A Situation Awareness Interface for a Bi-Wheeled Industrial Hovercraft: Design, Development and Evaluation

José Gouveia Pereira Corujeira

(Bachelor)

This thesis is presented for the degree of Master of Software Engineering of the University of Madeira

Competence Centre of Exact Sciences and Engineering

Funchal – Portugal

November 2010

ABSTRACT

The intention of this thesis is to develop a prototype interface that enables an operator to control a bi-wheeled industrial hovercraft that will work within a fusion power plant if the automation system fails. This fusion power plant is part of the ITER project a conjoint effort of various industrialized countries to develop cleaner sources of energy.

The development of the interface prototype will be based on situation awareness concepts, which provide a means to understand how human operators perceive the world around, then process that information and make decisions based on the knowledge that they already have and the projected knowledge of the reactions that will occur in the world in response to the actions the operator makes. Two major situation awareness methods will be used, GDTA as a means to discover the requirements the interface needs to solve, and SAGAT to conduct the evaluation on the three interfaces. This technique can isolate the differences an operator has in situation awareness when presented with relevant information given by each of the three interfaces that were built for this thesis. Where the first interface presents the information within the operator's focal point of view in a pictorial style, the second interface shows the same information within the same point of view has the first interface but only shows it in a textual manner. While the third interface shows the relevant information in the operator's peripheral field of view. Also SAGAT can provide insight on the question to know if providing the operator with feed-forward information about the stoppage distances of the bi-wheeled industrial hovercraft has any effect on the operator's decision making.

KEYWORDS

Software Engineering, Situation Awareness, Teleoperation, Telerobotics, ITER, Pathfinding, SA Interface, Transfer Cask System, Air-Cushion Transfer System, Transportation Robot, Industrial Hovercraft.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Professor Ian Oakley, for guiding me along the thesis progression and for being an intermediary.

I would also like to thank professors Maurício Reis, Luis Gomes and Mário Cunha for helping me understand some mathematical and physics equations.

Last but not least thanks to all my colleagues and friends that had to put up with me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DoF – Degree of freedom

ATS – Air-Cushion Transport System

TCS – Transfer Cask System

IR – Information Requirement

AC – Action

SC – Screen Content

GDTA – Goal Directed Task Analysis

SAGAT – Situation Awareness Global Assessment Technique

MG – Major Goal

SG – Sub-goal

KD – Key decision

I. INTRODUCTION

Along the recent decades humanity has become aware that its actions and ways of life are contributing for the destruction of our planet's capacity to sustain life. The constant natural disasters that have been happening with more frequency and intensity in these past decades created by the by-products of the means we use to satiate our constant and increasing need for energy, have made some countries rethink in new ways to create energy in a cleaner more sustainable way. So in 1985 various industrialized nations agreed on an international project to develop fusion energy, thus the ITER project was born. Which consists in proving the viability of an electric-producing fusion power plant. A test fusion power plant is being constructed in Cadarache, near Aix-en-Provence in Southern France and is intended to be finished and operational by 2019. (ITER Organization)

Figure I-1: Aerial view of the ITER construction site

Due to the hazardous nature of the environment inside the fusion power plant no human can go inside to diagnose the system, do maintenance (repair or substitute faulty components) or refuel the fusion reactor, thus operations must be handled remotely through a control room using robots. One such robot is the Transfer Cask System that serves to move cargo (fuel, reactor components) around the fusion reactors facility. With a size similar to an autobus it is composed of three parts, the cask, where the cargo for transportation is stored, the pallet, where the cask sits on and the Air-cushion Transfer System (ATS) that is the transportation robot itself.

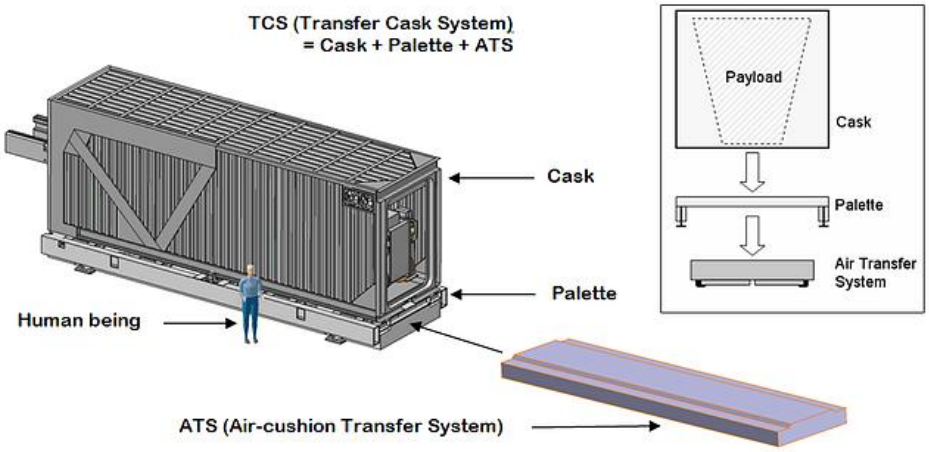

Figure I-2: Hovercraft dimension comparison to a human

The ISR (Instituto de Sistemas e Robótica) Lisbon is developing the transportation robot and its automation systems as part of the Portuguese responsibilities in the ITER project. While the functionality for a human to control the transportation robot in case the automation system fails is being developed at UMa (University of Madeira), by being divided in two projects, one that will focus on the creation of a simulator and physical manual controls for the operator. And this project which will focus on the development of three graphical user interfaces prototypes to provide an augmented situation awareness capability to the teleoperation activity of the operator. The challenge consists in providing the operators the right information, in a precise way so they can be fully aware of each situation and make the necessary decisions.

As such a thorough consideration in relation to the uniqueness of controlling a robot remotely must be taken into account when developing the interfaces. Since, it has to convey information about the environment where the robot is working as if the operator was there, without creating more complexity or more difficulty to the underlying operating task.

On this light, the field of situation awareness will be the basis for the design concepts, as well as the development and evaluation of the interfaces. Where it gives a way to understand how operators perceive the world, then process that information and make decisions based on the knowledge of the projected reactions that will occur in the world in response to the operators decisions.

I.1. GOALS

This thesis intends to:

- Present a detailed understanding of the known knowledge in teleoperation concepts and situation awareness applied to teleoperation;
- Provide a plausible list of information requirements that any interface implementation for controlling the ATS needs to satisfy in order to provide the operator with adequate situation awareness.
- Design an interface, based on teleoperation and situation awareness concepts, that allows a human operator to safely and successfully control the transportation robot if the automation fails, presenting all the necessary information adequately to enable the operator to make the best decisions and take the most adequate action at any given time.
- Show a possible prototype implementation of three interfaces that take the same elements identified in the design but presents them in a different arrangement. Where the first interface presents the information within the operator's focal point of view in a pictorial style, the second interface shows the same information within the same point of view has the first interface but only shows it in a textual manner. While the third interface shows the relevant information in the operator's peripheral field of view.
- Conduct evaluation on the three interfaces, which shall follow the situation awareness SAGAT method, with the intention to isolate the differences an operator has in situation awareness when the presented relevant information is showed in different points of view and in different representations. As well as provide insight on the question to know if providing the operator with feed-forward information about the stoppage distances of the bi-wheeled industrial hovercraft has any effect on the operator's decision making.

I.2. CONTRIBUTIONS

The contributions this thesis wants to give the scientific community are:

- Elaborate a critical information requirements list that shows which information is relevant to for the operators so they can have a good situation awareness.

- Giving an example of how design can be done using the situation awareness requirements concepts and encompassing the integration of situation awareness information in the final design.

- A method on how to evaluate these types of interfaces.

- Insight on how different methods of information representation affect the situation awareness of an operator.

- Contribute to the ITER project with a viable interface solution for the manual control of the ATS.

I.3. THESIS OVERVIEW

This thesis is divided into the following chapters: chapter II is the state of the art, here an overview of the ITER project is done, also introduces the concepts of teleoperation and situation awareness; chapter III will describe the development of a game that served as a practical way to experiment with the concepts of situation awareness and also describes the design process done to come up with a feasible interface prototype model that would encompass the requirements; chapter IV talks about the implementation of the interface prototype, telling the changes done to the TUIO client protocol 1.1 for communication, the implementation of the pathfinding feature and other functionalities and the overall final look of the interface; chapter V talks about the evaluation process to the interface introducing the user test methodology; chapter VI gives the conclusions found, referring limitations that occurred in the thesis.

II. STATE OF THE ART

"The ideal engineer is a composite ... He is not a scientist, he is not a mathematician, he is not a sociologist or a writer; but he may use the knowledge and techniques of any or all of these disciplines in solving engineering problems."
—N. W. Dougherty (1955)

This chapter is divided into three major sections where the first section focuses on giving an overview of how the ITER project was created and what it consists of. It also introduces the working environment where the situation awareness interface developed in this project is going to be used. Finally it provides a description of the transfer cask system that will be controlled through the situation awareness interface.

The second section introduces teleoperation and telerobotic concepts, giving an overview of what types of telerobotics exist, also what considerations must be taken when designing such systems and common problems during design these systems.

In the third section the concept of situation awareness is described, presenting the human information process, and various aspects of situation awareness. The relation between situation awareness and teleoperation is discussed. Finally a description of how situation awareness requirements are captured and evaluation are conducted is provided.

II.1. ITER PROJECT

Here an overview of the ITER project is given how it came to be and in what it consists of. The physical structure and functioning of the fusion reactor will be presented, as is the description of the vehicle which will serve as the transport for fuel and replacement components of the reactor.

In 1985, during Geneva Superpower Summit various industrialized nations had a discussion on new, cleaner, sustainable sources of energy, in the end of it they agreed on an international project to develop fusion energy for peaceful purposes. Thus the

ITER project was born, with the initial signatories being the former Soviet Union, USA, European Union and Japan and later they were joined by Republic of Korea (in 2003) and India (in 2005). (ITER Organization)

The "ITER is a large-scale scientific experiment intended to prove the viability of fusion as an energy source, and to collect the data necessary for the design and subsequent operation of the first electricity-producing fusion power plant" (ITER Organization). Also it is intended to give an understanding to each ITER member on how to produce its own fusion energy plant. To accomplish these goals the ITER member nations agreed to share every aspect of the project.

The location for the ITER facility and its adjoining buildings is at Cadarache, near Aix-en-Provence in Southern France. (ITER Organization)

Figure II-1: ITER Installation

The ITER reactor key components include the Tokamak reactor, Transfer Cask System. These are described below.

II.1.1.Tokamak Reactor Description

The Tokamak reactor has a height of nearly 30 meters and weighs 23000 tons. It has forty four ports distributed throughout three levels: 18 upper ports, 17 equatorial ports, and 9 lower ports. (ITER Organization)

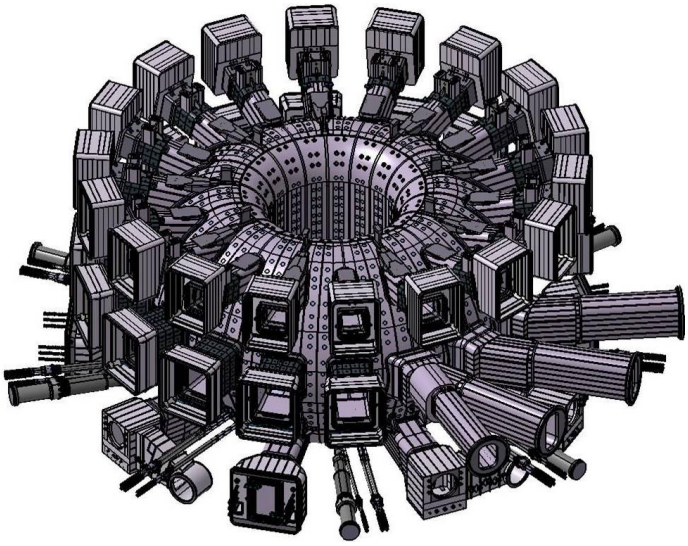

Figure II-2: Vacuum vessel

The reactor is based on the magnetic confinement concept by the same name "tokamak", which describes a doughnut-shaped vacuum vessel containing plasma in a strong magnetic field, so it cannot touch the walls. The plasma is the fuel to the reactor, which is composed of the mixture of two Hydrogen isotopes, Deuterium and Tritium, heated to temperatures in excess of 150 million ^{o}C by passing electric current through the isotopes. (ITER Organization)

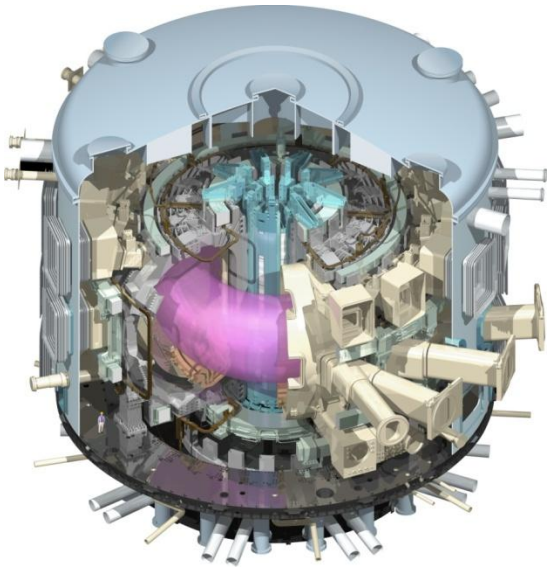

Figure II-3: Reactor

In the plasma state the Deuterium and Tritium fuse to form Helium, this fusion creates energy, 80% of that energy is within the neutron released upon fusion. Since a neutron has no electrical charge it is not affect by the magnetic fields surrounding the plasma, in contrast the atom of helium produced has a charge and therefore stays confined in the magnetic fields. The neutron is then absorbed by the walls

surrounding the tokamak, transforming the energy that the neutron contains to heat. This is subsequently dispersed by the cooling towers in the DEMO fusion plant, where it will then be used to produce steam that, by way of turbines and alternators, will generate electricity. (ITER Organization)

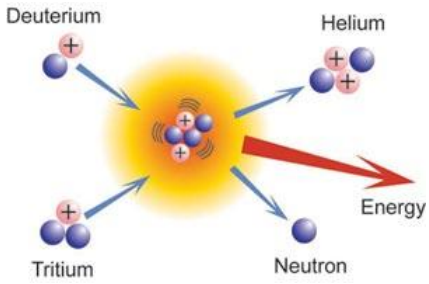

Figure II-4: Fusion reaction

DEMO fusion plant description (Layout)

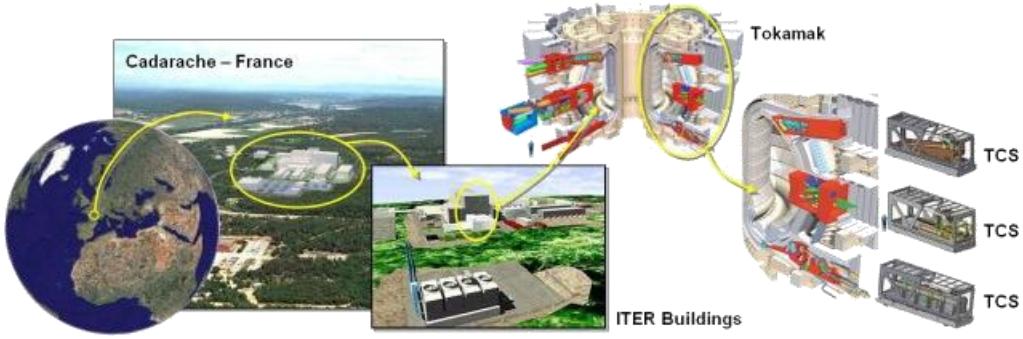

Figure II-5: DEMO fusion plant location

The Demo fusion plant is the building where the reactor is going to be, this building will have three levels the lower level, equatorial level, and upper level (ITER Organization). Each of these levels will have the layout as shown on the next figure, II-6.

