Title	Collision Avoiding Motion Planning of Autonomous Robots with MANETs
Author(s)	岡田，崇
Citation	
Issue Date	2007-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/3616
Rights	
Description	Supervisor:丹　康雄，情報科学研究科，修士

Collision Avoiding Motion Planning of Autonomous Robots with MANETs

By Takashi Okada

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Yasuo Tan

March, 2007

Collision Avoiding Motion Planning of Autonomous Robots with MANETs

By Takashi Okada (510021)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Yasuo Tan

and approved by
Associate Professor Yasuo Tan
Professor Yoichi Shinoda
Associate Professor Mikifumi Shikida

February, 2007 (Submitted)

Abstract

In the research topic of large scale autonomous networked mobile robots, the experiment is difficult in various aspects. On the purpose of constructing above system, at first this research mentions the experiment platform which supports above condistions. Next it mentions the mothion planning algorithm which suits these case.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In disaster area or office building, autonomous robots act instead of human beings. Rescue robots are able to accomplish many tasks in dangerous places where humans cannot enter, such as sites where harmful gases or high temperature are present, the hard environment for human. Cleaning robots can also work automatically and save costs by performing various routine tasks. In all these examples robots have to move to their destination in order to perform their function. For this purpose they need to be able to recognize the changes of environment around them and equip a motion planning method in order to avoid obstacles which have probability making collision with them.

In this research topic, the experiments for evaluating such researches are difficult, since the cost of these real autonomous robots is high. This is particularly true if researchers want to experiment more than a few robots, and need to test systems with tens or even hundreds of robots. Then many researchers try to make experiments or evaluate their algorithms or methods on software simulators. But the results from these simulators are not really precise. The difference between real system and simulators is much big. In real environment, many kinds of noise affect conditions. For example, the connection on real environment is affected by electric waves and magnetic field and so on. Researchers can not always obtain good results on real system, even if they have once obtained good results from experiments on simulators. Then the system needs some solutions on purpose of covering this difference.

In this paper, I propose an experiment platform which large number of autonomous mobile networked robots are executed. And to fulfill it, I propose an new algorithm of motion-planning as an application.

Chapter 2

Related Works

In this chapter, at first one of popular motion-planning method is introduced. Next is the experiment platform which enables researchers to perform and evaluate their researches.

2.1 Motion Planning

In the situation that multiple robots act in same environment, they need navigation in order to move to their destination on purpose of accomplishing their tasks and avoid possible collisions with other robots or obstacles. Multiple robot motion planning are usually classified as *centralized* or *decentralized* method. *Centralized* planners construct plans by one robot and this robot distributes the plan. In *Decentralized* method, each robot plans independently.

One of traditional way is sampling based motion-planning [2]. In this approach, samples are organized into regular grids or hierarchical ones. These grids express the location of the free space. And the planner remember these locations where robots already visited. But the size of these grids increases exponentially according to the dimension of the configuration space, for example the number of degrees of freedom of the robot. Then the calculating the free configuration space takes high cost, if the dimension of the configuration space is more than four or five.

A probabilistic roadmap (PRM) planner has become popular method in these planning case because of the speed of calculation. Many of PRM planner of multiple robot are *decentralized*. Basic PRM executes the following steps :

PRM planner method

Roadmap Generation

At first the planner selects some milestones from configuration space at random. And next the planner tries to joint each neighborhood if the path between each neighborhood is collision-free.

Roadmap Extension

The planner tries to joint each roadmap created. If the roadmaps could not joint with them, it tries random-walk between each roadmaps which could not joint.

Route Search

If the planner can joint the constructed roadmap with the source and the destination, then searches a shortest path using Dijkstra's algorithm.

Figure 2.1 describes one simple example of constructing a roadmap. As described in the figure, S is a source of a robot and D is a destination, triangles of dark grey color are shape of obstacles and light colors means the space where a robot makes collision if it enters the range. Then at first of path-planning the entire map is divided into trapeziums. This rule of divisions draws perpendicular lines from vertices of all obstacles. Next it puts milestones on the center of these perpendicular lines and on the center of the trapeziums, and connects these milestones. Finally S and D connects the milestones in same trapeziums and a path is found.

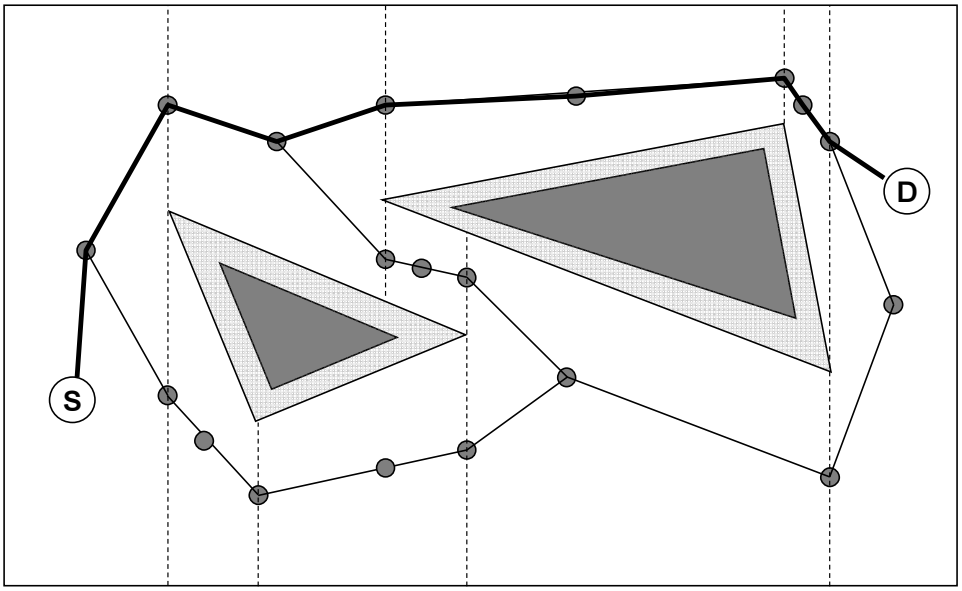

Figure 2.1: Find a path from Roadmap

A PRM planner randomly samples the configuration space of robots and register the collision free samples as milestones. And the planner tries to connect pairs of these milestones and saves this collision free connections as the trajectories of robots. Probabilistic

roadmap means this graph, the edges are the trajectories of robots, the nodes are the milestones, and the undirected graph jointing the trajectories is the probabilistic roadmap. And the planner finds the optimal path from the graph, for example it may set some weights on these edge and use Dijkstra's algorithm.

PRM planners are not complete in the traditional sense. But they are *probabilistically complete* under certain assumptions. It means that the probability of failure decreases exponentially to zero with iterations [3].

Now I mention two types of PRM, one is single-query planners. It computes a new roadmap from scratch for each new query [3]. Second one is multi-query. It precomputes the roadmap and re-use the roadmap for answering queries [9]. It has been proven that, under reasonable assumptions about the geometry of the robot's configuration space, a relatively small number of milestones picked uniformly at random are sufficient to capture the connectivity of the configuration space with high probability.

In addition to PRM method, robots need technique which avoid collisions of each robot. Simply if robots exchange their trajectories and start to plan, this method does not care about the trajectories of each other. "*Velocity turning*" [13] method computes the relative velocities of the robots to avoid inter-robot collision. Another way to solve this problem is "*Prioritized Planning*" [14]. The robots avoid possible collisions depending on their priority.

2.2 Robot Implementation

When researchers are implementing or evaluating something related to autonomous robots, for example path planning algorithm or dynamic network construction and so on, one of the choices is simulation, by using a product such as Webots [10], co-developed by the Swiss Federal Institute of Technology in Lausanne, Switzerland. Software simulator models robot components, networks and motions and time scheduling and surrounding environment. If researchers select to use such simulators, they have to implement simulator-specific modules by themselves. Surely these software simulators can test various kinds of environment that it is hard to implement in a real system. But the results from these software simulators may differ with respect to those that would be observed in a real system. It means that the difference between the results from software simulator and results of a real system is maybe much larger.

The second one is to implement on real environment. RoboCup [5] involves teams of five small robots, each up to 18cm in diameter and 15cm in height. The robot teams are entered into a competition to play soccer against opponent teams fielded by other research groups. Hsu [7] made experiments on his research team testbed. This robots move frictionlessly on an air bearing on a 3 m x 4 m table. ARL (Stanford Aerospace Robotics Laboratory) citeStanford makes various experiments of real robots. These real environment systems take much cost for real robots and sensing. If researchers want to test large number of robots, it is hard to make the ready all robot hardware equipping an ability to compute and connect wireless network.

Third one is emulation. The difference between simulation and emulation is like the difference between modeling and emulation. A simulation is a modeling system out of approximations or inferences. An emulation is emulating or imitating a different environment of hardware or software. This method can not only get reliable experiment results but also not take much cost. If researchers can emulate robots and other environment on computers, the const is only computers. And this approach covers the gap between real implementation and software simulation. StarBED [8] is a large scale, realistic and real time network testbed, using hundreds of PCs, and switched networks. StarBED2 [18] expands StarBED so as to be suitable for emulating ubiquitous networks.

Chapter 3

Motion Planning

As mentioned already, one of the essential feature of autonomous mobile robots is a motion-planning algorithm. Autonomous robots need some path-finding algorithm to avoid probability which cause collisions with robots or obstacles.

Many path-planning algorithms have been proposed to date. The performance of motion-planning algorithm can be characterized by the following properties: speed, completeness, and optimality. In dynamic and unknown environment, robots must plan and re-plan their motion many times, because the environment dynamically changes in time. Therefore in the case when robots need to continuously plan on-the-fly their trajectory, the algorithm speed is one of the most important properties.

3.1 Path Planning in Expansive Configuration Spaces

In proposed system, robots equip a kind of PRM method that base on "Path Planning in Expansive Configuration Spaces" [3]. The definitions of this algorithm are the following. For example, a configuration can be specified by d parameters $q = (q_0, q_1, \cdots, q_{d-1})$, where d is the number of defined degrees of freedom of a robot. The set of all configurations forms the robot's configuration space C. A configuration q is free if the robot placed at q does not collide with obstacles. The basic idea of this paper is using single-query path planning methods, there are given two configurations q_{source} and $q_{destination}$. They sample at random from configuration space C and register only q that have collision-free path from q_{source} or $q_{destination}$. Then they grow two trees from q_{source} or $q_{destination}$. These two trees keep on building till accomplishing the connection both q of these two trees with collision-free.

This algorithm iteratively executes two basic steps, *expansion* and connection until either a path is found or the maximum number of iterations is counted.

3.1.1 Expansion

Expansion is the first step of this planner, by which the planning method build two trees $T_{source} = (V_{source}, E_{source})$ and $T_{destination} = (V_{destination}, E_{destination})$.

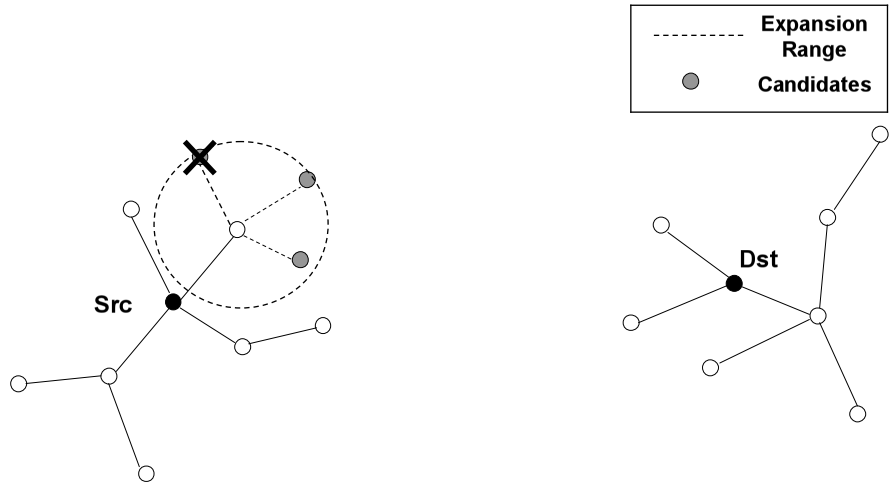

Figure 3.1: Expansion

Figure 3.1 describes this appearance. These two operations of trees are identical, and extend tree $T = (V, E)$ starting from the robot initial position (source) and the robot destination, respectively. When building each of these trees, the planner picks up a node x from existing milestones , which is chosen with a probability proportional to $1/w(x)$. The value $1/w(x)$ represents the weight of node x , and it is equal to the number of neighbors of x plus 1 (the node x itself). The larger the number of neighborhood of the node x is, the more the node x is selected. After the planner selected a node x, it tries to pick some candidates of milestones to expand the tree. And the planner adds them to the tree, if those candidates are effectively reachable from x.

3.1.2 Connection

Connection is the second step of the algorithm, in which this planner tries to connect the two previously built trees, T_{source} and T_{dst}.

Figure 3.2 describes the situation which two trees connect to each other. This connection method is simple. For every nodes in the set of vertices of the corresponding trees , V_{source} and $V_{destination}$, for example x is picked from V_{source} and y is picked from $V_{destination}$, the planner checks whether these vertices can see each other or not and check whether the distance that it is smaller than $distance(x, y)$ or not, and if the planner find two milestones which satisfy the above conditions from every tree and the edge is collision free, the planner terminates successfully.

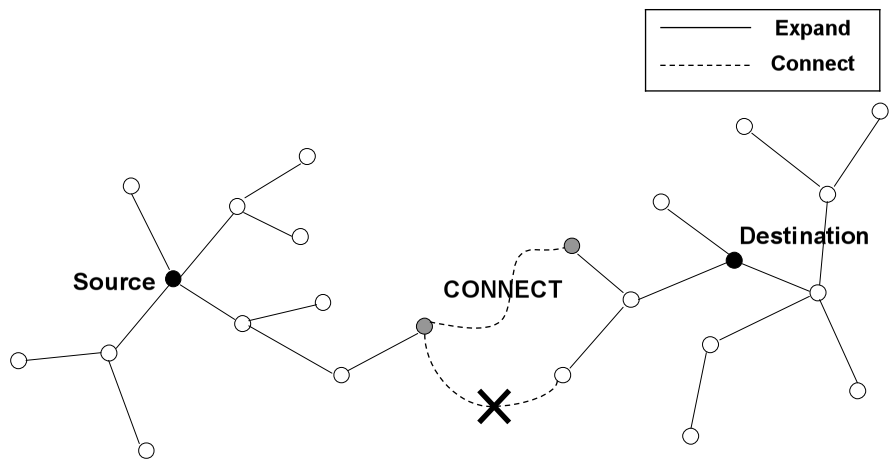

Figure 3.2: Connection

3.2 Proposed approach

In the case study of this paper, robots are considered to be placed in environment that are unknown or change in a dynamic manner. The motion-planning method described above does not suite such a case. For example, the previous planner can not predict whether there are any collisions or not in the tree built from destination, $T_{destination}$. It means that the planner has no way to know the time when the robot will reach the milestones in the tree, $T_{destination}$. Hence the planner cannot expand the tree, $T_{destination}$, in these cases.

In order to adapt the planner to these conditions of unknown and dynamic environment, the notion of the time is important in these cases. Because even if the planner found some path to destination, the path might be inefficient with much calculating time. In dynamic environment, robots have to plan their path in real time, it is important that the time keeps on running during planning new path. And I adapted the motion-planning method discussed in the following sections.

3.2.1 Definitions

At first the parameters of this motion-planning method should be defined. The definitions of q and C are similar to Section 3.1. The parameter q is the element of degrees which robots can move free. C is the configuration space, means all the set of this q. A configurations q is *free* if the robot placed at q does not collide with obstacles. The set of all free configurations are defined the *free space F*.

