# Chapter 9: Experiments in a Physical Environment

*The new agent architecture, INDABA, was proposed in chapter 5. INDABA was partially implemented for the purpose of the simulations and experiments described in chapters 7 and 8. The agents based on INDABA were implemented and they executed allocated tasks. The fourth layer of INDABA, the interaction layer has proved to be a valuable addition to the standard three layers. The interaction layer allowed for ease of implementation of various team selection methods. As a final confirmation of the applicability of INDABA to a robot architecture, this chapter provides results from a physical environment implementation. An introduction to the chapter is given in section 9.1. Section 9.2 describes the physical environment set-up, including an overview of the physical robots used, and environment types. Section 9.3 describes an implementation of the INDABA architecture to a hardware platform with limited capability, namely LEGO Mindstorms [185]. The social networks approach was used for robot selection for a scouting task. The results are presented in section 9.4.*

## 9.1 Introduction

Section 3.2 discussed Shakey, a robotic architecture [137] implemented in physical environment. One of the lessons learned from the Shakey project was that although a certain architecture may perform well in theory, the same architecture, when implemented in a physical environment, may not perform up to the expectations [120]. The view proposed by some of the leading researchers [28][112] and adopted in this thesis is that a robotic architecture must be implemented in a physical environment in order to accurately evaluate its performance. To prove the applicability of the INDABA architecture, and more specifically the social based approach to task allocation, the implementation was done in a physical environment. However, the focus of this thesis is not in creating a sophisticated multi-robot team environment. Instead, a simplistic physical environment was chosen for the purpose of this thesis, to show proof of concept.

## 9.2 Physical Environment Set-up

The physical environment used in this thesis uses standard "off-the-shelf" hardware and many simplifications (in comparison with a real-world application) were made. For example, the navigation method used is dead-reckoning (as applied to a LEGO platform [64]). The dead-reckoning navigation method is not ideal, especially considering the selected platform (due to the inaccuracies of the available sensors).

The physical environment set-up can be described by describing its three main components: robotic platform, robot population and the environment set-up.

### 9.2.1 Robotic Platform

For the purpose of a physical robot implementation, LEGO Mindstorms robotic platform was selected. LEGO Mindstorms robotic platform [185] was developed by LEGO and it was inspired by research done on MIT's Programmable Brick [162]. LEGO Mindstorms is an easy to use, reliable and cheap robotic platform that comes with a variety of development tools, the majority of which were developed by LEGO and the LEGO users community.

At the heart of every LEGO robot is a RCX Brick [185], a simple computer that supports the concurrent execution of up to 10 processes at a time. The number of available variables is 32 for global variables and 16 for local variables. All variables are of 16-bit signed integer type. An RCX has a Hitachi H8/300 CPU and 32K RAM, and it also has a limited communication capability. It is equipped with an IR port that is capable of sending and receiving messages. The LEGO Mindstorms was investigated in [155].

There are a few major shortcomings of the LEGO Mindstorms communication capability, as implemented in standard RCX, that makes the communication unreliable and of limited use:

- The communication is done via an IR port and as such it is basically line of sight communication with a limited range (in ideal conditions, about 10 meters).

- The messaging protocol is extremely simple and there is no addressing mechanism. This in turn means that it is impossible to send a message to a particular robot in a multi-robot team.

- The application programming interfaces (APIs) for LEGO IR tower are usually a third-party software with limited usability and they are usually not error-free. Often the provided APIs are not general enough to provide simple integration into a more complex software application.

The RCX Brick also supports numerous standard sensors (up to three at a time) and it can control up to three actuators. The supported sensors are very basic and not very accurate. The standard sensors include light, temperature, touch and rotation sensors. The standard actuators include micro motors and light sources.

The RCX can be programmed using a variety of programming languages. The two most popular LEGO programming languages are:

- Not Quite C (NQC) [182], which is a subset of the C programming language, adapted for RCX, and
- LASM (Lego ASEMBLER) [185].

NQC [182] was selected as the programming language for robot implementation.

### 9.2.2 Robot Population

Unfortunately, the choice of agent attributes was mainly limited by the availability of LEGO Mindstorms sensors and actuators, which tend to be very basic (refer to section 9.2.1). In order to increase the variety of attributes, some of the agent attributes are

implemented as software features and they depend on particular robot programming. The list of attributes, together with possible attribute values, is given in table 24.

| AGENT ATTRIBUTE | POSSIBLE VALUES |
|---|---|
| OBSATCLE AVOIDANCE | NO_AVOIDANCE |
| | AVOIDANCE |
| DRIVE | DRIVE_WHEEL |
| | DRIVE_TRACK |
| SPEED | SPEED_MEDIUM |
| | SPEED_HIGH |
| DETECTION | NORMAL |
| | LIGHT_ONLY |
| POWER | TETHERED |
| | BATTERY |

Table 24. Robot attributes and possible values

Due to the limited availability of LEGO Mindstorms kits at the University of Pretoria, the robot population was restricted to six robots. The attribute values of agents's attributes are given in table 25. Each row represents one of the agents in the population.

| ID (TYPE) | OBSTACLE AVOIDANCE | DRIVE | SPEED | DETECTION | POWER |
|---|---|---|---|---|---|
| 0 | AVOIDANCE | DRIVE_WHEEL | SPEED_MEDIUM | LIGHT_ONLY | BATTERY |
| 1 | NO_AVOIDANCE | DRIVE_WHEEL | SPEED_HIGH | LIGHT_ONLY | TETHERED |
| 2 | AVOIDANCE | DRIVE_WHEEL | SPEED_MEDIUM | NORMAL | BATTERY |
| 3 | NO_AVOIDANCE | DRIVE_WHEEL | SPEED_HIGH | NORMAL | TETHERED |
| 4 | AVOIDANCE | DRIVE_TRACK | SPEED_SLOW | NORMAL | BATTERY |
| 5 | AVOIDANCE | DRIVE_TRACK | SPEED_SLOW | LIGHT_ONLY | BATTERY |

Table 25. Robot population

An example of a robot (type 5) defined by the 5-tuple (AVOIDANCE, DRIVE_TRACK. SPEED_MEDIUM, LIGHT_ONLY, BATTERY) is given in figure 38.

