Autonomous Environmental Mapping in Multi-Agent UAV Systems

by

Linus Jan Luotsinen
B.S. University of Dalarna, 2002

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2004

Major Professor:
Dr. Ladislau L. Bölöni

ABSTRACT

UAV units are by many researchers and aviation specialists considered the future and cutting edge of modern flight technology. This thesis discusses methods for efficient autonomous environmental mapping in a multi-agent domain. An algorithm that emphasizes on team work by sharing the agents local map information and exploration intentions is presented as a solution to the mapping problem. General theories on how to model and implement rational autonomous behaviour for UAV agents are presented. Three different human and tactical behaviour modeling techniques are evaluated. The author found the CxBR paradigm to be the most interesting approach. Also, in order to test and quantify the theories presented in this thesis a simulation environment was developed. This simulation software allows for UAV agents to operate in a visual 3-D environment with mountains, other various terrain types, danger points and enemies to model unexpected events.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

UAV (Unmanned Aerial Vehicle) units or agents are by many researchers and aviation specialists considered the future and cutting edge of modern flight technology. Remotely controlled UAV units have been researched on and used since the late 1910's. Today's UAV units are being utilized widely in modern military warfare, by governments and by various hobbyists. The area of usability for unmanned units like these is huge, e.g. consider military operations with no human casualties, lower risks to operator and crew, coast and border surveillance guarding and entrance of contaminated areas.

The obvious improvement of the remotely controlled UAV for any intelligent agent, multi-agent and AI (Artificial Intelligence) interested researcher is to introduce autonomous UAV units. There are many interesting issues that need to be addressed in order for a UAV unit to operate autonomously and in teams with other units. E.g. team work communications, cooperation, dynamic role assignment, strategic plans, decision making and tactical behaviour modeling to name a few.

Removing the pilot controlled aspects from air vehicles implies many challenges that need to be addressed. Information acquisition and accumulation requires quality sensors. Air collision detection is also very important in the case where for instance multiple units are working together, thus powerful radars are needed. The autonomous UAV needs to incorporate intelligence to maintain the stability and control of the UAV in dynamic environments.

This thesis discusses methods for autonomous environmental mapping in a multi-agent domain. An algorithm is presented to solve the mapping problem. This algorithm emphasizes on team work by sharing the agents local map information and exploration intentions. Each agent is individually allocating unexplored areas for exploration, based on the other agents activities. It presents theories on how to achieve rational autonomous UAV units that are controlled by its own built-in mind and intelligence. Three different human and tactical behaviour modeling techniques are presented and evaluated. The author found the CxBR (Context Based Reasoning) paradigm to be the most interesting approach. CxBR agents make decisions by the use of visible and audible contextual information and are relatively simple to understand and to implement programmatically. The CxBR modeling procedure is thoroughly described in this thesis. Thoughts on how to achieve satisfactory UAV team work, social abilities and role assignment are also discussed.

In order to test the theories presented in this thesis a simulation environment was developed. This simulation software allows for UAV agents to operate in a visual 3-D environment with mountains, other various terrain types, danger points and enemies to model unexpected events.

Chapter 2 contains the literature review of all the researched techniques used. It discusses fundamental knowledge about intelligent agents and MAS (Multi-Agent Systems). Behaviour modeling techniques such as BDI, PECS and CxBR are described in detail. The history, current activity and other aspects in the research area of UAV units are examined. Also, current topics in envrionmental mapping techinques are presented. Chapter 3 describes the UAV simulator. Chapter 4 describes the approaches for environmental mapping that was used for the UAV simulation domain. Chapter 5 shows the results. Chapter 6 presents the author's conclusions. Chapter 7 introduces future work that may lead to new discoveries and improvements.

CHAPTER 2

LITERATURE REVIEW

2.1 Unmanned Aerial Vehicles

2.1.1 History

UAVs were successfully used prior to the use of manned aerial vehicles. The first recorded use of UAVs were in February 1863 [Eis]. The UAVs at this time were extremely primitive, nevertheless they could perform both combat and surveillance missions. The first combat UAV was developed by Charles Perley. It was a UAV bomber unit built of a hot air balloon that could carry a basket of explosives. The basket was released on a timeout basis.

In 1895 a man named William A. Eddy developed an aerial kite that could take photographs of the ground [Edd97]. This kite idea was later used in the Spanish-American War of 1898, where it provided information to the American troops about the Spanish activities and locations.

Since then UAVs have been used in World War I, World War II, and the Vietnam War. Nowadays we have highly complex UAV units that can fly at altitudes of up to 90,000 feet and for 52 consecutive hours. Today's UAV units have sensors that allow night and day missions. They also have sophisticated camera equipment and radar systems.

2.1.2 Current UAV Systems

The information in this section can be found in [Def01], which is a document created by the DoD (Department of Defense). This document was created as a roadmap to inform research institutes and other organizations about the DoD's intentions of developing UAV systems from year 2000 to year 2025. This section demonstrates some of the capabilities currently available in remotely controlled UAV systems. Note that the word system is used. This means that there in most cases are several operators, video receivers and many UAVs in one system.

2.1.2.1 RQ-1 Predator

The RQ-1 lands and takes off the conventional way. It can carry a payload of 450 lbs and its operational air time including payloads is approximately 24 hours. RQ-1 has been used by the Air Force since 1995 in surveillance warfare situations such as those in Iraq and Kosovo. It is equipped infrared sensors and radars so that it can operate during day as well as night and in various weather conditions.

2.1.2.2 RQ-2 Pioneer

This UAV system is mainly used in battle ships, where it utilizes rocket assistance and catapult technology for take-off. Landing is done by using nets. It has been used by the Navy since 1986 for reconnaissance and surveillance missions in the Persian Gulf, Bosnia and Kosovo. It can fly up to 4 hours with a total payload of 75 lbs. It is equipped infrared sensors and a line-of-sight data link that relays video information in real-time.

2.1.2.3 RQ-5 Hunter

This UAV takes off and lands the conventional way using runways. It has a total flying time of 11 hours and can carry a total payload of 200 lbs during that time. The production of this system was cancelled as of 1996. A few units were created though. These have been used in NATO operations in the Kosovo area. It is equipped with infrared sensors and a line-of-sight link that feeds video information.

2.1.2.4 RQ-4 Global Hawk

This UAV is classified as a high-altitude and long endurance unit. It was designed to cover very wide areas for surveillance purposes. Landing and take-off is done the conventional way. It can carry a payload of up to 1950 lbs for approximately 26 hours. This unit is equipped with a moving target indicator component, line-of-sight video relay and infrared sensors.

2.1.2.5 Fire Scout

This UAV has a vertical take off and landing procedure. It has a flying time of at least 3 hours with a total payload of 200 lbs. It is equipped with an infrared sensor with integral laser designator/rangefinder. This unit was selected by the US Navy to operate from all air-capable ships.

2.2 Agents and Multi-Agent Systems

Autonomous software agents are one of artificial intelligent's most recent research areas. These agents often reside within an agent framework, where they may operate independently on open environments [Woo02]. If multiple agents are executing simultaneously within an agent framework, the agent system is simply recognized as an MAS (Multi-Agent System). Agents in such systems make use of well-defined ontologies to perceive contextual information and share domain specific knowledge with each other [NM00]. Within MAS, agents may incorporate sophisticated communication and cooperation skills. This enables the agents to either work together towards a common shared goal or towards an agent independent goal. Thus, social capabilities are a major concern in MAS. Agents have, to date, been successfully used in applications such as negotiators in auctions and web browsing personalization, using contexts, on the internet [BC03].

Figure 2.1 illustrates a typical MAS. The small grey-filled circles illustrate the agents. The lines between the agents are communication links and each big circle containing a set of agents represents a team of collaborational agents. Each agent's ability to change the environment is restricted to its area of influence which is represented by the projected circles in the figure.

The use of MAS and cooperative agents have several advantages over single agent systems [BFM00]. First of all, a cooperating team of agents may increase the efficiency to perform a certain mission. Consider an exploration mission where an agent needs to explore an unknown environment. The time to do this could be greatly reduced if there where several cooperative agents exploring the environment systematically. Another benefit of MAS is the fact that a system becomes more fault-tolerant because of the redundancy implied by a MAS. Consider a military combat operation where the com-

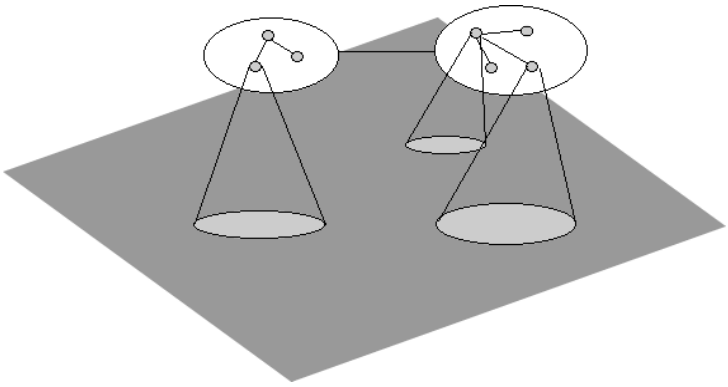

Figure 2.1: Typical Multi-Agent System

manding officer for some reason is taken out of action. In this scenario the next lower ranked soldier will take over the commanding post.

2.2.1 Automatic vs. Autonomous

There has been some confusion in the past about the words automatic and autonomous among people [Clo02]. It is important to understand that the difference is significant. Basically automatic means that the agent will do exactly as it was told, programmatically or in any other way. The automatic agent has no choice to make decisions. An autonomous agent however has its own will and can make decisions based on its own interests. In [Clo02] Clough discusses a few systems that are autonomous and others that are automatic. An autopilot and a cruise missile are both automatic because its decisions are pre-made. While an autonomous guidance system decides which course to take and then stays on it.

2.2.2 Intelligent vs. Autonomous

There is a common misconception and misbelieve among people that the words autonomous and intelligent means the same thing. According to Clough [Clo02] intelligence is defined to have the capability of discovering knowledge and using it to do something. And autonomy, as mentioned before, is defined to have free will and the ability to generate one's own purposes without any instruction from outside. Hence, autonomous agents can be very dumb as long as they perform their objectives.

This is a very controversial subject that has been discussed many times in hundreds of papers. The problem lies within the definition or lack of definition of the word intelligence. No one seems to know for sure what can be defined as intelligence.

2.2.3 Communication

It is of great importance for agents in a MAS to speak the same language. There are a few standards available to use. However FIPA (Foundation for Intelligent Physical Agent) [FIP03] and KQML [FWW93] are the most commom languages. The next section will present the FIPA communication language.

2.2.3.1 FIPA

A typical FIPA message consists of a sender, a receiver, contents and a performative. The performative is used to classify the message into meaningful low-level intentions. There are 22 performatives available in the FIPA standard. See [FIP03] for a complete description

of these commands. The FIPA performatives can be classified into five different groups [Woo02]. These are:

1. Passing information
2. Requesting information
3. Error handling
4. Performing actions
5. Negotiation

The passing information group of perfomatives can be seen in table 2.1. These performatives are used to transport and inform an agent's decisions and data.

Table 2.1: FIPA Performatives - Passing infomation

Performative	Desciption
confirm	This message is sent whenever an agent wants to confirm the truth of some statement
disconfirm	This message is sent whenever an agent wants to confirm the misbelief of some statement
inform	This message is sent whenever an agent wants to share information
inform-if	This message is sent to verify if a statement is true or false. Typically as the contents within a request message
inform-ref	Similar to the inform-if message with the difference being that it is the value of an expression that is being asked for

The request information performatives are listed in table 2.2. These perfomatives are used whenever an agent wants to know something about another agent. Note that the cancel performative is also a member of the performing actions group.

Error handling performatives are used whenever something is not understood or something failed during communication between the agents. Table 2.3 lists all the performatives available in this group.

Table 2.2: FIPA Performatives - Requesting information

Performative	Desciption
cancel	This is sent whenever an agent wants to cancel a previously submitted request
query-if	This message allows for agents to query if something is true or not
query-ref	This message is used to ask for values of expressions
subscribe	Sent whenever an agent wants to be notified when some event occurs

Table 2.3: FIPA Performatives - Error handling

Performative	Desciption
failure	This message is sent to indicate that the agent failed to perform some action
not-understood	Sent whenever an agent doesn't understand the message of another agent

The performing actions performatives are listed in table 2.4. The intended use for these performatives are to inform agents about the various actions an agent may have taken.

Table 2.4: FIPA Performatives - Performing Actions

Performative	Desciption
agree	Indicates that the agent agrees to perform some action
cancel	This is sent whenever an agent wants to cancel a previously submitted request
propagate	This message is sent whenever an agent wants another agent to propagate a message to other agents
proxy	Used for forwarding of messages
refuse	This message is used to indicate that the agent refuses to perform some action
request	Request for some agent to perform some action
request-when	The same as request but only if some statement is true
request-whenever	Similar to request-when with the difference that the requested action should be performed whenever some statement is true

The negotioation group of performatives gives agents the possibility to perform negotiations, e.g. auctions and contract nets. Table 2.5 shows these performatives.

Figure 2.2 shows an example of a FIPA message. In this message we can see that the sender, tankl, wants to inform an ememy position to the receiver uav1.

