ELECTRICAL ENGINEERING

THE SELF AGENT FOR MOBILE ROBOT

KANOK KUSUMALNUKOOL

Thesis under the direction of Professor Mitch D. Wilkes

In robotics, we need robots which have the ability to understand what a human wants and can respond accordingly. On the other hand, robot should be able to provide information that is easy for a human to understand. Therefore, we developed the robot Self Agent to provide these abilities. The main purpose of the Self Agent is to be the center of the robot. It provides every basic need that a robot should have. For example, translating high-level command into basic commands so that a robot can understand and execute them; Self-monitoring of the robot's performance; and having an intelligent action selection that will make good decisions.

Since we needed to have a good decision mechanism, we created a new approach which consists of two algorithms. The first algorithm is Spreading Activation Network (SAN) that provides the basis for selecting appropriate behaviors (Action Selection) for completing a given task. To perform well, parameters of the Spreading Activation Network must be manually tuned. The second algorithm is the Reinforcement Learning (RL) technique that enables a robot to automatically learn multiple policies. This research will show we trained an ATRV-Jr robot, called Scooter to learn policies and automatically adapt to unexpected variables.

Approved By

D. Mitch Wilkes, Professor

THE ROBOT SELF AGENT FOR MOBILE ROBOT

By

Kanok Kusumalnukool

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2002

Nashville, Tennessee

Approved By

D. Mitch Wilkes, Professor

Richard Alan Peters II, Professor

ACKNOWLEDGEMENTS

Firstly, I would like to thank Dr. Mitch Wilkes who gave me the opportunity of pursuing graduate work at Vanderbilt University and working in the Intelligent Robotics Laboratory as well as for his academic, financial support and encouragement. Without his kindness, generosity and guidance, this work would have never been done In additions, I thank Dr. Kazuhiko Kawamura, Dr. Alan Peter III and Dr. Daniel Gaines for their guidance and suggestions throughout my research. Thank to Flotence C. Fottrell who assisted this work in many ways.

I would like to thank Dr. Pattie Maes for her research and papers on Spreading Activation Network. Thanks to the members of the Intelligent Robotics Laboratory for years of hard work, and contributions specially Anak, Jain, Ponchai, Surachai, Siripun, Palis, Turker, Carlotta, Tamara, Bugra, Natasha, and Roberto.

Finally, I would like to thank my parents and my brother for their continual support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Overview

This work investigates a Multi Agent System for mobile robots. Although many mobile robots today are capable of completing tasks in predictable environments, they still have problems when dealing with new unpredictable environments or unexpected events. Currently robots have limited sensing of their surroundings and they are only able to deal with situations that they have already experienced. The difficulty of building an autonomous robot is making a robot learn and believe in some policies in an unpredictable environment.

If we think about what makes humans behave intelligently, then we may realize that intelligence has many levels [1]. Many animals are intelligent to some degree, including the ability to act within their environments. How do animals act within their environments? Intelligence provides animals with the computational power and knowledge to sense, decide, and act in a complex and changing environment.

The development of agent-based technologies was initiated in the 1990s within a field called Distributed Artificial Intelligence (DAI) [2]. An agent is an autonomous entity that tries to fulfill a set of different goals in achieving tasks in complex and dynamically changing environments. An actuated agent can sense the environment through its sensors and act upon the environment using its actuators. This kind of action can be represented as a behavior. The robot can perform multiple behaviors to achieve its tasks. Each behavior may consist of one agent or many agents in the system. A system composed of a number of agents being able to interact, and differing from each other in their skill and their knowledge about the environment, is called a Multi-Agent System (MAS). There are many different frameworks that provide the architecture to implement multi-agent systems. The Intelligent Machine Architecture (IMA), an agent-based software system for robot control, bas been developed in the Center for Intelligent Systems at Vanderbilt University [3].

The Self Agent is a compound cognitive agent that activates and maintains the activities of the robot itself. It receives high-level commands from the Commander Agent and decomposes them into a set of executable commands. The Self Agent also monitors the system performance, translates high-level commands into low-level commands that robot can understand, and reports significant errors back to the commander Agent.

We propose a new approach to select the low-level commands, called SAN-RL. It combines two techniques, a Spreading Activation Network (SAN) and Reinforcement learning (RL) together. The Spreading Activation Network (SAN) allows the robot to pick plans to achieve goals. However, setting these parameters for the network could be difficult and require a lot of experiment time. Therefore, we integrated Spreading Activation Networks (SAN) with a Reinforcement Learning (RL) technique that can learn the appropriate parameters, dependant upon a reward, which the robot gets when it completes its tasks.

Outline of the thesis

The remainder of this dissertation is divided as follows:

Chapter II: Background:

This chapter briefly describes the development of our Multi Agent System. The Schema theory is summarized. Several architectures that relate to intelligent robot control are reviewed. We focus on behavior-based approaches and agent-based systems. The Intelligent Machine Architecture (IMA) is introduced. Illustrative examples of work that have been implemented in IMA are reviewed.

Chapter III The robot Self Agent:

The main objective of this work is to create the core (or brain) of the robot. The robot needs to have the abilities to understand what the user wants to do and to execute this. Therefore, we developed the robot Self Agent to provide these abilities. The robot Self Agent is composed of: Command I/O Agent, Action Selection Agent, Performance Agent, Info Agent, Description Agent and Path Planning Agent.

Chapter IV Spreading Activation Network with Reinforcement Learning:

In the robot Self Agent, we need to perform an action selection for selecting the best possible action to complete a given task. We propose a Spreading Activation Network as the action selection mechanism. To perform well, the parameters of the Spreading Activation Network must be manually tuned which is time-consuming. Reinforcement learning techniques will help the robot learn policies, and thus these parameters automatically.

Chapter V Experimental Results and Discussion:

This chapter demonstrates a practical application of mobile robot navigation and learning. Many basic behaviors are combined and implemented in an outdoor scenario. The results show the ability of the robot to move around the environment and to finish the tasks. The experiments are setup for three different sets of policies. The results from all experiments are given and discussed in detail.

Chapter VI Conclusions and Future Research:

This chapter summarizes and discusses the significance of the results and gives an overview for the future research.

CHAPTER II

BACKGROUND

Agent-Base System

An agent-based system provides a way to combine intelligent behaviors. The concept is used, not only as a model for computer programming, but also in applications in robotics research.

Many people ask why we should use an Agent-Based System. There are several answers that bay be given in response to this question. One of them is the current robotics research is becoming more and more complex with numerous functionalities and interactions with several users. Also an Agent-Based System requires appropriate for a large-scale project, such as one distributed across a network, which requires many computers to work together to accomplish the overall objective. Each agent needs to solve the task that it is able to do and the others will take care of the rest. Together they do what is necessary to make the system function.

The Multi-Agent Systems (MAS)

Multi-Agent Systems (MAS) are becoming an important paradigm in intelligent mobile robot research. In MAS agents need to communicate with each other to complete a given task. Although an atomic agent can solve some particular problems, a complex problem typically requires multiple agents. MAS can provide distributed information, flexibility, fault-tolerance, and reusability.

The choice to use the MAS can therefore be explained by the following:

- Problems are physically distributed: Some complex problems require that computation be distributed over several locations. For example, airplane traffic management. We need to have agents connected together at airports around the country.
- Problems are widely distributed and heterogeneous in functional terms: It's hard to find one person who can solve every problem. Even if you find that person, he or she could not possibly be knowledgeable enough and have enough time to finish such a system on his or her own. For instance, in car racing we need to have a driver, engineer, specialists in engines, tires, etc. Everyone will put his or her knowledge together to try to make the best possible car.

- Networks force us to take a distributed view: In the current real world, much of the data processing power is distributed over the network. MAS can thus be seen as an appropriate candidate for the construction of open, distributed, heterogeneous and flexible architectures, capable of offering high-quality service for collective work, and also without changing any prior structure.
- The complexity of problems dictates a local point of view: When the problem is too big and complicated, there is no way to solve it by looking at the whole system. It must be decomposed into local approaches.
- Systems must be able to adapt to changes in the processing structure or in the environment: There is no need to spend a lot of time changing the entire system structure when there are only small changes or growth in the future. MAS allows for the integration and the appearance or disappearance of agents even while the system is functioning. It's especially suitable for making the growing adaptations necessary for the functioning of a growing system.

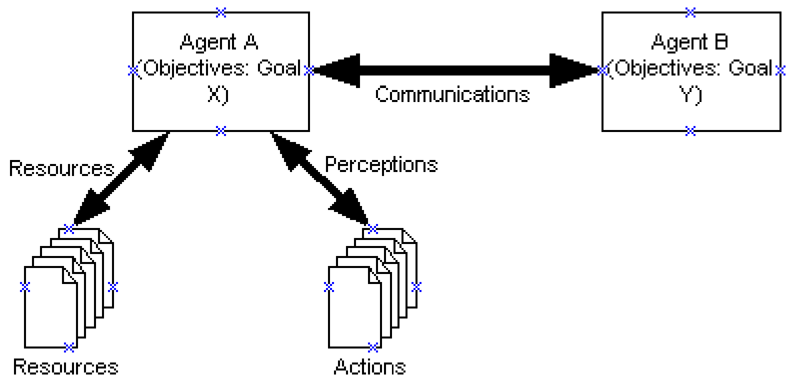


Figure 1: An agent interacting with its environment and other agents

The Intelligent Machine Architecture (IMA).

At Vanderbilt's Intelligent Robotic Lab (IRL), we developed a framework for building the intelligent software called the Intelligent Machine Architecture (IMA) to be used as the framework [4][5]. It was initially designed for a humanoid robot system called ISAC (Intelligent SoftArm Control) [6][7][8]. We are applying IMA to the mobile robot problem to test the robustness and flexibility of the software architecture across different types of robotic applications and operating system platforms. Our design process within the IMA is to decompose the system into a set of atomic agents. Within IMA, an atomic agent is an element in the system that is independent and is capable of acting in an environment, much like the concept of an object in object-oriented systems.

To complete the concept described above, IMA provides a two-level software framework for developing intelligent software:

- The robot-environment level: describes the system structure in terms of a group of atomic agents connected by a set of agent representations. At the robot-environment level, IMA defines several classes of atomic agents and describes their primary functions in terms of environmental models, behaviors or tasks.
- The agent-object level: describes each of the atomic agents and agent representations as a network of software modules called components.

Another benefit of IMA is ease of integration of new algorithms into the current system. Moreover, different systems and architectures can be implemented parallel with different agents so that the overall system can be fast interacting with their own environment. The communication among agents within different architectures is easier because the internal structures are independent.

The difference between IMA and traditional software systems has two aspects.

1. The IMA uses a system-level model that is based on atomic agents. Thus, each agent task or domain element is modeled in software as a primitive agent.
2. The IMA uses the Distributed Component Object Model (DCOM) [9] for communicating between components under Windows NT/2000. DCOM is a service of MS Windows, which allows remote objects to be controlled when they were registered.

Taxonomy of IMA Agents

In the IMA framework, the objects are arranged in a hierarchy. This taxonomy has also been described in [1][2]

Component

Components are reusable software objects developed under the Distributed Component Object Model (DCOM). Components can be created by any high-level language that supports DCOM.

Atomic Agent

There are four basic types of atomic agents, and a special type that combines more than one type together.

1. Hardware/Resource Agent: This is an agent that is used to interface with the actual hardware. This agent is on the lowest level of the hierarchy. Its duty is to provide raw data from hardware to other atomic agents with regular updates. Those agents could provide other services such as a control loop, rearranging data or filtering data for other agents.

2. Behavior/Skill Agent: Encapsulate basic behaviors or skills: This agent is a higher level than the Hardware/Resource agent. It consists of behaviors or primitive skills needed for the robot to perform some tasks. For example avoid-obstacle, tracking-object, wander and move-to-goal, etc.

3. Environment Agent: Provides an abstraction for dealing with objects in the robot's environment such as cone, ball, or other robots. This abstraction includes operations that the robot can perform on the object, e.g., "look at." or "move to".

4. Sequencer Agent: A sequencer agent provides a sequence of actions, which may interact with one or more other atomic agents. Sequencer agents may call other sequencer agents.

5. Multi-Type Agent: Combines the functionality of at least two of the atomic agent types. It provides a complicated agent that can combine the resource, behavior, environment and sequencer agents into a single agent.

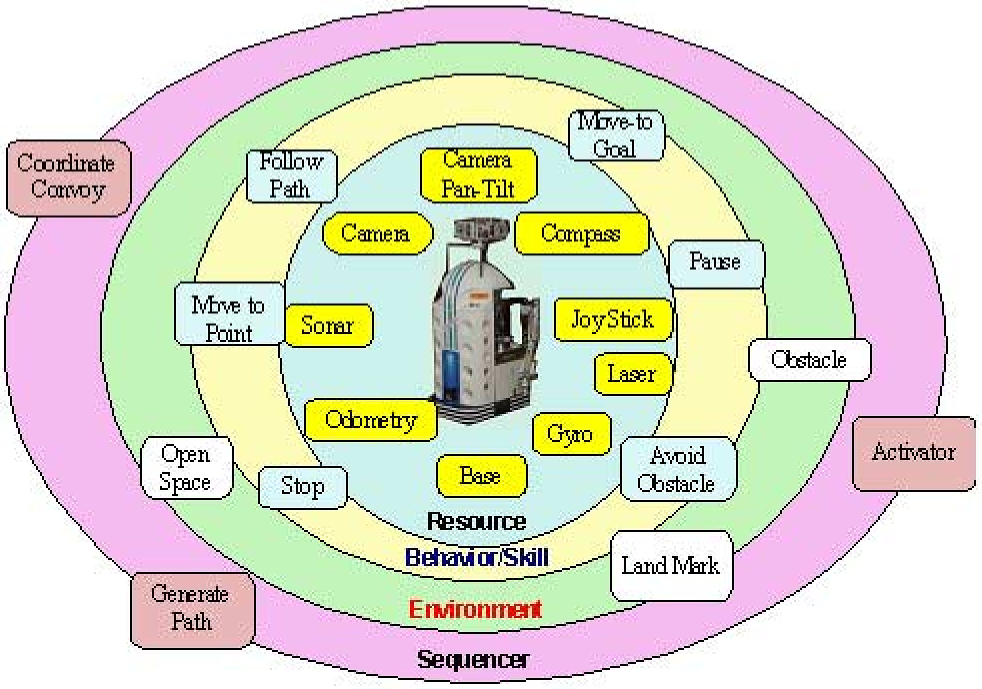


Figure 2: Examples of atomic agents that were built for the HelpMate robot

Compound Agent

An IMA Compound Agent is a group of atomic agents that are coordinated or sequenced by one or more sequencer agents. The highest-level sequencer agent can be envisioned as the root node of a tree with connections and dependencies on other agents or branches.

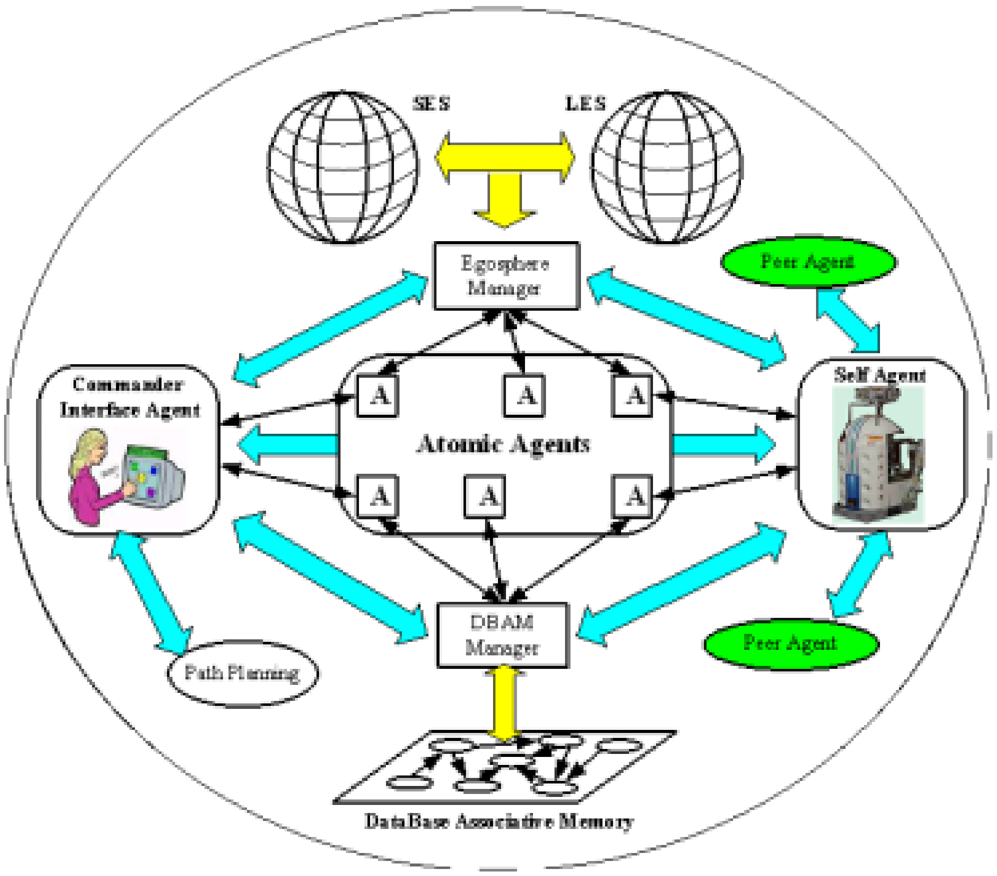


Figure 3 IMA agents and the Self Agent

The Commander Agent

The Commander Agent is an agent that is a communication gateway between the robot and user. The Commander Agent concept has been developed to ease communication and allow the user to access the robot's abilities. The Commander Agent not only receives commands from the user, but also gathers the status of progression of the robot's task and sends it back to the user.

