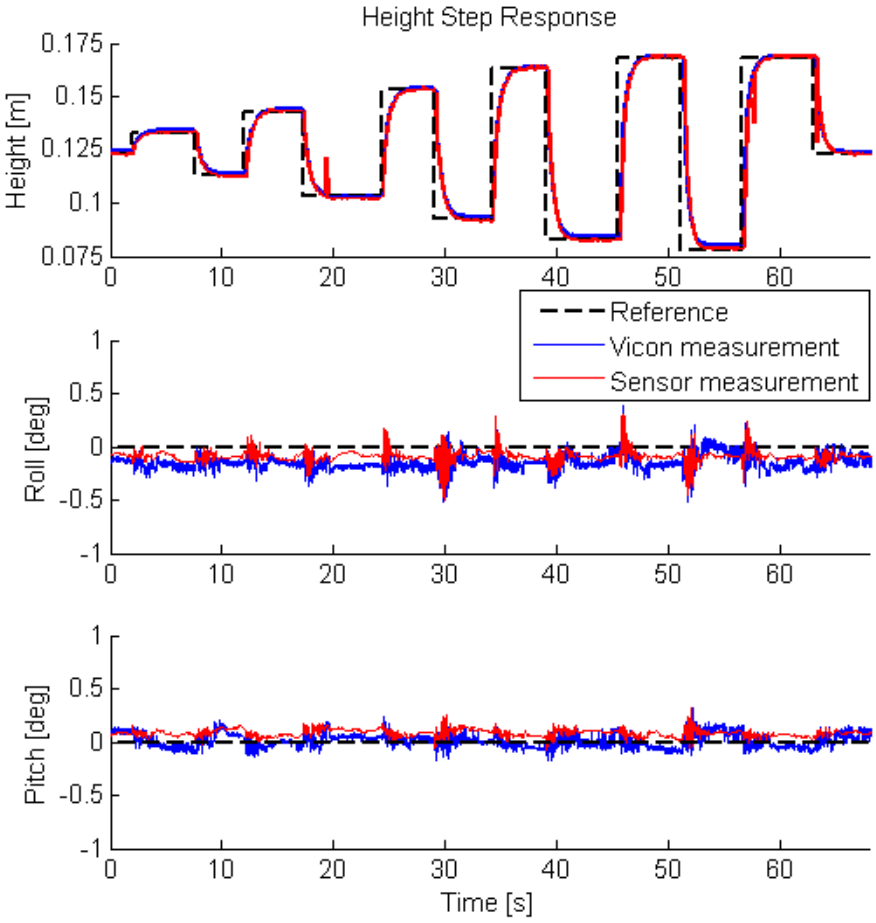


Figure 6.7: Posture response of system to step changes in reference height [19]

Firstly, it can be seen that the Vicon measurements of orientation compare favourably to the orientation estimates from the IMU. There is, however, a small offset, of up to 2 – 3 mm, between the estimated height and that measured by the Vicon system. It is likely that this offset is a result of small errors in the kinematic model, as well as inaccuracies in the servo position readings which have a finite resolution, giving rise to cumulative round off errors. Another source of error is the tiny deformation experienced by the force sensors under load. Although, the magnitude of these deformations would

only be a fraction of a millimetre under the static loads experienced by the sensors in these tests and would thus only have a minor contribution to the height errors observed (as compared to the other sources of error). Yet another factor which could play a role is a potential error in the vertical feedforward force value used in the controller, as the weight of the robot is affected by an assortment of aspects, such as the pull of the tether between the computer and laptop, the particular battery pack being used at the time, and the placement of Vicon markers on the body, among others. Again, this source of error is not expected to play as large a role as the errors from the kinematic calculations, but it should be kept in mind.

Furthermore, the momentary spikes observed in the sensor-measured height are attributed to occasional errors in reading the servo positions (which are used to calculate height through the kinematic model), such as corrupt return packets received from the hardware. These hardly have any significant impact on the overall response of the robot though, evidenced by the fact that the Vicon measurements of posture are unperturbed when these sensor measurement errors occur.

Despite the small height measurement errors described above, the height response is sufficiently accurate, usually settling to within 1.5 mm of the reference value. The magnitude of the maximum steady state error is approximately 2 mm for large steps, as measured by the Vicon, and slightly more than 0.5 mm for the estimated height.

Given that only PD control is used by the VMC for controlling posture, without an integration term (see Equation (6.15)), some steady state error is expected. It also seems sensible that the error would be larger for reference values further away from the nominal pose, as the orientation of the foot sensors would become increasingly less vertical the more the legs are raised or lowered, resulting in misalignment of the force sensor measurement frame with respect to $\{\boldsymbol{B}\}$. This effectively decreases the measured forces used by the force controllers on each leg, resulting in tracking errors. It is, in fact, quite gratifying that the response shows such a small steady state error in spite of these factors.

The speed of the height response is also quite good, generally having a 10% – 90% rise time of approximately 0.9 s. The response does not overshoot the reference, settling smoothly without any significant oscillation. Of course, the controller could be tuned for an even faster response, but this would result in more oscillatory, and possibly jerky, motion, which would be undesirable.

Both [4] and [11] obtained considerably larger steady state height errors in their experimental tests, despite using smaller step sizes, as well as an integration term in the controller in the case of [4].

Additionally, the plots in Figure 6.7 indicate that the roll and pitch angles only experience small perturbations (<0.6°) when a height step is applied, and these are reduced quickly by the controller.

Also, the steady state error values of roll and pitch when the height reference has been stepped are exceptionally small, with a maximum magnitude in the region of 0.3° at large step sizes. As with the height error, the larger roll and pitch errors at larger step sizes can be explained by the errors in force measurements in these poses.

6.3.1.2 Roll and Pitch Step Responses

Posture response plots for escalating roll setpoint steps are presented in Figure 6.8, while those for pitch angle steps are depicted in Figure 6.9.

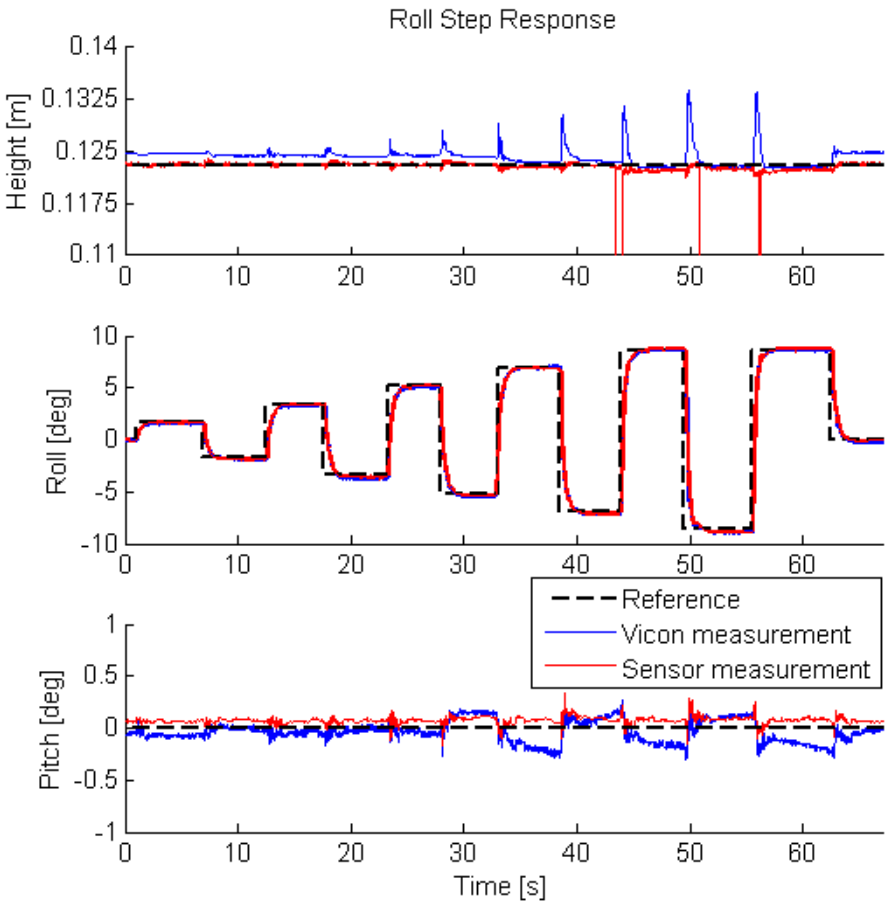


Figure 6.8: Posture response of system to step changes in reference roll angle [19]

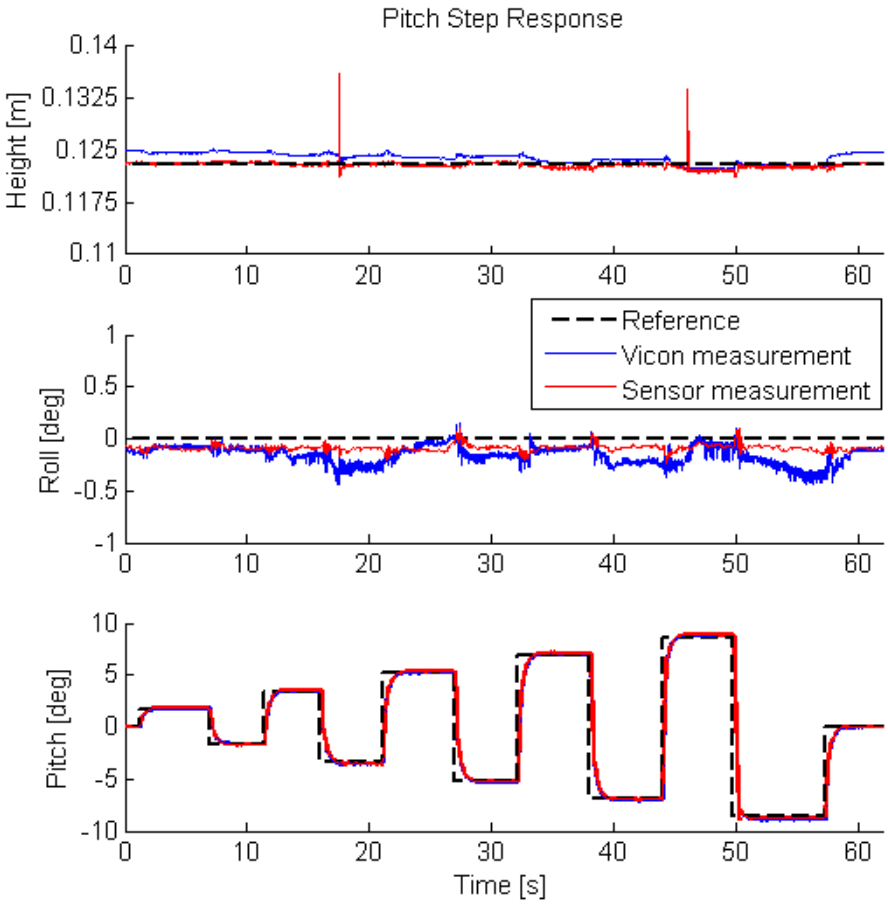


Figure 6.9: Posture response of system to step changes in reference pitch angle [19]

In both cases above, the responses are remarkably fast and accurate, having 10% – 90% rise times in the region of 0.5 s and steady state error magnitudes which remain below 0.3°. Little oscillation is observed in the responses, only really becoming noticeable for large step sizes. For roll angle steps, the steady state errors are much smaller for positive reference values than negative ones. A possible cause could be an offset of the COM towards the left of the robot body. Considering that the COM position offset (in both the x- and y-directions) has not been taken into account in the control algorithm, the errors of the steady state responses in the above plots are actually extremely small.

With regard to the response of Figure 6.8, it is apparent that pitch angle perturbations are kept small by the controller when stepping the roll angle, with the pitch angle errors not exceeding 0.3°. Similarly, Figure 6.9 indicates that roll angle perturbations are kept below 0.5° when stepping pitch.

However, the height responses in both of these tests show a slightly more notable steady state error at large roll and pitch angles, particularly for the Vicon measurements. The reason that the estimated height values start deviating from the actual height at larger angular displacements is that they are calculated in the body frame, $\{\boldsymbol{B}\}$, which rotates with the robot body, thus resulting in a misalignment

between $\{\boldsymbol{B}\}$ and the world frame. Nevertheless, the actual (ground truth) height of the robot stays within about 3 mm of the reference value, at steady state.

Figure 6.8 also reveals that the transient response of the height exhibits momentary jumps, of up to 11 mm at most, when applying large roll angle steps. The most probable cause of this is the slight leg lift observed as the robot initially starts adjusting the roll angle, in response to large reference step changes. This likely happens because the VMC initially commands negative forces from some of the legs in an attempt to achieve a large enough roll moment. It would be possible to eliminate this behaviour by constraining the foot force commands to positive values, but this is likely to slow down the roll response slightly. Moreover, since the lifting of the legs does not appear to have any severely negative effects on the response of the robot, it was left as is.

Overall, the roll and pitch step response results are comparable to the experimental results obtained in [4], [11]. Furthermore, the fact that the perturbations observed in the unstepped posture components have very small magnitudes (in all three of the above tests), indicates that the posture controller is quite effective at decoupling the three posture components in spite of the actual coupling present in the dynamics of the system (as alluded to in Section 6.1.2).

6.3.2 Uneven Terrain Tests

As described previously, the uneven terrain tests comprised of comparing the uncompensated and compensated postures of the robot while standing on an artificial rock arrangement, as well as a step test of the height while also standing on these rocks (with posture control enabled). The results of these tests are presented and discussed below.

6.3.2.1 Uncompensated vs. Compensated Postures While Standing on Uneven Terrain

Figure 6.10 demonstrates the robot standing on the rock arrangement in both the uncompensated and compensated states. It is quite evident from the images that, without compensation, the robot posture is unfavourable, having roll and pitch angle magnitudes of approximately 9.5° and 1°, respectively. Furthermore, while not being clearly visible in the image, it must be noted that not all the feet were in contact with the terrain when the posture controller was disabled.

On the other hand, with active posture control the body of the robot is kept level, exhibiting maximum roll and pitch magnitudes of around 0.3° and 0.06°, corresponding to improvements of approximately 96.8% and 94.0%, respectively. Furthermore, all six feet were also observed to be in firm contact with the terrain. This condition is preferable for a legged robot as it improves “traction” when walking, while also improving the weight distribution among the legs, and thus the energy efficiency as well. These results show a very clear improvement over the uncompensated case, as expected.

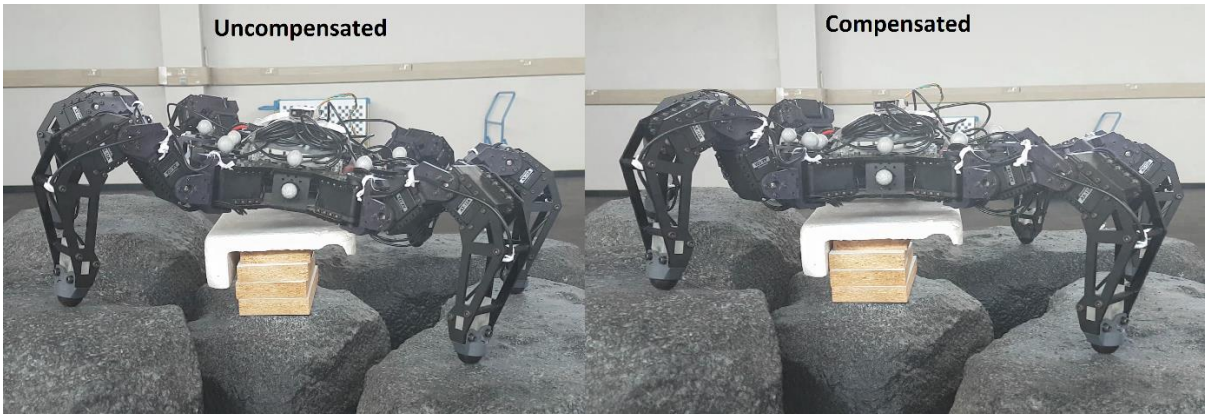


Figure 6.10: Comparison of PhantomX hexapod robot platform standing on uneven terrain, with (right) and without (left) posture control activated [19]

6.3.2.2 Height Step Test on Uneven Terrain

Plots of the height step response of the robot while standing on the same uneven terrain as above are illustrated in Figure 6.11. Only the estimated height is shown here, as there is no single reference height from which to calculate the robot height from Vicon measurements, for this test.

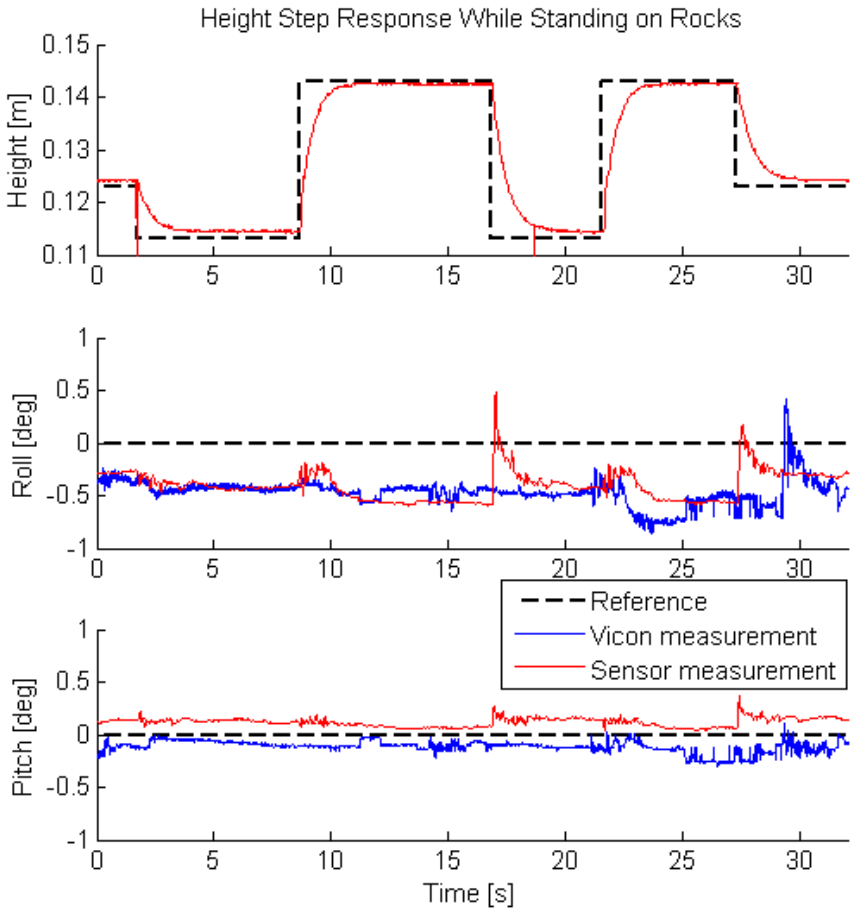


Figure 6.11: Posture response of system to step changes in reference height, while standing on uneven terrain [19]

The height response is very similar to that of the step tests conducted on flat and level terrain, with the largest observed steady state error magnitude being close to 2 mm. Roll and pitch angle magnitudes are limited to less than 1° and 0.4°, respectively, with the roll having a constant offset of approximately 0.5° in magnitude. Larger roll and pitch angle errors than those seen earlier are to be expected, as most of the feet are significantly higher or lower than their nominal height. In addition, the inclination of the contact surface beneath some of the feet results in vertical force measurement errors. Overall, the performance of the controller on uneven terrain compares satisfactorily with its performance on flat and level terrain.

6.3.3 Balance Board Tests

The results of the balance board test are presented in the form of roll and pitch angle plots in Figure 6.12. Both the board and robot orientations, as measured by the Vicon system, are displayed for comparative purposes, with the robot orientation indicated in blue and the board orientation denoted by magenta lines.

Clearly, disturbances induced by tilting the board are responded to well by the robot. The roll and pitch angle magnitudes are kept well below 4°, which is approximately one third of the maximum angular displacement of the board. These results appear to be comparable to those obtained in [4] for a similar test using the Warp1 quadruped platform.

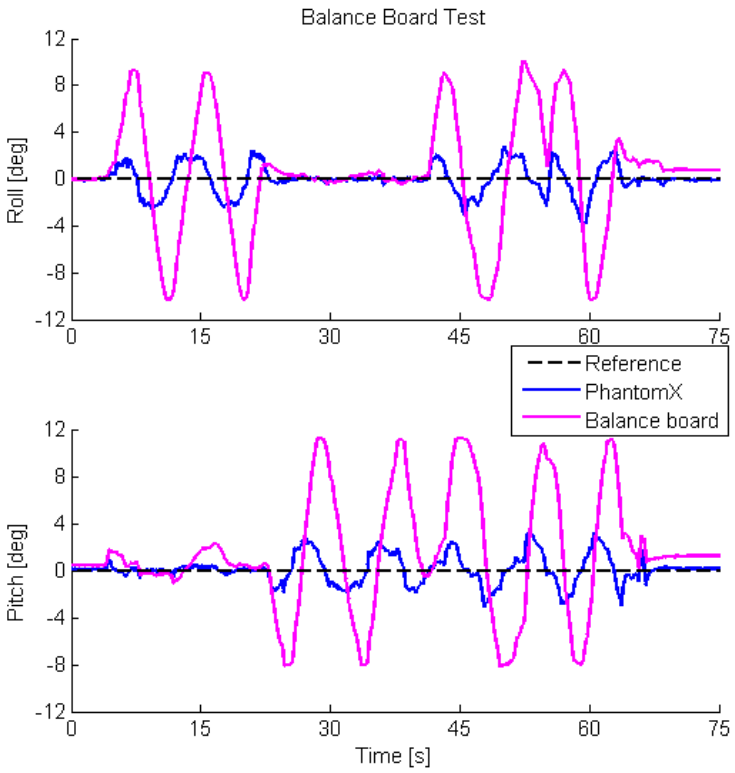


Figure 6.12: Roll and pitch response of the robot to tilting of balance board, as measured by Vicon [19]

A point to be noted is that since the posture controller's bandwidth causes a slight delay in responding to disturbances from the board, changes in the direction of the board's motion actually act to restore the robot's orientation. This restorative effect from the board, coupled with the delayed response of the controller, results in the robot's orientation response appearing to lead that of the balance board in Figure 6.12. This is likely also the reason that larger peak values of the robot's response are observed later in the test, when coupled roll and pitch motions are induced on the board, rather than sudden changes in one direction only, which have a greater restorative effect on the robot's orientation.

It is also worth pointing out that this was the only test in which the legs approached their saturation limits, and saturation conditions were only experienced momentarily, if at all. Therefore, the relatively large tracking time constant of 1 s (corresponding to a gain of $1/T_t = 1\ \mathrm{s}^{-1}$) was sufficient for these tests. However, it was mentioned in Section 6.1.4.1 that T_t determines how far outside the saturation limits the controller output can wind up and how fast it is restored after the saturating conditions cease. Thus, if the legs were to operate near saturation more often (as in the case of walking on highly uneven terrain), a smaller tracking time constant (corresponding to a higher gain $1/T_t$) would be preferable, to keep the posture control system sufficiently responsive. Of course, the gain cannot be made arbitrarily high, as a very potent response from the anti-windup mechanism could cause undesired behaviour in the leg motion, such as oscillations or instability. These factors are taken into consideration when selecting T_t in Section 7.2.5.

6.4 Conclusion

A Standing Posture Control algorithm was successfully implemented and tested on a low-cost, commercially available hexapod robot. The experimental results exhibited controller performance comparable to that achieved on more sophisticated, custom designed robotic platforms.

Moreover, the use of only one component of the foot force measurements to achieve this performance indicates that one-dimensional force sensors could be employed, reducing the costs and complexity of the platform even further. This is discussed in more detail in Section 8.3.2.

Some modifications could be made to the control system, such as including an integration term to eliminate, or at least reduce, steady state tracking error. However, the improvements in performance would likely be minor.

The next step in the development of this posture control system is to implement it on the PhantomX hexapod while walking on uneven terrain. This is precisely the topic to which the next chapter is dedicated.

Chapter 7: Walking Posture Control

In Chapter 6 the implementation of a posture controller for the specific case of the robot standing on all six legs was discussed. Experimental tests revealed that the implemented controller worked well to regulate posture while the robot stood on even, uneven, and dynamically varying terrain conditions. Of course, for the robot to be practically useful, it will be required to perform posture control not only while standing, but also while walking on uneven terrain.

Therefore, this chapter discusses the shortcomings of the Standing Posture Controller when walking is brought back into view, as well as modifications which were made to the control structure to address those drawbacks. The resulting "Walking Posture Control" system continues to distribute foot forces using simple strategies, even during walking. Furthermore, the new control system introduced in this study successfully achieves active compliance for legs in the swing phase, while still providing direct force control during the stance phase, with a simple switching strategy. It uses essentially the same overall Single Leg Force Control structure as the Standing Posture Controller, without requiring any modification to the underlying position control method utilized by the locomotion engine.

Furthermore, an artificial, uneven terrain setup developed for this study is introduced, followed by the presentation and discussion of results of experimental tests performed with the Walking Posture Control system.

7.1 Shortcomings of Standing Posture Control System for Walking

In this section the various components making up the Standing Posture Control structure will be reviewed, to determine the modifications required for posture control while walking.

7.1.1 Posture Calculation

Most of the posture calculation methods described in Section 5.2 were derived for general applicability, regardless of whether the robot was in motion or not. Therefore, the principles of the height and orientation calculations are still valid for use while walking.

However, a simplification was made for the implementation of height calculation for Standing Posture Control, wherein the positions of all six legs were always used, as they were always assumed to be in contact with the terrain (see Equation (6.1), Section 6.1.1). During walking, though, it is necessary to determine the ground contact status of each leg and use only the contact legs when calculating height, as per Equation (5.19) of Section 5.2.2. Specifics regarding the determination of the legs' contact statuses during practical implementation will be discussed in Section 7.2.1 below.

7.1.2 Virtual Model Controller

Since the virtual model controller takes estimations of the posture as its inputs and calculates only the required overall force and moments to be applied to the main body, this control structure is unaffected by the robot's motion and number of contact legs. Therefore, it can still be used as is, save for some retuning of the parameters to values more applicable for uneven terrain walking.

7.1.3 Foot Force Distribution

Basing the foot force distribution rule on an assumption of all six legs participating in the control of the robot, as was done for the Standing Posture Controller, is insufficient when the robot walks. It is thus more appropriate to consider the phases of the legs and utilize only the stance legs for distributing the required forces. Section 7.2.3 will describe the procedure employed to determine the leg phases, along with a revised foot force distribution methodology for walking with the PhantomX.

7.1.4 Single Leg Force Controller

While walking, the legs will be required to switch between force and position control regimes, with only some of the legs partaking in posture control at any given time. Therefore, the following factors are required to be taken into account for the force controller:

1. Since the gait engine commands will no longer be constant at all times, as was the case for the Standing Posture Controller, the saturation and anti-windup mechanisms are required to account for the actual position of the foot, rather than just the position adjustment value output by the force controller.
2. A smooth transition would be required when switching from direct force control, in the leg's stance phase, to motion control, in the swing phase. In particular, the position adjustment, Δz_i, present from the previous stance phase would need to be "wound down" gradually in the

swing phase, rather than performing a sudden, "hard reset" of this value to zero, as this would induce significant disturbances on the robot's motion.

3. The gait engine assumes the ground to be flat; thus, environmental modelling errors and uncertainties, in the form of unevenness in the terrain, would result in undesired contact between the ground and the foot of a swing leg, during motion. Hence, the interaction of the swing leg with the environment would need to be controlled, in the presence of external forces.
4. It would be preferable to achieve the differing interaction control schemes, in the different leg phases, with the same general force control structure, avoiding the complexities involved with introducing a completely different force controller to the system for the swing phase, such as:
    a. having to intelligently and seamlessly switch between the different force controllers;
    b. a significant increase in the number of parameters to be determined and tuned; and
    c. most likely increased computational load.

Points 3 and 4 above culminate in the idea that the ideal solution would lie in simple modifications, which could be made to the existing force control structure, allowing the legs to,

a) still operate under direct force control during the stance phase, but
b) undergo motion control from the gait engine during the swing phase, with adjustments from indirect force control to achieve compliant behaviour when interacting with the environment.

The resulting modifications made to the Single Leg Force Controller, in harmony with this philosophy, are presented in Section 7.2.4 below.

7.2 Modifications Made to Control Structure for Walking

Based on the considerations outlined in the previous section, relevant modifications were made to various components of the posture control structure. These modifications are discussed below, and the updated control strategies are presented.

7.2.1 Height Calculation

To account for the more general case where not all legs are in contact with the terrain, the height calculation in the `posture_control` node was updated according to Equation (5.19) of Section 5.2.2, which is repeated here:

$$h = -\frac{\sum_{i=1}^{n} p'_{i,z}}{n}, \qquad 1 \leq n \leq N. \tag{7.1}$$

The number of contact legs, n, to be used in this calculation is determined from the force information provided by the OptoForce sensors. For this, the `posture_control` node was also made to

subscribe to the force data published by the OptoForce driver, with the subscriber's callback saving the sensors' z-direction readings as the measured contact force values for the associated feet. Each foot's contact status is determined using a threshold force of 0.5 N, a value which was found to be small enough to quickly detect contact, while also being large enough to not misinterpret noise and measurement drift as contact.

7.2.2 Virtual Model Controller

As mentioned in Section 7.1.2 above, the virtual model control structure remains unchanged during walking. This ability to carry the high-level control structure forward, when making changes to the lower levels of the control system, distinctly highlights the advantages of using a modular, hierarchical control structure.

Updated VMC gains are presented in Section 7.2.5 below, while updates to the posture setpoints for experimental testing are discussed in Section 7.3.2.

7.2.3 Foot Force Distribution

As discussed in Section 7.1.3 above, the leg phases are required to be determined before being able to update the force distribution rule.

7.2.3.1 Foot Force Distribution Theory

While the most straightforward option would be to extract the leg phases from the internal workings of the gait engine, by modifying it to output this information, doing so would make the posture controller intrinsically linked to this particular gait engine. Since it would be preferable to have the control structure be as easily adaptable to a variety of locomotion engines as possible, as well as having the potential to be gait agnostic, it would be more suitable to estimate the phase information only from the standard outputs of the gait engine, i.e. the foot position commands.

In view of the fact that most standard gait engines command a constant vertical position during the stance phase, the z-position is sufficient to determine the leg's phase. In this study, a leg is considered to be in stance if the z-position command from the gait engine is below a threshold value of 0.001 m larger than the nominal standing height (a threshold value is used, rather than the exact nominal height, to account for small amounts of numerical noise which may be present in the position command signal). A threshold of just 1 mm may appear too small, as this is certainly within the margin of error that could be expected from the foot tip of the PhantomX, using AX-12A servos. However, this threshold value is applied only to the gait command signal, to determine when the gait engine has switched a leg into the stance phase. Therefore, the threshold value does not deal with the actual position of the foot at all, and so, for the determination of leg phase, it does not matter how accurately the foot is tracking the commanded position.

Another point to note is that although a derivative of the z-position command signal could also be used to infer the leg phase (identifying a stance phase when a zero derivative is obtained), in practical implementation the computation of a derivative as a backward difference would result in slight delays in determining the change of a phase. In addition, the possibility of obtaining a zero derivative at the top of the swing phase, before the leg changes direction, would also have to be accounted for. Since the former option (using a threshold value on the position command signal) is both simpler and, in most cases, more accurate, it was decided to be used for this study.