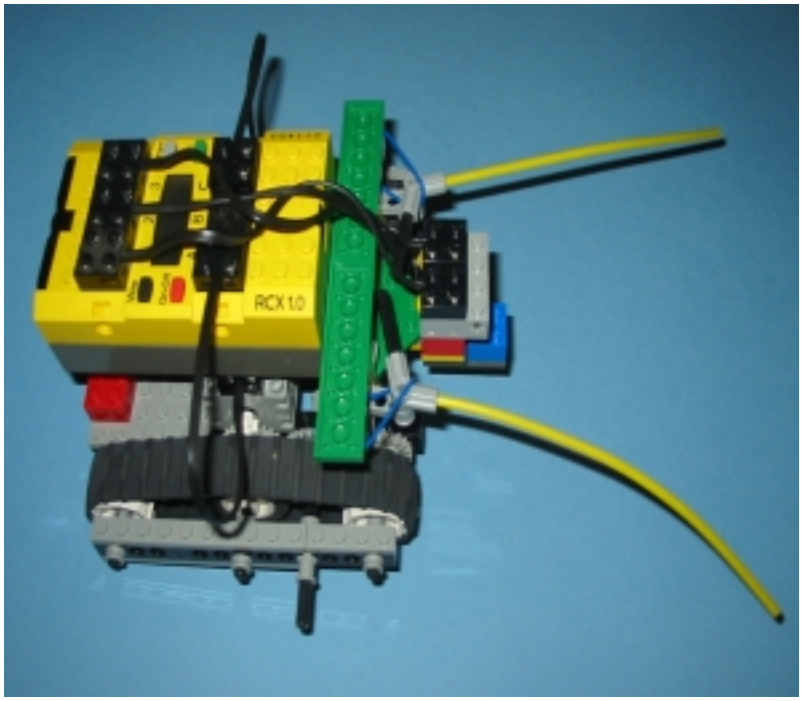


Figure 38. An example of a robot used in the experiments (type 5).

Robot type 5 utilises differential steering and it uses tracks as the main means of propulsion. As obstacle detection mechanism, two antennae are used. The robot has a simple light sensor for detection of collectable objects.

An example of another robot (type 3), defined by the 5-tuple (AVOIDANCE, DRIVE_WHEEL, SPEED_MEDIUM, NORMAL, BATTERY) is given in figure 39.

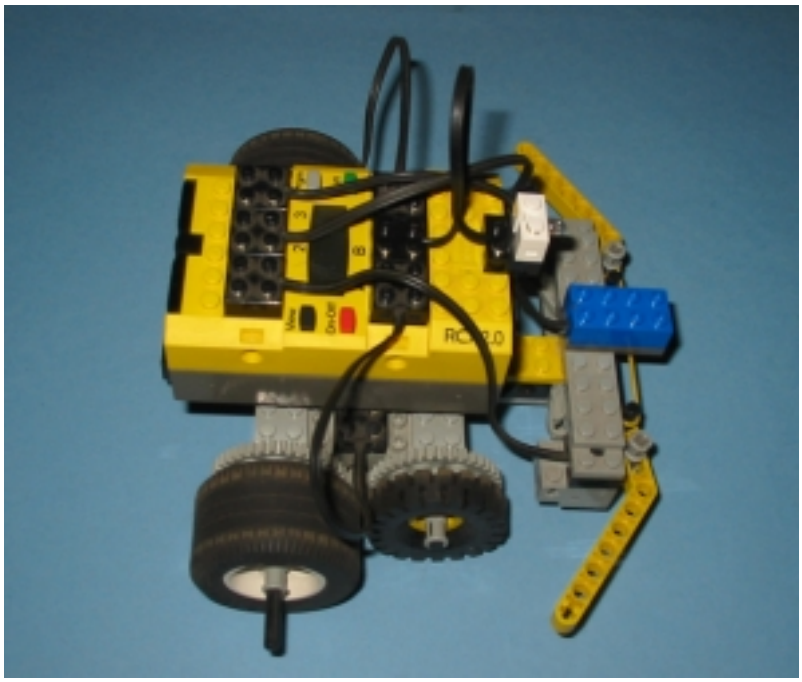

Figure 39. An example of a robot used in the experiments (type 3).

Robot type 3 again utilises differential steering but uses wheels as the main means of propulsion. Two bumpers are used as obstacle detection mechanism sensors. The robot has a simple light sensor for detection of collectable objects. Furthermore, the robot type has a Light Emitting Diode (LED) that allows for object detection in dark environments.

### 9.2.3 Environment Set-up and Types of Environment

For the purpose of the experiments in a physical environment, various environments were created. Each environment can be described by a set of attributes. The environment attributes and valid attribute values are given in table 26.

| ENIVRONMENT ATTRIBUTE | POSSIBLE VALUES |
|---|---|
| TERRAIN | NORMAL |
| | ROUGH_AREA |
| LIGHT | NO_SHADED_AREA |
| | SHADED_AREA |
| FOOD DISTANCE | FAR |
| | CLOSE |
| OBSTACLES | NO_OBSTACLES |
| | OBSTACLES |

Table 26. Environment attributes and possible values

For the purpose of this thesis, four environments were created. These environments are described next. The first environment is described by the 4-tuple (ROUGH_AREA, NO_SHADED_AREA, FAR, NO_OBSTACLES). The environment (illustrated in figure 40) has two areas of a rough terrain.