Sensory EgoSphere

The Sensory EgoSphere (SES) is a short-term memory structure that enables a user to view the current sensory information in the view of a geodesic dome [10]. We define the SES as a database, a 2-D spherical data structure, centered on the coordinate frame of the robot. Its implicit topological structure is that of a geodesic dome, each vertex of which is a pointer to a distinct data structure. The sensory information includes visual imagery, sonar and laser signals, gyroscopic vestibular data, speed of each motor, compass heading, GPS position, camera pan and tilt angles, and odometery. The concept of an Egosphere for a robot was first proposed by Albus [11].

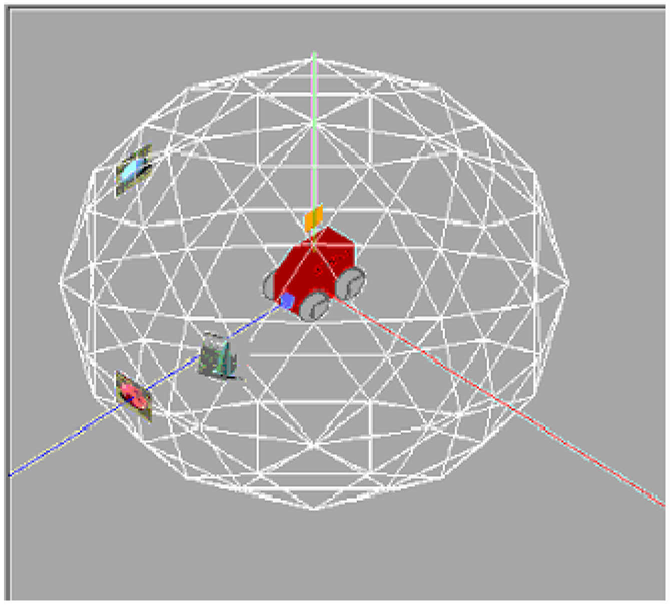


Figure 4: The Sensory EgoSphere of the mobile robot.

Basic Behaviors

A Behavior is a fundamental building block of natural intelligence. It is a mapping of sensory inputs to a pattern of motor actions, which are used to achieve a given task. There are many definitions of a basic behavior:

- Behavior is relatively simple, incrementally added to the system, and not executed in a serial fashion (Mataric)[AB.12]
- A Behavior can be represented as a generated response to a given stimulus computed by a specific behavior (Arkin Stimulus-Response Diagrams) {AB.1]
- Each behavior can take inputs from the robot's sensors (e.g., camera, ultrasound, infra-red, tactile) and/or from other behaviors in the system, and send outputs to the robot's effectors (e.g, wheels, grippers, arm, speech) and/or to other behaviors. Thus, a behavior-based controller is a structured network of such interacting behaviors (Mataric[12])

Behavior can be divided into three broad categories.

1. Reflexive behaviors are stimulus-response (S-R) behaviors, such as when your knee is tapped, it jerks upward. Essentially, reflexive behaviors are “hardwired”. Neural circuits ensure that the stimulus is directly connected to the response in order to produce the fastest response time.
2. Reactive behaviors are learned, and then consolidated to where they can be executed without conscious thought. Any behavior that involves what is referred to in sports as “muscle memory”, is usually a reactive behavior (e.g., riding a bike or skiing). Reactive behaviors can also be changed by conscious thought (e.g., a bicyclist riding over a very narrow bridge might “pay attention” to all the movements).
3. Conscious behaviors are deliberative (assembling a robot kit, stringing together previously developed behaviors, etc.)

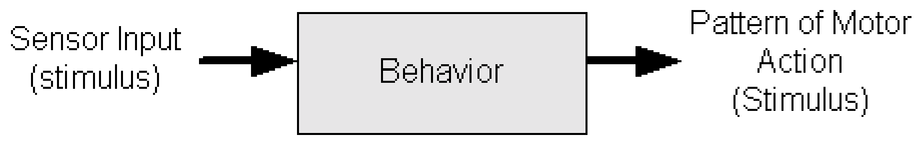


Figure 5: A structure of a simple behavior

Schema Theory

Schema theory provides a helpful way of casting some of the action to the object-oriented programming format. An overview of Schema Theory is:

- Another behavior architecture approach, motivated by the biological sciences.
- Michael Arbib is the first person who first brought this to the serious attention of AI robotists while at the University of Massachusetts.
- Later used extensively by Arkin and Murphy for mobile robotics.
- Consist both of the knowledge of how to act and / or perceive (knowledge, data structures, models) as well as the computational process (the algorithm).
- It is also a generic template for doing some activity, like riding a bicycle.
- It is a template because a person can ride different types of bicycles without needing to learn how to ride again.

The behavior schemas are composed of Motor Schemas and Perceptual Schemas.

- A Motor Schema represents the template for the physical activity.
- A Perceptual Schema provides the environmental information specific (sensor) for that particular behavior.

In advanced applications, the agent may have a choice of either perceptual or motor schemas to tailor its behavior. For example, a person usually uses vision (perceptual schema) to navigate out of a room (motor schema). But if the power is off, the person can use touch (an alternate perceptual schema) to feel their way out of a dark room. In this case the schema uses perceptual schemas differently for different environmental conditions.

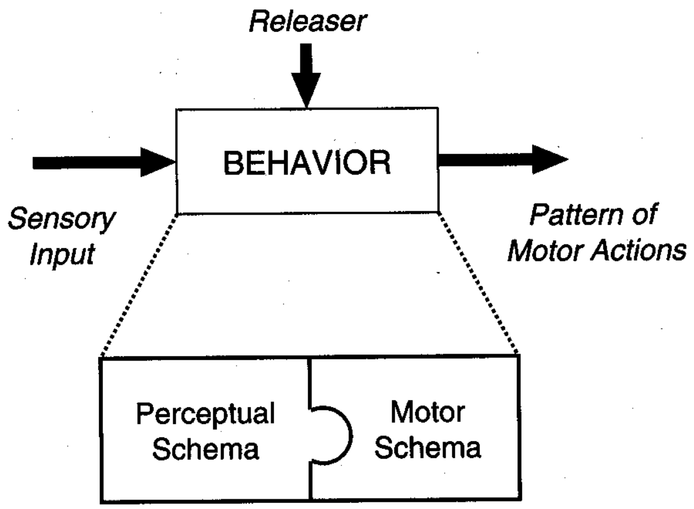


Figure 6: Behaviors decomposed into perceptual and motor schemas

Schemas have some basis in biology. For instance, the Rana Computatrix model explained what occasionally happens when a toad sees two flies at once.

A toad can be characterized as responding visually to either

- small, moving objects (Feeding Behavior).
- large, moving objects (Fleeing Behavior).

In the case that the toad see two flies (two vectors), the Rana computatrix model sums the two vectors, resulting in a third vector.

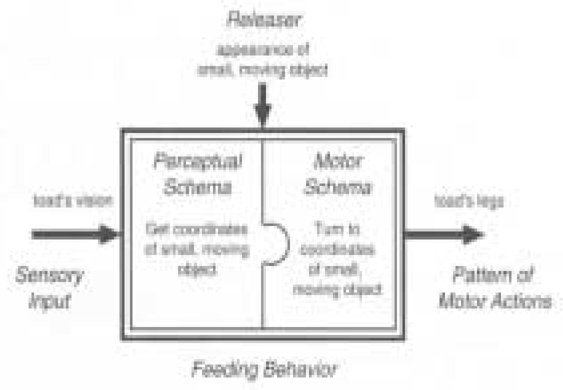

Releaser
appearance of small, moving object
Perceptual Schema
Get coordinates of small, moving object
Motor Schema
Turn to coordinates of small, moving object
toad's vision
Sensory Input
toad's legs
Pattern of Motor Actions
Feeding Behavior

Figure 7 Shows the Feeding Behavior.

Schema theory for autonomous robotics.

Many different motor schemas have been defined, including.

- Move-ahead: move in a particular compass direction.
- Move-to-goal: move towards a detected goal object.
- Avoid-static-obstacle: move away from barriers.
- Dodge: sidestep an approaching ballistic projectile.
- Escape (Hiding): move away from an approaching predator.
- Stay-on-path: move toward the center of a path, road or hallway.
- Noise (Wander): move in a random direction for a certain amount of time.
- Follow-the-leader: move to a particular location displaced somewhat from a possibly moving object.
- Probe: move toward open areas.
- Dock: approach an object from a particular direction.
- Avoid-past: move away from areas recently visited.
- Move-up, move-down, maintain-altitude: move upward or downward or follow an isocontour in rough terrain.
- Teleautomony: allows a human operator to provide internal bias to the control system at the same level as another schema.

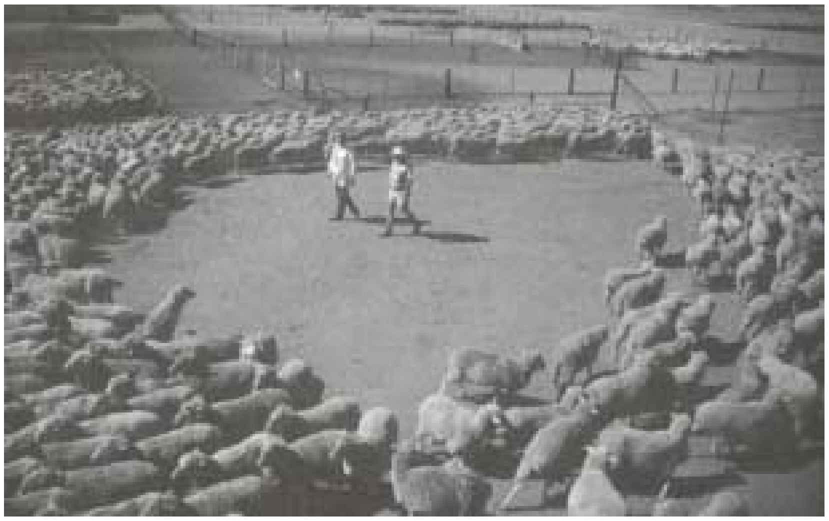


Figure 8: Example of the avoid Behavior in animals

Vision based Navigation

Vision based positioning or localization uses the same basic principles of landmark-based and map-based positioning but relies on video sensors rather than ultrasound, dead-reckoning and inertial sensors.

Many techniques have been suggested for localization using vision information, the main components of which are listed below:

- Representations of the environment.
- Sensing models.
- Localization algorithms.

The environment is perceived in the form of geometric information such as landmarks, object models and maps in two or three dimensions. Localization then depends on the following two inter-related considerations:

- A vision sensor (or multiple vision sensors) should capture image features or regions that match the landmarks or maps.
- Landmarks, object models and maps should provide necessary spatial information that is easy to be sensed.

The primary techniques used in vision-based positioning to date are:

- Landmark-Based Positioning
- Model-Based Approaches

- Feature-Based Visual Map Building

Although it seems to be a good idea to combine vision-based techniques with methods using dead reckoning, ultrasonic and laser-based sensors, applications in realistic conditions are still difficult.

Clearly, vision-based positioning is directly related to most computer vision methods, especially object recognition. So as research in this area progresses, the results can be applied to vision-based positioning.

Real world applications in most research demands very detailed sensor information to provide the robot with good environment capabilities. Visual sensing can provide the robot with an incredible amount of information about its environment. Visual sensors are potentially the most powerful source of information among all the sensors used on robots to date. Hence, at present, it seems that high-resolution optical sensors provide the greatest information for mobile robot positioning and navigation. So that it is the sensor that we want to use.

Summary

This chapter introduced us to the intelligent robotics system. We have briefly described the architecture of an agent-based system and inter-agent communications. The intelligent robot control system in this study will use a behavior-based approach.

Since we are concentrating on a mobile robot, schema theory has been presented. It is important for implementation to have basic behavior schema. The intelligent machine architecture (IMA), an agent-based software system for robot development, was also presented. The IMA has been used in several examples systems that have been successfully implemented in the past several years. The basic behaviors, which will be used in mobile robot navigation, will be discussed in the next chapter.

The basic idea is to help the robot become more intelligent, therefore we have tried to find algorithms that enables the robot to think and learn in some way similar to a human. That is why we developed a robot self agent. It is a compound agent that has the ability to learn, make decisions, and communicate with humans and other robots.

Many of the basic behaviors that we developed and tested were the vision based behaviors. Although, the vision sensor is very noisy and requires a lot of processing time, it is also perhaps the best sensor that a human has and mainly uses.

CHAPTER III

CONCEPT OF THE ROBOT SELF AGENT

Introduction

In the real world, we need robots which have the ability to understand what a human says. Additionally, robots should be able to provide information that is easy for a human to understand. This includes status, performance, error, etc. Therefore we developed the robot Self-Agent to provide these abilities. The Self-Agent is a compound agent, which is designed for a multi-agent software system under Intelligent Machine Architecture (IMA). The main purpose of this agent is to be the center of the robot. It provides the basic information that a robot can give to a human (or other robots) such as translating high-level communications to basic commands that robots can understand; asking an atomic agent inside the robot to activate or deactivate; and monitoring the performance of the robot itself; etc. The Self-Agent is composed of six atomic agents that are listed below:

1. Command I/O Agent
2. Activator Agent
3. Affect Agent
4. Info Agent
5. Description Agent
6. Path planning Agent

Each of these agents and their functions will be described in more detail in the next section. Since in this research we will emphasize the mobile robot, we also add a Path planning agent into the self-agent. Path planning is an essential function that every mobile robot needs to have.

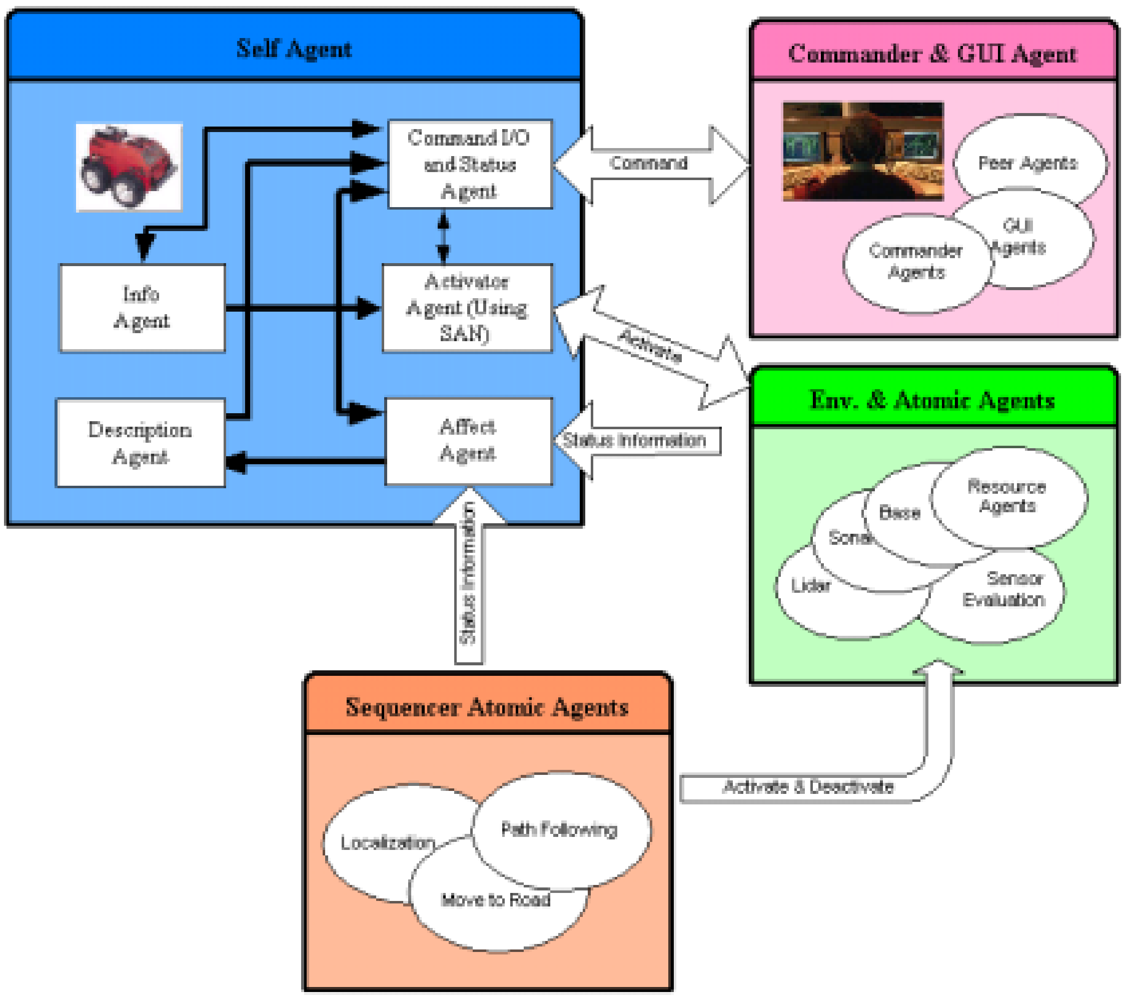


Figure 9: Data flow among the agents, with an emphasis on communication with the Self Agent.