Table 2.5: FIPA Performatives - Negotiation

Performative	Desciption
accept-proposal	Allows for an agent to state that it accepts another agents proposal
cfp	Call For Proposals is used to initiate negotiations
propose	This message is sent to make a proposal to another agent
reject-proposal	This message rejects a suggested proposal

```
(inform   :sender tank1
          :receiver uav1
          :content (enemy SAM 120 100 872)
)
```

Figure 2.2: A FIPA message example

2.3 Agent Behaviour Modeling

When modeling and implementing behaviour of various types (e.g. human and tactical) in agents or robots it is necessary to utilize some kind of behavioural framework to maintain sufficient quality and understanding of the implementation. There are many different frameworks, methods and techniques that can be used to model agent behaviour. Of course, some are better than others. This section aims to introduce some of the most

commonly used frameworks as well as some of the more recent and innovative approaches. In this section we will discuss:

1. BDI (Belief, Desire, Intention)
2. PECS (Physical, Emotion, Cognitive, Social)
3. CxBR (Context Based Reasoning)

The most known and commonly used frameworks are based on the BDI methodology. BDI has been described in many papers, however the paper by Rao and Georgeff [RG95] is probably the most informative and descriptive BDI paper as of yet. PECS is an organizational approach that aims to categorize the behavioural features of an agent into modules. PECS is based on the work of Schmidt and Urban, they describe the methodology in [Sch01], [Urb00], [Urb01] and [US01]. CxBR is a very useful modeling mechanism for tactical behaviour agents such as military units and various game creatures. It can be viewed as a specialized finite state machine. CxBR has very good implementation features such as ease of reuse, inheritance and a well defined structural design mechanism in the form of contexts. CxBR is based on the work of Gonzalez and Ahlers and is described in [GA98], [GA96] and [GA94]. CxBR is the choice of use in this thesis for modeling UAV agents.

2.3.1 BDI

Using BDI modeling, the agent is provided three mental states or attitudes. These are belief, desire and intention. The attitudes represent the informational, motivational and deliberative states of the agent respectively. It is these attitudes that determine the behaviour of the agent [RG95]. BDI modeling can be used for implementing any kind

of intelligence within an agent, so why is this approach a good one for human behaviour simulation? The answer is that BDI framework stems from well-known folk psychology philosophies, where human behaviour is predicted and explained by attitudes such as believing, fearing, hoping etc. Therefore BDI closely maps to the way human beings ordinarily explain their reasoning and facilitates their knowledge [Nor03].

2.3.1.1 Attitudes

To understand why we need all of the BDI agent attitudes, we need to know the environmental and system characteristics necessary for a realistic simulation. In [RG95] Rao and Georgeff describe an air traffic system where an agent calculates the ETA (Expected Time of Arrival) for arriving aircrafts and issues control directives to the pilot in the aircraft etc. The listing below describes what environmental and system concerns we need to take into consideration when modeling such agent system.

1. The environment is non-deterministic. Meaning that the environment may change in many ways at any instant in time.
2. The system is non-deterministic. Meaning that the system may take many different actions to execute at any instant in time.
3. The system has many different objectives or goals that it is asked to accomplish at any instant in time. These goals may interfere with each other, and thus not being simultaneously achievable.
4. The best actions and procedures chosen to achieve the objectives are dependent on the state of the environment, not on the internal system itself.

5. The environment can only be sensed locally. Meaning that one sensing action cannot determine the state of the entire environment.

6. The state of the environment may change during execution and computation of actions.

An agent architecture that can uphold all of the above characteristics is difficult to create, however BDI is an attempt to do exactly this. We need beliefs in the system, as it is essential for the system to have information on the state of the environment. Also, the beliefs need to be updated properly after every sensing action. The beliefs can be viewed as the informative part of the system. Desires are needed because the system needs to have information about what objectives it wants to accomplish. The desire component holds this information and can be considered to be the system's motivational component. The intentional component of the system is needed because of the fact that the environment changes instantaneously, therefore the system must commit to its actions and not continuously re-evaluate them. The actions that are committed are considered to be the system's intentions, and that is why this component can be viewed as the deliberate component of the system.

2.3.1.2 Data Representation

In decision tree structures there are decisions nodes, chance nodes and terminal nodes. To each arc emanating from a chance node there are probabilities assigned, and to each arc emanating from a decision node there are payoff values assigned. There are also probability and payoff functions that performs the mapping of probability values to chance nodes and payoff values to terminal nodes respectively. In such a tree structure, we can utilize a deliberation function so that the best possible path of actions can be taken. We can use

decision trees when representing the BDI attitudes in a BDI agent. This representation can be achieved by transforming a complete decision tree, containing all the possible action paths and environmental states available, into a set of decisions trees by expanding the chance nodes. Thus creating a set of possible worlds where each possible world tree represents a unique state in the environment. Consider for example when an airplane is flying between different waypoints towards an airport [RG95]. In such scenario there might be an uncertainty in the environment because of the wind velocity at the various waypoints. There will be a possible world representing each of these points. These worlds are called the belief accessible worlds. In each belief accessible world there are choices available, such as at which speed and altitude to use, choices like these are represented by multiple branches in the tree. The final waypoint is the airport. This node is equal to the desired ETA for the air-plane. Thus, the desires of the airplane agent are those where the calculated ETA is equal to the desired ETA. Therefore, the desire accessible worlds can be obtained by pruning the belief worlds in a way that the ETA constraint holds. The intention accessible worlds are obtained from the desire accessible worlds in a way so that the best paths are chosen to be executed. These paths might be depending on the best fuel consumption or aircraft performance etc. In [RG95] Rao and Georgeff present a decision tree to possible worlds transformation algorithm and they argue that by performing such transformation one can model systems where insufficient probabilities and payoff values exists, and that the system improves its dynamic adaptation aspects.

2.3.1.3 Abstract Architecture

The BDI architecture presented here stems from the work of Rao and Georgeff [RG95]. They have designed an abstract BDI interpreter which can be used in any BDI based agent system. The architecture manages the three dynamic attitude states of BDI, external and

internal events. The events are assumed to be atomic operations and they are recognized by the system after they have occurred. Also, the internal events are considered to be asserted to the event queue as they arrive. The abstract code listing below shows the necessary functions needed for the interpreter to work.

```
initialize-state();
repeat
    options := option-generator(event-queue);
    selected-options := deliberate(options);
    update-intentions(selected-options);
    execute();
    get-new-external-events();
    drop-successful-attitudes();
    drop-impossible-attitudes();
end repeat
```

In the listing it is assumed that the event queue, belief, desire and intentional data structures are global. The first thing that occurs in each iteration is that options are generated by calling an option generator. This option generator uses the event-queue to gather proper options (these options are the desired options). Next, the deliberate function selects a subset of the options and adds them to the intentional data structure using the following update function. After this, the action in the intentional structure is executed. Next, the agent senses for any new external events, and adds them to the event queue. What remains to do is some cleaning up, the successfully executed desires and intentions are dropped, and also the impossible attitudes are removed from the structures.

2.3.1.4 Practice

When practically implementing BDI agents based on the architecture covered above, there will be hurdles such as agent real-time requirements and lack of agent reactivity that we must overcome somehow. The option generator and the deliberator procedures must be made fast enough to cover these aspects. Also, the abstract interpreter is not a practical system for rational reasoning because it is based on a closed set of beliefs, desires and intentions. Somehow, we must trade off some of the expressive powers of BDI for a practical system. A few choices were provided by Rao and Georgeff [RG95] to simplify the data representation in BDI so that the agent's rational reasoning mechanism becomes more realistic for practical systems.

1. Only represent beliefs about the current state of the world. These states are explicit, and with no implications or disjunctions.

2. Represent means of how to achieve future world states. These means are called plans and can be considered to be a special form of beliefs that contains abstract specifications of means for achieving desires, and options available for the agent. Each plan consists of a body describing the primitive actions or sub-goals that have to be executed for a plan execution to be successful, an invocation condition that triggers the plan and preconditions that must be true for the plan to be executed.

3. Each intention the system forms is represented implicitly by using a structured set of hierarchically related plans.

2.3.2 PECS

Recently, agent based systems based purely on cognitive abilities have been more and more criticized by researchers and industrial organizations for not having enough capabilities to simulate complex human behaviour. Agents in these kinds of systems are considered to be pure rational decision makers, while humans have more capabilities than that. PECS aims to expand cognitive models, such as BDI, so that more human-like simulations can be performed. The basic ideas behind PECS was to incorporate component oriented hierarchical modeling, meaning that the structure of the framework can be decomposed into a set of components instead of having one complex structure. The smaller and less complex components, in turn, have internal states where values are being generated depending on the environmental inputs, the agent outputs and by interactions with the components themselves. PECS has been designed in a way so that in attempting to model human behaviour agents, the agents will be equipped with similar properties and behavioural patterns as real human beings would have had in a given scenario [Urb01]. These properties include support for learning, social capabilities, emotions, rational thinking, cognition and physical representations. In terms of agent architectures, PECS is classified as an hybrid architecture [Urb00]. The meaning of an hybrid architecture is that the architecture supports modeling of both reactive and deliberate agent modes. Thus, compared to the aforementioned BDI method that only supports static pre-programmed beliefs and desires, an agent modeled with PECS is able to expand its knowledge base while executing in the environment and pursue agent created plans that the agent wants to accomplish.

2.3.2.1 Architecture

PECS is based on eight major components, distributed over three layers [Urb00]. There is an input layer which handles inputs and perceptions from the environment where the agent is operating, and then there is an output layer which makes sure that the agent's behaviour and actions are realized on the environment. The most interesting layer of PECS is the layer located in the middle of the aforementioned layers, namely the internal layer. In the internal layer the agent's physical, emotional, cognitive and social status is being modeled. The components available in system can be, and are in most cases, interactively dependent on each other. Which components that are interacting with each other is a decision based on the underlying simulation scenario. Each component in the internal layer is using a state variable and a transition function to simulate different states, e.g. to simulate an angry agent, the emotional state variable in the emotional component is set to angry. The states are controlled by a transition function which operates on transition rules, these rules determine the state of the component by collecting data about the other component states, current knowledge and environmental data in the agent. For example, a surfing interested agent may be set to happy, if the knowledge of the agent shows the work day is over, the sun is shining outside and that the ocean waves are surfing friendly. The remainder of this section provides a detailed description about all layers, components and available interactions between components.

2.3.2.2 Input Layer

The input layer has two components, the sensor component and the perception component.

The sensor component is the agent's environmental input interface. Thus the sensor makes information available for other components to evaluate and process. There are two types of information that can be received by the sensor component, visual information and audible information. Visual information is classified as environmental information while audible information is classified as messages from other agents in the system [Urb01]. Also, the sensor has an internal state which makes it possible for agents to interpret sensory information differently. The sensor component interacts by providing its data to the perception component or to the physical component based on the nature of the sensor data. Data that trigger physical actions are passed directly to the physical component, while data that require cognitive processing are delivered to the perception component. The sensor component also interacts by receiving data from the physical component. This enables physical factors to influence the sensor information.

The perception component retrieves information about the environment through the sensor component, it is up to the perception component to interpret and organize this information into useful data, e.g. a digital picture might be analyzed to detect enemy unit positions in a warfare simulation by a neural network in the perception component. The perceptive component organizes the data in perceptive chunks. These data chunks are then sent to the internal layer's cognitive component for further processing. The sensor information that is being processed is influenced by the agent's current cognitive, emotional and physical appearance.

2.3.2.3 Internal Layer

The internal layer consists of the status component, cognition component, emotion component and physical component. As mentioned above, each component has a state variable,

a transition function and transition rules that all are being used to represent the agent's dynamic human states. It is important to understand how these components are interacting and which kind of logic that needs to be located in each one of the components.

The cognitive component is used to store knowledge about the agent's own internal state as well as the environmental state. To keep the knowledge base up to date with the environmental perceptions and consistent with the agent's own actions a knowledge pre-processing mechanism is implemented within the component. This mechanism is activated whenever the perceptions component has new data or when the agent performs an action. The cognitive component also has a mechanism that simulates what the agent remembers, that is what data in the knowledge base that might be forgotten under certain circumstances. This mechanism is called the retrieval mechanism. Another important mechanism in the cognitive component is the knowledge acquisition mechanism. This mechanism is used to model cognitive processes intended to extend the knowledge base. The knowledge acquisition is a goal directed thinking process that incorporates logic, deduction and algorithms.

The emotion component represents the agent's emotional state. This component is useful whenever we need agents that simulate how emotional changes and states are affecting the human behaviour. The emotional component interacts with the cognitive, physical and behaviour components both ways, thus emotional state is decided based on the other components states and other components states are influenced by the emotional state.

The physical component represents body related information, such as in which way an agent is walking, how old the agent is or how tall the agent is. This component interacts with the sensor component and the emotional component both ways. Based on the actor component the physical component may change some of its information, e.g. when the actor component performs a move action the physical component wants

to update its walking direction information. Also, the physical component affects the perception component and behaviour component in either a positive way or a negative way. For example, bad health might harm the way the agent perceives information and the way behaviour is carried out.

The social status component represents the agent's social attributes, such as social security number, wealth, name or even relationships to other agents. This component can be used to simulate agent families or agent groups, thus based on attributes stored in the social component one can determine where an agent socially belongs in MAS.

2.3.2.4 Output Layer

The output layer consists of two components, the behaviour component and the actor component.

In the behaviour component the reactive and deliberate behaviours of the agent is realized, the component interacts with all the internal layer components so that all of these have influence on which behaviour the agent have. When reactive behaviour is used the agent simply relies on pre-defined rules or strategies to generate its actions, similarly to BDI agents. An agent that dynamically needs to come up with plans of actions to perform has problems using only reactive behaviour due to its pre-defined nature. The deliberate behaviour kicks in whenever such behaviour is needed. The deliberate mechanism can create plans dynamically using the agent's environmental and internal data. Thus, an agent that is presented into a new environment can adapt to this environment and perform its actions in rational way.

The actor component uses the behaviour generated plans and executes them on the environment.

2.3.2.5 Simulation Scenarios

PECS have been used to simulate social human behaviour, in this simulation PECS agents are operating as agents who are joining study groups and leaving study groups based on their knowledge and social states. The social need and knowledge are simulated by using differential equations. The papers [US01] and [Sch01] contain a complete description of this group model simulation.

Another simulation called Adam shows a PECS agent which operates on a small world that has danger points, food resources and neutral land points. Based on the agent's need for food and its emotional state, the agent will make decisions whether to move to a food source or to move to a natural point. The agent also memorizes the environment so that known food and danger point can be avoided or visited again. The agent's need for food is simulated by simple logarithmic functions. This simulation is described in detail in [Urb01].