3.2.2 New Expansion

As mentioned above, in unknown and dynamic environment, introduced path-planning [3] does not work well. Then my algorithm uses following method :

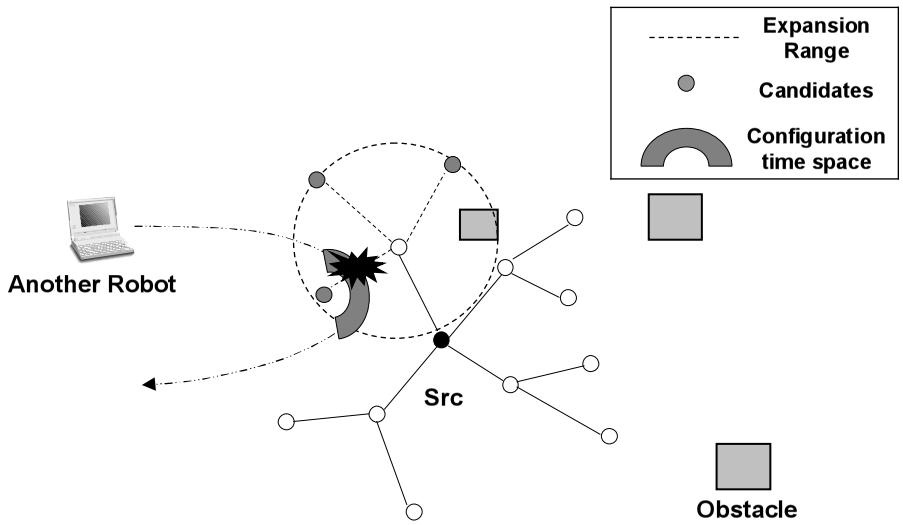

Figure 3.3: New Expansion

Grows only source tree

This means that the planner grows only one tree (T_{source}), this is why the planner cannot check the collisions of $T_{destination}$. This method is discussed in [3], but they recommend this way only when the robot is highly constrained around q_{source} or $q_{destination}$, then avoid to grow the tree from this constrained q.

Adapt time parameter

The milestones have the **time** parameter when the robots reach to the milestone. If milestones do not include this value, the planner cannot predict the collision with other autonomous mobile networked robots or obstacles. This is fulfilled by *Configuration time space*. It is a kind of configuration space but including the time parameter. Figure 3.3 describes this. This configuration space covers the other moving robots as moving obstacles, and the robots avoid to pick this configuration.

Then I adapt above conditions to "*Expansion*" method.

Algorithm *Expansion*

1. Pick a node x from V with probability $1/w(x)$.
2. Sample K points from $N_{Dd}(x, time) = \{q \in C(time) | d_{minimum} < dist_c(q, x) < D_{maximum}\}$, where $dist_c$ is some distance metric of $C(time)$. (K and $d_{minimum}$ $D_{maximum}$ are parameters.

3. **for** each configuration y that has been picked **do**

4. calculate $w(y)$ and register y with probability $1/w(y)$

5. **if** y is registered, $clearance(y) > 0$ and $link(x, y)$ return YES

6. **then** put y in V and place an edge between x and y.

In step 1, the meaning of probability $1/w(x)$ is same as based method.

In step 2, K is the number which the planner tries to pick the milestones from above conditions. $d_{minimum}$ and $D_{maximum}$ are the minimum and maximum distance which the planner can set the candidates of milestones. Then by $N_{Nd}(x)$, the planner picks the milestones which are plotted in the above range and are collision-free in predicted current time. $C(time)$ is "*Configuration time space*", it is the configuration space in time.

In step 5, $clearance(y)$ is the minimum distance, which is defined as its minimal distance to the boundary of F. And $link(x, y)$ is the checking whether x can "see" y without any obstacles on its sight.

In this *Expansion* phase, the constant numbers, $K, d_{minimum}, D_{maximum}$ affect the ability of this planner. If the planner increases K, the optimality also increases but the calculation costs increase too. No less than the number K, if it increase the range between $d_{minimum}$ and $D_{maximum}$, same situation causes. Then the definition of K and $d_{minimum}$ and $D_{maximum}$ is much important for this motion-planning.

3.2.3 New Connection

As expressed in *Expansion* method, the planner tries to connect only newly added milestones last step.

Algorithm *Connection1*

1. **for** every newly added $x \in V_{source}$ **do**

2. **if** $dst_w(x, q_{destination}) < l$ (l is a parameter.)

3. **then** $link(x, q_{destination})$.

In step 1, the newly added x are the milestones which the planner added last *Expansion* step. In step 2, l is some constant distance. This l also influence the planning time and the ability of this planner. If $link(x, q_{destination}$ returns YES for some x, then a path is found between q_{source} and $q_{destination}$ through x.

But there is a possibility which the algorithm run faster. Basically the robots can not know the entire map at once, but the robot use PRM planning for avoiding collisions in some complicated area. Then the idea is going straight trajectory after passing through this complicated area. This is simply realized by taking away the step2 above algorithm

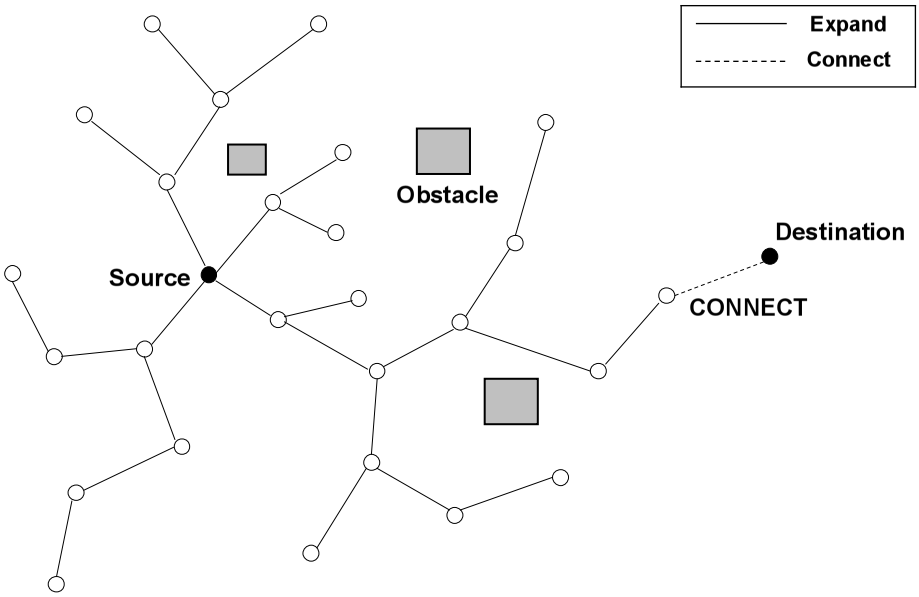

Figure 3.4: New Connection

as follows :

Algorithm *Connection2*

1. **for** every newly added $x \in V_{source}$ **do**
2. $\quad link(x, q_{destination})$.

3.2.4 Prioritized Planning

In addition to these two methods, the system needs to correspond to large-scale multiple robots. Every robot has to communicate and avoid possible collisions with each other if robots detect the collision in their current trajectory. Then these robots avoid the collision according to the priority, for example there are two robots A and B, and assume the priority of A is higher than the priority of B. Now their trajectories make collision if they keep on moving their trajectories. Then robot A keep on its trajectory and robot B needs to find a trajectory which avoid possible collision.

This prioritized planning means that the lower priority robot regards the higher priority robot as a moving obstacle. Then the configuration space of this lower priority robot is limited. This limited configuration space is "*Configuration time space*", and all of robots have to consider their trajectory on attention of this limitation. This means that the lower the priority of robot is, the more the configuration space of the robot decreases.

3.2.5 Algorithm

By using above *Expansion* and *Connection* algorithm and *Prioritized Planning*, I adapt PRM planner to multiple robot in real time on unknown and dynamic environment as follows :

Algorithm1

1. **If** $time < t$ (t is a parameter.)
2. **then** *Expansion*
3. **If** *Connection1* returns YES
4. **then** register new found path
5. Compare all found paths and return fastest path

Algorithm2

1. **If** $time < t$ (t is a parameter.)
2. **then** *Expansion*
3. **If** *Connection2* returns YES
4. **then** register new found path
5. Compare all found paths and return fastest path

In step 1, *time* is current time, and t is a parameter when limit the steps which the planner executes.

In step 4, if the planner found a path between q_{source} and $q_{destination}$.

In step 5, the planner returns the path which the time when a robot reaches to $q_{destination}$ is earliest from all found paths.

The difference between *Algorithm 1* and *Algorithm 2* is only selecting which *Connection* method.

In this chapter, at first I introduce the based motion-planning algorithm and next, represent the problems of this algorithm in unknown and dynamic environment. After that, I propose a new algorithm which suite in these cases.

Chapter 4

Experiment Platform

Researchers need to confirm and evaluate the effects of their proposed research. When they implement large-scale autonomous mobile networked robots, this is very difficult phase by many reasons as follows.

If they try to evaluate their contents of researches on software simulator, there may appear big gaps when they implement on real system in next step. If they try to create real mobile multiple robots system, the costs of the various hardware are much expensive and they may encounter unexpected problems to control them.

Then emulation is one of the effective and appropriate method to implement these system. In this chapter, I propose a new experiment platform to emulate large-scale autonomous mobile networked robots.

4.1 Overall Architecture

The robots in the proposed system cooperate in order to reach a destination while avoiding collisions and accomplish some given tasks. To express these large-scale robots, a suitable network testbed is needed. For this purpose I use StarBED [8]. StarBED is large-scale network testbed which has a large number of PCs (more than 700) and network interfaces. These all experiment PCs are equipped with at least two network cards (100 Mbps or 1Gbps type). These PC can easily construct large-scale networks by changing switches. All of robots are emulated on StarBED PCs and communicate through Ethernet configurations. Figure 4.1 describes the entire architecture of the system.

To fulfill executing emulated robots on StarBED, following contraptions are needed :

Hardware emulation

For the system assumes real PCs to be autonomous mobile networked robots, the modeling and emulations are needed. This is given in Section 4.2

Network emulation

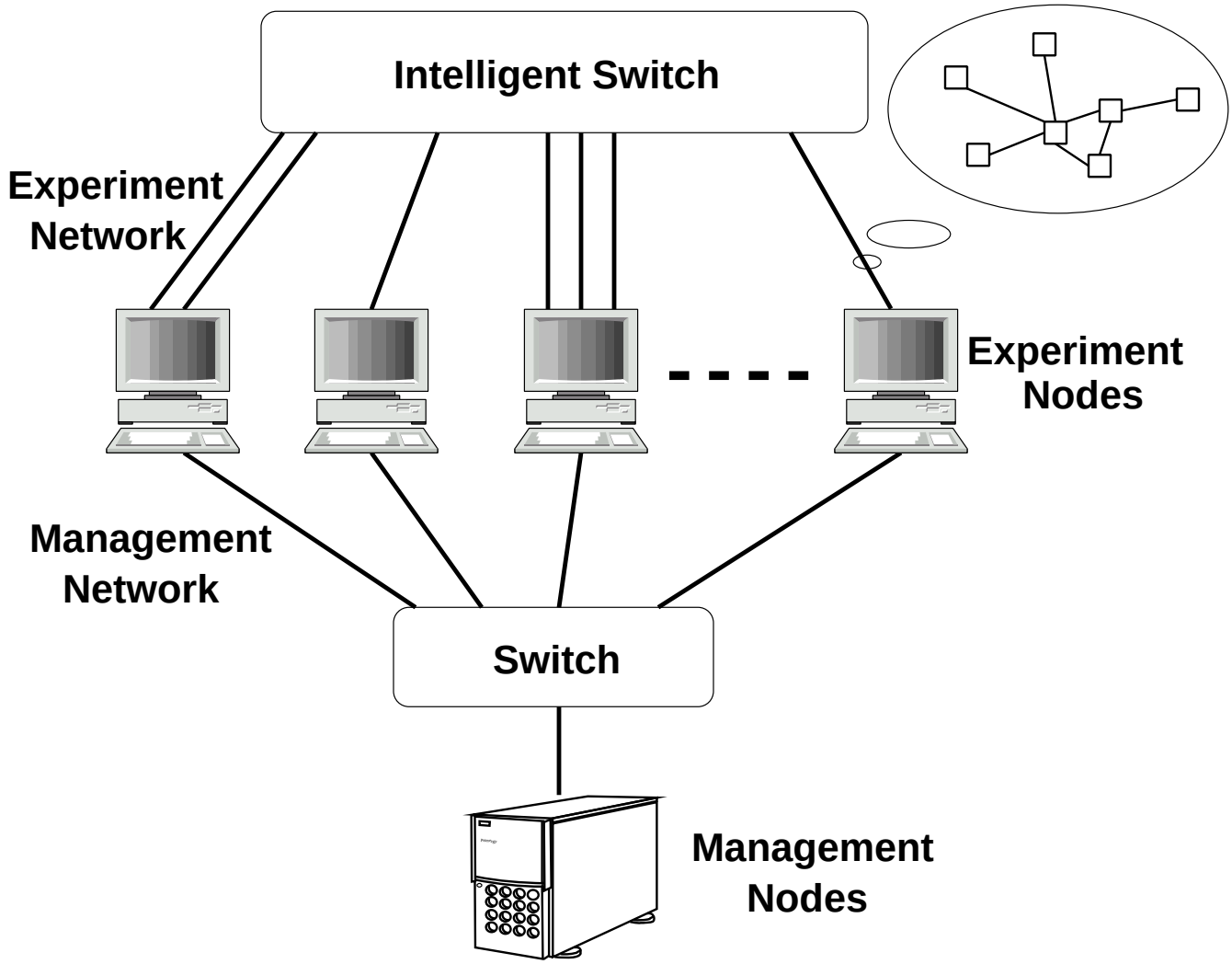

Figure 4.1: Emulation on StarBED

The networks of StarBED are Ethernet or 802.3 networks, these this networks also need some method to emulate WLAN communication. This is given in Section 4.3 and 4.4.

Expand into large-scale

This system executes large-scale networked robots, and to fulfill this, it needs some experiment integration method. This is given Section 4.5.

Robot application

Finally of course these robots execute applications to communicate to each other. In this system, robots equip motion-planning (proposed in Section 3.2) application.

4.2 Hardware Emulation

For emulating robots on StarBED PCs, the system needs some modeling to represent the hardware of robots, for example motors or some sensors and so on. In unknown and dynamic environment the system needs to know dynamic changes of environment and some structure which synchronize the events which happens on every robots in same time. In this system, *Map Manager* administrates all these hardware information. Figure 4.2 describes the general overview of controlling hardware messages in this system.

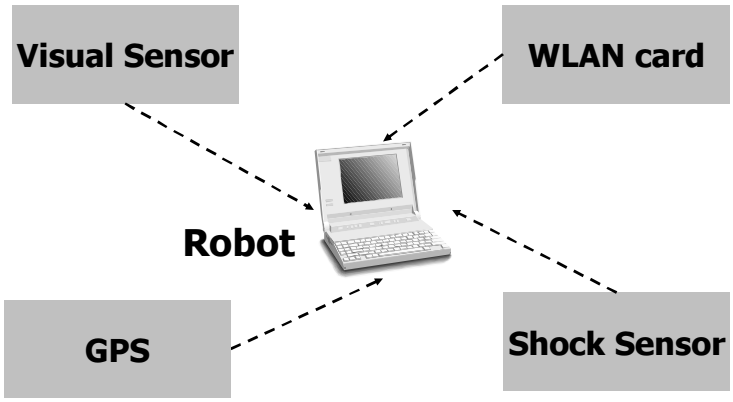

Figure 4.2: Hardware Emulation Architecture

Each robots is able to know what happens on their hardware from the emulated hardware information, for example images from "*Visual sensor*" or detection of WLAN radio signal or some alarm messages from "*Shock sensor*" and so on. These hardware information is treated by the application of robots from the messages of "*Map Manager*", it means that this system consider these emulated hardware information to be the messages from device driver. Figure 4.3 describes the architecture of "*Map Manager*".