The Command I/O and Status Agents communicate with the Commander Agent. After receiving a high-level command, the Command I/O and Status Agents translate the command into executable commands. The Activator Agent sends the command to activate the necessary atomic agents. Finally, the Affect Agent gets status information from the atomic agents in the robot, and then sends their status to the Description Agent to collect and provide this information for feedback to the human.

What is the Self-agent?

The Self-Agent is a compound agent that is used to activate and monitor the atomic agents until the goal is achieved. The Self-agent also contains what might be called the "brain" of the robot. It is responsible for high level planning and reasoning. It contains the basic information of the robot. This includes long-term and short-term information such as the information about an atomic agent inside the robot; status and performance of an atomic agent; the collection of functionalities of the robot; etc. It also provides the interface for the user or other robots to communicate with an atomic agent inside the robot.

Furthermore, the robot Self-Agent acts as part of the deliberative architecture. In this issue, the Self-Agent must be able to receive commands from the user then select actions. In other words, the user determines the goals and the Self-Agent will activate the proper agents to satisfy those goals. The Self-Agent can provide feedback to the human when the robot faces a problem or cannot achieve its goals. Hence, it requires the ability to detect problems, and to send feedback and a description of the state of the robot to the user.

The Parts of the Self Agent.

The Self Agent consists of six parts:

- Command I/O agent: Communicates with the commander and other agents.
- Activator Agent: Selects the appropriate atomic agent depending on the situation.
- Affect Agent: Detects failure of atomic agent and report to commander agent.
- Info Agent: Stores information about all atomic agents that are used by the robot.
- Description Agent: Provides the current status of the robot's atomic agents.
- Path Planning agent: Generates the best path for the robot.

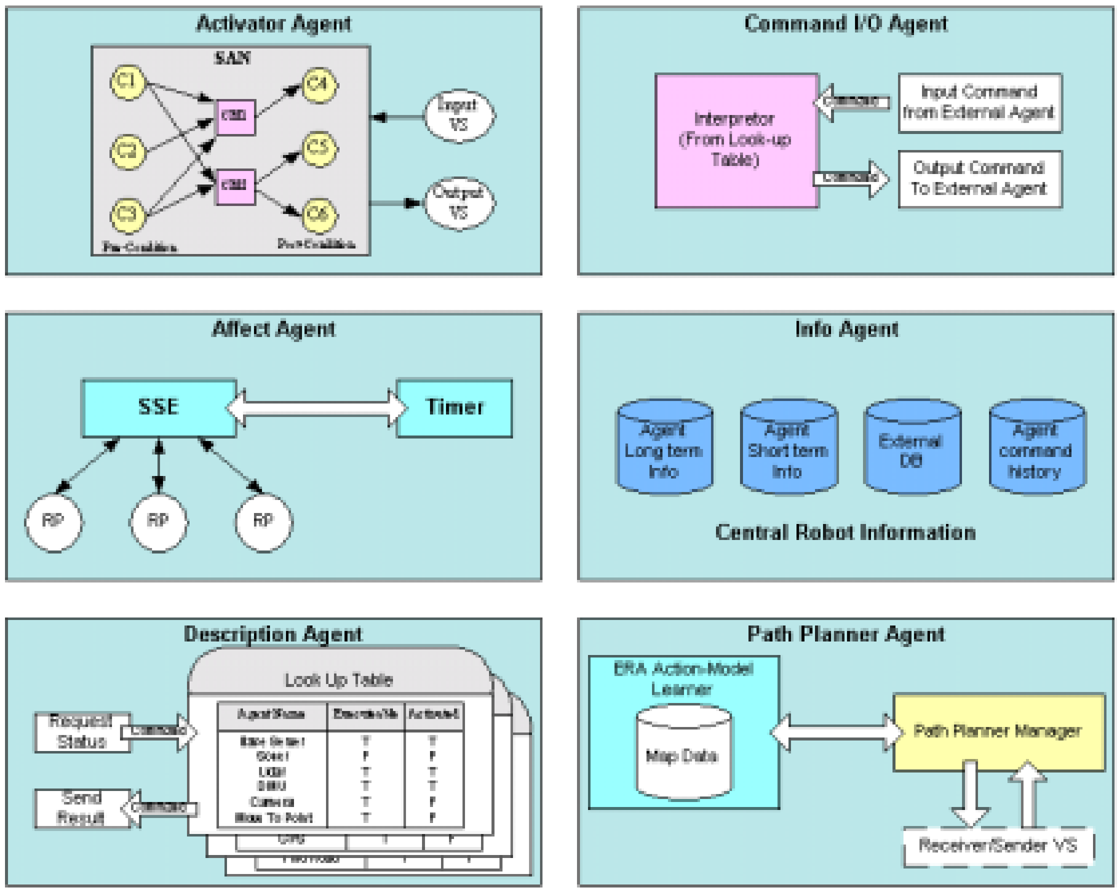


Figure 10 Shows part of the robot Self Agent

Command I/O Agent

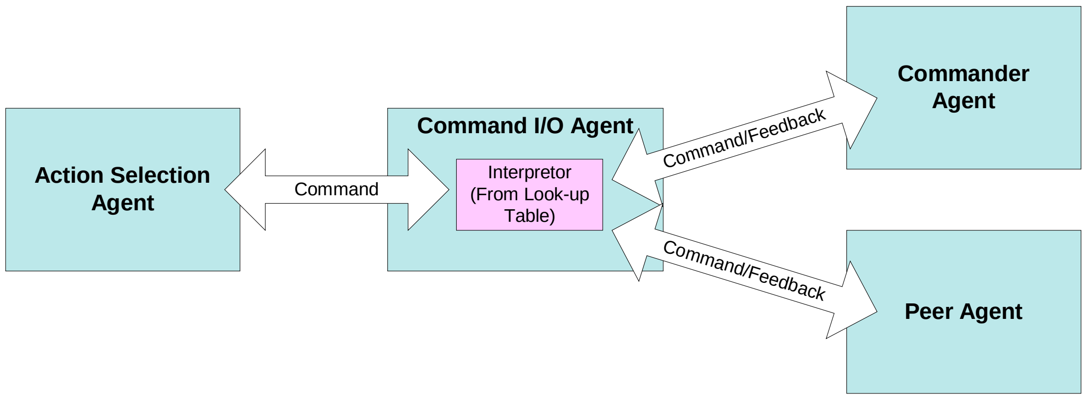


Figure 11 shows communication between command I/O agent and others

The command I/O agent gets input command from an external agent such as the Commander Agent, Peer agent or directly from the user (via a Representation manager), etc. It then processes the command and sends the commands to the Activator Agent in order to command the robot to follow the given tasks. If the receiving command needs to be translated, it will search from a look-up table. In the case that it cannot interpret the command, it will send feedback to the external agent that there is something wrong.

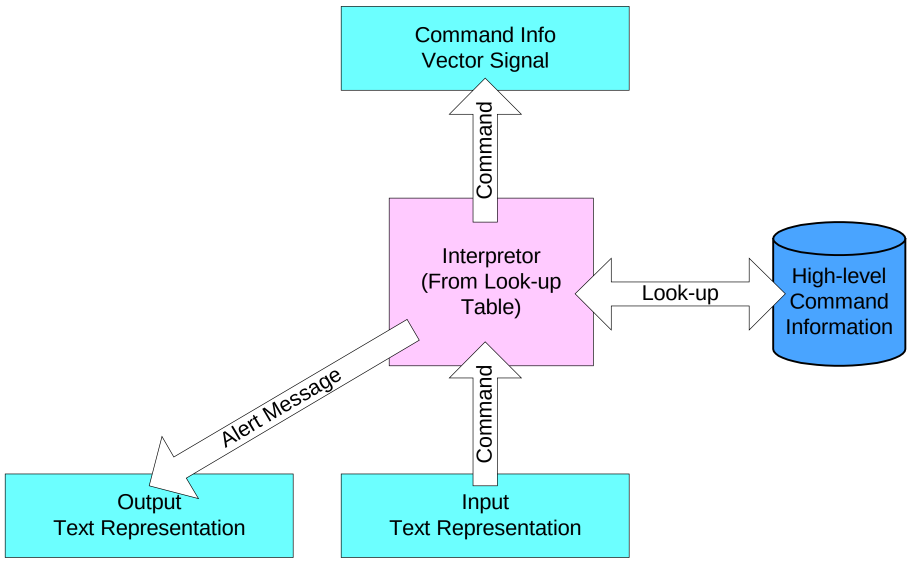


Figure 12 shows detail of the Command I/O agent

After the Command I/O agent gets the command from user (using the Text representation). It searches for the given command in a look-up table in the Info agent. Then it sends the lower level interpreted commands to the Activator Agent. There are two outputs from this agent, to the Activator Agent (using Vector Signal) and to the user (Text Representation).

Activator Agent

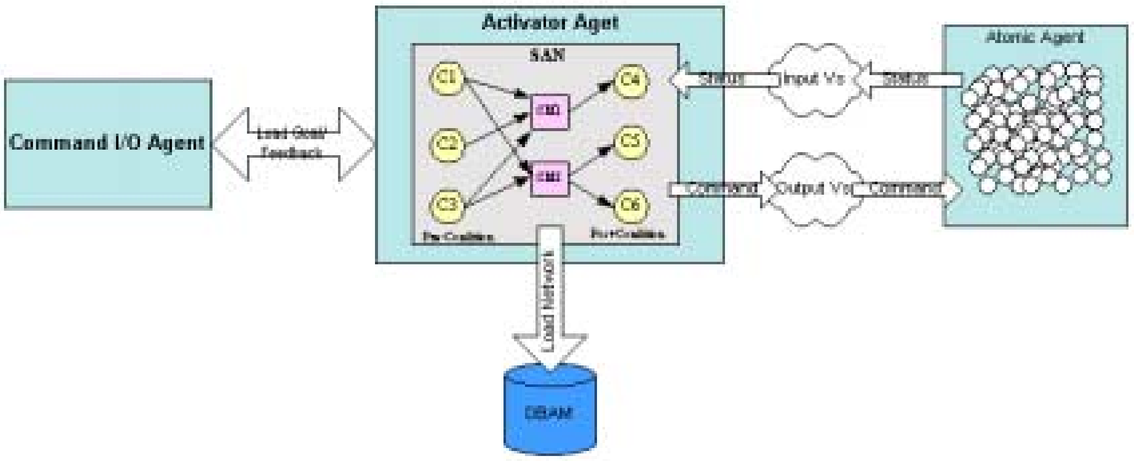


Figure 13 Shows the communication between the Activator Agent and others

The Activator Agent provides the abilities to control the robot from the given command. The Activator Agent selects action, choosing the atomic agent having the highest activation in the spreading activation network. The details of how the Spreading Activation Network works will be described in the next chapter.

The functions of Activator Agent are listed below:

- Use a Spreading Activation network (SAN) to select the next action for the robot using Patties Mae's approach.
- After the action has been completed, then its post condition in the spreading activation network will become true.
- Select goal of the command by setting their priorities.
- Add the reinforcement learning into Spreading Activation Network to allow learning a policy.
- Store network information into a DBAM.

There is an agent called the Activator link. It consists of two vector single agents, Input Link agent and Output Link agent. The activator link is an interface between the Atomic agent and the Self agent to update the status of their agent and then select the best possible behavior to complete a given task.

Affect Agent

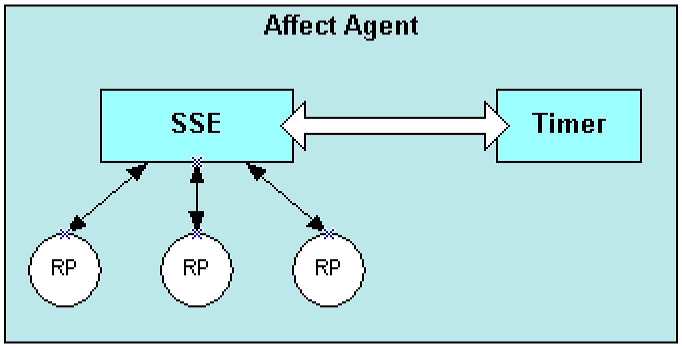


Figure 14: Communication between the Affect Agent and others

System Status Evaluation (SSE)

- If an agent is unable to meet its goals or is experiencing abnormal communication with another agent, this will indicate a problem
- An agent's failure to meet its goals will cause abnormal communication to propagate upward in the hierarchy of Atomic agent.
- The detection of abnormal communication with another agents is based on communication timing patterns
- Using the data from a histogram of the timing patterns, each agent will classify the status of its communication with other agents as normal or abnormal

Performance Evaluation (PE)

For basic Performance Evaluation (P.E.) this system will use a timer to determine the performance in a very simple way. After the time is up, the performance measure of the agent will decrease by a specific value (5% is the default). On the other hand, if the agent does not time out, the performance value will also increase by the same value .

The following is an expression that shows how to calculate PE.

```
{
        if time is up then
                P.E.  = P.E.(old value) – Step; If not less than 0
        Else
                P.E.  = P.E.  (Old value) + Step; If not more than 100
        End if
    }
```

Info Agent

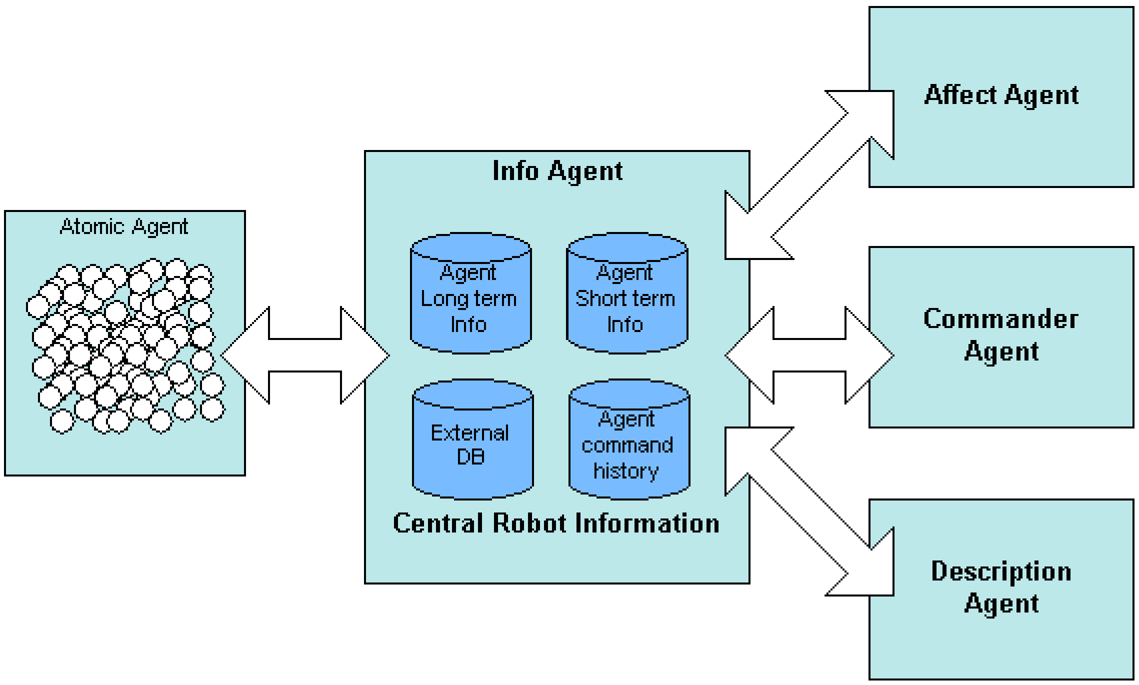


Figure 15: Communication between the Info Agent and other agents.

This agent is responsible for collecting long-term information of the robot and sharing this information with other agents. For instance, the Agent Name and Id (for referencing the agent), the information of the agents that can be used by the robot, the Agent path address (for activating the agent) and the history of the robot performance are a few examples.