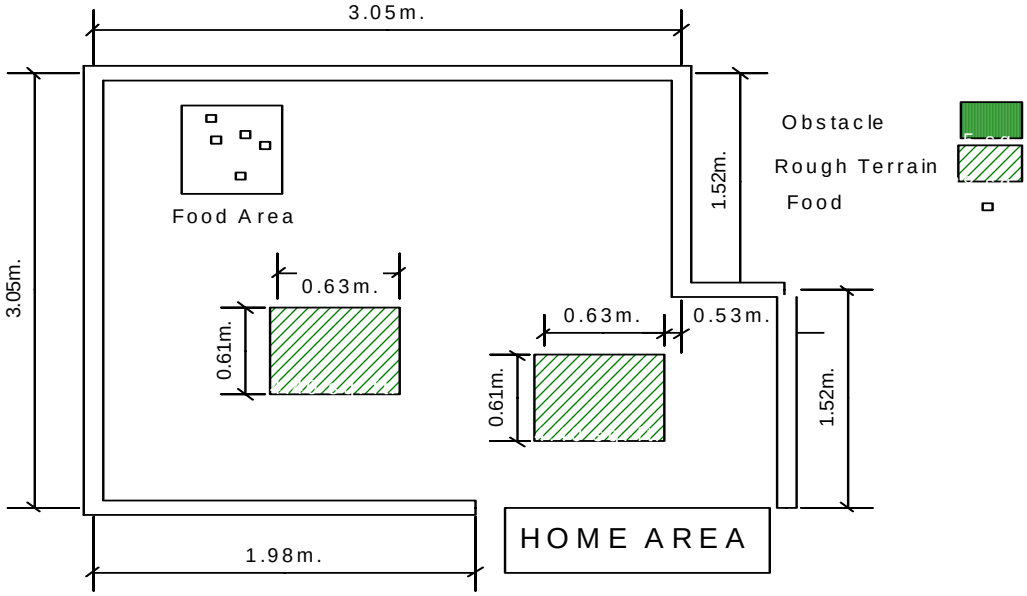


Figure 40. First environment used in experiments

The second environment used in these experiments is described by the 4-tuple (ROUGH_AREA, NO_SHADED_AREA, FAR, OBSTACLES) and is illustrated in figure 41.

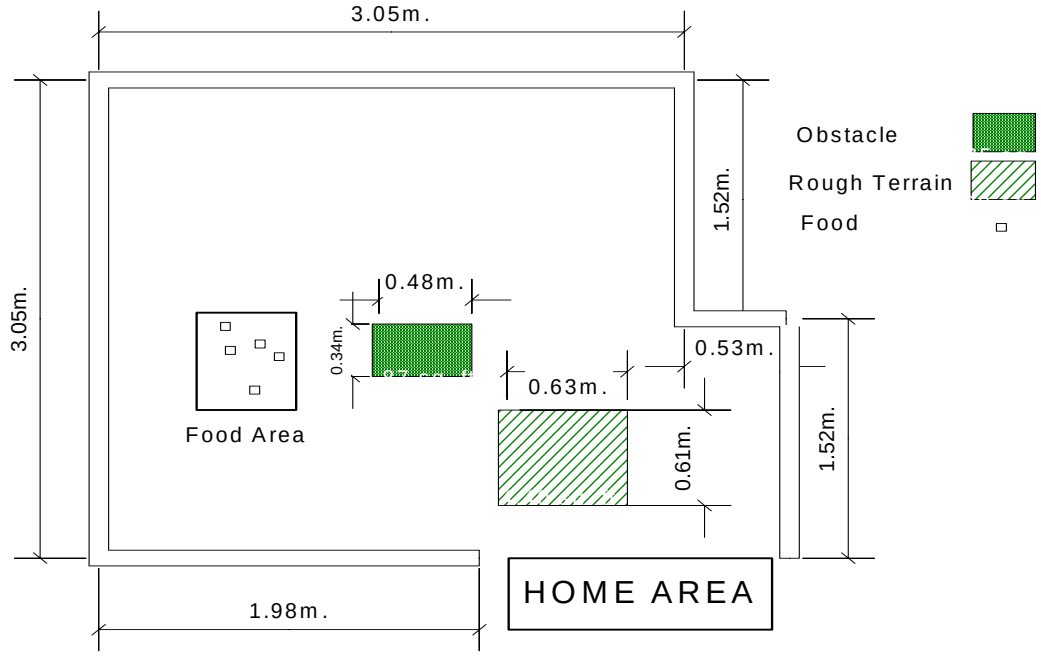


Figure 41. Second environment used in experiments

The 4-tuple (ROUGH_AREA, NO_SHADED_AREA, CLOSE, NO_OBSTACLES) describes the third environment, as illustrated in figure 42.

