Chapter 3

ACCURATE LOCALISATION SYSTEM

Several localisation algorithms rely on using all or most of the available references to enhance their performance. In contrast, the ALWadHA (**a**n efficient **l**ocalisation algorithm for **w**ireless **ad** hoc sensor networks with **h**igh **a**ccuracy) does not rely on using a high number of references to enhance the accuracy of estimation; rather it relies on using a smart reference-selection method. ALWadHA selects almost the minimum possible number of references that could contribute most to high accuracy. In order to evaluate and compare ALWadHA with other localisation algorithms, the network simulator, ns-2, was used. The current version of ns-2 was extended to simulate wireless sensor networks and, mainly, the localisation system by adding new modules. The extended ns-2 has a user-friendly interface, which enables a normal user, who has basic knowledge of simulating a simple wireless network, to simulate the proposed localisation system without any extra knowledge. The ALWadHA algorithm was evaluated with respect to the effects of network size and the deployment and density of nodes on the location estimation error and the number of references used. The researcher also examined the impact of increasing the distance-measurement error on the accuracy of the ALWadHA.

3.1 ALWADHA ALGORITHM

An ALWadHA has been developed to enhance the accuracy of position estimation. The idea is to select those references that are willing to help other nodes estimate their position. Based on this method, the node will only select those references that could contribute most to an accurate position estimate, and will eliminate irrelevant references from participating in the final position estimate.

Table 3.1. ALWadHA localisation algorithm

```
1. Initialisation
   If (final = true) then exit
   Broadcast “location request” messages
   Receive “location response” messages from neighbouring references ($R_i$)
   If $(C(R_i) < 3)$ then exit.
2. Initial position estimation
   Select a subset of references $S_i$ from $R_i$
   Measure distance to the references in $S_i$
   Apply MMSE to determine an initial position $\hat{z}_i^0$ .
3. Refined position estimation
   for $(j = 1$ to $C(S_i))$
       $\hat{e}^d_{i,j} = | \|\hat{z}_i' - \hat{z}_j\| - \hat{d}_{i,j} |$
       if $(\hat{e}^d_{i,j} > e^d_{max})$ then (enhancement = true); break.
   If (enhancement = true)
       for $(j = 1$ to $C(R_i))$
           $\hat{e}^d_{i,j} = | \|\hat{z}_i' - \hat{z}_j\| - \hat{d}_{i,j} |$
           if $(\hat{e}^d_{i,j} > e^d_{max})$ then eliminate $r_j$.
       Estimate refined position $\hat{z}_i$ as shown in 2
   else
       $\hat{z}_i = \hat{z}_i^0$ .
4. Position update
     $D_{acc} = \sum_{j \in S_i} | \|\hat{z}_i - z_j\| - \hat{d}_{i,j} |$
   if $(D^k_{acc} < D^{k-1}_{acc})$
       $\hat{z}_i$ will be accepted
       if $(D_{acc} < T_{acc})$ then (final = true).
```

The ALWadHA localisation algorithm consists of four phases, as shown in Table 3.1. In the first phase, the node collects information from nearby references. In the second phase, the node selects a subset of references to estimate its initial position. In the third phase, the node checks the possibility of improving the current position. In the final phase, the node decides if it will accept this position, and if the accepted position can be considered as a final estimate.

The ALWadHA algorithm follows two types of optimisation. Firstly, the node increases the number of participating references till it reaches a certain level of accuracy based on location error. Secondly, the node eliminates irrelevant references, based on distance error, which could bias the estimated location toward incorrect references.

3.1.1 Initialisation

Beacons assign their probability of accuracy to one $(P_{acc} = 1)$ (the probability of accuracy will be introduced in Section 3.1.2.1). Each reference holds the information required to localise other nodes, which is the location (or location estimate) and the probability of accuracy. A node initiates localisation by broadcasting a "location request" message, which includes the required accuracy level (L_{acc}), to the first neighbours (one-hop neighbours). The accuracy level of unknowns is equal to zero, while for knowns it is equal to the minimum probability of accuracy in the subset S that they used to estimate their position.

$$L_{acc} = \begin{cases} 0 & \text{unknowns} \\ \min\limits_{j \in S} P_{acc}^{j} & \text{knowns} \end{cases} \tag{3.1}$$

References receiving a request from an unknown will respond with a "location response" message containing the tuple $\{z \text{ (or } \hat{z}),\, P_{acc}\}$. However, if the request came from a known, then references will not respond unless their probability of accuracy is higher than the required accuracy level and also higher than the probability of response $(P_{acc} \geqslant L_{acc} \text{ AND } P_{acc} \geqslant P_{res})$; this response mechanism is shown in Figure 3.1.

The advantages of a response mechanism include:

- In addition to beacons, only knowns with a high probability of accuracy will act as references, which could minimise incorrect convergence.
- The time of response is specified to be only when it could contribute to a more

accurate estimate.

- The number of response messages is reduced, which reduces the communication cost.
- Knowns will not re-estimate their position unless the new estimate could enhance the accuracy of their current position, thus reducing the computation cost.

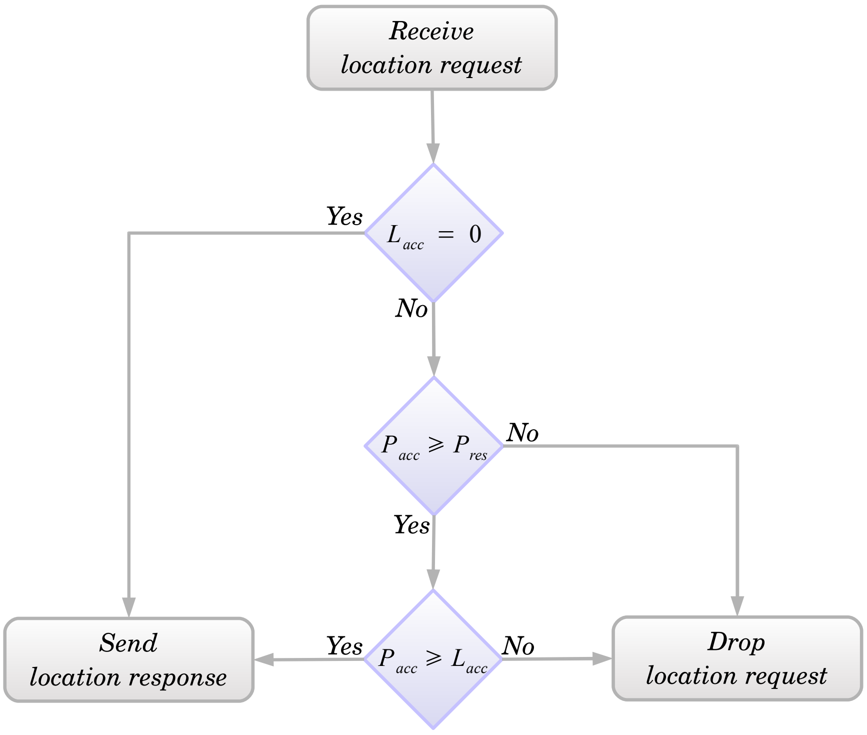


Figure 3.1. Response mechanism

The requesting node could receive several "location response" messages from neighbouring references (R_i), which it ranks in a descending order list based on their probability of accuracy. Each record of this list consists of a reference's *id*, location and probability of accuracy

$$R^i_{list} = \{id_j,\, z_j\ (\text{or } \hat{z}_j),\, P^j_{acc}\} \tag{3.2}$$

which will be used in the next phase to select a subset of references S_i.

3.1.2 Initial position estimation

This phase consists mainly of two parts. In the first part, the node selects the most accurate subset of references S_i using smart reference-selection method. In the second part, it measures the distance to each reference on this subset using RSS. Then the node applies the MMSE method to estimate its initial position $(\hat{z}_i^{'})$.

3.1.2.1 Smart reference-selection method

The key idea is to select a subset of references S where $S \subseteq R$ could contribute to an accurate location estimate, i.e. the selection method should satisfy the following two conditions. Firstly, it should not cause a high computation error. Secondly, it should select only those references that have relatively low location and distance errors. To satisfy the first condition, the selection method should have the following characteristics:

- Is very simple
- Selects the minimum possible number of references.
- Deals with location and distance error separately.

Because of these characteristics the computation cost of the selection method is low, which could reduce the *computational error*. To satisfy the second condition, the references are selected on the basis of what are called accuracy levels, where the node selects a subset of references that could have the minimum *location error*. In addition, the next phase (refined position estimation) deals with *distance-measurement error*.

Initially, one can ignore the distance error $(e^d=0)$ and the total error can then be represented as:

$$e^l = e^c + \sum_{j \in R} e_j^l \quad . \tag{3.3}$$

The goal is to select a specific subset of references S that minimises the location error over all possible subsets. If R is the set of available references with cardinality m, then $\varsigma = \{S^1, S^2, S^3, \ldots, S^C\}$ is the set of all possible subsets from R, where C is the cardinality of ς and the cardinality of $S^k \in \varsigma$ is c, where $c \geqslant 3$. The number of combinations to select a subset S^k, with cardinality c, from R is:

$$\frac{m!}{c!\,(m-c)!} \tag{3.4}$$

and so the total number of possible subsets in ς is:

$$C(\varsigma) = \sum_{c=3}^{m} \frac{m!}{c!\,(m-c)!} \quad . \tag{3.5}$$

After finding all of these subsets, S can be specified as follows:

$$S = S^k, \text{ where } S^k \in \varsigma \text{ and } S^k \text{ has } min\ (\sum_{j \in S^k} e_j^l) \tag{3.6}$$

where S represents the subset of minimum location error that could contribute most to accurate location. However, finding this subset requires an exhaustive search and expensive computations, which makes this approach not practically feasible for very resource-constrained networks such as WSNs. Therefore, a different approach is applied to find this subset.

The references are divided into several levels based on their location accuracy, starting with level 0, which includes all beacons with a location error equal to zero $(e^l = ٠)$. Level 1 consists of nodes that use only beacons to estimate their position, i.e. their location error is caused by computation error only $(e^l = e^c)$. Level 2 consists of nodes that use one reference from level 1 and the rest of the references are from level 0; their location error is $e^l = e^c + e_j^l$, where $j \in \text{level } 1$. This carries on with the rest of the nodes based on the reference level used in their position estimation. To take advantage of these levels, a probability of accuracy (P_{acc}) will be assigned to each level. Beacons at level 0 have the maximum probability of accuracy $(P_{acc} = 1)$, while the probability of accuracy of other references can be estimated as follows:

$$P_{acc} = ١ - \frac{١}{\sum_{j \in S} P_{acc}^j} \tag{3.7}$$

where P_{acc}^j is the probability of accuracy of reference r_j, and so each reference keeps not only its location but also the corresponding probability of accuracy. The probability of accuracy will not be used in the position estimation, but it is used to distinguish between the levels of references and to allow nodes to rank the references on the basis of these levels.

As mentioned in the previous section, after receiving the response messages, the node ranks references in a descending order list (R^i_{list}), based on their probability of accuracy. The node selects the first three references in R^i_{list} and then calculates the probability of the accuracy of S_i using Equation (3.7). If P_{acc} is less than a certain value, the next reference will be added to S_i and P_{acc} is recalculated. The node carries on with this process till the subset S_i satisfies a certain value of probability of accuracy $(P_{acc} \geqslant P_{min})$, otherwise it will stop and postpone the estimation to the next iteration, where more accurate references could be available. Figure 3.2 shows the proposed smart reference-selection method.

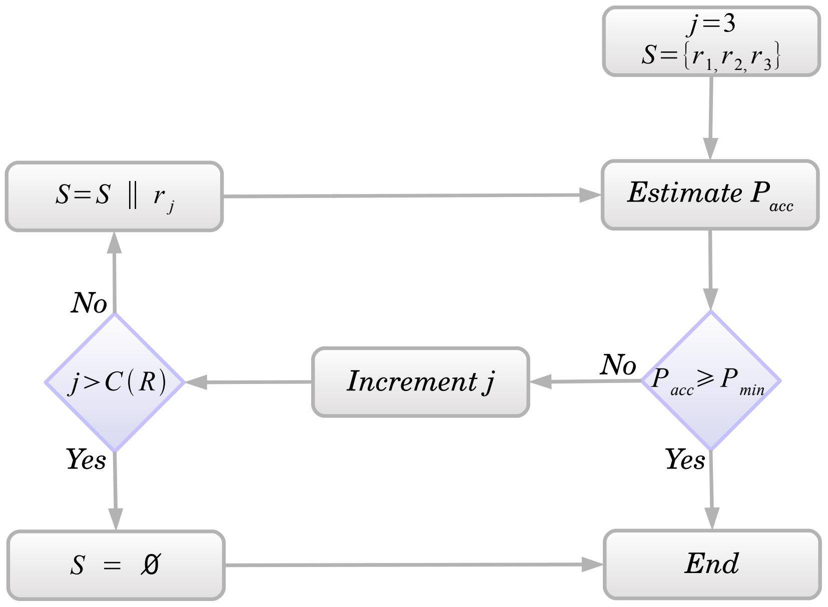


Figure 3.2. The smart reference-selection method

The smart reference-selection method has the following characteristics:

- It selects references that could have the lowest location error.
- The estimation of P_{acc} requires very simple computation.