7.2.3.1.1 Tripod Gait

Moving on to the actual distribution of foot forces to the stance legs, the reader is reminded that the force-torque equilibrium equations can only be solved directly when $n = 3$ (see Section 6.1.3.1), while force distribution cases involving more contact legs are usually solved using an optimization method. This is still not desirable in the current study, on account of the computational load this will place on each iteration of the VMC loop. However, in this particular case, since the `hexapod_ros` gait engine currently only provides the tripod gait, the only scenarios which require force distribution involve either three contact legs, which can be solved directly, or six contact legs, a solution to which has been developed for the Standing Posture Controller.

7.2.3.2 Foot Force Distribution Implementation

In the ROS software, the outputs of the gait engine were set to be published from the `hexapod_controller` node, allowing the `posture_control` node to subscribe to them and compute the leg phases during the main control callback. For the scenario of six stance legs, the same force distribution rule as the Standing Posture Controller is employed. On the other hand, when the number of legs in stance phase is found to be three, the force-torque equilibrium equation, in matrix form, is solved as (see Equation (6.34) of Section 6.1.3.1)

$$\boldsymbol{F}_{i,z} = \boldsymbol{P}^{-1}\boldsymbol{F}_{\boldsymbol{VMC}}, \tag{7.2}$$

for the relevant tripod of stance legs, using the functions built into the Eigen library for C++. In particular, the `colPivHouseholderQr` method (which carries out Householder rank-revealing QR decomposition of a matrix with column-pivoting) is used to solve the linear system. This method provides a good compromise between solution speed and accuracy for small-to-medium sized matrices, without imposing any requirements on the structure of the matrix [106]. The force setpoints of all swing legs are set to zero, the reason for which will become more apparent in the next section. As before, all six force setpoints are published by the `posture_control` node, for use by the Single Leg Force Controllers.

Some situations do arise when stance phase calculations do not coincide precisely with the tripod model. For example, when changing between stance tripod sets, the phase calculation sometimes yields a momentary result of less than three stance legs. This occurs regardless of how small the threshold for stance is set to, indicating that the primary cause lies in the method used by the gait sequencer to generate foot trajectories. More specifically, in this gait engine the gait commands are calculated purely based on sinusoidal trajectories, without having any dedicated phase switching mechanism. As a result, numerical and round-off errors can sometimes cause some legs in a stance tripod to start lifting off just before the legs in the swing tripod have touched down. This is most likely also the cause of the slight drift over time in the yaw angle, or heading, that can sometimes be observed when the robot is commanded to walk in a straight line. A similar situation occurs just as the robot starts walking, or just before it stops, in which case the number of stance legs briefly registers in the range $3 < n < 6$.

Therefore, for the purposes of foot force distribution, these anomalies are accounted for by a series of checks on the number of stance legs. If n is found to be smaller than three, i.e. $n < 3$, the previous set of legs used for force distribution is used once more. On the other hand, if $3 < n \leq 6$, the full set of all six legs is used for force distribution. Of course, in the event that $n = 3$, the particular tripod of legs found to be in contact is used for force distribution. These occasional situations only occur momentarily, for a single time step, and thus do not have any notable, negative impact on the performance of the posture controller.

It may appear as though the use of a tripod gait model destroys the point of ensuring that the posture controller implementation is not inextricably linked to the gait engine in use. However, the assumptions discussed above are based on high-level concepts and outputs of the gait engine. As such, they do not bind the leg phase calculations or force distribution to this particular implementation of a gait engine. Since the leg phase information is inferred from the gait output, these force distribution calculations could hypothetically be performed using any hexapod gait engine providing a tripod gait, without having to delve into its internal workings to determine how it computes and stores phase information. Their usage is also likely to remain unaffected by a change in the implementation of the existing gait engine. Furthermore, if different gaits are provided by the engine, the force distribution method needs to just be updated to handle different numbers of contact legs, after which the inferred leg phase information can still be used to distribute the forces accordingly. Hence, despite the simplifications made in the current implementation, the potential for broader application to other gait engines, along with gait agnosticism, is still prevalent in this posture control system.

7.2.4 Single Leg Force Controller

The various modifications made to the Single Leg Force Controller structure, to address the issues outlined in Section 7.1.4, will first be discussed individually in this section, before presenting the final, designed controller and its implementation.

7.2.4.1 Single Leg Force Controller Modifications

This section individually describes the various modifications made to the Single Leg Force Controller for use during walking.

7.2.4.1.1 Saturation and Anti-Windup

As discussed in Section 7.1.4 above, the first modification to make to the Single Leg Force Control structure is to move the artificial saturation and anti-windup mechanisms further downstream when compared to that used in the Standing Posture Controller (see Figure 6.1), to account for the position command from the gait engine. A block diagram of the force controller for a single leg i, incorporating this change, can be seen in Figure 7.1 below.

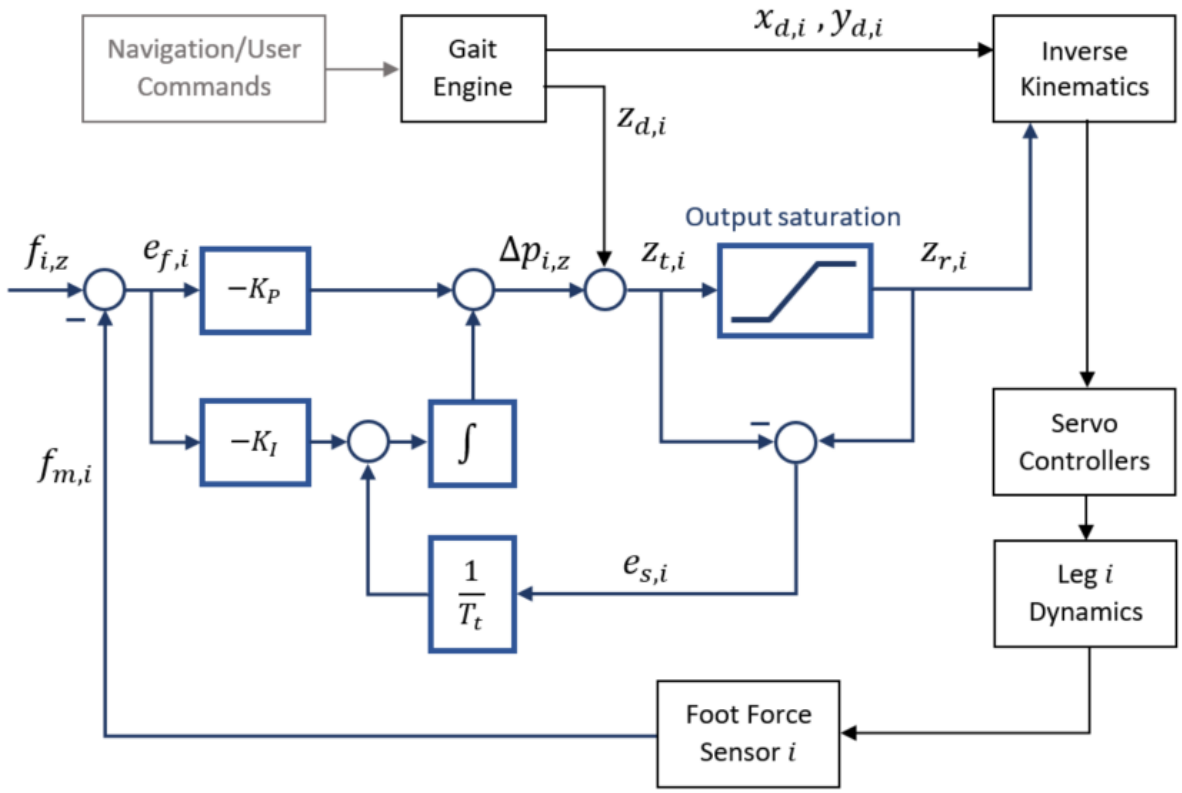


Figure 7.1: Block diagram of Foot Force Controller for a single leg i, with saturation and anti-windup mechanisms moved further downstream

Rather than saturating the force controller output, $\Delta p_{i,z}$, before adding it to the gait command, this output is added directly into the gait command to yield a temporary reference position, $z_{t,i}$. That is,

$$z_{t,i} = z_{d,i} + \Delta p_{i,z}. \tag{7.3}$$

This temporary reference position is then fed through the artificial saturation model to obtain the actual, adjusted reference position, $z_{r,i}$. As a result, the tracking error, $e_{s,i}$, is computed as

$$e_{s,i} = \left(z_{r,i} - z_{t,i}\right). \tag{7.4}$$

During normal operation, the result of this computation is equivalent to that of Equation (6.39) in the case of the Standing Posture Controller, and produces effectively the same outcome during saturation. However, it should be kept in mind that feeding back $e_{s,i}$, as calculated in Equation (7.4) above, does introduce $z_{d,i}$ as an additional, almost hidden, input to the force controller's integrator. This can be seen more clearly when analysing the integrator input:

$$\begin{aligned} -K_I e_{f,i} + \frac{1}{T_t} e_{s,i} &= -K_I e_{f,i} + \frac{1}{T_t}\left(z_{r,i} - z_{t,i}\right) \\ &= -K_I e_{f,i} + \frac{1}{T_t}\left(z_{r,i} - z_{d,i} - \Delta p_{i,z}\right). \end{aligned} \tag{7.5}$$

Of course, when the output is within its saturation limits, the expression in brackets in this equation evaluates to zero. However, during saturation, this expression does act as an input to the integrator, affecting the controller output, $\Delta p_{i,z}$, to drive the integrator input to zero.

Ordinarily, especially when the leg is in the stance phase, the gait command, $z_{d,i}$, is well within the saturation limit, z_{lim}, thus $\left|z_{d,i}\right| < \left|z_{r,i}\right|$ (since $z_{r,i} = z_{lim}$ at saturation). Therefore, during saturation, the effect of $z_{d,i}$ is, in essence, "cancelled out" in the difference equation $z_{r,i} - z_{d,i}$ (which appears when $z_{r,i} - z_{t,i}$ is expanded in Equation (7.5)), resulting in only $\Delta p_{i,z}$ contributing to the excess between $z_{t,i}$ and $z_{r,i}$. Feeding back $e_{s,i}$ into the integrator thus just acts to wind-down the output, $\Delta p_{i,z}$, which is too large during saturation anyway, exactly as in the case of the Standing Posture Controller.

However, considering a hypothetical situation in which $\Delta p_{i,z} = 0$, but $z_{d,i}$ is made large enough to exceed the saturation limit (of course this could only happen if the leg is in the swing phase, as $z_{d,i}$ is constant during stance), i.e. $\left|z_{d,i}\right| > \left|z_{r,i}\right|$, the tracking error then becomes

$$e_{s,i} = z_{r,i} - z_{d,i} = z_{lim} - z_{d,i}. \tag{7.6}$$

Clearly, the only contributor to the excess of $z_{t,i}$ above z_{lim} is the gait command. Feeding back this tracking error effectively makes $z_{d,i}$ the only input signal affecting the integrator ($\Delta p_{i,z} = 0$ before saturation naturally implies that $e_{f,i} = 0$), and as a result the controller is forced to output a $\Delta p_{i,z}$ value equal in magnitude, but opposite in sign, to the amount by which $z_{d,i}$ exceeds z_{lim}, so that $e_{s,i}$ will be driven to zero. With both $e_{s,i}$ and $e_{f,i}$ being zero, there is now no input to reduce $\Delta p_{i,z}$ back to zero once the gait command is brought back within the saturation limits. The result is a constant

deviation of $\Delta p_{i,z}$ between the actual position of the leg and that commanded by the gait engine, while the leg moves through space. This suggests that the wind-down mechanism for the controller output, proposed in Section 7.1.4, should operate throughout the entire swing phase of the leg, rather than only when switching from stance to swing.

It is prudent to point out that while the above consideration is mostly academic (as the gait engine generally does not command positions outside of the saturation limits), changes in the implementation of the gait engine, or physical changes to the robot that affect the saturation limits, could potentially lead to this issue becoming a reality. Hence, the controller design should be robust to such an eventuality. Furthermore, irrespective of whether this particular issue does arise, the considerations above do emphasize the importance of a vital feature necessary in the control system, especially if the existing force controller is desired to be utilized for the swing legs as well.

7.2.4.1.2 Controller Output Feedback

In order to address the issues pointed out in the above section, as well as those discussed previously, including the need for compliance control with minimal changes to the overall control structure, a modified force controller for a swing leg i is introduced, in block diagram form, in Figure 7.2. For the sake of simplicity and clarity, the saturation and anti-windup mechanisms are not indicated in the diagram at this stage, focussing only on the actual force controller from the input force to the output position adjustment.

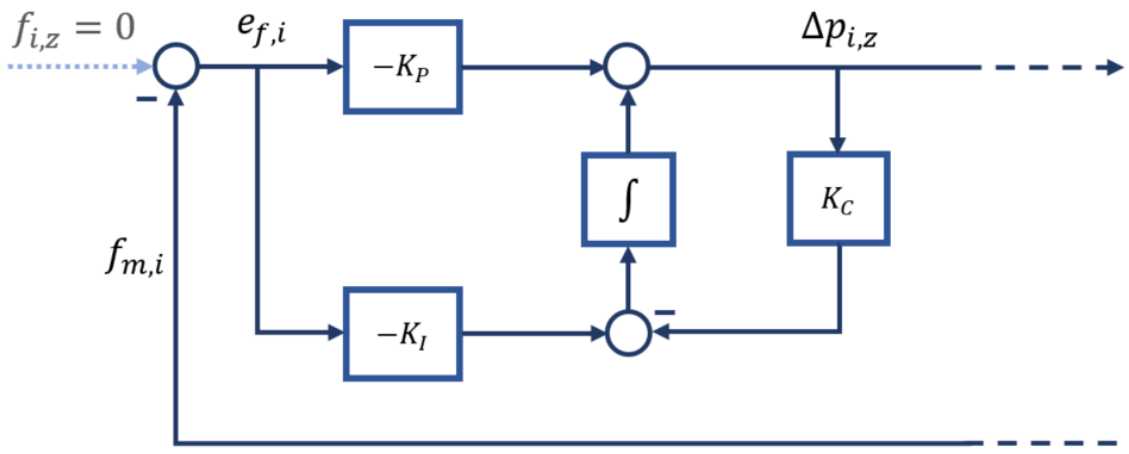


Figure 7.2: Block diagram of modified Foot Force Control structure for a single, swing leg i, providing output wind-down and active compliance during interaction

In the block diagram of Figure 7.2, it can be seen that the only significant modification made to the controller, as compared to the block diagram of Figure 6.1, is the addition of a negative feedback path from the controller output, $\Delta p_{i,z}$, to the input of the integrator, through a gain K_C. While this

modification may initially appear to be deceptively simple, it does, in fact, address many of the requirements identified in Section 7.1.4.

The first, and most obvious, advantage of closing this controller output feedback loop is that, in free space with no force error (or leg saturation), the controller output is driven to zero, achieving the crucial wind-down effect required during the swing phase. Given that, in free space, no contact force is exerted onto the foot-tip, having a force setpoint of zero is necessary to ensure zero force error during swing. This explains the decision to set the force setpoint to zero for all swing legs in the foot force distribution process, as mentioned in the preceding section.

Although this works to set the force error, $e_{f,i}$, to zero in theory, it is important to consider that, in practice, small magnitudes of measurement noise do exist in the force feedback signal, creating small force errors even when no contact is exerted on the foot. Left unchecked, these small errors would be integrated over time and eventually result in a large enough $\Delta p_{i,z}$ value to create a noticeable drift in the position of the foot. Here, once again, the feedback loop of the controller output comes in handy to ensure that undesired force controller outputs do not linger in the reference position of the foot, during the swing phase.

In order to observe the other advantage of this controller formulation, it is enlightening to consider the situation of a contact force being exerted onto the foot (for example, when the swing leg contacts the terrain earlier than expected). Since the force setpoint is zero during the swing phase (and hence indicated by a light, dotted arrow in Figure 7.2), the negative of the measured force is transmitted directly into the force controller as an input. As it goes through the negative controller gains, the effect is comparable to the feedback of force used in the indirect force controller structures discussed in Section 3.4.2. The resulting control action is described mathematically as follows:

$$\Delta p_{i,z} = -K_P e_{f,i} + \int_0^t \left(-K_I e_{f,i} - K_C \Delta p_{i,z}\right) d\tau, \tag{7.7}$$

and since $e_{f,i} = \left(f_{i,z} - f_{m,i}\right) = \left(0 - f_{m,i}\right) = -f_{m,i}$, it follows that

$$\Delta p_{i,z} = K_P f_{m,i} + \int_0^t \left(K_I f_{m,i} - K_C \Delta p_{i,z}\right) d\tau. \tag{7.8}$$

At steady state, the controller output remains constant and hence the integrator input is zero.

Therefore,

$$K_I f_{m,i} - K_C \Delta p_{i,z} = 0$$
$$\therefore f_{m,i} = \frac{K_C}{K_I} \Delta p_{i,z}. \tag{7.9}$$

The result of Equation (7.9) demonstrates that, at steady state, the new force controller allows a leg in the swing phase to act as a generalized spring, with stiffness K_C/K_I, in respect of the applied contact force. This is exactly the steady state behaviour observed in manipulators under compliance control, as described in [68], indicating that the direct force controller has successfully been modified to provide active compliance to a leg in the swing phase, without having to modify the underlying motion, or position, control method utilized by the locomotion engine.

Although the dynamic behaviour of the leg may differ slightly from that of a manipulator under compliance control, the overall effect is still as desired, in that the reference position of a swing leg is adjusted to retract it in the presence of an external force, so as to limit the disturbance induced onto the robot's body.

An additional observation from Equation (7.9) is that, at steady state, a non-zero force will still be present if the foot of a leg in swing phase is in contact with the terrain. Given that the setpoint is zero, a non-zero force at steady state means the controller does not precisely regulate force to the desired value and is thus definitely no longer a direct force controller. A non-zero force in steady state further implies that, while active compliance does attenuate the disturbance to the body, it cannot be eliminated completely.

This points to a requirement of high compliance, or low stiffness, in the behaviour of the leg, to reduce disturbances as much as possible. More specifically, if Equation (7.9) is rearranged as

$$\Delta p_{i,z} = \frac{K_I}{K_C} f_{m,i}, \tag{7.10}$$

it can be seen that K_C should be as small as possible for a given contact force, to allow the leg's position to be retracted as much as possible, thus inducing as little motion onto the main body as possible when the swing leg interacts with the ground. Alternatively, Equation (7.9) can be seen to suggest that the smaller the value of K_C, the smaller the disturbance force exerted for a given steady state position of the leg. It is thus clear that a small K_C value is desired to achieve highly compliant behaviour. Since K_C dictates the amount of compliance (assuming that the value of K_I is governed by the direct force controller requirements), it will be referred to as the *compliance gain* hereafter.

Unfortunately, the compliance gain cannot be made arbitrarily small to achieve unlimited compliance, as the smaller it is made the more the wind-down action in free space is reduced. In other words, if K_C is too small, the rate at which $\Delta p_{i,z}$ winds down when the leg is unforced would become impractically slow. Therefore, a compromise needs to be attained between compliance and wind-down speed when tuning the controller.

Now that the modified control structure has been analysed for its applicability to a leg in the swing phase, it is useful to reintroduce the saturation and anti-windup mechanisms, as well as consider the task of switching between direct force control and active compliance control. While the saturation and anti-windup mechanisms can be incorporated into the modified controller just as described earlier this section, switching control regimes requires a further addition to the controller.

In particular, since the only major modification to the force control structure during the swing phase is the output feedback loop, switching between control strategies essentially entails enabling or disabling this feedback, depending on the current phase of the leg. This can be simply accomplished by multiplying the feedback through an additional gain of either 1, for the swing phase, or 0, for the stance phase. Since the leg phases are being determined for force distribution anyway, this information is available to the force controller. Thus, the use of a gain as described here presents a simple method of mathematically switching control regimes. Of course, the other necessity when changing from stance to swing is to set the force setpoint to zero, but this has already been taken care of in the foot force distribution procedure.

7.2.4.2 Single Leg Force Controller Design

All the modifications discussed above are brought together to form the new Single Leg Force Controller introduced in this study, which achieves both direct force control during the stance phase and compliance control during the swing phase, with a simple switching strategy. This controller is presented in block diagram form in Figure 7.3 for a single leg, i, in which the switching gain, $K_{S,i}$, can be seen multiplied by the feedback of the controller output, $\Delta p_{i,z}$, to yield the *compliance error*, $e_{c,i}$.

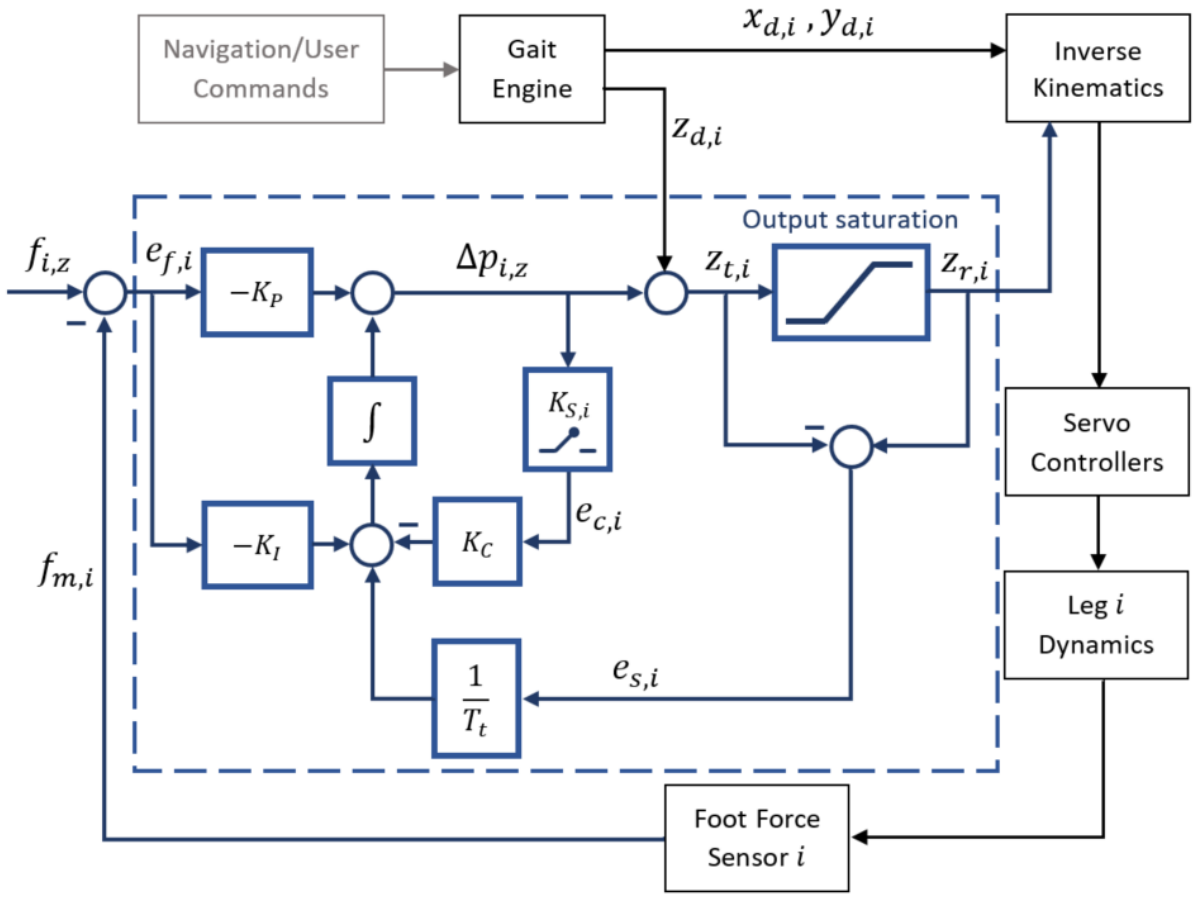


Figure 7.3: Block diagram of Foot Force Controller for single leg i, *for both stance (direct force control) and swing (compliance control) phases during walking*

The control law describing the system illustrated in Figure 7.3 is represented mathematically as

$$\Delta p_{i,z} = -K_P e_{f,i} + \int_0^t \left(-K_I e_{f,i} - K_C e_{c,i} + \frac{1}{T_t} e_{s,i}\right) d\tau, \tag{7.11}$$

where $e_{s,i}$ is as defined in Equation (7.4) above, and

$$e_{c,i} = K_{S,i} \Delta p_{i,z}. \tag{7.12}$$

Given that

$$K_{S,i} = \begin{cases} 1, & phase_i \equiv swing \\ 0, & phase_i \equiv stance \end{cases}, \tag{7.13}$$

it follows that

$$e_{c,i} = K_{S,i} \Delta p_{i,z} = \begin{cases} \Delta p_{i,z}, & phase_i \equiv swing \\ 0, & phase_i \equiv stance \end{cases}. \tag{7.14}$$

An advantage of having the switching mechanism operate through the integrator is the "filtering" effect that integration has on the discontinuities in the signal. In the deliberations of Section 7.1.4, it

was alluded to that performing a hard switch between control regimes, by suddenly resetting or introducing signals, would generally produce undesirable behaviour, such as abrupt changes in leg position. Of course, the switching gain does indeed result in the $\Delta p_{i,z}$ feedback loop being suddenly enabled or disabled when changing phase, creating discontinuities in the form of steps in the signal. However, integration over such discontinuities yields a continuous integral. Thus, feeding this discontinuous feedback signal into the integrator input results in a continuous signal being observed in the controller output, effectively providing a smoothed transition from one control regime to another.

Once more, it is necessary to discretize the updated control equation for digital implementation. Since the proportional action of the controller has remained unchanged (see $P_i(t_k)$ in Equation (6.48)), the discretization process of only the integral part will be presented. From Equations (7.11) and (7.12), the integral action of the continuous system, $I_i(t)$, takes the form

$$I_i(t) = \int_0^t \left(-K_I e_{f,i}(\tau) - K_C e_{c,i}(\tau) + \frac{1}{T_t} e_{s,i}(\tau) \right) d\tau, \tag{7.15}$$

where

$$e_{c,i}(\tau) = K_{S,i}(\tau) \Delta p_{i,z}(\tau). \tag{7.16}$$

Differentiation of Equation (7.15) leads to

$$\frac{dI_i(t)}{dt} = -K_I e_{f,i}(t) - K_C e_{c,i}(t) + \frac{1}{T_t} e_{s,i}(t). \tag{7.17}$$

As before, using a forward difference to approximate the derivative at a particular sampling instant, t_k, results in

$$\frac{I_i(t_{k+1}) - I_i(t_k)}{t_{k+1} - t_k} = -K_I e_{f,i}(t_k) - K_C e_{c,i}(t_k) + \frac{1}{T_t} e_{s,i}(t_k). \tag{7.18}$$

Letting $\Delta t_{k+1} = (t_{k+1} - t_k)$ and rearranging Equation (7.18) gives the following discrete equation for the integral action:

$$I_i(t_{k+1}) = I_i(t_k) + \Delta t_{k+1} \left(-K_I e_{f,i}(t_k) - K_C e_{c,i}(t_k) + \frac{1}{T_t} e_{s,i}(t_k) \right), \tag{7.19}$$

with

$$e_{c,i}(t_k) = K_{S,i}(t_k) \Delta p_{i,z}(t_k). \tag{7.20}$$

A noteworthy observation can be made from Equations (7.19) and (7.20): the use of a forward difference to approximate the derivative allows for the computation of the control law at the current time instant, despite the recursive nature of Equation (7.11), in which the current controller output is required in order to calculate the current controller output itself! This is not an issue in the discretized equations above, as the controller output from the previous sampling instant is used to compute the integral action for the current sampling instant. If a backward difference were used for approximating the derivative in Equation (7.17), however, the signal values at the current sampling instant would be propagated backward over the time difference, Δt_k, resulting in the computation of $\Delta p_{i,z}(t_k)$ requiring the value of $\Delta p_{i,z}(t_k)$ itself – clearly a practically unachievable computation.

7.2.4.3 Single Leg Force Controller Implementation

Before presenting the updated digital computation algorithm, it is apt to discuss some additional modifications made to the control system, to overcome issues arising from its practical implementation. When performing tests with the robot running the updated force controller as described above, peculiar behaviour was observed on occasion, especially when walking on uneven terrain or over obstacles. During these occurrences, one, or some, of the legs would go into a state of indefinite oscillation between the upper and lower saturation limits, rendering the robot essentially uncontrollable. This was potentially indicative of a condition of instability in the closed-loop response of the leg.

Upon further investigation and inspection of the various signals acting as inputs and outputs of the control system, it was found that, when this phenomenon occurred, the force controller output, $\Delta p_{i,z}$, was indeed exhibiting unstable behaviour, oscillating between positive and negative values with an exponentially growing magnitude. This observation clearly pointed to undesirable negative feedback action, causing instability in the controller.

Since the instability only manifested in stance legs, it was definitely not being caused by the output feedback loop used for compliance and wind-down in the swing phase, leaving only the tracking error feedback signal as the possible culprit. Scrutiny of the controller block diagram (Figure 7.3) and Equations (7.3), (7.4), (7.5), and (7.11) reveals that, during conditions of saturation, there does indeed exist a path of negative feedback of $\Delta p_{i,z}$ in the control structure, through the integrator input. As a result, large values of $e_{s,i}$, especially when integrated over time, could, in theory, lead to large controller outputs in the opposite direction – so large in fact, that they cause larger magnitudes of $e_{s,i}$ at the other saturation limit. Subsequently, the effect would be the same once again, but more pronounced, and this cycle would continue, causing larger and larger magnitudes of the controller output being “thrown” back and forth. Naturally, the commanded position of the leg would oscillate

between the saturation limits, causing the legs to continually be raised and lowered over the entire range of motion.

Now, in a continuous system, the appropriate selection of gains would prevent such an eventuality, with the controller output never reaching large enough magnitudes to experience exponential, "runaway" growth from negative feedback. From a theoretical perspective, this would be tantamount to the gains being selected so as to keep the real parts of all the closed-loop poles negative, ensuring stability [94], [97]. Similarly, in a digital system, with a (relatively) constant sampling frequency, the gains would be selected for that particular frequency, limiting the feedback magnitudes for stability. However, in Section 6.1.4.2 the issue of variable controller loop frequency in the `hexapod_controller` node was illuminated. The consequence of this to the force controller is that, if the control frequency drops significantly, the time over which error signals are integrated is significantly larger than expected, which effectively produces the same outcome as an exceedingly large gain in the integral action.

In the particular case of the current force controller implementation, the onset of this unstable occurrence is initiated by a large force error on a specific leg, causing a large controller output beyond the saturation limit. Accordingly, the leg is commanded to move to the limiting position. This usually means that a substantial change in position is required, causing the interpolation loop of the servo driver to execute over a long time, blocking the main control loop, as described previously. At the next sampling instant, the (possibly small) tracking error is integrated over the large time difference, leading to the leg being commanded to move across its entire range of motion at once, to the other saturation point. Of course, the time taken by the interpolation loop is then even larger and the cycle continues, resulting in the controller output quickly becoming unbounded.

Thus, the emergence of instability in this controller implementation can be attributed to a combination of factors. The two main problems identified from the above considerations are:

1. Enormous force errors, arising from large force setpoints; and
2. Feedback of immense values of the tracking error, further amplified by integration over large time periods arising from drops in the controller frequency.