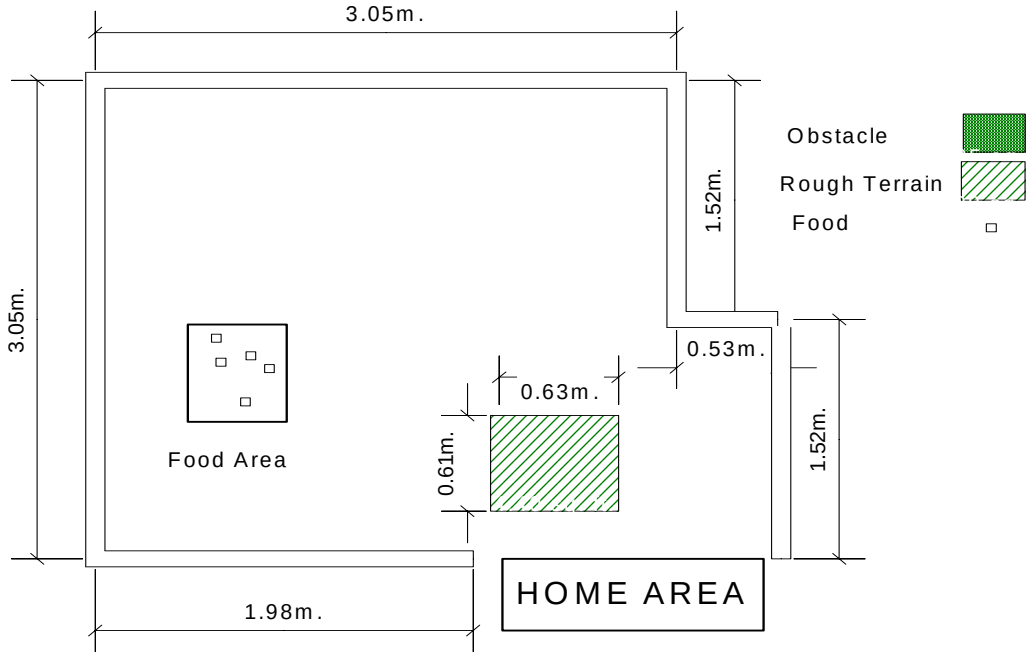


Figure 42. Third environment used in experiments

The fourth and last environment is described by the 4-tuple (NORMAL, SHADED_AREA, FAR, OBSTACLES) and is illustrated in figure 43.

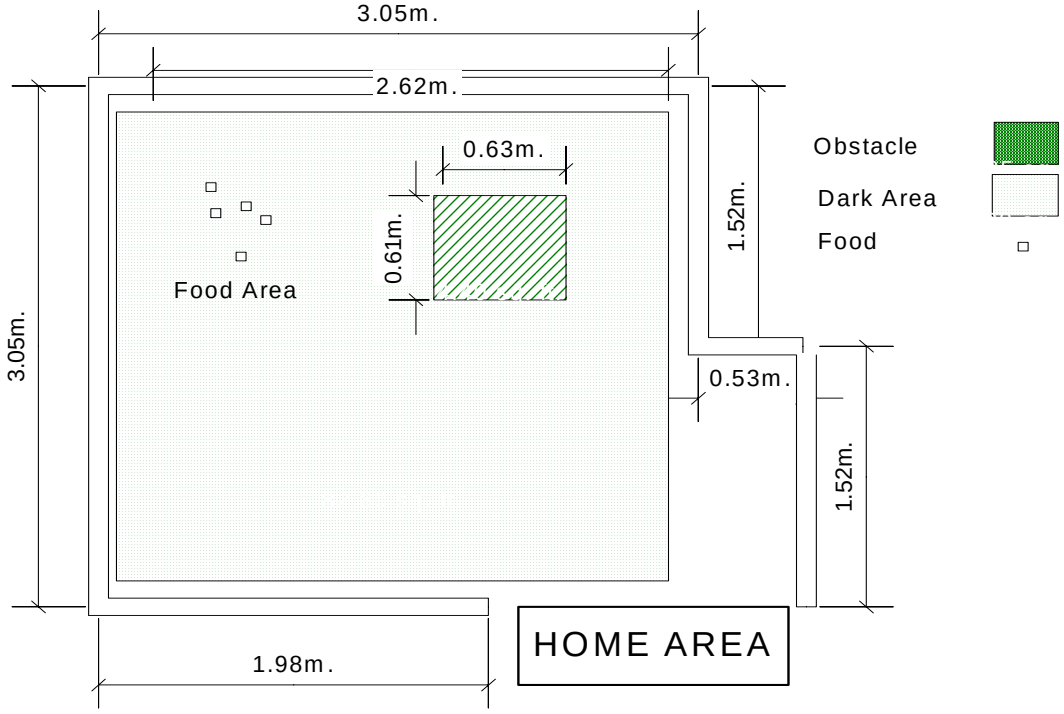


Figure 43. The fourth environment used in experiments.

Table 27 summarises the environments used in the experiments, together with their attribute values.

| ID (TYPE) | TERRAIN | LIGHT | FOOD DISTANCE | OBSTACLES |
|---|---|---|---|---|
| 0 | ROUGH_AREA | NO_SHADED_AREA | FAR | NO_OBSTACLES |
| 1 | ROUGH_AREA | NO_SHADED_AREA | FAR | OBSTACLES |
| 2 | ROUGH_AREA | NO_SHADED_AREA | CLOSE | NO_OBSTACLES |
| 3 | NORMAL | SHADED_AREA | FAR | OBSTACLES |

Table 27. Summary of environment types used in experiments

## 9.3 INDABA Implementation

The INDABA implementation presented in this chapter splits the four layers of INDABA into two groups, one implemented in the robots and the other one implemented as an application on a desktop PC equipped with an IR tower for communications with the robots. The reason for such implementation is that the

computational requirements of a full INDABA four-layer architecture exceeded the computing capabilities of the selected robotic platform.

Communication between the deliberator layer and the sequencer layer uses the infrared channel. The Phantom Application Programming Interface (API) [183] is used for communication purposes.

The Phantom Control API allows for direct access to variables stored on the RCX Brick. This mechanism was utilized to set active goals (by setting the B variable) and for retrieving the status of each goal (by getting the G variable). In order to reduce communication traffic (as the RCX can send and receive only a byte at time) one variable was used for setting active behaviours and another for retrieving the status of each behaviour. Behaviours and statuses were encoded using simple binary encoding. The overall architecture is illustrated in figure 44.