2.3.3 Context Based Reasoning

Context Based Reasoning (CxBR) is primarily used to model tactical agent behaviour. The CxBR paradigm contributes with a simple and easily understood modeling technique that can be used to concisely represent knowledge and behaviour in intelligent agents. The idea is to make contextual information influence the agent in its various decisions. The contextual representation implicitly makes the key features from each accommodated situation the most important features based on the background or experience of the agent in each situation. CxBR is based on the work of Gonzalez and Ahlers and is described

in [GA98], [GA96] and [GA94]. Because of CxBR's productive modeling method it is the choice of use in this thesis for modeling tactical UAV agents.

2.3.3.1 Architecture

CxBR is based on the ideas that any recognized situation inherently defines a finite set actions that address the current situation. And that the current situation then can be used to simplify the identification of future situations by focusing on those that are likely to happen [GA98]. The CxBR paradigm of knowledge and tactical behaviour representation is built upon the following major components.

1. Agent
2. Mission context
3. Major contexts
4. Sub contexts
5. Inference engine

The agent component is used as a base for a CxBR agent and it contains valuable information and capabilities, e.g. localization and velocity, about each individual agent in a system. The mission context component is used to describe the agent's overall objective and detect when it has been reached. Hence, each agent is assigned with a mission. Major contexts is an interesting component in CxBR, it contains transition rules and sub-contexts that may be activated during the agent's life cycle. Basically a mission context is built upon a set of major contexts and their sub contexts. The last component, the inference engine, is a general purpose component that shall be used when applying

rule-based, expert-system like, knowledge to agents. By using the inference engine and the fact base, new knowledge may be derived using either forward-chaining or backward-chaining. Figure 2.3 illustrates the relationships within CxBR and its components. These component will now be described in detail.

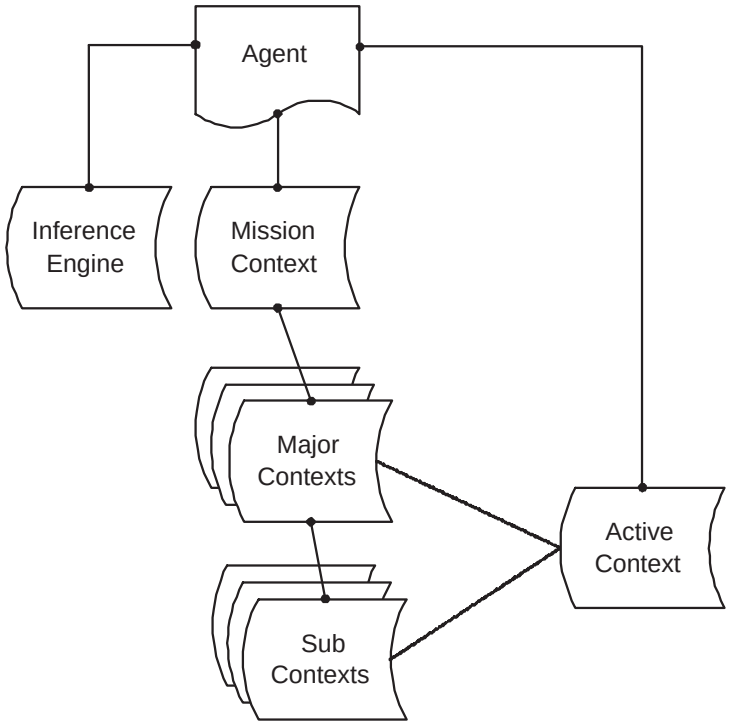

Figure 2.3: CxBR architecture.

2.3.3.2 Agent

The agent component in CxBR is responsible for the initial execution of contexts. Procedures that enables or disables an agent's reasoning is placed in this component. An agent

should contain valid information and properties specifically tailored for the simulation or application domain. However, the agent generally contains a mission context and its currently active major context. The agent also maintains the inference engine of the system. Hence, knowledge acquired and derived are stored within the agent. See figure 2.3.

2.3.3.3 Mission Context

As mentioned before, the mission context manages the overall objective of each agent. Mission contexts are mutually exclusive. Thus, only one mission may be allowed to execute within one agent. The objective rule for each mission is described by simple if-then-rules. Before activating a mission context, it is usually initialized with a set of valid major contexts. Also, to allow for transitions between these major contexts transition tables are created.

2.3.3.4 Major Context

Major contexts are activated whenever their activation rules are fired. These rules, called sentinel rules, are repeatedly executed to determine each contexts activation status. If a context's sentinel rule was fired, then it will become the active major context within the agent. The behaviour of major contexts is modeled using scripts. Hence, once activated the script for the context will execute. Major contexts can also contain sub contexts which will be described below.

2.3.3.5 Sub Context

Sub contexts have the same physical architecture as major contexts. They consist of sentinel rules as well as scripts. However, sub contexts are used perform sub-tasks that are asked for by a major context. Hence, sub-contexts add another layer of intelligence within the CxBR agent. Sub-contexts simplyfies design by providing hierarchal structure to the already modularized context based approach.

2.3.3.6 Inference Engine

The inference engine is a part of the agent component. It is a general purpose component that may be used to derive and maintain knowledge in an expert system like manner. Also, the inference engine may contain the rules that activate the various context types. A good inference engine have support for both goal directed, backward chaining, algorithms as well as the popular data driven, forward chaining algorithm.

2.3.3.7 CxBR in Practice

CxBR has recently been used to model autonomous automobile driving behaviours [GPG00]. The automobile in this simulation was intelligent enough to determine when to stop, accelerate or decelerate. The following major context where used in the simulation:

1. Rural-road
2. T-Intersection
3. Traffic light

4. 4-way intersection

This major context library was then expanded with a set of sub-contexts that provided sufficient detail to control the agents.

2.4 Environment Exploration

The problem of having an agent exploring an unknown environment has been the topic of many research papers in the past. However, performing this task collaboratively and in an coordinated way in a MAS has not been exploited by many researchers. There are many advantages in utilizing a MAS for this task, the most obvious one being a reduced time factor. This section will discuss the problems as well as possible solutions of environmental exploration. The solutions presented here are focused on probabilistic approaches. We will also introduce the basics of map representations and environment characteristics.

2.4.1 Problem Statement

Environmental exploration in multi-agent systems mainly consists of the following different problems:

1. Map representation
2. Path-finding
3. Obstacle aviodance
4. Information merging and sharing

5. Expected visibility

Map representation is the designer's choice of the underlying map architecture for agents. The choice of map architecture is of great importance for perfomance and efficiency. The map should be chosen and tailored based on the agents application domain. Path-finding represented by item two in the list above is basically the process of generating and planning a path for a movable robot or any type of moveable agent in an environment. Obstacle avoidance is simply recognized as a mechainism to ensure a safe journey through the planned path. Information merging is the problem of sharing and merging the agents different perceptions of the environment. By doing this the agents can expand their knowledge by learning from other agents in the system. The last and most difficult task is to predict what will be seen by an agent if it decides to move to a particular way-point. If this prediction is good the agent will have a good idea of how much of the environment it will sense. Such a prediction is needed to realize a good MAS collaboration strategy.

2.4.2 Environment Characteristics

Environments within computer generated applications have several characteristics that need to be explained to fully understand the problem of agent life cycles and behaviours. According to [Woo02] environments can be characterized as a combination of the following classification items:

1. Accessible vs. inaccessible
2. Deterministic vs. non-deterministic
3. Static vs. dynamic

4. Discrete vs. continuous

In this section we will discuss each one of these items so that agent environments can be classified and evaluated accordingly. As we will see real-world environments are inaccessible, non-deterministic, dynamic and continuous. The material presented here originates from a thorough description by Wooldridge in [Woo02].

2.4.2.1 Accessible vs inaccessible

Within accessible environments an agent can get accurate and immediate information about the state of the environment at any time. Inaccessible environments on the other hand will not provide complete information about the state of the environment at all times. E.g. an agent residing in an inaccessible environment may receive environmental information based on its position or equipment capabilities. Hence, inaccessible environments map to real-world environments. Accessible environments do not.

2.4.2.2 Deteministic vs Non-deterministic

Within deterministic environments an agent's actions always result in single guaranteed result. Hence, there is never any uncertainty for the agent's action outcomes. This will clearly simplify design of agents since they don't have to incorporate any intelligence for failures. Non-deterministic environments forces agents to handle failures and limited environmental influence events. For real-world environments non-determinism must be modeled into the agent's perspective.

2.4.2.3 Static vs Dynamic

Static environments simply remain unchanged over time unless actions performed by agents, intentionally or unintentionally, change its properties. In such environments there exists no external uncontrolled, from the agent's point of view, influence sources. Agents within dynamic environments must realize that the environmental state may change over time without any actions performed by the agents. Also, an agent in such an environment must take into consideration that the action it performs may have a different outcome, than the one it expected, because of the dynamic nature. Real-world environments are highly dynamic.

2.4.2.4 Discrete vs Continuous

Environments are considered discrete if there is a finite set of possible outcomes and actions that may be performed in it. Continuous environments on the other hand have infinite number of outcomes and actions. Computer generated environments with discrete properties are simpler to implement than continuous ones. A computer may however simulate continuous environments to desired precisions.

2.4.3 Centralized vs. Decentralized

There are centralized and decentralized environment exploration approaches. Within a centralized approach the agent share their knowledge in global knowledgebase. These systems are also called black-board systems. Then there is the decentralized system in

which the agent stores its environmental state in a local knowledge base. The information is then distributed among the agent by various communication schemes.

2.4.4 Map Representations

A map is needed for an agent to know its own location as well as locations of other agents in an environment. The problem of building maps was defined in [TGF98] as the problem of determining the location of certain entities, such as landmarks or obstacles, in a global frame of reference. The map can be stored locally for each agent or in a central unit. It is very important to choose map architecture based on the problem that is to be solved. Using a well defined, robust and well understood map architecture is critical for search algorithms and their performance. In this section we will focus on three different map representations or architectures:

1. Topological maps
2. Geometrical maps
3. Probabilistic maps

2.4.4.1 Topological maps

Topological maps, see figure 2.4, are based on graph-like structures. The map consists of edges and nodes. Edges represent trajectories that the agent can move towards and away from. Nodes represent the configuration space of the agent. The use of these maps and their representation varies widely based on its application domain. However, common topological maps pretty much always incorporate the following features [RK02]:

1. Use of sensory input to identify nodes

2. Connectivity relationships among nodes

3. Local metrical information associated within edges

According to the work of [RK02] topological maps can also be hierarchically ordered. Hence, one node may have an edge towards another layer of nodes and edges. Hierarchies have always been considered good practice for human beings and their understanding. Topological maps have been successfully applied to navigation robots in systems such as [NPB95], [HK96] and [TGF98]. Lines in figure 2.4 represent edges. Filled dots represent nodes.

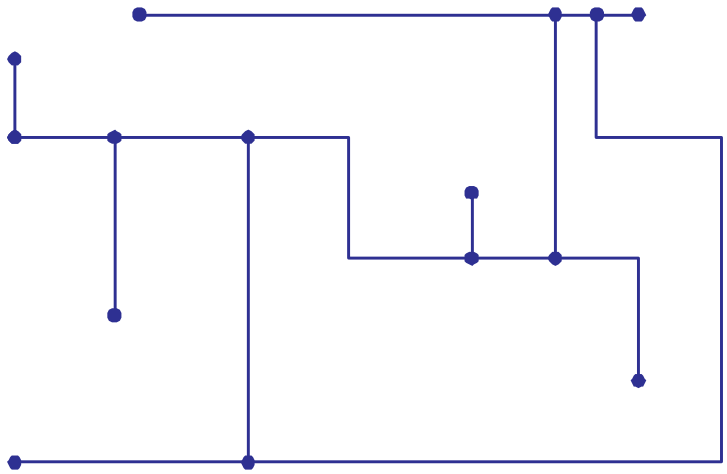

Figure 2.4: Environment representation using a topological graph.

2.4.4.2 Geometrical maps

Geometrical maps use the very primitives of geometrical mathematics, i.e. rectangles, triangles, polygons and lines, to represent maps, see figure 2.5. Mapping in these representations consists of estimating the parameters of each primitive so that they fit the current situation. One of the most interesting approaches for navigational maps using the geometrical approach is the navigation mesh (NavMesh). The NavMesh comes with two major flavors [Toz04]:

1. Triangle based
2. N-sided polynomial based

The difference between these is self-explanatory, triangle based meshes consists only of triangles while N-sided polygonal based meshes can contain any type of polygon in its representation. One important rule to remember when using these meshes is that the polygons must be convex, otherwise navigation can not guarantee that an agent is able to move anywhere within a polygon. Geometrical maps have been used to represent three-dimensional structures in many recent papers, see [TBF00]. In figure 2.5 the rectangles depict obstacles.

2.4.4.3 Probabilistic maps

Probabilistic maps are represented by grids, normally in hexagonal or square form, with a probability value assigned to each cell. One important offspring of the probabilistic grid map is the occupancy map. The occupancy map was originally derived by the work of Elfes and Moravec in the mid-eighties [ME85]. The main idea is that the probability value

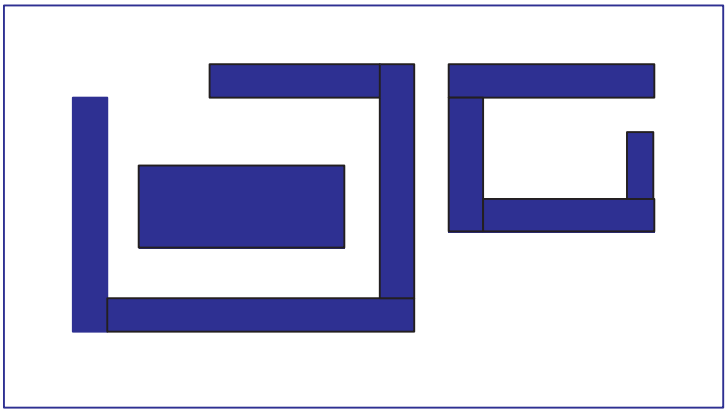

Figure 2.5: Environment representation using a geographical map.

assigned to each grid-cell in the map relates to how possible it is that this particular cell is occupied by another object, landmark or obstacle. Many benefits, such as ease of map integration and good uncertainty management, derive from using occupancy grid maps. These map architectures are mainly used in robots or agents that maintain sensory noise in their perceptions of the environment. Such robots or agents are equipped with sensors that are laser based or sonar based. Integrating and sharing occupancy maps within multiple agents is very easy to implement. Sound probabilistic mathematical formulas can be used to perform this. These formulas can be found in [Yam98] and [Mor88]. However, the basics of map integration are summarized in [BFM00]:

$$P(occ_{x,y}) = \frac{odds_{x,y}}{1 + odds_{x,y}} \tag{2.1}$$

where

$$odds_{x,y} = \prod_{i=1}^{n} odds^{i}_{x,y} \tag{2.2}$$

and

$$odds^{i}_{x,y} = \frac{P(occ^{i}_{x,y})}{1 - P(occ^{i}_{x,y})} \tag{2.3}$$

Where, $P(occ_{x,y})$ is the merged probability that cell (x, y) in the map is occupied by another object and n is the number of maps to merge. In figure 2.6 two agent operate in a joint environment. The two agents merge their maps whenever a new sensor scan is performed. The light-grey areas corresponds to known space, the dark-grey on the other hand is to be recognized as unknown space.