A possible remedy to the first problem would be to "detune" the VMC gains, resulting in smaller forces being demanded of the legs. However, this would impact on overall performance, even in cases when smaller posture errors are measured and the resulting force setpoints would not have led to instability. Therefore, the solution opted for in this study was to introduce force setpoint limits in the foot force distribution rule. That way, the desired dynamic behaviour would still be achieved during normal

operation, with only extreme cases being affected by setpoint limitation. Of course, this setpoint limitation does not guarantee that unstable behaviour would never arise; however, it does benefit the controller by reducing the occurrences of extremely large controller outputs being generated. The second of the two problems, on the other hand, more directly influences the unstable growth of the controller output, and it is thus imperative for it to be counteracted.

Since modification of the entire locomotion generation engine is outside of the scope of this study, the problem of inconsistent controller frequency will need to be "circumnavigated," so to speak, for the time being. Thus, an option for limiting the growth of the controller output from feedback is to introduce bounds on the magnitude of the tracking error signal. This would keep the integration, and hence the controller output, in check, not allowing it to grow unstably. For this reason, limiting bounds were applied for both positive and negative magnitudes of $e_{s,i}$.

Technically, limiting the value of $e_{s,i}$ limits the anti-windup capability of the control system. The tracking gain itself becomes saturated when reaching its limit, allowing large enough force errors to continue winding up the controller output further. However, in a practical situation, the continued application of such large forces to the foot would most likely result in damage to the force sensors or even the robot itself. In fact, such a force would not even occur when the robot walks on most terrains, unless the terrain was dynamically changing in an extreme manner, in which case the robot would likely be lifted or overturned and posture control would not be possible anyway. Obviously, such conditions or applications of the robot would be avoided as far as possible. In a real-world implementation, rather than attempting to prevent such unlikely eventualities, it is much more critical to ensure stability during operation.

It is also important to keep in mind that this limitation of $e_{s,i}$ only forms part of the controller design for this particular implementation. This feature would not be necessary in an implementation where the time periods between controller loop iterations are kept relatively consistent, or real-time constraints are enforced.

It should also be noted that since there is kinematic dependency between the foot force and position, an anti-windup mechanism, for saturation at the kinematic limits, could cause instability even in a system with a consistent execution frequency. This is particularly true in the case of improper gain selection (as mentioned earlier), which could also result in the inner control loops having slower bandwidths than outer control loops. However, in this case, the drops in controller frequency from the "blocking interpolation loop" of the locomotion engine, rather than poor selection of controller gains, are the cause of excessively large signals and an inappropriate inner control loop bandwidth.

Accordingly, the modifications described above are necessary to avoid instability in the current implementation. Hence, for this study, the mathematical algorithm which digitally implements the modified force controller for a single leg i becomes:

$$
\begin{aligned}
&\Delta t_k = t_k - t_{k-1};\\
&I_i(t_k) = I_i(t_{k-1}) + \Delta t_k \left(-K_I e_{f,i}(t_{k-1}) - K_C e_{c,i}(t_{k-1}) + \frac{1}{T_t} e_{s,i}(t_{k-1}) \right);\\
&e_{f,i}(t_k) = \left(f_{i,z}(t_k) - f_{m,i}(t_k) \right);\\
&P_i(t_k) = -K_I e_{f,i}(t_k);\\
&\Delta p_{i,z}(t_k) = P_i(t_k) + I_i(t_k);\\
&z_{t,i}(t_k) = z_{d,i}(t_k) + \Delta p_{i,z}(t_k);\\
&z_{r,i}(t_k) = \text{simulateSaturation}\left(z_{t,i}(t_k) \right);\\
&e_{s,i}(t_k) = \text{limitTrackingError}\left(z_{r,i}(t_k) - z_{t,i}(t_k) \right);\\
&K_{S,i}(t_k) = \begin{cases} 1, & phase_i(t_k) \equiv swing \\ 0, & phase_i(t_k) \equiv stance \end{cases};\\
&e_{c,i}(t_k) = K_{S,i}(t_k) \Delta p_{i,z}(t_k);
\end{aligned}
\tag{7.21}
$$

where limitTrackingError() represents a function that applies the limiting values to the tracking error, yielding a bounded $e_{s,i}(t_k)$.

The `ForceController` class in the `hexapod_controller` package was updated to execute the algorithm described above, independently for each leg. A subscriber was also created to receive the leg phase information from the force distribution process. Accordingly, the `posture_control` node was also updated with two modifications, one to apply the force setpoint limits before publishing them and the other to publish the leg phase information for each leg together with the force setpoints.

In the section that follows, the updated Foot Force Controllers are shown integrated into the greater robotic control structure.

7.2.5 Overall Control Structure

While the individual components of the control system have been modified to handle walking, the overall control structure still takes the same general form as the Standing Posture Controller. An updated block diagram of this structure is illustrated in Figure 7.4 below, with only a few significant changes visible, as compared to Figure 6.2 for the Standing Posture Control system. In particular, the foot force measurements are now also fed into the Posture Calculation block, where they are used to assist the height calculation. In addition, the z-position command from the gait engine also gets fed into the Foot Force Distribution block, so that the leg phases can be determined. Subsequently, the

leg phase information is transmitted from the Foot Force Distribution block to the stack of Foot Force Controllers, along with their setpoints.

Furthermore, the outputs of the main control blocks have been indicated on the diagram, for clarity. It should be noted that the individual gait engine outputs ($x_{d,i}$, $y_{d,i}$, and $z_{d,i}$) and Foot Force Controller output ($z_{r,i}$) have been replaced by their vector equivalents, $\boldsymbol{x}_d$, $\boldsymbol{y}_d$, $\boldsymbol{z}_d$, and $\boldsymbol{z}_r$, which include the relevant signals for all six legs of the robot.

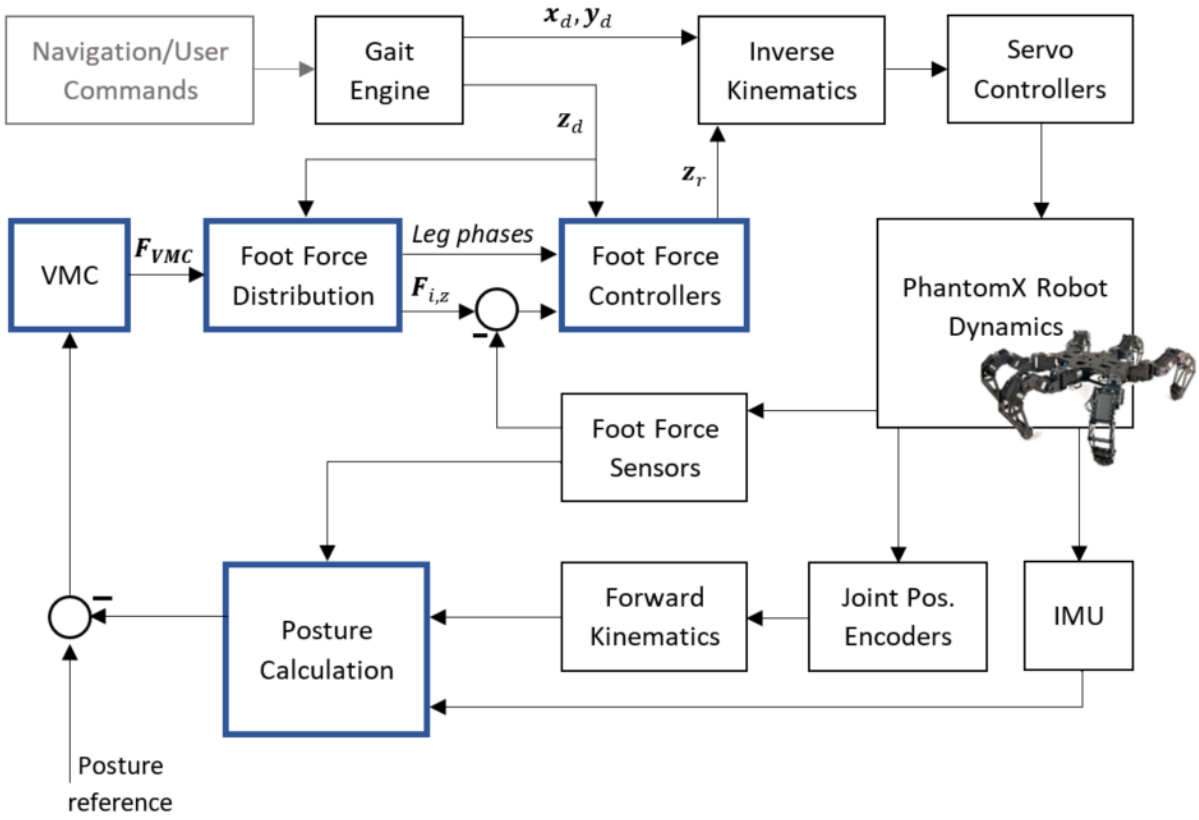


Figure 7.4: High-level block diagram depicting overall control structure on hexapod robot, incorporating Walking Posture Control system

Existing controller gains were also updated for the walking case. For the VMC and main force controller gains, the posture response while walking on both flat and uneven terrain (emulated using a variety of obstacles and contoured surfaces, over which the robot was made to walk) were used to adjust the gains, rather than step responses as in the case of the Standing Posture Controller. To update the tracking time constant, T_t, saturating scenarios were observed both while walking the robot and while manually applying a force to a foot until its position became saturated. The parameter was adjusted so as to ensure that the controller output could be restored quickly enough when the saturating force was removed, while also not resulting in undesirably large values of the tracking error, which cause oscillatory motion and, as described earlier, more easily lead to instability.

As mentioned previously, the selection of the compliance gain, K_C, requires a compromise between the virtual stiffness of the swing leg and the wind-down speed of the force controller output. Therefore, with respect to the posture response while walking, the effects of choosing the gain for stiffer leg behaviour were compared to those for a more compliant swing leg motion. Additionally, the wind-down speed of the reference position, $z_{r,i}$, was examined for various values of K_C, when switching a leg from the stance phase to the swing phase while walking on uneven terrain. Values of K_C that could return the reference position to the same trajectory as that commanded by the gait engine ($z_{d,i}$), before the end of the swing phase, were favoured over values which resulted in slower wind-down. The final value that was used provided a fairly good compromise between these criteria.

Limiting values for the force setpoints were based on typical values demanded by the force distribution, as well as measurements of the actual contact forces experienced by the legs while walking on uneven terrain, with posture control enabled. Similarly, tracking error limits were selected by taking into account conservative values for the lowest control loop frequency which can typically be expected, as well as the largest magnitudes of integral action which can be handled before instability becomes a possibility. The limiting values selected for these signals successfully prevented the occurrence of unstable behaviour for the duration of the study.

A final note should be made on the selection of the N_f parameter, which determines the frequency of the derivative filter used in the virtual model controller. From Equation (6.9) in Section 6.1.2.1, the controller parameters were related to the filter time constant, τ_c, as

$$\tau_c = \frac{T_d}{N_f}. \tag{7.22}$$

Since the time constant is the inverse of the filter cut-off frequency, that is [96]

$$\omega_c = \frac{1}{\tau_c}, \tag{7.23}$$

and that $T_d = K_d/K_p$, it follows that

$$\omega_c = \frac{K_p N_f}{K_d} \text{ [rad/s]}. \tag{7.24}$$

This value must be divided by 2π rad to obtain the frequency in units of Hertz. Thus,

$$f_c = \frac{K_p N_f}{2\pi K_d} \text{ [Hz]}, \tag{7.25}$$

which shows how N_f determines the filter cut-off frequency, for given values of proportional and damping gains.

In the Walking Posture Controller, the filter was used mainly to limit the influence of noise occurring from external disturbances and anomalies, as most of the measurement noise in the signals already gets filtered by the sensor DAQs, as well as by the averaging operation in the case of the height calculation. Frequency response function (FRF) plots of posture estimate signals, recorded while walking on both flat and uneven terrain, confirmed that the only significant magnitudes in the signal were of relatively low frequencies, with most of the high frequency noise already having been appreciably attenuated.

The various parameter values selected for walking posture control are summarized in Tables 7.1 to 7.3 below. Table 7.1 also includes the derivative filter cut-off frequencies corresponding to the VMC parameter values selected.

Table 7.1: Parameter values used for VMC in Walking Posture Control system

Posture Component	Symbol (X)	Parameter Values			
		K_X	C_X	$N_{f,X}$	$f_{c,X}$
Height	z	1000.0 N/m	30.0 N·s/m	7.5 rad	39.79 Hz
Roll	ϕ	90.0 Nm/rad	1.5 Nm·s/rad	4.5 rad	42.97 Hz
Pitch	θ	90.0 Nm/rad	1.5 Nm·s/rad	4.5 rad	42.97 Hz

Table 7.2: Parameter values used for Single Leg Force Controller in Walking Posture Control system

Parameter	Value
K_P	2.0e-4 m/N
K_I	1.0e-2 m/(N·s)
K_C	0.3 s^{-1}
T_t	0.1 s

Table 7.3: Limiting values used to prevent instability in Single Leg Force Controller

Signal	Upper Limit	Lower Limit
Force setpoint, $f_{i,z}$	45.0 N	-5.0 N
Tracking error, $e_{s,i}$	0.015 m	-0.015 m

7.3 Experimental Setup and Testing

While the Standing Posture Controller was evaluated primarily with step tests, the performance of the Walking Posture Controller system is, of course, required to be evaluated while the robot walks. Since the aim of the control system is to improve posture on uneven terrain, it was necessary to test its performance on a setup which emulated uneven terrain conditions. In addition, testing on flat, level terrain was also carried out, to obtain a baseline for the performance.

The uneven terrain "testbed" developed in this study is presented in the following section.

7.3.1 Uneven Terrain Test Setup

The term "uneven terrain" is quite broad and not very specifically defined. Before being able to create an artificial terrain setup, it is necessary to consider what characteristics such a terrain ought to have, to suitably test a walking robot's capabilities. In addition, it is important to identify an appropriate measure or description of the unevenness of the terrain, or consider whether any standards exist in this regard. These considerations are essential for reproducibility of the same, or similar, testing conditions, both for follow up studies to this one and for other researchers wishing to compare their results to those obtained on the platform being used herein.

In an attempt to seek answers to these contemplations, a brief investigation was conducted on the existing methods used for testing, as well as on possible measures and standards available to described terrain unevenness. For the sake of brevity, this investigation is discussed in detail in Appendix E-1, and only the main findings will be outlined here.

It was found that most of the existing test setups used with quadruped and hexapod robots in the literature are not created to any set standards. The height variations and unevenness of many of them are not easily measurable and the majority cannot be replicated exactly by other researchers, for comparative studies. Terrain unevenness measurement methods utilized in other fields were also explored, such as vehicles dynamics, where road roughness is generally classified using Power Spectral Density (PSD) functions. However, it was found that such methods are not very applicable to legged robot locomotion studies.

As a result of the findings arising from this investigation, recommendations for uneven terrain testing of legged robots are discussed in Section 8.3.4. Furthermore, it was decided to create a "field" of closely packed blocks, of varying heights, for use as the uneven terrain in the experiments conducted in this study. This block field type of terrain is quite popularly used in legged robot literature. However, the methods used to generate the layout of the block field created in this study attempt to overcome some of the drawbacks of similar terrains encountered in the literature. Appendix E-2 explains the

design and development of this uneven terrain testbed in detail, including the specification of the heights of each block in the field, for easier replication in other studies.

The resulting testbed consists of a 10 × 10 field of square blocks of length 7.5 cm, for a block field size of 75 cm × 75 cm, which is located in the centre of a larger platform on which the robot can walk. The maximum height difference in the block field is 8 cm. An image of the test platform is presented in Figure 7.5.

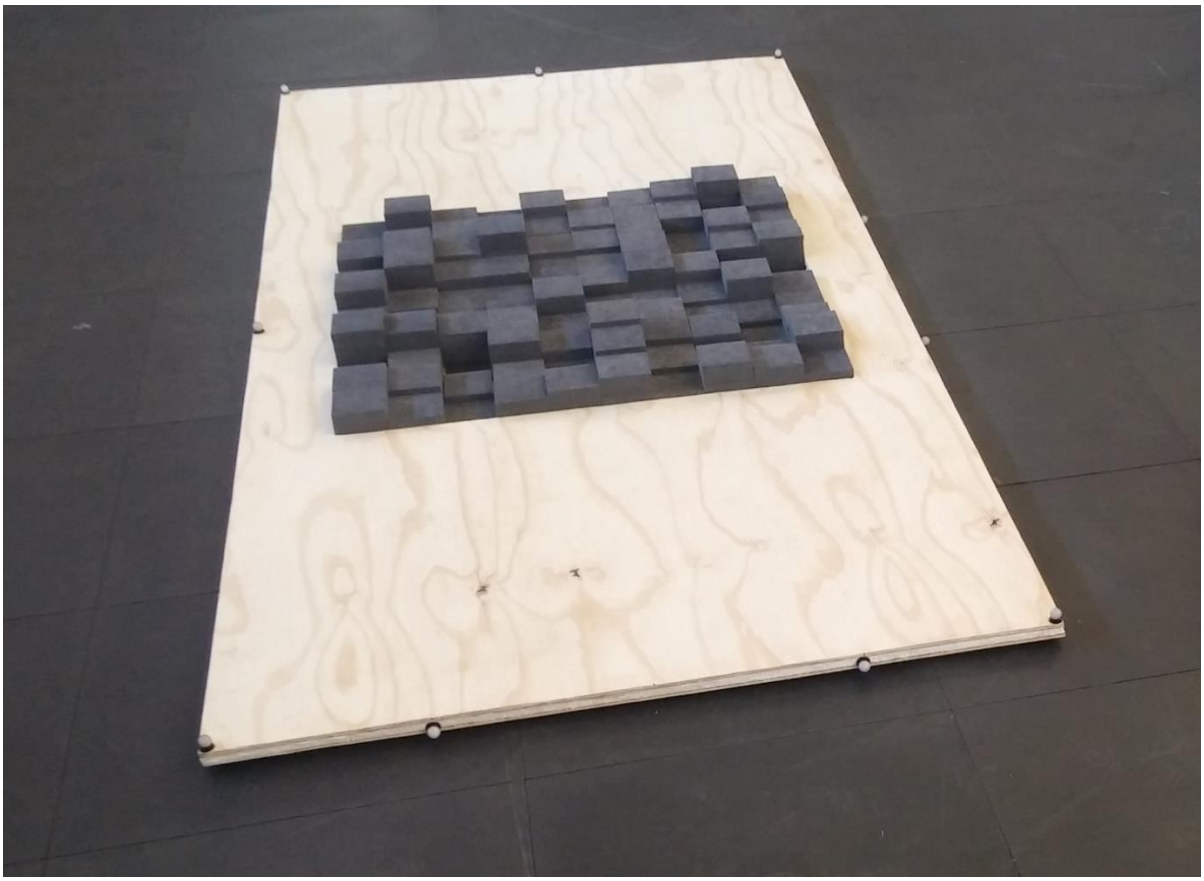


Figure 7.5: Uneven terrain test platform with 10 × 10 block field in the centre

7.3.2 Test Procedure

As mentioned above, experimental tests to evaluate the walking posture controller performance were conducted on both flat and uneven terrain. For flat terrain, the robot was made to walk on the floor of the Vicon arena, in the lab at MIAS, CSIR. This floor is covered in tiles made of a hard, rubber-like material, laid flush against each other (visible in Figure 7.5 above). Although minute height variations are present along the edges where some tiles meet, these were deemed small enough to hardly have any effect on the robot's posture while walking. Of course, for uneven terrain, the robot was made to walk back and forth on the developed testbed, presented in the previous section.

On both terrain types, tests were performed with the posture control system both disabled (uncompensated condition) and enabled (compensated condition), to analyse the effects the posture

control system has on the robot's motion. Three runs of walking forward and backward were performed for each control condition on each of the terrains.

The robot's locomotion was controlled with manual inputs from the PS3 remote control. However, care was taken to not influence the motion to improve uneven terrain traversability in any way. Remote control inputs were given to primarily make the robot walk forward and backward at full speed. Occasionally, very small sideward motions were required to ensure that the robot did not walk off the edge of the test platform, when it started veering away from the straight path of motion desired. No yawing motion or speed modulation were induced, so that the robot was not assisted in navigating over the uneven terrain. Furthermore, the robot was made to start walking from a different position in each run, to ensure that it was exposed to various sections of the uneven terrain, rather than, for instance, taking only the path that worked best.

For all tests, the Vicon motion capture system was used to obtain ground truth measurements, as in the case of the Standing Posture Controller. Specific details regarding the setup of the Vicon system are discussed in the next section.

A final note should be made on the posture setpoints used during testing. Considering that the robot could encounter steps as large as 8 cm on the block field, it was necessary to increase the leg lift height, and subsequently its standing height, accordingly in the gait engine settings. Exposing the robot to terrain which it cannot even traverse would destroy the point of testing the performance of the controller. Therefore, the standing height and leg lift height settings of the locomotion engine were updated to the values indicated at the end of Appendix C, while the corresponding, updated height setpoint for VMC is provided in Table 7.4 below.

Table 7.4: Default posture setpoint values for Walking Posture Controller

Setpoint Component	Value
Height, h_d	0.153 m
Roll angle, ϕ_d	0°
Pitch angle, θ_d	0°

While smaller values could possibly have been used with posture control enabled, the uncompensated locomotion would not be able to get the robot across the block field with a lower standing height and leg lift. In order to effectively compare the differences in performance with and without posture control, it is more appropriate to have a test case in which the robot is able to traverse the terrain in both of these control conditions.

Changing the nominal standing height of the robot naturally changes the amount by which the legs can be raised or lowered before reaching the ends of their kinematic ranges. Consequently, the saturation limits were required to be adjusted to values of +8 cm and -2 cm. It should be noted that these values are relative to the vertical position of the leg when the robot stands at its nominal height.

7.3.3 Vicon Setup and Data Processing

Reflective Vicon markers were placed on the uneven terrain test platform, in a similar configuration as the balance board used for the Standing Posture Control tests. As before, a Vicon object model was created for the platform using these markers. The coordinate frame of this model was positioned in the centre of the platform's horizontal plane. In the vertical direction the frame origin was aligned with the central markers on either side of the platform. Additionally, the frame was orientated with its x-axis pointing along the length of the platform, while the z-axis points upward.

The platform was tracked with the Vicon system in an attempt to obtain a reference against which to measure the motion of the PhantomX, by translating its frame down in the z-direction to obtain the height at the centre of the platform's surface, as was done for the balance board in the Standing Posture Controller tests (Figure 7.6 shows the Vicon object models of the PhantomX and test platform, when the PhantomX was standing on the platform).

Unfortunately, this did not provide as useful a height reference as the balance board did in the previous tests. The reasons for this are elaborated in more detail in the next section.

The same model as described in Section 6.2.4 (and shown in Figure 6.6) was used for the PhantomX, and the same procedures were followed for recording and processing the data measured with the Vicon motion system.

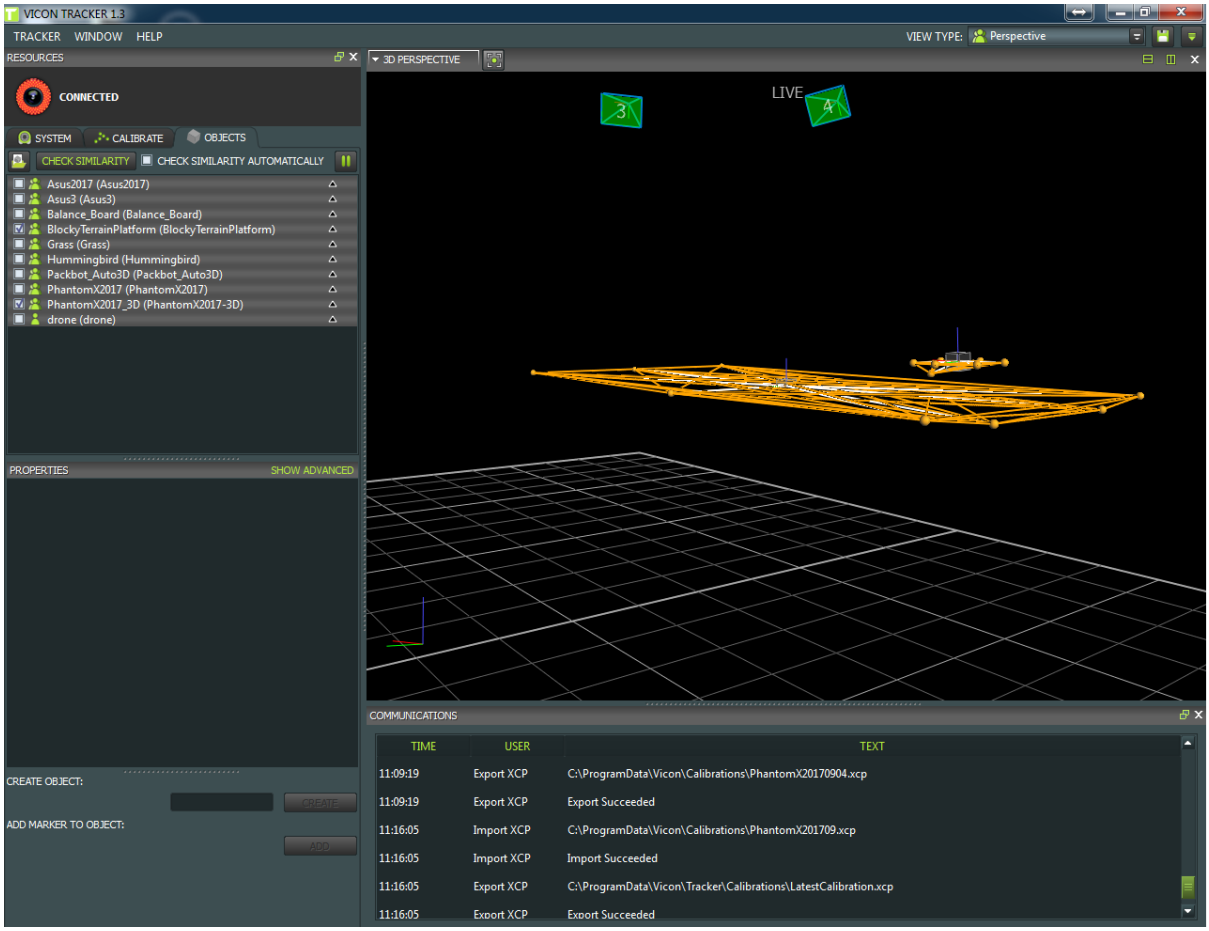


Figure 7.6: Screenshot of Vicon Tracker GUI, showing the PhantomX object model above the uneven terrain test platform object model

7.4 Results and Discussion

Results of the experimental tests described above, for walking on both flat and uneven terrain, are presented in the following sections. As in Chapter 6, the results of multiple runs were found to compare well with each, and so only representative plots of a single run of each test are presented below, for the sake of brevity. The discussions of the various results do, however, take into account data obtained from all the test runs conducted.

Furthermore, although the position of the uneven terrain platform was tracked with the Vicon system, to be used as a reference for the PhantomX, the local height of the robot above the terrain, at any given point, cannot really be measured relative to a single, fixed reference frame somewhere in space. As the robot walks over obstacles its vertical position in the world changes significantly, resulting in ground truth measurements indicating large variations in height, rather than providing a measure of the actual height of the body above the terrain which it stands over. Even on flat terrain, any small changes in elevation across the test area result in a perceivable drift in the vertical position of the robot as it walks.

Additionally, since the plywood sheets of the test platform underwent some warping during manufacturing, the two ends along the length of the platform are raised slightly higher than the centre. Consequently, the Vicon measurements pertaining to height were found to not be particularly useful in analysing the height controller's performance. The results presented below thus only show the estimated height values, using sensor measurements.

In fact, it should be kept in mind that, when the robot walks, the contact legs continually change, coming into contact with points on the terrain at different elevations, resulting in abrupt changes to the calculated height of the robot. This measure essentially just gives an idea of the ground clearance of the body, rather than information about its absolute position and motion in space. Thus, the control of the height during walking only really acts to ensure that the robot maintains sufficient ground clearance as it walks, without the legs eventually being pushed too far down or retracted too high up in trying to control orientation. The discussions that follow will, therefore, focus predominantly on the orientation response of the robot.

7.4.1 Walking on Flat Terrain

The posture response of the robot when walking on flat terrain, without posture control (uncompensated), is plotted separately for walking forward and backward in Figures 7.7 and 7.8, respectively. Similarly, results of the response when walking on flat terrain with posture control (compensated) are plotted in Figures 7.9 and 7.10.

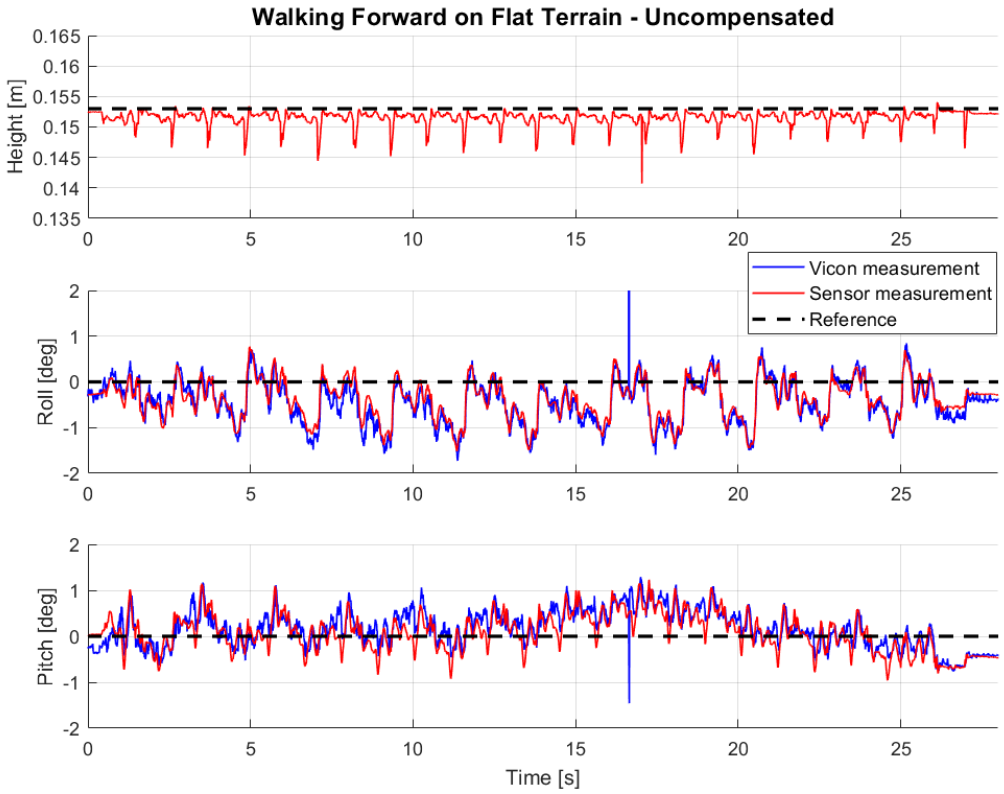


Figure 7.7: Posture response while walking forward on flat terrain, with posture control disabled

Figure 7.8: Posture response while walking backward on flat terrain, with posture control disabled

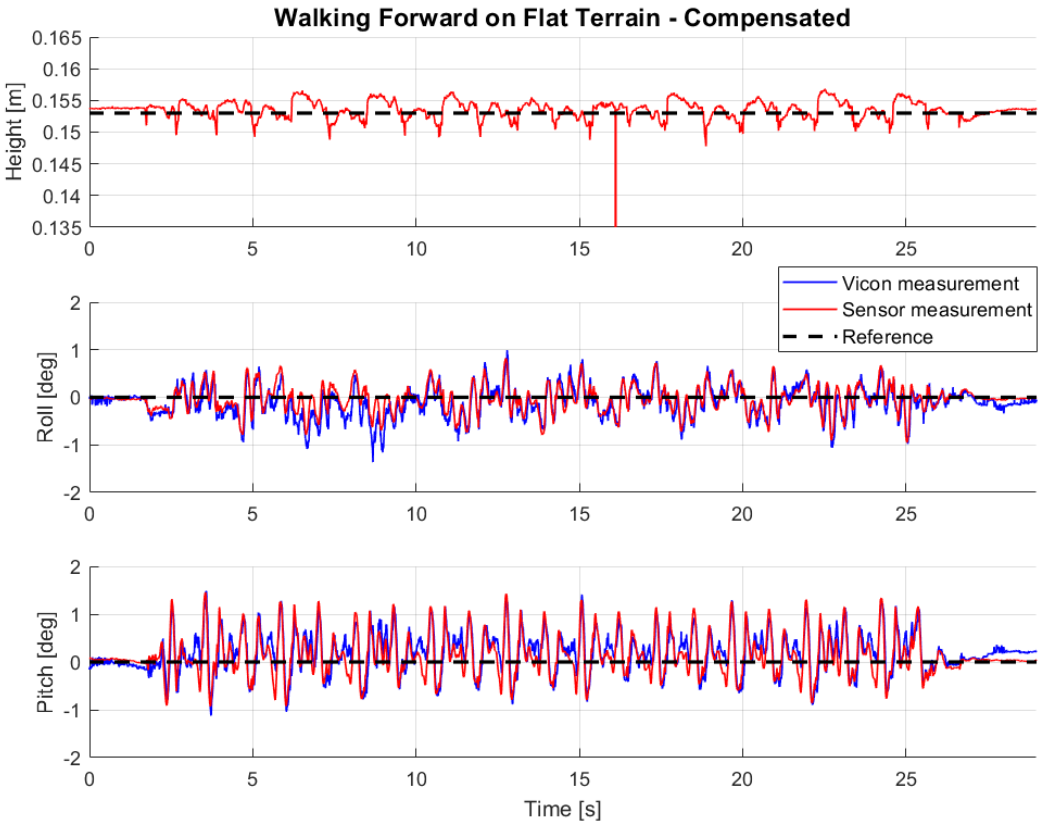


Figure 7.9: Posture response while walking forward on flat terrain, with posture control enabled

Figure 7.10: Posture response while walking backward on flat terrain, with posture control enabled

Firstly, it should be mentioned that the Vicon and sensor measurements compare favourably, as in the case of the Standing Posture Controller experiments. In addition, momentary spikes are observed in the sensor measurements of height, once again. As discussed in Section 6.3.1.1, these result from occasional errors in reading the servo positions, due to corrupted data packets returned by the servos.