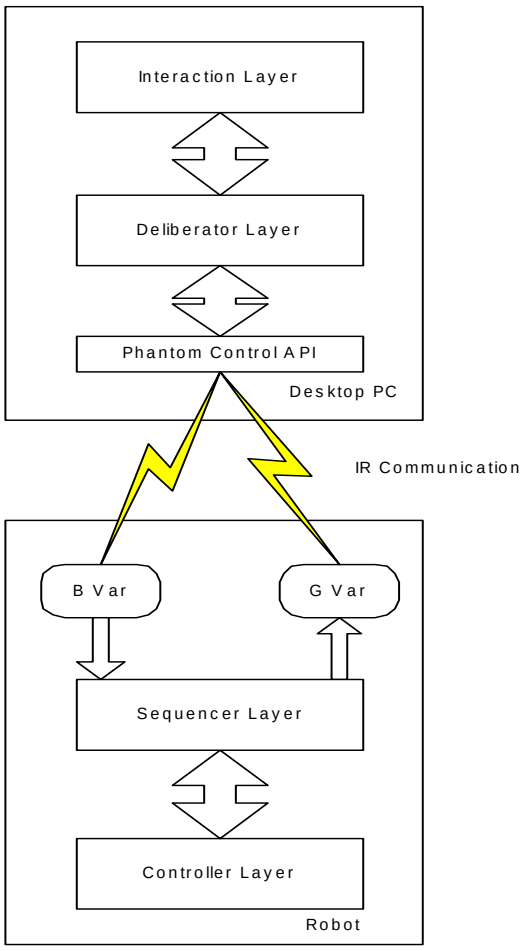


Figure 44. Implemented Hybrid Architecture

Due to the fact that the implemented method of communication allows only a PC to initiate communications, polling is used to gather date from robots. Polling occurs every 250ms.

### 9.3.1 Implemented Robot Components

Because of the processing power limitations of LEGO Mindstorms, only the lower two layers of INDABA could be implemented in the selected robotic platform. The lower two layers, that are computationally less demanding, are the controller layer (together with corresponding behaviours that are implemented therein) and sequencer layer.

#### 9.3.1.1 The Controller Layer

The controller layer consists of several behaviours, namely basic behaviours and synthesised behaviours. Synthesised behaviours are combinations of basic behaviours.

The basic behaviours, provided by the NQC implementation [182], are:

- *On* (OUTPUT); this simple behaviour activates output OUTPUT indefinitely.
- *OnFor* (OUTPUT, TIME); this behaviour activates output OUTPUT for a period TIME (TIME is expressed in tenths of a second).
- *Off* (OUTPUT); deactivates output OUTPUT.
- *OnRev* (OUTPUT); reverses the polarity of output OUTPUT. If a motor is connected to output OUTPUT, this behaviour will reverse its direction.

The synthesised behaviours are implemented as tasks in NQC terminology [182]. The synthesised behaviours include:

- *beh_move_for* (DIRECTION, DURATION). This behaviour encapsulates the details in how motion is achieved. Only direction of the movement and its duration are provided as input to this behaviour.

- *beh_detect.* This behaviour constantly monitors the input received from the light sensor. If the input exceeds a certain, predefined value, the behaviour terminates, signalling the detection of a food object.
- *beh_spiral_search.* A behaviour that executes increasing spiral search.
- *beh_safe_move.* A behaviour that allows for a movement with obstacle detection. If an obstacle is detected, an attempt is made to avoid the obstacle.
- *beh_send.* The behaviour that transmits the coordinate where food is detected. It is important to note that the coordinates are calculated using a dead-reckoning navigation approach and as such is susceptible to error.

The synthesised behaviours were sufficient for a simple scouting task. These behaviours are activated and deactivated by the next layer of INDABA, the sequencer layer.

#### 9.3.1.2 The Sequencer Layer

The sequencer layer combines the behaviours, as implemented in controller layer, in order to achieve certain goals. The sequencer is also implemented as a task in NQC terminology. It uses NQC commands *start task* and *stop task* to activate and deactivate the behaviours in the controller layer. For the purpose of the scouting tasks, three goals are defined:

- *GOAL_AREA.* This goal is achieved when a robot is in the target area that is the start of the approximated food area.
- *GOAL_FIND.* This goal is activated when a robot is in the target area. It is achieved when a behaviour *beh_detect* terminates, indicating the detection of a food object.
- *GOAL_SEND.* This goal transmits the coordinates of the area where food was detected.

The goals are a combination of behaviours. For the purpose of this application the combinations are hard-coded. Table 28 illustrates the implementation of the sequencer layer.

| GOAL | ACTIVE BEHAVIOURS | | | | COMPLETION CRITERIA |
|---|---|---|---|---|---|
| | *beh_safe_move* | *beh_detect* | *beh_spiral_search* | *beh_send* | |
| *GOAL_AREA* | ACTIVE | ACTIVE | | | Time or *beh_detect* |
| *GOAL_FIND* | | ACTIVE | ACTIVE | | *beh_detect* |
| *GOAL_SEND* | | | | ACTIVE | *beh_send* |

Table 28. Illustration of the implemented sequencer layer.

The sequencer layer constantly monitors a change in value of the variable G (see section 9.3). If a change is detected, the variable is decoded and the appropriate goal is activated, which in turn activates the corresponding behaviours (as in table 21).

### 9.3.2 Components Implemented in the Desktop PC

The higher two layers, i.e. the deliberator and interaction layers of INDABA are implemented in a desktop PC Windows ™ environment. The programming language used was C++. The implementation of deliberator and interaction layers is very similar to the implementation as described in chapter 7, and there was extensive re-use of the code.