Description Agent

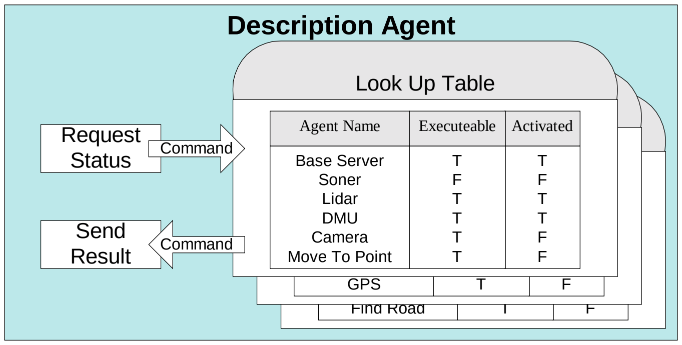
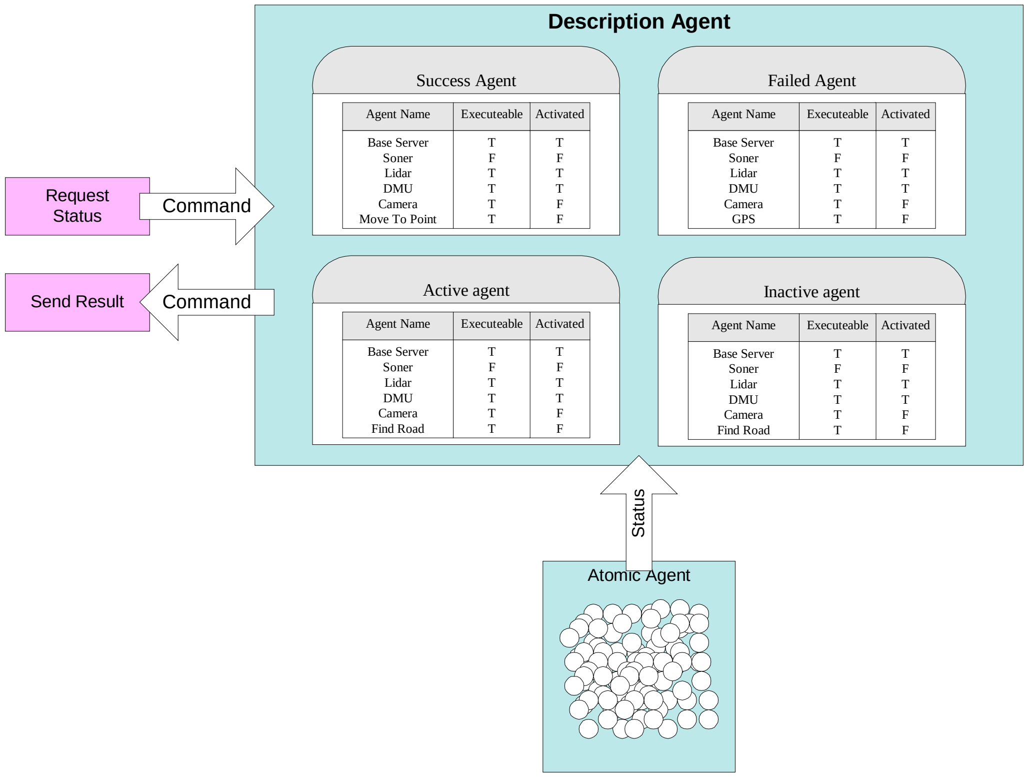


Figure 16: The description agent.

The Description Agent is responsible for creating the descriptions for the user. This description is based on information that registers with the Self-Agent. The Description Agent returns the status of the agents such as active or inactive; ready or not ready to execute; in progress, success or failure; etc. The Description Agent produces text representations containing the names and descriptions of all atomic agents, which are registered, and then sends it back to the requester agent.



Success Agent

Agent Name	Executeable	Activated
Base Server	T	T
Soner	F	F
Lidar	T	T
DMU	T	T
Camera	T	F
Move To Point	T	F

Failed Agent

Agent Name	Executeable	Activated
Base Server	T	T
Soner	F	F
Lidar	T	T
DMU	T	T
Camera	T	F
GPS	T	F

Active agent

Agent Name	Executeable	Activated
Base Server	T	T
Soner	F	F
Lidar	T	T
DMU	T	T
Camera	T	F
Find Road	T	F

Inactive agent

Agent Name	Executeable	Activated
Base Server	T	T
Soner	F	F
Lidar	T	T
DMU	T	T
Camera	T	F
Find Road	T	F

Figure 17: The Description agent

There is a description link agent that allows the atomic agents to contribute information such as its name, a description of its purpose, its status, etc. These texts will be used to provide the description for the users.

Path Planning Agent

One of the most well-studied mobile robot tasks is Path Planning. Path planning is important for a mobile robot. Finding the safest, shortest, and most efficient paths in the presence of obstacles and other robots can be difficult. The Path planning cost is dependent upon considerations suck as the path length, number of abrupt trajectory turns, and safety costs.

Path planning is provided for autonomous or semi-autonomous robots operating in either an indoor or outdoor environment. We propose the way to ensure optimality is by a preprocessing analysis of the terrain before sending the robot out. This approach uses pre-processing maps that show the characteristics of the terrain in the region. The Pre-processing map assigns descriptions to every point in that region. The descriptions should be simple enough that the optimal direction at any point can be quickly calculated in real time. Then, a robot needs only to determine where it is periodically, and look up this position in the map. Then this algorithm gives a planning path through areas of terrain, allowing a robot to explore optimal paths safely.

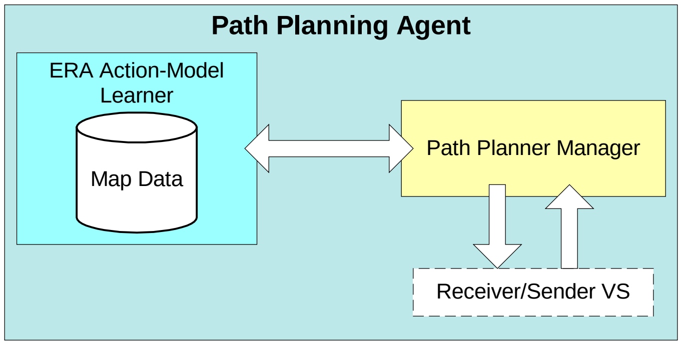


Figure 18 Path Planning Agent

The Path Planning Agent will get the start and target positions from other agents. Then it returns the set of way-points that the robot should follow. Before we can use it we need to have a - map of the area.

CHAPTER IV

SPREADING ACTIVATION NETWORK WITH REINFORCEMENT LEARNING

Introduction

Our approach consists of two algorithms. The first algorithm is the Spreading Activation Network (SAN) that provides the basis for selecting appropriate behaviors (i.e., action selection) for completing a given task. To perform well, the parameters of the Spreading Activation Network must be manually tuned. The second algorithm is a Reinforcement Learning (RL) technique that enables a robot to learn a policy automatically.

A Spreading Activation Network attempts to achieve a number of goals in a complex dynamic environment [12], which is too complicated to be predictable and the agent has limited computational and time resources. We need to have action selection that will make a good decision. It not only favors actions that contribute to several goals at once, but also contributes to following the plans that are given by the user or by other agents. Additionally it provides the capability of both deliberative and reactive planning.

In the real world we always have different preferences in how a robot should finish its tasks. An example could be a robot that has to explore Mars and collect samples of rocks along the way. Under normal situations, the robot will collect rock while it is moving along the given path. However, if it is in a hazardous situation, our preferences will need to be changed. We may want the robot to come back to the mission base as soon as it can and neglect its rock collecting duty.

In this approach which is called SAN-RL, it enables the learning of different configurable behaviors. We want it to learn behaviors from the current preferences for completing a given task. Our solution is to combine two techniques a Spreading Activation Network (SAN) and Reinforcement learning (RL). The Spreading Activation Network (SAN) allows the robot to select plans to achieve goals. However, setting these parameters for the network could be difficult and require a lot of experiment time. Therefore, we integrate the Spreading Activation Network (SAN) with a Reinforcement Learning (RL) technique that can learn the appropriate parameters, dependant upon the reward, which the robot gets when it completes its tasks.

Reinforcement learning techniques have been successful in allowing an agent to learn a policy for achieving tasks. The overall behavior of the agent can be controlled with an appropriate reward function that it receives based on the sequence of actions that it takes. The objective of the learner is to find the best policy that will maximize the expected reward.

Spreading Activation Network

What is SAN?

A Spreading Activation Network (SAN) is a technique that is derived from the learning processes that take place to form learning networks [13]. The origins of the Spreading Activation Network can be traced back to the work on associative memory in the late nineteenth century. Artificial Intelligence researchers have emulated this process many ways, and then applied it over the network to solve problems such as in the field of robotics.

In this approach we use a Spreading Activation Network as an action selection mechanism. A Spreading Activation Network consists of competency modules. Each competency module is linked to the condition (Pre-condition) that must be satisfied before the module can be executed. There are conditions that become true after the competency modules have successfully completed. These conditions are called Post-condition. If a competency module has all of its pre-conditions satisfied by the current state, then it is executable. In the case that there is more than one executable competency module, the algorithm selects the executable competency module with the highest activation. If its activation is above the threshold, then it is selected. Otherwise, the threshold is reduced by a pre-determined amount (e.g, -10%) and then the process repeats.

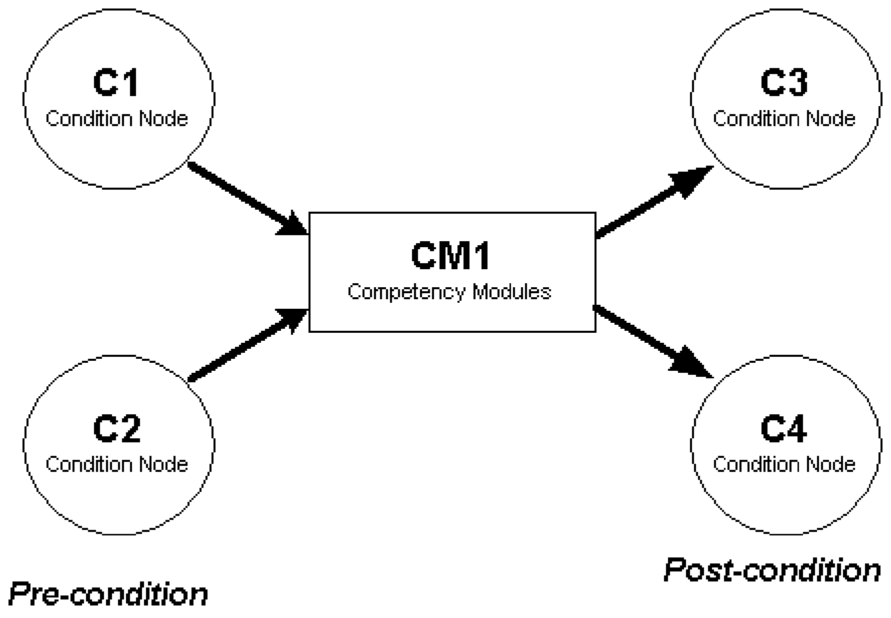


Figure 19 An example of Spreading Activation Network (Boxes are Actions. Circles are Conditions.)

Why SAN?

As robotics and automation application researchers tend to put robots in uncertain environments, we need to have an action selection method that is robust, able to execute a plan and makes a good decision. A good action selection behavior should have the following characteristics:

- Favor actions that contribute to more than one goal at the same time
- Is able to react to the current situation and is highly adaptive to unpredictable and dynamic environments
- Contributes to the plan (unless another action has higher priority). For example, it should keep working towards the same goal unless there is a good reason to switch to a different goal.
- It should look ahead (i.e. it should try to avoid hazardous situations).
- It is robust (recovers itself when certain components fail).
- It should be Reactive and fast.

Algorithm

A competency module is executable at time t when all of its preconditions are true. If an executable competency module is selected, it will perform a real action. A competency module could be described by (c_i,a_i,d_i,α_i). c_i is a list of preconditions. a_i and d_i represent an add list and a delete list of a competency module. The competency module also has a level of activation that we call α_i.

Every competency module is linked in the network through three types of links: Successor links, Predecessor links, and Conflicter links. The description of each link is shown below:

Successor link

If the competency module x (c_x,a_x,d_x,α_x) has the competency module y (x_y,a_y,d_y,α_y) as its successor, for every proposition p that is a member of the add list of x is also a member of the precondition list of y. We can define this as:

For every proposition $p \in a_x \cap c_y$

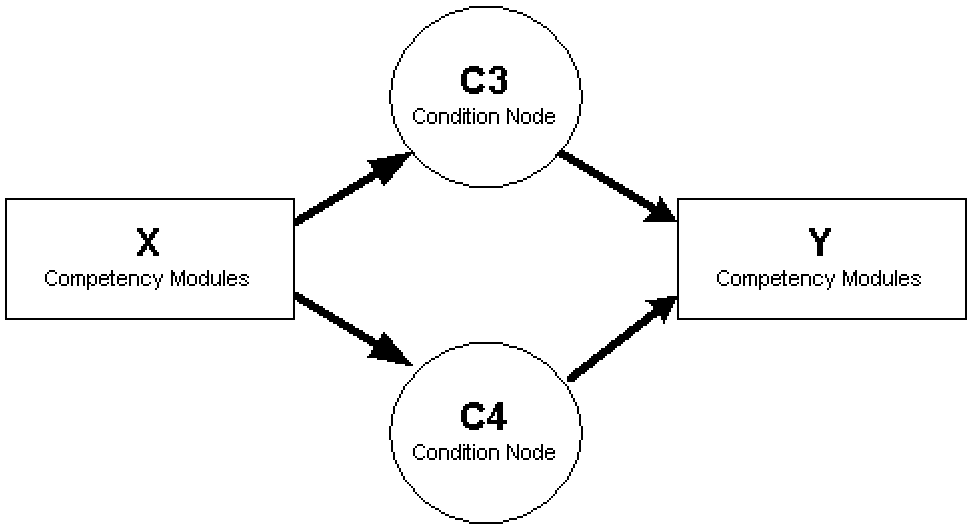


Figure 20 Successor Link

Predecessor link

If the competency module x (c_x,a_x,d_x,α_x) has the competency module y (x_y,a_y,d_y,α_y) as its predecessor, then every proposition p that is a member of the precondition of x is also member of the add list of y. We can define this as:

For every proposition $p \in c_x \cap a_y$

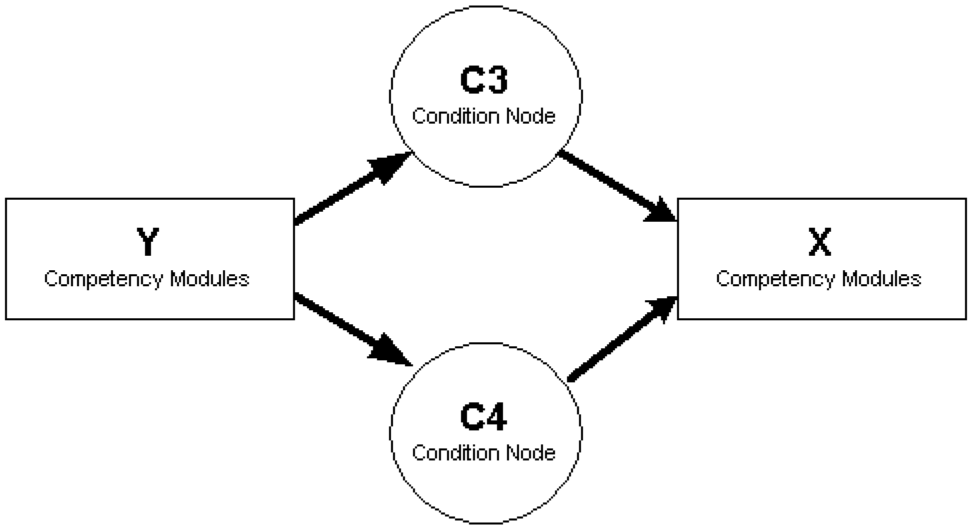


Figure 21 Predecessor link

Conflicter link

If the competency module x (c_x,a_x,d_x,α_x) conflicts with the competency module y (x_y,a_y,d_y,α_y) then every proposition p that is a member of the delete list of y is a member of the precondition list of x. We can define this as:

For every proposition $p \in c_x \cap d_y$

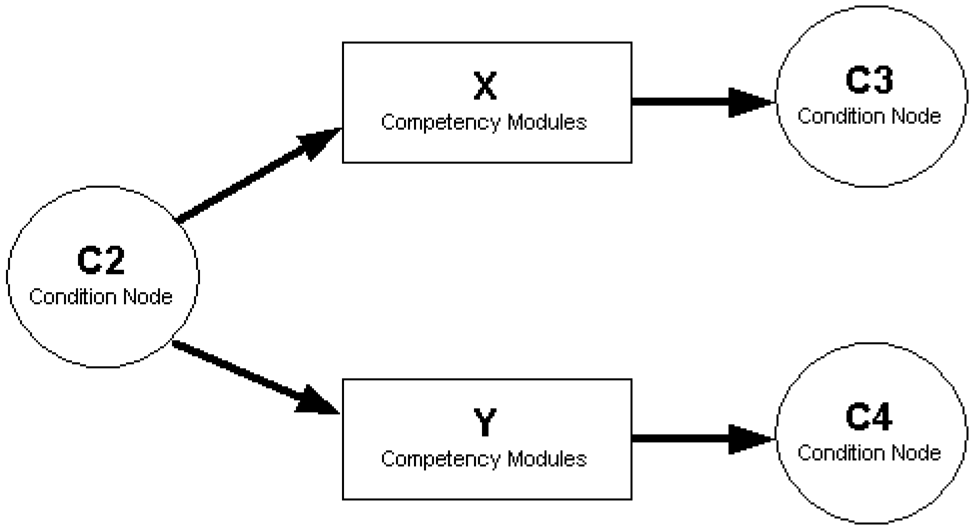


Figure 22 Conflicter link

Since competency modules use these links to activate and inhibit each other, sometimes the activation energy accumulates from the current situation (Input from state) and goals (Input from goal). Once the module is executable only acts if it has highest activation.