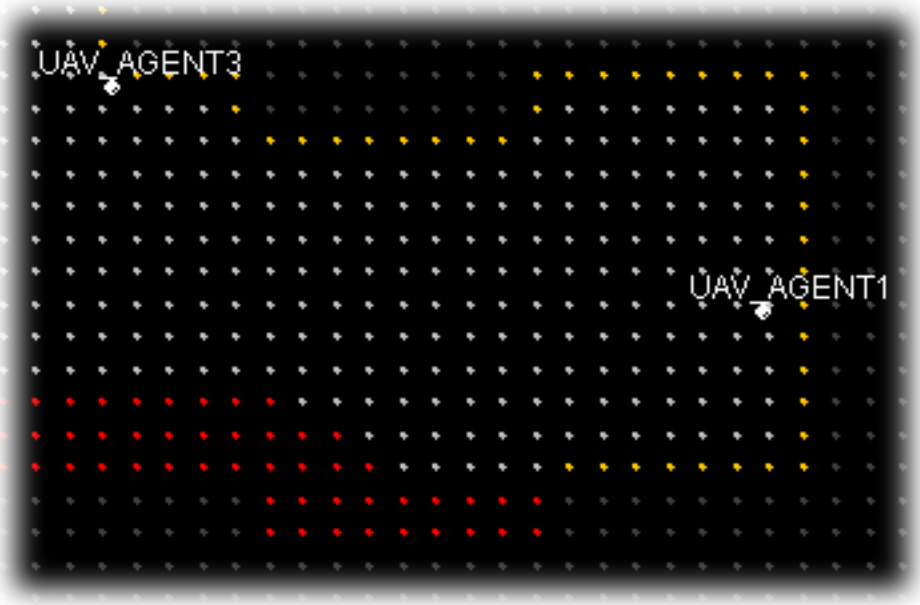

Figure 2.6: Environment representation using merging occupancy grid map.

2.4.5 Path-Finding

Path-finding is the process of generating or planning a path for a movable robot or any type of moveable agent in an environment. The path-finding problem and its solutions are classified as classic artificial intelligence practices. Hence, there are many approaches for solving this problem. The problem can be solved with a wide range of search algorithms, such as breadth-first, depth first, iterative deepening and A-star (A*) [SN95]. However, only the last mentioned A-star, which is one of the most promising and complex search algorithm, will be discussed in this section.

2.4.5.1 A-Star Overview

A-star is very flexible and will, assuming that two points are given, attempt to find the optimum path between the points. A-star is a directed search algorithm which means that it doesn't search blindly for its desired path. The algorithm makes use of back-tracking so that paths can be reconsidered during execution [Mat02]. A-star uses a map as a search space, and it is this map that represents the area of which A-star works. A-star maintains its map information within several nodes. Hence, nodes are the structures that represent positions within a map. Also, nodes maintain heuristic information and cost values used for determining its suitability in the search process. It is important to know that the heuristics and costs are completely application specific.

2.4.5.2 A-Star Algorithm

The aforementioned nodes maintain the search algorithms map positions, fitness values, heuristics and costs. In the algorithm presented below g represents cost, h represents heuristic value and f the fitness value. To guarantee an optimum path, the heuristic must be admissible. Hence, it must never over-estimate any heuristic values. A frequently used heuristic for path-finding is the Manhattan distance, which basically sums absolute values of the differences of the positional coordinates. The g value can be, in the case of occupancy maps, the occupancy probability. The f value simply corresponds to the sum of g and h. The A-star algorithm presented below was inspired by Matthews paper about A-Star [Mat02] and by Higgins work in [Hig02a], [Hig02c] and [Hig02b].

1. Let P = starting point.
2. Assign f, g and h values to P.
3. Add P to the open list. (Now P is the only node in the open list).
4. Let B = the best node from the open list (best node has the lowest f-value).
   (a) If B is the goal, then the path has been found so quit.
   (b) If the open list is empty, then no path could be found so quit.
5. Let C = a valid node connected to B.
   (a) Assign f, g and h values to C.
   (b) Check whether C is in the open or closed list.
      i. If so, check whether the new path has a lower f-value.
         A. If so, update the path.
      ii. Else, add C to the open list.

(c) Repeat step 5 for all valid children.

6. Repeat from step 4.

2.4.6 Obstacle Aviodance

Avoiding obstacles in path-finder problems for known environments is necessary so that an agent can have a certain space for error and mistake. E.g. for robots in indoor environments one would want to calculate a path that will be able to carry the robot through narrow spaces without being trapped. The following equations can be used as a fast and reliable solution to the problem for occupancy grid maps.

$$P(occ_{x_i,y}) = \frac{1}{4} * P(occ_{x_{i-1},y}) + \frac{1}{2} * P(occ_{x_i,y}) + \frac{1}{4} * P(occ_{x_{i+1},y}) \tag{2.4}$$

$$P(occ_{x_0,y}) = \frac{2}{3} * P(occ_{x_0,y}) + \frac{1}{3} * P(occ_{x_1,y}) \tag{2.5}$$

$$P(occ_{x_{n-1},y}) = \frac{1}{3} * P(occ_{x_{n-2},y}) + \frac{2}{3} * P(occ_{x_{n-1},y}) \tag{2.6}$$

These equations should be applied, in the case of a two dimensional map, to both rows and columns. Equation 2.4 should be used for the general case where the cells in the map are located inside the map borders. Equation 2.5 should be used when the cells reside within the very first row or column and equation 2.6 should be used when the cells reside on the very last row or column of the map.

2.4.7 Information merging

Information merging for occupancy grid maps was previously descibed by equations 2.1, 2.2 and 2.3. These equations will merge occupancy maps so that probabilities are matched against each other and adjusted appropriately. For instance, consider the case where one agent is 90% (occupied) sure that a specified location is occupied and another agent has a 30% (open) belief that the same position is occupied. These two values will be weighed against each other. The result can easily be derived to approximately 80% (occupied). Thus, the concluding remark is that the space will remain occupied with a lower probability value.

2.4.8 Expected Visibility

Expected visibility in environmental mapping for multiple cooperative agents is the approximate measurement of which points in the map that will be explored after the sensory sweep is performed by an agent. It is an attempt to predict future outcomes. Based on expected visibility an agent may eliminate some of the available way-points of other agents by distributing its intent to cover the specified way-points. Hence, agents will not attempt to explore way-points that have a high expectation of visibility. In [BFM00] an adaptable heuristic is presented. This heuristic is calculated based on frequency of distance to obstacle sensor sweeps.

2.4.9 Probabilistic Mapping Methods

The research performed on probabilistic mapping mainly concentrates on systems that don't have the ability to fly. Such systems are rovers and other land oriented robots. The main goal of most exploration strategies are to minimize the time consumed to seach the environment. However, there are cases when time can be traded for factors such as safty or energy. This section will discuss the probabilistic approaches for mapping environments in multi-agent systems.

2.4.9.1 Frontier Based Exploration

Yamauchi [Yam98] developed the theory of frontier based exploration. The key idea of this theory is to gain as much information about the world by moving to boundaries between unknown territory and known territory. Frontier regions are defined as the boundary that separates known territory from unknown territory. The use of evidence grids was proposed because of their known and simple integration procedure of maps from several different agents into one global and shared map. The contents of evidence grid maps are a for each grid cell a probability value whether or not that particular cell is occupied. The cell contents are classified in the environment into three different classes:

1. Open
2. Unknown
3. Occupied

The open class is represented by probabilities lower than the initial prior probability. The unknown probability is represented by probabilities equal to the initial prior probability.

The environment is always unknown when starting the exploration. The occupied class is represented by a probability higher than the initial prior probability. Usually the initial prior probability is set to 0.5. This value was proven successful in the experiments performed in [Yam98]. Once the frontier cells are located they are collected into frontier regions. These regions are then used for exploration waypoints. Each agent uses a greedy algorithm to select an appropriate frontier to visit. This greedy algorithm is purely based on the distance from the agent to the frontiers. Thus, the closest frontier is always selected. In MAS each agent is responsible for a separate map. Whenever a new frontier is visited by an agent, the agent creates a local map with the newly accumulated knowledge. This local map is then merged and distributed to the other agent units in the system.

2.4.9.2 Bid Based Exploration

In [SAB00] Simmons, Apfelbaum, Burgard, Fox, Moors, Thrun and Younes present a MAS approach that uses coordinated mapping by constructing agent specific bids and having a centralized unit assign tasks to winning bids. The bids are constructed of each agent's cost and possible information gain for reaching a certain frontier cell. Frontier cells was introduced in [Yam98] and they depict positions close to unexplored regions. Each agent is reporting its information about the environment to the centralized unit. The centralized unit then merges and propagates the map information to other agents in the system. Each time an agent receives an updated map it will place bids on the available frontier cells. This mapping algorithm was successfully tested on land oriented robots which explored an indoor environment.

2.4.9.3 Visibility Based Exploration

In [BMS02] and in [BFM00] a probabilistic approach that basically scatter the agents so that frontier cells and cells in their vicinity are not covered, visited, several times by the same or different agents. This approach will reduce the time for exploring the map as well as it will in an intuitive way will have multiple agents explore different areas of the map. In order for the agents to rationally select appropriate waypoints in a map, it is essential to keep track of which areas of a map that have already been visited or is about to be visited. This is accomplished by utilizing occupancy grid maps to represent the environment [Mor88]. These grid maps are used locally by every agent. The agent maps are then merged into one global map, that later can be propagated to all the agents. In order to choose which waypoints or targets each agent will be assigned to visit, the cost of getting to the target is computed against the probability of occupancy in the map for each frontier point. The cost and the target point's utility will then decide which agent that is assigned the task. The utility for each target point is calculated using a simple heuristic that states that an agent can cover larger areas if it visits open spaces than if it visits spaces such as narrow passages or small spaces. Thus the agents tend to explore open large areas before small areas.

CHAPTER 3

UAV SIMULATOR

The UAV simulator consists of two parts:

1. Server
2. Client

The server has a graphical interface where the simulation can be monitored under weak real-time constraints. The server basically serves as the simulation rule base and environment for the clients. The client implements the intelligence of each UAV agent. The client/server model allows for researchers to implement their own UAV agent clients using any kind of programming language that supports standard network communications (TCP/IP).

This chapter will describe the process of writing clients that can communicate with the server, how the server processes its commands and how the server and the client communication and connection models are designed.

3.1 Server

The server application serves as a unified environment for the client agents. The server specifies the rules that each agent has to follow. Speed and fuel usage are some of the properties that needs to be controlled by the server. It uses the TCP/IP protocol, with

a multi-threaded message buffer, so that messages and commands that are sent within short time-frames are processed in order.

3.1.1 Visual Representation

The server's visual representation is built upon OpenGL, a graphics library commonly used for games and simulations. The server provides a texture based visualization that has several different environmental type representations such as mountains, rocks, walls, sand, snow and grass. The server uses height-maps, which consists of a grayscale image, to model height differences within the simulation. Figure 3.1 is a screen dump of the server's visual representation. Here we can see the different terrains in action.

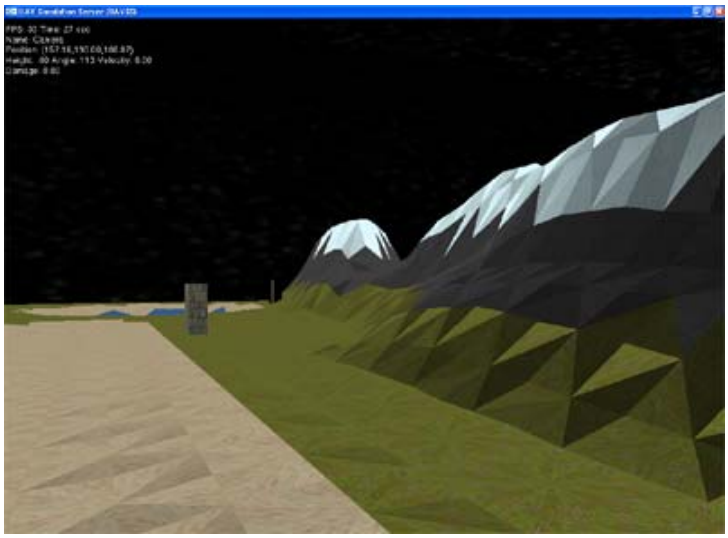

Figure 3.1: Visual representation of the server environment

3.1.2 Agent Connections

The agent connections are established using the TCP/IP protocol. This implies that the server is able to accept connections from any computer on the Internet. The server mainly consists of four components:

1. The connection listener, which performs the important task of detecting incoming connection attempts.
2. The command server, which is spawned by the connection listener for every new connection. Its task is to receive control commands from the agent. E.g. change direction, increase velocity and so on.
3. Agent data, which stores each agent's properties. These properties are altered either through the command listener or by the simulation environment.
4. The simulation environment, which updates the GUI (Graphical User Interface) and performs weak real-time updates to the agent data.