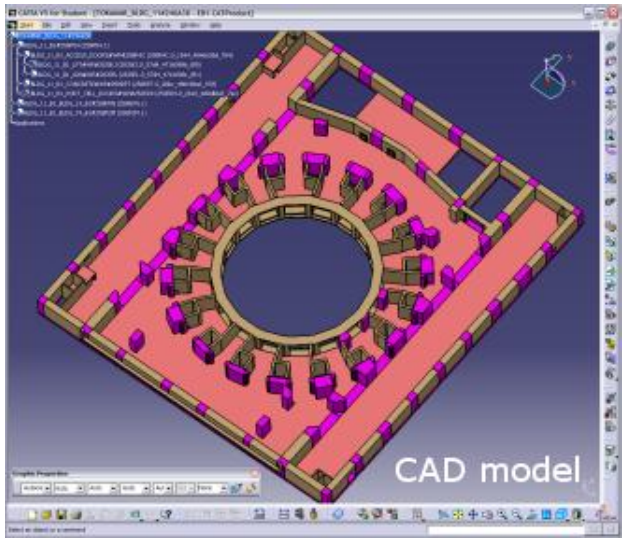

Figure II-6: CAD model of one of the level's layout

Depending on the level there will be a different number of ports to access the reactor, providing remote handling operations, diagnostic systems, heating and vacuum systems. Besides the reactor and its ports there's a storage room for storing fuel and spare components of the reactor. Since the space in these levels is not accessible to humans, due to being a hazardous environment, there has to be autonomous vehicles to transport fuel and/or components within those locations. Also all operations (changes, inspections, repairs of components) to the reactor will be made by remote handling. (ITER Organization)

II.1.2.Transfer Cask System (TCS) Description

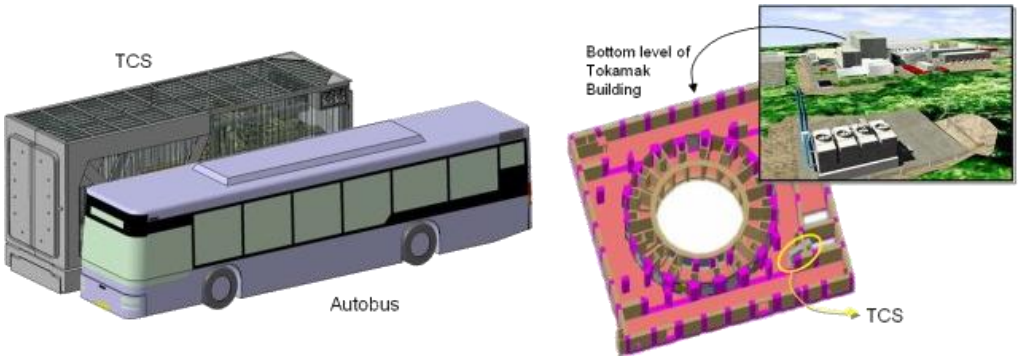

Figure II-7: Comparison of the TCS to an autobus and its resting position within the DEMO

The TCS enables transport of components and fuel around the DEMO during maintenance operations, its size is similar to an autobus being 8.5m long, 2.62m wide and 3.7m tall. The TCS is composed of three parts: the cask that has the payload (components or fuel), which can have a load of 10 to 100 tons; the palette, which the cask sits on enables the Air-cushion Transfer System (ATS) to move the cask around; the Air-cushion Transfer System (ATS) is the part that provides mobility to the TCS, it's able to guide itself through the space and dock autonomously and weighs around 1ton. (ISR Lisbon)

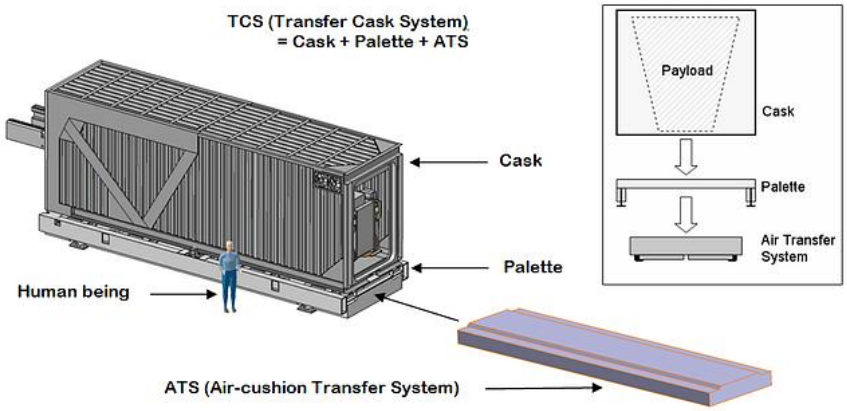

Figure II-8: The TCS decomposition and a comparison of size with a human being

Air-cushion Transfer System (ATS) Description

The ATS is 8.5m long and 2.6m wide, has various air-cushions underneath it that provide lift making it a low friction hovercraft. The system is driven by two fully independent 2 degrees of freedom wheels, that means that each wheel can rotate 360 degrees and move back or forward independently of the other wheel, this enables the ATS to navigate the tight corners easily and make the docking also an easy task. The wheels provide a top speed of 10cm/s. Also the ATS may have sensors incorporated to enable the automation system to know where it is. At the moment of writing this thesis it can only be assumed that this will be some kind of range finder. (ISR Lisbon)

Automation

As stated previously the ATS is autonomous, that means that it has to plan a path from a pair of start/goal locations considering that the trajectory needs to be collision free, also with the least amount of steering manoeuvres and preserve a safe area around it. After the planning the ATS has to follow the path in the minimum amount of time and monitoring its position to be able to synchronize with the entire DEMO system, i.e. doors, additional TCS units, docking gates. Another important task is that the ATS has to do is make a successful dock, so it can load or unload casks. Work on these activities is taking place at ISR/IST on Lisbon. (ISR Lisbon)

Manual Navigation (during system failure)

In a critical operation such as this, there has to be a fall-back system for the TCS to enable a human operator supervising the normal automation system to take control, in case of failure consequently, the TCS has a manual override system. With this system the operator has to be able to, as mentioned above, supervise the automation's actions, enabling him/her to see the trajectories that the automation plans and the movement of the TCS. For if is necessary to make corrections to return the TCS to the defined trajectory and helping it make corners safely. In an eventual fail the override system must enable the operator to take full control of the TCS, to be able to stop it or navigate it through the DEMO, be able to deliver loads safely and taking it back to the storage for repairs.

This system has to overcome many difficulties: it will only be used by the operators as a last resort safety measure, meaning it may only be used sporadically; the system will be integrated in the DEMO control room, which provides a noisy environment for the operator, being it audible, visual or other; the wheel configuration and functionalities are not standard, meaning the layout of the controls most encompass this and provide

an intuitive way for the operator to manoeuvre the TCS; it has to be precise both in the terms of control and display, due to the gap between the TCS and side walls being at maximum a short 30cm and less when cornering; the inertia in the system has also to be taken into account since the TCS can weigh 100tons and is sitting on top on air-cushions it's movements will not be sharp.

II.1.3. ITER Project Section Summary

As was previously stated the ITER project started as a means for various countries to research and trade information on how to efficiently produce energy from a fusion reactor. Also a description of the prototype reactor was given, as well as how the nuclear fusion reaction occurs. Then the TCS prototype was introduced and a description of what it is and how it works was given.

II.2. TELEOPERATION/TELEROBOTICS

This section provides a brief description of teleoperation and telerobotics. It introduces the two major types of telerobotics existent at the moment, it also describes the peculiarities of a teleoperation environment.

A definition to teleoperation is that it enables a human operator to perform a continuous mechanical action at a remote location where the operator cannot go, be it because it is a dangerous and hostile environment or because it is simply in a remote location from the operator. So teleoperated mechanisms (be it robots, arms, vehicles, etc.), as stated by Bejcsy, "serve to extend, through mechanical, sensing and computational techniques, the manipulative, perceptive and cognitive abilities into an environment that is either hostile to or remote from the human operator" (Bejczy, 1999). They are mostly used to do servicing, maintenance or repair work.

The subsequent part of this section will focus on the two major types of telerobotics arm-hand and vehicles it will also discuss teleoperation environment and the issues in task visualization.

II.2.1.Telerobotic Arm-Hand

Telerobotic arm-hand systems allow operators to remotely perform mechanical actions as if the operators were using their real arms or hands to manipulate an object

on a remote location. Being first introduced in the nuclear industry, where afterwards they have been used in space and medical industries as well. (Bejczy, 1999)

For the teleoperated robotic arm-hand to enable the human operator to successfully control with dexterity an object, it must perform as closely as it can to a human arm or hand. So, since a human arm and hand are powerful mechanical tools and also have delicate sensory organs that can receive information (contact force, torque, touch information) from the outside world, enabling humans to perform actions that require complex positions, rate of movement or applied force, the teleoperated robotic arm-hand must try to emulate those as faithful as it can. Therefore this means the operators need to experience the same transfer of forces and torques from the handling device to the object and the other way around, as they would with their own arms and hands, so they can have a perfect control over how they make the contact between the handling device and the object. This also means that a teleoperated robotic arm-hand has normally more than 4 degrees of freedom to resemble as closely as it can to the movements made by a human arm or hand. (Bejczy, 1999)

There are two main types of robotic arm-hand systems the "Bilateral force-reflecting master-slave manipulator system" and the "Hand-controller system". For more information on these refer to (Bejczy, 1999).

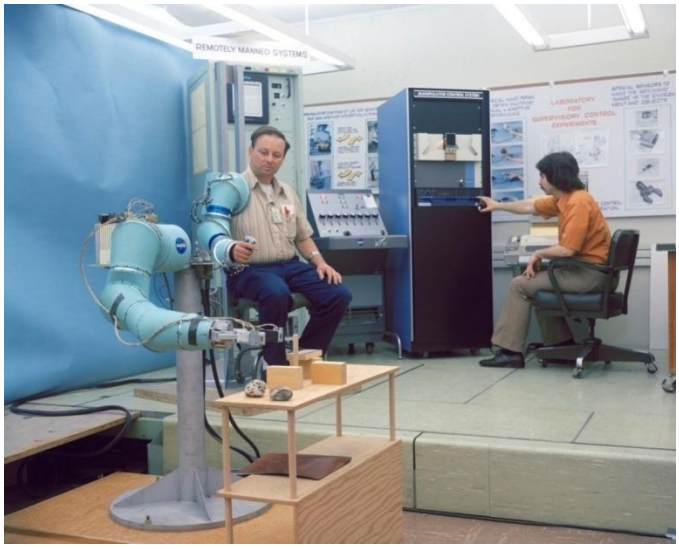

Figure II-9: Example of a telerobotic arm-hand system used by NASA

Figure II-10: Telerobotic arm-hand system in medical operations

II.2.2.Telerobotic Vehicle

Teleoperated robotic vehicle is any kind of vehicle that is controlled from a remote location by a human operator. A simple example of this is a remote controlled toy car, airplane or boat.

Figure II-11: Remote control toy car

Teleoperation of robotic vehicles has a strong emphasis in navigation and exploration in remote locations, for space and underwater science research. They are also used by the military to do reconnaissance, surveillance and target acquisition (RSTA) or explosive ordnance disposal (EOD). There are four types of teleoperated robotic vehicles, Unmanned Ground Vehicle (UGV), Unmamned Aerial Vehicle (UAV), Unmanned Water Vehicle (UWV) and Unmanned Underwater Vehicle (UUV). (Trouvain, 2006) (Durfee, Kenny, & Kluge, 1998)

Figure II-12: Talon UGV does reconnaissance or weapon delivery

Figure II-13: A reconnaissance small UGV for identifying targets around corners or tracking them.

Figure II-14: Hawk a military UAV for field reconnaissance

These can have two types of control, one where the vehicle has an autonomous behaviour to aid the operator and the other where the vehicle is fully dependent on the operator's actions (Ferland, Pomerleau, Le Dinh, & Michaud, 2009) In either case the operator controls the vehicle at a distance via wired or wireless connection, basing its actions on the received sensory feedback from line-of-sight visual observation or remote sensory input (such as video cameras, microphones, etc.). This brings a set of different problems to tackle, than the teleoperation of robotic arms-hands, to be able to successfully perform a task. For an operator controlling a telerobotic vehicle is in general a dual-task problem. Where the first the operator's task is piloting the vehicle (e.g. obstacle avoidance, moving from A to B, maintaining operational effectiveness,

etc.), the second problem is the payload task objectives which is the primary mission for using the telerobotic vehicle (e.g. applying sensors for surveillance, searching, delivering cargo, etc.). This in some cases leads to systems that need two operators to control the telerobotic vehicle, one for driving, and one for applying the payload. Another problem is in conditions of non-line-of-sight where the interface represents the only link between the operator and the vehicle. In these cases the feedback information necessary to provide the operator with sufficient context where the vehicle is, must be given by the display components of the interface. In general this information consists in information about the state of the vehicle, the environment and the state of the tasks, which enables the operator to effectively supervise and control the telerobotic vehicle. It has to be taken into account that the performance limiting factor becomes essentially the human-machine-interface in the conditions where non-line-of-sight happen, because the operator must completely rely on the data transmitted from the vehicle's sensors. (Trouvain, 2006)

In non-line-of-sight conditions, when the operator must completely rely on the data transmitted from the vehicles' sensors the human-machine-interface becomes essentially the performance limiting factor. (Trouvain, 2006)

II.2.3.Teleoperation Environment

One of the major concerns for teleoperation environments is enabling the operator to virtually share the same time and space as the robot. In other words the intervening artefacts (e.g. controls, displays, interfaces) used by the operator must not create complexity, nor augment the difficulty of the operating task, so the operator can work both naturalistically and skilfully. To accomplish this, operators need to have their natural reactions stimulated, where the perception given by an action executed in the teleoperation environment is observed as corresponding with the response they would expect in the real world. So the use of inherent perception-action cycles to support interactions can result in the operator not having more cognitive load, as can help the operator's more arduous process of problem solving by providing the appropriate computer support. (Tetsuo & Yukio, 2000)

Giving an example to clarify the subject imagine if you will that you have a remote controlled toy car but you can only see it through a television that is not able to show the real colour of the walls or obstacles, showing them instead in a very similar colour to that of the floor. So when you are driving the toy car you cannot see distinguish the walls and obstacles, thus not understanding why the toy car those not go in that particular direction. So television altered the perception of the world environment not being able to convey it in a way that you would expect.