Competency modules get their activation value from determining the current state of the environment and the current global goals shown below:

$$\text{Activation} = \text{Input\_from\_state} + \text{Input\_from\_goal} + \text{taken\_away\_by\_protected\_goal} + \text{SpreadBackward} + \text{SpreadForward} + \text{From\_Value\_.Function} \quad (1)$$

The four global parameters can be used to change the spreading activation network dynamically, and the action selection behavior of the robot:

1. Threshold (θ) If its activation is above the θ, then it is selected. Otherwise, the threshold is reduced 10% each time that none of the competency modules are able to be selected. It is reset to its initial value when a module is selected.
2. The amount of activation energy a proposition (ϕ) that is observed to be true injects into the network
3. The amount of activation energy a goal injects into the network. (γ)
4. The amount of activation energy a protected goal takes away from the network. (δ)

5. These parameters also determine the amount of activation that the modules spread forward, backward or take away. These parameters were chosen carefully because the internal spreading of activation will be affected by appropriate tuning.

Activation by the State

There is input activation coming from the state of the environment towards modules that match the current state. A competency module matches the current state if at least one of its preconditions is observed to be true. The activation express is shown below:

$$\text{Input\_from\_state(x,t)} = \sum_j \phi \frac{1}{\#M(j)} \bullet \frac{1}{\#cx} \qquad (2)$$

where ϕ is the amount of activation energy injected by the state per true proposition and M(j) returns the set of modules that match proposition j (condition).

Activation by the Goal

Activation energy is determined by the global goals of the agent. They increase the activation level of modules that achieve one of the global goals. A competency module is used to achieve the global goals if one of the goal is a member of the add list of the competency module. Note that there are two types of goals: first, a goal that has to be achieved only once, which is called a "permanent goal." ,and second, a goal that has to be achieved continuously. An example of the first goal is the 'Reached Target', an example of the second goal is "Obstacle is clear". The expression for how to calculate the activation by goals is shown below:

$$\text{Input\_from\_goal(x,t)} = \sum_j \gamma \frac{1}{\#A(j)} \bullet \frac{1}{\#ax} \qquad (3)$$

where γ is the amount of activation energy injected by the goals per goal and A(j) returns the set of modules that achieve proposition j.

Inhibition from the protected Goals.

The activation should be reduced if it is trying to remove the global goals that have already been achieved. Because these goals should be protected, these ‘protected goals’ remove some of the activation from the modules that would undo them. The expression is shown below:

$$\text{Taken\_away\_by\_protected\_goal}(x,t) = \sum\nolimits_j \delta \frac{1}{\#U(j)} \bullet \frac{1}{\#dx} \tag{4}$$

where δ is the amount of activation energy taken away by the protected goals per protected goal. And U(j) returns the set of modules that undo proposition j.

Activation of Successors

Every “executable” competency module x (c_x,a_x,d_x,α_x) spreads activation forward. It increases the activation level of those successors y for which the shared proposition $p \in a_x \cap c_y$ is not true. We want these successor modules to become more activated because they are “almost executable”. The expression is shown below:

$$\text{SpreadBackward}(x,y,t) = \sum\nolimits_j \alpha_x(t-1)\frac{1}{\#a(j)} \bullet \frac{1}{\#ax} \qquad \text{if executable}(x,t) = 0$$

$$\text{O} = 0 \qquad \text{if executable}(x,t) = 1 \tag{5}$$

where α_x is the amount of activation energy that spreads from the non-executable module to its predecessors and A(j) returns the set of modules that achieve proposition j.

Activation of Predecessors

A competency module x (c_x,a_x,d_x,α_x) that is not executable spreads activation backward. It increases the activation level of predecessor y for which the shared proposition $p \in c_x \cap a_y$ is not true. Competency module x spreads backward through those predecessor links for which the proposition $p \in c_x$ is false. The expression is shown below:

SpreadForward(x,y,t) $= \sum_j \alpha_x(t-1)\frac{\phi}{\gamma} \bullet \frac{1}{\#M(j)} \bullet \frac{1}{\#cy}$ if executable(x,t) = 0

Or = 0 if executable(x,t) = 1 (6)

Where α_x is the amount of activation energy that spreads from a non-executable module to its predecessors and M(j) returns the set of modules that match proposition j (condition).

Inhibition of conflicter

A competency module x (c_x,a_x,d_x,α_x) decreases the activation level of its conflicter y for which the shared proposition $p \in c_x \cap d_y$ is true. We do not allow a module to inhibit itself. Competency module x takes away activation energy through all of its conflicter links for which the proposition that defines then $p \in c_x$ is true, except those links for which there exists an inverse conflicter link that is stronger. The expression will be shown below:

Takes-away(x,y,t) = 0 if $((\alpha_x(t-1) < \alpha_y(t-1)) \wedge (\exists i \in S(t) \cap c_y \cap d_x)$

Or = max{ $\sum_j \alpha_x(t-1)\frac{\phi}{\gamma} \bullet \frac{1}{\#U(j)} \bullet \frac{1}{\#dy}, \alpha_y(t-1))$ } otherwise (7)

where α_x is the amount of activation energy that spreads from a non-executable module to its predecessors ,U(j) returns the set of modules that undo proposition j and S(t) returns the propositions that are observed to be true at time t.

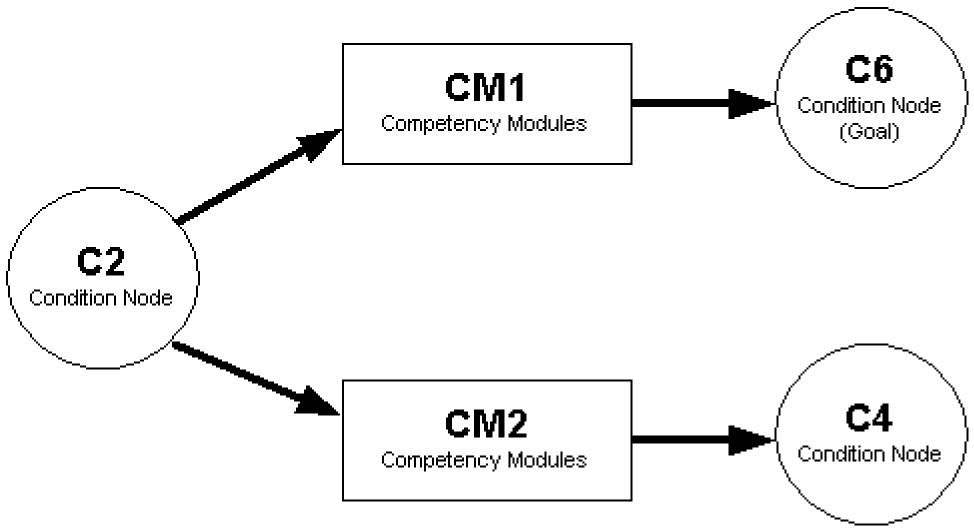


Figure 23 Another example of a spreading Activation Network

For Example, in Figure 5, if condition C_6 is a goal, then competency module CM_1 would receive activation (Activation by the Goals). A competency module receives activation from the current state for each of its pre-condition nodes that are true in the current state. Thus, if condition C_2 is true in the current state, then both competency modules CM_1 and CM_2 would receive activation from the state.

In every loop there are four steps that are repeated continually to keep the system running. These four steps are shown below:

1. Compute the impact of the state goals and protected goals on the activation level of a competency module.
2. Recompute the activation level of the competency modules that are related through its successor links, predecessor links and conflicter links.
3. A decay function ensures that the overall activation level remains constant.
4. The competency module that fulfills the following three conditions become active:
   - It has to be executable
   - Its activation has to be higher then a certain threshold
   - It must have a higher activation than all other competency modules.

When the competency module that fulfills these conditions is chosen, the activation level of the competency module that has become active is reset to 0. If none of the modules fulfill these conditions then the threshold is lowered by 10%

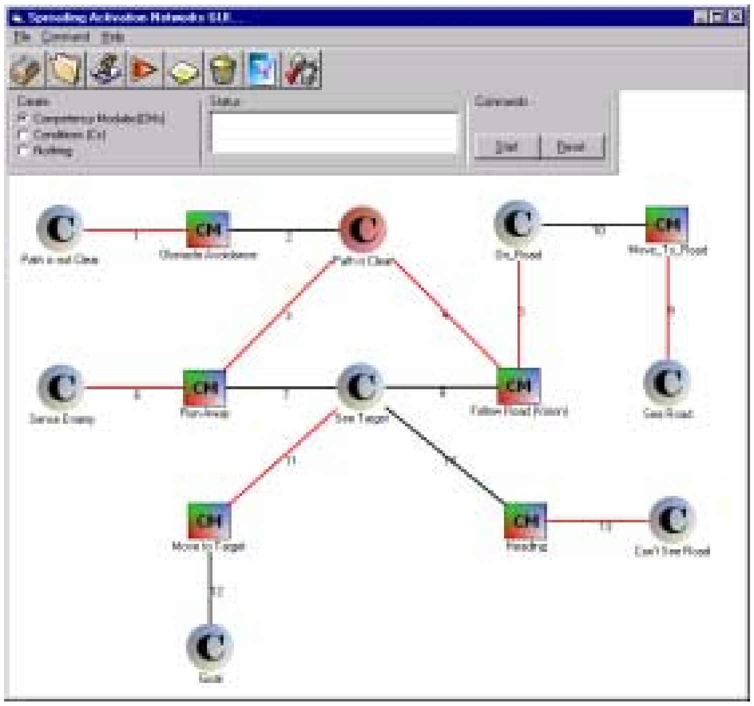


Figure 24 Spreading Activation Network User Interface in IMA Environment

Reinforcement Learning

Reinforcement Learning enables an agent to learn in an interactive environment, typically through a reward system that it receives based on the sequence of actions that it takes and/or the states through which it travels. The objective of the learner is to find a policy that will maximize the expected reward.

Combining spreading activation network with Reinforcement learning

Our objective is to combine spreading activation networks with reinforcement learning to allow a robot to learn a policy for selecting among its competency modules to achieve its goals and to allow a robot to adapt its behavior when priorities among goals are changed without requiring retraining. To achieve this goal we modified the spreading activation network technique to allow a value function to be learned during execution. We also modified the spreading activation algorithm to include activation from the value function and to allow a priority to be associated with each goal.

Modification of the spreading activation algorithm

We need to modify the spreading activation algorithm to include activation from the value function and to allow a priority to be associated with each goal. The modifications are listed below:

Goal Priorities

Goal priorities allow a user to select preferences on how he wants the robot to complete its task. Our objective is that, the robot should try to find a sequence of actions that will maximize the number of the goal priorities that will be achieved through those actions. We modified this by including a weight for each goal. The modification of the algorithm is shown follows:

$$\text{Input\_from\_goal}(x,t) = \sum_j \omega_j \gamma \frac{1}{\# A(j)} \bullet \frac{1}{\# ax} \tag{8}$$

where ω_j is a weight for each goal which is added into the original.

Value function

The original Spreading Activation Network selects the winner from the executable competency module with the highest activation level that exceeds the threshold. Since we want this system to explore the possible actions, we modified this algorithm with an e-greedy policy. With probability 1-e the winning module is activated. Otherwise some other executable action model is selected at random. This will enable this system to activate the modules that may never be selected otherwise.

We modified the expression by including the weight ω_j into the original value function. The original algorithm has constants to indicate how much activation a module can receive from the state, from the goals and from other modules. Therefore we added a new constant that is the amount of activation can be received from the value function.

$$\text{Input\_from\_value\_function}(x,t) = \omega\, Q(s,x) \tag{9}$$

Learning a policy

Before the robot can learn the new policy, we need to provide a reward function. The reward function will be given to the sequence of actions that the robot chooses. The higher the reward, the higher the weight that will apply to its actions.

The reward function that we used was:

$$\text{HealthReward} = \max(-1, 0-\text{health})$$

$$\text{TimeReward} = \max\left(-1, 1-\frac{time}{\min time}\right)$$

$$\text{Reward} = 0.5*\text{healthReward} + 0.5*\text{TimeReward} \text{ (in equal priority)} \quad (10)$$

Where health is a robot's health at the end of the episode, time is the time in milliseconds that it took to reach the target.

In the next chapter we will discuss more about the implementation of SAN-RL on a mobile robot.

CHAPTER V

EXPERIMENTAL RESULTS AND DISCUSSION

Objective of Experiment

This chapter describes the testing of the Spreading Activation Network with reinforcement learning. These experiments demonstrate the effectiveness of the technique in selecting the best possible action that is applied to the mobile robot. In these experiments, we used the mobile robot called Scooter (ATRV-Jr robot) to be our developing model.

We will show that the Spreading Activation Network will select the action with the highest activation. The robot will learn a policy for selecting among its competency modules to achieve its new goals, and adapt its behavior when priorities among goals are changed, without needing retraining.

The experimental tasks can be divided into three main tasks: the initial task, learning task, and run-time task. Firstly, in the initial task, we design the network and setup all parameters and atomic agents. Secondly, in the learning task, the robot will be trained until it knows how to get the highest reward. Finally, in the run-time task we run the demo and change its goal priorities.

Aside from testing the Spreading Activation Network, we are also testing the robot Self-Agent. The user will send his or her command from the Commander Agent and then it will send the command to the Self-Agent. We also show failures can be detected, and how this information can be used to generate feedback to the user about the state of the robot.

<u>Experiment Test bed</u>

The experiments were conducted using the ATRV-Jr mobile robot called “Scooter”. It was develop by IS Robotics, Inc. It came with an onboard CPU, Laser and Sonar sensors, Compass, Drive Motors Unit (DMU), and a wireless Ethernet [14] added the Camera and pan/till head, DGPS, and a laptop.

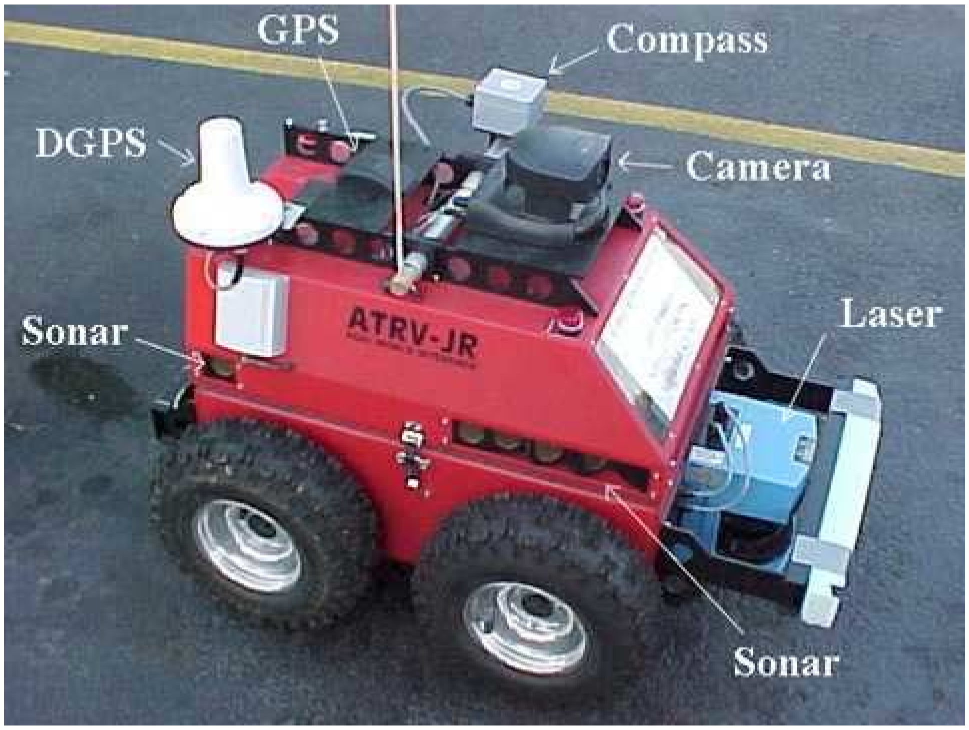


Figure 25 Show Atrv-Jr mobile robot "Scooter "

In the experiments, the user gives commands to the Self Agent by using the commander agent. Then the Self Agent translates and sends them to the Spreading Activation Network. Scooter’s main task is to move to a given target location. Figure 26 shows the atomic agents in the system and the interconnections among them. We will describe each atomic agent later.