Another anomaly observed in the above results is an occasional spike in the Vicon measurements, such as those visible in the roll and pitch plots of Figure 7.7, at approximately 16.7 s. This possibly occurs when the object being tracked is partially occluded from the Vicon cameras, by the operator of the robot or large objects present in the Vicon arena, resulting in erroneous estimations of its position and orientation.

When inspecting the estimated height values in the uncompensated tests, it can be seen that the estimated height almost always drops below the height setpoint while the hexapod walks. This is due to the fact that the gait engine only lifts the legs above their nominal height during walking, resulting in the average robot height being reduced when legs contact the terrain before the end of their swing phase (which occurs on flat terrain because of the variations in pitch and roll as the robot walks).

With compensation, however, the stance legs can be pushed further down, past the nominal position, resulting in the average height values fluctuating both above and below the setpoint value. These

fluctuations rarely exceed a 5 mm band above and below the setpoint, while the steady state error, when standing still, remains below 1 mm. Similar steady state errors are also observed in the uncompensated results, suggesting that small foot position errors occur from the commanded values. These likely result from numerical errors in the inverse kinematics calculations and mappings from joint angles to servo positions, as well as errors between the commanded and actual servo positions.

During the uncompensated tests, the magnitude of the largest roll angle deviation observed is approximately 1.8°, while pitch angle magnitudes of up to 1.4° are observed. It should also be noted that the roll and pitch responses deviate from the setpoint value quite consistently, experiencing a general, back-and-forth drift over time. Furthermore, the largest steady state error magnitudes observed are in the region of 0.3° and 0.4° for roll and pitch, respectively.

With compensation, the largest roll angle deviations observed are approximately 1.4°, while pitch angle magnitudes rarely exceed 1.5°. One extreme value with a magnitude of 3.3° was observed in the pitch angle, as can be seen in Figure 7.10. However, this perturbation occurred immediately after a particularly unpleasant servo read error at around 4.49 s, which resulted in the height estimation momentarily dropping to only 0.047 m, causing the controller to react extensively, thus inducing an inadvertent pitching motion. This type of rare occurrence could be prevented by making appropriate provisions in the control software to deal with servo read errors.

In addition, the largest steady state error magnitudes observed for both roll and pitch are approximately 0.2°. It is also clear from the above plots that, with compensation, the roll and pitch angles oscillate more uniformly about the setpoint of 0°, as compared with the uncompensated responses, which exhibit consistent deviations for longer periods of time.

Apart from the one anomaly with pitch, in general the roll and pitch response are observed to have been improved slightly over the uncompensated case, even though the robot is only walking on flat terrain.

7.4.2 Walking on Uneven Terrain

Plots of the posture response of the robot, while walking on uneven terrain, are presented in Figures 7.11 and 7.12 for the uncompensated case, with posture control disabled, while those for the compensated case, with posture control enabled, are presented in Figures 7.13 and 7.14.

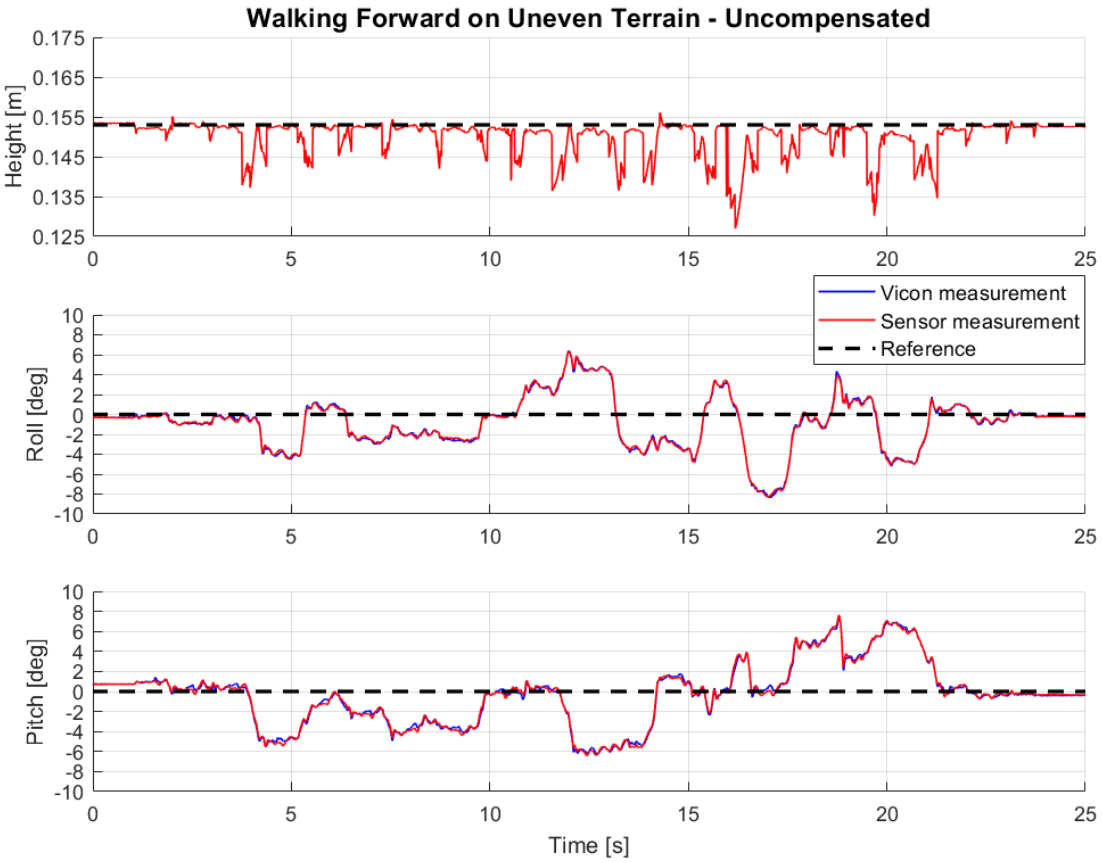


Figure 7.11: Posture response while walking forward on uneven terrain, with posture control disabled

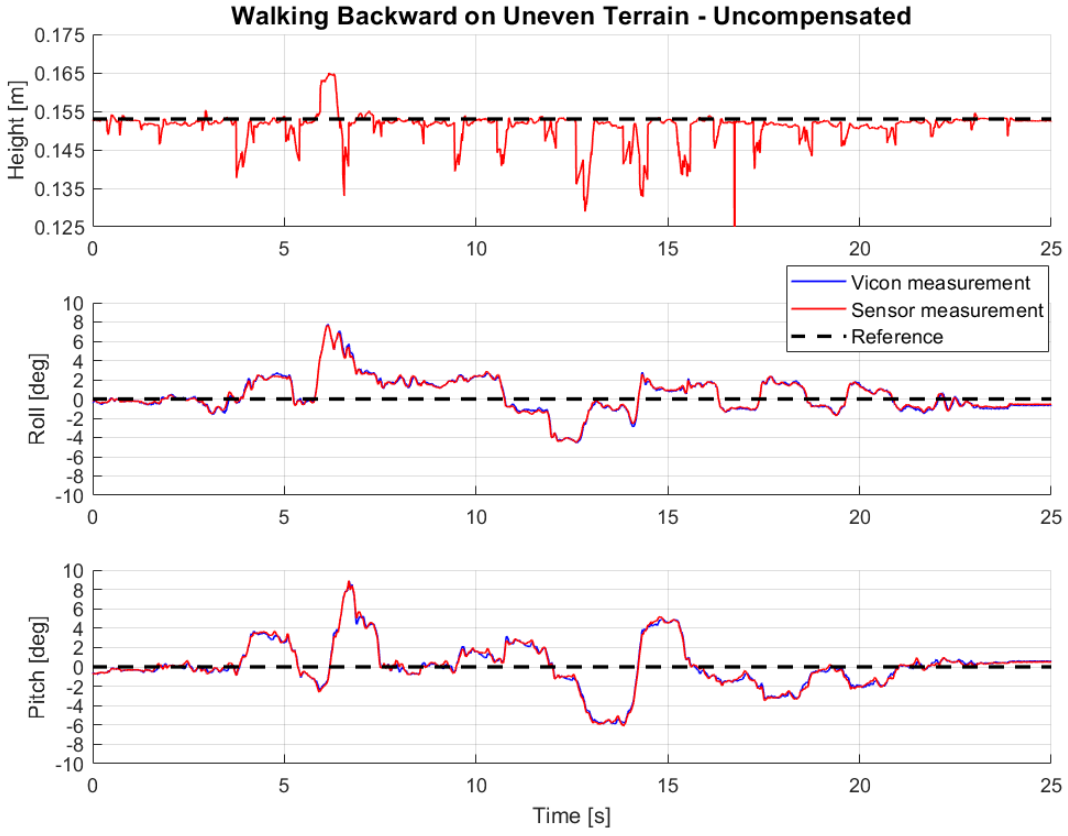


Figure 7.12: Posture response while walking backward on uneven terrain, with posture control disabled

Figure 7.13: Posture response while walking forward on uneven terrain, with posture control enabled

Figure 7.14: Posture response while walking backward on uneven terrain, with posture control enabled

When comparing the uncompensated height response on uneven terrain to that obtained on flat terrain, it can be seen that the general shape is similar, with the height values mostly dropping below the setpoint. The magnitudes of these drops are considerably larger (up to approximately 15 mm larger) on uneven terrain. This is to be expected, as the legs come into contact with blocks of significant heights, resulting in the perceived ground clearance upon contact being reduced by a greater amount than on flat terrain. Steady state height values, when the robot stops walking on the flat sections of the terrain setup, still remain within 1 mm of the desired height.

Likewise, the compensated height response on uneven terrain demonstrates a similar shape to that on flat terrain, with the magnitudes of deviations increasing. Height deviations as large as 30 mm from the setpoint are observed in the results, once again being expected due to the highly varying elevation of the terrain. Steady state error magnitudes for the height response are found to remain considerably smaller than 1 mm.

With regard to the orientation response, however, it is clear from the above plots that the uncompensated roll and pitch response of the robot is far worse on uneven terrain than on even terrain. Maximum roll and pitch angle deviation magnitudes of 8.4° and 8.9°, respectively, are observed across the runs. In addition to the maximum values being significantly larger, the plots above demonstrate that the roll and pitch angles deviate far from the setpoint for long periods of time, with the robot body having a notably undesirable orientation throughout most of its locomotion across uneven terrain. Furthermore, during testing the robot was sometimes observed to be lifted to such an extent when a leg stepped onto a high block, that many of the legs could not make contact with the terrain, resulting in a temporary loss of “traction” until the robot tipped over the other way and enough legs made contact again. Such an instance can be seen in Figure 7.12, at approximately 6 s, when all three posture components exhibit appreciably large deviations.

Steady state error magnitudes for the uncompensated tests are found to be as large as 0.7° and 0.8° for roll and pitch, respectively. The reason for these values being larger than those obtained in the flat terrain tests can most likely be attributed to the fact that the slight warping of the test platform results in the flat sections not being completely level.

When posture control is enabled, the difference in the orientation response on uneven terrain is easily perceptible. Roll and pitch angles both oscillate more consistently about the setpoint value of 0°, spending less time at extreme values of deviation. Maximum magnitudes of roll and pitch across the runs are found to be around 6° and 6.4°, corresponding to improvements of 28.6% and 28.1%, respectively, from the maximum values observed on uneven terrain with posture control disabled. However, it should be noted that such large deviations were quite rare, unlike in the uncompensated

case where the response frequently approached values close to the observed maximums. The cases of the maximum roll and pitch values quoted here, for the compensated tests, both occurred when the robot had most of its stance legs on high blocks and one of the swing legs stepped onto a low block, or a “dip” in the terrain (such an incident can be seen in the pitch response of Figure 7.14, at approximately 12.5 s). Naturally, this would have initially resulted in only two of the new stance legs having contact, causing the robot to tip over briefly until the other stance leg managed to make contact with the low section of terrain.

It was discussed in Section 5.1 that such an incident results in a temporary loss of posture controllability, which is exactly the effect observed here. Such occurrences are unavoidable when using a purely proprioceptive posture control system, as the robot has no knowledge of the elevation of the terrain at the planned foothold position. Despite these momentary losses in controllability, the plots above demonstrate that the controller does well to quickly restore the posture of the robot, as soon as it regains controllability. The posture deviations are limited and sufficient traction is maintained as the robot traverses the uneven terrain, unlike the behaviour observed in the uncompensated tests.

Much of the time, the roll and pitch responses remain well within a ±2° band with posture control enabled, resulting in the response being much closer to that observed on flat terrain. Larger deviations in orientation are restored quickly and it is evident from the above results that the overall response is improved notably from the uncompensated response. Also, the steady state error magnitudes for these tests remained below 0.3°.

Although the tests conducted and reported on formally here were designed to allow for locomotion over the terrain in both the uncompensated and compensated conditions, for comparative purposes, it is valuable to also mention that the differences in the two control conditions were sometimes observed to be even more drastic on other uneven terrains. For instance, when walking the robot over obstacles and artificially contoured terrains (used for controller tuning) with posture control disabled, considerably larger maximum orientation deviations were observed and the effects of a loss of traction without compensation were much more severe. In many cases, the robot would get into a situation where it simply could not regain sufficient ground contacts to proceed with locomotion in the uncompensated state. With posture control enabled, however, this problem was easily overcome and the robot managed to traverse the terrains without any difficulty. Therefore, the presence of posture control not only improves the posture response on uneven terrain, but, in some cases, it also improves the robot’s ability to traverse terrains it otherwise might not have been able to.

To put the results obtained in these experimental tests into context with the literature, it is useful to compare them to those obtained in similar tests with other hexapods. For experiments performed with the HITCR-II hexapod walking on artificial uneven terrain with a tripod gait, Zhang et al. [10] reported roll and pitch angle plots that exhibited maximum deviation magnitudes of approximately 6° and 3°, respectively. It is important, however, to keep in mind that the terrain irregularities used in their experiment were simply a few rectangular blocks of 50 mm height, scattered in the hexapod's path of motion, as depicted previously in Figure 2.16. This is a much less severe uneven terrain condition than that used for the experiments in this study. Furthermore, HITCR-II was made to walk significantly more slowly, with the period of each phase set to 5 s. To put this into perspective, at the speed that the PhantomX was made to walk, the leg phases changed roughly every 0.8 s, meaning the PhantomX experienced disturbances from the terrain approximately six times more frequently than HITCR-II while walking. It should also be remembered that HITCR-II is a much more sophisticated, custom designed platform, with smoother and more accurate joint actuators, as well as a slightly more complex control structure, which includes optimization methods for foot force distribution. Taking all of the above factors into account, it is quite clear that the results obtained with the PhantomX in this study compare well to those reported in [10].

Another recent study worth comparing to is the one conducted by Bjelonic et al. [13] with the Weaver hexapod. As mentioned previously, the proprioceptive control system implemented on Weaver in [13] was experimentally tested on a block field, such as that used in this study, as well as on more natural terrain conditions consisting of actual rocks and stones. While the maximum heights of the blocks used in their testbed appear to be greater than those used in this study, it should be realized that Weaver's body is approximately one and a half times larger than the PhantomX, and it also has a mass of 7.03 kg, which is more than three times that of the PhantomX platform used in this study. Furthermore, the 5 DOF legs are considerably longer than those of the PhantomX, having improved reach and higher leg lift heights, while also giving Weaver a higher nominal standing height. Based on the results reported by them, for walking on the block field with a tripod gait at similar speeds to the PhantomX, Weaver appeared to experience roll and pitch angle deviations as large as approximately 5° and 10°, respectively. Considering that Weaver makes use of impedance controllers in the legs, as well as an inclination controller that adjusts the two additional joints in each leg to improve the robot's orientation, the results obtained in the current study on the PhantomX are evidently favourable when compared to those obtained in [13].

7.5 Conclusion

The Standing Posture Controller described in Chapter 6 was modified for walking on uneven terrain. Modifications were made to the Single Leg Foot Force Controller, to provide active compliance during

the swing phase, using the same general control structure used for direct force control in the stance phase, with a simple switching strategy between phases. Furthermore, the foot force distribution procedure was updated to distribute forces to only the stance tripod of legs when walking.

An uneven terrain testbed, in the form of a block field, was developed and used to experimentally test the performance of the Walking Posture Controller. The results of these tests revealed that the controller significantly improves the robot's posture response when walking on uneven terrain, as compared to the case of walking without any posture control. Furthermore, the results were found to compare favourably to those obtained in the literature, for similar tests performed with custom designed hexapod platforms.

Additional modifications which could be made to improve the performance further include making provisions in the control software to better handle servo read errors, adjusting parameters such as walking speed on highly uneven terrain, improving the foot force distribution rule, and expanding the force controller to allow for dynamic behaviour to be influenced during interaction in the swing phase (for example, by using techniques similar to impedance control), among others.

It is worth pointing out that although the control system design in this study was targeted toward commercially available platforms (primarily due to their cost advantages and potential to reduce barriers to entry in the related research field), it could conceivably also be applied on more expensive robotic systems. One of the key limitations overcome by the design is the lack of torque controllability in the joint actuators. Therefore, this particular design could also be useful on other, possibly more expensive, robotic platforms that make use of position-controlled actuators.

Chapter 8: Conclusion

Having completed the requirements of the study, the sections that follow summarize the implemented methods and main findings, discuss recommendations for future improvements and developmental studies, and assess whether the aim of the investigation has been accomplished.

8.1 Summary of Implemented Methods

In reviewing the literature pertaining to uneven terrain locomotion of multi-legged robots, it was found that posture control is a vital component in realizing such locomotion. Furthermore, the majority of this research was found to have been implemented on specialized, custom designed robotic platforms, with very little development and implementation having taken place on low-cost, commercially available robots. These findings provided the primary motivation for the study, while also identifying the most applicable posture control method to be implemented in this study.

The first step taken in addressing the issues identified above was to upgrade the PhantomX MK-II based hexapod robot platform available for this study. In particular, OptoForce sensors were incorporated into the platform to measure contact forces at the feet. Basic tests were conducted to validate the measurements provided by these sensors, while new feet were also designed and manufactured for the platform, to be able to mount these sensors on the robot's legs. Further updates were made to the platform to utilize ROS-based locomotion software, by equipping it with the necessary hardware and making minor modifications to the locomotion engine settings.

Thereafter, a simplified controllability analysis was performed on the robotic system, from which the main scenarios that would lead to a loss of posture controllability were identified. The study then proceeded on to matters more closely related to the actual implementation of a posture control system, by deriving the methods used to calculate the robot's posture, in the form of the body's height above the ground, as well as roll and pitch angles relative the horizontal plane of the Earth. This included computation of the individual foot positions (for height estimation), by feeding the joint

position readings from the actuators into a kinematic model of the PhantomX, as well as transformation of the orientation estimate provided by the IMU driver, in order to obtain the relevant roll and pitch angles of the robot body.

It was then possible to begin developing and implementing the posture control system for the hexapod robot. Firstly, the case of the robot standing on all six legs, without walking, was considered. For this case, a high-level Virtual Model Control system was derived, which calculates the necessary vertical force, as well as roll and pitch moments, required to restore the body's posture to the desired state. A simple foot force distribution rule was also employed to divide these forces evenly among the six stance legs. In order to control each leg to achieve these forces, a lower level Single Leg Foot Force Controller was formulated, based on a direct force control method typically used in manipulators, which provides position adjustments to the leg's position control system, to regulate contact forces without the need for torque controllable joint actuators. The resulting Standing Posture Control system is founded on simple, well understood control laws, specifically PD control in the case of VMC and PI control for the Foot Force Controller.

This control system was implemented digitally in ROS, on a laptop computer to which the robot was tethered. A variety of experimental tests were conducted to assess the performance of the system, including step tests on both flat and uneven terrain, as well as dynamic tests with the robot on a balance board. During all these tests, a Vicon motion capture system was utilized to obtain ground truth measurements of the robot's pose, for the dual purposes of analysing controller performance and also validating posture measurements calculated from sensor data.

Reflecting on the insights gained from implementing and testing the Standing Posture Control system, modifications were made to enable uneven terrain walking with the hexapod platform. While the VMC structure remained unchanged, the foot force distribution method was expanded to also distribute forces to a tripod of legs, which is necessary during walking as the PhantomX uses a tripod gait for locomotion. The other major update to the control system was the addition of an output feedback path in the Foot Force Controller. This allows for the force control system to be utilized during the leg's swing phase, providing wind-down action on the position adjustment during free-space locomotion, as well as active compliance during interaction with the environment, to absorb impacts from contacting the uneven terrain earlier than expected. A simple switching strategy is used to transition between direct force control and compliance control, when the leg's phase changes from stance to swing and vice versa.

An uneven terrain testbed, in the form of a block field, was designed and manufactured for testing the Walking Posture Control system. Experimental testing of the robot involved walking on both flat and

uneven terrain, with the posture control system both enabled and disabled. Once again, the Vicon motion capture system was used for ground truth measurements, and the results were analysed to gauge the controller's performance and ability to improve posture during locomotion.

8.2 Main Findings of the Study

One of the primary, meaningful findings of the study came from the controllability analysis, which revealed that a general condition required for posture controllability (of height, roll, and pitch) of an N-legged robot, with statically stable motion, is that it must have at least three, unique, non-collinear ground contact points. It was later found that it is not always possible to avoid violating this condition during uneven terrain locomotion, especially when using a purely proprioceptive control system. However, the implemented posture control system was observed to be able to quickly restore significant deviations in posture arising from the temporary loss of controllability during locomotion.

Furthermore, analysis of the experimental test results revealed that the sensor measurements of posture compare well with ground truth measurements, validating their use for posture control. In this regard, the Standing Posture Control system was found to be able to track step changes in the posture setpoints quickly and accurately. The steady state height response remained within 2 mm of the reference value, while a 10% – 90% rise time of approximately 0.9 s was exhibited. Steady state error magnitudes for roll and pitch remained below 0.3°, with the 10% – 90% rise time for these posture components being in the region of 0.5 s. In addition, the control system was observed to effectively decouple the three posture components during testing, despite the actual coupling present in the robot's nonlinear dynamics. These results were found to compare favourably with those obtained in similar tests with more sophisticated, custom designed platforms, such as Warp1 [4] and StarlETH [11].

While standing on a particular configuration of artificial rocks, which emulated uneven terrain conditions, the robot exhibited roll and pitch angle magnitudes of approximately 9.5° and 1°, respectively, when posture control was disabled. Enabling posture control reduced the roll and pitch magnitudes to roughly 0.3° and 0.06°, corresponding to improvements of approximately 96.8% and 94.0%, respectively. During a height step test on this rock arrangement, steady state height errors were observed to still remain below 2 mm, while sufficient decoupling of the posture components also took place. Moreover, on the dynamic balance board, the robot's roll and pitch angle magnitudes were kept smaller than 4°, approximately one third of the maximum angular displacement induced on the board. This result is comparable to that reported in [4], for a similar test using the Warp1 quadruped platform.

When walking, especially on uneven terrain, the robot's height naturally varies quite considerably over time. Nonetheless, the posture controller was observed to keep the height oscillating about the setpoint, rather than drifting off indefinitely throughout the robot's locomotion. Experimental testing of the Walking Posture Control system also demonstrated that similar maximum magnitudes of roll and pitch deviations occurred when walking on flat terrain, with posture control both disabled and enabled. However, with posture control, these deviations oscillated more consistently about the setpoints than when not using posture control. This, coupled with the improved steady state response with posture control, suggested that the posture control system improves the robot's response slightly on flat terrain.

On uneven terrain, the difference made by posture control was found to be much more distinct. The maximum roll and pitch angle deviations demonstrated were observed to be reduced, by the Walking Posture Controller, by as much as 28.6% and 28.1%, respectively, from the uncompensated case. In fact, the robot spent a large portion of its time with its orientation at values close to the observed maximums when posture control was disabled. On the other hand, the maximum deviations exhibited with posture control enabled only occurred rarely, in scenarios when controllability was momentarily lost as a result of insufficient ground contact points. In these occasional situations, the posture deviated by considerable magnitudes only briefly, being restored quickly by the control system when controllability was regained. In general, the Walking Posture Control system managed to keep the roll and pitch angles well within a ±2° band for much of the time while walking on uneven terrain, resulting in a response fairly similar to that observed on flat terrain.

When comparing these results to those obtained on other, custom designed hexapod platforms, such as HITCR-II [10] and Weaver [13], and considering the differences between those platforms and the PhantomX, as well as the test conditions used, it was concluded that the Walking Posture Control system exhibited favourable performance.

Therefore, it has been shown that a low-cost, commercially available hexapod robot, using only position control of the joints, i.e. without the joint torque control available on more sophisticated legged robot platforms, can be endowed with effective posture control. Thus, the potential to increase its usefulness as a low-cost research and development platform for legged robotic work should be further investigated.

8.3 Recommendations

Suggestions for improvements on the methods presented in this study, as well as potential focus areas for future studies, are highlighted in the sections that follow.

8.3.1 Locomotion Engine

The most significant issue that is required to be overcome in the `hexapod_ros` locomotion software, used in this study, is the blocking, inner interpolation loop present in the main control loop. Resolving this problem would likely require significant refactoring of the software implementing the locomotion engine. Considering the large amount of modification this will entail, it is recommended that the motion control system also be updated to implement closed-loop position control of the legs, at the same time, which would improve the robot's locomotion, along with increasing both the motion and interaction control capabilities of the legs.

Another useful modification is also required for the gait engine, in the form of implementing additional gaits to be used for locomotion with a hexapod.

8.3.2 Robotic Platform

It was mentioned in Chapter 6 that since only one component of the foot force measurements was used in the posture control systems designed in this study, one-dimensional force sensors could potentially be employed to further reduce the costs and complexity of the robotic platform. Of course, the force measurements from extremely low-cost sensors, such as the FlexiForce™ A201 force sensors discussed in Section 4.2.3, would be inadequate. Therefore, 1D force sensing equipment that could provide a similar quality of measurements to that of the OptoForce z-axis measurements (or, at least, satisfactory quality measurements) would need to be utilized. As OptoForce can design sensors tailored to a customer's requirements, one option could possibly be to approach them to design a 1D sensor using similar technology to their 3D sensors, especially if the market is large enough for them to manufacture sufficient quantities of such sensors at an affordable price. Alternatively, an example of already available force sensing equipment that could be considered is micro load cells, which are available at a fraction of the cost of OptoForce 3D sensors.

This should be investigated in the future and possibly tested, to compare the performance attainable with less expensive force sensors. Methods of estimating contact forces, based on the joint current feedback available from the Dynamixel servos, should also be investigated, as well as potentially combining these estimates with measurements from simpler force sensors.

Another important factor related to the real-world application of a low-cost hexapod platform is the issue of onboard computing. Throughout this study, emphasis was made on keeping the computational requirements of the control algorithms relatively inexpensive. It will be vital to further

modify and optimize the control software to be implemented on less powerful computing hardware, that could be mounted onto the robot and powered by an onboard battery, eliminating the need for a wired tether in favour of a wireless "tether" (using wireless communication such as Wi-Fi, for instance).

Furthermore, improvements should be made in the software to better handle the occurrences of servo position read errors. This would reduce undesired posture perturbations when using the posture control system.

Later studies could also be dedicated to equipping the hexapod platform with exteroceptive sensing capabilities, especially for the implementation of some of the suggestions presented for the posture control system, in the next section.

8.3.3 Posture Control System

Firstly, it was mentioned in the previous section that the use of 1D force sensors should be investigated. Of course, already having the 3D force sensors means that it is equally instructive to investigate the use of all three components of the force measurements, to improve performance. Comparing this to the current implementation could provide useful insights for future development of the posture control system.

An improvement which could be made to the control system, without much additional development, is to update the controller gains and parameters. Some of the methods outlined in Chapter 6 could be considered, along with more formalized trial-and-error methods, such as Monte Carlo simulation with varying parameters.

Another modification that could be investigated is to change the parameter values when switching between stance and swing phases. This could allow for the direct force control and compliance control actions to be tuned independently, for the desired behaviour. An example would be to keep K_C large to improve output wind-down speed, while increasing the value of K_I in the swing phase to still achieve sufficient compliance.

This approach of dynamically changing parameters could be taken a step further, as in a study with the Weaver hexapod where parameters such as duty factor, leg lift height, and leg stiffness (compliance) were dynamically, and automatically, adjusted throughout locomotion, depending on vision-based estimates of the degree of terrain roughness [107]. Of course, such an approach would require exteroceptive sensing on the platform, to assess the unevenness of the terrain.

More detailed measurements could also be taken of the hexapod platform in use, to reduce errors in the kinematic model, improving posture measurements used for control. Some provision could also be included to account for the deformation of the force sensor contact surfaces under load.