II.2.4.Task Visualisation

The first teleoperation systems were developed in the mid-1940s to handle radioactive materials, in these the operator could observe the environment where he/she was performing the task through radiation resistant viewing ports in the wall. Nowadays this method is still the predominant one, where for example in the space shuttle missions a window that looks directly outside is used to view operations of a robotic arm. (Bejczy, 1999)

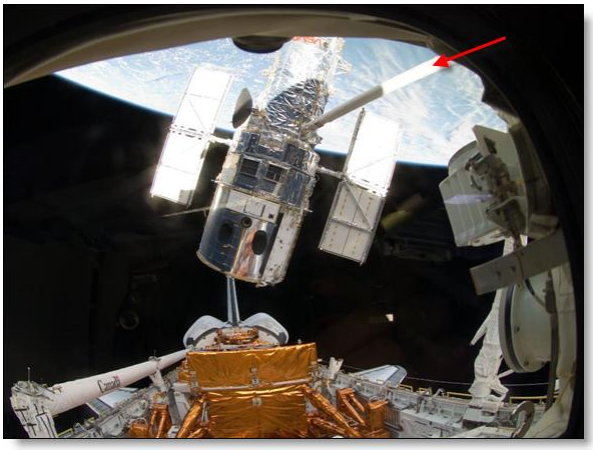

Figure II-15: Space shuttle view port to the robotic arm

But in the situations where the operator is not able to have a direct field of vision to a robotic arm or robotic vehicle the visualization of the environment is made through some kind of video feed, for example deep sea exploration in an unmanned submarine. (Bejczy, 1999)

Therefore as Bejczy (Bejczy, 1999) points out "task visualization is a key problem in teleoperation, since most of the operator's control decisions are based on visual or visually conveyed information". So the way the remote environment is displayed to the operator has a big impact on the control decisions that are going to be made. The main issue to have in mind for displaying the remote environment is that the operator needs to have a good sense of the spatial locations of the objects in the remote environment. (Drascic, 1991)

II.2.4.1.Spatial ability

The definition given by Menchaca-Brandan (Menchaca-Brandan , Liu, Oman, & Natapoff, 2007) for spatial ability is that it is "our ability to generate, visualize, memorize, remember and transform any kind of visual information such as pictures, maps, 3D images, etc." This ability has at least three components relevant to teleoperation which relate to image processing mental functions, *perspective-taking*, *mental rotations* and *dynamic spatial performance*.

Perspective-taking, also known as spatial orientation is the capability of imagining oneself seeing how an object or scene looks from different perspectives. This means an observer has to mentally go through various egocentric views or reference frames that show an object or scene within the same fixed world frame. (Menchaca-Brandan , Liu, Oman, & Natapoff, 2007)

Mental rotations, also known as spatial relations, means that the observer has the ability to mentally rotate an object to see it from different angles, maintaining the same fixed egocentric view or reference frame. (Menchaca-Brandan , Liu, Oman, & Natapoff, 2007)

Dynamic spatial performance is the perceiving and extrapolation of real motion, the prediction of trajectories and estimation of the arrival time of moving objects. (Menchaca-Brandan , Liu, Oman, & Natapoff, 2007)

Menchaca-Brandan (Menchaca-Brandan , Liu, Oman, & Natapoff, 2007) believes that spatial abilities contribute to teleoperation performance, by the use of an egocentric and exocentric frame of reference it was showed that a better performance is obtained when using egocentric frame of reference.

II.2.4.2.Visualization viewpoints

There are two visualization viewpoints in teleoperation, the egocentric viewpoint, and the exocentric viewpoint.

Egocentric Viewpoint

Egocentric viewpoint is seeing the remote location from the robot's perspective, meaning that it is utilized the robot's cameras and sensors to have a comprehension of the remote location's structure and navigating through it. This viewpoint is considered to be the best way to navigate and avoid obstacles with the robot.

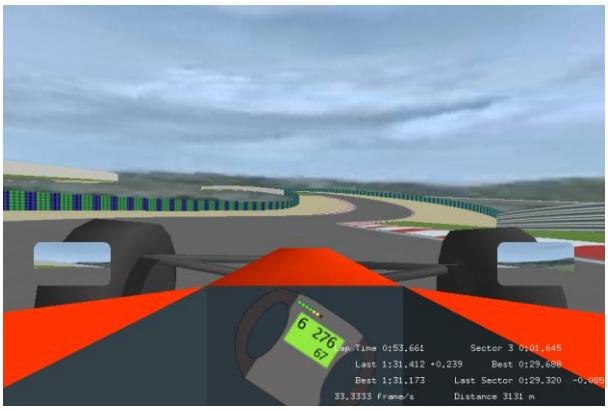

Figure II-16: Egocentric Viewpoint

The egocentric viewpoint is very extensible. By using multi-modal displays the video feed coming from the robot can be superimposed with graphical information, for example distance markers, translucent 2D maps, combining feeds from other sensors or cameras on top of the normal video feed or also a superimposed virtual reality model of the remote location. Even though all this can be done, the egocentric viewpoint still has issues of proportion where distances and sizes of objects do not match the reality, and issues of localization since the operator con only see the information that the sensors and cameras are giving, so the sense of where the robot is in the world is very limited.

Exocentric Viewpoint

Exocentric viewpoint refers to observing the remote location from an external perspective, by making use of the cameras and sensors placed throughout the environment or by embedding a 3D model of the remote location in the operator's display, thus enabling an outsider's perspective of the robot. The exocentric viewpoint is best used to give the operator a better understanding of the environment's structure.

Figure II-17: Exocentric Viewpoint

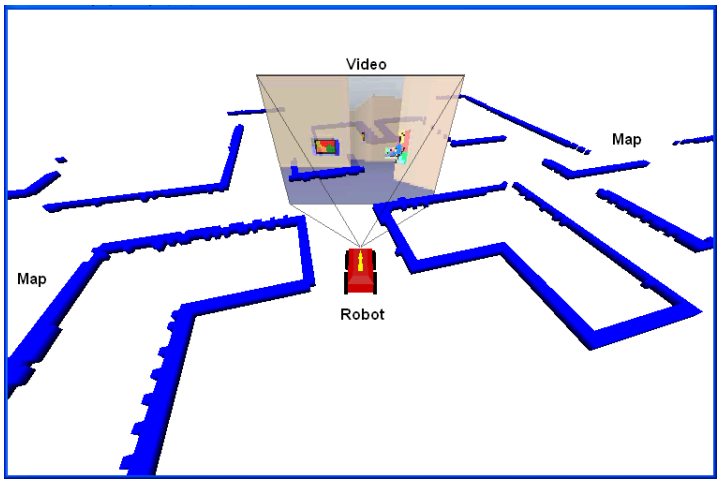

Figure II-18: Exocentric Viewpoint with camera video feed integrated (figure from (Nielsen, Goodrich, & Ricks, 2007))

In multi-modal displays that show the teleoperated robot along with its environment (the ecological paradigm (Nielsen, Goodrich, & Ricks, 2007)) normally use 2D data to create 3D structures. But difficulties emerge as the use of a fixed 2D plane video in the robot model can introduce occlusion, since the 2D plane can hide the 3D extrusion from the sensors readings. (Ferland, Pomerleau, Le Dinh, & Michaud, 2009)

It those not matter if it is being used 2D or 3D video feed the estimation of distance is difficult, but there are many factors that can influence the distance estimations in virtual environments, for example field of view. Ferland (Ferland, Pomerleau, Le Dinh, & Michaud, 2009) states that displaying the robot's model in conjunction with laser range data is a good idea.

II.2.5.Teleoperation Section Summary

A description of the various types of telerobotics was given, *Arm-Hand* and *Vehicle*, were it was described how they work, as also the different categories within those types and some challenge that they present. Also the main problems of a teleoperation environment were discussed. Finally the possible ways of how to represent to the operator the remote location where the robot is operating are described.

II.3. SITUATION AWARENESS

"In more general terms, it [situation awareness] could be described as an operators understanding of the system as a whole; as it was, as it is now, and how it is likely to be in the future." – S. A. Moyle 2005

This section will focus on what is the situation awareness, first providing an explanation of the human information process and then a formal definition of situation awareness. Also, a depiction of how humans solve complex problems is specified, as is a detailed presentation of various aspects of situations awareness. A relation between situation awareness and vehicular teleoperation is done and in the end of the section the methods for situation awareness requirements capture and evaluation are given.

II.3.1.Human information Process

Before explaining what situation awareness is, an introduction to the human information process must be done, since many of the situation awareness concepts are based on it.

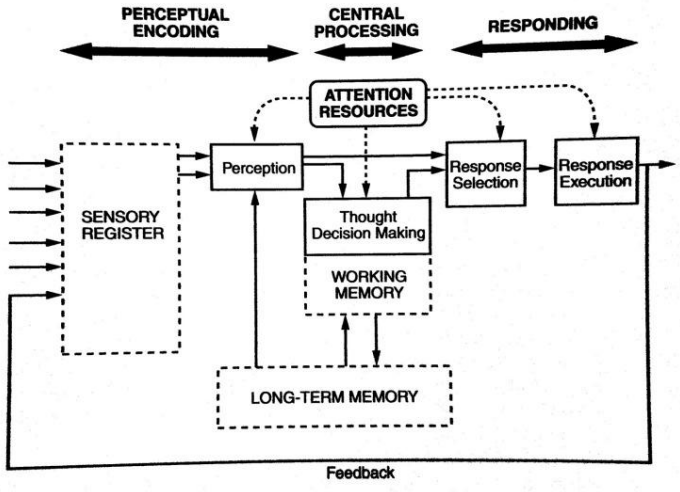

Figure II-19: Human Information Processing Model

Has can be seen the figure above shows all the aspects (perceiving, thinking about and understanding the world) that typically influence human cognition. We use the senses to gather information from the world that then is perceived with the help of the long-

term memory knowledge in order to provide a meaningful interpretation of what is happening in the world. Following the figure's schematics it can be seen that perception sometimes leads directly to the selection and execution of a response (action), but in most cases perception leads to information processing, these are "thought about" in the working memory, which can also get information from the long-term memory of past experiences to aid in the decision making or other mental activities such as rehearsing, planning, understanding, visualizing and problem solving. However, is not only retrieving information from the long-term memory that this thought process does, it also as an activity to save that processed information in the long-term memory for further use in the future, this is how we learn. Looking to the top of the figure it can be noted that almost all stages of the information processing depend on attention resources (mental or cognitive) these resources are of limited availability, being only used when necessary by each process. On important note is on the sensory part where due to attention limited resource only a few sensory channels can be selected for further information processing, for example focusing with our eyes in one part of the world and not on another. To end, note the feedback loop, representing that our actions generate information to be sensed and perceived. (Wickens, John, Liu, & Gordon-Becker, Cognition, 2003)

II.3.2.Situation Awareness Definition

Endsley (Endsley & Garland, 2000) defines situation awareness as "the perception of the elements in the environment within a volume of space and time, the comprehension of their meaning, the projection of their status into the near future, and the prediction of how various actions will affect the fulfilment of one's goals". By other words what is being expressed is that situation awareness is the understanding of what is happening in the system by the operator and the ability the operator has to envision how the system is going to react to an action. A relevant note to make is that situation awareness is goal driven and an important factor in teleoperation.

The subsequent part of this section will explain how humans solve complex problems, focusing on the aspects of situation awareness, gives a description of situation awareness applied in teleoperation. It will also discuss how situation awareness needs are captured, systems are designed and benefits are evaluated. (Moyle, 2005)

II.3.3.Complex problem solving

In a decision making environment, the operator's goal is one of analysis and problem-solving. In this environment the operator has to continuously adjust the mental path for each new information that is presented. Since situation awareness is most critical

in rich information environments that require the operator to continuously solve complex problems, this section will illustrates how a person does complex problem-solving and which design concerns must be taken into account. (Albers, 1999)

Normally an operator has a task to accomplish to do that a set of problems have to be resolved, so the operator first begins by understanding the situation. This is done by setting an immediate goal of locating the relevant information among the data around, then mentally establishing relationships within that information linking them to their real-world situations, by use of mental models and acquired domain knowledge, in the end using it to perform useful or correct actions. (Albers, 1999)

It needs to be highlighted that an operators understanding of how to perform a task cannot be used to describe other operators, this is due to the fact that the information needs and the amount of information an operator will view before making a decision based on it may in most cases not even be consistent between other operators. Therefore as Albers (Albers, 1999) states "the first step in the process of supporting complex problems solving is to help the users identify the important elements of the situation and the relationship between the elements". This can be accomplished by enhancing the way information is transmitted to the operator by identifying the knowledge that needs to be conveyed and utilizing the appropriate mechanism to convey it. By having in mind that humans can receive information from a variety of sources through a number of senses and can process that information in parallel, the possibilities for the appropriate mechanisms to convey information expand largely.

In situation awareness this means the decision maker has need of information which supports gaining a clear understanding of the problem and its possible solutions. When analysing the situation awareness needs of an operator, the focus must be on determining and defining the operator's informational and psychological needs, as the relationships between the information, and capture a complete set of the operator's goals. Since operators' decision-making ability is affected by workload, time, stress, and inexperience at solving the problem besides other factors, operators can be incapable of knowing what information may relate to the problem or even they may not consider all relevant information. Also an action is made by operators as soon as they feel confident in the decision. (Albers, 1999)

II.3.4.Aspects of Situation Awareness

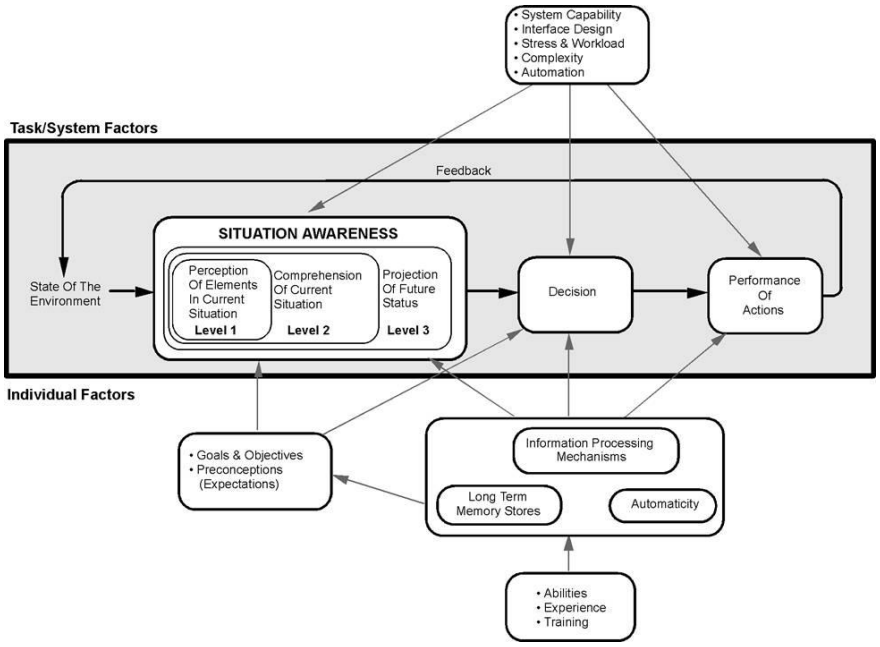

Figure II-1: Model of SA in dynamic decision making (Endsley & Garland, 2000)

II.3.4.1.Levels of Situation Awareness

Situation awareness is a three level model as defined by Endlsey (Endsley & Garland, 2000).