4.3 Network Emulation

The mobile networked robots communicate on wireless network, then the system has to emulate this network system. Figure 4.4 shows the overall network architecture of this system. These two networks are separated by switching and creating VLAN, and there are two types of streams on each different networks. On "*Management Network*",

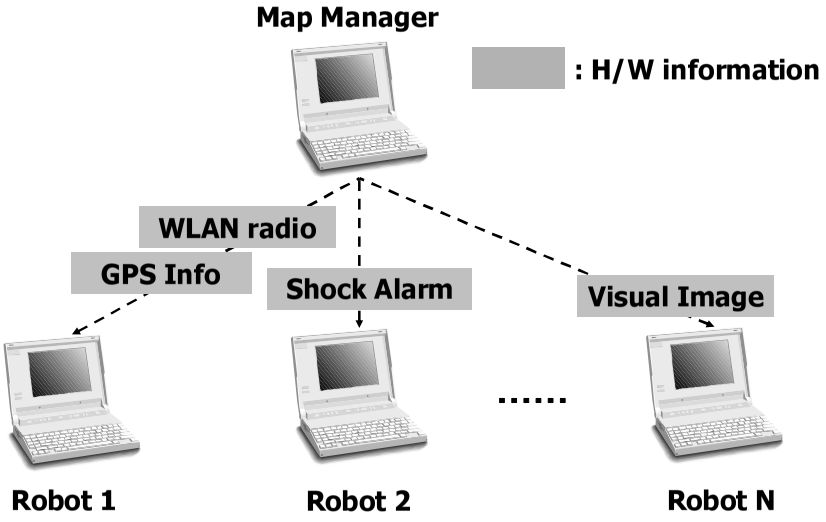

Figure 4.3: Map Manager Architecture

the messages of emulated hardware information which are shown in the previous section are sent. This network is Ethernet network without any limitations, and these emulated hardware messages are immediately received by emulated robot PCs and "*Map Manager*". "*Experiment Network*" is the network which is assumed as real WLAN network at disaster area or office building or home. The robots communicate and cooperate by sending the motion-planning messages through *Experiment Network*.

To fulfill the WLAN communication, the system apply some emulated WLAN configuration values (for example, the bandwidth or the delay or the jitter and so on) to Ethernet cable network. This emulation can be realized by WLAN emulator : QOMET [15] (see Section 4.4 more details). The WLAN communication emulation engine QOMET is deployed in the emulated robots to allow recreating network conditions similar to those occurring in a real WLAN environment. In order to adapt these WLAN configuration value to Ethernet cable network, *dummynet* [16] is effective in every constant steps. *Dummynet* supplies various conditions of network through IP queues. Emulated robots manages these queues for every robots other than itself. And these queues limit the communication according to WLAN configuration values, as it will be detailed in next section.

In order to implement such complex experiment system, the system includes an experiment-support software, called *RUNE* (Real-time Ubiquitous Network Emulation environment) [**?**]. It provides additional functionality which supports large-scale emulation system. Figure 4.6 shows the integration with *RUNE* of this system. The *Rune Master* manages the entire system (see Section 4.5 for more details).

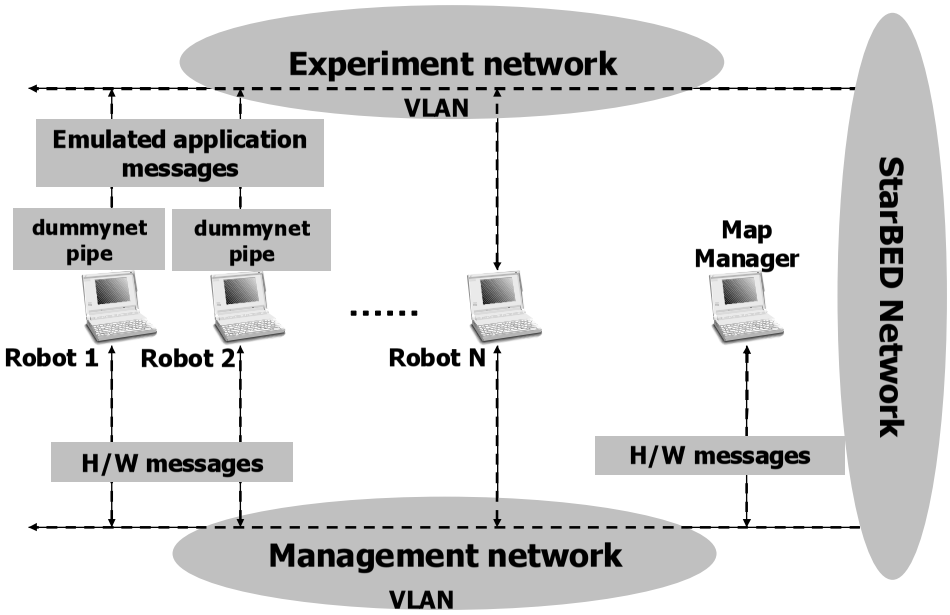

Figure 4.4: General System Overview

4.4 WLAN Emulation : QOMET

The scenario-driven architecture for WLAN emulation has two stages. In the first stage, from a real-world scenario representation QOMET create a network quality degradation (ΔQ) description which corresponds to the real-world events (see Figure 4.5).

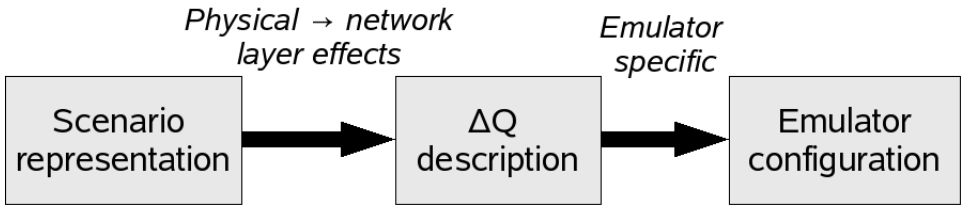

Figure 4.5: Two-state WLAN emulation

By quality degradation QOMET mean the change in network service quality between two measuring points; QOMET denote this degradation by the shorthand ΔQ. Since the ΔQ description represents the varying effects of the network on application traffic, the WLAN emulator's function is to reproduce it. The ΔQ description calculated in the first stage is therefore converted into an emulator configuration that is used during the effective emulation process to replicate the user-defined scenario in a wired network. This makes it possible to study the effects of the scenario on the real application under test. This

WLAN emulation model is an aggregation of several models used at the various steps of the conversion of the scenario representation to the network Δ Q description which is needed to recreate those scenario conditions. The following steps describe the these models at each level of the conversion: real world scenario to physical layer, physical layer to data link layer, and, finally, data link layer to network layer. Modeling stops at network layer because it is at this level that QOMET introduce the quality degradation using a wired network emulator.

4.5 Experiment Integration : RUNE

Rune (Real-time Ubiquitous Network Emulation environment) provides an API set which controls experiment environments. The fundamental function of *Rune* is to implement a test environment in which a number of "spaces" that emulate each experiment target can work on either single or multiple nodes. *Rune* provides a reasonably abstracted interface for easily implementing emulation targets as spaces without much concern about the interaction between emulation nodes. *Rune* has the following roles:

- experiment environment setup/cleanup and progress management;
- procedure invocation;
- interaction between spaces;
- time synchronization;
- mutual exclusion.

Figure 4.6 shows the structure of an experiment implemented using *Rune*. The "*Rune Master*" module manages the configuration of each experiment, and controls the progress of the experiment. The execution of all spaces deployed on multiple nodes is initiated by *Rune* master via modules called *Rune Manager*. The *Rune* manager is deployed on every emulation node and mediates communication between them through objects called "conduits". Spaces implementing emulation targets exist on emulation nodes in the form of shared objects, loaded dynamically by the *Rune* manager.

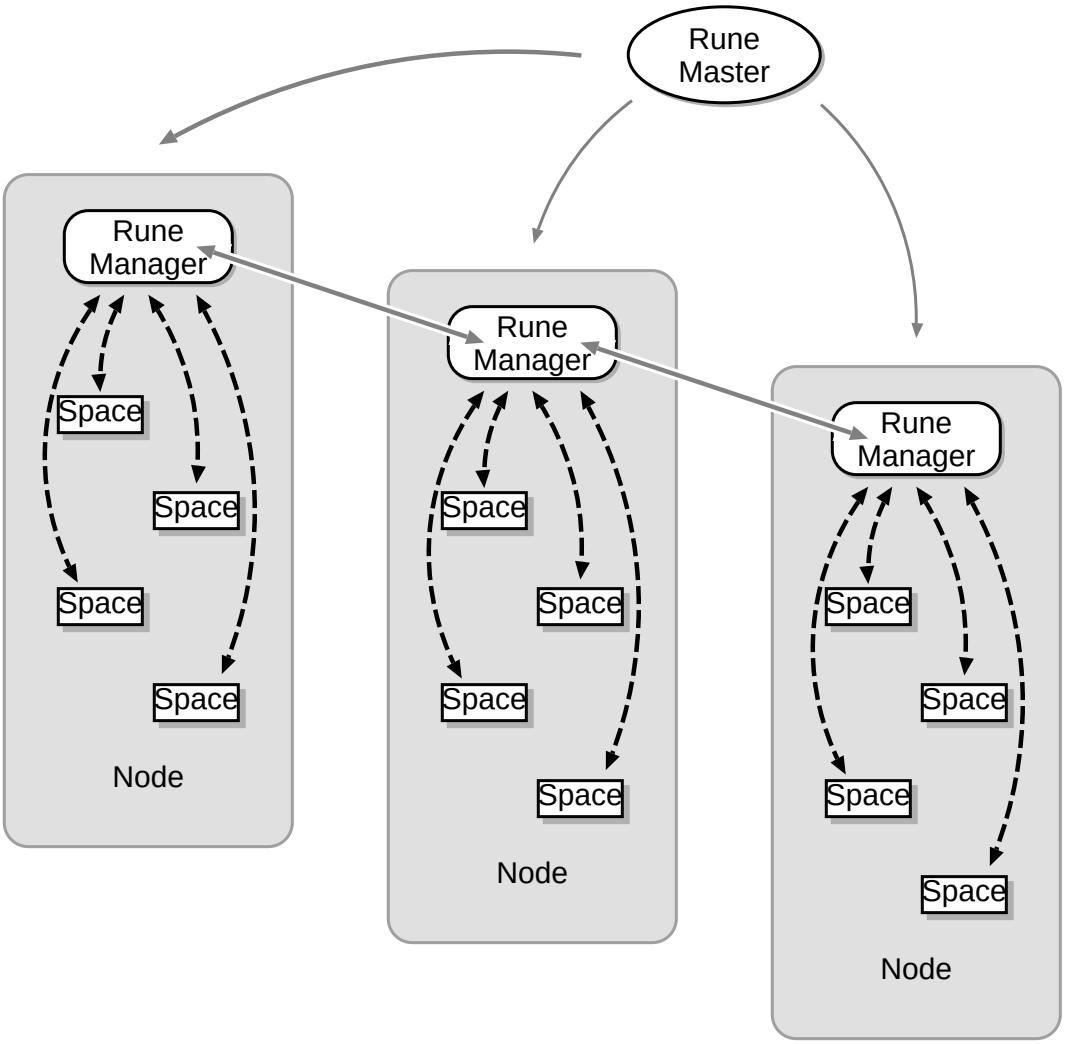

Figure 4.6: Structure of experiments using *RUNE*

Chapter 5

Robot Applications

When researchers implement a system of autonomous mobile networked robots on their own environment, they also have to describe behaviors of robots. The robots in the system equip proposed motion-planning algorithm (see Section 3.2 details). The main objects of this system are "*Map Manager*" and Robots. These objects execute cooperatively and communicate to each other. Then at first next subsection shows the messages which send in this system, and next the flow of each application is given.

5.1 Application Messages

The messages which are managed in this system are three types as follows :

Map Manager Message

These messages are sent from *Map Manager* to Robots through *Management Network*. The details of the contents of this messages are shown Table 5.1.

Robot Management Message

These messages are sent from Robots to *Map Manager* through *Management Network*. The details of the contents of this messages are shown Table 5.1.

Robot Application Message

These messages are sent from Robots to Robots through *Experiment Network*. The details of the contents of this messages are shown Table 5.1.

5.2 Application Flows

Figure 5.1 describes the flowchart common to the applications *Map Manager* and Robots. The transactions of these applications can be divided into following simple transactions.

Flowchart of general applications

type name	sub type	definition
message type	*detect neighborhood*	*WLAN card* detect new radio wave
	disappear neighborhood	disappear radio wave
	detect obstacle	*Visual sensor* detect obstacle
	management	unexpected events
node type	*Robot*	the message about robot
	Obstacle	the message about obstacle
ID	none	the number identifies the object
IP address	none	the IP address of neighborhood is set if message type is *detect neighborhood*
trajectory	none	the trajectory of the objects

Table 5.1: Map Manager Message

type name	definition
ID	the number identifies the robot
trajectory	the trajectory of the objects

Table 5.2: Robot Management Message

Initialize

In this phase, *Map Manager* and Robots initialize their objects and connections through *Management Network* (see Figure 4.4). At that time *Map Manager* send the settings (see Table 5.2 more details) to every robot and thereafter start at one time.

Update

These applications are known the time when the system finishes, and if current time is not time up, executes *Update*. This loop phase is divided by short time steps, and in every *Update* phase they also have similar structure. This transaction is difference between *Map Manager* and Robot.

Finalize

Finally the applications know the time over, they finalize their objects and finish the applications.

Flowchart of Map Manager

Check updates of environment

type name	definition
ID	the number to identify the robots
priority	the priority of the robots
trajectory	the trajectory of the robots

Table 5.3: Robot Application Message

setting name	description
ID	the number to identify every robot
radius	the radius of the robot
priority	the priority used planning
source	the coordinate of source
destination	the coordinate of destination
velocity	the maximum velocity value

Table 5.4: the parameters of setting of robot

Map Manager checks that all of robots detest any hardware messages in current time, for example some robots detect new neighbor robot from the WLAN card or new obstacle from *Visual sensor* and so on.

Send H/W messages

If the robots detect any emulated hardware messages, *Map Manager* informs this.

Get messages

Map Manager waits the messages from robots till finishing current step.

Flowchart of Robots

Check updates of environment

The robots check the changes of their environment from the messages from *Map Manager* and the other robots.

Planning

If the changes of their environment make collision in their trajectory, they start to plan (execute the algorithm Section 3.2). This flow is divided Figure 5.4. The implementation of this system, the robots wait constant time and keep on re-planning if they cannot find a trajectory.

Update myself

The robots update their objects, for example the trajectory (if they planned), the WLAN configuration.

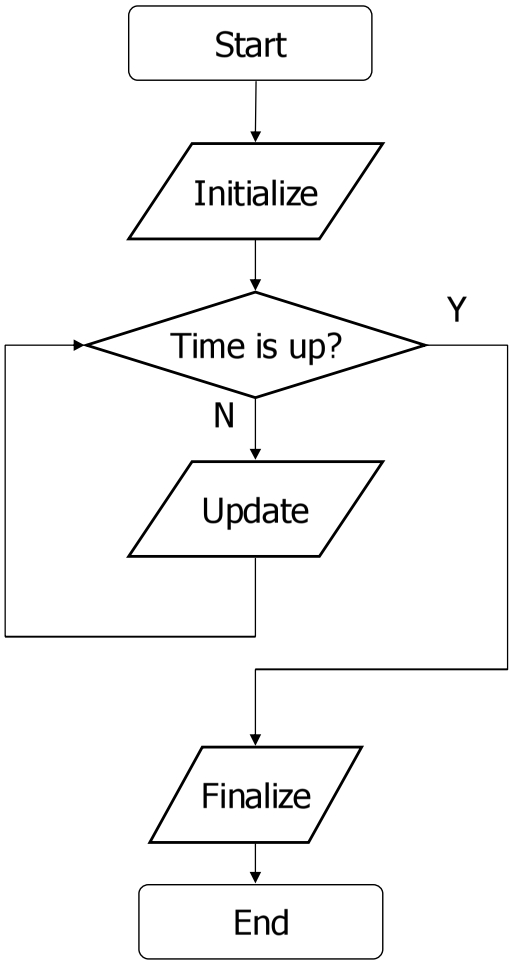

Figure 5.1: Flowchart of General Applications

Get messages

Finally the robots wait the messages from *Map Manager* or the other robots till finishing current step.