- It is not based on measured distance, which could be corrupted by multiplicative noise.

- It specifies the time of estimation; nodes that have a more accurate subset will estimate their position first. The technique followed for this purpose does not require any interaction or message broadcasting between nodes; rather, it is based on available information.

- Rejecting a position estimate with low accuracy could delay the time required by a node to know its position, but on the other hand it could enhance the position accuracy and minimise the incorrect convergence due to cumulative error.

3.1.2.2 Cardinality of subset references

Several localisation algorithms that are based on the subset-references approach focus on improving the selection method. However, specifying the proper number of references that should be used to guarantee a certain level of accuracy has rarely been studied in the literature. The cardinality of the subset references can be specified either manually or dynamically, as explained in Section 2.1.6.1. The proposed algorithm is based on the dynamic approach, where the node starts by using the minimum possible number of references $(c=3)$, and then increases this number gradually until the selected subset of references achieves the minimum level of accuracy

$$S = S^k \text{ , where } S^k \text{ has } min(c) \text{ and } \left(1 - \frac{1}{\sum_{j \in S^k} P_{acc}^j}\right) \geqslant P_{min} \quad . \tag{3.8}$$

The cardinality of S could vary from one level to another, for instance at level 1 the cardinality of S is equal to three $(c=C(S)=3)$, while it will be increased in the higher levels, because at level 1 the node uses only beacons with no location error, while on the higher levels references with location error start to participate in the estimation and so the node increases the number of references slightly to handle this error.

3.1.3 Refined position estimation

In the previous phase the node selected the subset of references based only on location error. In this phase the node enhances the accuracy of position estimation by considering the distance error, where references with high distance error are eliminated from R_{list}^{i} . This phase starts by checking the possibility of enhancement based on the estimated distance error $(\hat{e}_{i,j}^{d})$, which is the difference between the calculated distance (between the node's initial position $(\hat{z}_i^0)$ and references position $(z_j \text{ or } \hat{z}_j)$) and the measured distance $(\hat{d}_{i,j})$

$$\hat{e}_{i,j}^{d} = | \, \| \hat{z}_i^{\cdot} - \hat{z}_j \| - \hat{d}_{i,j} \, | , \quad \text{where } j \in S_i \ . \tag{3.9}$$

If the estimated distance error for at least one reference in S_i is greater than a certain value $(\hat{e}_{i,j}^{d} > e_{max}^{d})$, this indicates that position enhancement is required, otherwise the initial position is regarded as an accurate position $(\hat{z}_i = \hat{z}_i^{\cdot})$ and the process continues to the next phase. To enhance the position estimation, the node eliminates those references that have $\hat{e}_{i,j}^{d} > e_{max}^{d}$, where $j \in R_i$ from R_{list}^{i} , then a new subset of references will be selected to estimate a refined position $(\hat{z}_i)$ as described in the previous phase. It is remarked that position refinement is not required at each iteration; it is only required when at least one of the references in the subset S_i has a high distance error. Checking the estimated distance error not only enhances the accuracy of estimation but could also be used to detect the existence of malicious nodes.

3.1.4 Position update

In the final phase, the node performs two tasks. Firstly, it checks the acceptance of the estimated position. Secondly, it applies the termination criterion. To check the acceptance of the estimated position, the node computes the position's degree of accuracy (D_{acc}) as follows:

$$D_{acc} = \sum_{j \in S_i} | \, \| \hat{z}_i - z_j \| - \hat{d}_{i,j} \, | \ . \tag{3.10}$$

If the new degree of accuracy is better than the one from the previous iteration

$(D_{acc}^{k} < D_{acc}^{k-1})$, it will accept the new estimated position, otherwise it will be rejected. This test could ensure that the new accepted position is more accurate than the previous one, and it furthermore screens out incorrect convergence and divergence.

Finally, if the position is accepted and its degree of accuracy is less than the accuracy target $(D_{acc} < T_{acc})$, the node considers this position as a final one and stops sending “location request” messages. Following this approach for termination has the following advantages:

- Nodes that have a subset of references with high accuracy will terminate their position estimation at an early stage, which could reduce the computation and communication cost.

- More time will be allowed for those nodes with a low accuracy subset of references to enhance their position estimate.

Compared with the time-based or iteration-based termination approaches, where the nodes terminate the localisation process after a certain time or a specific number of iterations, the approach used could outperform these two approaches in terms of computation, communication and the accuracy of final position estimate.

3.2 IMPLEMENTATION

The difficulties of setting up a WSN with real nodes and the infeasibility of analysis make simulation an essential tool to study WSNs. Simulation is widely used in system modelling for applications ranging from engineering research, business analysis and manufacturing planning to biological science research [59].

3.2.1 Network simulator (ns-2) overview

Ns-2 [60] is an open-source event-driven simulator designed specifically for research in networks. Ns-2 was developed in C++ and uses Object-oriented Tool command Language (OTcl) as configuration and script interface (i.e., a front-end). Each language has two types of classes. The first type includes the standalone C++ and OTcl classes that are not linked together. The second type includes classes that are linked between the two languages.

These C++ and OTcl classes are called *compiled hierarchy* and *interpreted hierarchy*, respectively. These two hierarchies are linked together using an OTcl/C++ interface called TclCL [61].

Ns-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks. Recently, Morávek [62] investigated the ns-2 capabilities in node localisation in wireless networks. This investigation shows that ns-2 supports simulations of different localisation techniques (such as TOA and RSS). Ns-2 has several tools and modules that researchers can use to develop localisation schemes. The researchers can modify the existing modules, or create new modules from the very beginning, and ns-2 allows researchers a high level of independence from the designed framework of the simulator. Therefore, the ns-2 simulator will be used to implement and to evaluate the proposed localisation algorithm.

3.2.2 The extended ns-2

Ns-2 contains several flexible modules for energy-constrained wireless ad-hoc networks, which encourages researchers to use ns-2 to investigate the characteristics of WSNs. However, to implement and evaluate the proposed localisation algorithm, the current ns-2 version (ns-2.34) should be extended and new modules should be added. Figure 3.3 shows the new classes that were added to the ns-2. These classes can be divided into two types. Firstly, there are *standalone* classes, which are MMSE, Position and AlwadhaPosition classes. These classes are used only from the C++ domain. Secondly, there are *compiled hierarchy* classes, which are LocDisApp, LocReqAgent and LocResAgent classes. In order to access these classes from the OTcl domain, they should be linked to the corresponding *interpreted hierarchy* classes, Application/LocDis, Agent/LocReq and Agent/LocRes, respectively.

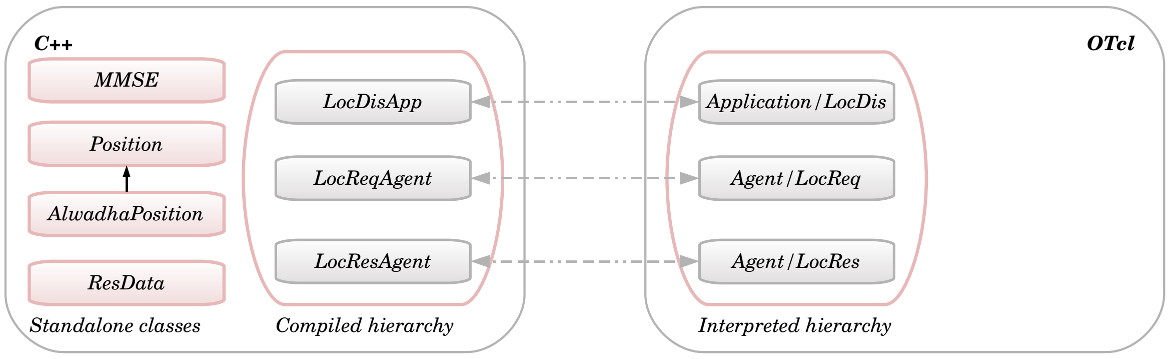


Figure 3.3. The new modules added to the ns-2

3.2.2.1 MMSE class

This class is responsible for all the mathematical matrices operations required to solve Equation (2.2). Instead of using a general matrices multiplication and matrix inverse, optimised methods dedicated mainly to MMSE were implemented. These optimised methods require less computation and shorter execution time.

3.2.2.2 Position class

This class performs the general multilateration method, which was explained in Section 2.2.2, to estimate the node position. This method uses all of the available references, does not distinguish between beacons and references, does not weigh the references used and performs the estimation only once. This class is the base class for ALWadHA and other localisation algorithms that will be explained later.

3.2.2.3 AlwadhaPosition class

This class is derived from Position class. It includes the implementation of an ALWadHA localisation algorithm, which was explained in Section 3.1. Compared with the Position class, AlwadhaPosition has more functionalities such as using a smart reference-selection method, specifying the number of references used, applying a termination criterion and other features as explained earlier.

3.2.2.4 ResData class

This class is responsible for storing and retrieving the location information included in the “location response” packets received from the neighbouring references. This information is the address, the location, the probability of accuracy of the sending references and the power with which the packet is received.

3.2.2.5 LocReqAgent class

LocReqAgent is derived from the Agent class. This agent is responsible for constructing a “location request” packet (PT_LOCREQ) and then broadcasting it to neighbouring nodes. This agent should be attached only to unknown nodes because beacons already know their location.

3.2.2.6 LocResAgent class

This class is a child of class Agent. It is responsible for receiving packets. This agent should be attached to all nodes (unknowns and beacons). Two types of packets could be received. The first is a *location request* packet (PT_LOCREQ). If this type of packet is received by a beacon or reference node it constructs a “location response” packet (PT_LOCRES) that includes location information and then sends it to the requesting node. Unknown nodes receiving this type of packet simply deallocate it. The second is a *location response* packet (PT_LOCRES). The requesting node that receives this packet sends it to the application layer (LocDisApp), which processes the included location information to estimate the node's position.

3.2.2.7 LocDisApp class

This class is derived from the Application class. Each node in the network uses an object from this class by attaching it to its agent(s). The LocDisApp class performs several functions, such as periodically invoking the broadcast method of LocReqAgent to broadcast a "location request" packet, processing the received “location response” packet and estimating the node location.

3.2.2.8 Interpreted hierarchy

In fact, no OTcl modules were created. However, in order to be able to access the newly compiled hierarchy classes LocDisApp, LocReqAgent and LocResAgent from the OTcl domain, these classes were mapped and linked to corresponding OTcl classes, which are Application/LocDis, Agent/LocReq and Agent/LocRes, respectively. In this way the users are able to create an object of the compiled hierarchy classes from the OTcl domain. For example, the OTcl command "set lreq [new Agent/LocRec]" will create a new object of class: LocReqAgent.

3.2.3 Class hierarchy

The Doxygen documentation system [63] was used to illustrate the class hierarchy of the new classes. For the sake of simplicity, only the new classes, the classes they are derived from (i.e. parent classes) and the classes used by these new classes were included. Solid lines show where a class is "inherited", or derived, from another class; for example $A \rightarrow B$ means class A is derived from class B. Dotted lines show when a class is using a method and/or member of another class.

3.2.3.1 Position and AlwadhaPosition classes

The Position class represents the general multilateration method for estimating the node position, which is called the M_Single algorithm. As shown in Figure 3.4, Position class is the base class of AlwadhaPosition class and the other localisation algorithms, which were implemented for the sake of comparison.

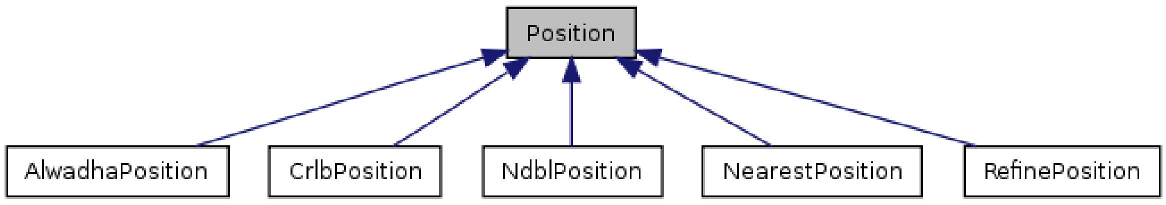


Figure 3.4. Inheritance diagram for Position class

Figure 3.5 shows the collaboration diagram for AlwadhaPosition class. Position class uses the MMSE and LocDisApp classes and ReferenceNode structure, which consists of two members, location variable and double variables, to store the measured distance. AlwadhaPosition class uses an array of this structure (ref_nodes_) to store the location information (location and distance) of neighbouring references. The Location class represents the X, Y and Z coordination of sensor nodes.