The first level deals with the operator's perception, what is perceived from the system and surrounding environment, the status, attributes and dynamics of task-related elements. With the perception of these cues the operator diminishes the odds of forming an incorrect picture of the situation. For example take a driving situation, the driver to avoid any accident has to be able to identify what is happening in the road ahead, the traffic, the conditions of the road, the performance of the car, if there are any incoming hazards, etc. (Endsley & Garland, 2000) (Salmon, Walker, Ladva, Stanton, Jenkins, & Rafferty, 2007)

The second level is the interpretation of the data perceived in the first level, combining, interpreting and retaining the information. This interpretation turns the data into information, that the operator comprehends is relevant in relation to the task and goals. Continuing with the driving example from above after the driver perceives what is happening at that moment he mentally takes all this despaired information creating connection between them to find out if anything that is happening will have any impact in the current driving trajectory. (Endsley & Garland, 2000) (Salmon, Walker, Ladva, Stanton, Jenkins, & Rafferty, 2007)

The third and last level is projection, it involves prognosticating the future states of the system and elements in the environment, with the help of what was perceived on the first level and understood on the second level. When the operators are capable of projecting future events and the implications that are caused by current events they are capable of making appropriate decisions. In this case from the example above if the driver had perceived that in the road ahead there are many stopped cars, so he could interpret it as a traffic jam, due to rush hour or an accident, finally the driver would identify a future state of the situation if he chooses to continue in that direction he might be stuck for a long time in the traffic jam, or maybe just changing direction and going through another path that is longer but might take less time to get to the destination. Upon that consideration the driver can make the appropriate choice. (Endsley & Garland, 2000) (Salmon, Walker, Ladva, Stanton, Jenkins, & Rafferty, 2007)

Taking in consideration level 1 of situation awareness todays systems are creating a large information gap. This is because the large amount of data captured does not automatically represent information, nor a better understanding of what's happening. More data is not equivalent to more information. Often are the times the operator feels that he/she does not know anything at all of what's going on. So the criteria for system designs has to ensure, besides providing the operator with the needed information and capabilities, that the information is provided in a way that is cognitively as well as physically usable. (Endsley & Garland, 2000)

II.3.4.2.Temporal Aspects

Situation awareness depends greatly on time, it is tightly linked to the temporal dynamics associated with events. This is due to the need of the operator to understand not only the parts of the environment that are of interest (space), but also how much time is available until an event occurs or some action must be taken (time). Therefore, time is an integral part of the second and third levels of situation awareness. As situations have dynamic natures, information changing in the current situation allows projections to be made of the future situations. (Endsley & Garland, 2000)

II.3.4.3.Decision Making in regard to situation awareness

Endsley considers situation awareness as a separate stage from decision making and action performance. This is due to several reasons, first as stated by Endlsey (Endsley & Garland, 2000) "it is entirely possible to have perfect SA, yet make an incorrect decision" and the inverse is also true. This is because even though decision making and situation awareness are integrally linked, decision making is bound by other

factors (lack of training, organizational or technical constraints, personality factors, etc.).

Likewise, a direct link between decision making and overall operator performance may not exist in many environments, for example an operator may have done a correct decision but when taking action the system may not permit it or the action may depend on external factor. So in a theoretical point of view situation awareness, decision making and performance "can be seen as distinct stages that can each affect the other in a circular on-going cycle, yet each can be decoupled through other factors" (Endsley & Garland, 2000).

II.3.4.4.Sources of information

Situation awareness is derived from information that our senses receive, visual, aural, tactile, taste and olfactory. But for remote operators, many of today's interfaces do not provide sufficient information to compensate for those cues that are normally perceived directly. Most of these system designs have the focus on the information the system provides, but it is important to take into account the information that operators also derive from other means. For example, the role of others as an important source of situation awareness information, through verbal and nonverbal communication. Operators in a team also tend to rely on each other for checking their own situation awareness. When analysing the situation awareness provided by the system, it is then important to analyse the potential cost in terms of interference with information the system gives and the operator gets via other means. It is important to note that the receiving of information is not passive, in many cases the operator is actively involved in the information the system gathers and displays. (Endsley & Garland, 2000)

So situation awareness is derived from a combination of the environment, the system, and other people as integrated and interpreted by the individual, therefore it can be stated that people are active participants in the situation assessment process. Thus, when developing systems it has to be taken into account that operators get information from various sources. (Endsley & Garland, 2000)

II.3.4.5.Situation assessment process

This process sees people picking information, weaving it together and interpreting it in a continuous and ever-changing dialogue between situations and operator's goals. Many factors influence how an operator derives situation awareness from their environment. (Endsley & Garland, 2000)

Attention and working memory

The first factors are the limited resources of working memory and attention, where the working memory can only retain 5 ± 2 items. The aspects of the situation that are processed to form situation awareness are dictated by how the operator employs attention in acquiring information. This attention can be dictated by various factors such as the perceptual salience of environmental cues, operator's learned scan patterns and information sampling strategies from the environment, goals that need to be achieved, expectations and other information already in the operator's memory. Also experienced operators tend to prioritise attention to information that they perceive is important, but this may lead to operators attending to some information and neglecting other important information. (Endsley & Garland, 2000)

The limits of working memory constrain situation awareness, since it is in here that an operator combines information, interprets it and makes projections. But experienced operators can reduce working memory load, with information prioritisation, chunking, "gistification" of information (this means an operator just encodes the relative values of information) and restructuring the environment to provide external memory cues. (Endsley & Garland, 2000)

Mental models

Experienced operators create mental models of the systems they use and environments in which they operate. These mental models are useful since they help efficiently direct limited attention, providing a way to integrate information without loading working memory and offer a mechanism for generating projection of future system states. But the use of mental models is also negative for situation awareness, as they may lead to biased selection and interpretation of information that may create errors. Also an incorrect formulation of a mental model leads to an erroneous situation awareness. (Endsley & Garland, 2000)

Nonetheless the use of mental models is considered to be somewhat (but not entirely) dependent on the ability of the individual to pattern match between critical cues in the environment and elements in the mental model. Pattern matching tends to classify situations on surface and deeper structural features. Where surface features tend to include contact information, critical cues and similar platforms, patterns, and sequences. And deep structures give importance to similar contexts and other background information. (Endsley & Garland, 2000)

Goals

Goals are an important factor in deriving situation awareness, since recognition of goals is crucial in understanding situation awareness, it is a goal-driven process. The goal-driven process directs attention across the environment seeking the information needed to fulfil a goal and goals simultaneous act as a filter in interpreting the information that is perceived. There is also in some cases a data-driven process where environmental cues that are perceived can indicate new goals that need to be active. (Endsley & Garland, 2000)

Preconception/Expectations

Situation awareness is also influenced by preconceptions/expectations. Since people may have certain expectations of what they should see or hear in a particular environment this influences how attention is directed and which perceived information is actually absorbed. So it is understandable that preconceptions /expectations may influence situation awareness negatively, as operators that have certain expectations of what is happening may misinterpret information that those not comply with their expectations. (Endsley & Garland, 2000)

Automaticity

Another important factor that influences situation awareness is the automaticity that is developed by experienced operators. The positive side of the influence of automaticity is that it reduces demand on the attention, particularly in complex tasks. But automaticity has a negative impact on situation awareness due to the reduction in responsiveness to new stimuli in cognitive processes. In such situations, attention is normally not given to information that comes from outside the routine sequence. (Endsley & Garland, 2000)

II.3.5.Vehicular Teleoperation related to situation awareness

Situation awareness in a vehicular teleoperation environment demands a specific approach, since teleoperators may have only awareness of reduced parts of the environment when compared to in local operation. Regarding this asymmetry of the two-away nature of the awareness relationship, situation awareness is better characterized by describing which aspects of situation awareness are attained by teleoperators and which aspects are lacking. Drury (Drury, Riek, & Rackliffe, 2006) describes this using the following table:

Component	Definition
Human-Robot	The understanding that the humans have of the locations, identities, activities, status and surroundings of the robots. Further, the understanding of the certainty with which humans know the aforementioned information.
Human-human	The understanding that the humans have of the locations, identities and activities of their fellow human collaborators.
Robot-human	The knowledge that the robots have of the humans' commands necessary to direct their activities and any human-delineated constraints that may require a modified course of action or command noncompliance.
robot-robot	The knowledge that the robots have of the commands given to them, if any, by other robots, the tactical plans of the other robots, and the robot-to-robot coordination necessary to dynamically reallocate tasks among robots if necessary.
humans overall mission awareness	The humans' understanding of the overall goals of the joint human-robot activities and the moment-by-moment measurement of the progress obtained against the goals.

Looking to the table four different relations can be seen, first the obvious relation between human and robot where it represents the knowledge acquired by humans of what is happening with the robot and its environment. Next is the relation between the operator and other humans this signifying the human collaboration needed to accomplish some goal. Then we there is the relation between the robot and human not to be mistaken with the first this happens when robots are autonomous and humans only intervene in some cases. Finally, the relation of robots with each other where they need to coordinate to accomplish a goal. (Drury, Riek, & Rackliffe, 2006) Some of these relationships are associated with environments that have multiple robots and people working together.

II.3.6.Situation Awareness Requirements Capture

For SA to effectively be integrated in a design, the designer needs to understand what information (as opposed to data) an operator needs for a particular task. This encompasses enabling the state awareness of the operator, the systems and the current operational situation. (Jones, Endsley, & Bolstad, 2004)

Goal directed task analysis (GDTA) is the most common method used to elicit requirements for situation awareness, addressing the need for delineating user SA requirements as part of the design process. This method is a form of cognitive task (Johns Hopkins University Department of Psychological and Brain Sciences, 2007) analysis that focuses on uncovering information needs and how that information is used to support complex decision making in dynamic and complex environments. GDTA is based on structured interviews, observations of operators performing their tasks and detailed analysis of documentation on users' tasks. GDTA consists of four steps (Cognitive Processes) (Endsley & Hoffman, 2002):

1. The identification of key decision makers. This means identifying which operators are responsible for doing a task.

2. The identification of major goals and the specification of main sub-goals necessary for meeting each of those major goals. Here the designer has to ascertain which goals each operator as to accomplish in the task.

1. Determining the key decisions needed for each goal/sub-goal.

2. Determining the SA information requirements for making those decisions and carrying out each sub-goal. These sub-goal SA requirements often focus not only on what data the operator needs but also on how that information is integrated or combined to address each decision. Providing a basis for determining what denotations the user must derive from data.

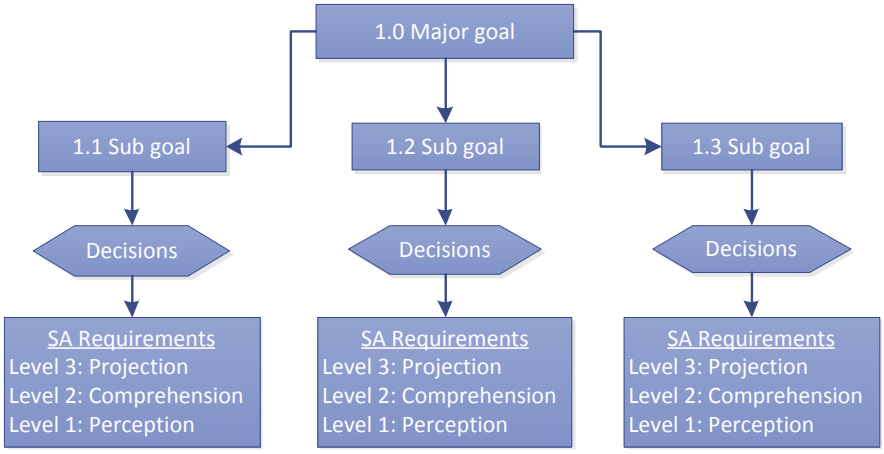

Figure II-2: Model of the GDTA

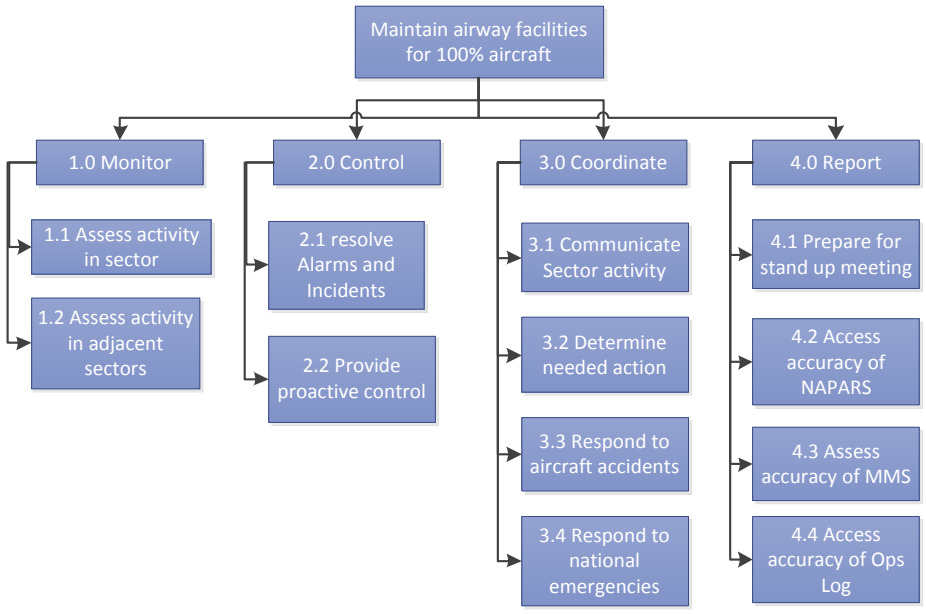

Figure II-20: Example of GDTA in and airport tower control

II.3.7.Situation Awareness Evaluation

New technologies and/or design concepts need to be evaluated based on a measure of SA, thus providing a determination of which ideas improve or degrade operator's SA.

In various systems an operator's SA and workload can be directly measured during design testing. But, even though one system design concept could be identified as superior to another in providing the operator with needed information in an easier format that conforms to the operator's needs, its benefits may go unobserved during the limited conditions of simulation testing or due to extra effort on the part of operators to compensate for a design concept's deficiencies. (Endsley & Garland, 2000)

By measuring SA directly, it should be possible to select design concepts that promote SA and thus increase the probability that operators will make effective decisions and avoid poor ones. (Endsley & Garland, 2000)

Requirements for SA Measures

Before any SA evaluation can be made, the criteria of validity and reliability must be established for SA measurement techniques. This means making sure that a metric is not just an echo of other processes, but that it measures in reality the construct it claims to be measuring. It must also not alter in a substantial way the construct it measures, as this would lead to biased data and altered behaviour. (Endsley & Garland, 2000)

Numerous theoretical viewpoints of SA have a significant impact on developing and selecting measures of SA.

First, SA is considered a *state* of knowledge about a dynamic environment, which is different than the *processes* used to achieve that knowledge. Endsley (Endsley & Garland, 2000) states that "different individuals may use different processes (information acquisition methods) to arrive at the same state of knowledge, or may arrive at different states of knowledge based on the same processes due to differences in the comprehension and projection of acquired information or the use of different mental models or schemata." So, if the metric to be used relates to how people acquire information, measures that tap into situation assessment processes should be selected. However, these will only provide partial and indirect information regarding a person's level of SA.