In figure 3.2 the main components and its flow of actions are illustrated.

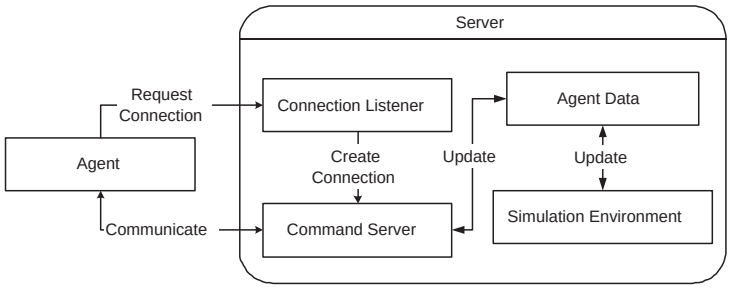

Figure 3.2: The four main components of the UAV simulation server

The flow of actions are as follows:

1. An agent requests to connect to the server.
2. The connection listener within the server detects a new agent connection attempt.
3. A new connection is spawned and a unique command server is instantiated within the server.
4. The agent send commands and requests to the command server through the established connection. Such commands could be to change angle of direction, check available fuel, check current velocity and height. Note that, the server never sends any data to the agent unless the agent requests it first.
5. The command server updates the agents internal data representation so that it reflects the current situation.
6. The simulation environment also influences the agent's data properties. Hence, it applies its changes on the agent.
7. The simulation environment updates the GUI.

The abovementioned process is repeated for every new command that the client agent sends to the server.

3.1.3 Communication

Figure 3.3 illustrates how the communication between the server the clients is implemented. The server receives control commands from the client via a TCP/IP connection and it sends communication messages, for collaboration and cooperation, to agents using a UDP/IP connection. The messages are organized according to the FIPA standard.

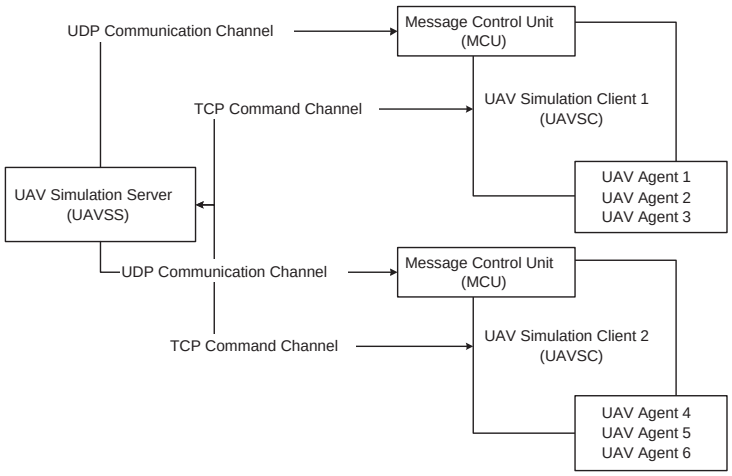

Figure 3.3: Server communication and command channels

3.1.4 Control Commands

The commands available for controlling each UAV agent can be divided into three categories.

1. Action commands
2. Request commands
3. Communication commands

The action commands are used to perform actions on the environment. Typical action commands would be to drop a bomb, increase height, change direction, increase speed and so on. The request commands are only used for the client to keep its environmental and internal states known. Typical request commands would be current position, current damage, distance traveled and so on. The communication commands are used to send customized messages to other agents or teams of agents. Table 3.1 shows the action commands that are currently available in the server environment. Table 3.2 defines the

request commands that are available and table 3.3 shows all communication commands. Please refer to Appendix A for a detailed description of each one of these commands.

Table 3.1: Server action commands

Command	Description
Init	Initializes the agent
Move	Moves the agent
Close	Terminates the agents connection
Drop Bomb	Drops a bomb
Team	Set agent to be a member of a team

3.1.5 Teams

To create agent teams in MAS simulations one can use the team command described above. Once a set of agent has been assigned a team they are able to transmit messages within this team by using the team communication command. Hence, team work, collaboration and coordination will be performed on a message based basis. Teams are represented internally by identification numbers corresponding to the Greek alphabet.

Table 3.2: Server request commands

Command	Description
Position	Request current position
Height	Request current height
Enemy	Request enemies within sight
Damage	Request current damage status
Angle	Request agent's orientation
Velocity	Request agent's velocity
Fuel	Request fuel meter status
Distance	Request agent's distance traveled
Line of Sight	Request information about obstacles in line of sight
Team members	Request agent's team members
Radar radius	Request agent's radar radius
Laser sensor	Request a 360 degree laser sensor
Height map	Request height map currently in use

Table 3.3: Server communication commands

Command	Description
Communicate team	Sends message to team
Communicate unit	Sends message to single agent

3.1.6 UAV Properties

Each UAV agent have a set of properties that can be changed based on the simulations that is being performed. Some properties may be changed dynamically other may only be changed prior to compilation of the server. The properties that are static are the ones that determine the rules in which each agent can relate their dynamic properties. For instance, maximum speed, minimum flight speed, maximum height, maximum acceleration and so on. Typical dynamic properties are the ones that can be retrieved and altered by the request and action commands respectively. That is, the interface towards the dynamic properties is the control commands.

3.1.7 Positioning

The positioning of agents is derived from the size of the map. The position of each agent is deterministic. Thus, it can be retrieved at any time and it will always be accurate. One can see the positioning and localization of each agent as a perfect global positioning system.

3.1.8 UAV Types

There are currently three different agents that can be invoked by a client. These are:

1. Normal
2. Scout
3. Bomber

The normal UAV agent has no extra characteristics. It performs the basic actions of an UAV agent. The scout agent has an enhanced radar attached to it. This radar may be used to detect enemy building and units. The bomber unit is as the name imposes an attack unit. It is equipped with two bombs that can be dropped upon enemy targets. This section will discuss in detail the differences and features of each one of the agents.

3.1.8.1 Normal

The normal, non enhanced UAV unit, have all the capabilities provided by the command set in Appendix A with the exceptions that it may not request a enemy command and it may also not perform the bomb command. If these commands are invoked nothing of value will be returned to the agent by the server.

3.1.8.2 Scout

The scout has the same features, or command set, as the normal type. However, the scout is also allowed to search for enemy units by using the enemy command. The enemy command will return, if enemies are within the radar range, a set of enemies and their positions. The radar range is a static property.

3.1.8.3 Bomber

The bomber has the same features as the normal type. In addition it is equipped with two bombs. The number of bombs is a static property. The bombs may be dropped on

enemy targets to permanently deactivate them. Once the bombs are all consumed the agent must be re-initiated in order to get more bombs.

3.1.9 Enemy Units and Landmarks

In this section the available enemy units are described. There are currently two enemy units available, these are:

1. Building
2. SAM site (Surface to Air Missile)

The building can not imply any damage to a UAV agent unless the UAV agent collides with the building. The second enemy unit, the SAM site, can destroy UAV agents by launching missiles.

3.1.9.1 Building

The building is a server generated agent that is may be located anywhere in the environment. The building cannot give damage to UAV agents unless the agents intentionally or unintentionally collide with it. The building can be destroyed by UAV bomber agent.

3.1.9.2 SAM Site

Enemy SAM sites have the capability of launching missiles towards UAV units and their weaponry. Thus, if a UAV agent decides to drop a bomb on a SAM site, the SAM site may launch a missile towards the bomb and destroy it. The SAM sites have a certain reload time. This time is a static property. Also, the SAM sites have an unlimited amount of missiles. SAM sites cause damage based on a random algorithm. There are static properties that may be altered to change the impact of each missile.

3.1.10 Damage and Danger System

Damage implied on both UAV units and enemy units are determined by static properties in the server application. In the case of SAM sites the damage are randomly generated. Each unit or agent possess it own damage function in the system. Hence they can easily be updated and altered to fit specified simulations.

3.2 Client

The client application is the interface in which UAV agents are created. The UAV agents are modeled according to the CxBR paradigm. The client is completely written in Java. Hence it is uses an interpretive execution style. The main benefit of Java is that it is platform independent. The client uses a TCP/IP connection to connect and perform commands on the server's environment. It also has a UDP/IP listener that receives FIPA messages and distributes them accordingly. The client mainly consists of the following components:

1. Agent and team control
2. CxBR framework and control
3. Path-finding
4. Debug facility

The agent control basically enables the client to create UAV agents of selected types. The team control is an interface to control in which team agents should reside within for communicational purposes. The CxBR framework is the base of all UAV agents, it a highly reusable package that contain control components for managing and administering each CxBR agent. The client also provides a user interface for experimenting with path-finding as well as debug facilities in the form of textual lists for the most common modules.

3.2.1 Visual Representation

The client's visual representation is built upon Java's Swing components, a component library commonly used for Java applications. Figure 3.4 is a screen dump of the client's visual representation. Here we can see the different modules in action. There is a map section in figure 3.4, which is used to draw the agent's occupancy maps and their merging progress, is located in the mid-right section. The map section also provides an interface for a path-finder, which is also used internally by the CxBR UAV agents. In the mid-left section, of figure 3.4, agent specific and dynamic properties are presented. The top section of figure 3.4 represents the CxBR control, command control and message control. The bottom section in figure 3.4 represents team and agent controls.

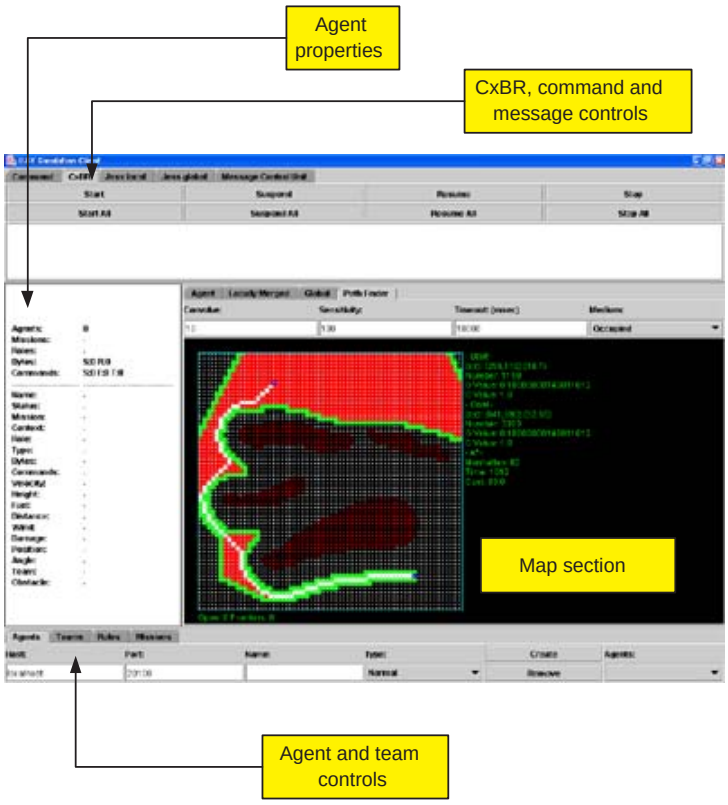

Figure 3.4: Visual representation of the client interface

3.2.2 CxBR Framework

The CxBR framework used for modeling the clients was ported from an existing C/C++ version of the same framework. The framework developed here is based on the review given earlier in this thesis. However, a few improvements are presented with the Java version:

1. Multi-threaded execution
2. Enhanced control set
3. Third party inference engine

The most significant one is that the Java CxBR version is multi-threaded. This means that each agent that is operating within the client executes its context rules and scripts in its own thread. This significantly increases the response times and performance of the client application and its agents. The second novelty is that there is a function control set for the execution state of agents. Agents can easily be started, stopped, suspended and resumed by assigning the control functions to GUI components. The third novelty is that the inference engine provided with the CxBR agent component is now a third party tool that can read and interpret CLIPS like expert system rules and facts.

3.3 Terrain Editor

To generate new maps with units and terrains for the simulation environment all that is needed is a piece of software that can draw grey-scale images and save them in raw file format. The raw file format will save the exact grey-scale value for each pixel in the drawing. Thus, providing a rich set of values that can be used to draw the different terrains. Table 3.4 illustrates which grey-scale that should be used to draw the available terrains.

Table 3.4: Grey scale terrain reservations

Terrain	From	To
Mountain	221	255
Hill	181	220
Basic land	156	180
Road	146	155
Water	100	145

This terrain modeling approach is known as height mapping. Height maps can be used for other purposes as well, e.g. wind modeling, sound intensity modeling and noise modeling. Table 3.5 shows which grey-scale values that should be used when buildings and other units such as SAM sites are desirable.

Table 3.5: Grey scale unit reservations

Unit	From	To
Building	11	40
SAM site	0	10

The major benefits of utilizing height maps are obviously its simplicity and effectiveness. Figure 3.5 illustrates a height map that was generated in a basic non-commercial graphical tool. The map contains the terrain types available, three buildings and three SAM sites.

The final rendering of this height map within the simulation environment is shown in Figure 3.6. The environment here is rendered using simple lines without any textures and lighting effects. The SAM sites are shown as white blocks with two spheres surrounding its detection and attack radius. Regular buildings are shown as bigger white blocks.

3.4 Simulation Scenarios

Based on the aforementioned server and client capabilities many interesting simulation scenarios may be derived. This section is included to give the reader a general idea of some of the main and most upfront scenarios. These scenarios are all based on multi-agent solutions.