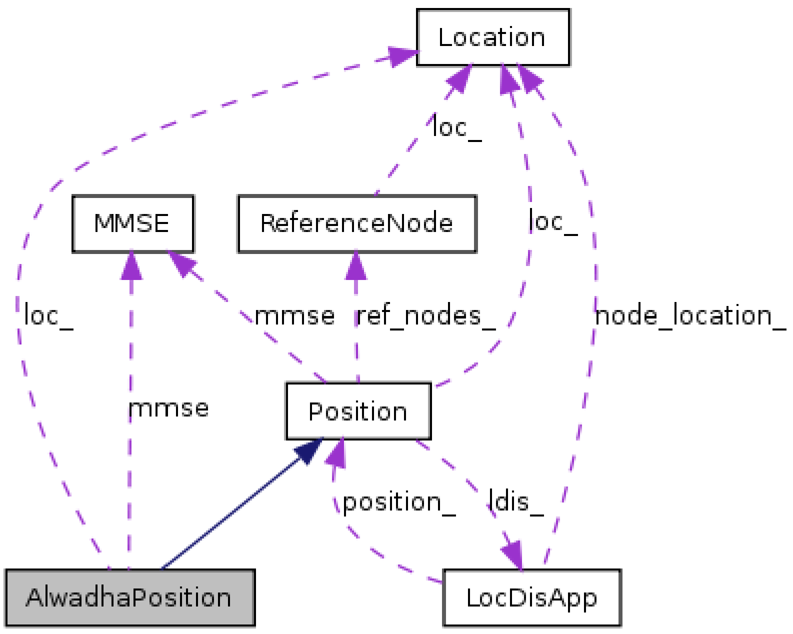


Figure 3.5. Collaboration diagram for Position and AlwadhaPosition classes

3.2.3.2 The Timer classes

Three timer classes were created, which are the ReqTimer, EstimateTimer and OutputTimer classes, as shown in Figure 3.6. These classes are derived from the TimerHandler class. These timer classes collaborate with LocDisApp to schedule several tasks during the run time. The ReqTimer is used to moderate how frequently sensor nodes broadcast a "location request" packet. After sending the "location request" packet, the EstimateTimer is used to schedule the estimation process after a specific delay, which is required to give "location response" packets enough time to receive from neighbouring references. The OutputTimer is used to schedule the action of recording the result, such as location error, number of references used and remaining energy, to the trace file.

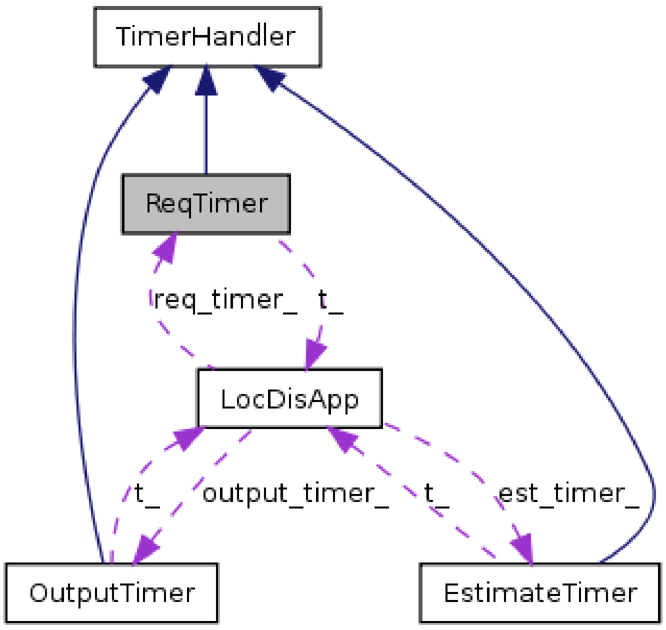


Figure 3.6. Collaboration diagram for the ReqTimer, EstimateTimer and OutputTimer classes

3.2.3.3 LocReqAgent and LocResAgent classes

As shown in Figure 3.7, these classes are derived from the Agent class. LocReqAgent constructs and broadcasts a "location request" packet. LocResAgent is responsible for handling the packets received, which could be "location request" or "location response" packets. Two new types of packet were created: firstly, a "location request" packet (PT_LOCREQ), which uses the new protocol-specific header (PSH) defined in the structure hdr_locreq, and secondly, a "location response" packet (PT_LOCRES), which uses the new PSH defined in the structure hdr_locres. LocReqAgent uses only hdr_locreq to construct "location request" packets. LocResAgent uses both of the headers' structures: it uses hdr_locreq to gain access and to process the received "location request" packets, while it uses the hdr_locres to construct the "location response" packets.

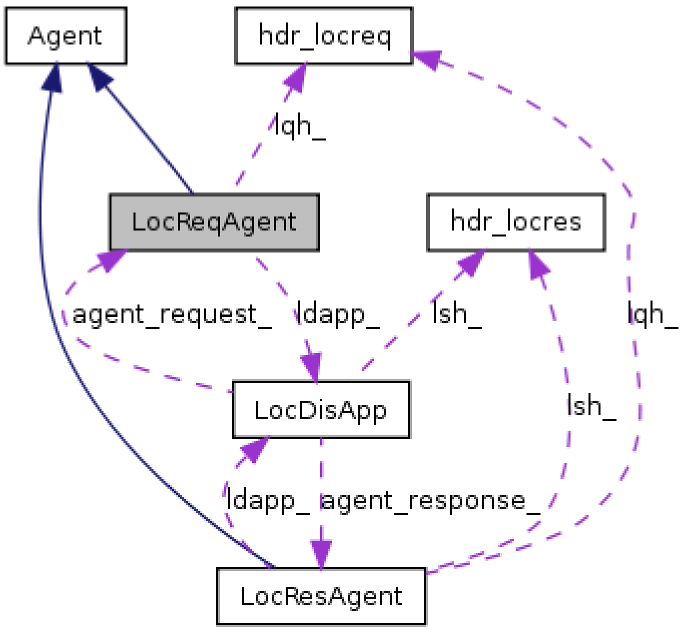


Figure 3.7. Collaboration diagram for LocReqAgent and LocResAgent classes

3.2.3.4 LocDisApp class

The LocDisApp class is derived from the Application class. As shown in Figure 3.8, this class collaborates with several classes, which are the three timers, the two agents, Location and Position classes. It uses the hdr_locres header structure to gain access to the received "location response" packets in order to process the included location information and estimate the node location. LocDisApp uses a vector of class ResData, which is used to store the location information received from neighbouring references. This information is the address, location and the probability of accuracy of the sending node and the power with which the packet is received.

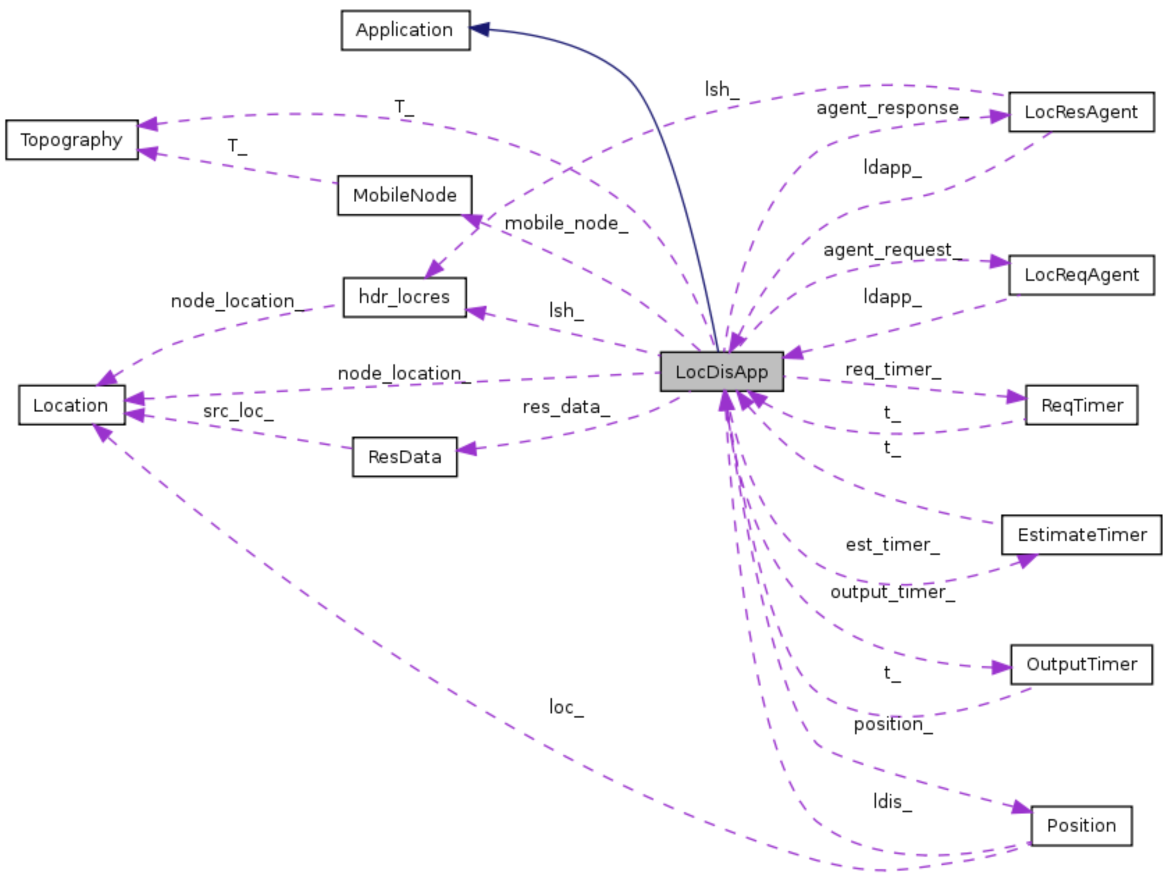


Figure 3.8. Collaboration diagram for LocDisApp class

3.2.3.5 The complete class hierarchy of the new modules

Figure 3.9 shows the complete collaboration diagram for the new classes. The complete procedures of the localisation process are as follows:

- LocDisApp schedules the OutputTimer with a specific delay (for example 1.0 second) to record the result into a trace file.
- LocDisApp schedules the ReqTimer with a specific delay, which determines how frequently the node broadcasts a "location request" packet.
- At the expiration time of ReqTimer, the LocDisApp invokes the LocReqAgent's method called broadcast( ) in order to broadcast a “location request” packet, schedules the EstimateTimer to start location estimation after a specific delay and reschedules the ReqTimer.

- LocReqAgent constructs a "location request" packet (PT_LOCREQ), which includes the required accuracy level, and then it broadcasts the packet to the neighbouring nodes.

- The LocResAgent of the references that received the "location request" packet requests from the LocDisApp the location information of the node, which is the node's location and the node's probability of accuracy. LocResAgent constructs a new "location response" packet (PT_LOCRES), which includes this information, and sends it back to the requesting node.

- The LocResAgent of the requesting node receives the "location response" packets from neighbouring references and then sends them to LocDisApp for more processing.

- LocDisApp extracts the required information from the packet received, namely the address, location and the probability of accuracy of the sending reference node and the power with which the packet is received, and then stores this information in a ResData vector.

- At the expiration time of EstimateTimer the LocDisApp invokes the AlwadhaPosition's method called estimate( ).

- AlwadhaPosition estimates the node location using the data stored in the ResData vector, based on the procedures explained in Section 3.1.

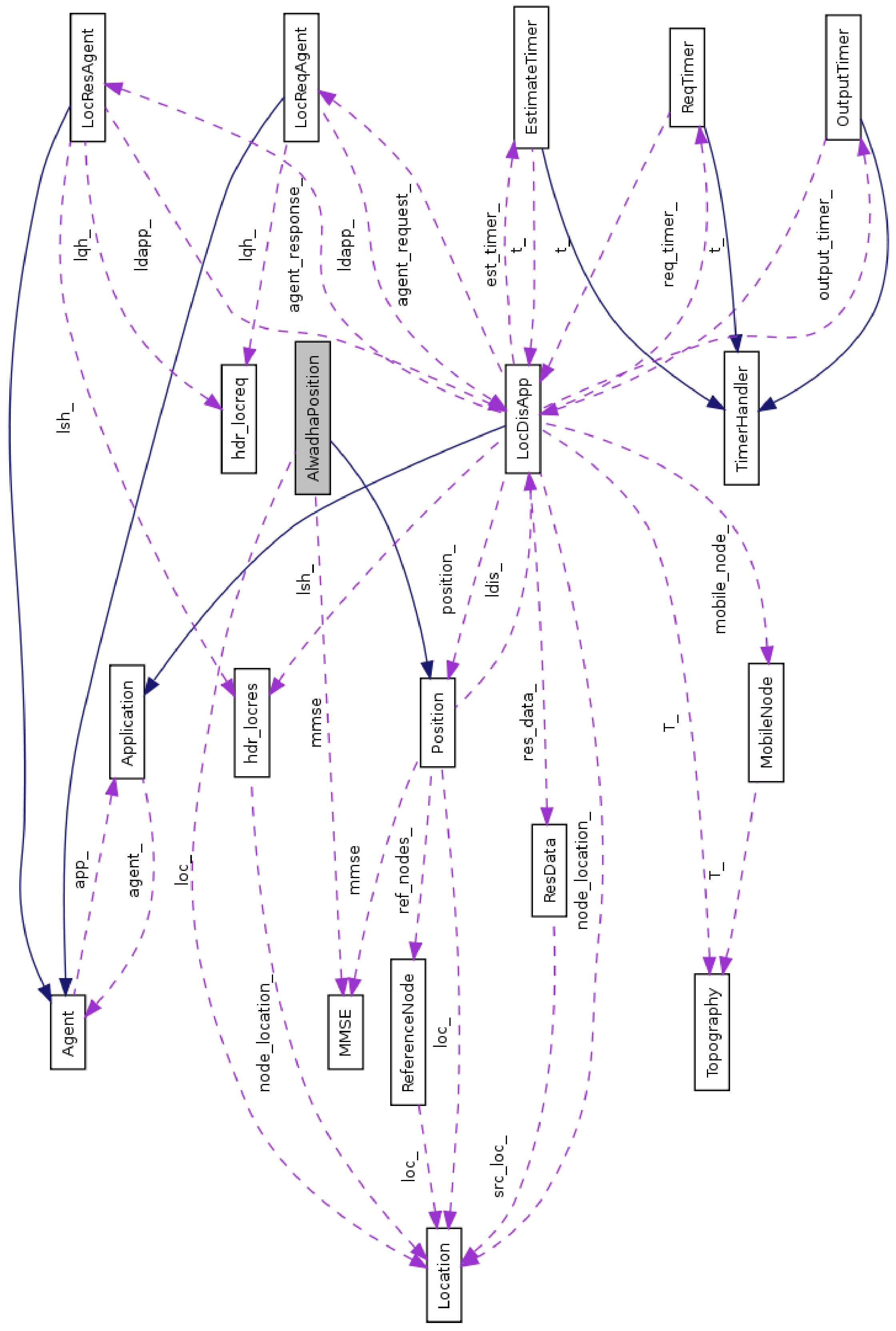


Figure 3.9. Collaboration diagram for the new classes

3.2.4 The structure of the new ns-2

Figure 3.10 Shows the structure of the new ns-2, where the files under the "location" directory represent the new files that were added to ns-2, while the other files (left-hand side) are the modified files.