Second, the direct link between level of SA and the decisions made may not always be truthful. A direct SA-decision link may exist when an operator has high levels of expertise in a well-understood environment where the understanding of the current situation leads the operator to directly select an appropriate action from memory. But this is not always the case, since individuals can still make poor decisions with good SA. So it can be assumed that the relation between SA and performance (any good or bad decision made by the operator) is a probabilistic link. Therefore when using behaviour and performance measures to find the level SA, we need to have in mind that these are only indirect indices of operator SA. (Endsley & Garland, 2000)

Thirdly, SA is also fundamentally impacted by how an operator directs his/her attention in acquiring and processing information. Predominantly this happens in complex environments where multiple sources of information compete for attention. And so any design change that influences attention distribution (either intentionally or inadvertently) can have a big impact on SA. Equally, measurement techniques that influence attention distribution should be avoided, since the measuring process may change the construct. (Endsley & Garland, 2000)

Fourthly, some direct measures of SA try to detect the state of knowledge a person has of the dynamic environment. This is a substantial concern, it is not clear to what degree people report on the mental processes that make this information accessible. Therefore, careful strategies for accessing directly a person's memory stores must be employed. (Endsley & Garland, 2000)

Finally, a designer can use as measure the degree to which a system design allows the operator to approach the ideal level of SA, perfect knowledge on all relevant aspects of the situation. (Endsley & Garland, 2000)

SA Measurement

Any SA measurement must take place in the context of the task domain and needs the participation of the domain operators.

One of the most successful techniques used to measure, directly and objectively, SA is *Situation Awareness Global Assessment Technique* (SAGAT). To deploy this test participants perform a simulated task using the system design to be evaluated and at random times the simulation is frozen, with the displays of the system design hiding the information they were showing at the time, the researcher then asks a few pre-planned questions based on the SA requirements and encompassing the three levels of SA. These freezes can occur three or four times, without disrupting the flow, in a scenario trial of 20 or so minutes. SAGAT allows taking the responses given and by looking at their accuracy to measure the impact that design decisions have on SA, therefore enabling us to see how well the integrated system design performs within the actual challenges of the operational environment. It also provides designers with diagnostic information on how aware operators are of key information, how well they understand the significance or meaning of that information and how well they can think ahead to project what will happen. (Endsley & Garland, 2000)

II.3.8.Situation Awareness Section Summary

In this section a brief overview to the human information process was given to introduce the basic concepts that SA relies on. Then a full description of what is situation awareness was given, where topics like complex problem solving, levels of SA, relation between SA and decision making, SA requirements capture and SA evaluation were addressed. Also, the relation between SA and vehicular teleoperation was described.

II.4. STATE OF THE ART SUMMARY

This chapter talked about the ITER project how it came to be and what it is, where it gave a description of the fusion reactor and the TCS that enables the transport of components and fuel within the reactor.

Then the subjects of teleoperation and telerobotics were also introduced and explained, focusing on the types of telerobotics that exist and the problem of the teleoperation environments, as it also described various types of visualisation possible for a teleoperation system.

Finally in this chapter situation awareness was introduced to the reader, where an explanation of the human information process was given to lay the basis of understanding the concepts of SA. Then, the definition of SA and its aspects were stated, as was the relationship between SA and teleoperation. And the description of the methods to do the SA requirements capture and evaluation was given.

III. DESIGN

This chapter is divided into two parts. Where, the first part will address the development process of a SA game that served to learn in a practical way the concepts of SA. This part as nothing to do with the development of the SA interface for driving the ATS.

The second part delves into the design process of the SA interface for driving the ATS, here it will be talked about the requirements gathering that were obtained by the use of the GDTA and Activity Modelling (Constantine, 2008), and the prototyping of how the interface could be.

III.1. SA SIMULATION GAME

This section will talk about a simulation game done in order to test an actual implementation of SA concepts on an interface. It will talk about the concept that was behind the game, the implementation of the game and the SA analysis.

In order to understand the concepts of situation awareness in a practical situation a simple simulation game was implemented. The idea for the concept of the game was taken from the "Enhancing RoboFlag users' situational awareness" (Shankar, Jin, Adams, & Bodenheimer, 2004) and "Gaming and Shared Situation Awareness" (Perla, Markowitz, Nofi, Weuve, & Loughran, 2000) papers. Therefore the basic idea was to create a game where the player uses various robots in order to search and defuse mines that are scattered throughout a minefield. Then this idea was refined, with each robot having a field of view (defining to the radius of the robot sensors), a battery state that requires the robots to go recharge, periodically, and some commands that can be given by the player (move to, search and defuse). Also if a robot would step on a mine it would stop functioning, this would also happen if the robot runs out of battery. Finally the game includes a safe zone where robots could go to recharge.

Taking the idea explained above as a premise, the next step was to identify pieces of information and how to represent them in the user interface in order to enable a higher SA of the task that the operator has to do. With the aim of accomplishing this,

the role of an army bomb squad member searching and defusing bombs had to be imagined. This mental process enabled to get a sense of how stressful the environment is and what information is needed to accomplish the task at hand. So after that a brainstorming session was conducted and the following design concepts were identified:

- Each command should have a distinguished icon representing it
- Each robot should explicitly show which action is performing at the moment.
- Make each robot clearly identifiable.
- Each robot should have their own set of commands, these commands should be identifiable in the same way has the robots.
- Each robot has to have an indication of its battery level.
- Indicate the operator when the state of the battery of a robot is at a level that only permits him/her to send the robot back to the safe zone to recharge.
- Show in the map the mines that have been found in an alarming colour, and show the mines diffused in another colour.
- Each robot should show the progress of completion of the command given and if it was successful or not in the end.

III.1.1.Development Phase

Next, followed the development of the game, by utilizing as a tool the processing language version 1.1, due to its easiness in programing for visual contents and a quick learning curve.

The first step was to develop the mechanics of the game, such as:

- The representation of the minefield, its dimensions and boundaries;
- Representation and properties of the safe zone (the dimensions, no mines would be spawned in the safe zone and the recharge ability);
- The properties of the mines (spawning at random positions in the minefield, and exploding when stepped on by the robots or being defused);
- The functionality of the command buttons for each robot

- Move command: Press the robot's move button then click anywhere in the field to instruct the robot to move to that point.

- Search command: Press a robot's search button and the robot will stop doing the action it is currently doing to begin searching for mines (within its sensor's radius).

- Defuse command: Press a robot's defuse button and the robot will stop doing the action it is currently doing to begin defusing one mine that is within its range of action.

- The robots battery state and movement actions.

All this process of developing the game mechanics was done iteratively among these steps some SA design decisions were done. The first six iterations done were to simply create the mechanical part of the game and how it would work only after were the SA design decisions fully implemented. The following table describes the implementation changes done in each iteration to the game:

Iteration	**Major Changes**
First	Decided number of mines and robots 15 and 3 respectively; Initial position definition for safe zone and robots; Code for randomly placing mines; Early stage of the movement action for robots, applied to only one robot
Second	Colour differentiation for each robot; Playing zone delimited by a rectangle; Movement action was refined, clicking on the movement button and then clicking anywhere on the window made the respectively robot move to that location; Representation of the command buttons, with each type of command button being represented by a different polygon to be easily identified; Each set of command buttons received the same colour as the robot it commands, in order to know which buttons command which robot
Third	Implementation of searching and defusing mines functionalities; Implementation of mines' states (hidden, found, defused)
Fourth	Early implementation of robot's battery functionality; Detonation functionality of mine implemented; Visual implementation of robot movement, with robot icon changing to a triangle
Fifth	Refinement of battery functionality (each robot now has its own battery, recharge is now possible inside the safe zone); Implementation of end game conditions (defuse/detonate all mines or lose all robots); Visual implementation of all robot states (idle, moving, searching, defusing, out of order)
Sixth	Mines were visually hidden; Last refinements on the robots' battery; Implementation of robots' working space limits; Addition of keys that represent the commands for the robots

Table III-1: SA game iterations' changes

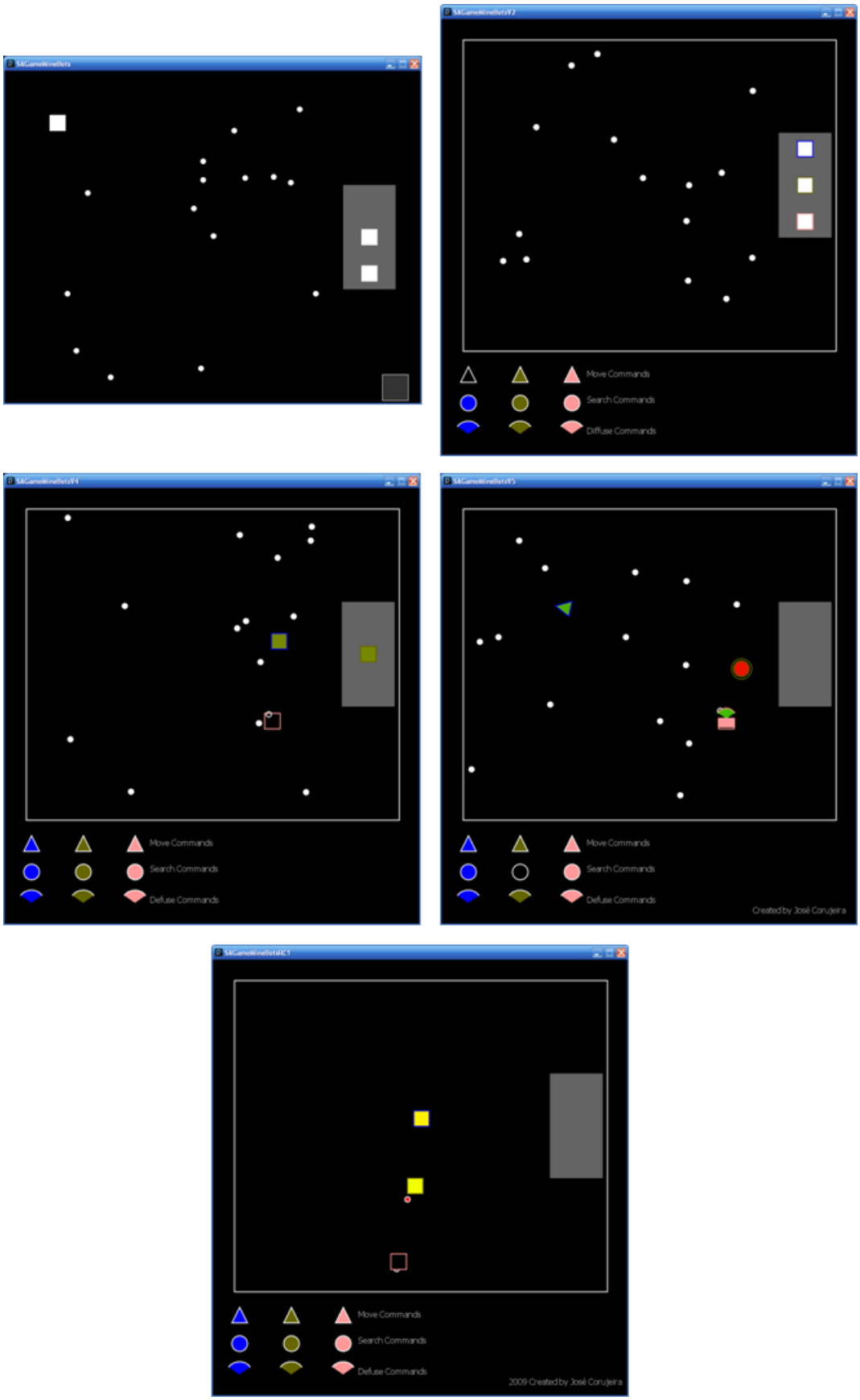

Figure III-1: Screens form various iterations. From left to right: (1) First iteration; (2) Second iteration; (3) Fourth iteration; (4) Fifth iteration; (5) Sixth iteration

The button representation for the movement command is a triangle since it gives the perception that it points the way and has a movement attitude; for the button representation of the search command e circle was used since the search is done in all directions by the robot; and the defuse command is represented as an arc, the reason for this was simply to have a different symbol than the others.

The hidden mine state is active when the operator still does not know a mine's location; the found state happens when a mine is identified by a robot on its search, hits state is shown has a red circle with a white line in order to convey danger; the last state, defused or detonated, happens when a robot steps on top of it or successfully defuses it, it is shown by a dark circle with white line, so that the user knows there is no risk about that mine.

The recharge function implemented for the robots is only activated when a robot is in the idle state.

The visual icons chosen to represent each robot state were:

- Idle is represented with a coloured square;
- Moving is represented by triangle;
- Searching is showed using a circle for the robot (the same icon used for the command button) and an out growing circumference to show the progress of the search;
- Defusing is illustrated with an arc with a rectangle below that fills up serving as progress indicator.
- Out of order is represented by a transparent square.

The reason to represent these states differently was to enable the operator to immediately identify what each robot is doing, as it was discovered in the brainstorm.

The ratios for depletion and recharging of battery are as follows:

State	Battery Ratio
Idle	-0,015% per frame
Moving	-0,04% per frame
Searching	-0,025% per frame
Defusing	-0,03% per frame
Recharging	+0,08% per frame

Table III-2: Battery depletion and recharging ratios

The bound command keys to the following robots blue, brown and pink are respectively:

		Command		
		Move	Search	Defuse
Robot	Blue	7	4	1
	Brown	8	5	2
	Pink	9	6	3

Table III-3: Respective numerical keys bound to the robots commands

III.1.2.Goal Directed Task Analysis of SA game

Before doing the final iteration, where the SA user interface components were to be implemented the GDTA was done, to compare what was found on the brainstorming and during development to what actually was needed, and implement what was possible.

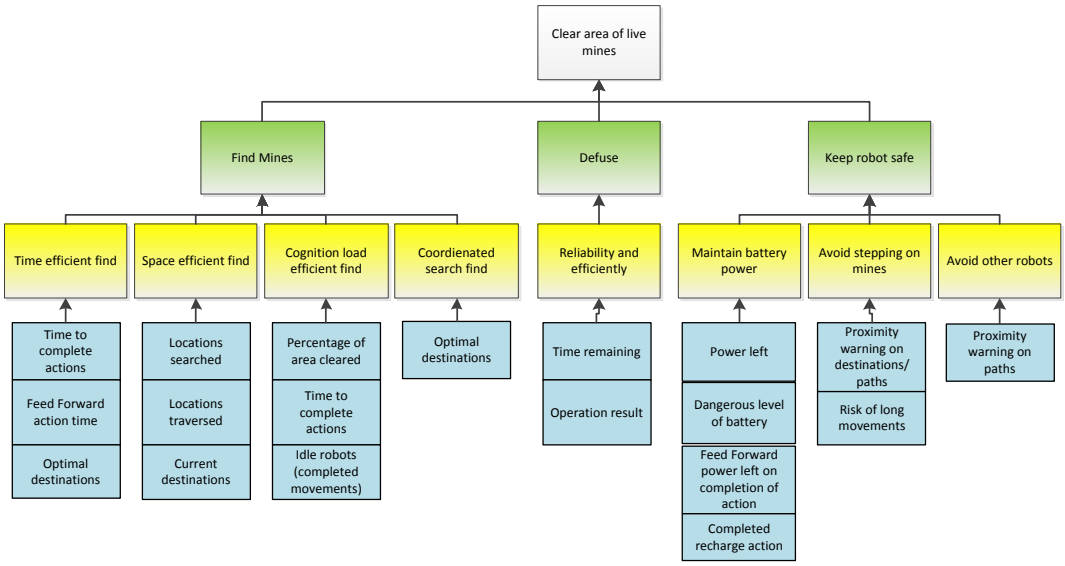

Figure III-2: GDTA of SA game. The white box represents the major goal, green boxes the sub-goals, the yellow boxes key decisions and blue boxes information needs

The GDTA identified the following needs:

- That one of the key decisions of the sub-goal finding mines was "space efficient search" and "coordinated search" between the robots. And the key decision "avoid robots" existed in the sub-goal "keep robot safe".