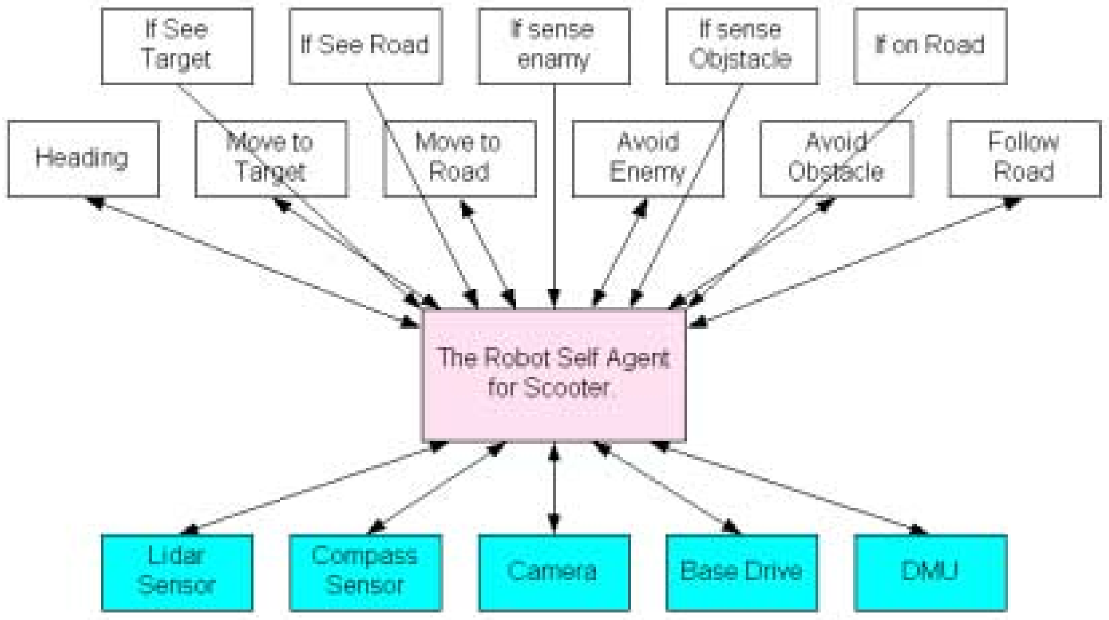


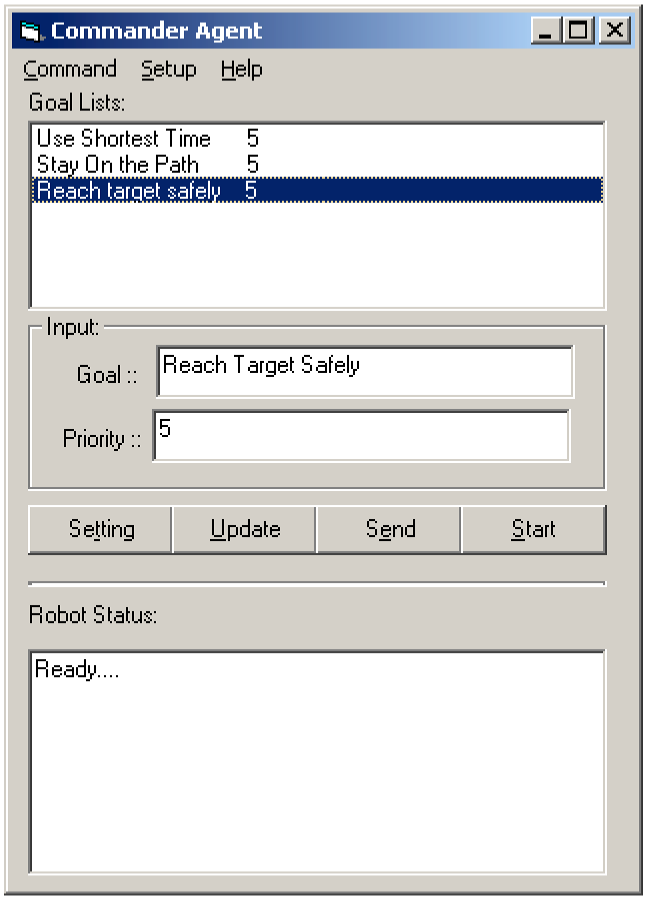
Figure 26 Atomic agents for the Test bed

Table 1 This is a list of the agents that we plan to work with SAN-RL.

Atomic Agents	Sensors	Type
Commander Agent	-	Common Agent
Avoid obstacle	Sonar + Lidar	Behavior/Skill Agent (On Scooter site)
Heading	Compass + DMU	Behavior/Skill Agent (Using compass instead of Odo)
Find enemy	Lidar	Behavior/Skill Agent (Detect Big object as Enemy)

Avoid enemy	Vision, Lidar	Behavior/Skill Agent (Using Lidar to detect enemy)
See road	Vision	Behavior/Skill Agent (In good light condition)
See Target	Vision	Behavior/Skill Agent (Detect orange cone.)
On road	Vision	Behavior/Skill Agent (In good light condition)
Move to road	Vision	Behavior/Skill Agent (There is a pre-program that the robot will turn 90 degree when reach the road)
Follow road	Vision	Behavior/Skill Agent (Using Vision sensor)
Move to target	Vision	Behavior/Skill Agent (Detect orange cone.)

COMMANDER AGENT: is an agent that the user uses to send the command to the robot. The commander agent that we used for this project was designed for this testing only. It does not have the full capability to command the robot. This agent will connect to the Command I/O agent in the Self Agent. Figure 3 Shows the GUI of the commander Agent.



Goal Lists:

```
Use Shortest Time    5
Stay On the Path     5
Reach target safely  5
```

Input:

Goal :: Reach Target Safely

Priority :: 5

Robot Status:

```
Ready....
```

Figure 27 Shows commander agent

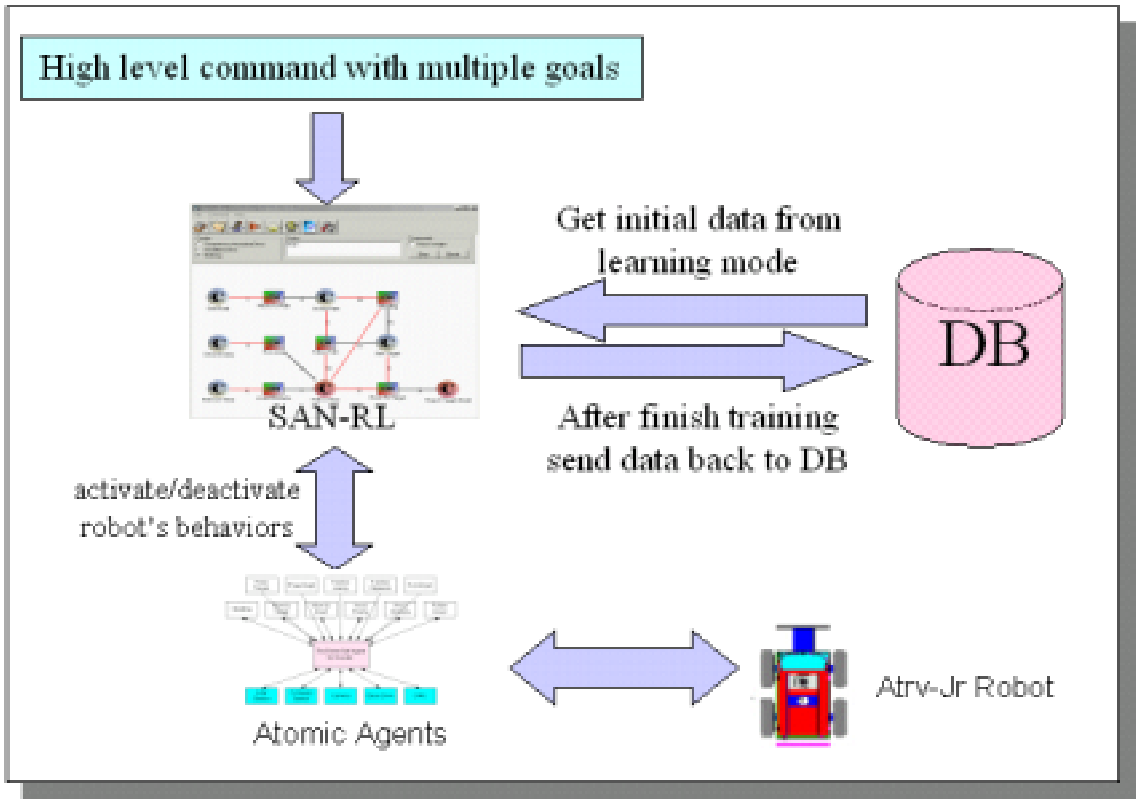

High level command with multiple goals
Get initial data from learning mode
DB
SAN-RL
After finish training send data back to DB
activate/deactivate robot's behaviors
Atomic Agents
Atrv-Jr Robot

Figure 28 Experiment Data flow diagram

Experimental Setup

We evaluated the performance of the approach by applying it to a navigation task. The equipment for this experiment includes:

- A mobile robot (the Atrv-Jr "Scooter")
- An Enemy (the paper "brick wall")
- The target (Orange cone)
- The open outdoor area that is a road in that area

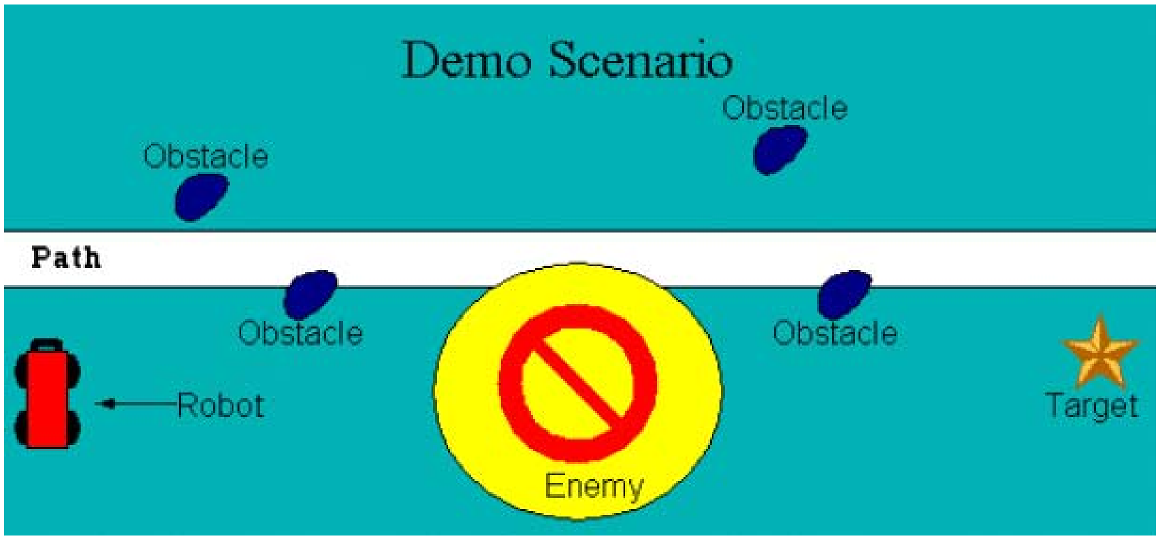


Figure 29 Experimental setup

The human commander will connect to the robot via the commander agent. The Commander Agent will be responsible for displaying the current status of the task to the user and receiving instructions from the commander to send to the robot.

Learning task

Before we let the robot do the experiment, we need to let the robot learn the initial policies. In this task, we set their priorities equally. After training the network in this way, we will change their priorities to see how our systems adapt. The procedure for learning new policies is given below.

1. Pick the action, having the highest activation value, and keep track of the previous action and state in which it was select.
2. After the action completes, record the reward that we get for updating their weight.
3. Wait for the next round, and then pick the next action.
4. Update the Q function for previous action/ state.
5. Make new state and action become the old state and old action.

Figure 30 shows how the system learns and updates the weight values. The system starts from add the goal (with priories) into the network, loads the initial weight of every node in the network then waiting for user to start the process. After the system start, find winner function will be activated. The system selects the winner, which has highest activation value and is higher than the system's threshold. If the winner's activation value is below the system's threshold, no behavior will be activated and the threshold value will be reduced by 10%. Then it will search for a winner again.

After the system activates the winner behavior, the next step is to update the Q function of the previous action and state that robot entered. Finally, we check if the system has achieved its goal. If it has, we calculate the reward, record what behavior that the robot used and then update the reward function.

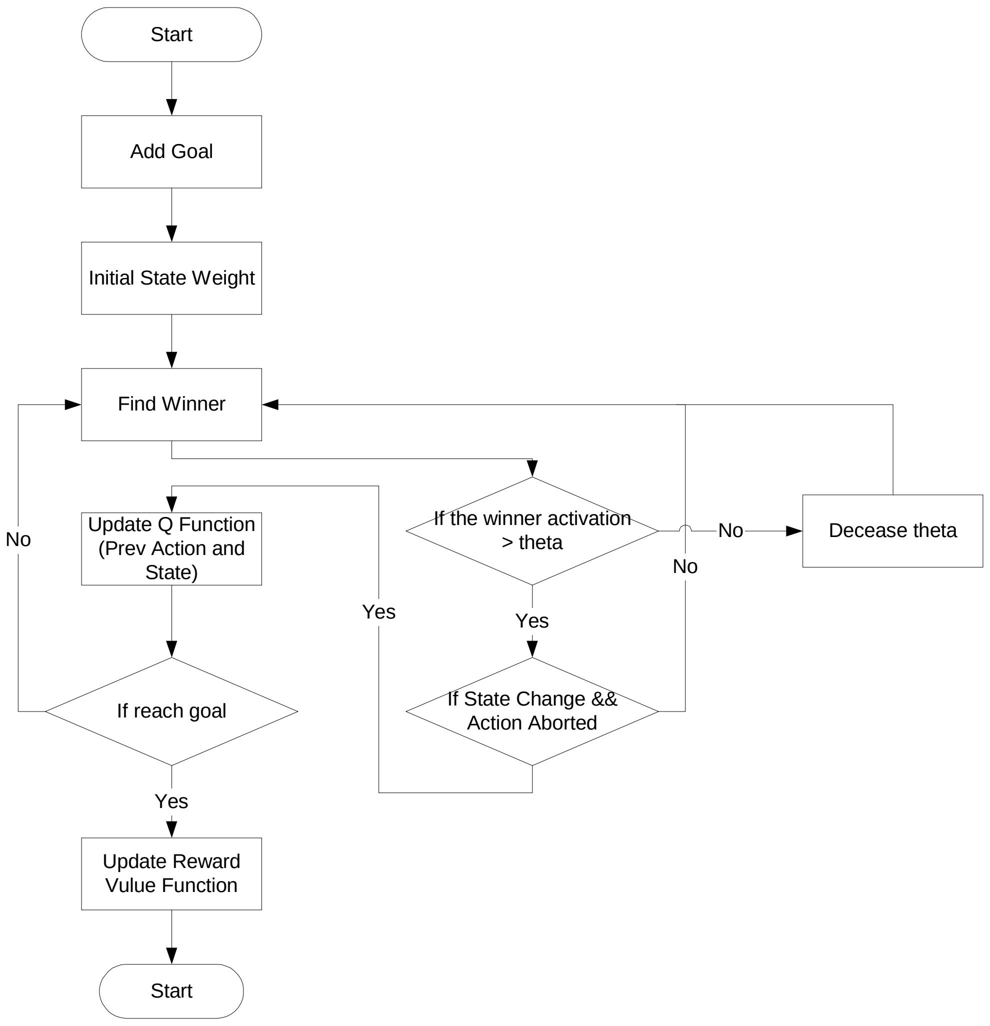

Start
Add Goal
Initial State Weight
Find Winner
Update Q Function (Prev Action and State)
If the winner activation > theta
No
Decease theta
No
Yes
Yes
If reach goal
If State Change && Action Aborted
No
Yes
Update Reward Vulue Function
Start

Figure 30 Show how system learning and selecting behavior

Update State Weight

Update Value = (ValueFunction(w) + alpha * (r-w))

Where alpha is the learning rate in this system, we set it to 0.1; w is the weight of the action; r is the reward value.

Q Update

Update = ValueFunction(s) + alpha * (r + gamma * maxQValue - ValueFunction(s))

Where alpha is the learning rate (in our system we set it to 0.1).

Gamma is the amount of energy injected by a goal (in our system we set it to 45).

MaxQValue is the highest value in given state.

ValueFunction is either from the initial state or from the previous action update it into its record (in Database).

Figure 31 shows the learning simulation program that was used to evaluate the system. This task includes only the avoid enemy and move-to-target behaviors. The robot must get to the target location without being killed by the enemy. The enemy is stationary and fires at the robot at 2-second intervals whenever the robot is within range.

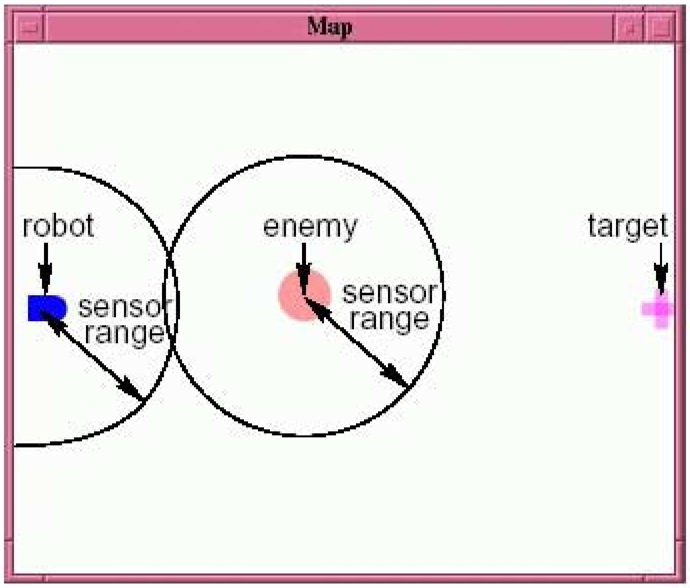


Figure 31 Shows a navigation learning task.

The robot begins with a health of 1 and decreases by 0.1 when it is hit. If the robot's health falls below 0 then it is dead and the episode ends.

Scenario

This demonstration will place the robot at a start position (See Figure 29) with a given target (Orange cone). The robot's mission is to follow the path to the target in the shortest amount of time and have the highest health value. If the robot is taking too much time to reach the goal, getting lost or is hit, then the robot will adapt the policy.