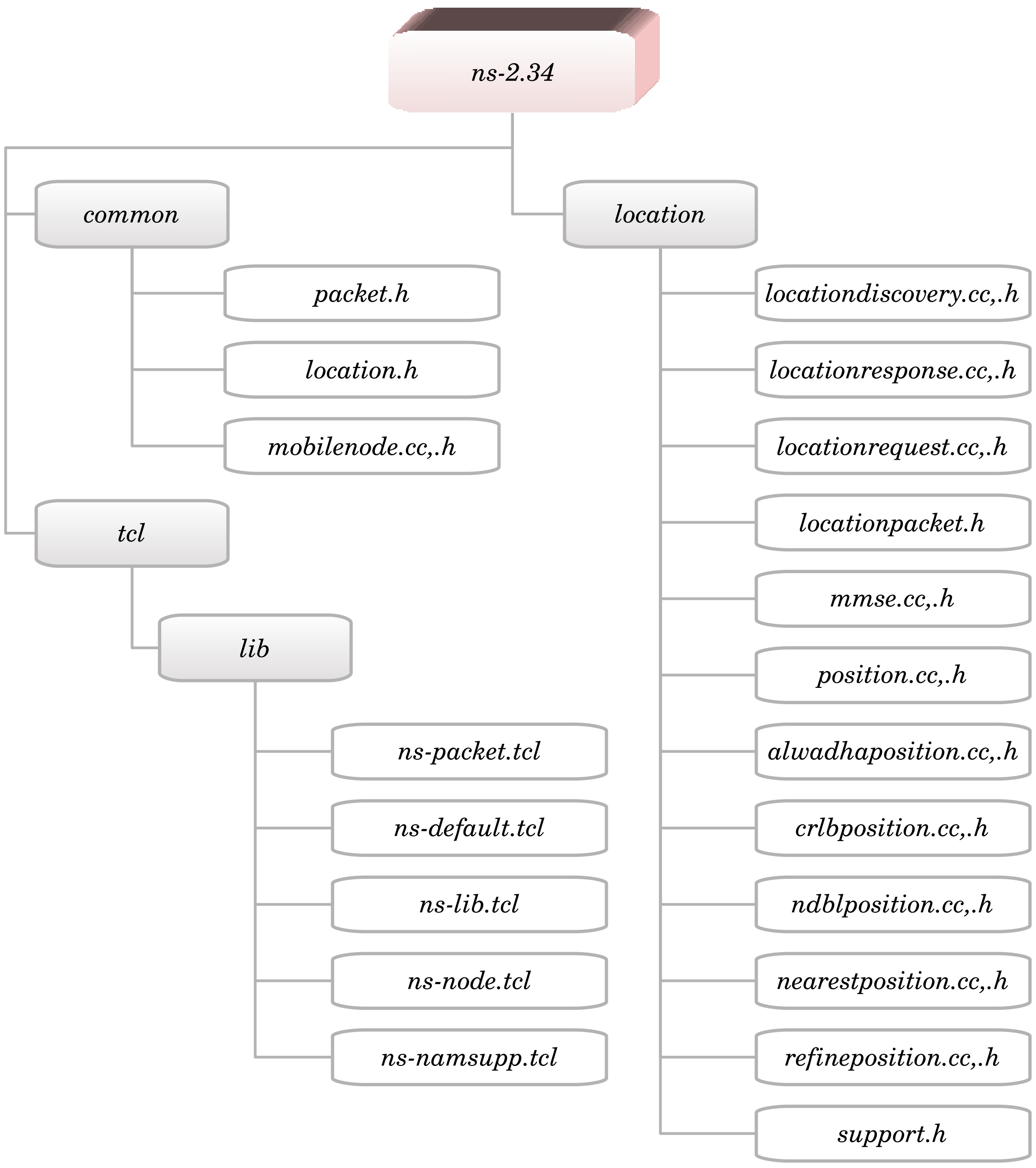


Figure 3.10. The structure of the new ns-2, showing the new files added to ns-2 (right-hand side) and the modified files (left-hand side)

The new classes that were discussed in the previous two sections are implemented in the files under the "location" directory. In addition to these new files, some other files were modified as follows:

- *common/packet.h:* Two packet types were created in the locationpacket.h file using two structures (hdr_locreq and hdr_locres). In order to use these two types of packet, their corresponding packet types (PT_LOCREQ and PT_LOCRES) were defined in the packet.h file.

- *common/location.h:* This file contains the Location class, which is used to represent the location coordination (X, Y and Z) of nodes. Some methods were added to this class, such as the getter and the setter of an individual coordinate, is_equal( ) method to check if two locations are the same and distance( ) method to find the distance between two locations.

- *common/mobilenode.cc, .h:* Two methods were added to these files. The first is to get an object to the topography. The second is to record the result in the trace file. The result could be the location error, number of references used, probability of accuracy and remaining energy.

- *tcl/lib/ns-packet.tcl:* In order to activate the new header classes, the OTcl class names were included in this file. The new OTcl header classes are "PacketHeader/LocReq" and "PacketHeader/LocRes", and so only "LocReq" and "LocRes" were added to the active protocol list.

- *tcl/lib/ns-node.tcl:* As mentioned before, from the localisation perspective, the node could be a beacon, reference or unknown. In order to specify the type of nodes, a new instvar, called "nodeAttribute_", and a new instproc to get the node attribute, called "attribute", were created.

- *tcl/lib/ns-lib.tcl:* In order to enable the simulator to deal with the node attribute, an instproc was created to set the instvar "attribute_". Within the "Simulator instproc create-wireless-nodes" the node is allowed to set its attribute ($node set nodeAttribute_ $attribute_).

- *tcl/lib/ns-namsupp.tcl:* During the simulation, when the unknown nodes estimate their position they change their colour (for instance to red). In order to enable the nodes to change their colour after running the simulator, the “Node instproc color” was modified within this file.

- *tcl/lib/ns-default.tcl:* Sometimes it is necessary to bind some variables in both hierarchies (i.e. interpreted and compiled hierarchies). The default value of these bind variables is initialised in the ns-default.tcl file. Several variables were bound, such as the packet size, the request frequency (reqFreq_), the showColor_ variable to enable showing the colour of the nodes based on their attribute and the subset_ variable to specify which localisation algorithm should be applied (e.g. ALWadHA, CRLB or Nearest).

3.2.5 Guidelines for running the simulation

Using the extended ns-2 does not require new knowledge or writing a specific code to run the simulator. Normal users who have the basic knowledge to run a simple wireless network using ns-2 are able to write a simple OTcl script to simulate the proposed localisation system. This section gives some guidelines for configuring the localisation simulation. It assumes that the reader is familiar with setting up wireless mobile network simulations in ns-2. Therefore, it will not explain the entire simulation procedures; rather, it will show how to configure nodes to simulate localisation. However, the reader is referred to [60] for some tutorials about configuring wireless networks.

At the beginning of the simulation there are only two types of nodes: beacons and unknowns. Each of them has a different configuration, as shown in Figure 3.11. Beacons already know their location, so they should be attached only with LocResAgent. Unknowns should be attached with LocReqAgent in order to allow them to broadcast "location request" messages. After estimating their position, unknowns could become a reference for other nodes, and so they are also attached with LocResAgent. These two types of agents should be attached to LocDisApp.

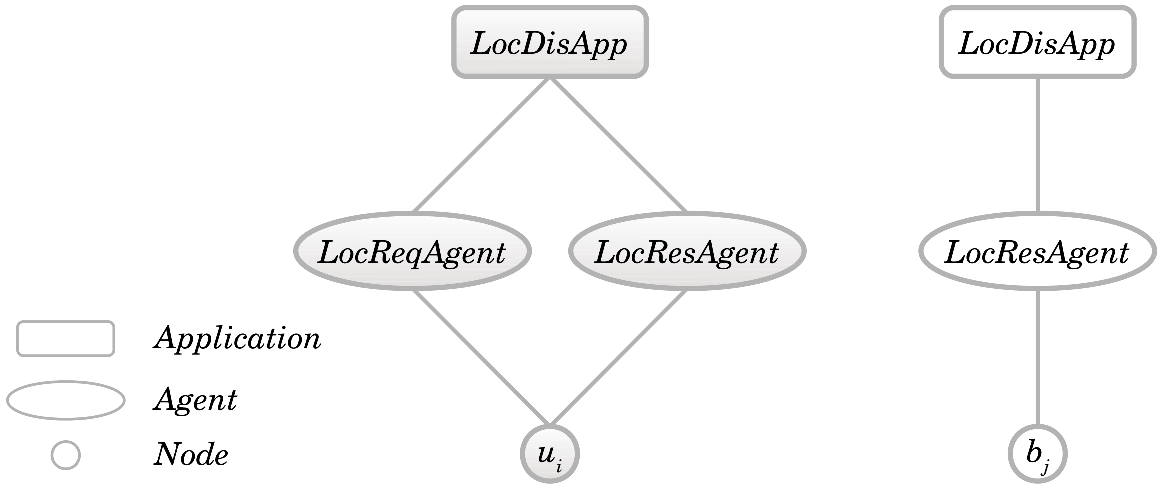


Figure 3.11. Node configuration, where u_i is an unknown node and b_j is a beacon node

Before creating the nodes it is necessary to specify the nodes' configuration, such as the routing protocol, MAC type and so on. In addition to these configurations it is also necessary to specify the attribute of the nodes, which can be done with the help of the command “node-config”

```
set val(nn)      10                ;# number of mobilenodes
set val(nu)      7                 ;# number of unknown nodes
set val(nb)      3                 ;# number of beacon nodes

# Beacon nodes:
# Nodes configuration
set val(attr)    BEACON            ;# node attribute
$ns_ node-config -attribute $val(attr)
# Nodes creation
for {set i 0} {$i < $val(nb)} {incr i} {
      set node_($i) [$ns_ node]
}

# Unknown nodes
# Nodes configuration
set val(attr)         UNKNOWN
$ns_ node-config -attribute $val(attr)
# Nodes creation
for {set i $val(nb)} {$i < $val(nn) } {incr i} {
      set node_($i) [$ns_ node]
}
```

The next step is to create the required agents and application based on the configuration shown in Figure 3.11. Beacon nodes require only LocResAgent and LocDisApp; this can be done as shown below:

```
# Beacon nodes have only response agent
for {set i 0} {$i < $val(nb)} {incr i} {
      # Setup the response agent
      set lres_($i) [new Agent/LocRes]
      $ns_ attach-agent $node_($i) $lres_($i)

      # Setup the location-discovery application
      set ldis_($i) [new Application/LocDis]
      $ldis_($i) attach-agent $lres_($i)
}
```

Unknown nodes also require LocReqAgent, as shown below:

```
# Unknown nodes have both request and response agent
for {set i $val(nb)} {$i < $val(nn) } {incr i} {
      # Setup the request agent
      set lreq_($i) [new Agent/LocReq]
      $ns_ attach-agent $node_($i) $lreq_($i)

      # Setup the response agent
      set lres_($i) [new Agent/LocRes]
      $ns_ attach-agent $node_($i) $lres_($i)

      # Setup the location-discovery application
      set ldis_($i) [new Application/LocDis]
      $ldis_($i) attach-agent $lreq_($i)
      $ldis_($i) attach-agent $lres_($i)
}
```

Finally, the location-discovery applications should be started at a specific time:

```
# Start the locdis applications
for {set i 0} {$i < $val(nn)} {incr i} {
      $ldis_($i) set random_ 1
      $ldis_($i) set subset_ 3
      $ldis_($i) set showColor_ 1
      $ns_ at 0.0 "$ldis_($i) start"
}
```

If the user does not want to use the default value of the bind variables he can change the setting of this variable before starting the applications. For instance, in the previous code the random_ variable was set to 1 to all the LocDis applications to start broadcasting the "location request" messages at a random time instead of starting immediately (random_ = 0). The subset_ variable specifies the localisation algorithm that should be applied to estimate the node location; the value of three refers to the ALWadHA algorithm. The variable showColor_ is used in all the unknowns to change their colour after they estimate their location; setting this variable to zero disables this feature.