- In the key decision "time efficient find", that it was needed information on "optimal destinations" and "feed-forward movement time". And in the key decision "cognition load efficient find" an important information need was "percentage of minefield area cleared" and the "time to complete movements"

- In the "Maintain Power" key decision that important information was "feed-forward power left on the robot when for each action made".

III.1.3.Final Iteration

The last iteration saw the implementation of the most part of the findings from brainstorming and GDTA.

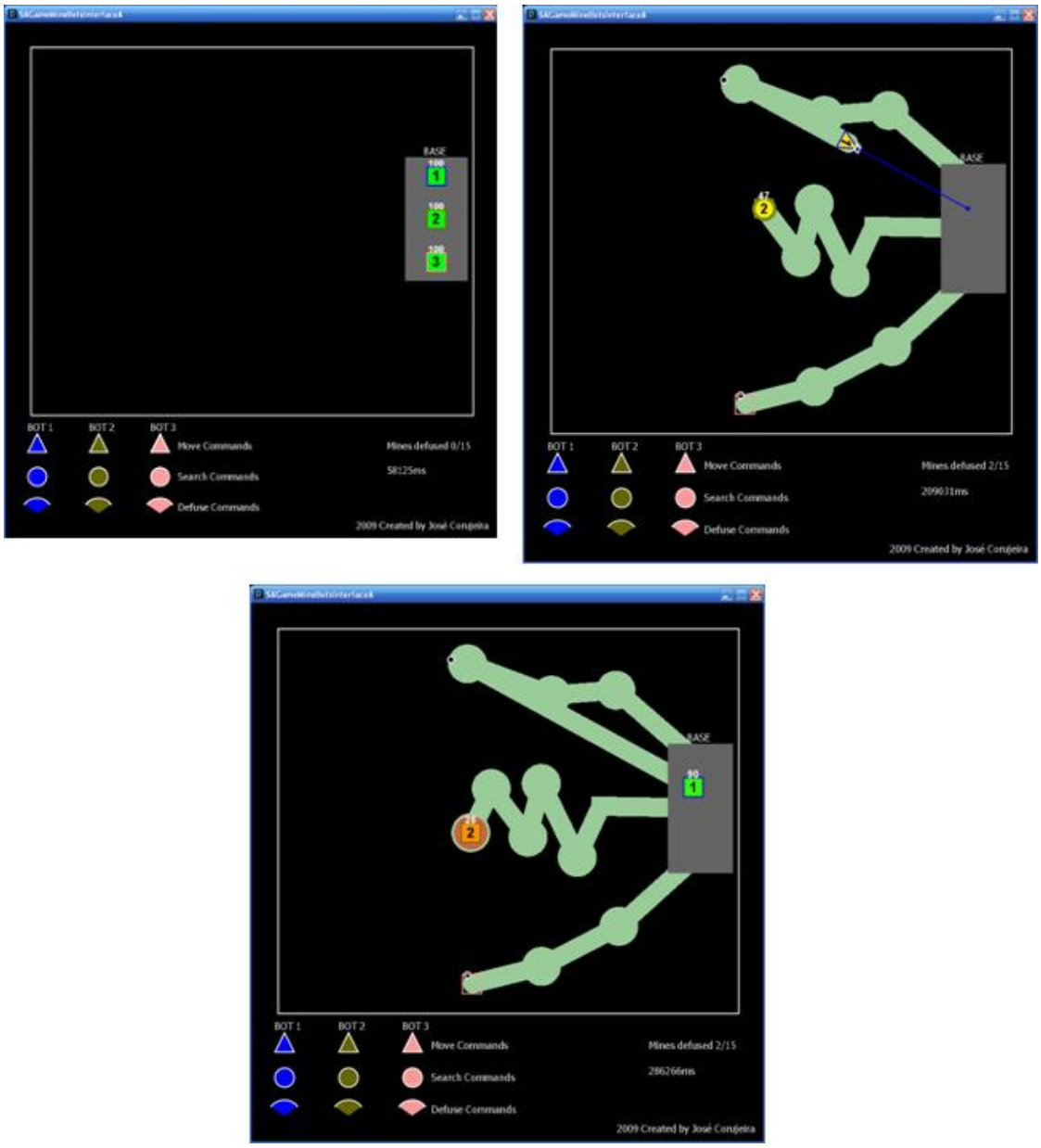

Figure III-3: Final iteration screenshots

Iteration	Major Changes
Last	Redundancy of robot identification implemented, by numbering them
	Implemented visual redundancy for state of battery by use of percentage numbers on top of robot icon
	Visual implementation of history of areas where search has already been conducted by the use of a "snail path"
	Visual implementation of movement trajectory
	Visual warning of battery below 30% implemented

Table III-4: Last iteration changes

As can be seen by the figures above, a number was added to the robot so the user can identify the robot not only by the line colour but also by its number. Likewise, the

robots now have on the top a numeric percentage to show the value of battery remaining.

Also by looking at the above figures it can be noticed that each robot when moving has a line showing the path it will do and a "snail trail", represented in light green, that shows where it has already been or searched. These visual functionalities were implemented in order to answer the GDTA information need of the sub-goals "space efficient find" and "cognition load efficient find".

Another visual cue implemented, is the red semi-transparent circle around the robot this represents a warning sign that the robot's battery level is at 30% and the user should send it to recharge, in accordance to the Brainstorm and GDTA findings on the sub-goal of "keeping the robot safe".

As can be noted the final version of the game did not implement some of the information needs that were found on the GDTA, these were mainly because the game mechanics did not allow for them and some required a complicated implementation. No attempts were made to include these information needs since the game served only as an exploration of the SA domain rather than a key project output.

The following table is an exposition of the information that was found in the GDTA and was not implemented:

Information Need	Description
Time to complete actions	It shows the operator the remaining time that took the robot to complete the action it was doing.
Feed-forward action time	Happens while selecting an action for the robot, it provides the operator with the overall time that the robot would take to perform that action.
Optimal destinations	Shows information to the operator of possible destinations where mines could be.
Feed-forward battery power left on completion of action	This is almost the same as the "feed-forward of action time", but here, while an operator is choosing an action for the robot it would show the remaining battery
Feed-forward battery power left on completion of action	It is like the other feed-forward functionalities but instead shows the remaining battery level on that actions completion.
Proximity warning to other robots and mines	This information intends to provide the operator with a means of knowing when a robot's movement may send it in direction of another robot or a mine that was found and not defused.

Table III-5: GDTA information needs not implemented

In doing this SA simulation game has a thought exercise some conclusions were made, first and the most important that GDTA should be done while in the design phase, because it enables an interface design oriented to show that SA information in a more thoughtful way and also enables in the implementation phase integrated functionality while the program is being developed. It helped also to understand how graphical oriented implementation is done. Furthermore, in the end there were some new ideas of how some functionality could be better implemented to improve the overall SA information feed to the operator. These include using animation to show when each robot finishes its action, likewise animations for warnings and the use of sound, such techniques can direct the operator's attention to these actions.

III.1.4.Summary

This section introduced the practical exploration of SA domain by the use of a small game. The concept for this game was taken from the articles "Enhancing RoboFlag users' situational awareness" (Shankar, Jin, Adams, & Bodenheimer, 2004) and "Gaming and Shared Situation Awareness" (Perla, Markowitz, Nofi, Weuve, & Loughran, 2000), and consisted on having a user control 3 robots in order to successfully defuse all the mines in a minefield. A description of the work process for this game was given in order to show the reader how the process of development of the game occurred, and how the order by which the brainstorming and GDTA were done affected the final result. In this case it was found that GDTA must be done in the design process of the project in order to in advance know which information is needed to present the operator, so when implementing the mechanics of the interface these already support the displaying of that information.

With this little game, knowledge was gained on the specific requirements needed to program for a graphical interface. First, the information shown in the screen is updated and redrawn 60 times per second, this means that some functionalities that required timing needed to be implemented around this, also that static visual objects on the screen needed to be drawn every time. Second, that a good understanding of physics equations and algebra is needed even for the simplest thing as to move an object on screen. Thirdly, if the underlying game mechanics or physics are not done prior to implementing all the visual effects intended, that some of those effects may not be possible to implement or a time consuming reimplementation of the core mechanics and physics must be done.

III.2. SA INTERFACE DESIGN

In this section an exposition of the results from the design process for the SA interface for driving the ATS is going to be made. These include the results from the two requirements gathering methods, GDTA and Activity Modelling (Constantine, 2008). As well as the prototyping solutions, these are sketches of how certain elements of the interface will look like.

Before beginning the design of the interface two methods for requirements gathering were used the GDTA and Activity Modelling (Constantine, 2008). Both of these methods need interviews and observations of operators conducting their tasks to be made, but since at the time of the elaboration of this thesis there was no possibility of doing any of those, both of the methods were done with help of articles, also by knowledge of similar tasks and suppositions.

III.2.1. Goal Directed Task Analysis

Upon doing the GDTA session, the following was obtained:

The key decision maker is the ATS operator.

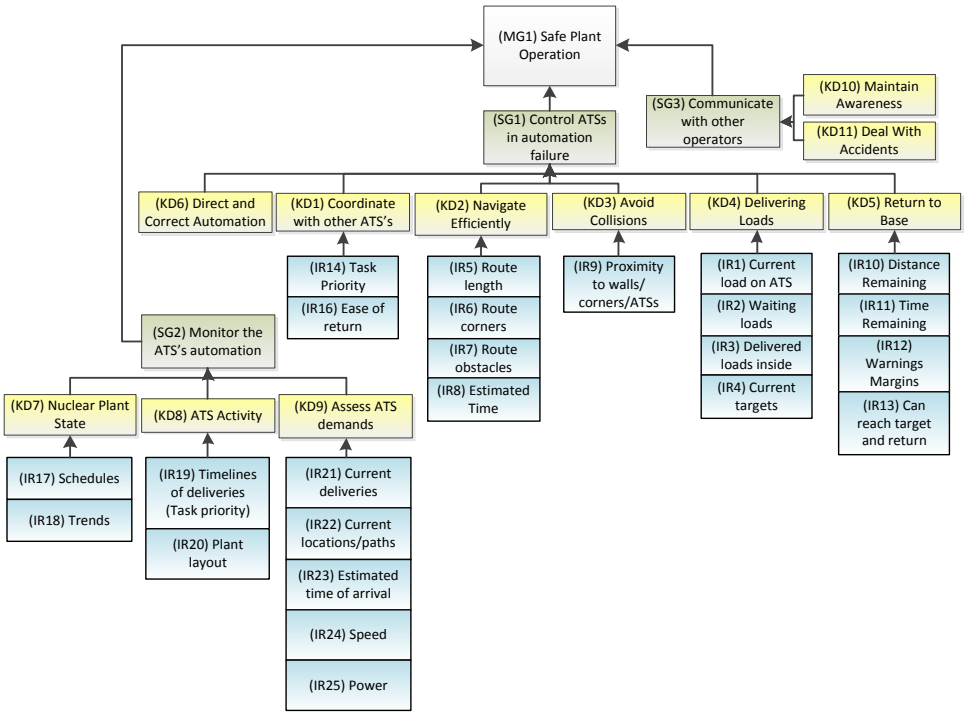

Figure III-4: ATS interface GDTA, the white box is the major goal (MG), the green boxes are the sub-goals (SG), the yellow boxes are the key decisions (KD) and the blue boxes are the information requirements (IR)

Has can be seen in figure III-4 the major goal identified is maintaining "safe plant operation" (MG1), with its three sub-goals being (Monitor the ATS's automation [SG2], Control ATSs in automation failure [SG1] and Communicate with other operators [SG3]), so an ATS operator could do his part in maintaining the DEMO fusion plant working safely. Where "Monitoring the ATSs" (SG2) is primarily done when the ATSs are on automation mode, consequently "Controlling the ATSs" (SG1) is done when the automation fails.

It is important to state that the key decision "direct and correct the automation" (KD6) has no information requirements since the focus of the project is the driving the ATS manually in the occurrence of an automation failure. Also, even though the "Communicate with other operators" (SG3) was identified along with its key decisions, a choice was made to not consider it, since this project's focus is to have an SA interface were a single operator drive a single ATS.

III.2.2. Activity Modelling

The decision to also use activity modelling to gather requirements was made because it would be interesting to see what more information could be gained from it and it enables to get a prototype sketch of the interface.

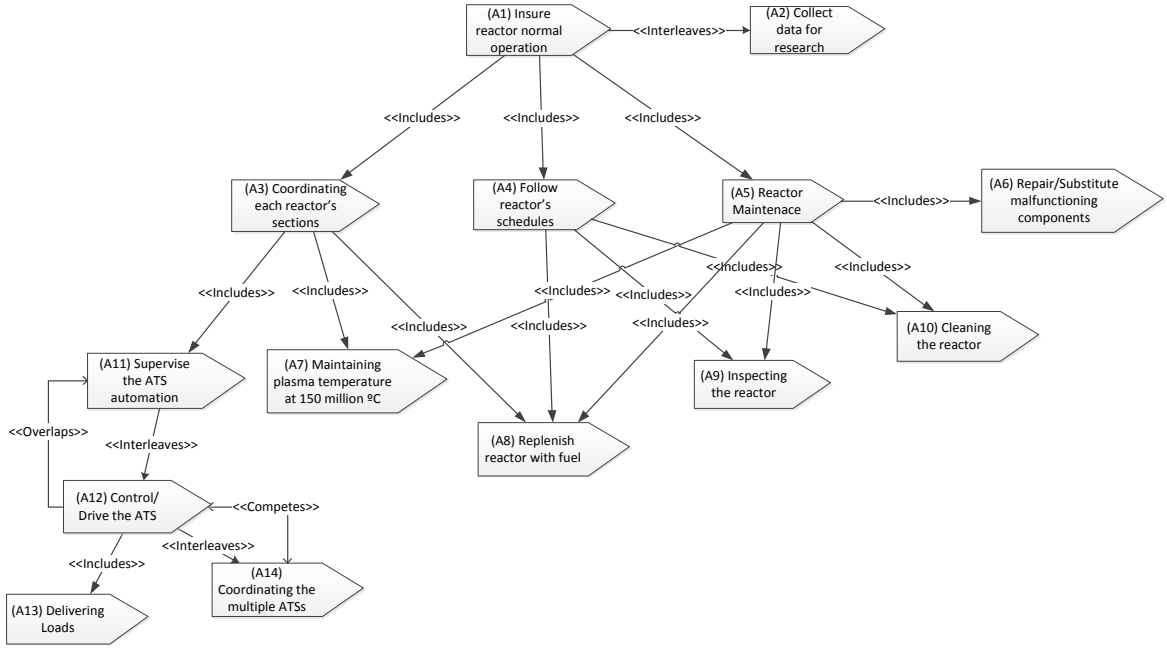

Figure III-5: Activity Map

The figure above shows an overview of the activities done in the fusion plant along with their interrelationships. Special attention to the activities A11 through A14 and their relations should be made, since these are the activities where the operator is involved and are the ones going to be developed from now.

Activity profiles

The next step in activity modelling is the description of each of the relevant activities.