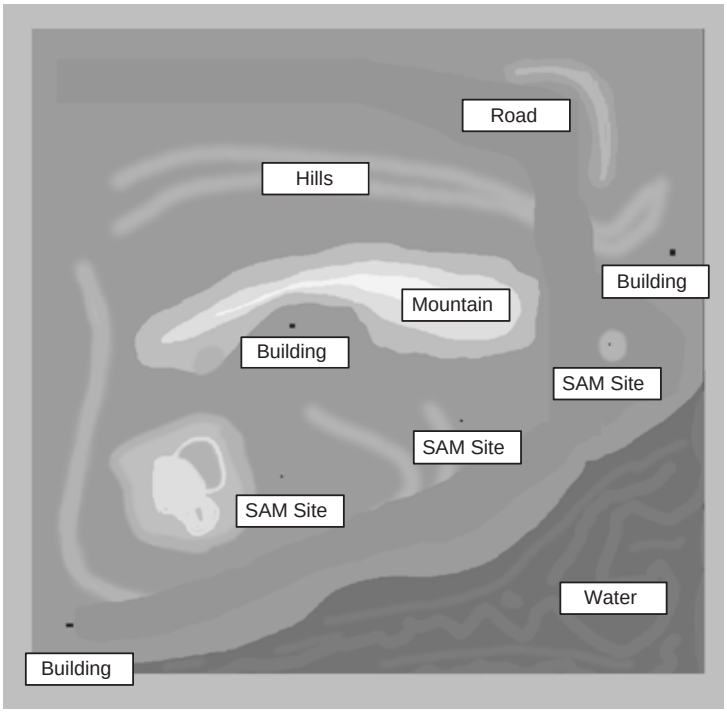

Figure 3.5: Height map modeling

3.4.1 Seek Scenario

In this scenario a set of scout UAV agents departure from the UAV base. These agents have as a mission to find enemy landmarks, such as buildings or SAM sites, as fast as possible. The main issue in this scenario is the fact that the agents must cooperate and coordinate their actions so that map spaces are not visited several times. In this type of scenario as well as in other scenarios each agent must maintain its own path-finding algorithm so that planning of navigation is performed in a secure and reliable way. The agent can distribute the locations of hostile units among each other so that danger points can be avoided when planning for routes. This scenario may end once the agents have covered the whole map or when all the landmarks have been identified.

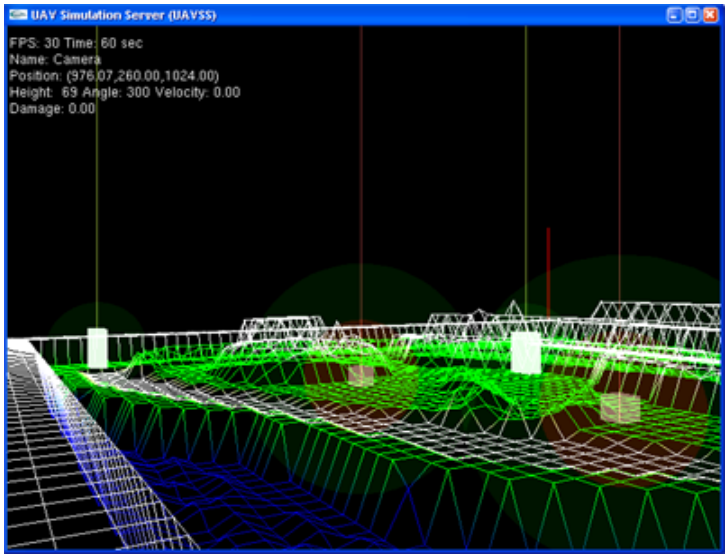

Figure 3.6: Height map rendering

3.4.2 Seek and Destroy Scenario

This scenario basically is an extension of the aforementioned seek scenario. Once a scout has detected an enemy unit it will report the enemies location to a bomber unit. The bomber unit will, if properly designed, move toward the landmark and drop its bombs, which will eliminate the enemy threat. This scenario may end whenever all enemy threats are eliminated. Remember that a SAM site may need several bombs in order to be successfully eliminated. This is due to the fact that SAM sites may disable UAV bombs that enter its range of attack.

3.4.3 Environmental Mapping Scenario

In this scenario, any type of agent may departure from the UAV base. The objective in this scenario is to rapidly map the environment. This scenario is closely related to the seek scenario. However, agents do not have to worry about enemy threats.

CHAPTER 4

MULTI-AGENT ENVIRONMENT EXPLORATION

Environmental exploration for MAS in open areas, such as those in which UAV's normally operate, must have some algorithm that enables coordinated waypoint choices for each agent. In this section a frontier based approach is presented for choosing appropriate waypoints by using occupancy grid maps and local path-finding algorithms. We will also see how a safe path can be generated by using sensitivity factors and convolving factors for static environments.

4.1 Local Path-Finding in Occupancy Maps

As mentioned before, occupancy grid maps provide good management of uncertainty. Occupancy grid maps are also, due to their probabilistic nature, very easy to merge and integrate with other occupancy maps using equation 2.1, 2.2 and 2.3. Also, we have seen that occupancy maps easily can be convolved, by applying equation 2.4, 2.5 and 2.6, so that static obstacles in the environment are avoided. So how is occupancy grid maps used in path-finding algorithms such as the aforementioned A-star algorithm? This will be answered in the upcoming sections.

4.1.1 Applying A-star

Before applying A-star on an occupancy grid map, one has to decide which heuristic function that should be used to calculate the h value for the nodes. This is fairly easy. A simple Manhattan distance function will do the job. The next thing that needs to be considered is what g value is to be used to calculate the cost for the nodes. In an occupancy grid map the g values are represented by the probabilities in the map. Consider the situation in figure 4.1, here we can see the initial start and goal points as well as obstacles and movable spaces.

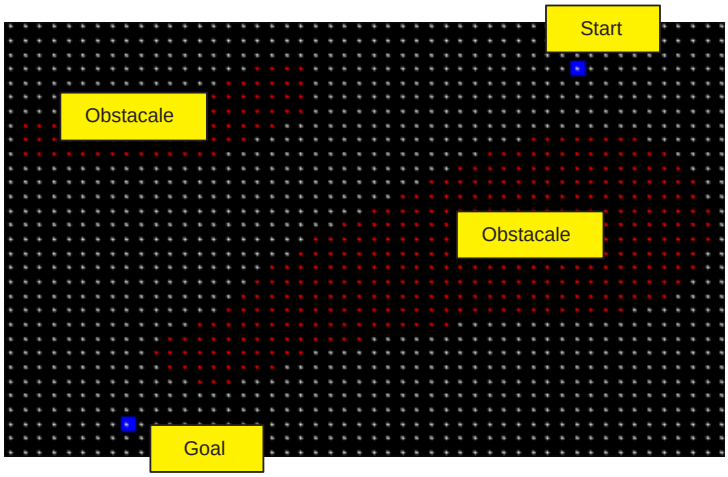

Figure 4.1: Initial path finding status

Applying the A-star influenced path-finding algorithm to the start and goal points in figure 4.1 results in a path depicted by figure 4.2.

As we can see in figure 4.2, there is a collision with an obstacle in the planned path. The reason why the path finding algorithm produces this unacceptable result is because no pre-processing of the occupancy probabilities have been performed. The heuristic

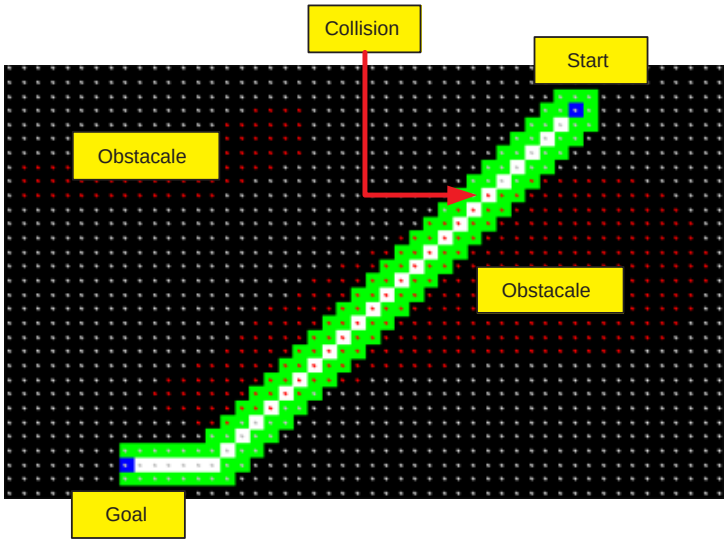

Figure 4.2: Unacceptable path generation

weighs more than the costs. Hence the heuristic is the main influence of search direction. This problem can easily be solved by multiplying a sensitivity (scaling) factor to every probability value in the occupancy map.

4.1.2 Sensitivity Factor

By multiplying each occupancy probability by a sensitivity factor, and assigning the produced values as costs, in the path finding algorithm the costs have more of an influence in the search algorithm. Figure 4.3 illustrates this with a sensitivity factor of 100. As we can see there is no collision with obstacles in the generated path.

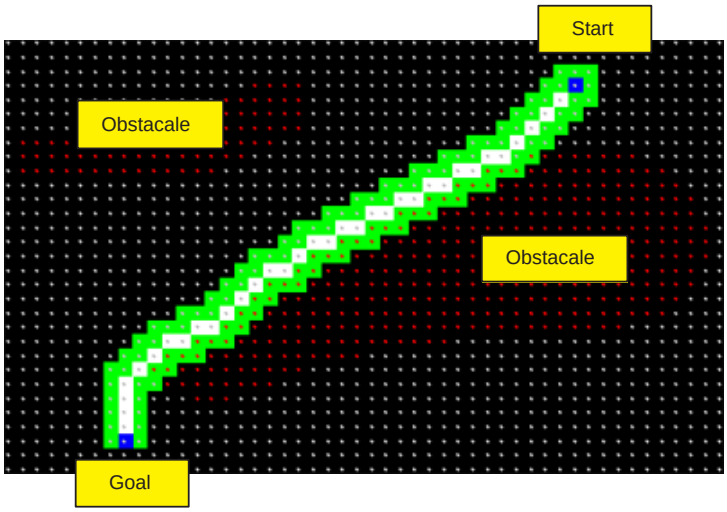

Figure 4.3: Collision free path

4.1.3 Convolving Factor

As figure 4.3 depicts, there is no collision in the generated path. However, having an agent's path planned that close to an obstacle may incur a risk of collision anyways due to real-world constraints, erroneous positioning or sensory noise. To solve this problem a convolving factor is introduced. The convolving factor simply depicts how many convolutions that should be performed on the occupancy probabilities. As we can see in figure 4.4, convolving the map produced a safer path by the path finding algorithm. Figure 4.4 was produced with a sensitivity factor of 100 and convolve factor of 3.

4.2 Cooperative and Collaborative Mapping

In order for agents to cooperate and collaborate in the mapping process they have to select frontier points so that expected visibility is satisfied properly. One would want an

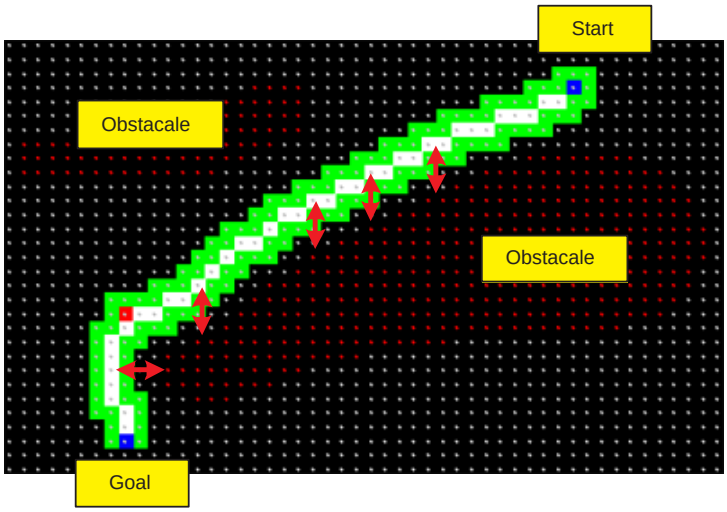

Figure 4.4: Safe and collision free path

agent to select a frontier point to visit, and explore, based on the other agents currently selected frontier points. In this section we will discuss a set of local frontier selection algorithms as well as the fact that allocating these frontiers will influence the choice of frontier waypoint the other agents in the system. The latter can be viewed as a resource allocation problem.

4.2.1 Frontier Selection Algorithm

Selecting a frontier point can be done in many different ways. In this section we will look at two possible selection algorithms:

1. Greedy selection
2. Lowest cost selection

The greedy algorithm is based on the shortest distance to a frontier point while the lowest cost selection algorithm is based on the cost for moving towards a certain frontier.

4.2.1.1 Greedy Selection

Greedy selection of frontier points is solely based on the distance towards each frontier point. The frontier point that lies in the closest distance of the current agent location will always be chosen. This approach is easy to implement and it is also one of the fastest algorithms one can use for these kinds of problems. The search time for a frontier point is based on the number of available frontiers. The list of frontiers only needs to be iterated once in order to determine the frontier point. It is classified as an $O(n)$ algorithm.

Note that the distance to a frontier point may or may not be the optimal point in terms of generated path cost. For instance, obstacles between the frontier point and the agent location may cause the path finder to generate a longer path than if a frontier was selected in an obstacle free path.

4.2.1.2 Lowest Cost Selection

This approach is a bit more complicated and slower, in terms of performance, than the aforementioned greedy selection algorithms. However, the problem that occurs when obstacles are present in the greedy algorithm is avoided here. The idea is to choose frontier point based on the lowest costs for moving there. Hence, optimal paths, e.g. using A-star, for each frontier point must be found and compared against each other to determine the frontier point with the lowest cost. However, once a frontier is found using

this algorithm it is guaranteed to be the one with the lowest cost and will therefore be always be the best choice of path. The A-star search is $O(n^2)$ by its pure nature. Hence, this frontier selection requires significantly more processing power compared to the greedy algorithm.

In the case where there simply are too many frontier points to find costs for, one can select a set of frontier points that are closest to the current agent location and find the costs for these. By limiting the cost value generation to these frontier points only one can increase the performance significantly. However, this approach cannot guarantee that the best path is always chosen.