3.2.6 Manipulate output files

After simulation, ns-2 outputs either text-based or animation-based simulation results. As shown in Figure 3.12, several tools could be used to interpret these results graphically or interactively. The Network AniMator (NAM) [60] is an animation tool that uses the animation-based results to view the network simulation traces and real-world packet traces. NAM supports topology layout, packet level animation and various data inspection tools.

Text-based results consist of a lot of details on events that occur at the network, such as sending or receiving packets, nodes' movement and nodes' remaining energy. A new method was written that records the localisation-related information to the trace file. This information includes the localisation error, number of references used, remaining energy and number of iterations. To analyse particular data such as localisation error, the relevant information needs to be extracted from the traces and transformed to a more easily conceived presentation. Perl language [64] was used for this purpose. Perl stands for “Practical Extraction and Report Language”. Perl can be used to filter and to process the ASCII trace files in Unix. For example, a simple Perl script could be written to estimate the average localisation error of all nodes at each second.

The Gnuplot software was used to represent the relevant results graphically. Gnuplot [65] is a portable command-line-driven graphing utility for Linux, as well as many other platforms.

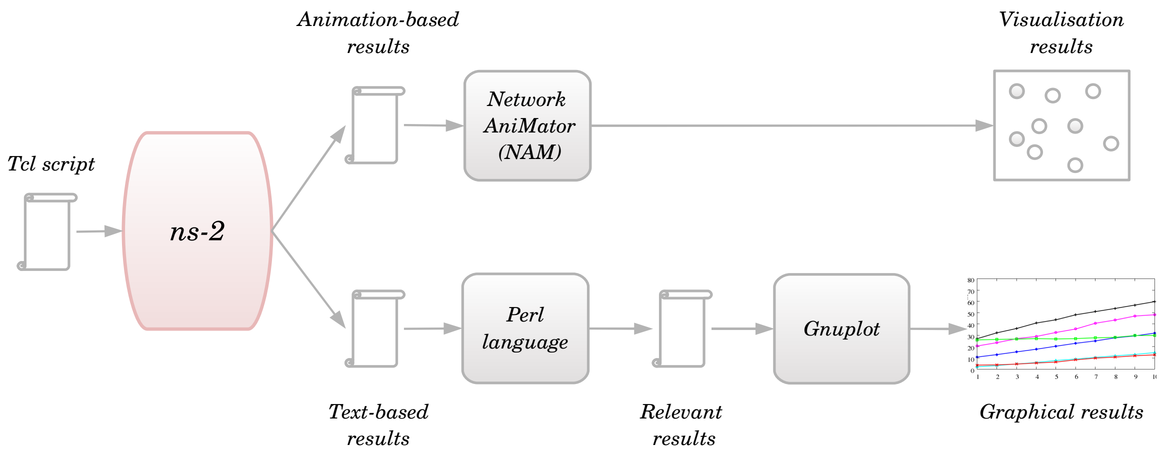


Figure 3.12. Tools used to manipulate the result

3.3 SIMULATION

This section evaluates the ALWadHA algorithm, looking at the effects of nodes' deployment, nodes' density and network size on the estimation error and the number of references used. It will examine the impact of increasing the distance-measurement error on the accuracy of the ALWadHA algorithm. It will compare the performance of the ALWadHA algorithm with other popular schemes.

3.3.1 Localisation algorithms

Five localisation algorithms, in addition to the ALWadHA algorithm, have been implemented for the performance comparison, using the same assumptions. These algorithms will be explained briefly and then a comparison of them will be presented in Table 3.2, where each algorithm is classified based on the number of estimations and the set of reference categories used. The table also shows how each algorithm selects the subset of references (S_i) and how it specifies the number of references ($C(S_i)$), which is either predefined manually or specified dynamically, based on certain conditions at run time.

3.3.1.1 General multilateration

The node uses all the available references to estimate its position $(S_i = R_i)$ using the basic multilateration method. Two algorithms based on this method were implemented.

The first one is based on the *single-estimation* approach (M_Single), while the second one is implemented based on the *successive-refinement* approach (M_Refine).

3.3.1.2 Nearest three references

The node selects a subset of three references to estimate its position [37]. The selected references have the minimum measured distance to the node. The selected subset is:

$$S_i = S_i^k, \text{ where } S_i^k \in \varsigma \text{ and } S_i^k \text{ has } max \ (\sum_{j \in S_i^k} 1 - \frac{\hat{d}_{i,j}}{r_{tx}}) \ . \tag{3.11}$$

3.3.1.3 CRLB-based algorithm

The proposed algorithm in [13] tries to minimise the localisation error by selecting a subset of references based on the measured distance and CRLB on localisation error. The selected subset is

$$S_i = S_i^k, \text{ where } S_i^k \in \varsigma \text{ and } S_i^k \text{ has } max \ (\sum_{j \in S_i^k} P_{crlb}^j) \ . \tag{3.12}$$

3.3.1.4 Node distribution-based localisation algorithm

The authors of [43] modify the refinement dwMDS algorithm [42] by simplifying the computation and reducing the processing time. The proposed algorithm is called the node distribution-based localisation (NDBL) algorithm. In the NDBL algorithm each node selects the references that are within a measurable range

$$S_i = \{r_j, \text{ where } \hat{d}_{i,j} \leqslant r_{tx}\} \ . \tag{3.13}$$

In fact, this selection criterion does not exclude many references and it could be equivalent to the all-references set $S_i = R_i$. The algorithm starts by estimating the multi-hop distances to the beacons and then using the trilateration method to estimate a rough initial position. This inevitably implies error, due to compounding of error from the approximated measurement at each hop. In the next refinement phases, all the computation is based on the measured distances to the one-hop references. For the sake of comparison and enhancing the accuracy of the initial position, the multilateration method is used based on the measured distances to the one-hop references to estimate the initial position, without any modification in the refinement phases.

Table 3.2. Implemented localisation algorithms

Localisation	**Estimations**		**References**		**S$_i$**	**C(S$_i$)**
algorithms	**Single**	**Refinement**	**All**	**Subset**		
M_Single	√		√		R$_i$	-
M_Refine		√	√		R$_i$	-
Nearest	√			√	$max\ (\sum_{j\in S_i^k} 1-\frac{\hat{d}_{i,j}}{r_{tx}})$	Manual
CRLB	√			√	$max\,(\sum_{j\in S_i^k} P^j_{crlb})$	Manual
NDBL		√		√	$\{r_j, \text{where } d_{i,j}\leqslant r_{tx}\}$	Dynamic
ALWadHA		√		√	S_i^k with $min(c)$ and $(1-\frac{1}{\sum_{j\in S_i^k} P^j_{acc}}) \geqslant P_{min}$	Dynamic

3.3.2 Performance comparison

The performance of each algorithm was then evaluated based on two metrics: The first was the localisation error, which reflects on the location accuracy. The second was the number of references used in the position estimation. (This has a significant impact on the computation cost, since the complexity of the localisation algorithms is proportional to the number of references used.)

3.3.2.1 Localisation error

The mean error is estimated every second for all knowns as a ratio of transmission range. The mean error at a specific time *t* is equal to the summation of the location error of all knowns, divided by the number of these knowns, and then it is divided by the transmission range as follows:

$$Mean\ error_t = (\frac{1}{n}\sum_{i=1}^{n} \| \hat{z}_i - z_i \|)\ \frac{1}{r_{tx}}\ 100\% \tag{3.14}$$

where n is the total number of knowns at a specific time t $\left(n = C(K) \right)$.

3.3.2.2 Number of references ($C(S_i)$)

Increasing the number of references could enhance the accuracy of the position estimation. However, it increases the complexity of computations [13]. The multilateration method uses all the available references. The number of references for the Nearest and CRLB algorithms was predefined manually by three references. This number was selected to show the impact of using the minimum required number of references on the performance of these two algorithms, and also to make them comparable with the ALWadHA algorithm, which uses almost the same number of references $\left(C(S_i) \approx 3 \right)$. The ALWadHA and NDBL algorithms specify the number of references dynamically at each iteration, based on a specific criterion, as shown in Table 3.2. The average number of references used at a specific time t is calculated as follows:

$$\# \, of \; References_t = \frac{1}{n} \sum_{i=1}^{n} C(S_i) \tag{3.15}$$

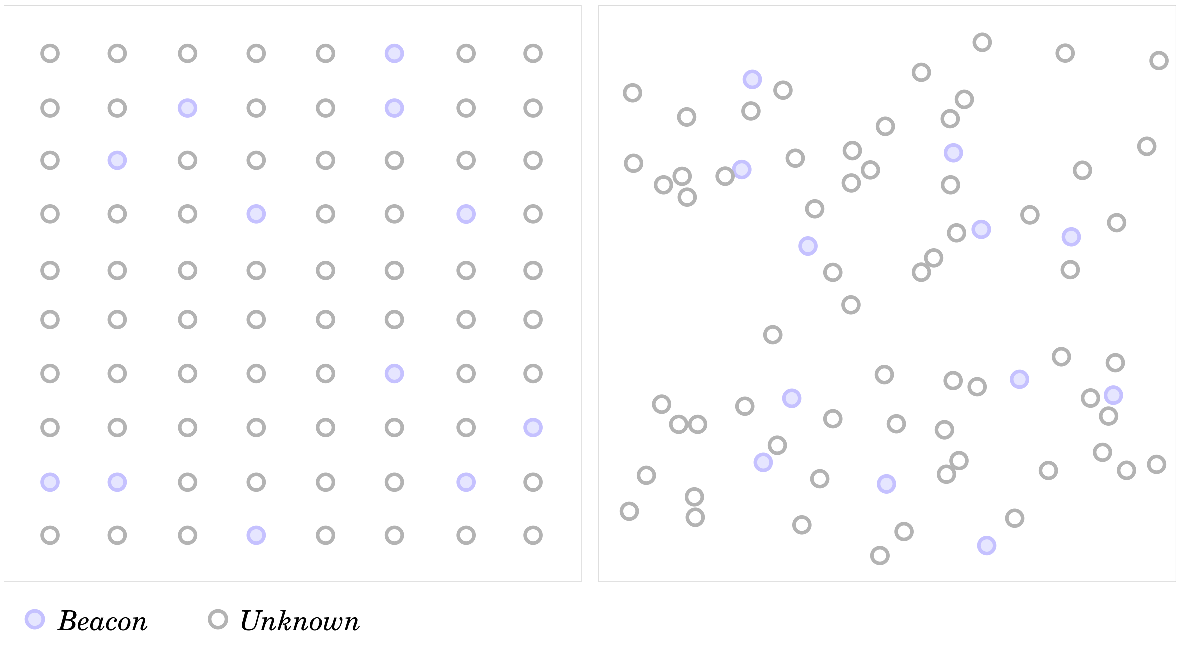
where n is the total number of knowns at a specific time t.

3.3.3 Setup and environment

The localisation algorithms were evaluated using the network simulator (ns-2). In order to satisfy the applicability of the proposed algorithm in large-scale WSNs, the field was divided into several sub-areas, each area containing the same number of unknowns and beacons without ignoring the randomness of node distribution. All nodes had a limited transmission range (r_{tx}) of 50 m. At each experiment the simulation was run 100 times; the duration of each run was 600 sec (the total duration was 60 000 sec), and at each run nodes were redistributed randomly in different places (using a different seed value). RSS was used to measure the distance between nodes. However, to simulate noise, each measured distance was disturbed by a normal random variant with the following settings: a mean of 0.1% of the measured distance and a standard deviation of 1% of the measured distance. To check the impact of increasing the distance-measurement error on the localisation algorithms, the standard deviation was varied from 1% to 10% of the measured distance.