<table>
<tr><th></th><th>(A11) Supervise the ATS automation task</th><th>(A12) Control/Drive the ATS</th><th>(A13) Delivering loads</th><th>(A14) Coordinating multiple ATSs</th></tr>
<tr><td>Purpose</td><td>To insure the ATS automation is operating correctly, verifying if the ATS is doing the task it needs to do and if it is not going to run out of power, crash or malfunction.</td><td>To prevent a crash of the ATS, in automation malfunction to take control over the ATS and perform the tasks manually.</td><td>In case of automation malfunction to manually move casks from one place to another with the ATS.</td><td>In case of an automation malfunction the ATSs have to be coordinated manually, prioritising ATSs by criticality of task or remaining battery power. Avoid conflicting tasks and collisions between ATSs.</td></tr>
<tr><td rowspan="3">Place and Time</td><td colspan="4">Is performed remotely in the control room, with good light, in a noisy environment.</td></tr>
<tr><td rowspan="2">It is a continuous activity, performed all day long.</td><td>The duration can range from a few minutes to days in a row.</td><td>The duration can range from an hour or days in a row.</td><td rowspan="2">The duration can range from a few minutes to days in a row. It may occur one time in five years or with less or more frequency than that.</td></tr>
<tr><td colspan="2">It may occur two times a year or with less or more frequency than that.</td></tr>
<tr><td rowspan="2">Participation</td><td colspan="4">Of the system operator (actor) playing the role of ATS driver. The players involved are the scientists, technicians, engineers that also work in the control room. The system actors are the ATS and automation system. The artefacts used are computers, other monitors in the control room, paper and pen.</td></tr>
<tr><td></td><td colspan="2">The driver may need to communicate with other operators and players in the room to get some information needed.</td><td>The operator needs to communicate with other operators and players in the room to get the information needed.</td></tr>
<tr><td rowspan="2">Performance</td><td rowspan="2">Is predominantly a relaxed activity and monotonous.</td><td colspan="3">Performed under stress and pressure, it is a critical activity.</td></tr>
<tr><td colspan="2">The actor may need to coordinate with other ATSs.</td><td>The actor needs to coordinate with other ATSs.</td></tr>
</table>

Table III-6: Activities Profiles

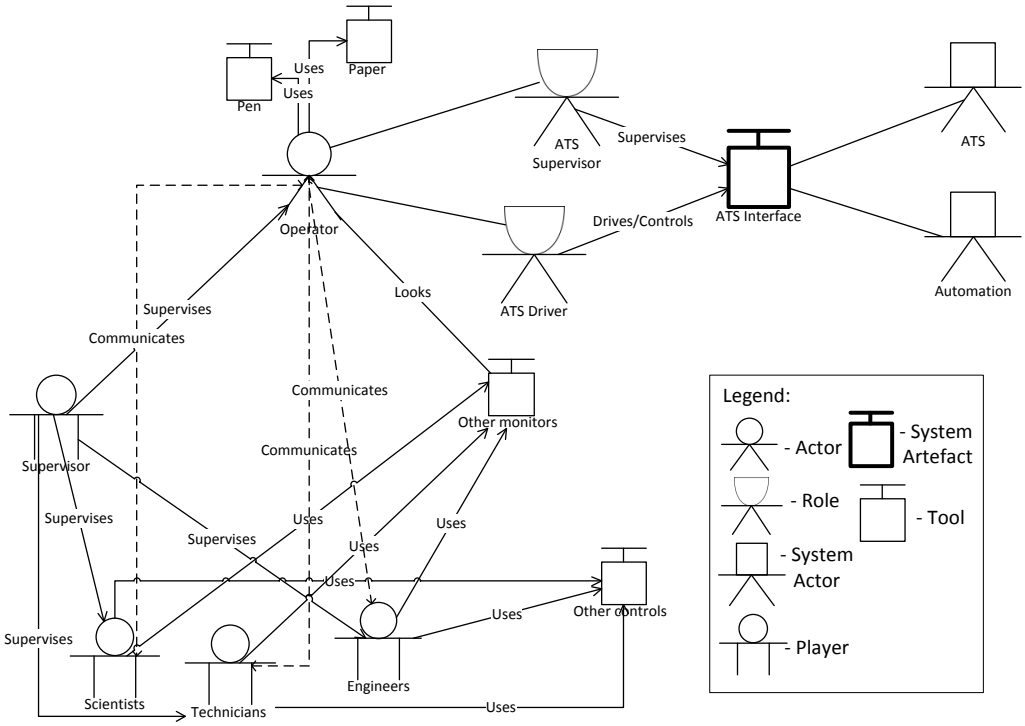

Figure III-6: Participation Map

The participation Map shows all the tools and persons involved in the four activities that were described in the activity profiles. It is important to note that the operator in this case represented as an activity actor can perform two rules when interacting with the system (SA Interface for driving the ATS) being these the ATS Supervisor which occurs when the operator has to monitor the ATS automation system and the ATS Driver which occurs when the operator has to manually control the ATS. For the role profiles refer to appendix I.

Next is showed the correspondence of which roles does the actor (Operator) perform in each activity.

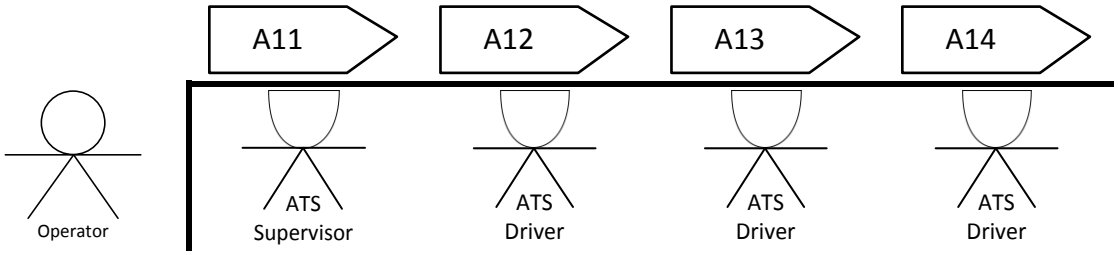

Table III-7: The correspondence of the actor with the activities, which role does the actor does in each activity

Then all the tasks and actions identified are grouped with the activities were they belong in a performance map has showed above. So it show which actions and tasks are performed within an activity.

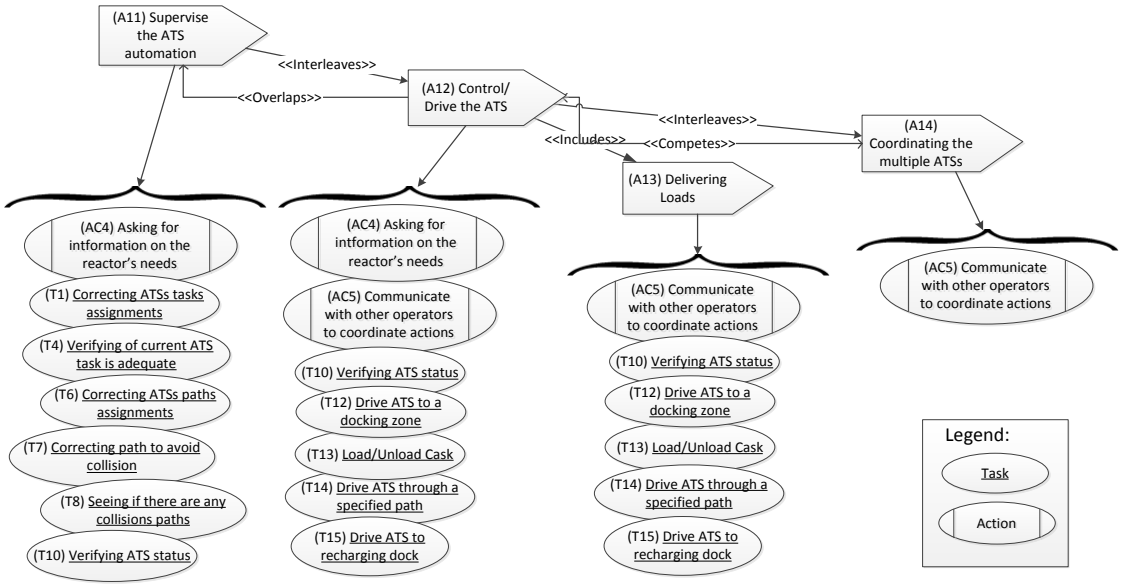

Figure III-7: Performance Map

After the tasks description, refer to appendix II, a role support matrix is done which shows the relation of the tasks to the roles. Meaning identifying which roles perform which task.

	Role 1	Role 2
Task 1	X	
Task 4	X	
Task 6	X	
Task 7	X	
Task 8	X	
Task 10	X	X
Task 12		X
Task 13		X
Task 14		X
Task 15		X

Table III-8: Role Support Matrix, shows which tasks each role does

The next step was grouping together the tasks that are associated with each role in a visual form, obtaining a task clustering.

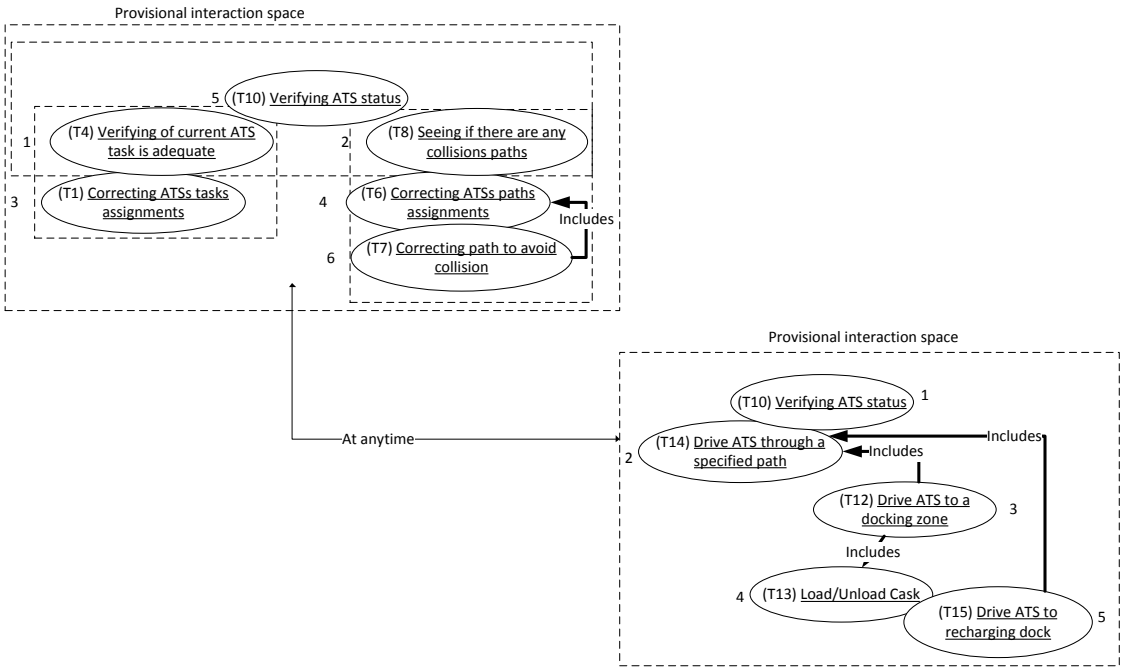

Figure III-8: Task Clustering

The task clustering represented in the figure above serves to understand which tasks must be supported together by the interface. The two bigger doted boxes represent an interaction space, meaning one or more set of interface screens dedicated to enable the operator to accomplish the tasks that are inside them. Also has can be seen in the top interaction space there are three additional doted boxes inside it, that means that inside the main interaction space there should be three more interaction spaces that need to be interconnected between them.

III.2.3.Prototyping

Taking the information gathered from the Activity Modelling the next step was to create a provisional navigation map, this represents what interface screens there are and how they are grouped together.

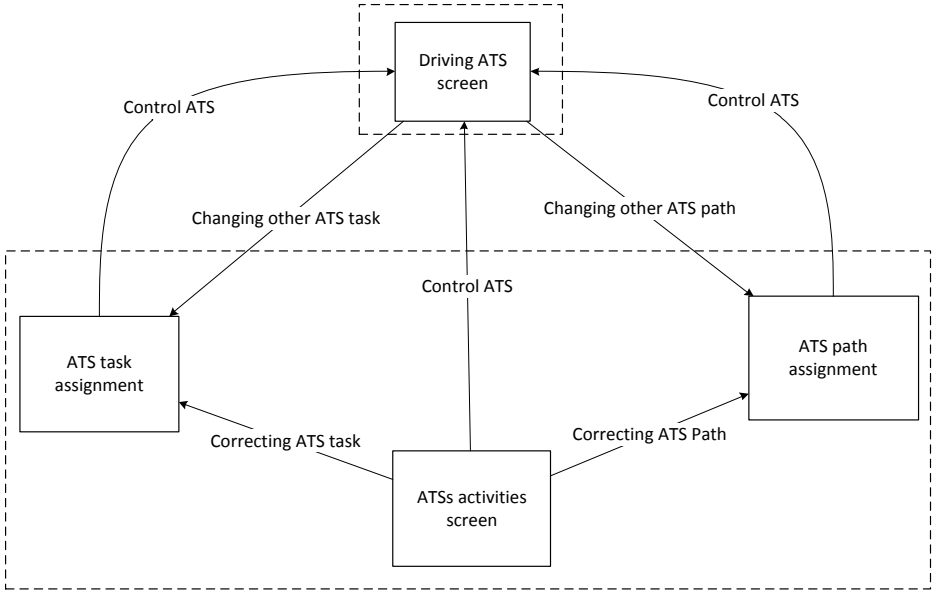

Figure III-9: Provisional Navigation Map

As can be seen from the figure above there are two doted boxes, each of this boxes include the interface screens that are relative to the driving/controlling the ATS (the doted box on top) and the interface screens for the monitoring of the ATS automation.

From here on only the screen for driving/controlling the ATS is going to be prototyped, as it is the focus of this thesis.

The screen content inventory for the interface "Driving ATS screen" is elaborated with the help of information taken from the task cases and the GDTA.

Screen content inventory

Driving ATS screen

- **(SC1)** Information about other operator's actions (or who's controlling what), to coordinate actions
- **(SC2)** ATS status (battery power, speed, direction, position)
- **(SC3)** Distance and time remaining of battery life
- **(SC4)** Warning the operator when the distance possible to travel is only enough for getting the ATS to a recharge docking.
- **(SC5)** Current load transported by the ATS
- **(SC6)** Next load to be transported

- **(SC7)** Warning of any malfunctioning ATS component

- **(SC8)** Docking zone chooser. (optionally, depends on how the automation system works)
- **(SC9)** Show where the target docking zone is
- **(SC10)** Show the nearest available recharging docking zone

- **(SC11)** Optimal path information to target, highlighting corners and obstacles
- **(SC12)** Manual path definition (optionally, depends on how the automation system works)
- **(SC13)** Show information if the battery power is enough to do the whole path
- **(SC14)** Show doors and externally controlled ATSs.
- **(SC15)** Show path length and estimated time of arrival
- **(SC16)** Proximity information of walls, other ATSs and obstacles

- **(SC17)** Information cues for safe docking, and that docking has been accomplished successfully complete
- **(SC18)** Information about load/unload operation progress

Wire-Frame Layout

The wire-frame layout is obtained by sectioning the screen space into parts that will represent the content identified above.