4.2.2 Resource Allocation

Once an agent have selected a frontier point to visit it is important to allocate this frontier point and its vicinity points as the agent's resource. This is done to decrease the amount of overlapping environmental mappings performed by the agents. This implicitly forces agents to collaborate. In this section we will discuss two types of resource allocation which may be performed on multiple agent environmental mapping application domains.

1. Heuristic based
2. Sensor based

The first resource allocation algorithm is based on a heuristic that is re-calculated whenever new mappings are performed by an agent. This approach has been discussed in [BFM00] and also in sections 2.4.8 and 2.4.9.3. Please refer to these references for more information. The second resource allocation algorithm is the main focus of this section. It is based on the range, or functionality, of the agent's sensor.

4.2.2.1 Sensor Based

The sensor based approach relies on the functionality, or maximum coverage, of the sensor. If such value is available it can assist in the estimation of expected visibility by allocating the frontier points in its range. It is this value that decides the behaviour of a team. This value may be overestimated or underestimated. If overestimated the agents in the team will map the area in more scattered way. If underestimated the agents in the team will move to frontier points in less scattered way. Another way of perceiving this value is that overestimating the value for an agent will allocate more resources for that particular agent. And underestimating the agent will allocate less resources for that particular agent.

The main advantage of using this allocation algorithm is that it requires very little communication. Basically a team of homogeneous agents, with equal sensor range, only need to provide each other with its current frontier point selection. Hence, each agent in the system will have a list of already allocated frontier points. The filtering of available frontier points in the local map can then be performed onboard each agent in the system based on the already known sensor visibility range and the list of allocated frontier points.

4.3 Algorithm for Team Mapping

In the algorithm below it is assumed that the list of frontier points is generated for every loop. It is also assumed that the allocated frontier list is updated whenever the agent receives allocation information from other agents in the team. The allocated frontier list preferably use a hash table structure with agent names as keys and frontier points as values in its representation.

1. Let F be the list of available frontier points

(a) If F is empty then quit

2. Let AF be the list of allocated frontier points
3. For each frontier point P in AF
   (a) Remove P from F
   (b) Remove vicinity frontiers of P from F based on current estimation value
4. Select a frontier point A from F based on either lowest cost or greedy selection algorithms
5. Inform team members that A now is allocated
6. Plan path using A-star
   (a) Set stating point to be current location
   (b) Set goal point to be A
7. Move along the planned path
8. When A is reached start over at 1

4.3.1 Frontier Starvation

Frontier starvation occurs in the algorithm when there are too many agents trying to search the environment at the same time. The system gets over-crowded because of the fact that there is a limited amount of frontier points available at any instant of time. Since the number of frontiers available for the agents is dynamically changing, agents may not be able to find free frontier point. Frontier starvation generally occurs when agents are launched too close to each other in time. The impact of frontier starvation on

the algorithm is that the overall mapping efficiency is reduced. Selecting the amount of agents to search an environment is mainly limited by the frontier starvation problem.

CHAPTER 5

EXPERIMENTAL RESULTS

The experiments and the results concluded in this section are based on a multi-agent system that uses the collaboration algorithm described in section 4.3. The main goal of the experiments is to determine the algorithm efficiency in terms of time. The simulation environment descibed in chapter 3 is used to generate quantitative results in terms of time for a wide range of collaborative agents. Figure 5.1 describes the terrain map used when performing the experiments. In all of these experiments the agents use the same sensor capabilities, convolving factors (section 4.1.3) and sensitivity factors (section 4.1.2). The experiments are performed using greedy frontier selection (section 4.2.1.1) combined with the sensor based resource allocation procedure (section 4.2.2.1).

5.1 Experiment: Environmental Mapping Scenario

This experiment is based on the scenario described in section 3.4.3. It is purely a mapping scenario where no enemies and danger points are taken into consideration. The experiment was performed for the following constallation of cooperative agents:

1. One agent
2. Two agents
3. Four agents

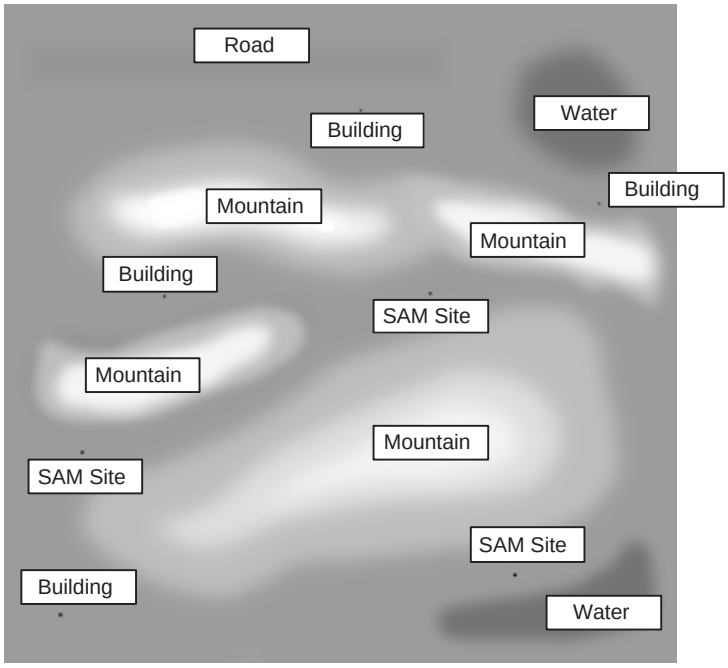

Figure 5.1: Height map used for the simulation experiments

4. Six agents

The results are represented in terms of mean values, maximum values, minimum values, speed-up factors as well as confidence intervals. The summary section provides charts and tables decribing the perfomance of each experiment. Please see Appendix B for a complete representation of the data.

5.1.1 One Agent

The results in table 5.1 was derived based on 15 mapping simulations performed by one agent.

Table 5.1: Results for one agent in the environmental mapping scenario

Result	Time (s)
Mean time	2159.73
Standard Deviation	182.41
Confidence Interval (95%)	92.31
Max	2610.00
Min	1894.00

5.1.2 Two Agents

The results in table 5.2 was derived based on 10 mapping simulations performed by two agents.

Table 5.2: Results for two agents in the environmental mapping scenario

Result	Time (s)
Mean time	1129.35
Standard Deviation	102.56
Confidence Interval (95%)	63.57
Max	1294.00
Min	1009.50

5.1.3 Four Agents

The results in table 5.3 was derived based on 10 mapping simulations performed by four agents.

Table 5.3: Results for four agents in the environmental mapping scenario

Result	Time (s)
Mean time	717.90
Standard Deviation	78.10
Confidence Interval (95%)	48.41
Max	847.00
Min	599.75

5.1.4 Six Agents

The results in table 5.4 was derived based on 10 mapping simulations performed by six agents.

5.1.5 Summary

The results given in tables 5.1, 5.2, 5.3 and 5.4 are summarized in the histogram shown in figure 5.2 and in the curve of mean mapping times in figure 5.3. In the histogram the red line depicts maximum and minimum range, the white bar represents a 95% confidence

Table 5.4: Results for six agents in the environmental mapping scenario

Result	Time (s)
Mean time	681.83
Standard Deviation	64.43
Confidence Interval (95%)	39.93
Max	823.33
Min	618.33

interval and the yellow bar represents the mean mapping time. In the mean mapping curve we can see how the mean mapping times are improved based on the number of agents in the system.

Based on the mean time data for all experiments a speed-up factor can be derived for each experiment. Table 5.5 shows the speed-up factor and the ideal speed-up factor for each experiment. The first experiment, with one agent only, serves as a base for the speed-up factor. Hence, its speed-up factor is one. As can be seen in this table, the experiments with four and six agents are more than three times faster than the experiment with one agent only. Also, the speed-up improvement tend to slowly stabilize to a state where the number of agents don't improve the mean time significantly. In fact, having too many agents collaborating in the system may decrease the speed-up factor due to communication limitations and frontier starvation (see 4.3.1).

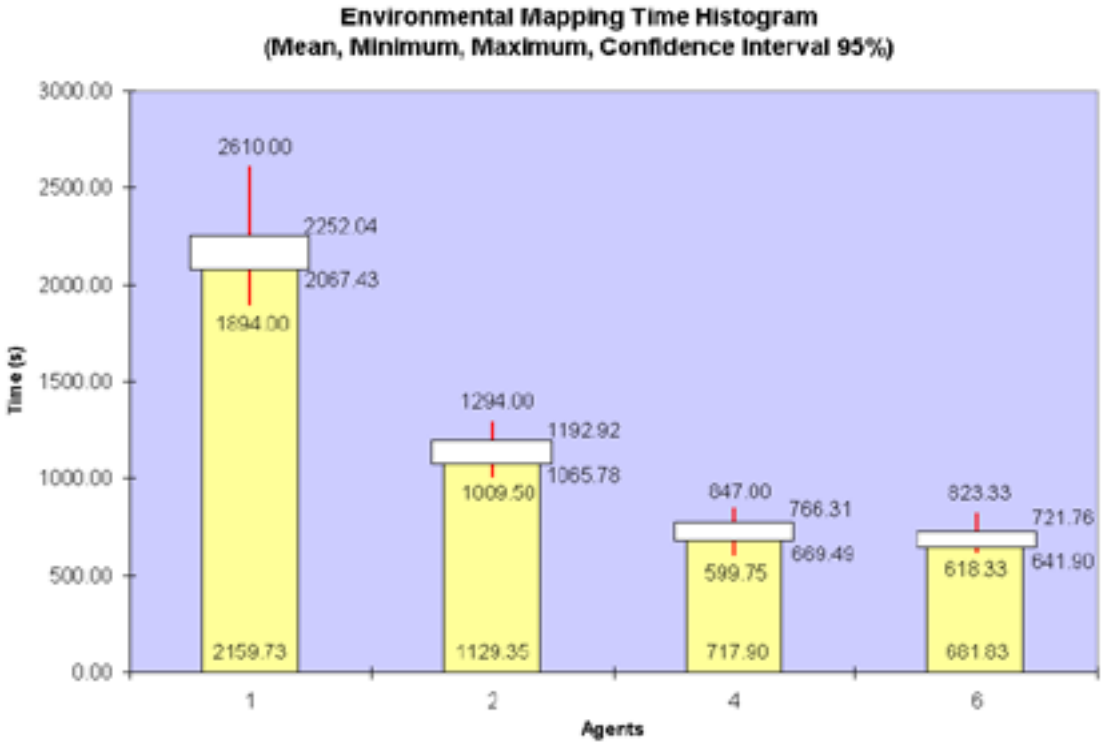

Figure 5.2: Environmental mapping histogram (Confindence interval 95%)

Table 5.5: Speed-up factor in the environmental mapping scenario

Agents	Speed-up	Ideal Speed-up
1	1	1
2	1.91	2
4	3.01	4
6	3.17	6

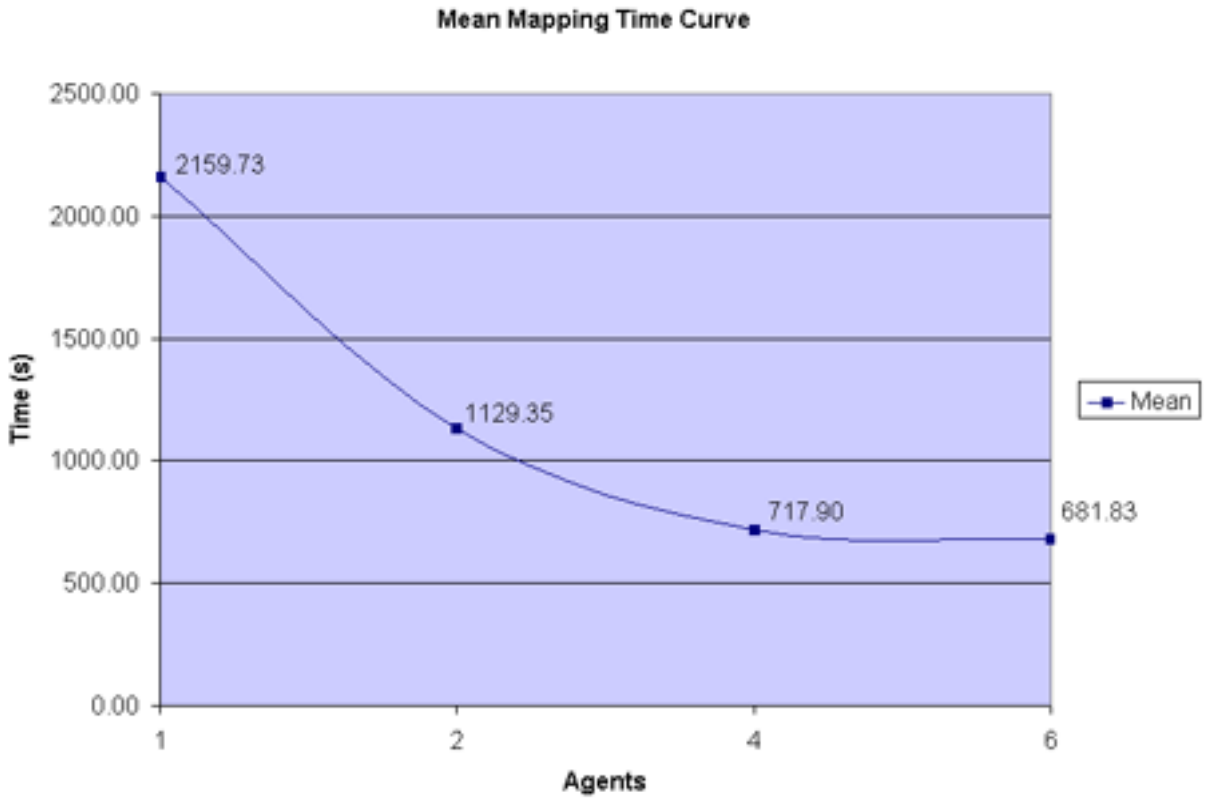

Figure 5.3: Mean mapping time curve

CHAPTER 6

CONCLUSIONS

UAV units and their history as well as a brief description of their many modern day implementations have been introduced in this thesis. One of the most interesting tasks for UAV implementations is autonomous mapping of environments in team collaboration and coordination domains. To efficiently search for landmarks or simply map an environment in a system consisting of multiple UAV units it is beneficial to use flexible agent frameworks that can handle contextual information and react to this information accordingly. Three different types of agent framework methodologies were presented and their key features were weighed against each other. The CxBR paradigm was proposed, implemented and utilized as a base agent framework for UAV agents mainly because of its simplicity and expressive design features.