#### 9.3.2.1 The Deliberator Layer

For the purpose of this particular INDABA implementation, a simple backward and forward chaining inference engine was developed. More on backward and forward chaining can be found in many AI textbooks, such as [170]. It is important to note that the backward chaining process is modified to suit the execution in a robot environment, as follows.

The deliberator layer loads the rules from a text file. When a goal is selected, the rules are back-chained until the first sub-goal is identified. The first sub-goal is a goal that cannot be back-chained further. The sub-goal is then passed as a goal to the sequencer layer. When the sequencer layer returns the sub-goal results, the inference engine

determines the next sub-goal and the process continues until the value of the goal can be determined.

To illustrate the process, consider the (very simple) set of rules as implemented for the purpose of the experiments described in this chapter. The implemented rules are:

Rule 1 : IF *GOAL_AREA* THEN *GOAL_FIND*
Rule 2: IF *GOAL_FIND* THEN *GOAL_SEND*
Rule 3: IF *GOAL_SEND* THEN *GOAL_SCOUT*

Consider the activation of goal *GOAL_SCOUT*. The inference engine back-chain rules until it gets to the first sub-goal from which it cannot back-chain further. That sub-goal is *GOAL_AREA*. The *GOAL_AREA* is then sent as a goal to the sequencer layer, utilising the mechanism described in section 9.3. Once the goal is achieved, the whole process repeats, but this time the first sub-goal is the *GOAL_FIND* and the process continues until the *GOAL_SCOUT* is satisfied or until the allocated time is exceeded.

The activation of a goal and the allocation of time to the task execution are done by the fourth and last layer of INDABA, the interaction layer.

#### 9.3.2.2 The Interaction Layer

The interaction layer consists of two main components: the task allocation and task evaluation components. The task allocation component utilises the social networks based approach, as described in section 7.1.

- *Task Allocation*

The algorithm starts with the task details propagation. For the purpose of the experiments and the selected task (scouting) implemented in this chapter, the task details consisted only of a time constraint, defined as maximum execution time.

Each of the available robots evaluates its own suitability to the task, by examining its own historical performance (using a trust relationship to itself, as described in section 7.1). The algorithm is similar to the general team leader selection algorithm 7.

The robot with the highest score is selected. In the case where the highest scoring robot is not available for the task, the next best one (based on kinship relationship as described in 7.1) is selected for the task execution. Setting the goal GOAL_SCOUT in its deliberator layer then activates the selected robot.

- *Task Evaluation*

If a robot is still executing when the allocated time expires, (the value of the goal *GOAL_SCOUT* in the deliberator layer is unknown), the agent is considered unsuccessful in the task.

If the execution of the task was successful (the value of the goal *GOAL_SCOUT* is true), its affinity to the task is increased. In other words, its own trust rating relative to a particular task's details improves (this represents its own historical performance; refer to section 7.3.). This in turn determines an agent's affinity to a particular task type.

## 9.4 Results

In order to provide some historical data, robots were initially randomly selected for the scouting task. For each environment, a robot was ten times randomly selected from the population of six robots (refer to table 25) and tasked with the scouting task.

To prove the validity of the social networks based team allocation mechanism, the fifth environment was randomly created and the social networks based team allocation mechanism was compared to random selection.

The results of random selection and a brief discussion on encountered issues for each environment are presented next, followed by the comparison of the social networks based approach with random selection.

### 9.4.1 Random Selection Results

The results of robots execution in various environments are presented in table 29.

| ENVIRONMENT | SELECTED ROBOT | RESULT (Min:Sec) |
|---|---|---|
| 1 | 4 | 0:23 |
| 1 | 5 | FAIL (Lost Position) |
| 1 | 4 | 0:42 |
| 1 | 2 | FAIL (Rough Area) |
| 1 | 0 | FAIL (Rough Area) |
| 1 | 1 | FAIL (Range) |
| 1 | 0 | 0:45 |
| 1 | 5 | 0:57 |
| 1 | 0 | FAIL (Rough Area) |
| 1 | 3 | FAIL (Range) |
| 2 | 5 | 1:05 |
| 2 | 2 | FAIL (False Detect) |
| 2 | 0 | FAIL (Avoidance) |
| 2 | 2 | FAIL (Avoidance) |
| 2 | 5 | FAIL (Avoidance) |
| 2 | 5 | 1:15 |
| 2 | 0 | 0:54 |
| 2 | 2 | FAIL (Rough Area) |
| 2 | 4 | FAIL (Avoidance) |
| 2 | 1 | FAIL (Avoidance) |
| 3 | 0 | 0:22 |
| 3 | 5 | 0:15 |
| 3 | 4 | 1:03 |
| 3 | 1 | 0:20 |
| 3 | 2 | FAIL (Rough Area) |
| 3 | 1 | 0:12 |
| 3 | 2 | 0:53 |
| 3 | 1 | 0:05 |
| 3 | 2 | 0:06 |
| 3 | 5 | 0:43 |
| 4 | 4 | 0:37 |
| 4 | 1 | FAIL (Range) |
| 4 | 1 | FAIL (Range) |
| 4 | 0 | 0:43 |
| 4 | 3 | FAIL (Range) |
| 4 | 1 | FAIL (Range) |
| 4 | 3 | FAIL (Range) |
| 4 | 5 | 0:16 |
| 4 | 4 | 0:48 |
| 4 | 2 | FAIL (Rough Area) |

Table 29. The results of random selection robot scout execution in physical environments.

The task was considered unsuccessful if a robot could not complete it in less than two minutes. The results reflect the time that it took a robot to complete the task, or alternatively the reason why it failed.