In this chapter, the method which emulates autonomous mobile networked robots on PCs is proposed and mention about details of these modeling and emulations.

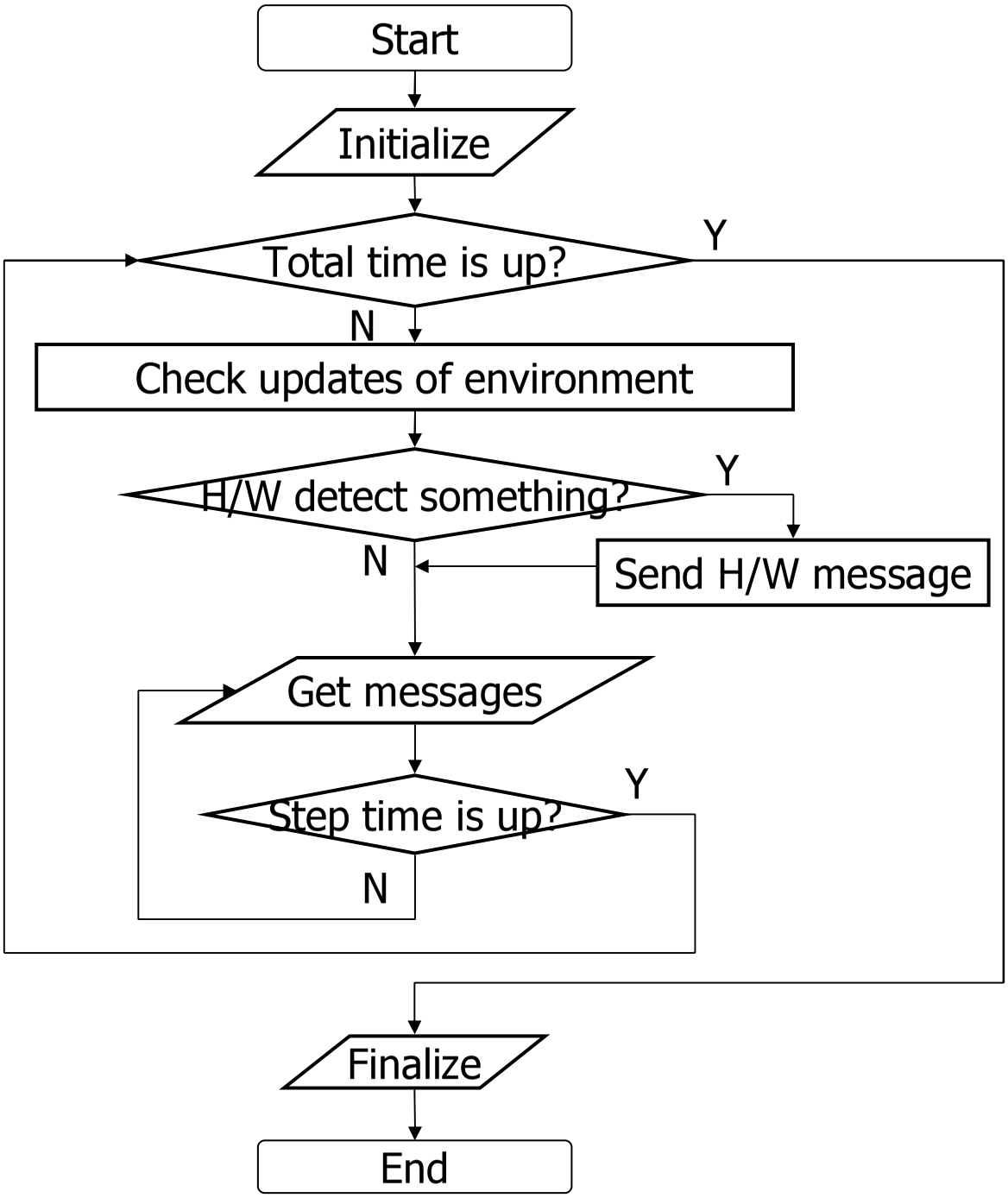

Figure 5.2: Flowchart of *Map Manager*

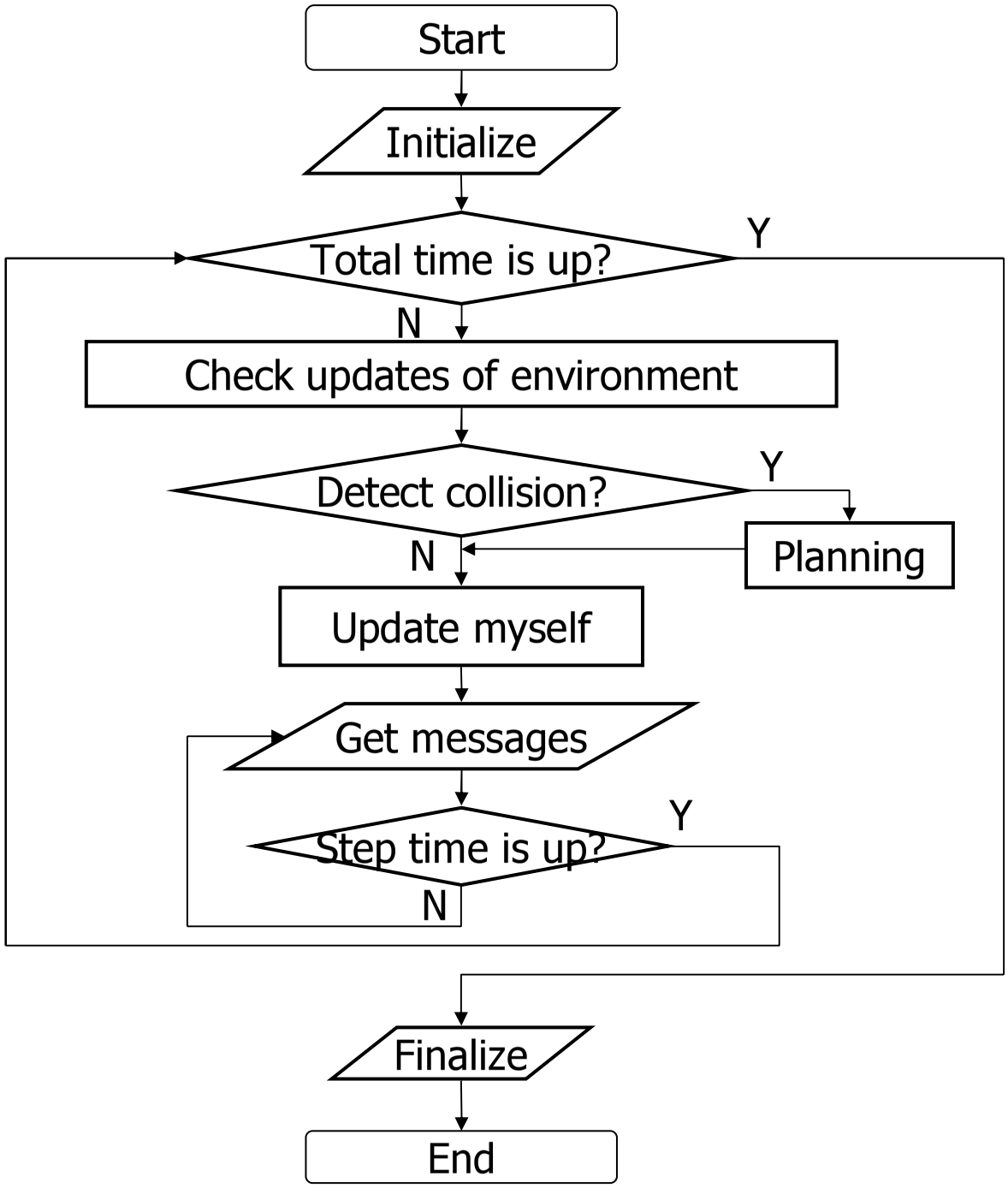

Figure 5.3: Flowchart of Robot

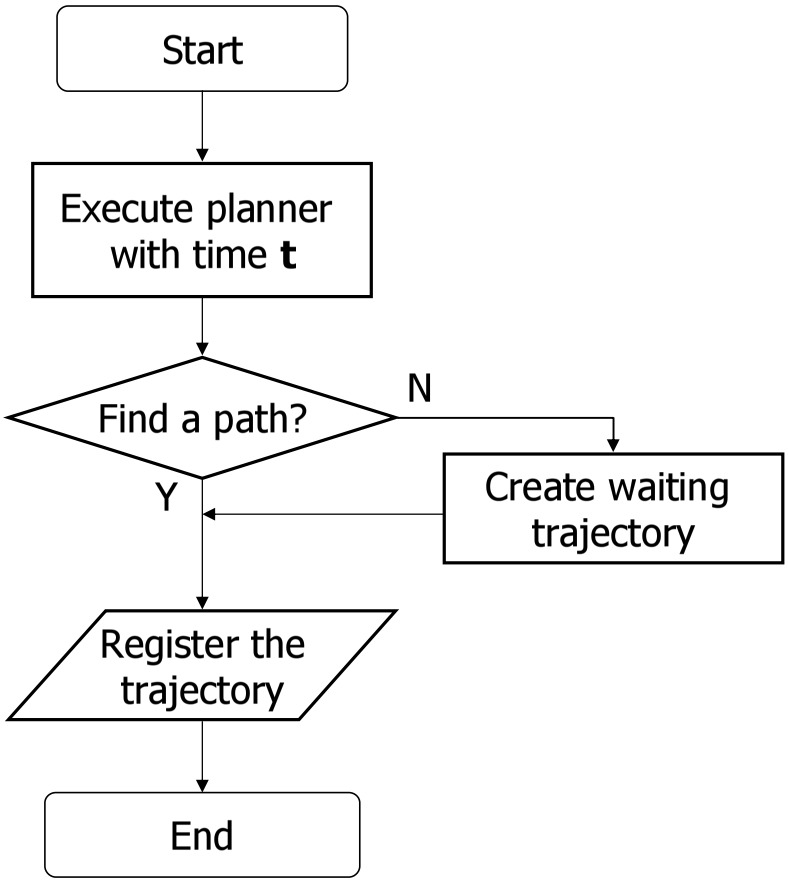

Figure 5.4: Flowchart of Planning

Chapter 6

Experiment

In this chapter, how to experiment this system and the definitions of the system environment and emulated descriptions are mentioned. After that the results from this system are shown every different scenario. The experiment which implements the system includes two meanings. First is to evaluate the proposed algorithm. The system enables to confirm the performance of the algorithm and practicability on real environment. Second are about the system. It make sure whether large-scale mobile networked robots correctly work and communicate each other, the results are compared and evaluated.

6.1 Experiment Definitions

At first the definitions of the experiments every scenarios are defined. The environments are related to the WLAN communication conditions. The definitions of robots set the parameters of the robots. And finally the scenarios define how the robots and obstacles are located and set the destinations.

6.1.1 Emulated Environment Definition

In this paper, the autonomous mobile networked robots are assumed to avoid endangering human life or to reduce cost of repetitive activities. In a disaster or dangerous area, autonomous rescue robots can accomplish various tasks instead of human rescue team. And in office buildings or homes, autonomous robots assist many kinds of human living, for example it can automatically clean the rooms or hallway. The environment usually classify into *indoor* and *outdoor*.

indoor

The WLAN communication conditions of the "*indoor*" environment are hard. For example at a office building or home, there exist desks and walls and various obstacles interfering communications.

outdoor

The "*outdoor*" environment enables to communicate easier than *indoor* environment. But the WLAN communication conditions of *outdoor* environment depend on the affections of "Walls". If there are many obstacles interfering the communication, the conditions are limited.

The conditions of WLAN communication given above are realized by the configuration values from QOMET. The table 6.1.1 describes the parameters of every environment.

environment	α	σ
indoor	5.6	2.0
outdoor	3.32	2.0

Table 6.1: the parameters of QOMET

α is the "path-loss exponent", and σ is used take into account the shadowing component. From the WLAN emulator QOMET, the connection ranges between robots are calculated as follows :

environment	**connection range**
indoor	$\simeq 18.3(m)$
outdoor	$\simeq 100.0(m)$

Table 6.2: the connection ranges from QOMET

In addition to mentioned hereinbefore, the scenarios describes these whole settings, the number of robots and obstacles and the coordinates of robots and obstacles and of course environments.

6.1.2 Emulated Robot Definition

In this system only the hardware of robots is modeled and simulated. As given previous chapter, these autonomous mobile networked robots rustle in disaster or dangerous area like rescue robots and in office building or home like cleaning or assisting robots. These robots equip motors which move around to accomplish their tasks. These motors are assumed which enable them to turn onmidirectional. And they also have various sensors, a GPS makes them know their absolute coordinate in real time, and a "*Visual sensor*" informs a visual map to them. The autonomous robots of this system equips "*Visual sensor*" and detect obstacles around them. The range of this "*Visual sensor*" is limited. The autonomous robots detect the obstacles at first time they are closed to them enough sensing them. These robots communicate each other through WLAN card, this radio wave is sensed by "*Map Manager*" on "*Management Network*" and after that, robots start to communicate emulated WLAN communication on "*Experiment Network*".

parameter	value	meaning
shape	circle	the shape of the robot
radius	$1.0(m)$	the radius length of the robot
velocity	$0.5(m/sec)$	the maximum speed of the robot
visual sensor	$10.0(m)$	the visual sensor range
WLAN sensor	depending on environment	(see previous section)
degree	onmidirectional	the degree the robots can move
time step	$250(msec)$	the step of execution

Table 6.3: the parameters definition of the robot

The table 6.1.2 describes the parameter definitions of the hardware of autonomous robots of this system.

Basically this system allow various shapes of robots, but both robots and obstacles have a circular shape so that their size can be described by the radius. The system assumes the longest arm of robots, which is the radius of the smallest circle which can cover the object (they have the same center point), the radius of the robots.

The maximum velocity of the robots of this system are set $0.5(m/sec)$ which is slower than walking velocity of humans.

The range of the "*Visual sensor*" equipped can see $10.0(m)$, this sensor is assumed onmidirectional sensor, for example the camera which the robots equip turn around onmidirection.

WLAN card senses the WLAN communication radio in the range which depend on the environment and the distance between the robots who communicates (see Table 6.1.1).

The system regards the motors of these robots as onmidirectional motors like caterpillar or warm or spider type.

The time step is the limitation of time when *Map Manager* and Robots execute one main "Update" phase (see Section 5.2).

6.1.3 Scenario Definitions

In this section, the five *Scenario*s which is defined the source and destination of every robots and the coordination of obstacles and its size are described. At first as simple case, two *Scenario* is defined, to confirm the basic flow of this application. Next is more complex *Scenario* which includes ten 10 robots and 6 obstacles in environment. And finally as large-scale *Scenario*s are drawn.

Simple Scenarios

The "Scenario 1" (Table 6.1.3) defines one robot and one obstacle in environment. The initial trajectory of the robot goes through the obstacle.

object	ID	priority	radius	source	destination
robot	rb01	10	$1.0(m)$	$(-20.0, 20.0)$	$(20.0, -20.0)$
obstacle	obs01	20	$1.0(m)$	$(0.0, 0.0)$	none

Table 6.4: Scenario 1 : One robot and one obstacle

The "Scenario 2" (Table 6.1.3) defines two robots in environment. The initial trajectories also cross each other.

object	ID	radius	source	destination
robot	rb01	$1.0(m)$	$(-20.0, 20.0)$	$(20.0, -20.0)$
robot	rb02	$1.0(m)$	$(20.0, -20.0)$	$(-20.0, 20.0)$

Table 6.5: Scenario 2 : Two robots

Ten Robots Scenario

This *Scenario* plots much complicated both robots and obstacles. It confirms two angles, first is to check the motion-planning method to execute correct. Second are to make them send lots of messages not only *Management Network* but *Experiment Network*.