Two types of deployment were used: grid and random deployment, similar to the networks shown in Figure 3.13. The grid deployment had the following characteristics:

- Deployment area: $200\,m \times 200\,m$ divided into four sub-areas
- Number of beacons: 12; each sub-area had three beacons (0.75 beacon per r_{tx}^2)
- Number of unknowns: 68; each sub-area had 17 unknowns (4.25 unknowns per r_{tx}^2)
- Beacon distribution: each sub-area was divided into 20 rectangles, then three rectangles were selected randomly where the three beacons were placed in their centres.
- Unknowns distribution: the unknowns were placed in the centre of the remaining empty rectangles.



a. Grid deployment b. Random deployment

Figure 3.13. Examples of node distribution

The random deployment was based on the following characteristics:

- Deployment area: $200\,m \times 200\,m$ divided into four sub-areas
- Number of beacons: 12; each sub-area had three beacons (0.75 beacon per r_{tx}^2)
- Number of unknowns: 64; each sub-area had 16 unknowns (four unknowns per r_{tx}^2)
- Beacon distribution: three beacons distributed randomly in each sub-area
- Unknowns distribution: each sub-area was divided into four regions $(50\text{m} \times 50\text{m})$, and then four unknowns were distributed randomly in each region.

Table 3.3. Experiments setup

Factors	Fig.	Deployment	Area	# B	# U	Notes	
Node deployment	3.14	Random	200 x 200	12	64	No error	
	3.15	Random	200 x 200	12	64	Sd = 1 %	
	3.16	Grid	200 x 200	12	68		
Node density	3.17	Random	200 x 200	12	64	4 *U* per r_{tx}^2	0.75 *B* per r_{tx}^2
	3.18	Random	200 x 200	12	96	6 *U* per r_{tx}^2	
	3.19	Random	200 x 200	12	160	10 *U* per r_{tx}^2	
	3.20	Random	200 x 200	12	64	0.75 *B* per r_{tx}^2	4 *U* per r_{tx}^2
	3.21	Random	200 x 200	16	64	1 *B* per r_{tx}^2	
	3.22	Random	200 x 200	20	64	1.25 *B* per r_{tx}^2	
Network size	3.23	Random	100 x 100	3	16	Small	0.75 *B* per r_{tx}^2 4 *U* per r_{tx}^2
	3.24	Random	200 x 200	12	64	Normal	
	3.25	Random	600 x 600	108	576	large	
Standard deviation	3.26.a	Grid	200 x 200	12	68	Sd = 1 % to 10 %	
	3.26.b	Random	200 x 200	12	64		
	3.27	Grid	200 x 200	12	68	Sd = 10 %	

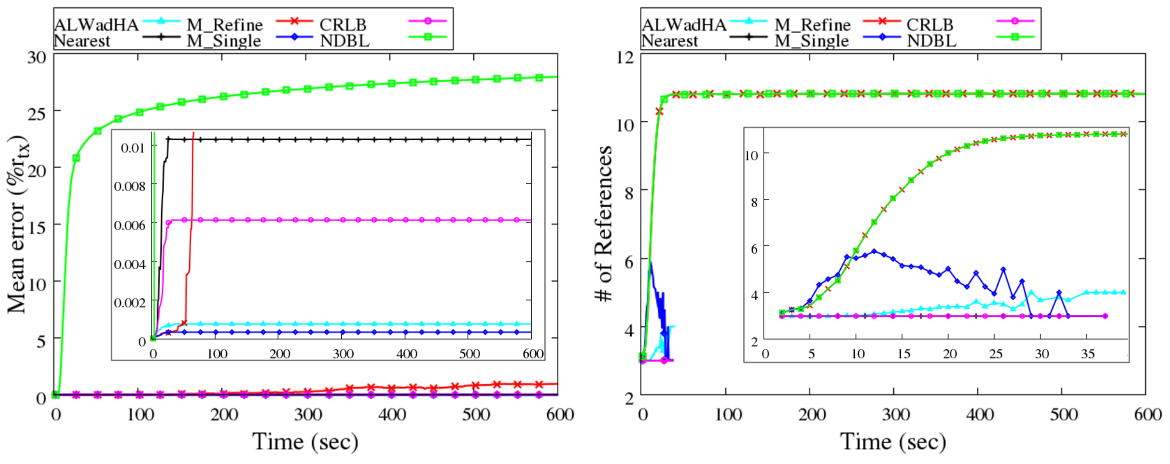
However, some of the random deployment characteristics could be different for some experiments, such as the number of beacons or unknowns or the network size. The exact setting for those experiments that have different characteristics will be provided. The reader can refer to Table 3.3 for the complete setup environment of each experiment that was performed in preparing this chapter.

3.3.4 Results and comparisons

Several experiments were performed to evaluate the localisation algorithms, based on three factors: node deployment, node density and network size. In each experiment, these localisation algorithms were evaluated and compared based on the two metrics mentioned above. Finally, the impact of increasing the distance-measurement error on the accuracy of these algorithms was examined.

3.3.4.1 Node deployment

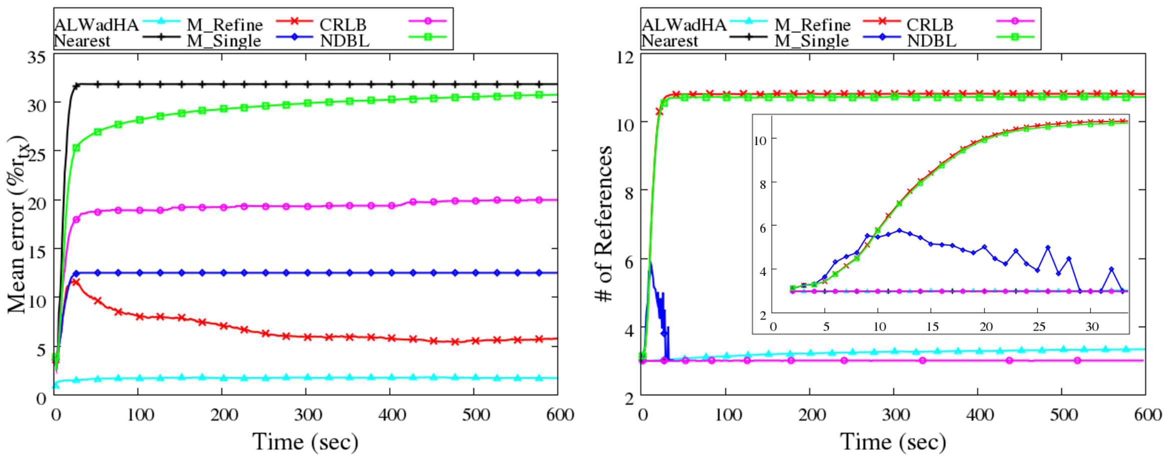
In the first two experiments, the nodes were deployed randomly, based on the characteristics mentioned earlier. In the first experiment, the nodes were deployed randomly without adding any error to the distance measurement, in order to check the error that could be introduced by the localisation algorithm itself. Figure 3.14.a shows the mean error of each algorithm versus the time, while Figure 3.14.b shows the average number of references used by each algorithm. Figure 3.14.a shows that the mean error of the M_Single, Nearest, CRLB and ALWadHA algorithms was close to zero, M_Single performed well in this experiment, but it performed much less well in the rest of the experiments. The mean error of the M_Refine algorithm increased gradually because of the computation error that accumulated during the refinement phases. This type of error does not affect the M_Single, Nearest and CRLB algorithms because they do not perform any refinement and the node stops directly after it gets its first position estimate. This type of cumulative computation error does not affect the ALWadHA algorithm because of the termination criterion used by the algorithm, where the nodes get their final positions within the first 39 seconds, as shown in Figure 3.14.b. The NDBL algorithm has a higher mean error compared with other algorithms. This error is caused by the algorithm itself and the cumulative error during the refinement phases. This result shows the importance of simplifying the localisation algorithm and using a proper termination criterion.



a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.14. Random deployment without distance error

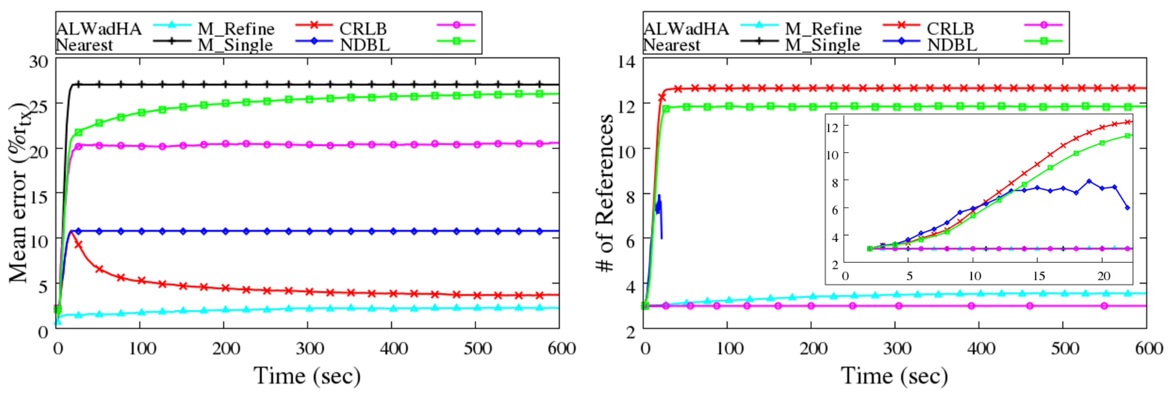
The previous experiment was repeated with distance error. Figure 3.15 shows that the ALWadHA algorithm outperformed other algorithms in terms of position accuracy. The multilateration algorithm (M_Refine) enhanced its estimation over time, because at the beginning a low number of references was available, then over time the nodes started to know their location, which increased the number of references. Using all these references enhanced the accuracy of the multilateration method. After about 25 seconds the M_Refine algorithm started to use more than 10 references to estimate the node position (Figure 3.15.b), while the ALWadHA algorithm kept using a low number of references with an average equal to 3.24, which is very close to the minimum possible number. It could be remarked that using more references could significantly increase the complexity of the algorithm. Figure 3.15.b also shows that the NDBL algorithm uses almost the same number of references as the M_Refine algorithm, which means that its reference-selection criterion does not always exclude references in this scenario.



a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.15. Random deployment with distance error

The third experiment was performed with the existence of distance error based on a grid deployment. From Figure 3.16 it can be noticed that the ALWadHA algorithm has less mean error compared with other algorithms. If this result is compared with experiment two (Figure 3.15), it is noticeable that in the random deployment all algorithms apart from the CRLB and ALWadHA algorithms have a higher mean error. The reason is that in grid deployment most of the nodes have the same number of neighbours, while in random deployment nodes could have a different number of neighbours; this leads to lower accuracy caused by nodes having a lower number of references. This supplies the motivation to rely on proper selection of references, rather than on using a high number of references to enhance the performance of the localisation algorithm. Figure 3.16.b shows that the NDBL algorithm used a lower number of references than the M_Refine algorithm, because in this grid deployment several references are close to the boundary of the transmission range, and so the addition of distance error takes them out of the transmission range and allows the node to exclude them from its subset of references.



a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.16. Grid deployment with distance error

The results of these three experiments are summarised in Table 3.4, where the mean error is recorded at the end of the run time, while the number of references represents the average number of references that were used during the run time.

Table 3.4. Performance comparison of different deployments

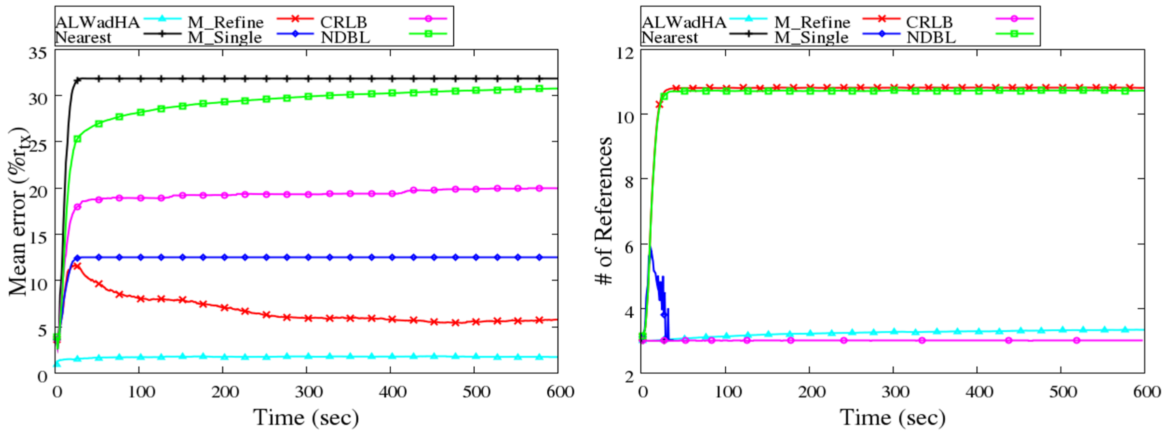
Deployment	**Random (no error)**		**Random (error)**		**Grid (error)**	
Algorithms	**Mean error**	**Average # Ref**	**Mean error**	**Average # Ref**	**Mean error**	**Average # Ref**
ALWadHA	0.0008	3.39	1.75	3.24	2.28	3.42
M_Single	0.0004	4.38	12.53	4.38	10.81	5.89
M_Refine	0.9699	10.67	5.77	10.67	3.73	12.46
CRLB	0.0061	3	19.99	3	20.57	3
NDBL	27.9454	10.67	30.77	10.58	26.01	11.66
Nearest	0.0103	3	31.86	3	27.02	3