Numerous problems were encountered during the robots' execution in a physical environment, mainly related to randomness in the environment and physical robot attributes. The two most common problems are:

- *Changes in Lightning Conditions*

The LEGO Mindstorms light sensor is very sensitive to changes in light conditions. The robotic lab had a window and while every effort was made to restrict the light, the slightest change required recalibration. It was impossible to use the same settings in the morning and afternoon. To counter these effects, the robot was "trained" to recognise appropriate light sensor input for a food object every time before its execution. In other words, each robot was calibrated before execution.

- *Changes in Navigation Accuracy*

Inevitably, after a few execution cycles robots frequently lost their capability to move forward in a straight line, due to dust and residue build-up in their drive assembly. Affected robots then start veering to one side. This in turn leads to an imprecise spiral search process, and in extreme cases, the robots would get stuck turning in only one direction. Dead-reckoning navigation method then becomes useless. Furthermore, this inaccuracy also affected the obstacle avoidance algorithm, which was particularly visible when considering the results presented in table 26 for the environment 2. Although a simple obstacle avoidance mechanism was implemented, the sensors did not always accurately detect an obstacle and even when they did, the performed corrective action sometimes lead to potential loss of positioning.

Table 30 provides a summary of results per robot.

| ROBOT | ATTEMPTED | SUCCESS | RATING (SUCCESS/ATTEMPTED) |
|---|---|---|---|
| 0 | 7 | 4 | 0.57 |
| 1 | 8 | 3 | 0.38 |
| 2 | 8 | 2 | 0.40 |
| 3 | 3 | 0 | 0.00 |
| 4 | 6 | 5 | 0.83 |
| 5 | 8 | 6 | 0.75 |

Table 30. The results sorted by robots

### 9.4.2 Social Network Based Selection vs Random Selection

In order to check the validity of the social networks based approach, a comparison was made to a random selection approach. The social networks based approach was used for scouting team selection. As in the previous experiment, the scouting team was limited to one member. The trust relationship between agents was non-existent, however agents had trust in their own capabilities (based on historical performance, refer to table 29).

The kinship relationship table between the robots is pre-calculated and given in table 31 (the maximum strength is 1.0 and the minimum 0.0 – refer to formula 7.1).

| | Robot 0 | Robot 1 | Robot 2 | Robot 3 | Robot 4 | Robot 5 |
|---|---|---|---|---|---|---|
| Robot 0 | 1.0 | 0.5 | 0.83 | 0.33 | 0.67 | 0.5 |
| Robot 1 | 0.5 | 1.0 | 0.33 | 0.83 | 0.17 | 0.0 |
| Robot 2 | 0.83 | 0.33 | 1.0 | 0.5 | 0.5 | 0.67 |
| Robot 3 | 0.33 | 0.83 | 0.5 | 1.0 | 0.0 | 0.17 |
| Robot 4 | 0.67 | 0.17 | 0.5 | 0.0 | 1.0 | 0.83 |
| Robot 5 | 0.5 | 0.0 | 0.67 | 0.17 | 0.83 | 1.0 |

Table 31. Kinship between the robots

A sociogram to illustrate the kinship based social network is given in figure 45. For the purpose of this illustration, a strong kinship relationship is defined as a kinship with strength of 0.8 or more, and is illustrated by a thicker link between robots.

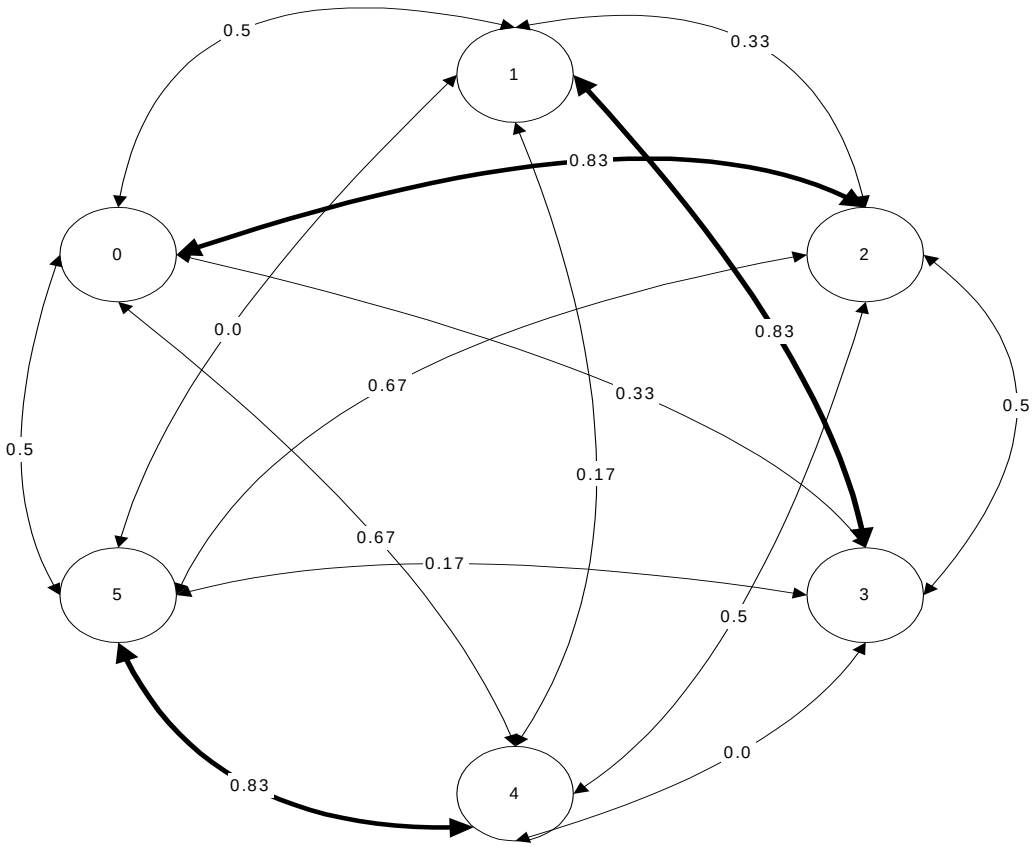


Figure 45. The sociogram of kinship relationship between robots

In order to introduce uncertainty, a new environment was created, as illustrated in figure 46.