Table 6.1.3 and figure 6.1 describes the coordinates of robots and obstacles.

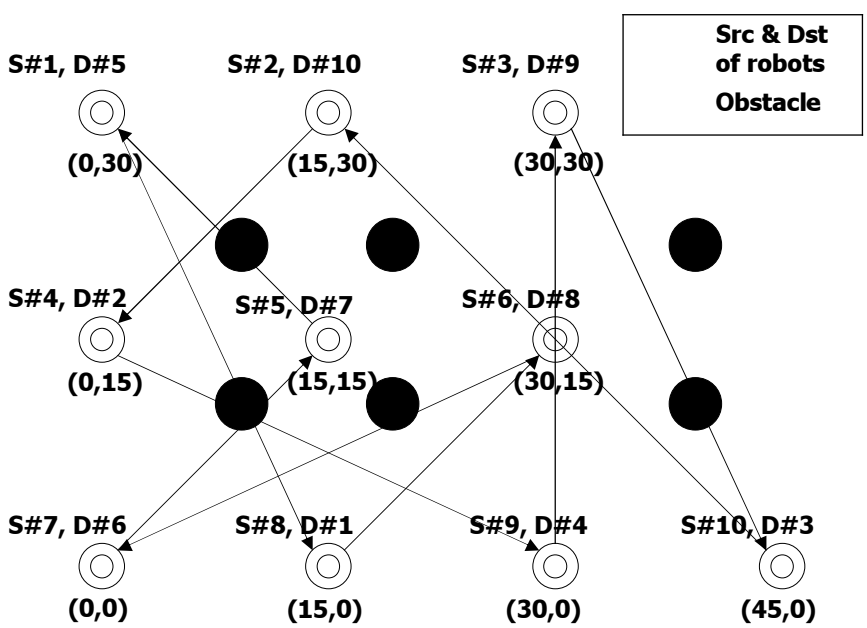

Figure 6.1: Scenario 3 : Ten robots and six obstacles

object	ID	radius	source	destination
robot	rb01	$1.0(m)$	$(0.0, 30.0)$	$(15.0, 0.0)$
robot	rb02	$1.0(m)$	$(15.0, 30.0)$	$(0.0, 15.0)$
robot	rb03	$1.0(m)$	$(30.0, 30.0)$	$(45.0, 0.0)$
robot	rb04	$1.0(m)$	$(0.0, 15.0)$	$(30.0, 0.0)$
robot	rb05	$1.0(m)$	$(15.0, 15.0)$	$(0.0, 30.0)$
robot	rb06	$1.0(m)$	$(30.0, 15.0)$	$(0.0, 0.0)$
robot	rb07	$1.0(m)$	$(0.0, 0.0)$	$(30.0, 15.0)$
robot	rb08	$1.0(m)$	$(15.0, 0.0)$	$(30.0, 15.0)$
robot	rb09	$1.0(m)$	$(30.0, 0.0)$	$(30.0, 30.0)$
robot	rb10	$1.0(m)$	$(45.0, 0.0)$	$(15.0, 30.0)$
obstacle	obs01	$1.0(m)$	$(10.0, 10.0)$	none
obstacle	obs02	$1.0(m)$	$(10.0, 20.0)$	none
obstacle	obs03	$1.0(m)$	$(20.0, 10.0)$	none
obstacle	obs04	$1.0(m)$	$(20.0, 20.0)$	none
obstacle	obs05	$1.0(m)$	$(40.0, 10.0)$	none
obstacle	obs06	$1.0(m)$	$(40.0, 20.0)$	none

Table 6.6: Scenario 3 : Ten robots and six obstacles

Large-Scale Scenario

This large-scale *Scenario* set fifty and a hundred number of robots. Every coordinate of robots and obstacles are defined by *Map Manager* at random in the range.

Scenario	number of robots	number of obstacles	initial range
Scenario4	50	20	$100(m)$
Scenario5	100	40	$200(m)$

Table 6.7: Scenario 4 and 5 : Fifty and a hundred robots

6.2 Experiment Results

Based on above conditions and definitions, the results of every *Scenario* are shown in this section.

6.2.1 Simple Scenarios

Scenario 1 : One robot and one obstacle

Figure 6.2 draws the trajectory of the robot. This *Scenario* is confirming simply the detection of the obstacle. The robot *rb01* starts from its source and approach the *Visual*

sensor range. The robot re-plans its trajectory and avoids the obstacle after detecting it about the coordinate $(-8, 8)$.

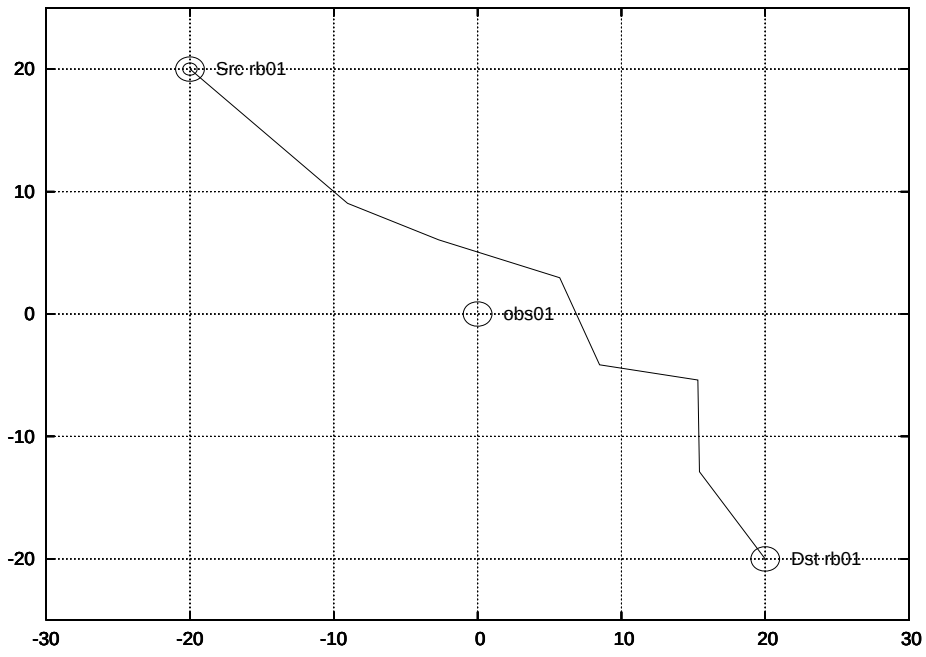

Figure 6.2: Robot trajectory of Scenario 1 : One robot and one obstacle

Scenario 2 : Two robots

This *Scenario* also tests another simple detection which the robots receive the WLAN radio wave. Figure 6.3 shows this situation, the robot $rb01$ and $rb02$ sense each other about $(-8,)$. As the robot $rb01$ has lower priority than $rb02$ and it re-plans and avoids possible collision.

6.2.2 Scenario 3 : 10 Robots Scenario

This *Scenario* is very complicated in small range, ten robots and six obstacles. The radius of robots and obstacles is $1.0(m)$, the priority is the higher the id of robot is, the more the priority is also.

Figure 6.4 describes *Scenario* 3 with algorithm 1, and figure 6.5 is algorithm 2's. The trajectories of algorithm 2 tend to go through linear. At first robot10 goes straight to its destination, and the other robots avoid it depending on the priority (the priority of robot10 is highest). Now from figure 6.5, some interesting results can be shown. For example robot8 go straight till sensing robot10 (about at $(22, 8)$), but it detects the collision on its trajectory and change it a little. The trajectory of robot3 and robot5 are

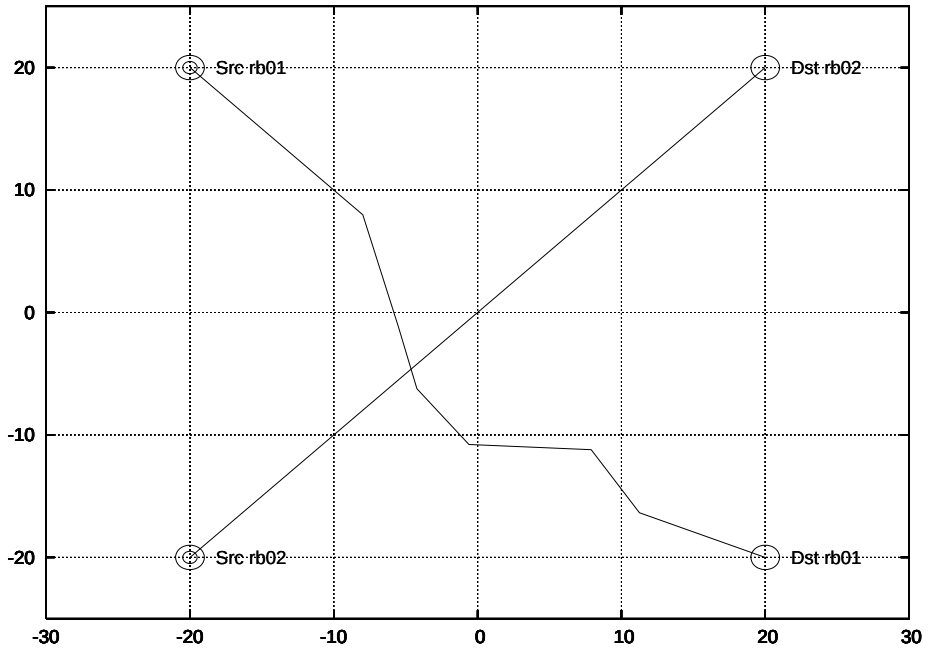

Figure 6.3: Robot trajectories of Scenario 2 : Two robots

same as the situation of *Scenario* 1, but robot3 did not choose the confusion area, this is why the proposed planning algorithm tends to expand the milestones more free area.

6.2.3 Large Scale Scenario

Following figure 6.6 and 6.7 describes large-scale environments. The radius of robots is $1.0(m)$ and the obstacle's is at random from $1.0(m)$ to $2.0(m)$. Accurate numbers of the robots are fifty-five and a hundred and ten. The lines are the trajectory of each robot, and + is the coordinate of the obstacles. These figures show collision avoidance which every robot does not cross with obstacles.

In this chapter, the definitions of the experiment environments and the *Scenario* every situation (simple and complicated and large-scale) are defined. Finally the results every *Scenario* are described.

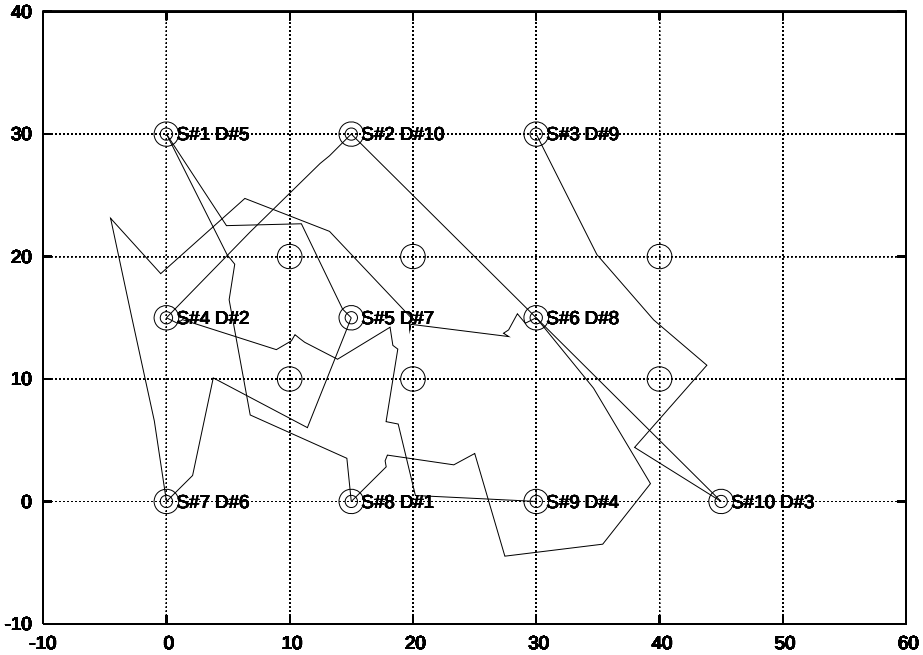

Figure 6.4: Robot trajectories of Scenario 3 : Ten robots and six obstacles with Algorithm 1

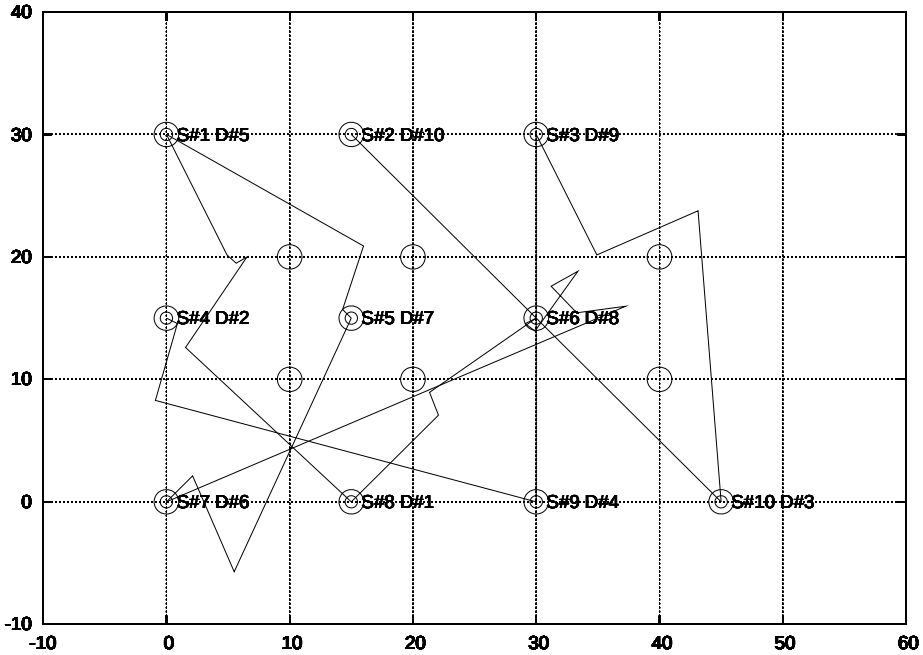

Figure 6.5: Robot trajectories of Scenario 3 : Ten robots and six obstacles with Algorithm 2