Regarding the actual control algorithms, the addition of an integral term in the VMC strategy could possibly be considered, to further improve steady state behaviour by reducing, if not eliminating, steady state errors. A more useful modification would be to improve the foot force distribution method, especially by allowing for any number of legs to be used for force distribution, so that the control system could be used with any gait. If additional gaits are developed for the locomotion engine, as suggested in Section 8.3.1 above, the updated foot force distribution could be tested on a variety of gaits.

A more drastic improvement to the control system, that could be considered, is the modification of the Foot Force Controller to allow dynamic interaction behaviour to be specified, by means of, for example, impedance control during the swing phase. Inspiration for the design of such a force control system could be gained from [108], in which a force-impedance controller was developed for manipulators, that achieves “force limited impedance control” and “position limited force control,” without the need for a supervising switching strategy.

Eventually, taking the research initiated in this study forward will entail combining posture control with higher level tasks such as foothold and path planning, with the use of exteroceptive sensing.

8.3.4 Uneven Terrain Testing of Legged Robots

In Chapter 7, limitations in the existing uneven terrain test setups used in legged robot literature were discovered (these are discussed in more detail in Appendix E-1). A significant outcome of these findings is a suggestion for future research to be carried out, on creating standardized test setups that could be manufactured and used by researchers throughout the field. These setups would need to be designed such that they could be scaled to present the same relative terrain unevenness to legged robots of different sizes.

Furthermore, standardized methods of measuring and characterizing uneven terrain more appropriately should be investigated, based on the elevation variations across their areas, rather than only the height differences at their maximum and minimum points of elevation. Potential ideas for this would be to specify heights at multiple points on the terrain (i.e. divide the area into a grid), or possibly utilize a two-dimensional equivalent of a PSD, that operates on the entire terrain surface, as opposed to a one-dimensional function as in the case of analysing road roughness for wheeled vehicles.

Another option would be for researchers to gather point cloud or 3D scan data of their uneven terrain testbeds and make that data available, allowing for the setups to be reproduced with 3D printing, or other manufacturing techniques.

In addition to development of uneven terrain testing standards, it is also recommended that the PhantomX platform and posture control system used in this study are tested on a multitude of other terrains, to assess wider applicability and improve the overall performance of the controller.

8.4 Study Aim Fulfilment

A posture control system was successfully implemented on the low-cost, commercially available hexapod robot platform available for this study, for the cases of standing on all six legs and walking on uneven terrain. Experimental tests of both of these cases confirmed the usefulness of the control system in improving the robot's response, especially on uneven terrain, while also demonstrating satisfactory performance comparable to that displayed on more sophisticated, custom designed platforms. Accordingly, the aim of the study has undoubtedly been fulfilled.

Bibliography

[1] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, “BigDog , the Rough-Terrain Quadruped Robot,” in *Proceedings of the 17th World Congress: The International Federation of Automatic Control (IFAC)*, 2008, pp. 10822–10825.

[2] J. Machado and M. Silva, “An overview of legged robots,” in *Proceedings of the International Symposium on Mathematical Methods in Engineering (MME)*, 2006.

[3] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, 1st ed. Cambridge, MA: The MIT Press, 2004.

[4] C. Ridderström and J. Ingvast, “Quadruped posture control based on simple force distribution – a notion and a trial,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001, vol. 4, pp. 2326–2331.

[5] X. Ding, Z. Wang, A. Rovetta, and J. M. Zhu, “Locomotion analysis of hexapod robot,” in *Climbing and Walking Robots*, B. Miripour, Ed. InTech, 2010, ch. 18, pp. 291–311 [Online]. Available: http://cdn.intechopen.com/pdfs/10075/InTech-Locomotion_analysis_of_hexapod_robot.pdf. [Accessed: 16-Jan-2018]

[6] M. Hutter, C. Gehring, M. Hoepflinger, M. Blösch, and R. Siegwart, “Towards Combining Speed, Efficiency, Versatility and Robustness in an Autonomous Quadruped,” *IEEE Trans. Robot.*, vol. 30, no. 6, pp. 1427–1440, 2014.

[7] H. Uchida and N. Shimoi, “Force/attitude control of mine detecting six-legged locomotion robot,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000, vol. 1, pp. 780–785.

[8] A. Irawan and K. Nonami, “Optimal impedance control based on body inertia for a hydraulically driven hexapod robot walking on uneven and extremely soft terrain,” *J. Field Robot.*, vol. 28, no. 5, pp. 690–713, 2011.

[9] A. Roennau, G. Heppner, M. Nowicki, J. M. Zoellner, and R. Dillmann, “Reactive posture behaviors for stable legged locomotion over steep inclines and large obstacles,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014, pp. 4888–4894.

[10] H. Zhang, Y. Liu, J. Zhao, J. Chen, and J. Yan, “Development of a bionic hexapod robot for walking on unstructured terrain,” *J. Bionic Eng.*, vol. 11, no. 2, pp. 176–187, 2014.

[11] M. Blösch, “State Estimation and Robust Control for a Quadruped Robot,” *Master-Thesis*, Autonomous Syst. Lab, ETH Zurich, Switzerland, 2011.

[12] G. M. Nelson and R. D. Quinn, “Posture control of a cockroach-like robot,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998, vol. 1, pp. 157–162.

[13] M. Bjelonic, N. Kottege, and P. Beckerle, “Proprioceptive control of an over-actuated hexapod robot in unstructured terrain,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2042–2049.

[14] E. Lubbe, “State estimation of a hexapod robot using a proprioceptive sensory system,” *MEng Dissertation*, North-West University, South Africa, 2014 [Online]. Available: https://repository.nwu.ac.za/handle/10394/15374. [Accessed: 21-May-2018]

[15] Trossen Robotics, “PhantomX AX Hexapod Mark II.” [Online]. Available: http://www.trossenrobotics.com/hex-mk2. [Accessed: 09-Nov-2017]

[16] “What is motion capture | VICON.” [Online]. Available: https://www.vicon.com/what-is-motion-capture. [Accessed: 15-Nov-2017]

[17] K. Ochs, “hexapod_ros,” *GitHub*. [Online]. Available: https://github.com/KevinOchs/hexapod_ros. [Accessed: 16-Jan-2018]

[18] J. Zhao, H. Zhang, Y. Bin Liu, and Z. W. Zhou, “Active Compliance Control for the Leg of Hexapod Robot HITCR-II,” *Appl. Mech. Mater.*, vol. 201–202, pp. 578–581, 2012.

[19] M. Tikam, D. Withey, and N. J. Theron, “Standing Posture Control for a Low-Cost Commercially Available Hexapod Robot,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3379–3385.

[20] KUKA AG, “Automation in the automotive industry.” [Online]. Available: https://www.kuka.com/en-de/industries/automotive. [Accessed: 05-Nov-2017]

[21] NASA, “Curiosity Overview.” [Online]. Available: https://www.nasa.gov/mission_pages/msl/overview/index.html. [Accessed: 06-Nov-2017]

[22] D. Carvajal, “Let a Robot Be Your Museum Tour Guide,” *The New York Times*, New York, 14-Mar-2017 [Online]. Available: https://www.nytimes.com/2017/03/14/arts/design/museums-experiment-with-robots-as-guides.html. [Accessed: 16-Jan-2018]

[23] M. H. Raibert, “Legged robots,” *Communications of the ACM*, vol. 29, no. 6, pp. 499–514, 1986.

[24] P. Gonzalez de Santos, J. A. Cobano, E. Garcia, J. Estremera, and M. A. Armada, “A six-legged robot-based system for humanitarian demining missions,” *Mechatronics*, vol. 17, no. 8, pp. 417–430, 2007.

[25] R. Campos, V. Matos, and C. Santos, “Hexapod locomotion: A nonlinear dynamical systems approach,” in *Proceedings of the 36th Annual Conference of IEEE Industrial Electronics (IECON)*, 2010, pp. 1546–1551.

[26] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, “Robots that can adapt like natural animals,” *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[27] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng, “A control architecture for quadruped locomotion over rough terrain,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 811–818.

[28] D. Belter and P. Skrzypczynski, “Posture optimization strategy for a statically stable robot traversing rough terrain,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 2204–2209.

[29] Boston Dynamics, “WildCat.” [Online]. Available: https://www.bostondynamics.com/wildcat. [Accessed: 06-Nov-2017]

[30] American Honda Motor Co. Inc., “ASIMO.” [Online]. Available: http://asimo.honda.com/. [Accessed: 06-Nov-2017]

[31] A. Preumont, P. Alexandre, and D. Ghuys, “Gait analysis and implementation of a six leg walking machine,” in *Proceedings of the 5th International Conference on Advanced Robotics (ICAR) “Robots in Unstructured Environments,”* 1991, pp. 941–945.

[32] DFKI GmbH: Robotics Innovation Center, “SCORPION.” [Online]. Available: https://robotik.dfki-bremen.de/en/research/robot-systems/scorpion.html. [Accessed: 06-Nov-2017]

[33] F. Tedeschi and G. Carbone, “Design Issues for Hexapod Walking Robots,” *Robotics*, vol. 3, no. 2, pp. 181–206, 2014.

[34] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, “Fast, robust quadruped locomotion over challenging terrain,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2665–2670.

[35] K. Walas and D. Belter, “Messor – Versatile Walking Robot For Search and Rescue Missions,” *Journal of Automation, Mobile Robotics & Intelligent Systems*, vol. 5, no. 2, pp. 28–34, 2011.

[36] A. Roennau, T. Kerscher, and R. Dillmann, “Design and Kinematics of a Biologically-Inspired Leg for a Six-Legged Walking Machine,” in *Proceedings of the 3rd IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2010, pp. 626–631.

[37] Australian UAV, “Drone Data vs LIDAR: Clarifying Misconceptions.” [Online]. Available: https://www.auav.com.au/articles/drone-data-vs-lidar/. [Accessed: 06-Nov-2017]

[38] N. Rotella, M. Bloesch, L. Righetti, and S. Schaal, “State Estimation for a Humanoid Robot,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014, pp. 952–958.

[39] L. Gyeong-Mok, Y. Seongyeol, S. Hyungwon, J. Bong-Huan, K. Hangoo, and L. Pan-Mook, “Kinematic walking and posture control of CR200 for subsea exploration in high tidal current,” in *Oceans - San Diego*, 2013, pp. 1–6.

[40] P. Arena, L. Fortuna, and M. Frasca, “Attitude control in walking hexapod robots: an analogic spatio-temporal approach,” *Int. J. Circuit Theory Appl.*, vol. 30, no. 2–3, pp. 349–362, 2002.

[41] K. Yoneda, H. Iiyama, and S. Hirose, “Sky-Hook Suspension control of a quadruped walking vehicle,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1994, vol. 2, pp. 999–1004.

[42] H. Qingjiu and Y. Fukuhara, “Posture and Vibration Control Based on Virtual Suspension Model Using Sliding Mode Control for Six-Legged Walking Robot,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 5232–5237.

[43] P. Arena, L. Fortuna, M. Frasca, L. Patane, and M. Pavone, “Climbing obstacles via bio-inspired CNN-CPG and adaptive attitude control,” in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS).*, 2005, pp. 5214–5217.

[44] FZIchannel, “Walking Robot LAURON V - Hexapod on Wooden Slope,” 2013. [Online]. Available: http://www.youtube.com/watch?v=sYrhI1pSpyQ. [Accessed: 07-Nov-2017]

[45] P. Cizek and J. Faigl, “On localization and mapping with RGB-D sensor and hexapod walking robot in rough terrains,” in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 2273–2278.

[46] T. Fischer, T. Pire, P. Cizek, P. De Cristoforis, and J. Faigl, “Stereo vision-based localization for hexapod walking robots operating in rough terrains,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2492–2497.

[47] J. Christie and N. Kottege, “Acoustics based terrain classification for legged robots,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3596–3603.

[48] D. Williamson, N. Kottege, and P. Moghadam, “Terrain Characterisation and Gait Adaptation by a Hexapod Robot,” in *Proceedings of the Australasian Conference on Robotics and Automation (ACRA)*, 2016.

[49] M. Stejskal, J. Mrva, and J. Faigl, “Road following with blind crawling robot,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3612–3617.

[50] M. Hörger, N. Kottege, T. Bandyopadhyay, A. Elfes, and P. Moghadam, “Real-Time Stabilisation for Hexapod Robots,” in *Proceedings of the 14th International Symposium on Experimental Robotics (ISER)*, 2016, pp. 729–744.

[51] J. Mrva, “Adaptive Body Motion for Blind Hexapod Robot in a Rough Environment,” in *19th International Student Conference on Electrical Engineering (POSTER 2015)*, 2015.

[52] R. Love, “Project Thread: ROS enabled PhantomX Hexapod,” *Trossen Robotics Community Forum*, 2015. [Online]. Available: http://forums.trossenrobotics.com/showthread.php?7411-ROS-enabled-PhantomX-Hexapod. [Accessed: 07-Nov-2017]

[53] K. Ochs, “Project Thread: ROS Hexapod project Golem: MX-64 4dof,” *Trossen Robotics Community Forum*, 2014. [Online]. Available: http://forums.trossenrobotics.com/showthread.php?6725-ROS-Hexapod-project-Golem-MX-64-4dof. [Accessed: 07-Nov-2017]

[54] “ROS.” [Online]. Available: http://www.ros.org/. [Accessed: 16-Jan-2018]

[55] Love Robotics, “PhantomX ROS IMU Auto Levelling,” 2015. [Online]. Available: https://www.youtube.com/watch?v=HOGOEA_taK8. [Accessed: 07-Nov-2017]

[56] K. Halvorsen, “Zenta Robotic Creations | About Me,” 2015. [Online]. Available: http://zentasrobots.com/about-me/. [Accessed: 07-Nov-2017]

[57] K. Halvorsen, “Zenta Robotic Creations | The MX-Phoenix hexapod robot project,” 2016. [Online]. Available: http://zentasrobots.com/mx-phoenix-hexapod/. [Accessed: 07-Nov-2017]

[58] Zenta, “MX-Phoenix hexapod robot outdoor part II,” 2017. [Online]. Available: https://www.youtube.com/watch?v=aH07qF_bhgA. [Accessed: 07-Nov-2017]

[59] Zenta, “Project Thread: MX-Phoenix hexapod and some more,” *Trossen Robotics Community Forum*, 2016. [Online]. Available: http://forums.trossenrobotics.com/showthread.php?8497-MX-Phoenix-hexapod-and-some-more. [Accessed: 07-Nov-2017]

[60] A. Schneider and U. Schmucker, “Force Sensing for Multi-legged Walking Robots : Theory and Experiments – Part 1 : Overview and Force Sensing,” in *Mobile Robotics, Moving Intelligence*, J. Buchli, Ed. InTech, 2006, ch. 23, pp. 447–470 [Online]. Available: https://www.intechopen.com/books/mobile_robotics_moving_intelligence/force_sensing_for_multi-legged_walking_robots__theory_and_experiments_part_1__overview_and_force_sen. [Accessed: 12-Jan-2018]

[61] M. Palankar and L. Palmer, “A force threshold-based position controller for legged locomotion: Toward local leg feedback algorithms for robust walking on uneven terrain,” *Autonomous Robots*, vol. 38, no. 3, pp. 301–316, 2014.

[62] “Coxa - Entomologists’ glossary | Amateur Entomologists’ Society (AES).” [Online]. Available: https://www.amentsoc.org/insects/glossary/terms/coxa. [Accessed: 04-Dec-2017]

[63] M. Schilling, T. Hoinville, J. Schmitz, and H. Cruse, “Walknet, a bio-inspired controller for hexapod walking,” *Biological Cybernetics*, vol. 107, no. 4, pp. 397–419, 2013.

[64] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. New Jersey: Pearson Prentice Hall, 2005.

[65] J. Diebel, “Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors,” Stanford University, CA, USA, 2006.

[66] N. Trawny and S. I. Roumeliotis, “Indirect Kalman Filter for 3D Attitude Estimation: A Tutorial for Quaternion Algebra,” *Technical Report*, no. 2005–2, MARS Lab, University of Minnesota, 2005.

[67] E. Lubbe, D. Withey, and K. R. Uren, “State estimation for a hexapod robot,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 6286–6291.

[68] B. Siciliano and L. Villani, “From Indirect to Direct Force Control : A Roadmap for Enhanced Industrial Robots,” *Invited Paper, Robótica*, 2000.

[69] Open Source Robotics Foundation, “About ROS | ROS.org.” [Online]. Available: http://www.ros.org/about-ros/. [Accessed: 14-Nov-2017]

[70] Open Source Robotics Foundation, “History | ROS.org.” [Online]. Available: http://www.ros.org/history/. [Accessed: 14-Nov-2017]

[71] Open Source Robotics Foundation, “Is ROS For Me? | ROS.org.” [Online]. Available: http://www.ros.org/is-ros-for-me/. [Accessed: 14-Nov-2017]

[72] Open Source Robotics Foundation, “ROS/ Introduction | ROS Wiki.” [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed: 15-Nov-2017]

[73] Open Source Robotics Foundation, “Core Components | ROS.org.” [Online]. Available: http://www.ros.org/core-components/. [Accessed: 15-Nov-2017]

[74] Open Source Robotics Foundation, “Client Libraries | ROS Wiki.” [Online]. Available: http://wiki.ros.org/Client Libraries. [Accessed: 15-Nov-2017]

[75] Open Source Robotics Foundation, “Documentation | ROS Wiki.” [Online]. Available: http://wiki.ros.org/. [Accessed: 15-Nov-2017]

[76] Open Source Robotics Foundation, “ROS/ Concepts | ROS Wiki.” [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed: 15-Nov-2017]

[77] Open Source Robotics Foundation, “roscpp/ Overview/ Callbacks and Spinning | ROS Wiki.” [Online]. Available: http://wiki.ros.org/roscpp/Overview/Callbacks and Spinning. [Accessed: 15-Nov-2017]

[78] ROBOTIS, “AX-12/ AX-12+/ AX-12A | ROBOTIS e-Manual.” [Online]. Available: http://support.robotis.com/en/product/actuator/dynamixel/ax_series/dxl_ax_actuator.htm. [Accessed: 09-Nov-2017]

[79] Trossen Robotics, “ArbotiX-M Robocontroller.” [Online]. Available: http://www.trossenrobotics.com/p/arbotix-robot-controller.aspx. [Accessed: 13-Nov-2017]

[80] Trossen Robotics, “ArbotiX Commander v2.0 Kit.” [Online]. Available: http://www.trossenrobotics.com/p/arbotix-commander-gamepad-v2.aspx. [Accessed: 12-Nov-2017]

[81] “PhantomX Hexapod Phoenix Code.” [Online]. Available: http://learn.trossenrobotics.com/10-interbotix/crawlers/phantomx-hexapod/68-phoenix-code.html. [Accessed: 12-Nov-2017]

[82] LORD Corporation, “3DM-GX3® -25 Miniature Attitude Heading Reference System,” *Product Datasheet*, no. 8400–33, 2014. [Online]. Available: http://files.microstrain.com/3DM-GX3-25-Attitude-Heading-Reference-System-Data-Sheet.pdf. [Accessed: 09-Nov-2017]

[83] OptoForce, “3-Axis Force Sensor: OMD-20-SE-40N,” *Product Datasheet*, no. V2.2, 2016. [Online]. Available: https://optoforce.com/file-contents/omd-20-se-40n-datasheet-v2-1.2.pdf?v11. [Accessed: 10-Nov-2017]

[84] OptoForce, “Optical Force Sensors – Introduction To The Technology,” *White Paper*, 2015. [Online]. Available: https://optoforce.com/file-contents/optoforcewhitepaperopticalforcesensors-0.pdf?v11%0A. [Accessed: 10-Nov-2017]

[85] OptoForce, “3D Force Sensor (OMD).” [Online]. Available: https://optoforce.com/3d-force-sensor-omd. [Accessed: 10-Nov-2017]

[86] Xevelabs, “Product: USB2AX.” [Online]. Available: http://www.xevelabs.com/doku.php?id=product:usb2ax:usb2ax. [Accessed: 13-Nov-2017]

[87] Xevelabs, “USB2AX: Advanced instructions.” [Online]. Available: http://www.xevelabs.com/doku.php?id=product:usb2ax:advanced_instructions. [Accessed: 13-Nov-2017]

[88] Open Source Robotics Foundation, “roscpp/Overview/Timers | ROS Wiki.” [Online]. Available: http://wiki.ros.org/roscpp/Overview/Timers. [Accessed: 08-Dec-2017]

[89] J. Leibs and B. Gassend, “microstrain_3dmgx2_imu | ROS Wiki.” [Online]. Available: http://wiki.ros.org/microstrain_3dmgx2_imu. [Accessed: 13-Nov-2017]

[90] Shadow Robot Company, “optoforce,” *GitHub*. [Online]. Available: https://github.com/shadow-robot/optoforce. [Accessed: 13-Nov-2017]

[91] SparkFun Electronics, “Serial Communication.” [Online]. Available: https://learn.sparkfun.com/tutorials/serial-communication/wiring-and-hardware. [Accessed: 13-Nov-2017]

[92] Trossen Robotics, “6 Port AX/MX Power Hub.” [Online]. Available: http://www.trossenrobotics.com/6-port-ax-mx-power-hub. [Accessed: 13-Nov-2017]

[93] Transcend, “Accessories | HUB3.” [Online]. Available: https://www.transcend-info.com/Products/No-402. [Accessed: 13-Nov-2017]

[94] B. Friedland, *Control System Design: An Introduction to State-Space Methods*, 1st ed. Mineola, NY: Dover, 2005.

[95] R. C. Dorf and R. H. Bishop, *Modern Control Systems*, 12th ed. New Jersey: Pearson Prentice Hall, 2011.

[96] M. Kamenetsky, “Filtered Audio Demo,” *Course Notes: EE102*, Stanford University. [Online]. Available: https://web.stanford.edu/~boyd/ee102/conv_demo.pdf. [Accessed: 10-Dec-2017]

[97] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*, 2nd ed. West Sussex: Wiley, 2005.

[98] K. J. Åström, “PID Control,” in *Control System Design*, 2002, ch. 6, pp. 216–251 [Online]. Available: https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf. [Accessed: 16-Jan-2018]

[99] K. J. Åström and R. M. Murray, “PID Control,” in *Feedback Systems: An Introduction for Scientists and Engineers*, 1st ed., New Jersey: Princeton University Press, 2008, ch. 10, pp. 293–314 [Online]. Available: http://www.cds.caltech.edu/~murray/books/AM08/pdf/am08-pid_28Sep12.pdf. [Accessed: 20-May-2018]

[100] J. Pratt and G. Pratt, “Intuitive control of a planar bipedal walking robot,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998, pp. 2014–2021.

[101] ROBOTIS, “DYNAMIXEL.” [Online]. Available: http://en.robotis.com/index/product.php?cate_code=101010. [Accessed: 11-Dec-2017]

[102] Open Source Robotics Foundation, “roscpp/Overview/Time | ROS Wiki.” [Online]. Available: http://wiki.ros.org/roscpp/Overview/Time. [Accessed: 25-Nov-2017]

[103] A. Lauber, “Virtual Model Control on ALoF,” *Master-Thesis*, ETH Zurich, Switzerland, 2011.

[104] “vicon_bridge | ROS Wiki.” [Online]. Available: http://wiki.ros.org/vicon_bridge. [Accessed: 25-Nov-2017]

[105] B. Charrow, “matlab_rosbag,” *GitHub*. [Online]. Available: https://github.com/bcharrow/matlab_rosbag. [Accessed: 25-Nov-2017]

[106] “Eigen: Linear algebra and decompositions.” [Online]. Available: https://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html. [Accessed: 27-Nov-2017]

[107] T. Homberger, M. Bjelonic, N. Kottege, and P. V. K. Borges, “Terrain-dependant Control of Hexapod Robots using Vision,” in *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2016, pp. 92–102.

[108] F. Almeida, A. Lopes, and P. Abreu, “Force-Impedance Control : a new control strategy of robotic manipulators,” in *Recent Advances in Mechatronics*, 1st ed., O. Kaynak, S. Tosunoglu, and M. H. Ang, Jr., Eds. Singapore: Springer-Verlag, 1999, ch. 2, pp. 126–137.

[109] “NMISA National Metrology Institute of South Africa.” [Online]. Available: http://www.nmisa.org/. [Accessed: 10-Nov-2017]

[110] MicroStrain Inc., “3DM-GX3® -25 Single Byte Data Communications Protocol,” *DCP Manual*, no. 8500–15, 2012. [Online]. Available: http://files.microstrain.com/3DM-GX3-25 Single Byte Data Communications Protocol.pdf. [Accessed: 10-Dec-2017]

[111] J. Leibs and B. Gassend, “microstrain_3dmgx2_imu,” *GitHub*. [Online]. Available: https://github.com/ros-drivers/microstrain_3dmgx2_imu. [Accessed: 10-Dec-2017]

[112] Open Source Robotics Foundation, “tf API Overview: Coordinate Frames, Transforms, and TF | ROS Wiki.” [Online]. Available: http://wiki.ros.org/tf/Overview/Transformations. [Accessed: 10-Dec-2017]

[113] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. A. Hoepflinger, and R. Siegwart, “Control of Dynamic Gaits for a Quadrupedal Robot,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 3287–3292.

[114] W. H. Chen, G. J. Ren, J. H. Wang, and D. Liu, “An adaptive locomotion controller for a hexapod robot : CPG , kinematics and force feedback,” *Science China Information Sciences*, vol. 57, no. 11, pp. 1–18, 2014.

[115] C. Li, S. Dong, and G. Zhang, “Evaluation of the root-mean-square slope of 3D surface topography,” *International Journal of Machine Tools and Manufacture*, vol. 40, no. 3, pp. 445–454, 2000.

[116] A. Duparré, J. Ferre-Borrull, S. Gliech, G. Notni, J. Steinert, and J. M. Bennett, “Surface characterization techniques for determining the root-mean-square roughness and power spectral densities of optical components,” *Applied Optics*, vol. 41, no. 1, pp. 154–171, 2002.

[117] The Membranes Research Environment, “Surface Roughness.” [Online]. Available: http://membranes.edu.au/wiki/index.php/Surface_Roughness. [Accessed: 12-Dec-2017]

[118] The Planetary Society, “Wentworth (1922) grain size classification.” [Online]. Available: http://www.planetary.org/multimedia/space-images/charts/wentworth-1922-grain-size.html. [Accessed: 12-Dec-2017]

[119] T. D. Gillespie, *Fundamentals of Vehicle Dynamics*, 2nd ed. Warrendale, PA: Society of Automotive Engineers, 1992.

[120] C. M. Becker and P. S. Els, “Profiling of rough terrain,” *International Journal of Vehicle Design*, vol. 64, no. 2–4, pp. 240–261, 2014.

[121] A. S. Voloshina, A. D. Kuo, M. A. Daley, and D. P. Ferris, “Biomechanics and energetics of walking on uneven terrain,” *The Journal of Experimental Biology*, vol. 216, no. 21, pp. 3963–3970, 2013.

[122] S. Sponberg and R. J. Full, “Neuromechanical response of musculo-skeletal structures in cockroaches during rapid running on rough terrain,” *The Journal of Experimental Biology*, vol. 211, no. 3, pp. 433–446, 2008.

[123] Sondor, “SPX 45,” *Product Datasheet*, no. 13, 2014. [Online]. Available: https://www.sondor.co.za/files/fileuploads/file/005311fd-dfbd-4b1e-8afc-ad314de34454/spx45rev 13.pdf. [Accessed: 12-Dec-2017]

[124] Smooth On Inc., “Durometer Shore Hardness Scale.” [Online]. Available: https://www.smooth-on.com/page/durometer-shore-hardness-scale/. [Accessed: 08-Dec-2017]

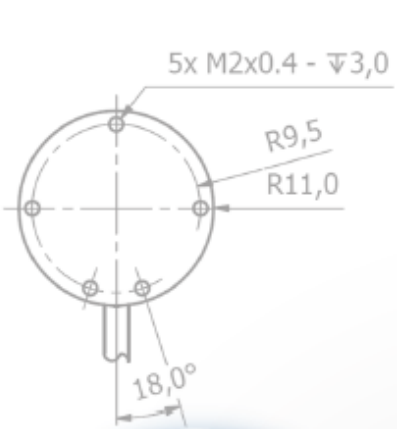
[125] T. Foote, E. Marder-Eppstein, and W. Meeussen, “tf | ROS Wiki.” [Online]. Available: http://wiki.ros.org/tf. [Accessed: 10-Dec-2017]

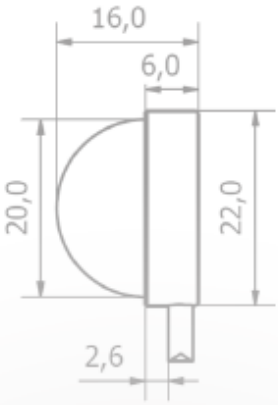
Appendix A: Foot Force Sensor

A-1 OptoForce 3-Axis Force Sensor OMD-20-SE-40N Datasheet [83]

OPTOFORCE
SENSING FLEXIBILITY

V2.2





3-Axis FORCE Sensor

OMD-20-SE-40N

Description:

OptoForce 3D sensors measure the magnitude and the direction of Fx, Fy, and Fz forces based purely on **optical principles**. Depending on the application, **semi-spherical** and **flat top** versions are available. We advise these sensors for low budget research programs and for measurements where torque sensing is unnecessary. Semi-spherical sensors are ideal as sensitive **fingertips** for humanoid robot hands, industrial **grippers**, harvesting robots, and due to its **high durability** there are various applications in the field of **medical robotics** (rehabilitation) and **advanced robotics** (e.g. exoskeletons) as well.

Benefits:

- Multi axis force measurement
- High resolution
- Highly adaptable product design
- Dust and water proof (IP65)
- High overload range
- Mechanical shock resistant
- Cost efficient solution
- Easy integration

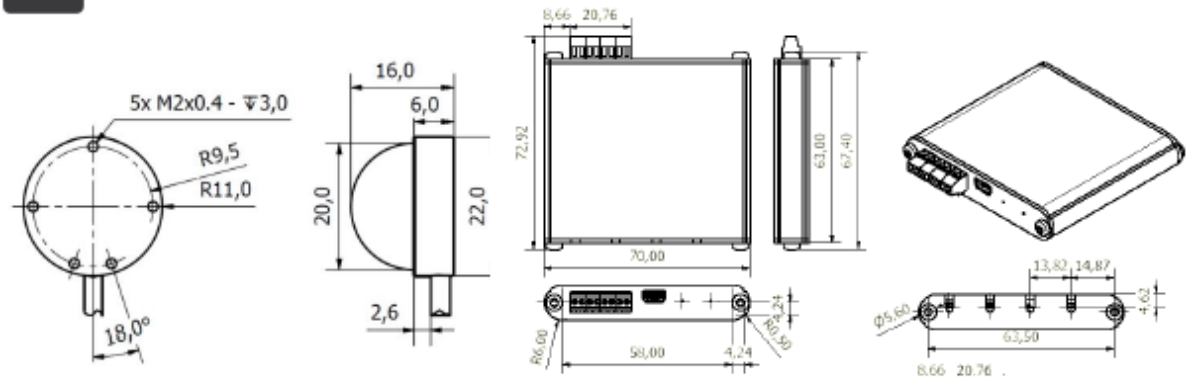
	Nominal Capacity	Typical Deformation
Fxy	± 10 N	± 1.5 mm
Fz – compression	40 N	2 mm

*: For F/T sensing kindly see our 6 axis datasheets

www.optoforce.com

info@optoforce.com

CAD DRAWINGS — OMD-20-SE-40N



Sensor

DAQ (Non-Detachable)

SPECIFICATIONS

Sensor Type	3 Axis Force Sensor		
Dimensions	Height x width x length		17 x 25 x 25 mm
Weight	With 1 m cable (without)		23 g (11 g)
	Fz Compression		Fxy
Nominal Capacity (N.C)	40 N		10 N
Single axis overload	200 %		200%
Full scale nonlinearity	2 %		2%
Resolution	2.5 mN		2 mN
Single axis deformation at N.C	1 mm		±1.5 mm
Crosstalk (typical)	<5%		
Hysteresis (measured on Fz axis, typical)	< 2 %		
Working temperature range			-40 °C - +80 °C
Power requirement	In continuous operation		10 mA

The semi-spherical sensors are only calibrated in Z+ direction
Parameters were measured at room temperature.