3.3.4.2 Node density

Initially, the network consisted of two types of nodes, beacons and unknowns. This section examines the impact of changing the number of these nodes on the performance of the localisation algorithms. First, a fixed number of beacons were used while changing the number of unknowns, then the number of unknowns were fixed while changing the number of beacons. In all the experiments done in this section, nodes were distributed randomly in a $200\,m \times 200\,m$ field, based on the same characteristics mentioned above for random deployment except for the number of nodes. To examine the unknowns' density, 12 beacons were used (0.75 beacon per r_{tx}^2) and the unknowns' density was varied as follows:

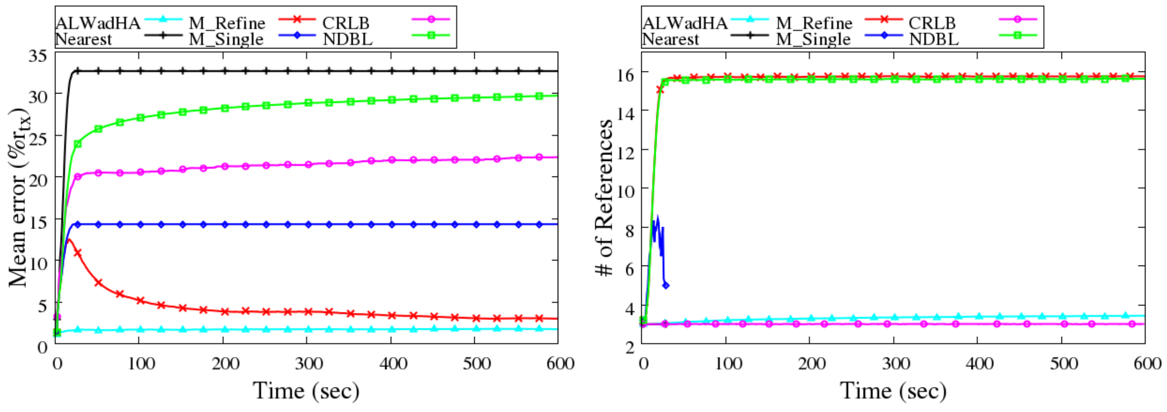
- 64 unknowns (four unknowns per r_{tx}^2)
- 96 unknowns (six unknowns per r_{tx}^2)
- 160 unknowns (10 unknowns per r_{tx}^2).

By comparing Figure 3.17, 3.18 and 3.19 it can be seen that the M_Refine and NDBL algorithms enhance the location estimation accuracy slightly by increasing the nodes' density. On the other hand, these two algorithms use around 25 references in the high unknown density (Figure 3.19.b). Using such a high number of references would dramatically increase the overheads for computation and communication. Figure 3.19.b shows that the NDBL algorithm uses almost the same number of references as the M_Refine algorithm. This result confirms the disadvantage of using the elimination criterion that was mentioned earlier, and shows the impact of using a real selection criterion that is able to select a low number of the references that contribute most to high location estimation accuracy. When using different unknowns' density, the smart reference-selection method used by the ALWadHA algorithm enables the nodes to estimate their location with the best accuracy, compared with other algorithms. The ALWadHA algorithm uses on average fewer than 3.37 references.



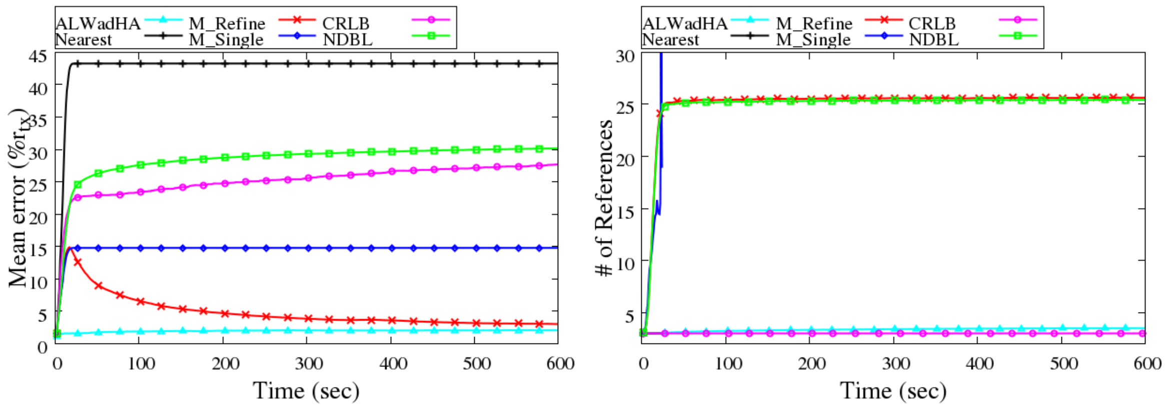
a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.17. Unknowns' density: 4 unknowns per r_{tx}^2



a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.18. Unknowns' density: 6 unknowns per r_{tx}^2



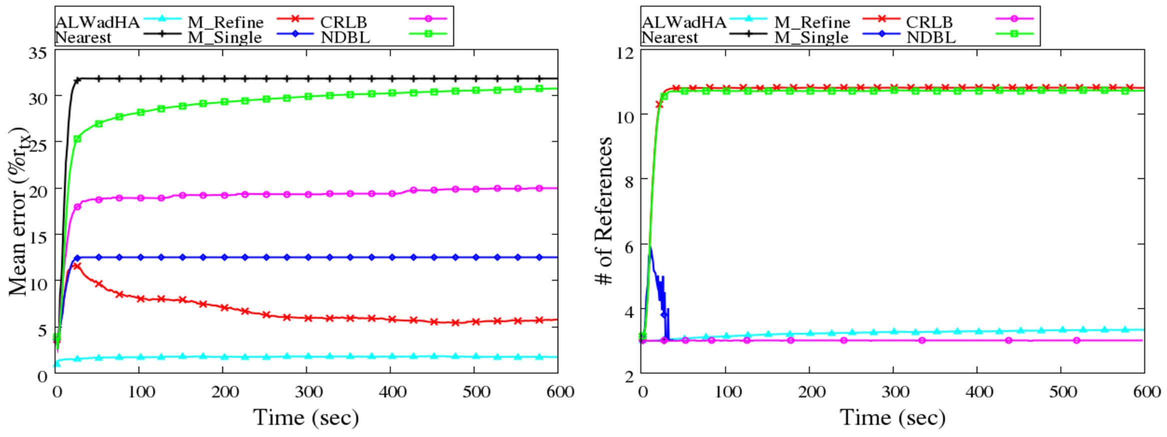
a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.19. Unknowns' density: 10 unknowns per r_{tx}^2

In the second part of this section, 64 unknowns were used (four unknowns per r_{tx}^2) and the beacons' density was varied as follows:

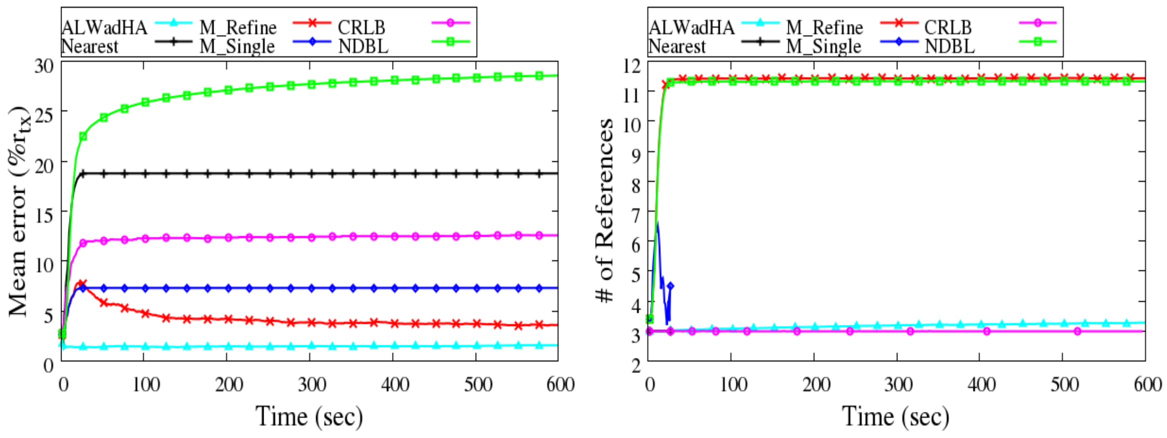
- 12 beacons (0.75 beacon per r_{tx}^2)
- 16 beacons (1.0 beacon per r_{tx}^2)
- 20 beacons (1.25 beacon per r_{tx}^2).

As expected, Figure 3.20, 3.21 and 3.22 show that increasing the beacons' density enhances the accuracy of all the localisation algorithms. However, the NDBL algorithm enhances its accuracy only slightly, compared with the other algorithms. When using a different beacon density, the ALWadHA algorithm achieves the best accuracy of all the algorithms. An interesting observation about ALWadHA is that the average number of references used is reduced by increasing the beacon density. This is because from the smart reference-selection method perspective, more beacons mean more references with a high probability of accuracy, which enables the nodes to select a smaller subset of references fulfilling the condition $P_{acc} \geqslant P_{min}$.



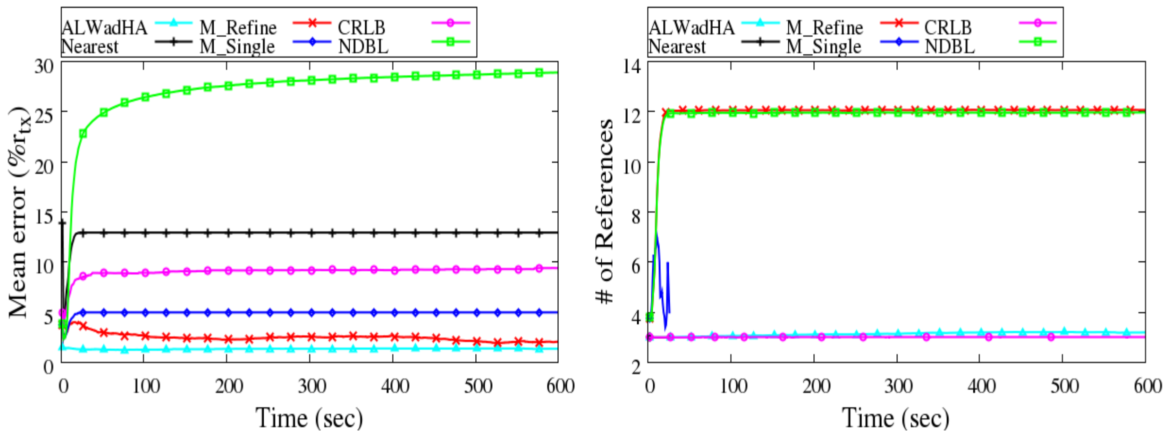
a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.20. Beacons' density: 0.75 beacons per r_{tx}^2



a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.21. Beacons' density: 1 beacon per r_{tx}^2



a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.22. Beacons' density: 1.25 beacons per r_{tx}^2

A performance comparison of the localisation algorithms using different unknowns and beacon density is shown in Table 3.5, where *B* and *U* refer to beacons and unknowns respectively.

Table 3.5. Performance comparison of different node densities

Density # B	**12**		**12**		**12**		**16**		**20**	
Density # U	**64**		**96**		**160**		**64**		**64**	
Algorithms	Mean error	Avg. # Ref	Mean error	Avg. # Ref	Mean error	Avg. # Ref	Mean error	Avg. # Ref	Mean error	Avg. # Ref
ALWadHA	1.75	3.24	1.74	3.31	2.09	3.37	1.58	3.17	1.37	3.14
M_Single	12.53	4.38	14.33	6.43	14.80	11.47	7.31	4.67	5	5.02
M_Refine	5.77	10.67	3.01	15.48	3.05	25.1	3.64	11.29	2.07	11.93
CRLB	19.99	3	22.36	3	27.69	3	12.58	3	9.41	3
NDBL	30.77	10.58	29.75	15.35	30.18	24.9	28.57	11.2	28.89	11.83
Nearest	31.86	3	32.7	3	43.27	3	18.77	3	12.95	3

3.3.4.3 Network size

To check the impact of changing the network size on the performance of localisation algorithms, three experiments were performed using different network sizes (small, normal and large), where nodes were distributed randomly using 0.75 beacon per r_{tx}^2 and four unknowns per r_{tx}^2 The size of networks and the number of nodes are specified as follows:

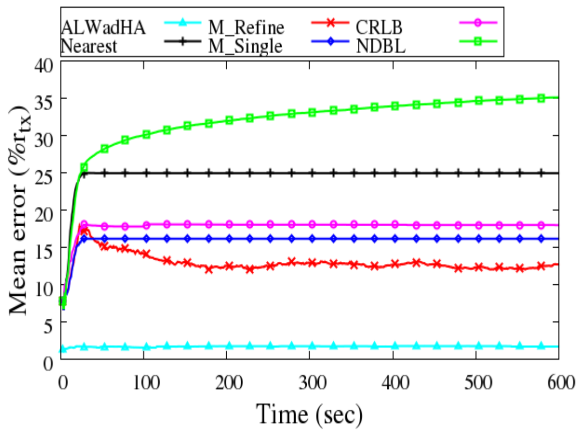
- Small: $100\,m \times 100\,m$ field with three beacons and 16 unknowns
- Normal: $200\,m \times 200\,m$ field with 12 beacons and 64 unknowns
- Large: $600\,m \times 600\,m$ field with 108 beacons and 576 unknowns.