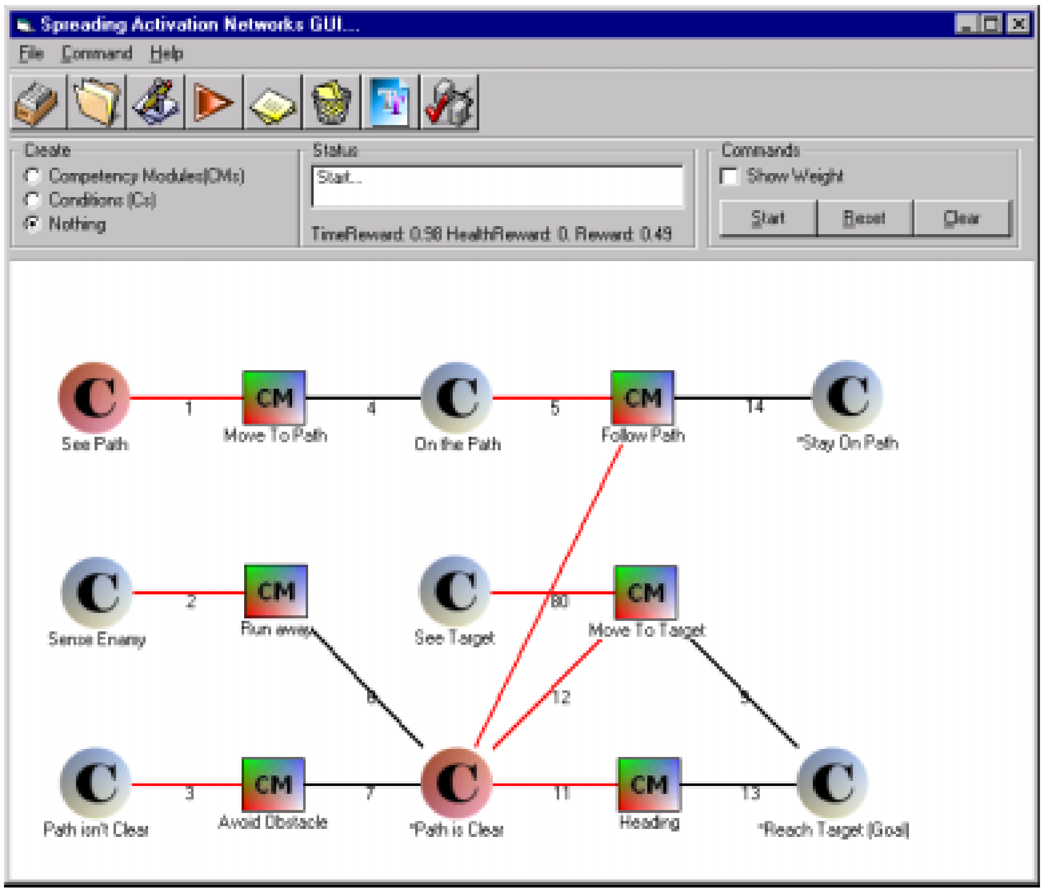
Our reward function depends on two functions: HealthRewardFunction and TimeRewardFunction, and be formulated as:

$$\text{Reward} = 0.5*\text{HealthReward} + 0.5*\text{TimeReward} \qquad (11)$$

In our test bed, we train the robot in simulation under an equal goal priority condition, then we test the robot under different goal priorities such as the following:

- Reach a target safely as well as avoid enemies.
- Reach a target safely.
- Accomplish a task using the shortest time.

How the robot reacts will depend on its priority. For instance, if we emphasize taking the shortest time, the robot will go directly to the goal.


Create: Competency Modules(CMs) Conditions (Cs) [✓ Nothing]

Status: Start...

TimeReward: 0.98 HealthReward: 0. Reward: 0.49

Commands: Show Weight

Figure 32 SAN for the demonstration

Experiment Results

We tested 3 different priority scenarios. As shown in the table below:

Table 2: The priority table of experiments

Scenario Name	Follow the Path	Away from Enemy	Reach Target
Reach a target safely as well as avoid enemies.	0.5	0.5	0.5
Reach a target safely.	0.5	0.9	0.5
Accomplish a task using the shortest time	0.1	0.1	0.9

Scenario 1: Reach a target safely as well as avoid enemies and following the path.

Experiment Setup

- The robot (scooter) is connected to a computer through a wireless network. This computer is a remote operating console using an application called Reflection X.
- A second computer is for the spreading activation network application and atomic agents which will handle the interaction between the user and system.
- The third computer is preparing for setup the client program that connected to the server site in robot site for transferring data.
- The maximum activation threshold is set to 200. In addition, the minimum threshold is set to 15.
- Equal priorities: $\omega_{\text{Reach target}} = 0.5$, $\omega_{\text{Following Path}} = 0.5$, and $\omega_{\text{Avoid enemy}} = 0.5$.

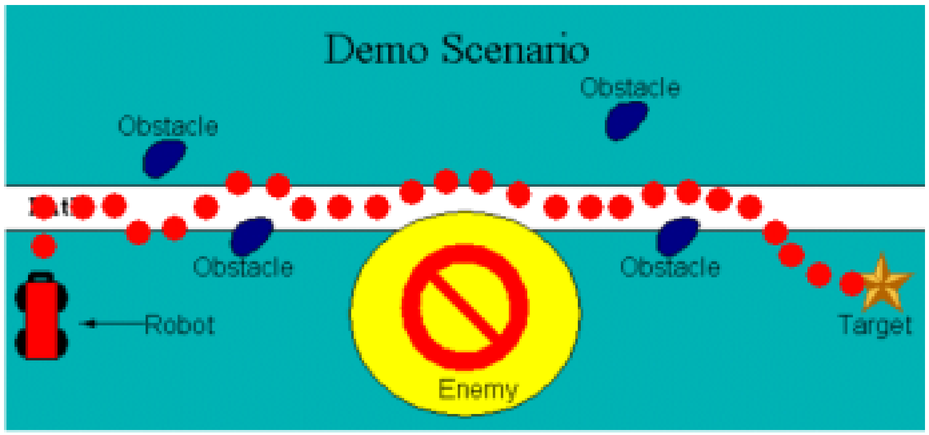


Figure 33 Simulation of the equal priority demo

Discussion of Experiment 1

For the first 30 seconds, robot tried to move to the path. After the robot reached the path, its activation was dramatically reduced. The Follow Path competency module is activated after that. The main reason that the Follow path competency module had a high activation value is due to the input-from-goal from the goal condition called Following the Path. At 41 seconds, the robot detected the enemy. However, in that time the activation of the Run Away is lower than that of Follow Path. The system will continue to select Follow Path until at 45 seconds, Run Away competency module has the highest activation value. At this point, the system activated run away behavior. After the robot moves past the enemy, the activation of Follow Path will be higher than Run Away again and the robot will be following the Path. Finally, the robot finds the target, causing the system to switch to Move to target (at 55 seconds).

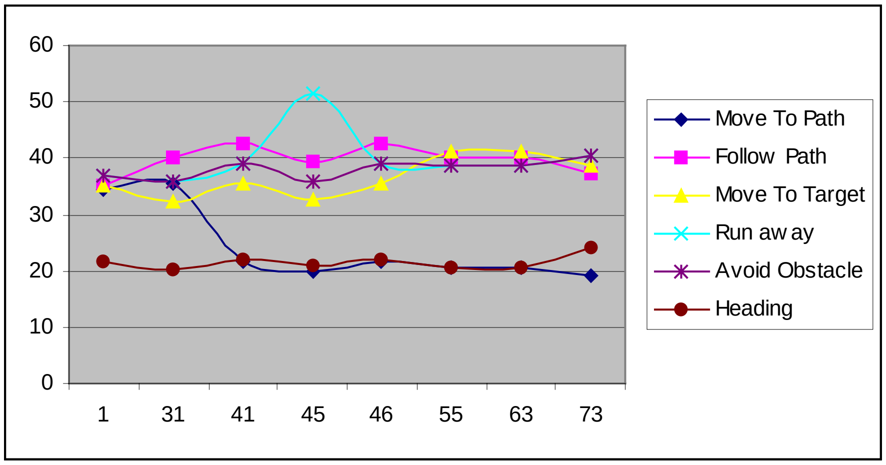
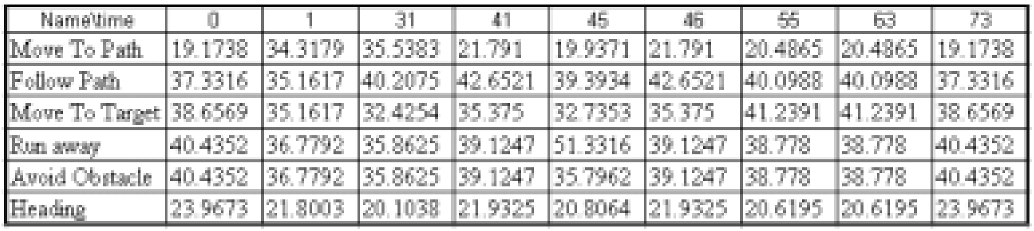


Figure 34 Graph of the result of Scenario 1

Table 3 Numerical results of Scenario 1 (Equal priority)



Name\time	0	1	31	41	45	46	55	63	73
Move To Path	19.1738	34.3179	35.5383	21.791	19.9371	21.791	20.4865	20.4865	19.1738
Follow Path	37.3316	35.1617	40.2075	42.6521	39.3934	42.6521	40.0988	40.0988	37.3316
Move To Target	38.6569	35.1617	32.4254	35.375	32.7353	35.375	41.2391	41.2391	38.6569
Run away	40.4352	36.7792	35.8625	39.1247	51.3316	39.1247	38.778	38.778	40.4352
Avoid Obstacle	40.4352	36.7792	35.8625	39.1247	35.7962	39.1247	38.778	38.778	40.4352
Heading	23.9673	21.8003	20.1038	21.9325	20.8064	21.9325	20.6195	20.6195	23.9673

Scenario 2: Reach the target safely.

Experiment Setup

- The hardware and software setup is the same for scenario 1.
- The maximum activation threshold is set to 200. In addition, the minimum threshold is set to 15.
- Priorities: $\omega_{\text{Reach target}}$ =0.5 , $\omega_{\text{Following Path}}$ =0.5, and $\omega_{\text{Avoid enemy}}$ =0.9.

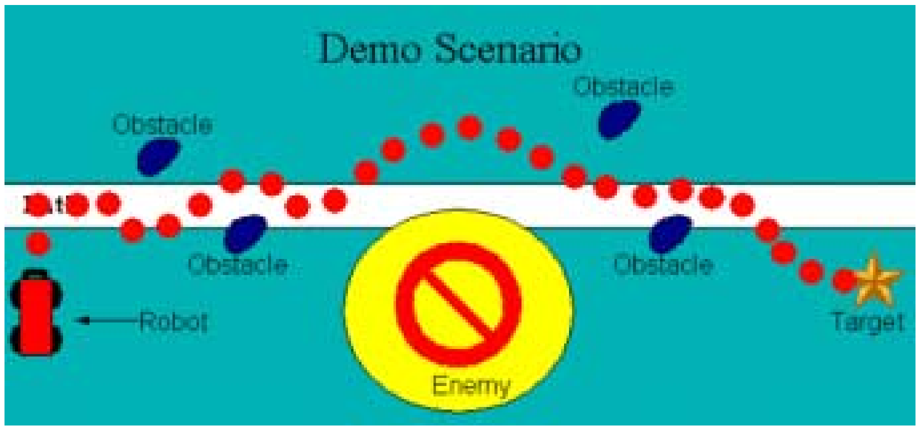


Figure 35: Simulation of the demo that emphasizes avoid enemies

Discussion of Experiment 2

For this scenario the system was set to give the avoid enemy condition high priority. What will happen is whenever the robot finds its enemy, the activation of the Run Away competency will grow very fast and dominated the whole network until it pass the enemy. From Figure 36 we see the Run Away competency was activated very soon (compared to experiment 1) and uses more time to avoid the enemy. Thus, the robot tried to be far from the enemy as much as possible. Then it will continue Follow Path and search for the target. In this scenario the reward function will give a reward, because it spends much more time than in the basic situation of scenario 1.

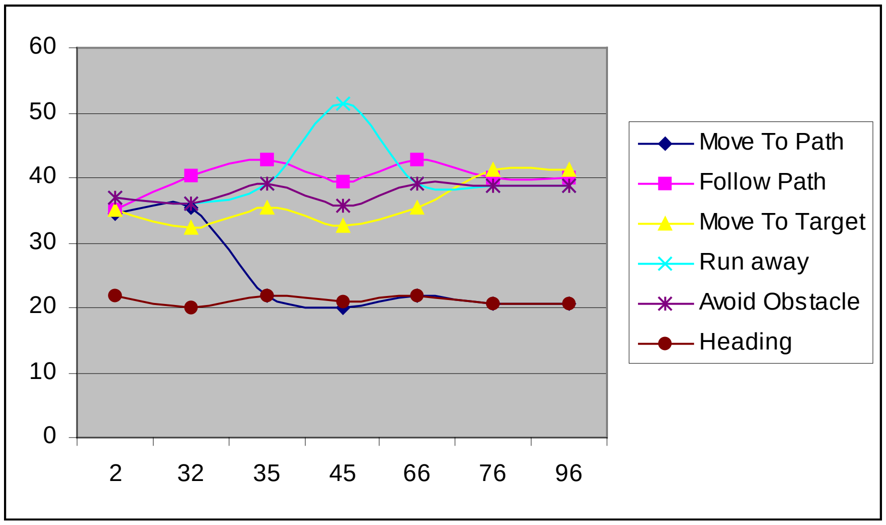
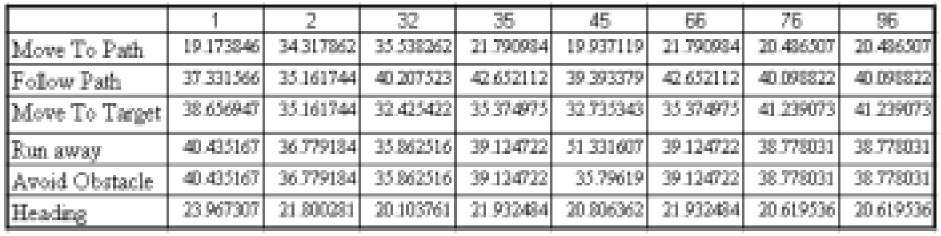


Figure 36 Graph of the result of Scenario 2

Table 4 Numerical results of Scenario 2 (Emphasize on avoid enemy)



	1	2	32	35	45	66	76	96
Move To Path	19.173846	34.317862	35.538262	21.790984	19.937119	21.790984	20.486507	20.486507
Follow Path	37.331566	35.161744	40.207523	42.652112	39.393379	42.652112	40.098822	40.098822
Move To Target	38.656947	35.161744	32.425422	35.374975	32.735343	35.374975	41.239073	41.239073
Run away	40.435167	36.779184	35.862516	39.124722	51.331607	39.124722	38.778031	38.778031
Avoid Obstacle	40.435167	36.779184	35.862516	39.124722	35.79619	39.124722	38.778031	38.778031
Heading	23.967307	21.800281	20.103761	21.932484	20.806362	21.932484	20.619536	20.619536

Scenario 3: Accomplish a task using the shortest time.

Experiment Setup

- The hardware and software setup is the same for scenarios 1 and 2.
- The maximum activation threshold is set to 200. In addition, the minimum threshold is set to 15.
- Priorities: $\omega_{\text{Reach target}}$ =0.9, $\omega_{\text{Following Path}}$ =0.1, and $\omega_{\text{Avoid enemy}}$ =0.1.