INTERFACE TYPES

USB	CAN	UART	Ethernet TCP/UDP
Maximum sampling frequency 1000 Hz			
Supported systems Windows; Linux; ROS; UR			

www.optoforce.com

info@optoforce.com

Specifications are subject to change without notice. Updated on December 15, 2016.

A-2 Force Sensor Sensitivity Reports

SENSITIVITY REPORT

Product name	: 3 AXIS FORCE SENSOR	Serial number	: ISE0A019
Model	: OMD-20-SE-40N	Interface type	: USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	15926	2,0 mm	398,16

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

SENSITIVITY REPORT

Product name :	3 AXIS FORCE SENSOR	Serial number :	ISE0A020
Model :	OMD-20-SE-40N	Interface type :	USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	16299	2,1 mm	407,47

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

SENSITIVITY REPORT

Product name	: 3 AXIS FORCE SENSOR	Serial number	: ISE0A023
Model	: OMD-20-SE-40N	Interface type	: USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	16264	1,9 mm	406,59

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

SENSITIVITY REPORT

Product name	:	3 AXIS FORCE SENSOR	Serial number :	ISE0A024
Model	:	OMD-20-SE-40N	Interface type :	USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	16202	1,9 mm	405,04

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

SENSITIVITY REPORT

Product name	:	3 AXIS FORCE SENSOR	Serial number	: ISE0A025
Model	:	OMD-20-SE-40N	Interface type	: USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	16267	1,9 mm	406,67

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

SENSITIVITY REPORT

Product name	: 3 AXIS FORCE SENSOR		Serial number	: ISE0A026
Model	: OMD-20-SE-40N		Interface type	: USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	16123	1,9 mm	403,07

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

SENSITIVITY REPORT

Product name	: 3 AXIS FORCE SENSOR	Serial number	: ISE0A027
Model	: OMD-20-SE-40N	Interface type	: USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	16282	1,9 mm	407,04

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

SENSITIVITY REPORT

Product name	: 3 AXIS FORCE SENSOR	Serial number	: ISE0A028
Model	: OMD-20-SE-40N	Interface type	: USB

	Nominal Capacity (N.C.)	Counts @ N.C.	Deformation (typical)	Counts/N
Fz (compression)	40 N	16212	1,9 mm	405,29

Inspection Date : 26.02.2016

OPTOFORCE
SENSING FLEXIBILITY

A-3 Force Sensor Testing

In order to assess the accuracy of the measurements provided by the OptoForce 3D foot force sensors, a simple dead weight test was carried out on each sensor, for compression in the z-direction. The setup involved balancing a cradle on the sensor, onto which weights could be placed to exert a compressive load on the sensor.

A-3.1 Apparatus and Test Setup

The following is a list of apparatus used for the testing procedure:

- Wall mounted platform onto which sensor is placed;
- Cradle;
- Weights, calibrated for a particular total force;
- OptoForce OMD-20-SE-40N 3D force sensors (×8);
- Data acquisition units (×2);
- USB cables – Mini Type B male to Type A male (×2);
- USB extension cable – Type A male to Type A female;
- Laptop with sensor drivers and OptoForce Data Visualization (ODV) software installed.

The main pieces of apparatus from the above list are indicated on the test setup displayed in Figure A.2.

While the weights were originally calibrated for the Mass Laboratory of the National Metrology Institute of South Africa (NMISA) [109], the tests were carried out in NMISA's Force Laboratory. Since the two labs are in close proximity at the Pretoria campus of CSIR, the geographical location should have a negligible effect on the weight. Furthermore, during personal e-mail communication in September 2016, Corne Gouws (cgouws@nmisa.org), from the Force Laboratory at NMISA, explained that the Mass Laboratory uses Primary Standards for calibration which are of high accuracy and quality, having controlled conditions such as a sealed room with climate control. The Force Laboratory, on the other hand, uses Working Standards for their tests, which, although being typically of high accuracy, are less accurate than Primary Standards. The differences in weight created by the geographical differences are therefore insignificant when compared to the differences in calibration standards used in the two laboratories.

It should also be noted that the weights were wiped down with tissue paper before testing, to remove any dirt, dust and oils which could affect their weight. In addition, they were handled with clean gloves at all times to ensure no oils were transferred to them during testing. Taking the above factors into account, it can be concluded that the accuracy of the weights was sufficient for the purposes of this

basic test procedure. A collection of these weights, used by the Force Laboratory at NMISA, is depicted in Figure A.1.

Figure A.1: Weights of various sizes, used for calibration and testing at NMISA's Force Laboratory

The setup for the force sensor testing is depicted in Figure A.2 below, with the cradle suspended from one of the sensors and 40 N of weight loaded onto it (see Figure A.3 for close up view of sensor).

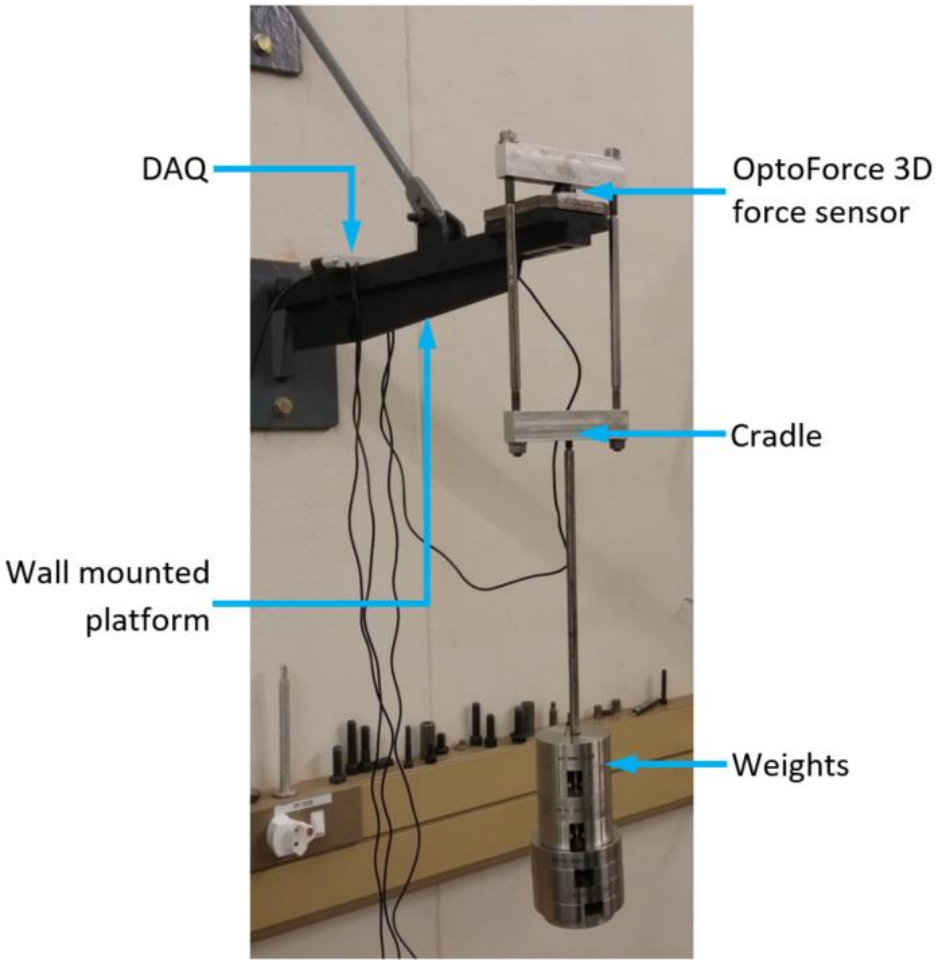


Figure A.2: Force sensor test setup with main apparatus labelled

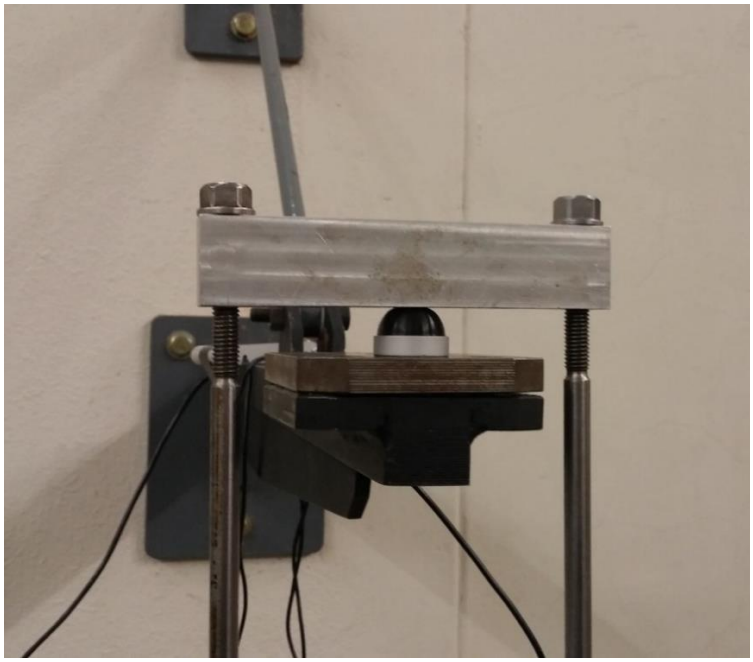


Figure A.3: Close up view of loaded cradle balanced on OptoForce sensor

Given that the sensors can be safely loaded to 200% of nominal capacity, the additional weight of the cradle (approximately 5N) poses no problem to the sensor and was simply zeroed, or "tared", out before applying additional load.

A-3.2 Testing Procedure

Each DAQ was connected individually to the laptop computer through USB cable and the ODV software was started up. The default sampling and filter frequencies of 100 Hz and 15 Hz, respectively, were used, and the following testing procedure was carried out for each of the eight sensors:

Step 1: Place sensor on platform and balance cradle on sensor surface. Take note to align the small hole on the underside of the cradle bar (see Figure A.4) as close to the centre of the sensor surface as possible.

Step 2: Warm up the sensor with a preload, by loading it up to 40 N, in two steps of 20 N each, then unloading. Repeat three times.

Step 3: Zero, or tare, the measurements, to remove the cradle weight and measurement offset.

Step 4: Load up to 40 N, in two steps of 20 N each. When loading, wait long enough for measurement to settle at each step, accounting for factors such as hysteresis and swinging of cradle. Unload sensor once measurement has settled with 40 N load. Ensure the ODV software is recording measurement data throughout this step.

Step 5: Once sensor measurements have settled, zero measurement and repeat step 4 once more, in order to test the *repeatability* of the sensor measurements.

Step 6: Rotate sensor by 180°, in the xy-plane, and balance cradle on sensor again. “Seat” sensor by preloading again, carrying out the procedure in step 2 (only once, no need to repeat three times), then zero the measurement.

Step 7: Load the sensor again as in step 4 (repeating the test with the sensor rotated by 180° tests the *reproducibility* of the sensor measurements).

In order to analyse the sensor accuracy in between the loading steps of 20 N and 40 N, one of the sensors (serial number ISE0A024) was loaded in eight steps, of 5 N each, during test Steps 5 and 7 in the above procedure.

All the data recorded from the tests above were processed and analysed using MATLAB, as described in the following section.

Figure A.4: Underside of cradle bar, showing chamfered hole (circled in red) where cradle is to be rested on force sensor

A-3.3 Post-processing of Measured Data

The OptoForce sensor measurements are sent out as integer ticks by the DAQs and recorded in this manner by the ODV software. In order to convert these recorded data to force measurements, they were divided, during post-processing, by the sensitivity values provided by the manufacturer for each individual sensor, resulting in force data for compressive z-direction measurements such as those illustrated in the next section.

As alluded to in the testing procedure above, as well as visible in the plots of the next section, the force measurements are not constant over time for a particular loading condition. Some reasons, such as swinging of the cradle, were already discussed, but another factor is the constant presence of small

magnitudes of measurement noise (even with the DAQ filtering the data), resulting in the measurement never being a constant value. Thus, rather than picking a single data point as the force measurement at a particular load, a MATLAB script was setup, to allow the user to select a range over which the measurements were to be used for each loading step, and the mean values of these ranges were calculated as the force measurements for the various loads. When selecting measurement ranges, care was taken not to include regions in which the measurements were varying significantly (for example, where the measurement exhibited small oscillations from the swinging of the cradle), rather using measurements which were well settled and fairly constant.

From the z-direction measurement value calculated above, the corresponding error value for each loading condition was calculated as the difference between measured value and applied load. Percentage error was calculated by taking the error value as a percentage of the applied load, while full scale error percentage was calculated from the error value as a percentage of the full scale, nominal sensor capacity of 40 N. These calculations are represented mathematically by the following equations:

$$Error = Measured\ Force - Load \tag{A.1}$$

$$Error\ [\%] = \frac{Error}{Load} \times 100 \tag{A.2}$$

$$Full\ Scale\ Error\ [\%] = \frac{Error}{40} \times 100 \tag{A.3}$$

All calculated measurements and error values were tabulated, along with the applied loads, and written to Excel spreadsheet files for each sensor, by the same MATLAB script. These results are presented in the section which follows.

A-3.4 Test Results

Compressive z-direction force measurement results of particular note are presented in this section, along with summaries of the maximum errors obtained across the various tests performed with the eight sensors. Detailed results for each run of the multitude of tests are reported in Appendix A-4.

A-3.4.1 Eight Step Test

Plots of the z-direction force measurements using sensor ISE0A024 are presented in Figures A.5 to A.7, while the mean measured forces and corresponding errors, as calculated with Equations (A.1) – (A.3), are presented in Tables A.1 to A.3 for the three runs of the test. It should be noted that the two repeatability tests are referred to as “run 1” and “run 2,” while the reproducibility test, in which the sensor was rotated by 180°, is referred to as “run 180,” for clarity. Furthermore, all values presented

in the tables have been rounded to two decimal places and are thus not the exact values used during the calculations.

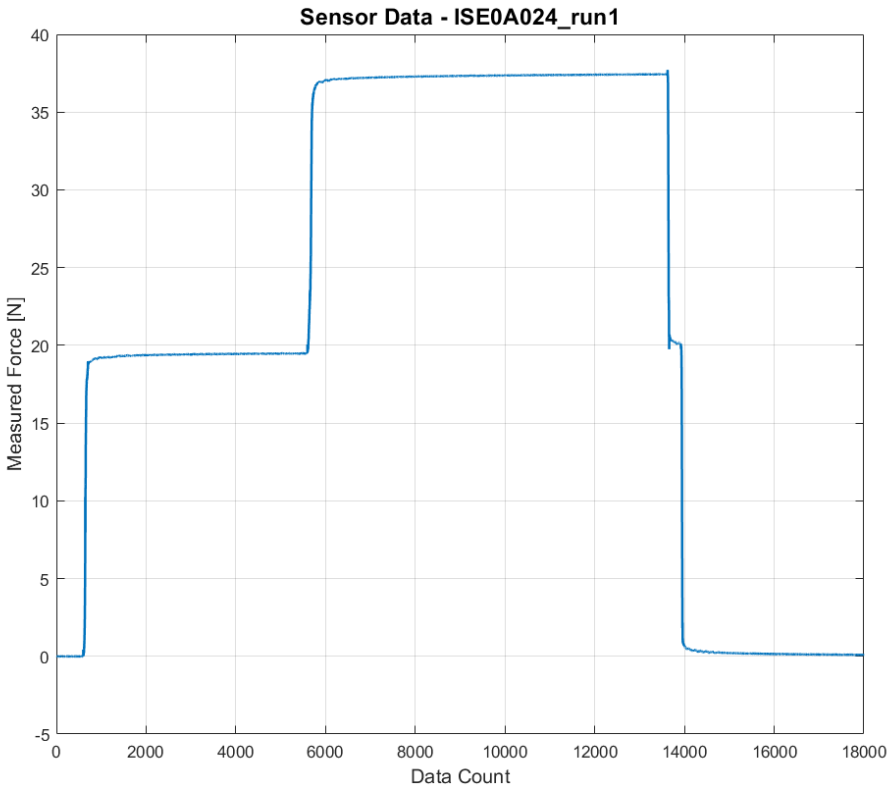


Figure A.5: Force sensor (ISE0A024) test measurements – "run 1" (using only 2 loading steps)

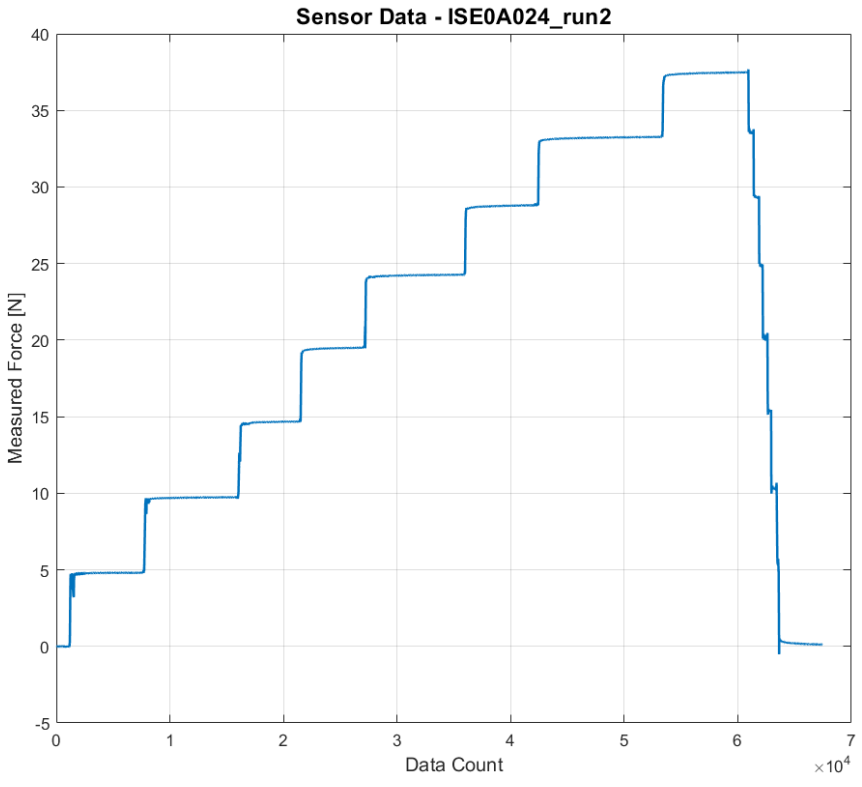


Figure A.6: Force sensor (ISE0A024) test measurements – "run 2" (repeatability test; using 8 loading steps)

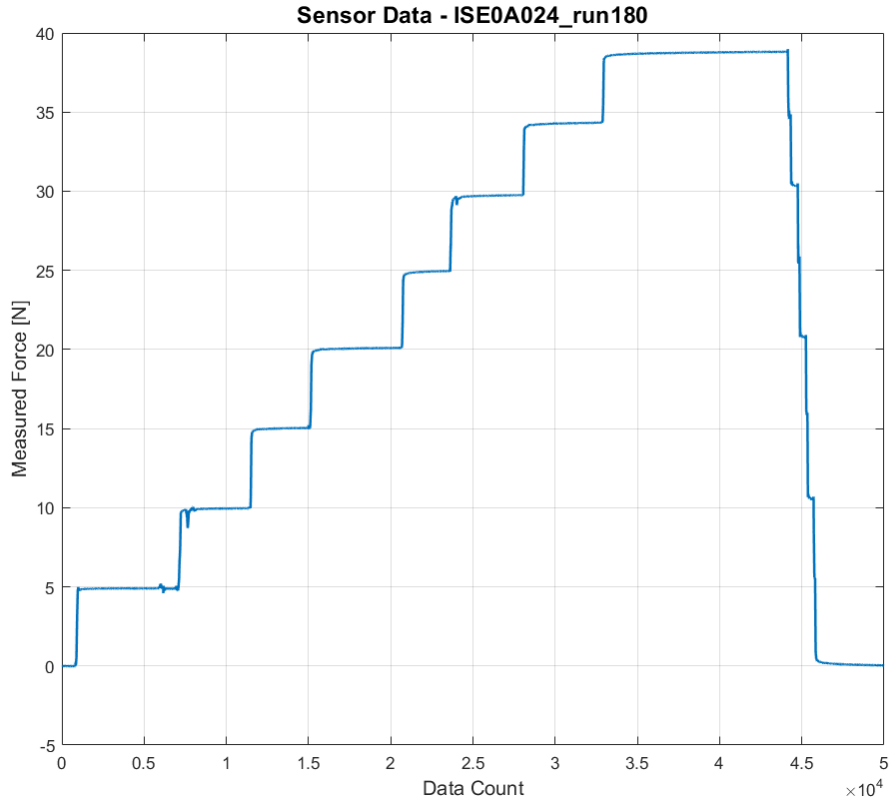


Figure A.7: Force sensor (ISE0A024) test measurements – "run 180" (reproducibility test; using 8 loading steps)

Table A.1: Force sensor (ISE0A024) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.47	-0.53	-2.63	-1.31
40	37.43	-2.57	-6.42	-6.42

Table A.2: Force sensor (ISE0A024) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
5	4.81	-0.19	-3.84	-0.48
10	9.74	-0.26	-2.65	-0.66
15	14.68	-0.32	-2.16	-0.81
20	19.50	-0.50	-2.51	-1.25
25	24.27	-0.73	-2.92	-1.82
30	28.80	-1.20	-4.00	-3.00
35	33.26	-1.74	-4.97	-4.35
40	37.49	-2.51	-6.29	-6.29

Table A.3: Force sensor (ISE0A024) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
5	4.90	-0.10	-1.97	-0.25
10	9.96	-0.04	-0.40	-0.10
15	15.04	0.04	0.26	0.10
20	20.10	0.10	0.48	0.24
25	24.96	-0.04	-0.17	-0.11
30	29.75	-0.25	-0.84	-0.63
35	34.33	-0.67	-1.92	-1.68
40	38.80	-1.20	-2.99	-2.99

The results of this test are of particular interest, as they show the sensor behaviour across the measurement range with higher resolution than the tests using only two loading steps. However, the plots and results of these runs (for 20 N and 40 N loads) are quite typical for the entire set of sensors. Therefore, only the maximum errors for the eight sensors are summarized in the following section, while detailed measurement results can be found tabulated in Appendix A-4.

A-3.4.2 Results Summary

Table A.4 summarizes the largest errors observed for the eight sensors across the three runs of the tests, for loads of 20 N and 40 N.

Table A.4: Maximum measurement errors of force sensor tests for loads of 20 N and 40 N

Sensor	Max. Error [N]		Max. Error [%]		Max. Full Scale Error [%]	
	20 N	40 N	20 N	40 N	20 N	40 N
ISE0A019	-1.65	-4.56	-8.26	-11.41	-4.13	-11.41
ISE0A020	-1.22	-3.65	-6.09	-9.13	-3.05	-9.13
ISE0A023	-0.27	-2.42	-1.35	-6.04	-0.68	-6.04
ISE0A024	-0.53	-2.57	-2.63	-6.42	-1.31	-6.42
ISE0A025	-0.34	-2.02	-1.70	-5.04	-0.85	-5.04
ISE0A026	0.54	-1.84	2.69	-4.60	1.35	-4.60
ISE0A027	-0.60	-3.15	-2.99	-7.89	-1.49	-7.89
ISE0A028	-0.39	-2.53	-1.94	-6.32	-0.97	-6.32

A-3.5 Discussion of Results

The plots in Figures A.5 to A.7 show that the sensors typically measure the load values quite accurately. Table A.4 indicates that the error magnitudes are generally below 3% for 20 N loads and 7% for 40 N loads on most of the sensors, although the maximum error magnitudes observed are 8.26% at 20 N and 11.41% at 40 N. Full-scale error percentage magnitudes usually remain below 1.5% at 20 N for the majority of sensors, reaching maximums of 4.13% at 20 N and 11.41% at 40 N, for sensor ISE0A019.

A consistent observation across the sensors is that these error values appear to increase proportionally with the applied load, which is also demonstrated more clearly in Figure A.6, Figure A.7, Table A.2, and Table A.3, where the load is increased in smaller steps.

While these error values come across as being larger than those quoted by the manufacturer, as well as undesirably becoming larger as the load approaches the nominal capacity of the sensors (where they are expected to be more accurate), it is important to keep in mind the conditions under which these tests were performed. Firstly, the experimental setup was not perfect, with the cradle not necessarily always being perfectly lined up with the centre of the sensor. The small hole in the cradle bar (Figure A.4) is also not perfectly in the centre of the cradle, resulting in small offsets of the applied load from the centre of the sensor. This results in load components being exerted in the x- and y-directions of the sensor, to counter the moments occurring at the sensor contact surface from the load offset. Another factor which contributed to the load offsets and moments was the fact that the gaps in the loading weights could not always be perfectly counterbalanced between each other. Thus, the measured z-component of force does not account for the full load on the sensor. Since the sensor measurements had to be zeroed during testing, the small x- and y-components were not even reliable enough to be used to correct the effects of load offsets.

This explanation is further corroborated by the fact that the results of the repeatability runs are quite consistent, while in some cases the reproducibility runs exhibited slightly different results. The reason for this is that rotating the sensors required the cradle to be repositioned and seated on the sensors again. Consequently, the cradle may not have always been balanced in exactly the same position in both cases, causing differences in the loading conditions, resulting in slight differences in measurements for these cases.

Additionally, as the load is increased, the moments and offset effects increase, resulting in more significant force components away from the z-direction, along with more significant, undesirable deformations of the sensor contact surface, explaining the increase in error values with increasing loads. This is further supported by the fact that in the few cases where the cradle was managed to be aligned quite well and the load acted almost completely in the z-direction of the sensor, such as in

"run 180" of the test with sensor ISE0A024, the results were very accurate and the errors did not grow as significantly with load as in other runs. Thus, the sensitivity slopes for each sensor, provided by the manufacturer, are likely quite accurate, with imperfections in the tests discussed here being a more probable cause of the observed deviations from the expected measurements.

Nevertheless, some noteworthy findings were made from this investigation. For instance, the results for sensors ISE0A019 and ISE0A020 exhibited significantly larger errors than the rest of the sensors. These two sensors seem to have slightly softer contact surfaces than the rest and thus deform slightly more under load, amplifying the effects of loading misalignments in these tests. Consequently, it was decided to use the other six sensors on the PhantomX, for more consistent sensor behaviour and measurements across the six feet.

Furthermore, the results of the repeatability tests were observed to be quite consistent for most of the sensors. The differences in the results for the reproducibility runs were also generally small, with reasons for the larger differences observed in some sensors having been discussed above. Taking these factors into account, the findings of these tests indicate that the force sensors provide sufficiently repeatable and reproducible measurements.

A-3.6 Conclusion

Overall, the results obtained from the dead weight experiments are reasonably satisfactory, exhibiting sufficiently small error magnitudes between the measurements and applied loads. Although the manufacturer claims a maximum full-scale error of 2%, the results obtained in these tests suggest slightly larger measurement errors. However, the effects of inaccuracies and imperfections in the experiment dominate the errors in the force measurements. These experimental imperfections are not detrimental to the tests performed, as it should be kept in mind that these tests were simply meant to be a quick check of the accuracy of the sensors and the provided sensitivity values, rather than a full sensor calibration procedure.

Taking the above into account, the observed full-scale error magnitudes, generally being well below 7%, are acceptable. They are definitely not large enough to invalidate or discard the sensitivities given by the manufacturer (which would have been calibrated using a much more comprehensive and accurate procedure and setup) and perform a recalibration of the sensors. Furthermore, for the application at hand, the sensor accuracy indicated by the results of the above tests is certainly sufficient. It can, therefore, be concluded that the force measurements provided by the OptoForce 3D Force sensors, using the manufacturer provided sensitivity values for each sensor, are sufficiently accurate for use on the PhantomX robot platform.

A-4 Results of Force Sensor Tests

ISE0A019

Table A-5: Force sensor (ISE0A019) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.15	-0.85	-4.24	-2.12
40	37.28	-2.72	-6.81	-6.81

Table A-6: Force sensor (ISE0A019) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.13	-0.87	-4.35	-2.18
40	37.27	-2.73	-6.83	-6.83

Table A-7: Force sensor (ISE0A019) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	18.35	-1.65	-8.26	-4.13
40	35.44	-4.56	-11.41	-11.41

ISE0A020

Table A-8: Force sensor (ISE0A020) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.10	-0.90	-4.49	-2.25
40	36.80	-3.20	-8.00	-8.00

Table A-9: Force sensor (ISE0A020) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.08	-0.92	-4.60	-2.30
40	36.91	-3.09	-7.73	-7.73

Table A-10: Force sensor (ISE0A020) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	18.78	-1.22	-6.09	-3.05
40	36.35	-3.65	-9.13	-9.13

ISE0A023

Table A-11: Force sensor (ISE0A023) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.74	-0.26	-1.28	-0.64
40	37.66	-2.34	-5.84	-5.84

Table A-12: Force sensor (ISE0A023) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.73	-0.27	-1.35	-0.68
40	37.58	-2.42	-6.04	-6.04

Table A-13: Force sensor (ISE0A023) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.87	-0.13	-0.63	-0.32
40	37.82	-2.18	-5.44	-5.44

ISE0A024

Table A-14: Force sensor (ISE0A024) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.47	-0.53	-2.63	-1.31
40	37.43	-2.57	-6.42	-6.42

Table A-15: Force sensor (ISE0A024) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
5	4.81	-0.19	-3.84	-0.48
10	9.74	-0.26	-2.65	-0.66
15	14.68	-0.32	-2.16	-0.81
20	19.50	-0.50	-2.51	-1.25
25	24.27	-0.73	-2.92	-1.82
30	28.80	-1.20	-4.00	-3.00
35	33.26	-1.74	-4.97	-4.35
40	37.49	-2.51	-6.29	-6.29

Table A-16: Force sensor (ISE0A024) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
5	4.90	-0.10	-1.97	-0.25
10	9.96	-0.04	-0.40	-0.10
15	15.04	0.04	0.26	0.10
20	20.10	0.10	0.48	0.24
25	24.96	-0.04	-0.17	-0.11
30	29.75	-0.25	-0.84	-0.63
35	34.33	-0.67	-1.92	-1.68
40	38.80	-1.20	-2.99	-2.99

ISE0A025

Table A-17: Force sensor (ISE0A025) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.66	-0.34	-1.70	-0.85
40	38.06	-1.94	-4.85	-0.85

Table A-18: Force sensor (ISE0A025) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.66	-0.34	-1.69	-0.85
40	37.98	-2.02	-5.04	-5.04

Table A-19: Force sensor (ISE0A025) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	20.08	0.08	0.40	0.20
40	38.84	-1.16	-2.90	-2.90

ISE0A026

Table A-20: Force sensor (ISE0A026) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	20.10	0.10	0.50	0.25
40	38.16	-1.84	-4.60	-4.60

Table A-21: Force sensor (ISE0A026) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	20.14	0.14	0.72	0.36
40	38.17	-1.83	-4.58	-4.58

Table A-22: Force sensor (ISE0A026) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	20.54	0.54	2.69	1.35
40	38.92	-1.08	-2.71	-2.71

ISE0A027

Table A-23: Force sensor (ISE0A027) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.41	-0.59	-2.94	-1.47
40	36.85	-3.15	-7.89	-7.89

Table A-24: Force sensor (ISE0A027) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.40	-0.60	-2.99	-1.49
40	36.86	-3.14	-7.85	-7.85

Table A-25: Force sensor (ISE0A027) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.65	-0.35	-1.74	-0.87
40	37.80	-2.20	-5.50	-5.50

ISE0A028

Table A-26: Force sensor (ISE0A028) test measurement results – "run 1"

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.93	-0.07	-0.33	-0.17
40	37.96	-2.04	-5.09	-5.09

Table A-27: Force sensor (ISE0A028) test measurement results – "run 2" (repeatability test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.95	-0.05	-0.26	-0.13
40	37.96	-2.04	-5.10	-5.10

Table A-28: Force sensor (ISE0A028) test measurement results – "run 180" (reproducibility test)

Load [N]	Measured Force [N]	Error [N]	Error [%]	Full Scale Error [%]
20	19.61	-0.39	-1.94	-0.97
40	37.47	-2.53	-6.32	-6.32

Appendix B: New Foot for PhantomX MK-II

B-1 Design of New Foot for PhantomX MK-II

A computer-aided design (CAD) model of the original foot of the PhantomX MK-II is displayed in Figure B.1.