To simulate the agent behaviours, and to gather quantitative measurements, a simulation environment was created. The simulation environment was described in detail in this thesis. It basically consists of a server and a client. The client represents the UAV agents and their behavioural framework while the server is a general environment that can be configured according to the simulation that is asked for.

The environmental exploration problem was proposed to consist of the choice of internal agent map representation, path finding, obstacle avoidance, information merging and expected visibility for resource allocation. A probabilistic approach that makes use of occupancy grid map representations was presented to solve the problem of map representation choice as well as information merging. A path finder, tailored for occupancy

grid maps, was developed not only to solve the problem of planning a collision free path but also to find a safe path by using convolving factors. A simple expected visibility solution, based on sensor capability, was proposed to determine each agent's allocated resources. An algorithm that incorporates all of the exploration problems was proposed and implemented. The algorithm was tested in a simulation environment and the results were statistically evaluated. The algorithm used in a multi-agent system, with 4 agents, produced over a 300% speed-up improvement compared to a single agent system.

CHAPTER 7

FUTURE WORK

In this section ideas about future work and future improvements are presented.

7.1 UAV Simulator

The future work of the UAV simulator is provided below. The list is ordered in implementation priority with the first item having the highest priority.

1. Automated test harness
2. Aviation physics
3. Dynamic enemy units.
4. Configuration tool

7.1.1 Automated Test Harness

In order to generate statistically sound simulation results the simulator needs to generate a large number of data points. The idea is to develop a test harness that can be used to automate the generation of these data points. The next step is to automate the

generation of the spreadsheet that analyzes the data points. By creating an automated testing harness changes in algorithms can easily be statistically measured and compared.

7.1.2 Aviation Physics

There are open source aviation physics engines available to use for free. The idea is to integrate such engine into the server environment. This will make the simulation very realistic.

7.1.3 Dynamic Enemy Units

Dynamic enemy units are currently not present in the server environment. The idea is to model intelligent computer opponents that can move autonomously in the environment on the server side. This will introduce more possibilities of simulation scenarios.

7.1.4 Configuration Tool

Create a configuration tool for the server, so that static properties can be altered without re-compiling the entire server. The configuration tool can be used to generate maps and to insert enemy units in the environment.

7.2 Environmental Mapping

The environmental mapping algorithm also has a list of prioritized future work.

1. Dynamic obstacle avoidance
2. Threats and failures

7.2.1 Dynamic Obstacle Avoidance

By introducing dynamic enemy units, the algorithm for mapping an environment must be able to detect dynamic obstacles.

7.2.2 Threats and Failures

Threats are currently not handled by the agents. A simple improvement of the algorithm would be to alter the occupancy grid map whenever a threat or enemy units is detected. Failures in UAV units also needs to be considered in the algorithm.

APPENDIX A

SERVER COMMAND SET

This appendix contains all the commands that the UAV Simulation Server can process.

Init

ID: 0

Description: Initializes the agent with a name and a type (Normal, Scout, Bomber).

Parameters: (ID name type)

Example: (0 uav1 1)

Move

ID: 1

Description: Moves the agent according to the provided parameters.

Parameters: (ID velocity height orientation)

Example: (1 1.2 205 12)

Position

ID: 3

Description: Request current position of the agent.

Parameters: (ID)

Example: (3)

Height

ID: 4

Description: Request current height of the agent.

Parameters: (ID)

Example: (4)

Enemy

ID: 5

Description: Request list of available enemies.

Parameters: (ID)

Example: (5)

Damage

ID: 7

Description: Request current damage percentage on agent (%).

Parameters: (ID)

Example: (7)

Angle

ID: 9

Description: Request current angle of direction.

Parameters: (ID)

Example: (9)

Velocity

ID: 10

Description: Request current velocity of agent.

Parameters: (ID)

Example: (10)

Close

ID: 11

Description: Closes the connection to the server.

Parameters: (ID)

Example: (11)

Fuel

ID: 12

Description: Request fuel/energy status of agent.

Parameters: (ID)

Example: (12)

Distance

ID: 13

Description: Request total Euclidian distance traveled.

Parameters: (ID)

Example: (13)

Drop bomb

ID: 14

Description: Drops a bomb at the current agent position.

Parameters: (ID)

Example: (14)

Team Communicate

ID: 15

Description: Send a message to all members of the provided team ID.

Parameters: (ID FIPAPerformative teamID message)

Example: (15 inform 1 (enemy dead))

Unit Communicate

ID: 16

Description: Send a message to a specific agent.

Parameters: (ID FIPAPerformative agent message)

Example: (16 inform uav1 (enemy dead))

Team

ID: 17

Description: Assign agent to a team.

Parameters: (ID teamID)

Example: (17 1)

Line of Sight

ID: 18

Description: Request information about obstacles in line of sight.

Parameters: (ID)

Example: (18)

Team Members

ID: 19

Description: Request team member list.

Parameters: (ID teamID)

Example: (19 1)

Radar Radius

ID: 20

Description: Request the range of equipped radar.

Parameters: (ID)

Example: (20)

Height Map

ID: 21

Description: Request the height map.

Parameters: (ID GridSize Height)

Example: (21 16 200)

Laser/Sonar Range Finder

ID: 22

Description: Request a new sensor sweep for the agents laser/sonar.

Parameters: (ID GridSize)

Example: (22 4)

APPENDIX B

EXPERIMENTAL RESULTS DATA

This appendix shows the raw data collected from the experiments in this thesis. The data presented here is the actual time it took for each agent to complete its objectives. The appendix is divided into 4 sections describing experiments for a single agent, two agents, four agents and six agents respectively.

One Agent

Agent	Time (s)
1	2077
2	2030
3	2034
4	2066
5	2097
6	2132
7	1894
8	2152
9	2001
10	2052
11	2241
12	2302
13	2342
14	2366
15	2610

Two Agents

Agent	Time (s)
1	1043
2	978
3	1206
4	1217
5	987
6	1032
7	1031
8	1079
9	1149
10	1202
11	1088
12	1042
13	1172
14	1146
15	1011
16	1112
17	1284
18	1304
19	1281
20	1223

Four Agents

Agent	Time (s)
1	644
2	696
3	651
4	620
5	776
6	779
7	778
8	818
9	701
10	718
11	731
12	719
13	590
14	596
15	604
16	609
17	662
18	723
19	656
20	666

Agent	Time (s)
21	726
22	731
23	738
24	796
25	895
26	725
27	922
28	846
29	753
30	801
31	816
32	840
33	688
34	697
35	709
36	711
37	616
38	644
39	649
40	676

Six Agents

Agent	Time (s)
1	662
2	678
3	702
4	700
5	711
6	778
7	623
8	632
9	640
10	645
11	663
12	681
13	599
14	600
15	608
16	623
17	639
18	641
19	601
20	600

Agent	Time (s)
21	645
22	656
23	655
24	677
25	607
26	625
27	617
28	628
29	659
30	664
31	599
32	610
33	649
34	669
35	672
36	700
37	781
38	773
39	800
40	884

Agent	Time (s)
41	902
42	800
43	692
44	714
45	720
46	780
47	760
48	701
49	703
50	712
51	717
52	735
53	754
54	776
55	595
56	635
57	634
58	641
59	657
60	686

LIST OF REFERENCES

[BC03] Cecile Bothorel and Karine Chevalier. "How to Use Enriched Browsing Context to Personalize Web Site Access." *CONTEXT 2003*, pp. 419–426, 2003.

[BFM00] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun. "Collaborative multi-robot exploration.", 2000.

[BMS02] Wolfram Burgard, Mark Moors, and Frank Schneider. "Collaborative Exploration of Unknown Environments with Teams of Mobile Robots.", 2002.

[Clo02] B. Clough. "Metrics and Performance Measures for Intelligent Unmanned Ground Vehicles." In *Measuring the Performance and Intelligence of Systems: Proceedings of the 2002 PerMIS Workshop*, 2002.

[Def01] Office Of The Secretary Of Defence. "Unmanned Aerial Vehicles Roadmap 2000-2025." Washington DC, 2001. DoD.

[Edd97] William A. Eddy. "Scientific Kite-Flying." *The Century Illustrated Monthly Magazine, Vol. Liv. New series, Vol. XXXII*, 1897.

[Eis] Kevin Eisert. "Aerial Bombers "Designs Ahead Of Their Time"." http://civilwar.bluegrass.net/ArtilleryAndArms/aerialbombers.html.

[FIP03] FIPA.ORG. "FIPA Communicative Act Library Specification." 2003. http://www.fipa.org/specs/fipa00037/SC00037J.html.

[FWW93] Tim Finin, Jay Weber, Gio Wiederhold, Mike Genesereth, Don McKay, Rich Fritzson, Stu Shapiro, Richard Pelavin, and Jim McGuire. "Specification of the KQML Agent-Communication Language – plus example agent policies and architectures.", 1993.

[GA94] A. J. Gonzalez and R. H. Ahlers. "A Novel Paradigm for Representing Tactical Knowledge in Intelligent Simulated Opponents." In *International Conference of Industrial Engineering Applications and A.I. and Expert Systems*, 1994.

[GA96] A. J. Gonzalez and R. H. Ahlers. "Context-Based Representation of Intelligent Behavior in Simulated Opponents." In *Computer Generated Forces and Behavior Representation Conference*, 1996.

[GA98] A. J. Gonzalez and R. H. Ahlers. "Context-Based Representation of Intelligent Behavior in Training Simulations." In *Naval Air Warfare Center Training Systems Division Conference*, 1998.

[GPG00] F. G. Gonzalez, G. Patric, and A. J. Gonzalez. "Autonomous Automobile Behaviour Through Context-Based Reasoning." In *Florida Artificial Intelligence Research Society Conference*, 2000.

[Hig02a] Dan Higgins. "Generic A* Pathfinding." In *AI Game Programming Wisdom*, pp. 114–121, 2002.

[Hig02b] Dan Higgins. "How to Acheive Lightning-Fast A*." In *AI Game Programming Wisdom*, pp. 133–145, 2002.

[Hig02c] Dan Higgins. "Pathfinding Design Architecture." In *AI Game Programming Wisdom*, pp. 122–131, 2002.

[HK96] J. Hertzberg and F. Kirchner. "Landmark-based autonomous navigation in sewerage pipes.", 1996.

[Mat02] James Matthews. "Basic A* Pathfinding Made Simple." In *AI Game Programming Wisdom*, pp. 105–113, 2002.

[ME85] Hans Moravec and A. E. Elfes. "High Resolution Maps from Wide Angle Sonar." In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pp. 116–121, March 1985.

[Mor88] H.P. Moravec. "Sensor Fusion in Certainty Grids for Mobile Robots." In *AI Magazine*, pp. 61–74, 1988.

[NM00] Natalya F. Noy and Deborah L. McGuiness. "Ontology Development 101: A Guide to Creating Your First Ontology." 2000. http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html.

[Nor03] Emma Norling. "Capturing the Quake Player: Using a BDI Agent to Model Human Behaviour." *AAMAS'03, ACM 1–58113–683–8/03/007*, 2003.

[NPB95] I. Nourbakhsh, R. Powers, and S. Birchfield. "Dervish: An office-navigating robot." In *AI Magazine*, 1995.

[RG95] A. S. Rao and M. P. Georgeff. "BDI Agents: From Theory to Practice." In *Proceedings of the First International Conference of Multi-Agent Systems (ICMAS)*, San Francisco, 1995.

[RK02] Emilio Remolina and Benjamin Kuipers. "Towards a general theory of topological maps." 2002.

[SAB00] Reid G. Simmons, David Apfelbaum, Wolfram Burgard, Dieter Fox, Mark Moors, Sebastian Thrun, and Hakan Younes. "Coordination Multi-Robot Exploration and Mapping." In *AAAI/IAAI*, pp. 852–858, 2000.

[Sch01] B. Schmidt. "Agents in the Social Sciences Modelleing of Human Behaviour." In *Modeling Artificial Societies and Hybrid Organizations (MASHO–01)*, Vienna, 2001.

[SN95] Russell Stuart and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 1995.

[TBF00] S. Thrun, W. Burgard, and D. Fox. "A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, 2000. IEEE.

[TGF98] Sebastian Thrun, Jens-Steffen Gutmann, Dieter Fox, Wolfram Burgard, and Benjamin Kuipers. "Integrating Topological and Metric Maps for Mobile Robot Navigation: A Statistical Approach." In *AAAI/IAAI*, pp. 989–995, 1998.

[Toz04] Paul Tozour. "Search Space Representations." In *AI Game Programming Wisdom 2*, pp. 85–102, 2004.

[Urb00] C. Urban. "PECS – A Reference Model for the Simulation of Multi–Agent Systems." In *Suleiman R., Troitzsch K. G., G.N. (ed.): Tools and Techniques for Social Science Simulation*, Heidelberg, New York, 2000. Physica-Verlag.

[Urb01] C. Urban. "PECS: A Reference Model for Human–Like Agents." In *Magnenat-Thalmann, N., Thalmann, D. (eds.): Deformable Avatars*, Boston, 2001. Kluwer academic publishers.

[US01] C. Urban and B. Schmidt. "PECS – Agent–Based Modelling of Human Behaviour." In *Emotional and Intelligent II – The Tangled Knot of Social Cognition, AAAI Fall Symposium*, North Falmouth, MA, 2001. AAAI Press.

[Woo02] Michael Wooldridge. *An Introduction to Multi–Agent Systems*. John Wiley & Sons LTD, 2002. ISBN: 0-471-49691-X.

[Yam98] Brian Yamauchi. "Frontier-Based Exploration Using Multiple Robots." In *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 47–53, Minneapolis, 1998.