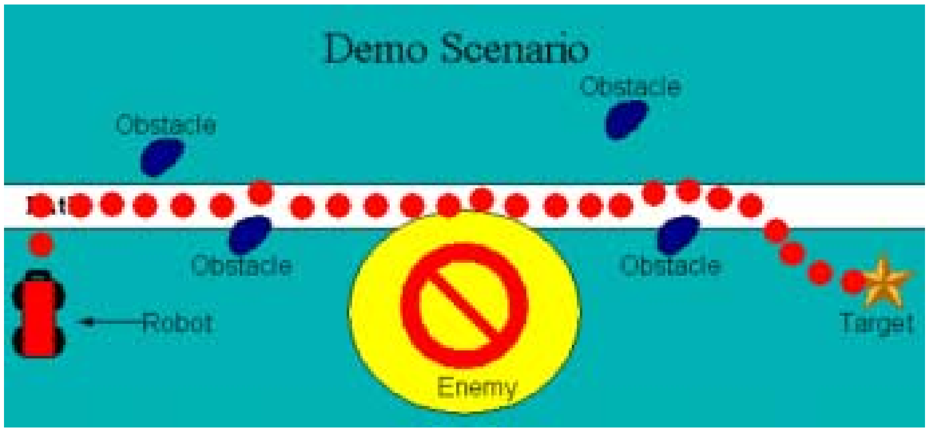


Figure 37 Simulation of the demo that emphasized shortest time

Discussion of Experiment 3

In this scenario the system priority for the Reach Target condition is set very high and consequently to ignore the other goal condition. The robot will start from Move to Path. Whenever robot cannot see the path, it will switch to the Heading competency module without caring about anything else. According to Figure 38 at time 35-59 second the robot can detect the enemy. However, the activation of the Heading is still higher than Run Away competency. The system will thus select Heading instead of Run Away. At the end of the episode the scenario may get the highest reward, if it survives. On the other hand, the robot may be destroyed and can't move anymore (health value =0)

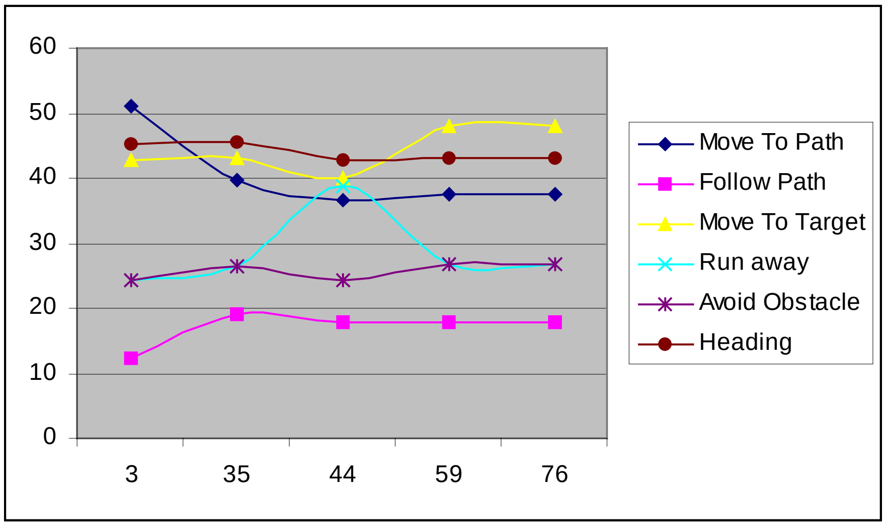
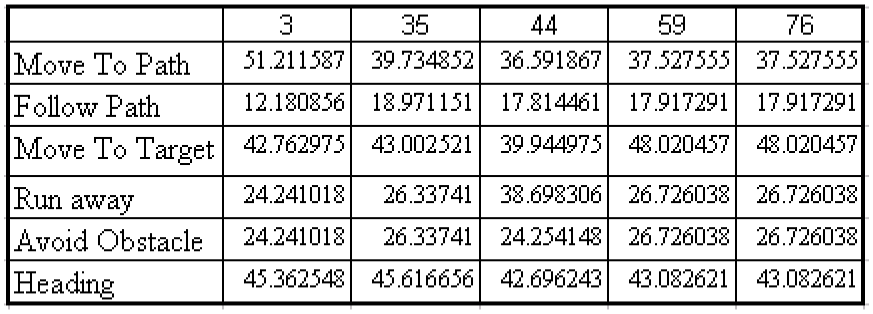


Figure 38: Graph of the result of Scenario 3

Table 5: Numerical result of Scenario 3 (Shortest time)



	3	35	44	59	76
Move To Path	51.211587	39.734852	36.591867	37.527555	37.527555
Follow Path	12.180856	18.971151	17.814461	17.917291	17.917291
Move To Target	42.762975	43.002521	39.944975	48.020457	48.020457
Run away	24.241018	26.33741	38.698306	26.726038	26.726038
Avoid Obstacle	24.241018	26.33741	24.254148	26.726038	26.726038
Heading	45.362548	45.616656	42.696243	43.082621	43.082621

The main difficulty that we faced when implementing this experiment is that we used the vision sensor for our main sensor. The vision sensor is sensitive to lighting conditions. For instance, the "If-see-target" agent can detect the target, but if the light condition changes, this agent may not be able to see the target and the episode may fail or give a bad result.

Communication between SAN-RL and Agents

Before we can use agents in the SAN-RL program we need to add two VectorSignals to communicate with the SAN-RL program. We call Input command and the Output command VectorSignal. There is a structure for sending information to SAN-RL, which is shown below:

Table 6 Input Command VectorSignal

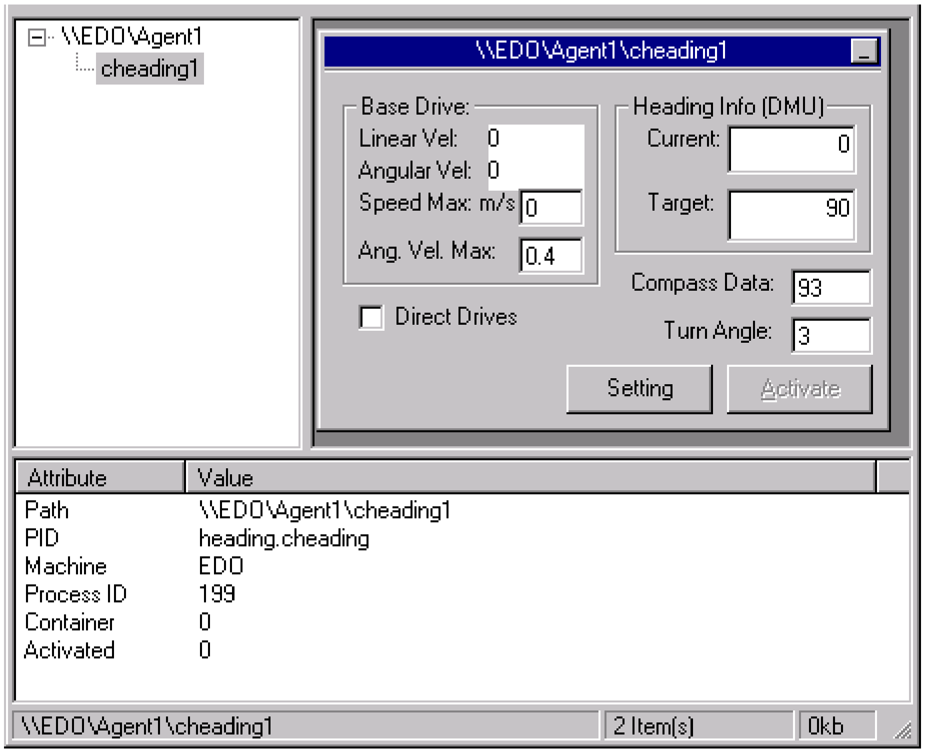
Column	Name	Description	Example
1	Node_Id	Node Id that is corresponding to the Spreading Activation Network which we create. (Drag mouse over its graphic. Then there is a message to tell what its Node_id is.	In our network "Move to road" behavior is 9.
2	Status	Telling the current status of the node that is selected.	0 is false, 1 is true. For competency module: 1 means the task was successful. 0 means the task was unsuccessful
3-n	Data	Some behavior provides additional information.	Heading behavior will send the current head direction of the robot back to SAN-RL program.

Table 7 Output Command VectorSignal

Column	Name	Description	Example
1	Node_Id	Node Id that is corresponding to the Spreading Activation Network which we create. (Drag mouse over its graphic. Then there is a message to tell what its Node_id is.	In our network "Move to road" behavior is 9.
2	Status	Telling Competency module to activate or deactivate.	0 means "Deactivate", 1 means "Activate".
3-n	Data	Some behavior provides additional information.	Heading behavior will need direction for SAN.

Heading Behavior

For this behavior, we have two options to set the angle of the robot. One is from the Compass sensor that is good for outdoor navigational use. On the other hand, if we use a compass sensor in an indoor environment where there is much magnetic interference then the compass sensor gives us very noisy data. Using the DMU is the better option in an indoor environment. For this demonstration, we will use the compass sensor.



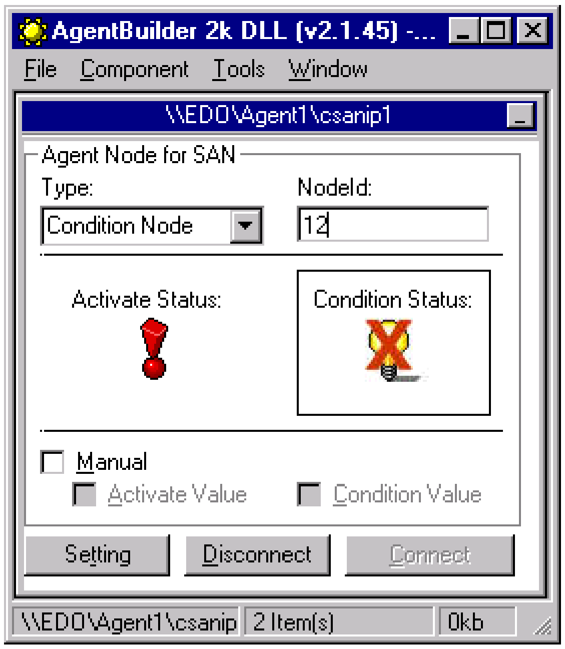
Linear Vel: 0
Angular Vel: 0
Speed Max: m/s 0
Ang. Vel. Max: 0.4
Current: 0
Target: 90
Compass Data: 93
Turn Angle: 3

Attribute	Value
Path	\\EDO\Agent1\cheading1
PID	heading.cheading
Machine	EDO
Process ID	199
Container	0
Activated	0

Figure 39 Show the UI of the Heading application

IMA TCP/IP Client manager for SAN-RL

Since we needed to communicate with the Scooter, which is using Linux as its operation system, we decided to use the TCP/IP protocol to be our solution. All of our TCP/IP servers are on the Linux site, and we already have the client for our basic component. But for the SAN-RL, we needed to have special commands that did not exist in the client that we currently had developed. Thus, we created a new client to have a connection with the SAN-RL program, and could talk to the server manager on the Linux site.


AgentBuilder 2k DLL (v2.1.45) -...
File Component Tools Window
\\EDO\Agent1\csanip1

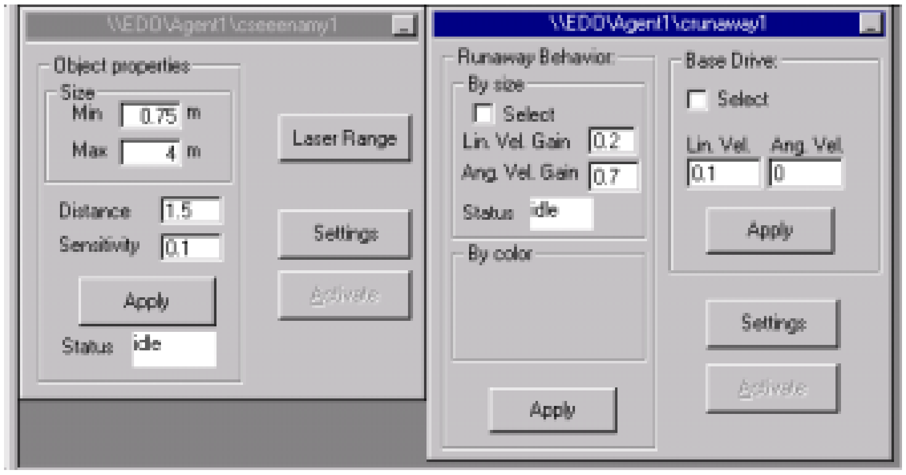
Agent Node for SAN
Type: Condition Node
NodeId: 12
Activate Status:
Condition Status:
Manual
Activate Value
Condition Value

Figure 40 Show communication application UI (client).

To use this program, we needed to set the NodeId parameter and the link to the SAN Input/Output command. We also needed to set a type of node that would be selected from either the condition node or the Competency module.

See Enemy and Avoid Enemy Behavior

These agents use only Lidar to detect the enemy (by size). We could set the size of the enemy and the distance between the enemy and the robot. If it were further than what we set, the robot would not respond to the enemy.


\\EDO\Agent1\cseeenamy1

Object properties

Size

Min 0.75 m

Max 4 m

Distance 1.5

Sensitivity 0.1

Status idle

\\EDO\Agent1\crunaway1

Runaway Behavior:

By size

Select

Lin. Vel. Gain 0.2

Ang. Vel. Gain 0.7

Status idle

By color

Base Drive:

Select

Lin. Vel. 0.1

Ang. Vel. 0

Figure 41 The UI of the See and Avoid Enemy application

CHAPTER VI

CONCLUSIONS AND FUGGESTIONS FOR FUTURE WORK

The development of intelligent control techniques for mobile robots was discussed in this thesis including a Multi Agent System for mobile robot navigation. Basic behaviors were created as building blocks for mobile robot navigation and separated into multiple agents depending on the purposes of each task. A Spreading Activation Network was used as the action selection mechanism for the robot self-agent system. Reinforcement learning techniqueswere also used to allow the robot to automatically learn policies. These were combined to design an intelligent control technique for mobile robot navigation.

The experiments relate to outdoor navigation using basic behaviors as described in chapter V. Most of the sensory components that we had tried to use are for vision-base behaviors. The system is also divided into two levels. The low-level control system consists of hardware interfaces that are represented as resource agents. The high-level control system is comprised of the behavior agents and other atomic agents that are linked to the resource agents. Together, the IMA plays an important role to integrate these agents. This makes the control system implementation more feasible. This shows the beauty of an agent-based system using the IMA framework and its ability to simplify the implementation process.

This chapter summarizes the contributions of this research and suggests directions for future work.

Contributions

This thesis contributes in the followings ways:

- The robot Self Agent architecture for a mobile robot is based on a multi-agent-based approach. The system provides an interface with the atomic agents based on Intelligent Machine Architecture.
- Performance Evaluation of the agent's failures in a multi-agent system.
- A Spreading Activation Network is used as an action selection mechanism for a mobile robot.
- Reinforcement learning allows the robot to learn policies.

- A basic command translation system.

Recommendation for future work

As discussed throughout this dissertation, the robot's Self Agent and Spreading Activation Network with Reinforcement Learning have been addressed. Future research directions include the following issues: adding a more efficient method for Performance Evaluation, creating a better commander agent for communicating with the robot self agent, exchanging data between the SES and LES, adding more information data in the Info agent, adding better abilities to translate commands from the user.

Adding a more efficient method for Performance Evaluation:

Since we used the timer to be our only rule to determine how good the agent performance was, it did not satisfy requirement for each agent. For example, if we want to measure the performance of the "Avoid enemy behavior" agent, we needed to determine if the robot was at a safe distance from the enemy. Instead of using a timer, it would have been better if we had used the distance to determine how good the agent performance was.

Creating a better Commander Agent for communicating with the robot self agent

While I was trying to test and debug this system, I needed to create a prototype of the Commander agent that could understand the commands that the user sent to and received from the robot Self Agent. Since it was not the main purpose of this thesis, it did not have much functionality in helping the user. Therefore, there should be a Commander Agent that not only has every essential command to control a mobile robot, but also can adapt its GUI.

Exchanging data between the SES and LES

When I was working on this project, we were not able to integrate these two systems within the Self Agent. It will be very interesting if we can work and share information between the SES and LES in the future.

Add more information data in the Info agent

Since the Self Agent is the center or brain of the robot, it should provide all information that the user needs to know about the robot. In the future, it may be a good idea to add the SES and LES into this agent too.

Adding better abilities to translate commands from the user

We could possibly get the commands from the user by some other sensor than the text representation. It would be very interesting to see if a robot could get the command from the user by visual signals or by microphone.

REFERENCES

[1] I. Ahrns, J. Bruske, G. Hailu, and G. Sommer. Neural fuzzy techniques in sonar-based collision avoidance. In L. C. Jain and T. Fukuda, editors, Soft Computing for Intelligent Robotic Systems, chapter 8, pages 185—214. Physica-Verlag Heidelberg, Germany, 1998.

[2] V. G. Moudgal, W. A. Kwong, and K. M. Passino. Fuzzy learning control for a flexible-link robot. IEEE Transactions on Fuzzy Systems, 3(2):199—210, May 1995.

[3] R. T. Pack. IMA: The Intelligent Machine Architecture. Ph.d. thesis, Electrical and Computer Engineering, Vanderbilt University, Nashville, Tennessee, May 1998.

[4] R.T. Pack. IMA: The Intelligent Machine Architecture. Ph.d. thesis, Electrical and Computer Engineering, Vanderbilt University, Nashville, Tennessee, May 1998

[5] R.T. Pack, D. M. Wilkes, and K. Kawamura. A software architecture for integrated service robot development. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pages 3774-3779, Orlando, September 1997

[6] Kawamura, K., et al, “Intelligent Robotic Systems in Service of the Disabled”, IEEE Transactions on Rehabilitation Engineering, v.3 pp. 14-21,1995

[7] Kawamura, K., et al, “Design Philosophy for Service Robots”, Robotics and Autonomous System, v.18, pp. 109-116, 1996.

[8] Wildes, D.M., et al, “Designing for Human-Robot symbiosis”, Industrial Robot, 6:1, pp.49-58, 1999

[9] TlL.Thai. Learning DCOM. O’Reilly and Associates, Inc., Sebastopol, California, 1999

[10] K. Kawamura, R.A. Peters II, C. Johnson, P. Nilas and S. Thongchai, “Supervisory Control of Mobile Robots using Sensory EgoSphere”, 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA 2001, Banff, Alberta, Canada. Page 531-537

[11] J.A. Albus. Outline for a theory of intelligence. IEEE Transactions on Systems, Man and Cybernetics, 21(3):473-509, May/June 1991

[12] Pattie Maes. How to Do the Right Thing Connection Science, 1(3):291-323,1989

[13] A Multi-Agent Spreading Activation Network Model for Online Learning Objects http://julita.usask.ca/mable/saad.doc

[14] Real World Interface, Inc. http://www.irobot.com/rwi