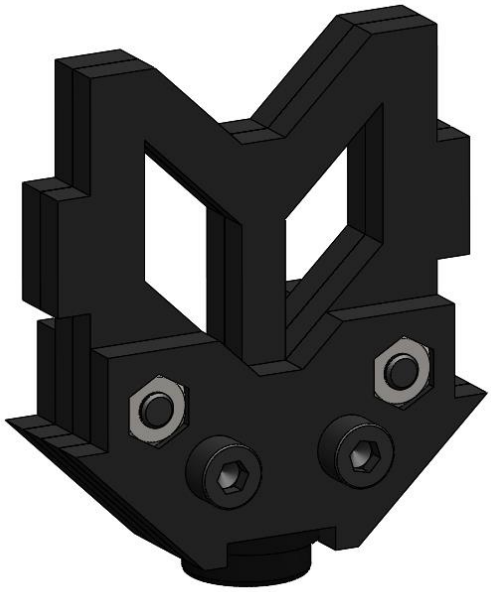


Figure B.1: CAD model of original foot assembly of PhantomX MK-II hexapod

These original feet are assembled from Plexiglas plates and have a small rubber pad, or “toe,” on the end, where the foot contacts the ground. Clearly, there is no means to mount an OptoForce 3D force sensor to this foot, nor is there a straightforward manner in which the foot could be minimally modified to integrate the sensor. New feet were thus required to be designed, onto which the OptoForce sensors could be mounted.

An important consideration in the design was the ability to mount and unmount the sensors from the robot’s leg. In Figure B.2 it can be seen that the original foot is held on the leg by slots in the two tibia plates. The foot is, therefore, fixed to the tibia segment of the leg and can only be removed by disassembling this whole leg segment. This would certainly have to be addressed in the design, allowing for the sensors to be separated from the leg whenever required, without having to disassemble the tibia segment.

(a) (b)

Figure B.2: Model of original PhantomX MK-II foot (a) mounted in tibia; (b) partially exploded view showing slot in tibia plate which holds foot in place

It was thus decided to separate the foot into two parts: a *centreplate*, which is fixed into the tibia segment of the leg, and a *sensor mount*, which is fastened onto the centreplate after assembling the tibia, allowing it to be detached from the leg at any time. CAD models of the two parts are depicted in Figures B.3 and B.4, while the foot assembly, including the OptoForce sensor, is displayed in Figure B.5. Figure B.6 also depicts the ability to detach the force sensor from the robot leg without disassembling the entire tibia segment. Detailed drawings of the centreplate and sensor mount, as well as the foot assembly, are provided in Appendix B-2.

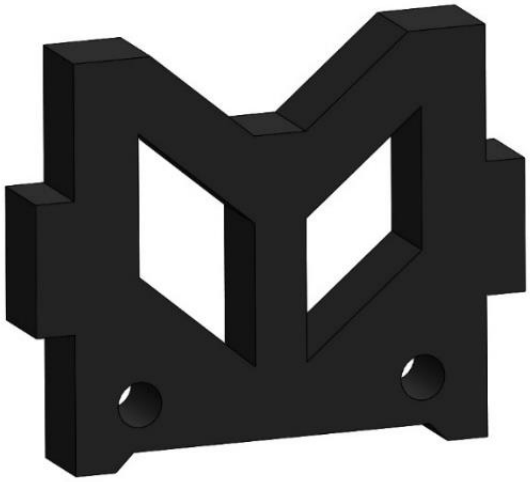


Figure B.3: Centreplate part of new PhantomX foot

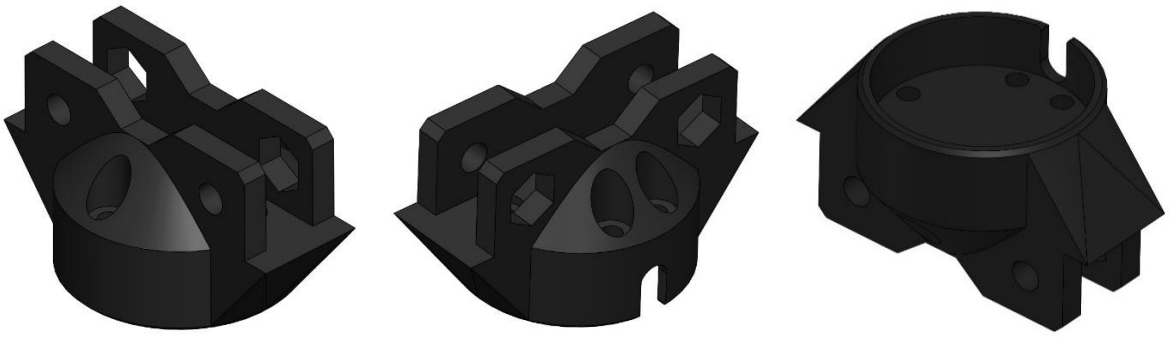
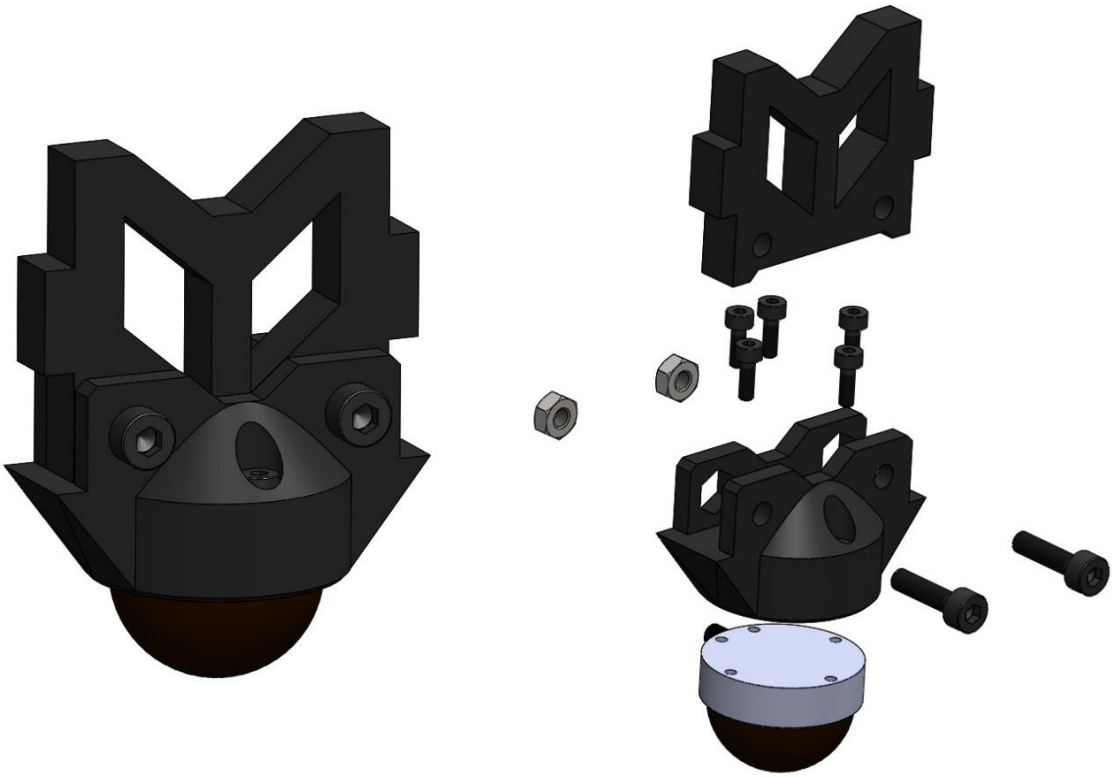
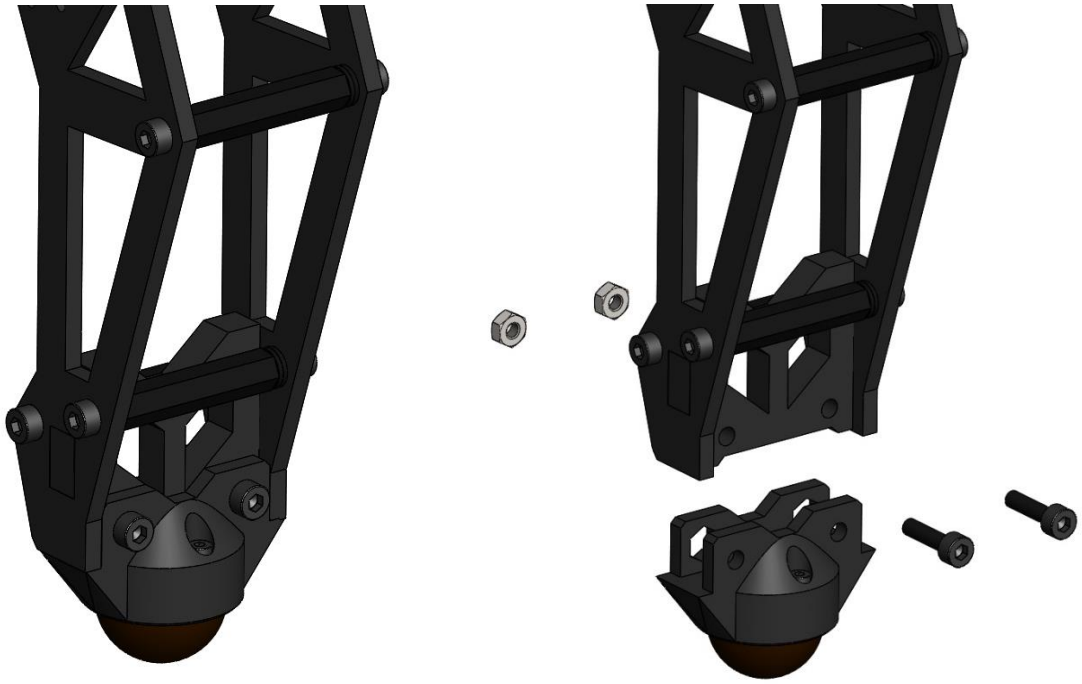


Figure B.4: Multiple views of sensor mount part of new PhantomX foot



(a) (b)

Figure B.5: Assembly of new PhantomX foot with OptoForce sensor: (a) fully assembled view; (b) exploded view



(a) (b)

Figure B.6: Models of OptoForce sensor (a) mounted onto robot foot, and (b) being detached from leg without disassembling entire tibia segment

These newly designed feet were manufactured by 3D printing, using plastic materials with similar properties to ABS plastic. Images of the feet attached to the PhantomX legs, with the OptoForce sensors mounted, can be seen in Figures B.7 and 4.9.

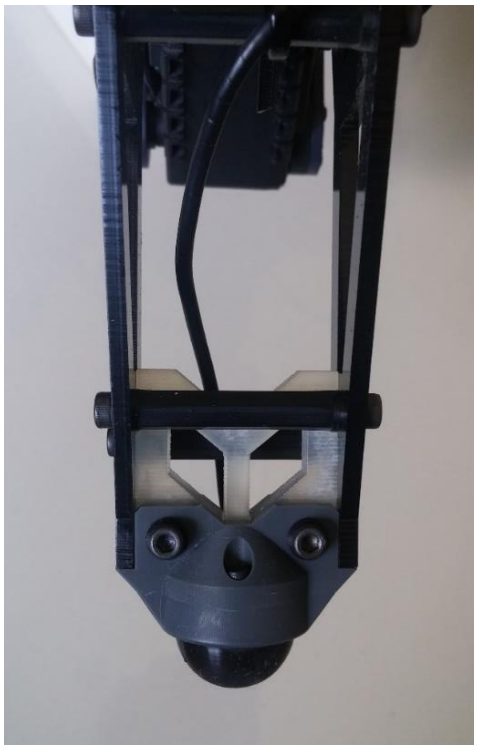
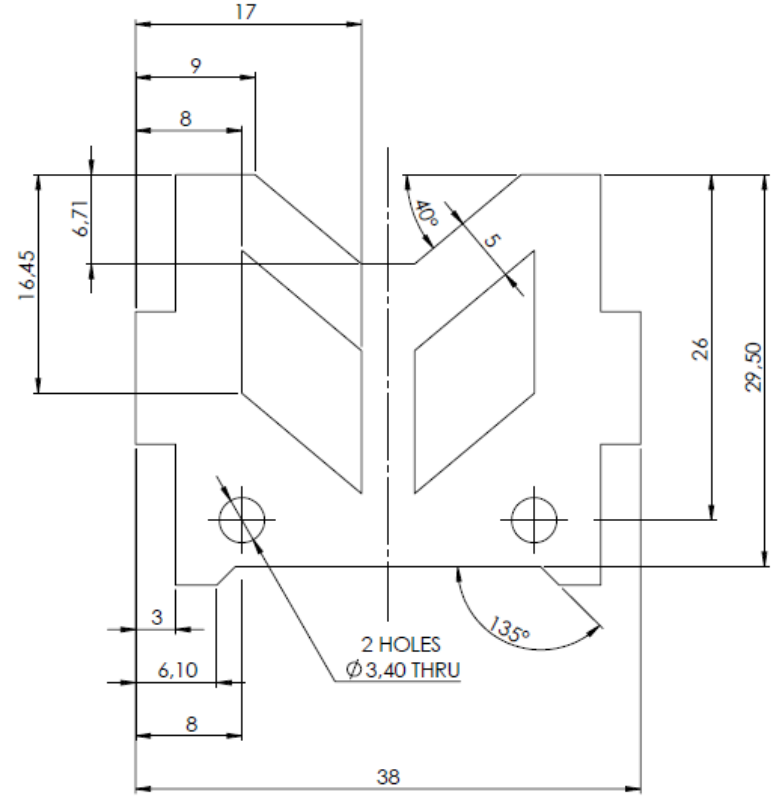
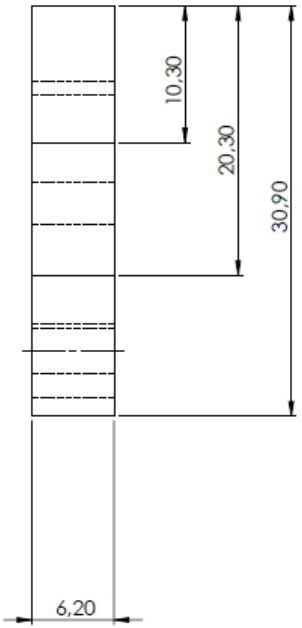


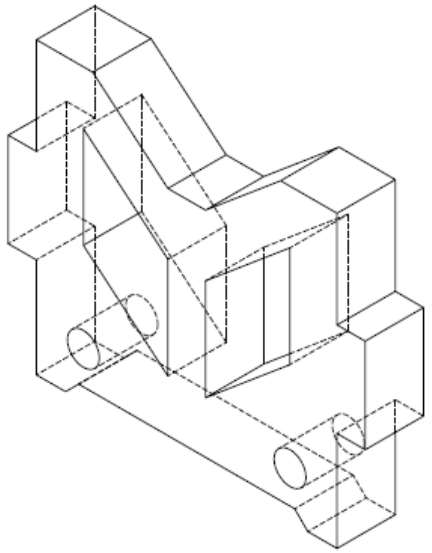
Figure B.7: Frontal view of new PhantomX foot with OptoForce sensor, mounted onto tibia segment

B-2 Manufacturing Drawings of Newly Designed Foot

Manufacturing drawings of the centreplate and sensor mount parts, as well as the foot assembly, are presented below.







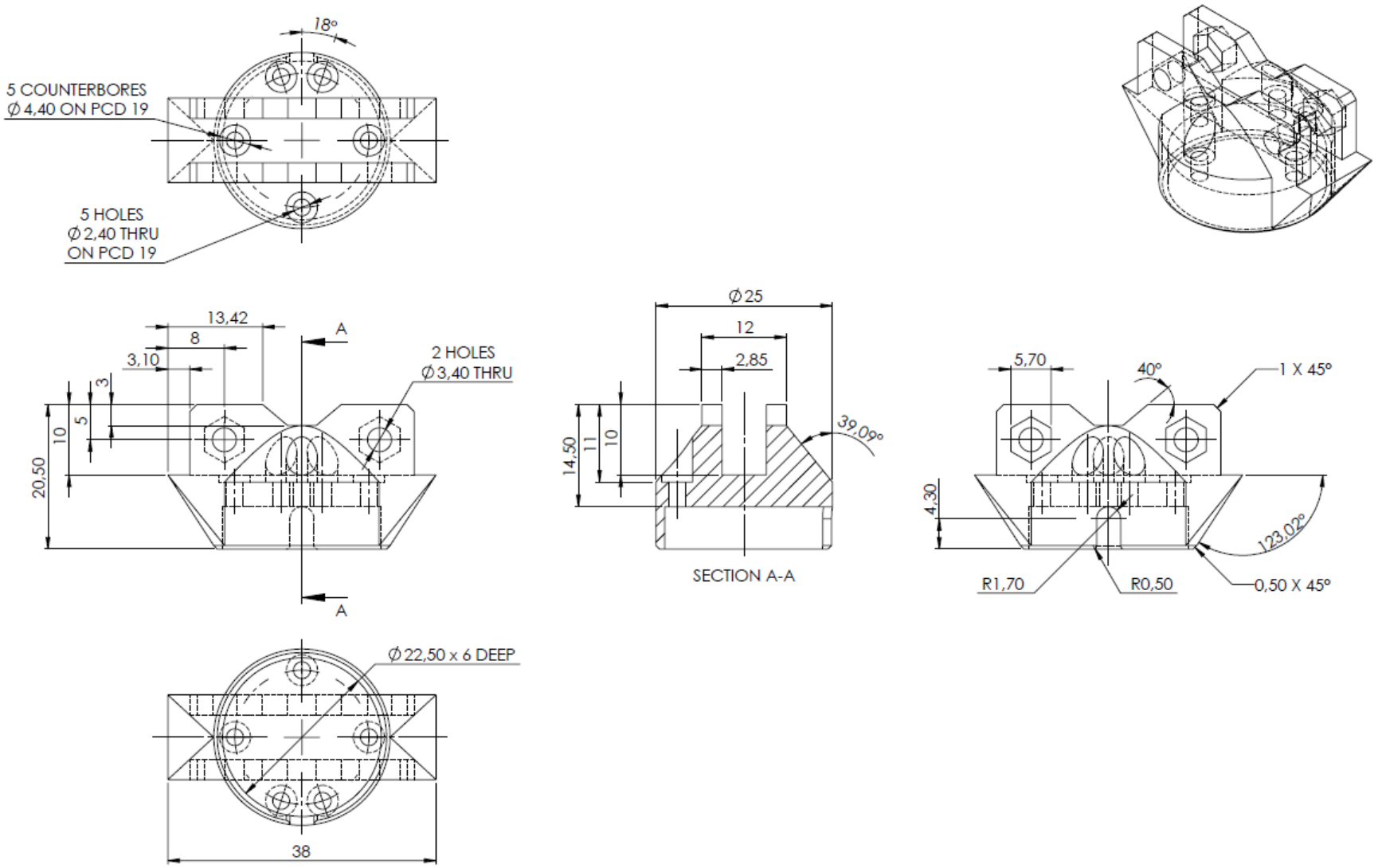
1. I understand what plagiarism is and am aware of the University's policy in this regard.
2. I declare that this drawing is my own original work.
3. I have not used another student's past work to hand in as my own.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.

.....

TOLERANCES FROM - TO	+/-
0 - 6	0.1
6 - 30	0.2
30 - 100	0.3
100 - 300	0.5
300 - 1000	0.8
1000 - 3000	1.2
3000 - PLUS	2.0
ANGLES	1°

ITEM NR / ITEM NO	AANTAL / QUANTITY	BESKRYWING / DESCRIPTION	MATERIAAL / MATERIAL	ONDERDEEL NR / PART NR
		FOOT CENTREPLATE		

VAN / SURNAME	TIKAM	VOORLETTERS / INITIALS	M	TITEL / TITLE	MEng
STUDENTE NO / STUDENTE NR	10055747	RIGTING / DISCIPLINE	M	TEKENING NR / DRAWING NR	
PROJEKSIE / PROJECTION	UNIVERSITEIT VAN PRETORIA / UNIVERSITY OF PRETORIA	A3	SKAAL / SCALE 3:1	DATUM / DATE	

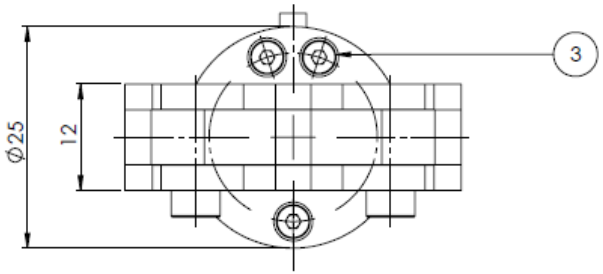


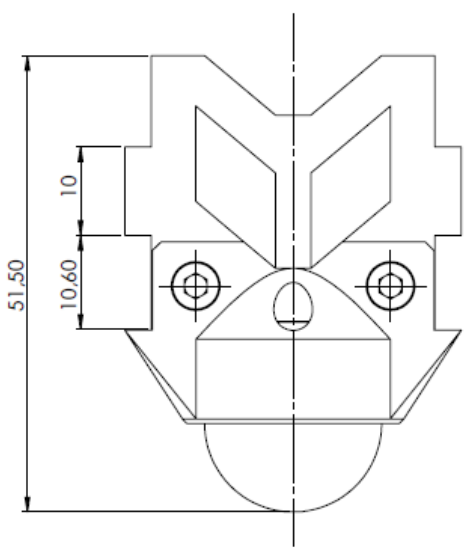
1. I understand what plagiarism is and am aware of the University's policy in this regard.
2. I declare that this drawing is my own original work.
3. I have not used another student's past work to hand in as my own.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.

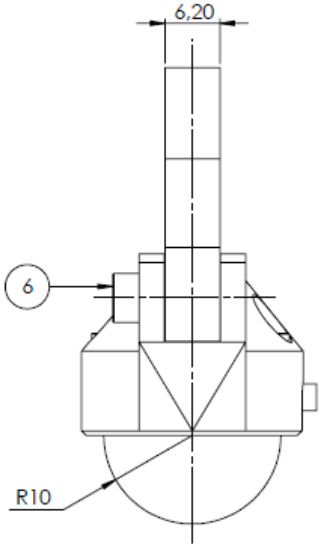
TOLERANCES FROM - TO	+/-
0 - 6	0.1
6 - 30	0.2
30 - 100	0.3
100 - 300	0.5
300 - 1000	0.8
1000 - 3000	1.2
3000 - PLUS	2.0
ANGLES	1°

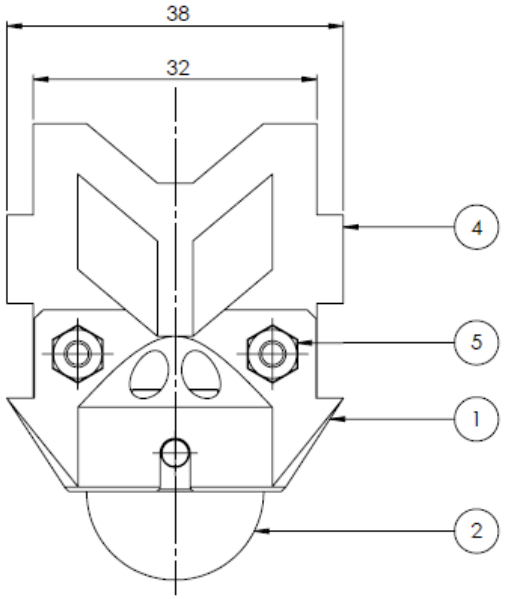
ITEM NR / ITEM NO	AANTAL / QUANTITY	BESKRYWING / DESCRIPTION	MATERIAAL / MATERIAL	ONDERDEEL NR / PART NR
		SENSOR MOUNT		

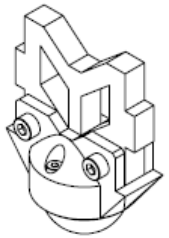
VAN / SURNAME	TIKAM	VOORLETTERS / INITIALS	M	TITEL / TITLE	MEng
STUDENTE NO / STUDENTE NR	10055747	RIGTING / DISCIPLINE	M	TEKENING NR / DRAWING NR	
PROJEKSIE / PROJECTION	UNIVERSITEIT VAN PRETORIA / UNIVERSITY OF PRETORIA	A3	SKAAL / SCALE 2:1	DATUM / DATE	











ITEM NO.	DESCRIPTION	QTY.
1	SENSOR MOUNT	1
2	OPTOFORCE 3D FORCE SENSOR OMD-20-SE-40N	1
3	HEX SOCKET HD CAP SCREW M2 x 6	5
4	FOOT CENTREPLATE	1
5	HEX NUT M3	2
6	HEX SOCKET HD CAP SCREW M3 x 12	2

1. I understand what plagiarism is and am aware of the University's policy in this regard.
2. I declare that this drawing is my own original work.
3. I have not used another student's past work to hand in as my own.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.

TOLERANCES FROM - TO	+/-
0 - 6	0.1
6 - 30	0.2
30 - 100	0.3
100 - 300	0.5
300 - 1000	0.8
1000 - 3000	1.2
3000 - PLUS	2.0
ANGLES	1°

ITEM NR / ITEM NO	AANTAL / QUANTITY	BESKRYWING / DESCRIPTION: FOOT ASSEMBLY		MATERIAAL / MATERIAL	ONDERDEEL NR / PART NR
VAN / SURNAME: TIKAM			VOORLETTERS / INITIALS: M	TITEL / TITLE: MEng	
STUDENTE NO / STUDENTE NR: 10055747			RIGTING / DISCIPLINE: M	TEKENING NR / DRAWING NR	
PROJEKSIE / PROJECTION	UNIVERSITEIT VAN PRETORIA / UNIVERSITY OF PRETORIA		A3	SKAAL / SCALE: 2:1	DATUM / DATE

Appendix C: Updates to PhantomX Parameters for `hexapod_ros`

Updates were made to some of the parameter values for the PhantomX hexapod, in the relevant YAML configuration file in the `hexapod_description` package of the `hexapod_ros` locomotion software stack. These updated values are listed below.

Details about the definitions of these parameters and their usage can be found in the YAML configuration files of the `hexapod_ros` source code [17]. All length and position values listed below are in units of meters, while angle quantities are measured in degrees.

```
INIT_FOOT_POS_X: [-0.1004, 0.0, 0.1004, -0.1004, 0.0, 0.1004]
INIT_FOOT_POS_Y: [0.1004, 0.142, 0.1004, 0.1004, 0.142, 0.1004]
INIT_FOOT_POS_Z: [0.055, 0.055, 0.0550, 0.0550, 0.0550, 0.0550]

COXA_LENGTH:   0.052
FEMUR_LENGTH:  0.0662
TIBIA_LENGTH:  0.13901

STANDING_BODY_HEIGHT: 0.042

LEG_LIFT_HEIGHT: 0.038
```

(kept as default value used in `hexapod_ros`)

For all femur servos:

```
offset: 13.09
```

For all tibia servos:

```
offset: -57.6
```

It should be noted that the **negatives** of the values calculated in Equation (4.1), Section 4.3.1, are set as the servo offsets here, as these values are **subtracted** from the desired joint angles, when mapping between joint space and actuator space in the source code of the `ServoDriver` class.

The following two modifications were made to the above values during the experimental tests of the Walking Posture Controller:

```
STANDING_BODY_HEIGHT: 0.072

LEG_LIFT_HEIGHT: 0.075
```

Appendix D: Computation of IMU Orientation from ROS Driver Output

The orientation estimate provided by the MicroStrain® 3DM-GX3® -25 IMU's onboard sensor fusion algorithm represents the rotation of the IMU's internal coordinate frame, relative to an Earth-fixed frame which has its z-axis pointing downward (in the direction of the gravity vector) [110].

However, the ROS MicroStrain® driver, discussed in Section 4.3.2.2, manipulates the rotation matrix returned by the IMU, so as to transpose it and rotate it about the y-axis by 180°, with the intention of providing a rotation that corresponds to a transformation from the IMU's internal frame to a world frame which has an upward-pointing z-axis [89]. This complicates the matter unnecessarily, as,

a) the IMU's coordinate frame has its z-axis pointing downward, even when mounted to the robot, and
b) the orientation of the body is required relative to the world, not the other way around.

While a simple transpose would solve the latter issue, the former would still mean that calculating Euler angles from the resulting, transposed transformation would yield yaw and roll angle values with 180° offsets from the actual yaw and roll of the robot body, as well as a pitch angle value which is the negative of the actual pitch of the robot.

In an attempt to remedy these issues, the necessary transposes and rotations were applied to the orientation from the driver, to reverse the transformations described at [89] and revert to the original rotation provided by the hardware. However, the resulting orientation measurements were still found to be incorrect. After thorough investigation the reasons for these errors were eventually discovered:

1. The IMU data communications protocol [110] uses the convention presented by Diebel [65] to represent rotation matrices.
2. On the other hand, inspection of the source code of the ROS driver [111] revealed that it uses functions from the standard ROS `tf` package (see Footnote 4 in Section 5.2.3) to compute the conversions between quaternions and rotation matrices. It turns out that the `tf` package follows the rotation matrix conventions presented by Craig [64], as illuminated by the `tf` API documentation [112].

As a consequence, converting the transformed rotation matrix to a quaternion using the function from `tf` has the inadvertent effect of transposing the rotation, resulting in the quaternion from the driver actually describing the rotation of the IMU relative to the new world frame (with z-axis up), rather than the rotation of the world frame relative to the IMU (as described at [89]). This type of confusion is precisely the reason why the importance of being aware of, and using consistent, conventions when working with rotation matrices is vehemently stressed in Chapter 3.