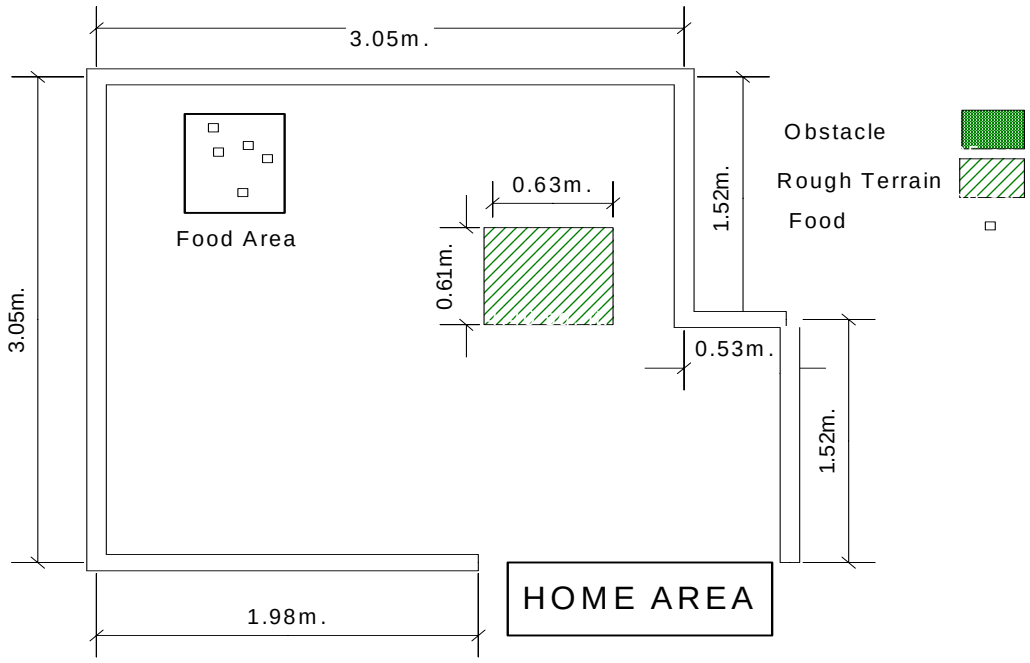


Figure 46. The fifth (test) environment used in experiments.

It is important to note that the test environment was a relatively easy one, as it had only one rough area and no shaded areas or obstacles. The social networks based approach was verified as follows: The best performing robot (robot 4) was considered unavailable. The social networks based approach therefore selected the next best available robot, based on the kinship relationship (refer to section 6.4.4). The result of the selected robot's execution was then compared to that of the randomly selected robot. For each scout selection method (i.e. social networks based and random), ten simulations have been done. Each simulation executed for a maximum of two minutes.

The task was considered unsuccessful if a robot could not complete it in less than two minutes. The results given in table 32 reflect the time that it took a robot to complete the task, or alternatively the reason why it failed. The results of the scouting task executions are presented in table 32.

| SOCIAL NET SELECTED SCOUT | RESULT (Min:Sec) | RANDOMLY SELECTED ROBOT | RESULT (Min:Sec) |
|---|---|---|---|
| 5 | 0:41 | 2 | 0:23 |
| 5 | 1:03 | 5 | 0:54 |
| 5 | FAIL (False Detect) | 1 | FAIL (Range) |
| 5 | 0:44 | 2 | FAIL (Rough Area) |
| 5 | 0:53 | 0 | FAIL (Lost Position) |
| 5 | 1:12 | 1 | FAIL (Range) |
| 5 | 0:57 | 0 | 0:26 |
| 5 | 0:48 | 5 | 0:39 |
| 5 | 0:38 | 0 | FAIL (Rough Area) |
| 5 | FAIL (Lost Position) | 3 | FAIL (Range) |

Table 32. Comaparisson of the results

The fluctuations in the time required for task execution were related to the intial robot positioning, changes in lightning conditions (which influenced light sensor readings) as well as the general failure of robots to maintain a straight direction without veering to one side.

The scout selected using the social networks based approach, successfully executed the task eight times, while randomly selected scouts succesfully executed the task only four times. The social networks based approach is therefore more reliable than random selection method, as demonstrated in this experiment.

## 9.5 Summary

This chapter presented a full INDABA agent architecture implementation, applied to physical robots. The primary purpose of this chapter was to verify the applicability of INDABA to physical robots. The secondary purpose of this chapter was to investigate limited application of the social networks selection method to physical robots. It is important to note that the social networks approach was the focus of chapter 7, where the social networks approach has been investigated in great detail in simulated environments. In this chapter, the social networks approach was implemented in a much-simplified manner.

The chosen task and chosen robotic platform were simplistic, as the focus was not on implementing a realistic real-world environment. Robots were given a scouting task to complete within a time constraint.

While the full physical implementation of a simulated environment was somehow restricted, it nevertheless provided proof of the applicability of the INDABA architecture to real-world robotic applications.

The social networks based approach, albeit using only a kinship relationship, performed better than a random selection strategy.

The next chapter summarises the work presented in this thesis.