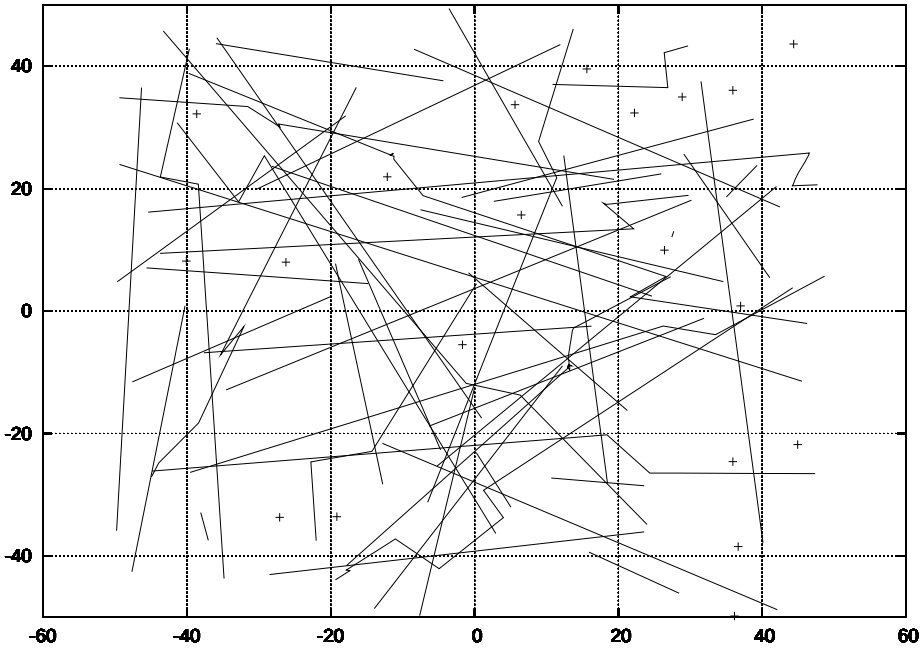

Figure 6.6: Robot trajectories of Scenario 4 : Fifty robots and twenty obstacles

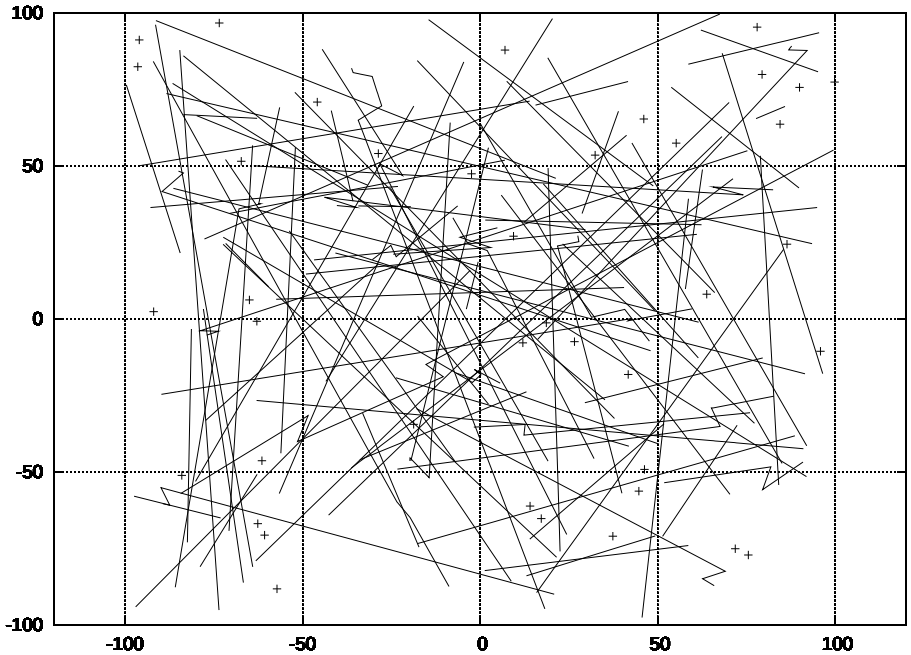

Figure 6.7: Robot trajectories of Scenario 5 : A hundred robots and forty obstacles

Chapter 7

Discussions

In this chapter, at first the evaluations of the proposed methods are shown and make it clear that what is improved and what is solved by them. Moreover the considerations about these proposed methods is mentioned.

7.1 Motion Planning

The proposed algorithms are adopted to the conditions which are large-scale autonomous mobile networked robots and are needed to find a path as soon as possible. These are came from the unknown and dynamic environment. Even if the motion-planning algorithm find optimal path, there may change the environment soon. On the basis of above conditions, I proposed new motion-panning algorithms.

The experiment results of figure 6.4 showed a possibility which improve the proposed algorithms. Actually proposed algorithm 1 grows the tree from near current coordinate, and it does not care about far range meaning unknown or not detecting area. Moreover for reducing the time when the robots reach the destination, the algorithm shall avoid only possible collision using PRM method and if the robot can see the destination without collisions, simply the robots go straight trajectory. This result is described in figure 6.5.

Figure 7.1 describes the comparison of total time when take from source to destination of algorithm 1 and 2 and ideal time when the robots go through straight trajectory to the destinations with maximum velocity. Some of robots of the graph go through very similar trajectory with ideal one. The results of robot number 1 and 3 and 5 with algorithm 1 are faster than algorithm 2's a little, but robot 4 and 6 and 8 take much time to reach their destinations.

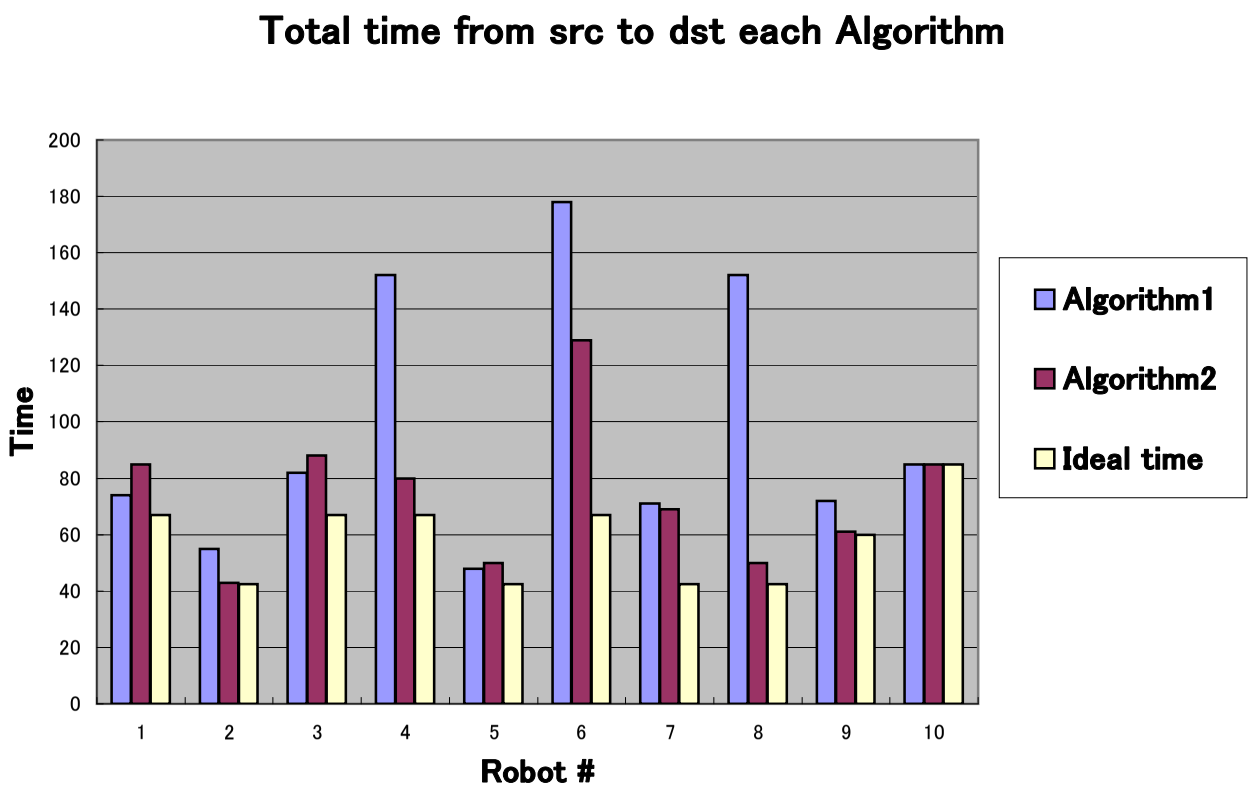

Figure 7.1: Total time from src to dst each Algorithm

Figure 7.2 and 7.3 describe the total waiting time and the total number which robots turn around every algorithm. These results clearly shows the difference of the each amount.

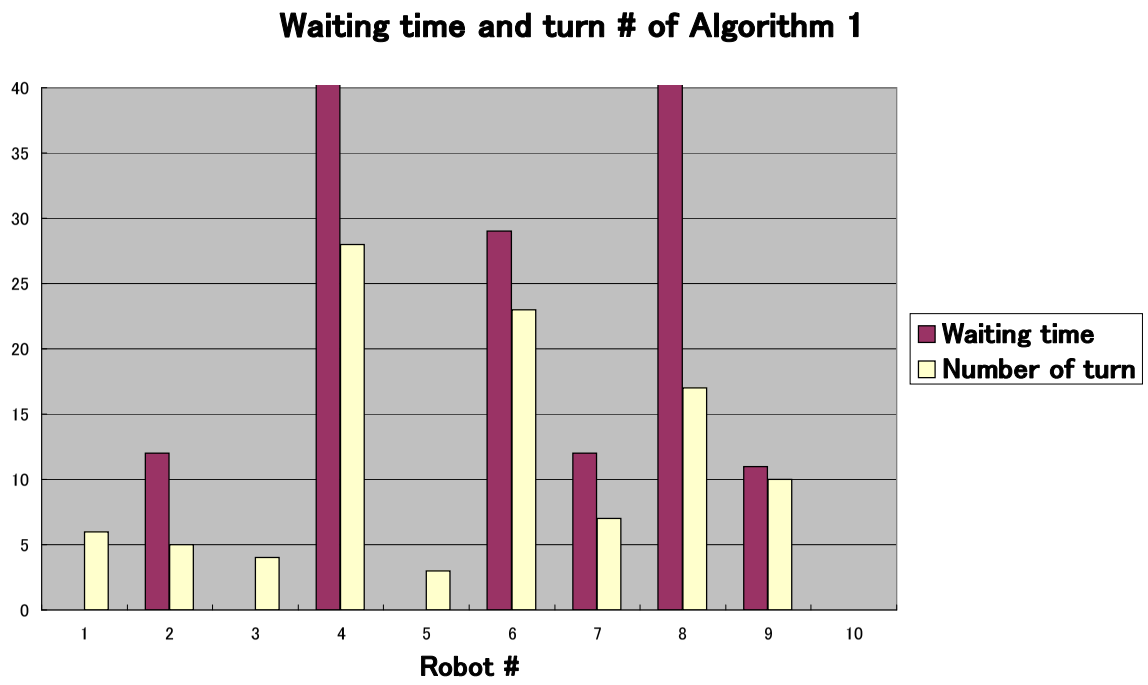

Figure 7.2: Waiting time and turning number of Algorithm 1

The total time when robots reach their destinations increase if they choose indirect paths or stay same position and try to re-plan a trajectory of collision-free. Then if the system increases the orders of the motion-planning, for example the number of milestones or the distances which pick up the milestones or the maximum time of re-planning, the robots do not tend to wait and re-plan same position. But it means that the planner has to takes longer time to re-plan and the robots need to allocate more memory storage. This relationship is tradeoff, the researchers shall choose the suitable orders to every scenarios. In this experiment, the robots shall equip the algorithm 2. If the conditions of the environment are more complicated, the algorithm 1 may return better result. For example "*Narrow passage*" [17] case, the possible trajectory of this case is limited by obstacles as very narrow chink. But in these case, the planner shall be equipped a different kind of sampling strategy.

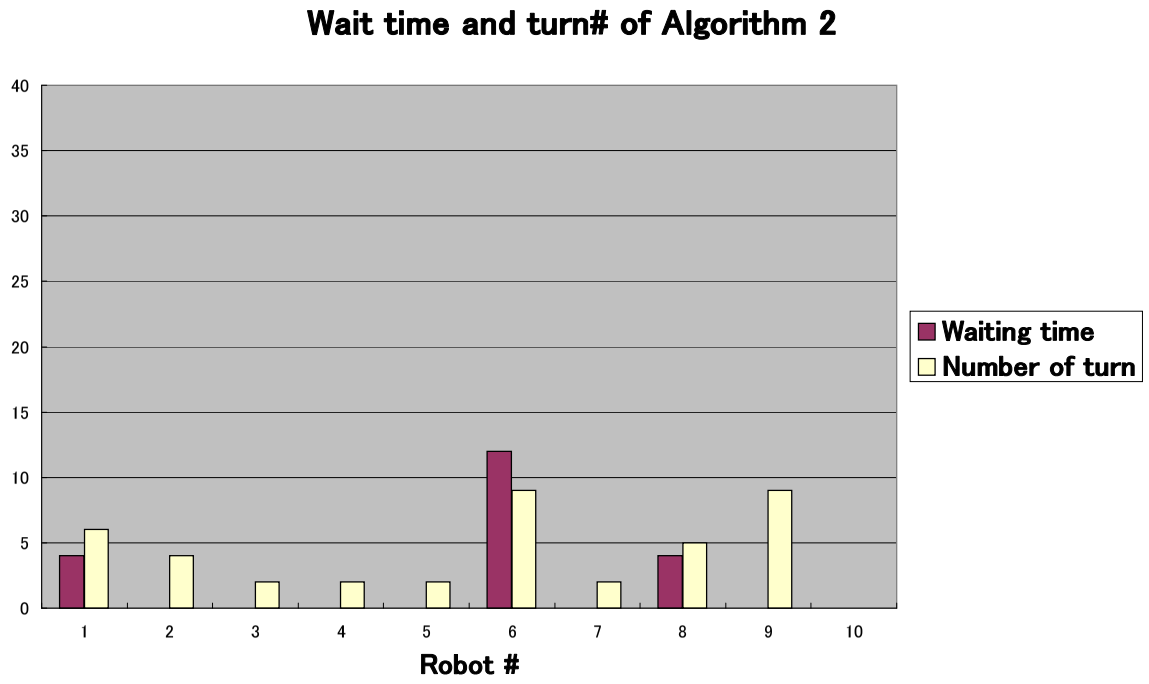

Figure 7.3: Waiting time and turning number of Algorithm 2

7.2 Experiment Platform

Various experiment methods of autonomous mobile networked robots (given Section 2) were tested before, but these method does not fulfill the scale and the accuracy and the cost problems at once. This system adopt large-scale experiments of them and emulations of environments.

7.2.1 Evaluation of the Experiment Platform

Scale

In this experiment, the most essential result is to accomplish the execution which a hundred and ten emulated autonomous mobile networked robots run successful. They communicate and cooperate each other and avoid possible collisions. And they fulfill to reach to their destinations. These numbers are much larger than previous researches.

There may exists problems which can not find in experiments with small number of robots, for example the intersections or delays caused by huge network traffic. These unexpected problems enable to be ready for experiment on real environment. And the bugs of the applications also tend to be found.

Next, important outcome is the experiments executed in real time, though large scale robots run. If the system executes more than hundreds robots, it should distribute

the network packets, especially *Map Manager* is. And if it try to execute more than 700 (the number of limitation of PCs of StarBED), the applications should be executed more than two processes on one PC.

Accuracy

The conditions of this experiment are emulated upon real environment, the messages from device driver are supported by *Map Manager*. *Map Manager* and robots share these conditions, they are able to receive the hardware messages in real time. In network emulation point, every robots communicate using real packets. These communication conditions are limited by WLAN emulation, and this limitations are depending on emulated parameters. Then the robots cannot transmit packets if they are in the connection range upon the environment. And this system is also running in real time.

Cost

This system especially saves costs on various points. At first these real autonomous mobile networked robots which equip motors and PC and WLAN card and more are expensive. If researchers try to experiment large scale robots, these cost expand huge. Next are the costs of the time. Some experiments on real environment take much time to prepare the system environment. And other view point, the software simulators also execute much times longer than the running time of real experiment. And if they implement their method on software simulators, they have to create modules or codes depending on software simulators. These methods and programming languages are many different kinds, and they have to do this with not accustoming way. When they use the proposed system, they just run their application on PCs.

Table 7.2.1 describes the above results of comparison.

Experiment platform	**Scale**	**Accuracy**	**Cost**
Software simulators	depending on performance of H/W	modeling	creat the modules or scenarios depending on each simulators, executing time
Real hardware	more less then 10	real	(i.e) the cost of one real robots is hundreds thousand yen
Proposed platform	currently 100	emulated	using testbed and integration

Table 7.1: Comparison Experiment platform

Chapter 8

Future Works

In this chapter, makes it clear that what accomplish to and what cannot realize, mentions the works from now on.