Knowing what rotation the driver's quaternion actually describes clears up the path for transforming this rotation back to the original orientation estimate provided by the IMU. The orientation matrix returned by the IMU's sensor fusion algorithm is represented as

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}, \tag{D.1}$$

while the quaternion provided by the ROS driver can be denoted by

$$\boldsymbol{q} = [q_0 \quad q_1 \quad q_2 \quad q_3]^T. \tag{D.2}$$

Using the convention of Diebel [65], the rotation matrix describing the orientation represented by $\boldsymbol{q}$ (computed using the mapping of Equation (3.23)) can be denoted by

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{D.3}$$

In other words, $\boldsymbol{R}$ represents the rotation provided by the ROS driver.

Based on the description of the transformations performed by the ROS driver (including the inadvertent transpose effect when converting to a quaternion), the relationship between $\boldsymbol{R}$ and the original IMU orientation $\boldsymbol{M}$ is described mathematically as

$$\boldsymbol{R} = \left(\boldsymbol{R}_y(180°)\boldsymbol{M}^T\right)^T, \tag{D.4}$$

where

$$\boldsymbol{R}_y(180°) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{D.5}$$

is the coordinate rotation matrix representing a rotation of 180° about the y-axis.

Equation (D.4) can be solved for $\boldsymbol{M}$ as follows:

$$\begin{aligned} \boldsymbol{R}^T &= \boldsymbol{R}_y(180°)\boldsymbol{M}^T \\ \boldsymbol{R}_y(180°)^T\boldsymbol{R}^T &= \boldsymbol{M}^T \\ \therefore \boldsymbol{M} &= \left(\boldsymbol{R}_y(180°)^T\boldsymbol{R}^T\right)^T. \end{aligned} \tag{D.6}$$

From linear algebra, it follows that

$$\begin{aligned} \boldsymbol{M} &= \boldsymbol{R} \cdot \boldsymbol{R}_y(180°) \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \\ \therefore \boldsymbol{M} &= \begin{bmatrix} -r_{11} & r_{12} & -r_{13} \\ -r_{21} & r_{22} & -r_{23} \\ -r_{31} & r_{32} & -r_{33} \end{bmatrix}. \end{aligned} \tag{D.7}$$

Geometrically, this signifies a rotation of the initial world frame (with z-axis pointing downward) about the y-axis by 180° into the ROS driver world frame (with z-axis point upward), followed by the rotation from the driver's world frame into the IMU frame as described by the driver's quaternion. The overall result is a rotation from the initial world frame into the IMU frame, which is equivalent to the rotation matrix returned by the IMU hardware.

By making use of Equation (D.7) in the mapping of Equation (3.19), the ZYX Euler angles describing the rotation of the IMU ($\boldsymbol{u}_{IMU}$) are calculated as

$$\boldsymbol{u}_{IMU}(\boldsymbol{R}) = \begin{bmatrix} \phi_{IMU}(\boldsymbol{R}) \\ \theta_{IMU}(\boldsymbol{R}) \\ \psi_{IMU}(\boldsymbol{R}) \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(-r_{23}, -r_{33}) \\ \operatorname{asin}(r_{13}) \\ \operatorname{atan2}(r_{12}, -r_{11}) \end{bmatrix}. \tag{D.8}$$

Thus, calculating the elements of $\boldsymbol{R}$ based on Equation (3.23), the Euler angles can be computed directly from the elements of $\boldsymbol{q}$ as

$$\boldsymbol{u}_{IMU}(\boldsymbol{q}) = \begin{bmatrix} \phi_{IMU}(\boldsymbol{q}) \\ \theta_{IMU}(\boldsymbol{q}) \\ \psi_{IMU}(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(-2q_2q_3 - 2q_0q_1, -q_0^2 + q_1^2 + q_2^2 - q_3^2) \\ \operatorname{asin}(2q_1q_3 - 2q_0q_2) \\ \operatorname{atan2}(2q_1q_2 + 2q_0q_3, -q_0^2 - q_1^2 + q_2^2 + q_3^2) \end{bmatrix}. \tag{D.9}$$

Although other rotations could be applied and the necessary angles could be extracted in different ways, the mappings presented above are suggested as they adhere to a single convention, eliminating any confusion or ambiguity. It also seems more appropriate to use the driver to simply obtain the original orientation estimate provided by the IMU, leaving the tasks of further manipulation and extraction of the desired information to the program executing the specific application, rather than having such conversions carried out by the driver itself.

Appendix E: Uneven Terrain Test Setup

E-1 Investigation on Existing Uneven Terrain Testing Methods

In Section 2.5, a wide variety of studies using quadruped and hexapod robots were discussed, including mention of experimental tests conducted by many of them. One of the frequently employed methods of introducing irregularity to the terrain, in these studies as well as others, is to simply have a single step or obstacle, often in the form of a block, in the robot's path of motion, such as in [7], [9], [35], [41], [113]. In the case of the experimental tests performed with the HITCR-II hexapod, multiple rectangular blocks were used as obstacles [10] (see Figure 2.16).

In some cases, many obstacles are scattered or arranged in a less ordered manner, such as in the tests with StarlETH [6] depicted in Figure 2.14, or those performed with BigDog [1], which can be seen walking on a "rubble pile" in Figure E.1 below.

Figure E.1: Experimental test of BigDog walking on a rubble pile [1]

Another test setup involving blocks, which can be found fairly frequently in the literature, is a grid, or "field," of closely packed blocks, of various heights. In Figure 2.27, the Weaver hexapod [13] was demonstrated standing on such a "block field" (which can also be seen as segment A in Figure E.3 below). Other examples of studies using this type of terrain to test hexapods include [51], [61].

In many cases, much less structured terrain setups with more natural characteristics are utilized. One example is a study with the LittleDog quadruped, where models of rock-like terrains, of varying difficulty levels [27], were utilized, as can be seen in Figure E.2 below.

(a)

(b)

Figure E.2: (a) Various uneven terrain models, and (b) typical experimental test setup with LittleDog quadruped [27]

Other attempts to emulate more realistic terrain conditions involve using a mixture of actual rocks, stones, pebbles, sand, and gravel in the testbed, such as in segments B–D of the multi-terrain testbed used for Weaver [13], portrayed in Figure E.3 below.

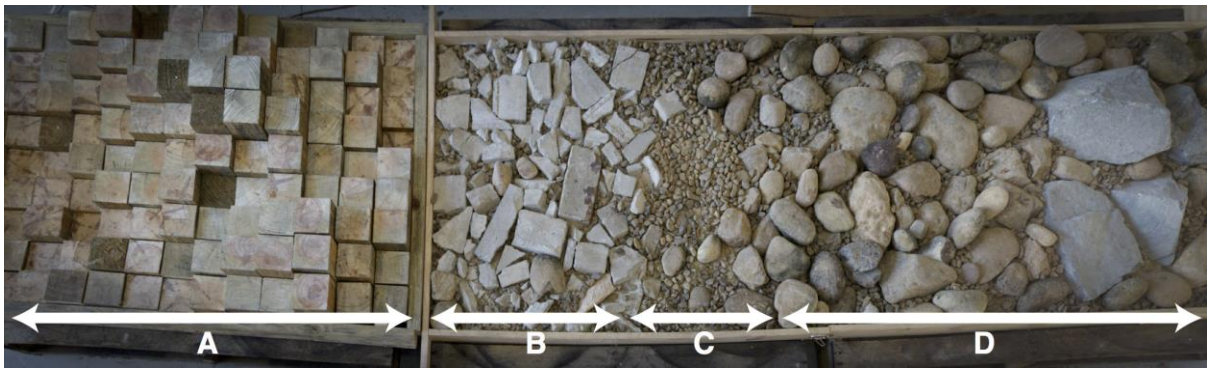


Figure E.3: Multi-terrain testbed, with four segments, used in experimental tests with Weaver hexapod [13]

Many studies even go further and opt to conduct tests in authentic, natural environments, instead of using artificial uneven terrain setups. Examples of studies which carried out experiments on outdoor terrains include [1], [8], [28], [114], two of which can be seen in Figures 2.7 and 2.12.

It appears that the variety of approaches taken for testing legged robots on uneven terrain is considerably wide-ranging. There does not seem to be a set standard to which these terrains are created, with many of them not even being easily measurable or straightforward to recreate. This is especially true for more natural terrains and ones in which obstacles or objects are simply scattered or piled together.

Even the more structured terrain types are often only described by their maximum height difference, as measured between the highest and lowest points occurring on them. Examples of terrain setups specified with such measures include the uneven models used with LittleDog [27] and the block fields used in [13], [51]. Clearly, this measure provides no information about the height variations in the terrain between the highest and lowest points, nor how these variations are distributed throughout the terrain area. Furthermore, in most cases it cannot even be said whether the robot will take footholds on the highest or lowest points, especially at the same time or consecutively (situations in which the maximum height difference becomes important). It may be that these points occur very far away from each other, or that they both lie somewhere on the extreme edge of the test setup where the robot's feet might not be placed during the test procedure. Moreover, it is hardly ever reported if the terrain variations were assigned purely at random, or if they were manually arranged (especially in the case of the block field), in a trial-and-error manner, to ensure that the robot would be able to achieve motion over the terrain.

Having said that the existing measures provided for most uneven terrain setups may be inadequate, or at least not completely sufficient, the question still stands – how is terrain unevenness actually classified, or what measures do, in fact, exist for quantifying or describing terrain variations?

Perusal of the literature seems to suggest that there is no straightforward answer. Natural terrains vary so much and their irregularities can be spread over such vast three-dimensional spaces that classifying them into set categories is not a trivial task. Measures of the magnitudes of the height difference are quite an intuitive gauge for how uneven a particular terrain is, but this suffers from the shortcomings described above. Other methods could possibly be used to obtain a quantitative measure of roughness using many points on the terrain. For instance, average or root-mean-square (RMS) values across a surface could be computed; however, these are generally more applicable to measuring the microscopic roughness of well controlled and engineered surfaces, such as those found on precision manufactured equipment, membranes, and optical components, among others [115]–[117]. On naturally occurring, and sometimes highly varying, surfaces such as the uneven terrains legged robots are typically expected to walk on, these single-value measures of roughness would still be limited by similar drawbacks as the maximum height difference. This is especially true because of the fact that legged robots use discrete footholds when moving, rather than experiencing the variations of the terrain in a continuous manner.

When using irregularities such as gravel or stones, the grade or classification of the particle sizes could be provided, based on a grain size classification scale. An example of this would be the Wentworth scale [118], which provides grain size classification grades for sediments ranging from clay with

microscopic particles, through multiple grades of silt and sand, all the way to large objects such as cobblestones and pebbles. Using one particular grade of stones or gravel would allow for fairly similar test setups to be created. However, when using a mixture of different grades, as in the multi-terrain testbed depicted in Figure E.3, reproducing the same test conditions becomes a problem once again. Even if a statistical distribution of the amounts of the various grades used is provided, recreating a terrain setup based on that statistical distribution would not necessarily result in the stones and rocks being distributed or laid out in the same way across the terrain area. With legged robots using discrete footholds, changes in the configuration of the terrain could have significant effects on the robot's behaviour. Furthermore, an assortment of other factors, apart from just the general size range of the particles, could play a vital role in the unevenness of the terrain, such as whether the individual stones and rocks are flatter or rounder, smoother or more jagged, uniform or irregular in shape, and many others.

Another subject area which could provide insights into terrain roughness measurements is vehicle dynamics, where analyses related to factors such as ride comfort and suspension performance rely on estimates of road roughness. By considering the road elevation profile, as a function of the longitudinal displacement along the road, to be a broad-band random signal, statistical properties of the profile can be used to describe it, with the primary representations being the Power Spectral Density (PSD) function [119]. By decomposing an elevation profile into a series of sinusoidal waves with varying amplitudes and phase relationships, the PSD can be determined as a plot of these amplitudes against their corresponding spatial frequencies, with spatial frequency being the inverse of the sinusoid's wavelength.

Road profiles are then classed based on the general amplitude level of their PSD plots. The PSD provides useful information of road roughness for vehicles, because they travel over long distances and experience elevation variations with wavelengths ranging from a few centimetres to many kilometres. Furthermore, the road profile in the longitudinal direction of motion is generally of significance in vehicle dynamics analyses, with lateral variations not being too common on most road surfaces, unlike uneven terrains on which legged robots are usually required to operate. In addition, road profiles can be considered as continuous signals because the wheeled vehicles that travel over them have predominantly continuous contact with the surface, experiencing the disturbances in a continuous manner, unlike legged robots which make contact with the terrain at discrete points.

Some surfaces of fairly significant unevenness are used quite commonly around the world for vehicle suspension testing, such as the "Belgian paving" surface which is constructed with cobblestones of varying heights and spacings [120]. However, even this surface is not constructed to any particular

standard or regulation across different vehicle test facilities, meaning that it would not be able to be used by others to reproduce the exact same testing conditions either.

These findings make it quite clear that no current method provides a complete solution to the issues of standardized uneven terrain testing of legged robots. Of course, the development of such methods is well outside of the scope of this study. Therefore, the best compromise would be to use a terrain type for which the height variations can be distinctly specified and controlled, providing precise information to be able to recreate the terrain, or at least compare setups used in later studies to the one used here. Hence, it was decided to create a block field. Although, to overcome some of the drawbacks of existing setups like this encountered in the literature, a workable option would be to randomly generate the block layout, while also specifying the individual heights of each block in the field. The development of this block field test setup is outlined in the section that follows.

E-2 Development of Test Setup for this Study

Before specifying the heights of the blocks to be used for the uneven terrain setup, it was necessary to determine an appropriate horizontal size for the blocks. In a study on the biomechanics and energetics of walking on uneven terrain [121], the human subjects were made to walk over blocks of varying heights to emulate uneven terrain conditions. Although the blocks were arranged in a repeating pattern, the researchers ensured that their length was not an integer fraction of the typical step length, so that the subjects were not able to learn or adopt a periodic compensation for the unevenness.

While the legged robot in this study is not selecting footholds based on the terrain and cannot learn a pattern, it is important to ensure that the block length does not correspond with an integer fraction of the step length so that the robot does not consistently step over, and thus avoid, certain sections of the terrain. Such a condition could potentially lead to the robot not experiencing the unevenness of the terrain sufficiently, thus skewing the results or making it appear to perform better than it actually would in a real-world situation.

Therefore, some measurements of typical stride lengths and distances between footholds were taken while the PhantomX walked around in different directions and at slightly varying speeds. It was found that the stride length is typically around 20 cm, with slight variations especially when changing directions. Thus, a block length of 10 cm would be inappropriate, and so a square block size of 7.5 cm × 7.5 cm was selected. With integer multiples being 15 cm, 22.5 cm, and so on, this length would ensure that approaching the terrain from slightly different starting positions would result in the robot's feet encountering different blocks, and hence height variations, rather than consistently stepping over the same ones. Furthermore, when comparing the block dimensions to the robot's feet,

this size is large enough for the robot to comfortably place its foot on, while still being small enough to provide sufficient variation and irregularity across the terrain. A 10 × 10 field of square blocks of length 7.5 cm, resulting in a field size of 75 cm × 75 cm, was thus deemed suitable for testing posture control with the PhantomX.

Having selected the block sizes, it was necessary to specify their heights across the terrain section. Based on the size of the robot, a maximum height of 8 cm was deemed appropriate. It was decided to divide the height range of 0 – 8 cm into 1 cm steps, resulting in nine different heights, or levels, of blocks throughout the one hundred blocks making up the field. To create a random instance of such a field, the `random` function, in the Statistics and Machine Learning Toolbox of MATLAB, was used to auto-generate a 10 × 10 matrix of random values, sampled from a normal, or Gaussian, distribution. This is similar to the method used in [122] to generate a random block field utilized in a study involving cockroaches running on rough terrain. For a range of 0 – 8 cm, a mean, μ, of 4 cm and standard deviation, σ, of 1.8 cm were specified for the normal distribution from which to be sampled. The resulting random values were rounded off to the nearest whole number, to generate discrete heights in 1 cm steps.

While a few iterations of the above method were computed and compared, it was found that an instance which created a slightly flatter discrete distribution, compared to the continuous normal distribution sampled from, resulted in greater variation in the terrain. This prevented too many of the blocks being at the mean height and reduced the number of "plateaus," or areas in which adjacent blocks on the terrain are at the same height. Therefore, this flatter distribution, with larger numbers of block heights just slightly off the mean, was opted for in this study. Figure E.4 below shows the resulting discrete distribution of block heights, over which the continuous normal distribution used for sampling is superimposed in red. A 3D bar plot of the associated block field is illustrated in Figure E.5, where the colours of the blocks are indicative of their heights.

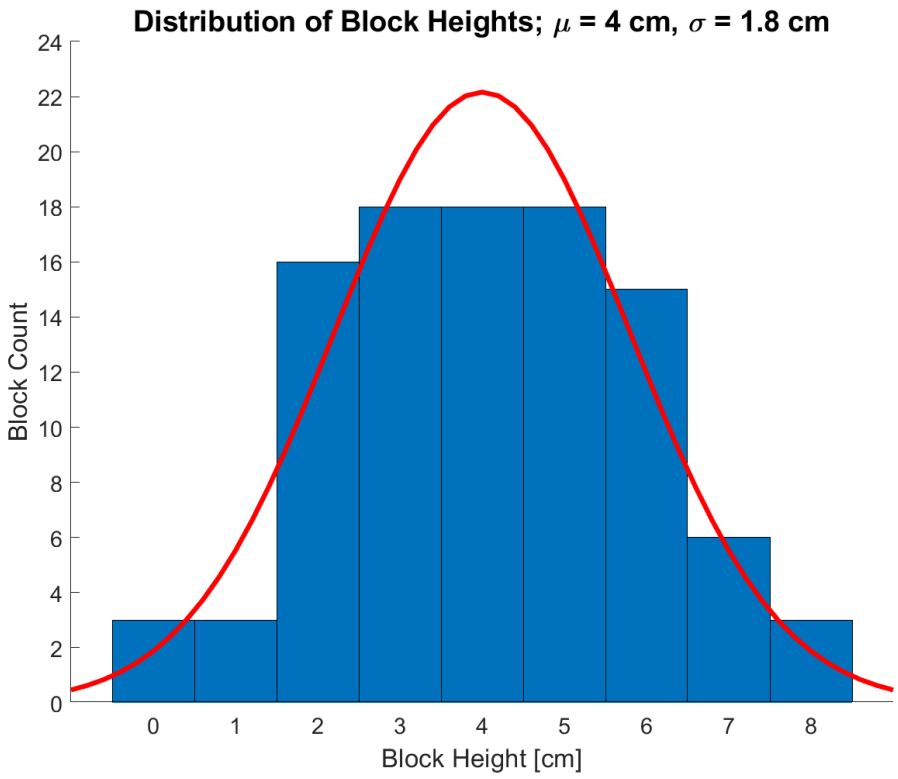


Figure E.4: Discrete distribution of block heights used in uneven terrain test setup (continuous normal distribution with $\mu = 4$ cm and $\sigma = 1.8$ cm shown in red)

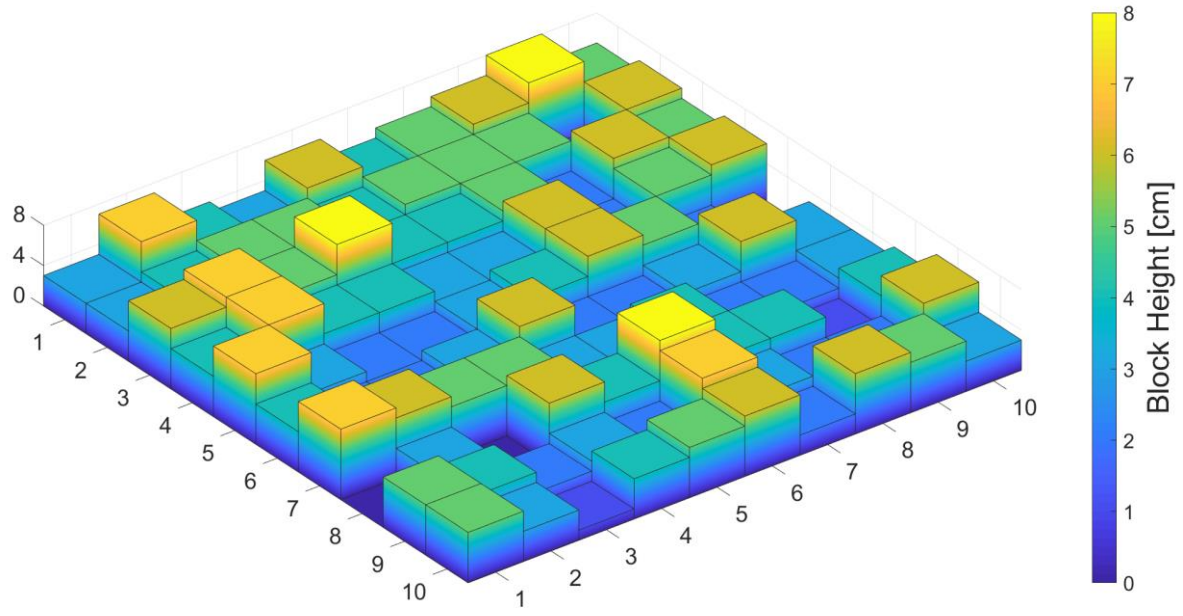


Figure E.5: 3D bar plot of 10 × 10 block field generated for uneven terrain test setup

Exact height values of each block in the field displayed above are presented in the form of Table E.1.

Table E.1: Height values of individual blocks making up the uneven terrain test setup (measured in units of centimeters)

	1	2	3	4	5	6	7	8	9	10
1	3	7	4	3	6	4	5	6	8	5
2	3	4	5	5	4	5	5	5	1	6
3	6	7	5	8	4	4	5	2	6	5
4	4	7	4	4	3	3	6	2	5	6
5	7	3	2	2	3	4	6	5	2	0
6	4	2	2	3	6	2	2	3	6	3
7	7	6	5	5	3	3	4	2	2	3
8	0	3	0	6	4	8	4	4	1	4
9	5	4	2	3	2	7	3	2	2	6
10	5	3	1	4	5	6	2	6	5	3

For the construction of this block field, the use of a hard material with sharp edges, such as wood (which is commonly used in the literature), was undesirable, as the edges could potentially cause damage to the OptoForce sensors if stepped on. Instead, high density foam was chosen for the blocks in this study. In particular, Sondor SPX 45, a Polyethylene foam that has a density of 45 kg/m^3 and a Shore 00 hardness of 65 [123] (see datasheet in Appendix E-3), was selected. Under the typical forces exerted by the robot foot, the foam experiences extremely small deformations, thus having a minimal effect on the robot's motion or posture response. However, in the few cases when the foot steps onto the edge of the foam block, the load is concentrated onto a small area and sufficient deformation takes place to prevent any damage to the force sensors, which is the exact behaviour desired.

To create the various heights of blocks required, a waterjet cutter was used to cut 7.5 cm squares out of SPX 45 foam sheets of three thicknesses (10 mm, 30 mm, and 50 mm). These squares were bonded

together, using contact adhesive, in the combinations necessary to produce the required numbers of blocks of various heights.

The resulting blocks, arranged in a grid as specified in Table E.1, were housed in a platform constructed from two sheets of 6 mm thick plywood, in between which a sheet of 10 mm thick polystyrene was sandwiched (by bonding it to the plywood using polystyrene adhesive), to increase the elevation of the platform without adding too much mass. A 75 cm × 75 cm square cavity was cut out of the centres of the top plywood sheet and the polystyrene sheet, into which the foam blocks were assembled. The height of the polystyrene and top layer of plywood acted as a frame, to hold the blocks together in the grid. An image of the 2.4 m × 1.2 m test platform, with the foam block field in the centre, is presented in Figure E.6. The flat, wooden sections alongside the block field provide an even base for the robot to start walking on, before encountering the uneven section, and to stop on after it passes completely over the uneven terrain.

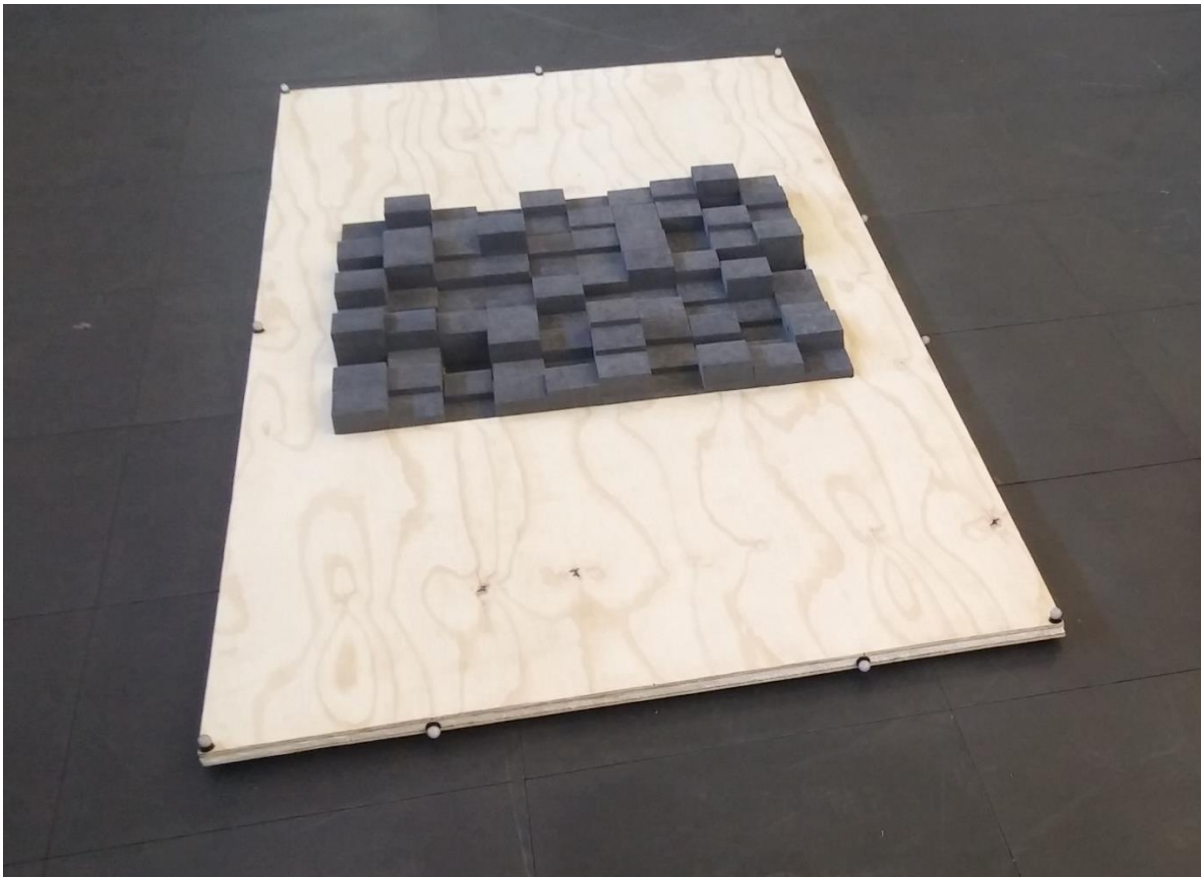


Figure E.6: Uneven terrain test platform with 10 × 10 block field in the centre

E-3 Sondor SPX 45 Foam Datasheet [123]

DATA SHEET: SPX 45

Date: Jun '14
Rev. No.: 13
Page: 1 of 1

SPX is a closed cell, cross-linked expanded Polyethylene foam available in various densities, which is suitable for use in packaging, padding, buoyancy, gasketing and footwear components. The SPX product range is free from CFC's and HCFC's.

PROPERTY	UNIT	TEST METHOD	NOMINAL (1)	RANGE
DENSITY:	kg / m^3	ISO 845	45	38 - 61 (2)
TENSILE STRENGTH:				
CD	kPa	ISO 1798	452	>340
MD	kPa	ISO 1798	437	>323
ELONGATION:				
CD	%	ISO 1798	226	>163
MD	%	ISO 1798	209	>140
COMPRESSION DEFLECTION:				
10 %	kPa	ISO 3386 / 1	81	55 - 106
25 %	kPa	ISO 3386 / 1	97	71 - 123
50 %	kPa	ISO 3386 / 1	157	124 -189
COMPRESSION-SET:				
25 % 22 hr COMP / 30 min REC	%	ISO 1856	10	<13
25 % 22 hr COMP / 24 hr REC	%	ISO 1856	4	< 6
50 % 22 hr COMP / 30 min REC	%	ISO 1856	24	<30
50 % 22 hr COMP / 24 hr REC	%	ISO 1856	15	<19
MAXIMUM OPERATING TEMPERATURE: (3)	°C	INTERNAL	100	N/A
BURN RATE: (4)	mm / min	INTERNAL		<100
BURN RATE:(5)	mm / sec	ISO 3582	0.6	<1.2
SHORE HARDNESS:	00	INTERNAL	65	61 - 69
THERMAL CONDUCTIVITY:				
10 mm	W / m.K	ASTM C-518	0.042	
20 mm	W / m.K	ASTM C-518	0.042	
WATER ABSORPTION:	%	ASTM D 570-98	< 1	< 1

1. NOMINAL:
   Indicative average value.
2. DENSITY:
   Based on 90 % net bun yield.
3. MAXIMUM OPERATING TEMPERATURE:
   Defined as the temperature which will typically cause an average linear shrinkage of no more than 5 % after a 24 hour exposure period. The percentage shrinkage of a sample, having the dimensions 100mm by 100mm by 10mm, with respect to its length, width and thicknesses is used to calculate the average linear shrinkage. The degree of shrinkage depends on the material type, density, temperature, exposure time, part dimensions and cell size. Other temperatures may prove to be limiting depending on the particular conditions of each application. The above quoted value will be deemed not applicable, if any deviation from the above mentioned sample dimensions are to occur.
4. BURN RATE:
   A 10mm thick sample is used to determine the horizontal burn rate of the relevant material. The above quoted value will be deemed not applicable, if any deviation from the above mentioned sample dimensions are to occur. Test based on FMVSS302.
5. BURN RATE:

   A 13mm thick sample is used to determine the horizontal burn rate of the relevant material. The above quoted value will be deemed not applicable, if any deviation from the above mentioned sample dimensions are to occur.

PLEASE NOTE:
The above results are obtained based on the referenced test methods and are to be regarded as typical values which are not usually directly comparable with those of any product tested to other test methods, i.e.: DIN. Tests were conducted at ambient temperature and humidity unless otherwise stated.

sondor
PERFORMANCE FOAMS

Member of the ZIMCO Group
SABS ISO 9001 SABS ISO 14001 SABS OHSAS 18001

CAPE TOWN: Tel: (021) 959 9400, Fax: (021) 959 9434, E-mail: ctn@sondor.co.za **DURBAN:** Tel: (031) 705 4220, Fax: (031) 705 4566, E-mail: dbn@sondor.co.za **JOHANNESBURG:** Tel:(011) 452 4530, Fax: (011) 452 4532, E-mail: jhb@sondor.co.za **PORT ELIZABETH:** Tel: (041) 486 2231, Fax: (041) 486 2234, E-mail: pe@sondor.co.za **PRETORIA:** Tel: (012) 803 4471, Fax: (012) 803 4400, E-mail: pta@sondor.co.za **EXPORTS:** Tel: +27-21-959 5900, Fax: +27-21-959 5901, E-mail: exports@sondor.co.za, **HEAD OFFICE**: Tel: (021) 959 5900, Fax: (021) 959 5901, E-mail: ho@sondor.co.za **EAST LONDON AGENT –** Jonny Grant 043 7433067/68 **BLOEMFONTEIN AGENT –** Build OFS 051 435 4880 **ZIMBABWE AGENT -** Security Devices/Leisure Lifestyle 00263 4 487064/5

WEB ADDRESS: www.sondor.co.za