The large network follows the same characteristics as those mentioned earlier for random deployment, except that the network is divided into 36 sub-areas, while the small network consists of only one sub-area. Figure 3.23 shows that the CRLB and Nearest algorithms perform better in small networks; increasing the size of the network reduces the accuracy of these algorithms. In contrast, the accuracy of the M_Refine and NDBL algorithms is enhanced by increasing the network size, because in a small network (Figure 3.23.b) the number of references used by these two algorithms is lower than in normal and large networks. This small number of references is not enough for these two algorithms to enhance the accuracy of position estimation. The accuracy of the ALWadHA algorithm is almost the same in the three experiments.

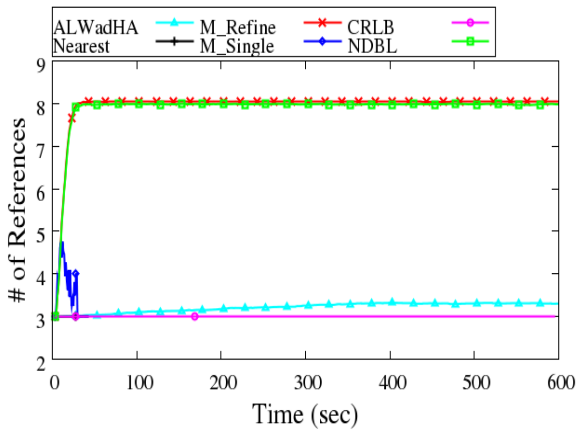
Table 3.6 shows the mean error and average number of references used by the localisation algorithms for the three experiments done in this section.

Table 3.6. Performance comparison of different network sizes

Network size	**Small**		**Normal**		**Large**	
Algorithms	**Mean error**	**Average # Ref**	**Mean error**	**Average # Ref**	**Mean error**	**Average # Ref**
ALWadHA	1.7	3.22	1.75	3.24	1.70	3.25
M_Single	16.15	3.77	12.53	4.38	16.75	4.86
M_Refine	12.71	7.95	5.77	10.67	4.42	12.55
CRLB	17.98	3	19.99	3	23.67	3
NDBL	35.11	7.89	30.77	10.58	26.96	12.43
Nearest	24.94	3	31.86	3	40.95	3

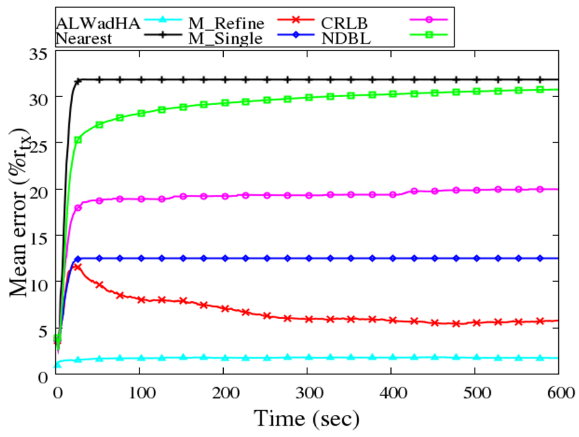


a. Mean error as a ratio of transmission range

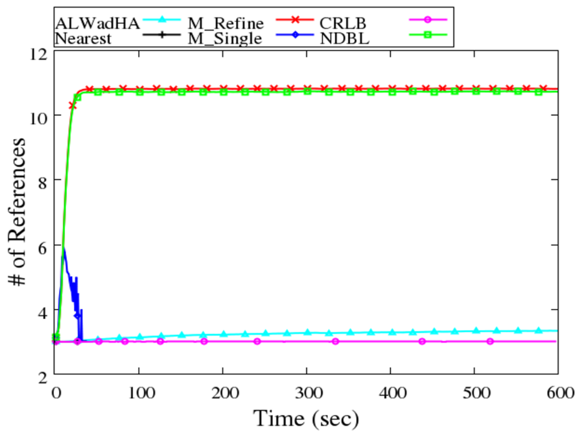


b. Average # of references

Figure 3.23. Small network

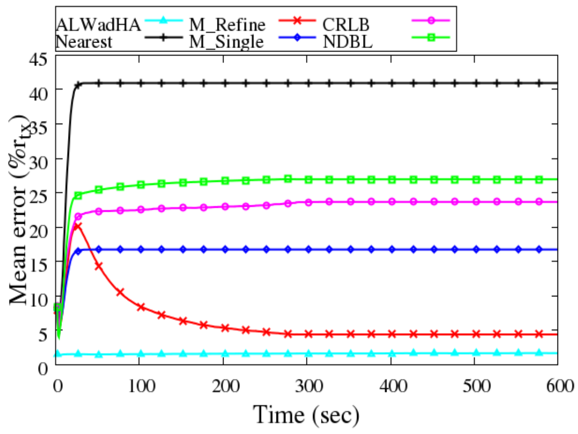


a. Mean error as a ratio of transmission range

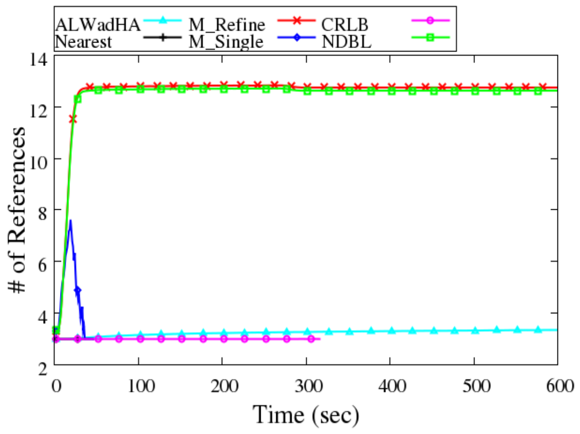


b. Average # of references

Figure 3.24. Normal network



a. Mean error as a ratio of transmission range

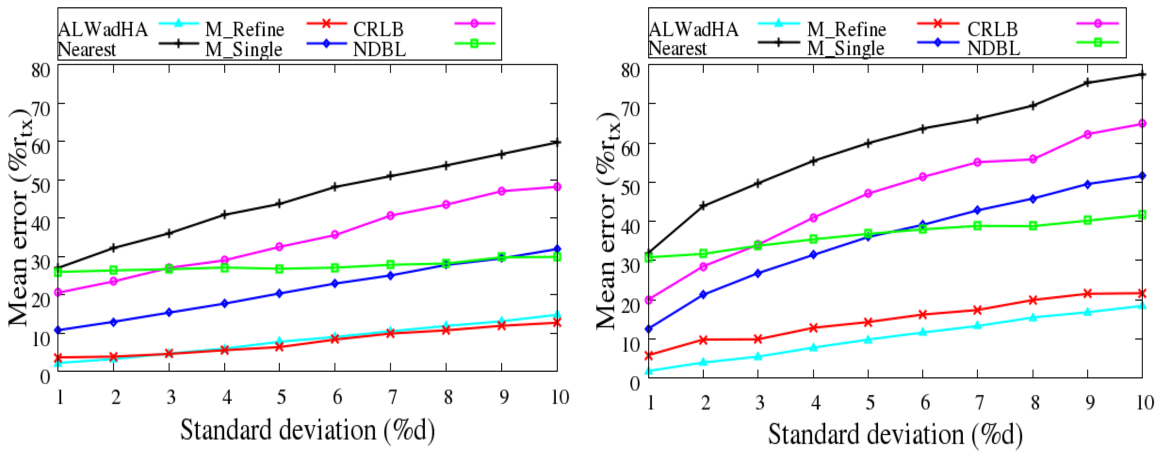


b. Average # of references

Figure 3.25. Large network

3.3.4.4 Distance-measurement error

The first experiment (Figure 3.14) was performed without adding any error to the measured distance, while in the rest of the experiments an error was added with a standard deviation equal to 1% of the measured distance. In this section, the value of the standard deviation is changed from 1% to 10% of the measured distance, in order to check the robustness of the localisation algorithms with the existence of error. Two networks were simulated: grid and random deployment in a field of $200\,m \times 200\,m$ with the same setup characteristics mentioned earlier. For each network, 10 experiments were performed with different standard deviation, then the mean error was recorded at the end of the run time. Figure 3.26 shows the mean error vs the standard deviation in the grid and random deployment. All the localisation algorithms performed better in grid deployment, as explained earlier. The mean error of the single-estimation algorithms (i.e. CRLB, M_Single and Nearest algorithms) increased dramatically with the increased distance-measurement error, while the successive-refinement algorithms (ALWadHA, M_Refine and NDBL algorithms) were more robust than the single-estimation algorithms. This result shows the main advantage of using the successive refinement approach in localisation algorithms.

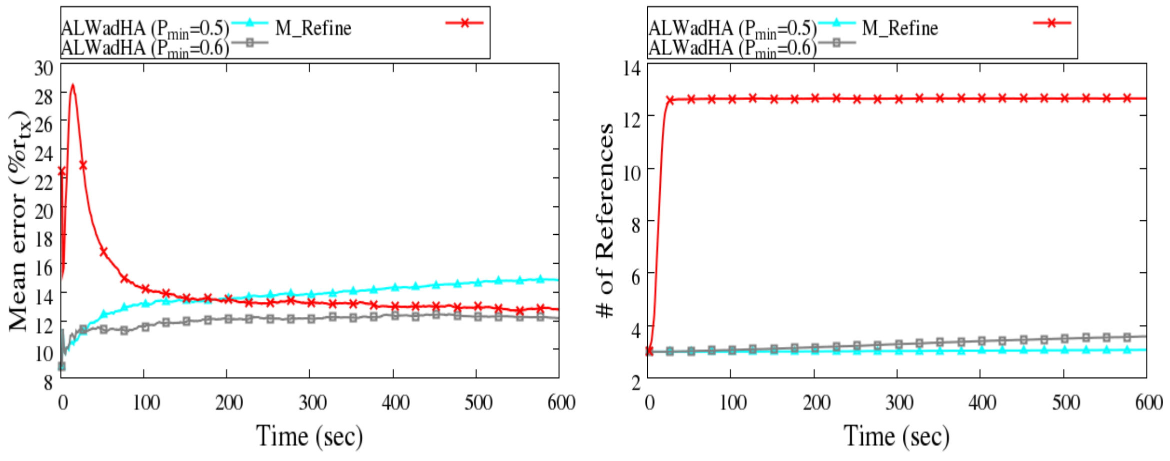


a. Grid deployment b. Random deployment

Figure 3.26. Mean error vs standard deviation

As shown in Figure 3.26, in the random deployment the ALWadHA algorithm has the lowest mean error. However, in grid deployment the M_Refine algorithm started to outperform the ALWadHA algorithm slightly when the distance-measurement error was increased. In the previous experiments, the ALWadHA algorithm was run with P_{min} equal to

0.5. To show the impact of changing the value of P_{min} and to show how the ALWadHA algorithm could outperform the M_Refine algorithm, an experiment was performed using the same grid network used in Figure 3.26.a with a standard deviation equal to 10% of the measured distance. Figure 3.27 shows that M_Refine uses an average number of references equal to 12.46 to outperform the ALWadHA algorithm with $P_{min} = 0.5$. Rerunning ALWadHA with $P_{min} = 0.6$ makes it outperform the M_Refine algorithm. Increasing the value of P_{min} allows the nodes to select a slightly higher number of references, as shown in Figure 3.27.b, which enhances the accuracy of the ALWadHA algorithm.



a. Mean error as a ratio of transmission range b. Average # of references

Figure 3.27. The impact of changing P_{min}

3.4 CHAPTER CONCLUSIONS

This chapter proved that using a proper selection method that is able to select the best set of references could relieve the burden of using all of the available references to enhance the accuracy of location estimation. Several experiments were performed by changing node deployment, node density, network size and distance-measurement error. The results of these experiments show that the ALWadHA algorithm has excellent accuracy compared with other localisation algorithms, in spite of its using a low number of references -- almost equal to the minimum possible number. This reduction in the number of references greatly improves the computation, communication, energy consumption, security, robustness and accuracy of a localisation algorithm.

The results of the experiments done in this chapter showed that the ALWadHA algorithm satisfies the following design objectives: accuracy, self-organising properties, simplicity and robustness. The ALWadHA algorithm achieved excellent estimation accuracy under different scenarios. The effective performance of ALWadHA in the two types of deployment (grid and random deployment) proved the self-organising properties of this algorithm, which also does not require any specific beacon placement. The simple computations required by the ALWadHA algorithm and the low number of references used can be considered as good evidence of the simplicity of this algorithm. Simulation results also show the robustness of ALWadHA, even when the distance-measurement error is increased. The next two chapters will discuss more evidence confirming the achievement of these design objectives and others.