8.1 Implementation of MANET protocols

A mobile ad hoc networks (MANET) is a collection of mobile nodes that can communicate with one another without using any fixed networking infrastructure. It can be used in tactical operations, rescue missions, national security or sensor networks and so on. Mobile ad hoc networks consist of many kinds of devices that can autonomously self-organize in networks. Recently, this technology has been used for robot coordination in fields. In a dynamic robot networks equipped with MANET, some mechanisms for collision avoidance are necessary. These robots can act as routers in the network and transfer the information to each other. For example, N robots equipped with MANET can be assigned different individual tasks. To fulfill their tasks, they have to move to different locations and implement their missions, step-by-step. Based on this information, the trajectories are pre-planned. And robots can also respond to the updated information each time by new trajectories to ensure the robot motion is free of collision. Some routing protocols have already been proposed, but hey either cannot be really collision-free or will incur deadlocks. Therefore, we need to provide a deadlock-free collision avoidance method for coordinating a dynamic robot network.

By equipping MANET protocols, the robots can predict larger range than current system. The similar function can realize on current system, but the messages exchanged by the robots are just forwarded. If robots create MANETs, It is easy to equip "*centralized*" motion-planning. The emulation of MANET protocols is not equipped in current system, that is why cannot solve the problems between *dummynet* and MANET routing implementation. The *dummynet* applies the various configuration values between *Layer 2* and *Layer 3*, it makes a pipe and set ip address from and to, but MANET protocols check messages *Layer 2* by MAC address. Figure 8.1 describes this situation. This assumes all of nodes to constructing MANET and connecting each other. If node A want to transmit a data to node C, the packets are bound for node C through node B.

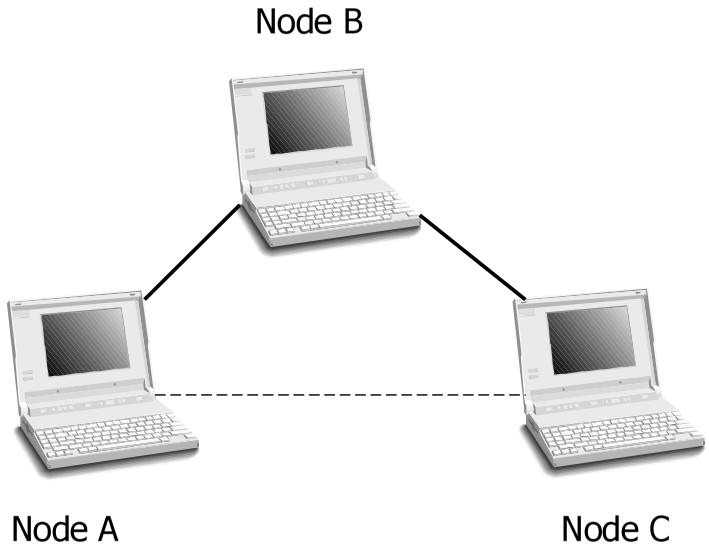

Figure 8.1: The problem of MANET emulation

But the packets can be received directly by node C and node C receives the messages twice. (see Figure 8.2).

Then this system needs some method to solve this problem.

Now there are temporal methods to fulfill applying MANET protocols. First is that treat the messages *Application Layer*, the messages includes the address which want to send and the application forward these messages. Figure 8.3 describes this situation.

But this cannot be said emulation, because there exists overhead which be caused when transmit upper *Layer 2*.

Another method is that the application pre-computes the route to a destination. Thereafter it applies the summation of the configuration value and sends messages directly. Figure 8.4 describes this situation.

But this method also don't care about the affection which cause when the nodes forward messages. But if the configuration parameters include this affection, this method can have reality.

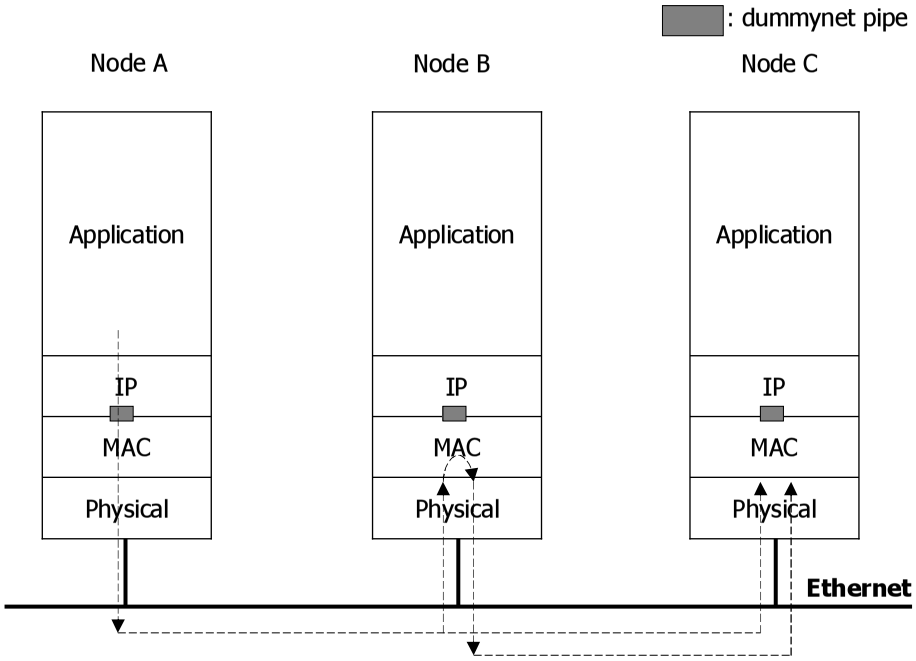

Figure 8.2: The problem of MANET emulation

8.2 Experiment Framework

This experiment platform supports various large-scale mobile networked nodes and enable to tests the proposals of researchers. It currently supports only WLAN networks and simple ad-hoc network, but the other various networks will be included in future.

If the number of nodes increases, it needs some integration functionalities, for example time synchronization or clustering or load balance. The time synchronization especially affects all of the conditions of this system. If the time difference between *Map Manager* and Robots causes, it means that the time when robots receive the hardware messages late or fast, this is critical. An experiment of large-scale nodes with hundreds or more has to include some structure which solve these problems.

Finally the system has demands how it can use clearly and easily, then the emulated objects should be defined to re-use simply and the system also has intelligible interfaces.

8.2.1 StarBED2

The proposed experiment platform aims to multipurpose platform supporting various networks, this needs generalization to suit various conditions. StarBED2 (mentioned in section 2) is one of the goal of these platform, it covers ubiquitous networks. Figure 8.5 describes a town emulation which is realized by StarBED2. Ubiquitous networks have various aspects, for example network, IEEE802 family and wireless networks and sensor networks, and telephone lines are used as access network for home networks. The figure

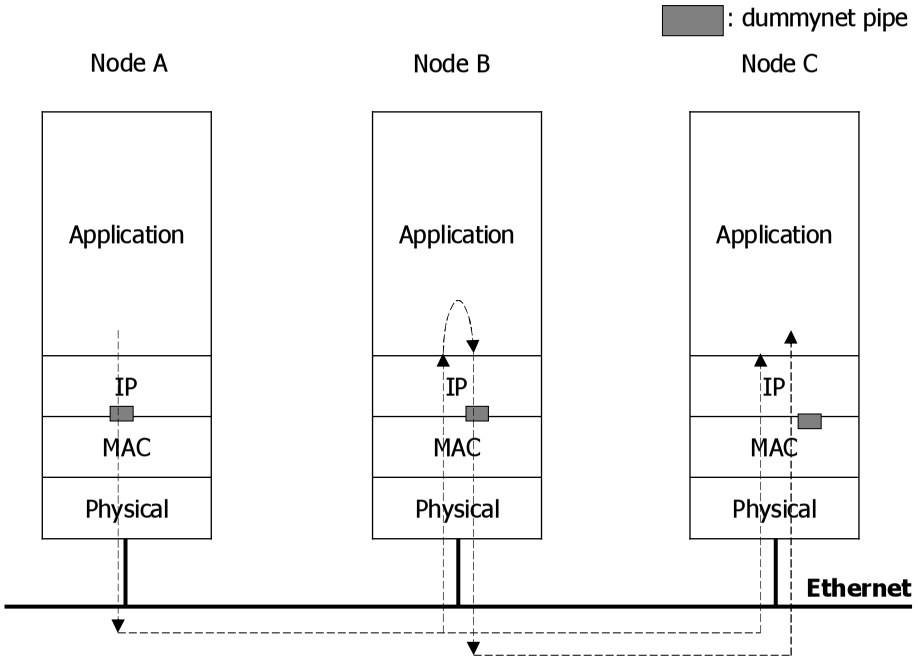

Figure 8.3: Application forwards messages

shows the emulation of entire of a town. By advancing this research, we drive for realizing StarBED2.

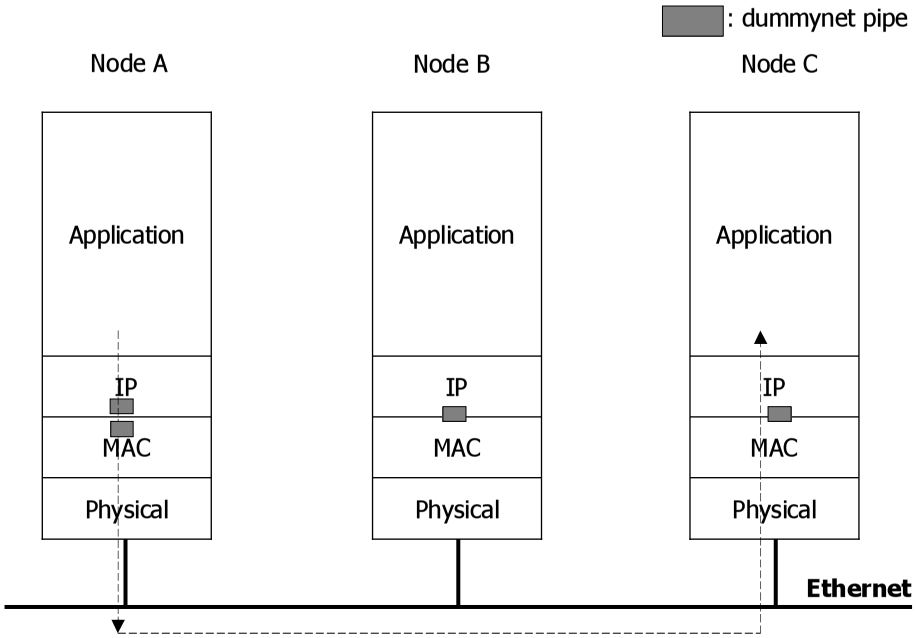

Figure 8.4: Apply configurations at once

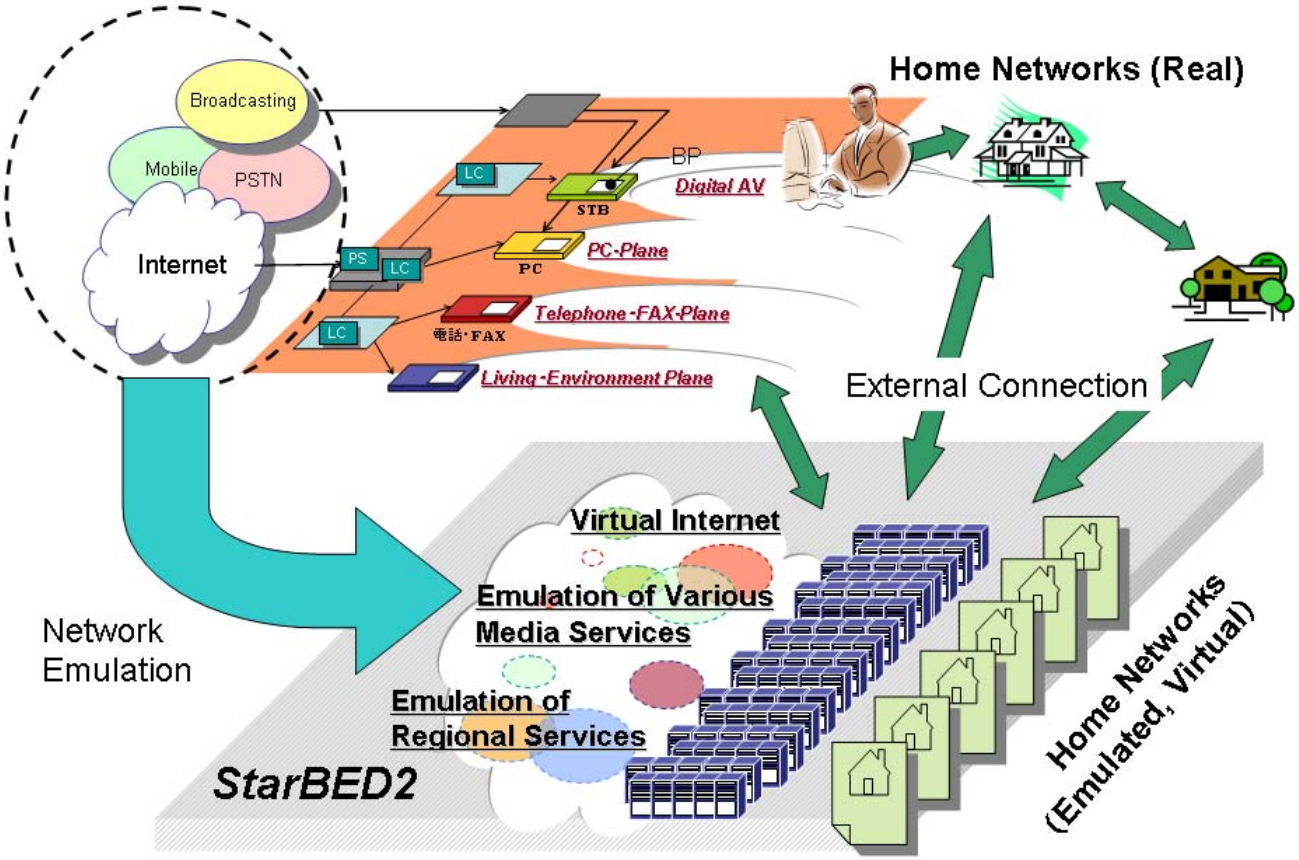

Figure 8.5: Town emulation by StarBED2

Chapter 9

Conclusions

This research focus on two main themes, first is the experiment platform of large scale autonomous mobile networked robots on unknown and dynamic environment. Second one is the motion-planning algorithms of above robots.

In above conditions, the motion-planner needs especially speed, and I improve PRM planner and propose new algorithms for executing our system. The planner is divided two phases, first phase expand the roadmap in direction which are low density. This roadmap expansion includes the time parameter and cover the configuration time space. The second phases are connection which try to connect between the roadmap and the destination. At that time proposed algorithm 1 tries to connect between only the milestones in constant range and the destination. Another algorithm 2 tries to connect the newly added milestones and the destination every step.

Next I proposed the experiment platform of above system. This method uses real PCs and assumes them to be autonomous mobile networked robots. Various objects are emulated depending on every modeling. The hardware of the robots is emulated by *Map Manager*. *Map Manager* controls all of the hardware messages of robots. The WLAN network emulation is realized by applying the configuration values to *dummynet* pipes, and this configuration values are given by WLAN emulator QOMET. The integration of entire system is managed by *RUNE*.

The experiment on this system enabled to accomplish more than a hundred numbers of the robots. And on our way to test the system, some new aspects about the algorithm can be found.

Acknowledgements

I would like to thank my superviosr, Associate Professor Yasuo Tan, for giving me various suggestions. I would also like to thank Research Associate Dr. Razvan Beuran, Mr. Junya Nakata for assists and useful discussion, and thank also Associate Ken-ichi Chinen, Research Associate Dr. Chao Peng and StarBED members and Tan-laboratory.

Appendix A

Communication between Map Manager and Robots

Map Manager emulates the hardware messages of robots, then a part of source code of *Map Manager* and robots are described as follows :

A.1 Initialization of Map Manager

Following is a part of source code of initializing connections with robots and sending setting parameters.

```
⋮
  /* definitions */
  struct robot_info robots[MAX_NODES]; /* structure of robots */
  struct map_manager_message msg; /* message structure from Map Manager */
  struct pollfd client[OPEN_MAX];
  struct sockaddr_in cliaddr;
⋮
  /* loop number of client times */
⋮
  client[i].fd = socket(AF_INET, SOCK_STREAM, 0);
  bzero(&cliaddr, sizeof(cliaddr));
  cliaddr.sin_family = AF_INET;
  cliaddr.sin_port = htons(MAP_MANAGER_PORT);
  inet_pton(AF_INET, robots[i].ipaddr, &cliaddr.sin_addr);
  connect(client[i].fd, (struct sockaddr *) &cliaddr, sizeof(cliaddr)));
  client[i].events = POLLRDNORM;

  /* send the initial configurations to robots */
  write(client[i].fd, &msg, sizeof(msg));
```

⋮

A.2 Initialization of Robots

Following is a part of source code of initializing connections with Map Manager and receiving setting parameters.

```
⋮
  /* definitions */
  struct pollfd map_manager[2];
  struct map_manager_message msg; /* message structure from Map Manager */
  int listenfd, mmlen;
  struct sockaddr_in servaddr, mmaddr;
⋮
  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  bzero(&servaddr, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(MAP_MANAGER_PORT);
  bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
  listen(listenfd, LISTENQ);
  map_manager[0].fd = listenfd;
  map_manager[0].events = POLLRDNORM;
  nready = poll(&map_manager[0], 1, INFTIM);
  /* loop till connecting with Map Manager */
⋮
  if (map_manager[0].revents & POLLRDNORM) {
    map_manager[1].fd = accept(listenfd, (struct sockaddr *) &mmaddr, &mmlen);
    map_manager[1].events = POLLRDNORM;
⋮
  /* receive the initial configurations from Map Manager */
  nready = poll( &map_managr[1], 1, INFTIM);
  if (map_manager[1].revents & (POLLRDNORM  POLLERR)) —
    read(map_manager[1].fd, &msg, sizeof(msg));
⋮
```

Appendix B

Configure dummynet

dummynet simulates and enforces queue and bandwidth limitations, delays, packet losses, and multipath effects on FreeBSD. These limitations are applied to IP Firewall. And the method how to apply these on IP Firewall is given as follows :

B.1 Making a new pipe

Following is a part of an example to make a new pipe of dummynet. These codes set a pipe from source address to destination address with pipe number.

```
⋮
  fucntion_name(in4_addr *src_addr, in4_addr *dst_addr, int pipe_nr)
⋮
  int clen = 0;
  ipfw_sn cmd[];
  struct ip_fw *p;
  /* apply src address */
  cmd[clen].opcode = O_IP_SRC;
  cmd[clen].len = 2;
  cmd[clen].arg1 = 0;
  ((uint32_t *)cmd)[clen + 1] = src_addr.word;
  clen += 2;
  /* apply dst address */
  cmd[clen].opcode = O_IP_DST;
  cmd[clen].len = 2;
  cmd[clen].arg1 = 0;
  ((uint32_t *)cmd)[clen + 1] = dst_addr.word;
  clen += 2;
  /* set pipe with pipe number */
  cmd[clen].opcode = O_PIPE;
```

```
  cmd[clen].len = 2;
  cmd[clen].arg1 = pipe_nr;
  ((uint32_t *)cmd)[clen + 1] = 0;
  clen += 1;
  p = (struct ip_fw *)malloc(sizeof(struct ip_fw) + clen * 4);
  bzero(p, sizeof(struct ip_fw));
  p->act_ofs = clen -1;
  p->cmd_len = clen + 1;
  p->rulenum = rulenum;
  bcopy(cmd, &p->cmd, clen * 4);
  getsockopt(s, IPPROTO_IP, IP_FW_ADD, p, sizeof(struct ip_fw));
⋮
```

B.2 Configure a pipe

Following is a part of an example to configure a pipe of dummynet. And the parameters of bandwidth and delay are applied to the pipe suiting pipe number.

```
⋮
  struct dn_pipe p;
  bzero(&p, sizeof(p));
  p.pipe_nr = pipe_nr;
  p.bandwidth = bandwidth;
  p.delay = delay;
  setsockopt(s, IPPROTO_IP, IP_DUMMYNET_CONFIGURE, &p, sizeof(p));
⋮
```

B.3 Remove a pipe

Following is a part of an example to remove a pipe with pipe number pipe number of dummynet.

```
⋮
  struct dn_pipe p;
  bzero(&p, sizeof(p));
  p.pipe_nr = pipe_nr;
  setsockopt(s, IPPROTO_IP, IP_FW_DEL, &p, sizeof(p));
⋮
```

Appendix C

Integration with RUNE

The proposed experiment platform uses *RUNE* for the experiment integration, the integration process with *RUNE* is menthioned in this appendix.

Figure C.1 describes logical structure of *RUNE*. As it is mentioned in section 4.5, "space" that emulates each experiment target can communicate through "conduits" without any extra operations. These are fulfilled by *RUNE-read* and *RUNE-write* (figure C.2).

The integration with *RUNE* is following :

Define the Spaces
"spaces" is divided into "initialize" and "iteration" and "finalize". *RUNE-read* and *RUNE-write* are call-back function which is called when *RUNE* manager inform these function calling. These also should be defined.

Write the experiment definition file
The experiment definition file should be written. It describes the "spaces" and "conduits" in the experiment (an example is given folloing).

Compile and execute the system
RUNE master is compiled with the definition file and sends the instruction "attach process" to the *RUNE* manager executed on each node.

The experiment definition file of RUNE

```
BGNSPACELIST
  SPACE(map_manager, 172.16.3.1, map_manager.so.1)
  SPACE(robot1, 172.16.3.2, robot.so.1)
  SPACE(robot2, 172.16.3.3,robot.so.1)
ENDSPACELIST

  BGNCONDUITLIST
  CONDUIT(robot1, map_manager)
  CONDUIT(robot2, map_manager)
  CONDUIT(map_manager, robot1)
  CONDUIT(map_manager, robot2)
  CONDUIT(robot1, robot2)
  CONDUIT(robot2, robot1)
ENDCONDUITLIST
```

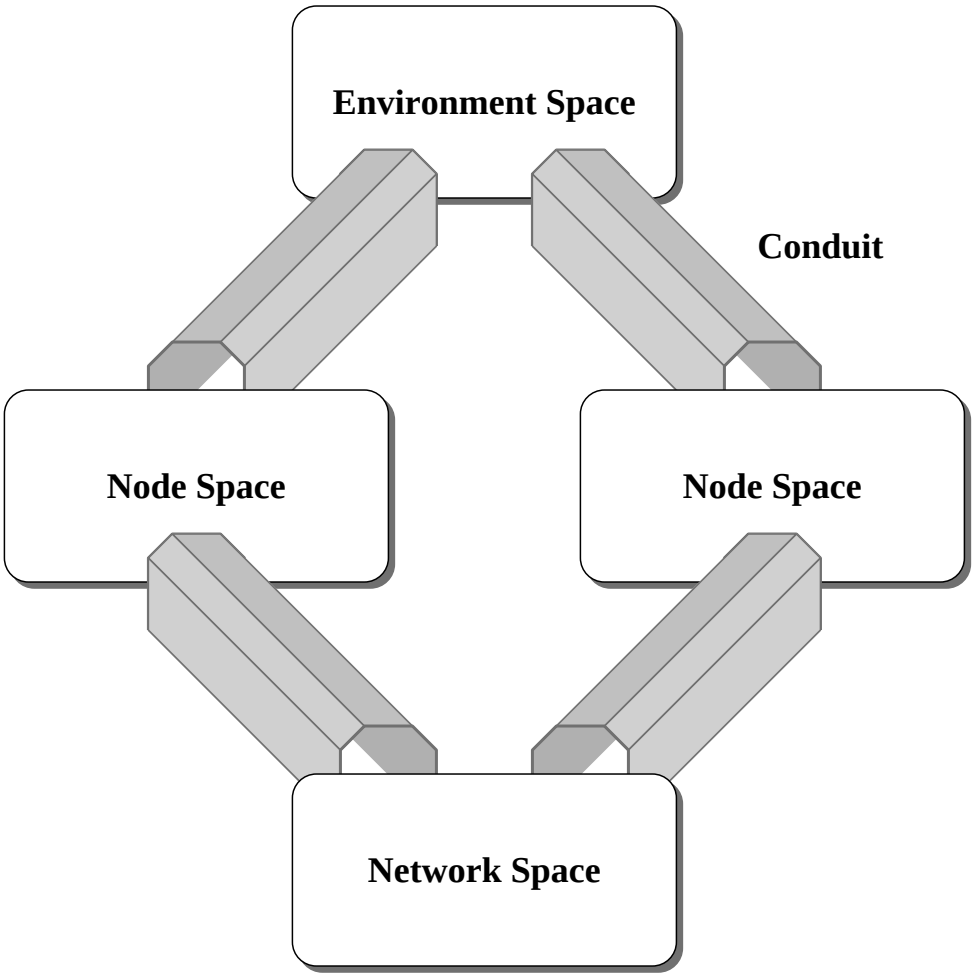

Figure C.1: Logical structure of RUNE

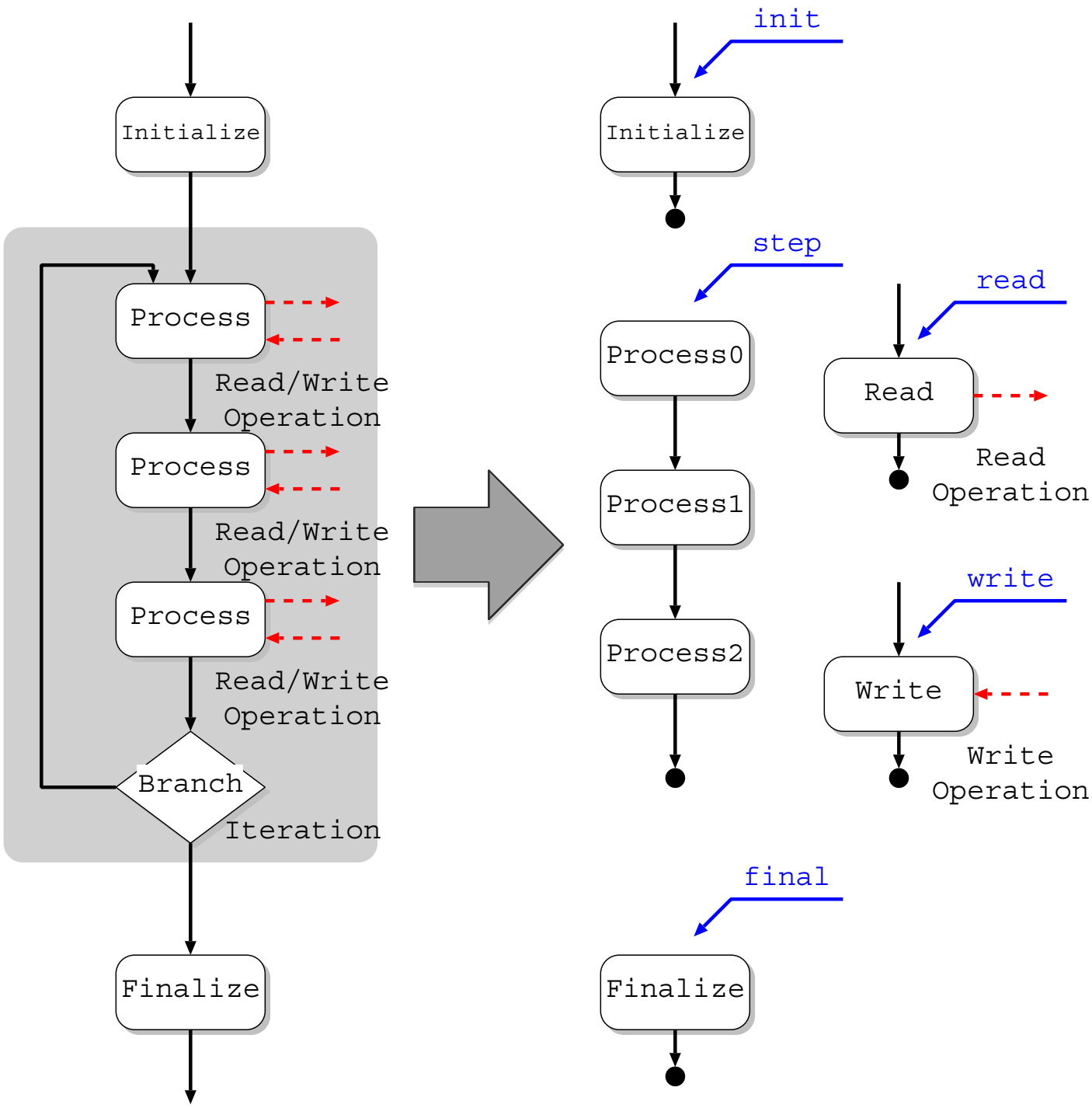

Figure C.2: Logical structure of RUNE

Bibliography

[1] Chao Peng *Coordination and Collision Avoiding in MANET.* Japan Advanced Institute of Science and Technology.

[2] B.R. Donald. *A search algorithm for motion planning with six degrees of freedom.* Artificial Intelligence, 31(3):295-353, 1987

[3] D. Hsu, J.C. Latombe, and R. Motowani *Path Planning in Expansive Configuration Spaces.* In Proc. IEEE Int. Conf. on Robotics and Automation, pages 2719-2726, 1997

[4] L. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars *Probabilistic roadmaps for path planning in high-dimensional configuration space.* IEEE Transactions on Robotics and Automation, 12(4):566-580, 1996.

[5] RoboCup *The Robot World Cup Soccer Games and Conferences.* http://www.robocup.org/

[6] C.M. Clark, S.M. Rock, J.C. Latombe *Dynamic Networks for Motion Planning in Multi-Robot Space Systems.*

[7] D. Hsu, R. Kindel, J.C. Latombe, S.M. Rock *Randomized Kinodynamic Motion Planning with Moving Obstacles.*

[8] StarBED *A Large Scale Experiment Network Environment.* http://www.starbed.org/

[9] L. Kavraki, J.C. Latombe, R. Motwani, and P.Raghavan. *Randomized query processing in robot path planning.* In Proc. ACM Symposium on Theory of Computing, pages 353-362, 1995.

[10] Webots *Fast Prototyping and Simulation of Mobile Robots.* http://www.cyberbotics.com

[11] Stanford Aerospace Robotics Laboratory http://arl.stanford.edu

[12] Chaimowicz, L., *et al. Deploying Air-Ground Multi-Robot Teams in Urban Environments.* Proc. of the 2005 International Workshop on Multi-Robot Systems, Washington DC, U.S.A, pp. 223-234. (March 2005)

[13] K.Kant and S.Zucker. *Toward efficient trajecotry planning: The path-velocity decomposition.* International Journal of Robotics Research, 5(39:72-89, 1986.

[14] Taixiong Zheng, D.K. Liu, Ping Wang *Priority based Dynamic Multiple Robot Path Planning* 2nd International Conference on Autonomous Robots and Agents December 13-15, 2004 Palmerston North, New Zealand

[15] R. Beuran, L.T. Nguyen, K.T. Latt, J. Nakata, Y. Shinoda *QOMET: A Versatile WLAN Emulator* IEEE International Conference on Advanced Information Networking and Applications (AINA-07), Niagara Falls, Ontario, Canada, May 21-23, 2007

[16] Rizzo, L *Dummynet FreeBSD network emulator.* http://info.iet.unipi.it/ luigi/ip_dummynet.

[17] D.Hsu, T. Jiang, J. Reif and Z. Sun *The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners.* Proceedings of the IEEE International Conference on Robotics and Automation, Taipei, 2003.

[18] J. Nakata, T. Miyachi, R. Beuran, K. Chinen, S. Uda, K. Masui, Y. Tan, Y. Shinoda *StarBED2: Large-scale, Realistic and Real-time Testbed for Ubiquitous Networks* TridentCom 2007, Orlando, Florida, U.S.A., May 21-23, 2007.