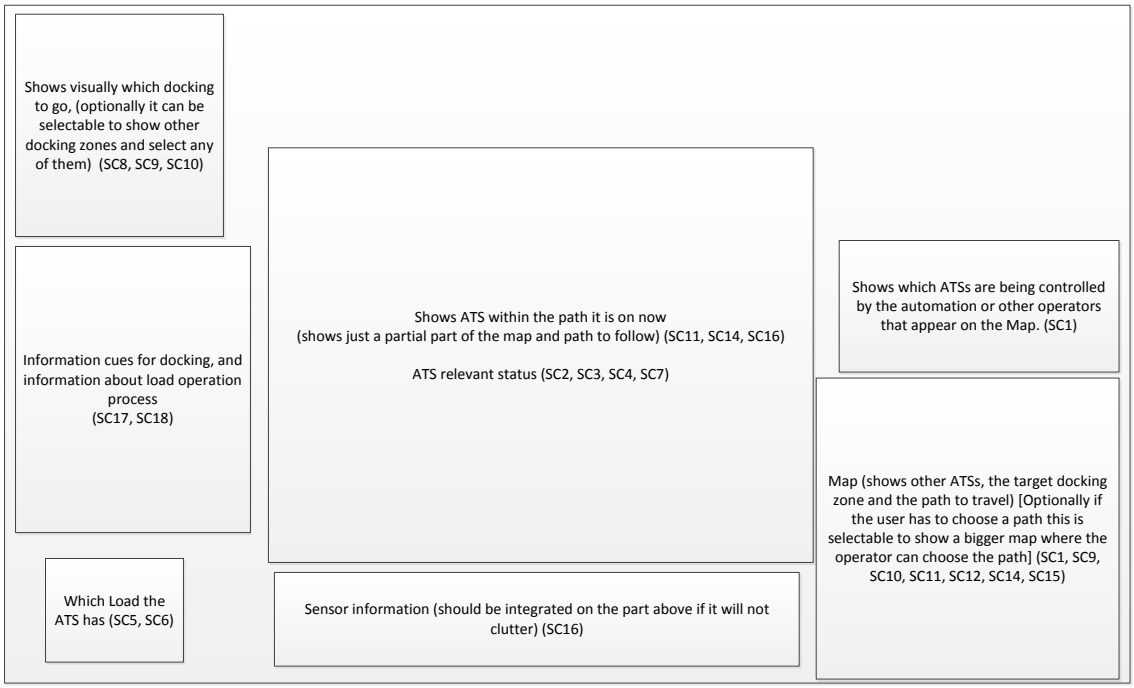

Figure III-10: Wire-frame layout of the driving ATS screen

Canonical Abstract Prototype

The canonical abstract prototype translates the distribution of information in the screen presented above into how that information is interacted.

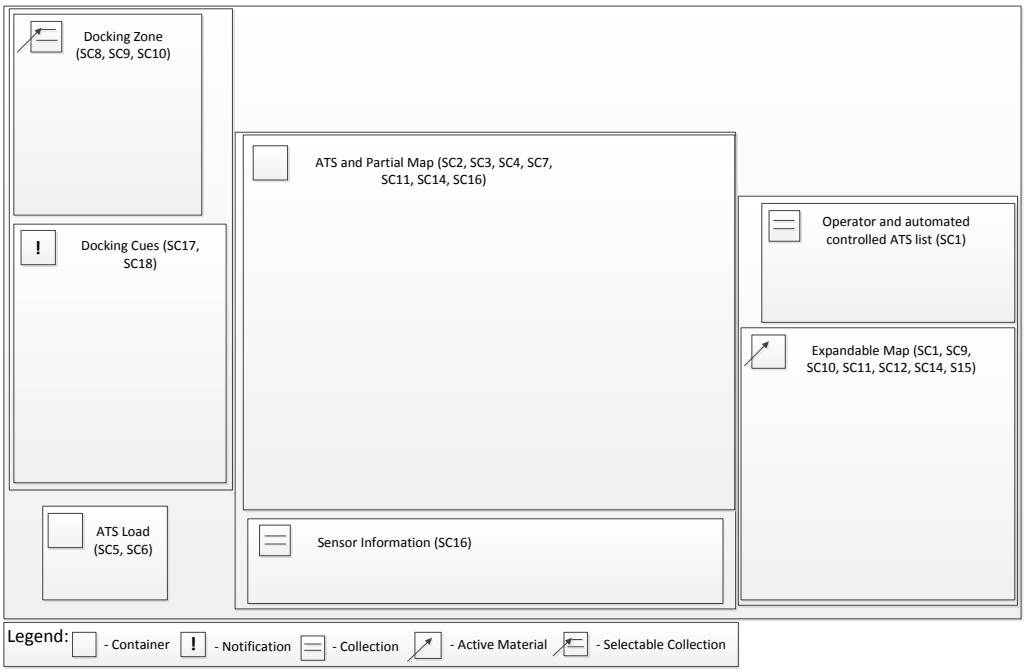

Figure III-11: Canonical abstract prototype for the driving ATS screen

High-Fidelity Prototypes

On the following stages of the prototype phase the sketches are to look like a final version of the SA interface, but indeed they are just prototype drawings to demonstrate how some elements of the interface should look like.

So looking to the following figures it can be noticed that some elements identified for the interface are not showed, such as docking cues (SC17, SC18), docking zone (SC8, SC9, SC10). These elements were taken out because it was decided to focus posterior SA evaluation on the ATS driving following a path by the operator with no concern for docking or other ATSs or obstacles, also the simulation program (not discussed in this thesis) it would run on did not support those functionalities.

A supposition made when elaborating the following three prototypes was that the sensors to detect the distance the ATS is from the walls would be integrated in the ATS.

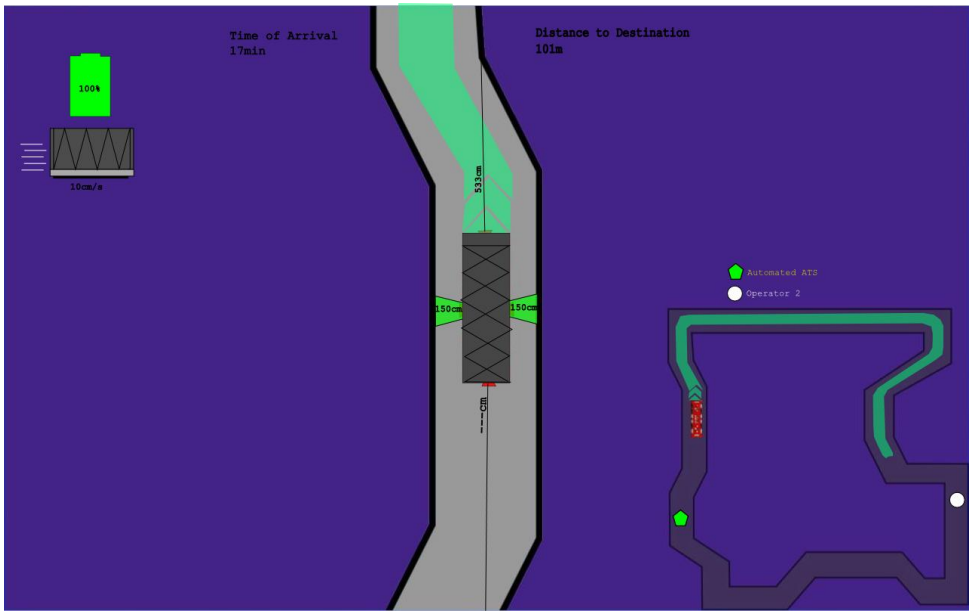

Figure III-12: prototype of ATS driving normally through the circuit

By looking at the figure above some elements on it can be immediately identified that were described previously on the wire-frame and canonical abstract. On the top left corner of the figure below the battery icon a decision was made to show a side view of the ATS in order to represent the current velocity of the ATS by use of white lines and the textual information (SC2), and also representing if the ATS as a load or not by the use of the darker grey rectangle that would show or not depending on the situation (SC5). The use of an auto-orienting partial map and a full fixed map is to enable the operator to more easily navigate through the circuit with the ATS and to be able to immediately see where the ATS is in the circuit (SC1, SC11, SC14, SC15, SC16), has explained in the (Wickens, John, Liu, & Gordon-Becker, Displays, 2003). The light

green path showed on both maps is the representation of the path the operator should follow to reach the target (SC11). The centre of the figure shows a top view of the ATS with two vertical lines shooting out of it, these lines represent the distance to the walls (SC16) and that it is a confortable distance thus no immediate precaution should be taken by the operator. While to each side it can be seen to green polygons that represent a much closer distance to the walls but are yet at a reasonable distance (SC16). In the next two figures it can be seen how this type of representation adapts to the different distances.

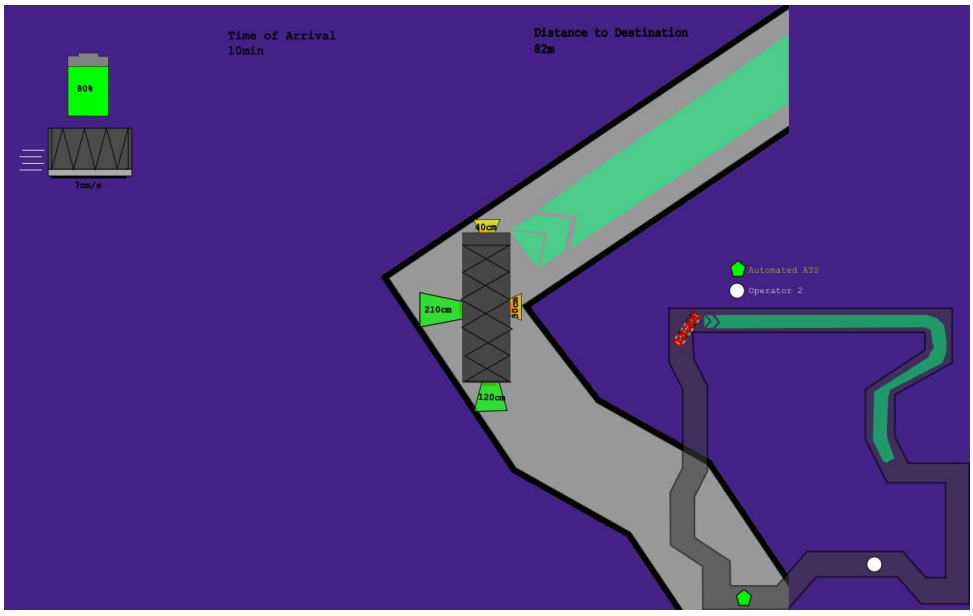

Figure III-13: prototype of ATS making a corner

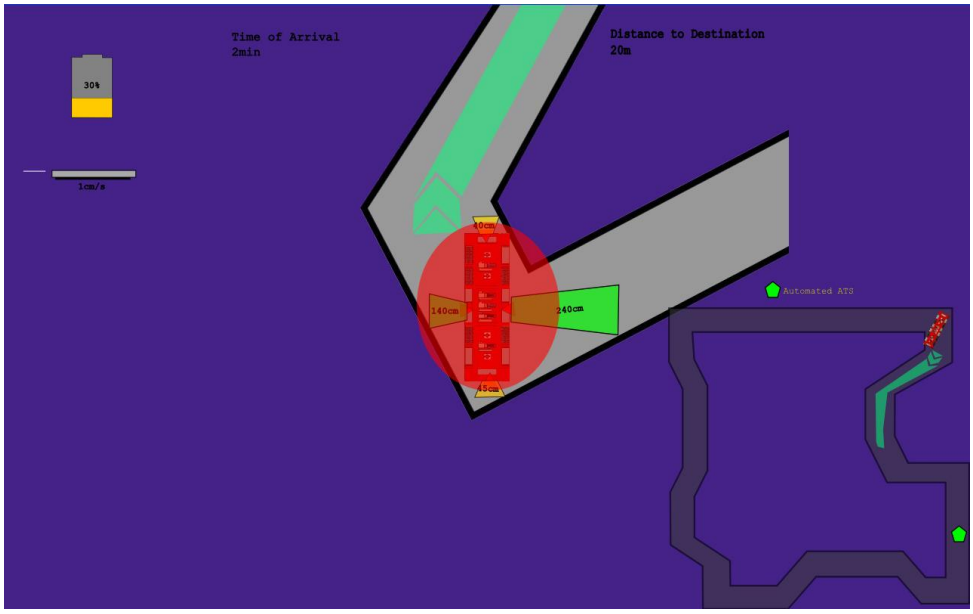

Figure III-14: prototype of ATS to close to the walls

The above figure is a prototype of a critical situation, in this case when the ATS hits a wall, a red ellipse is showed, also to note is the representation of the ATS in its side view and top view where it change to indicate that there is no load.

A second high-fidelity prototype was done to refine the previous one, and to include some changes that were known about how the sensors were going to be positioned, in this case in the walls instead of on the ATS.

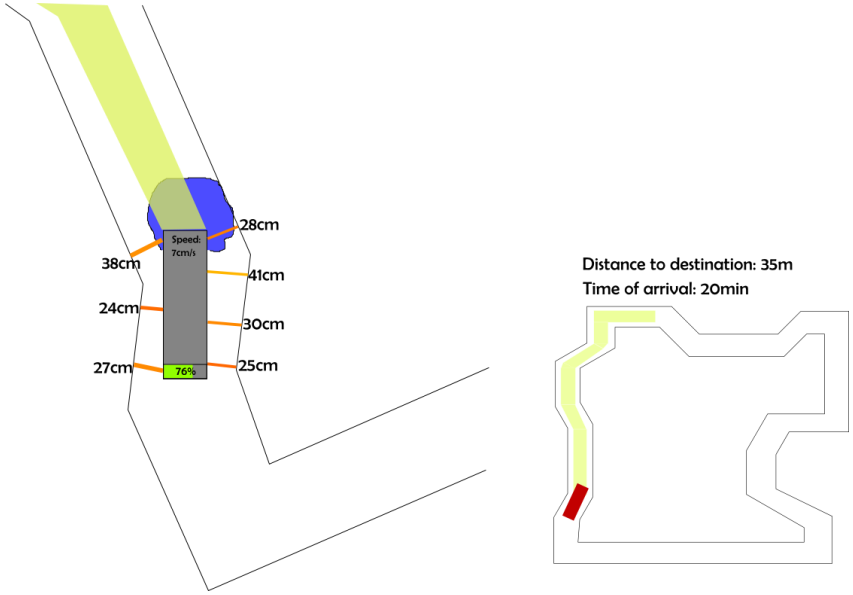

Figure III-15: Second high-fidelity prototype

So looking to the figure above it can be noted that the sensors are now attached to the walls and not to the ATS. Also, that only the sensors that detect a different distance than the width of the hallway are showed (SC16), this decision was made to not clutter the partial map with irrelevant information for the operator. Another change from the previous prototype is the integration of the speed and battery inside the ATS icon (SC2), on the top and bottom respectively. The blue blob on the figure is supposed to represent the inertial movement the ATS makes when at any given time it tries to stop (SC16), meaning if the operator tries to stop the ATS at this moment the ATS will move the distance indicated by the blue blob.

III.2.4.Summary

This section talked about the design process before the implementation of the ATS interface. By the use of the GDTA and Activity Modelling it was possible to obtain the requirements needed to know what information is critical for the operator to have available when performing the activities. Then a refinement of those requirements was done in order to come up with a consistent prototype that incorporates the information needed.

III.3. DESIGN SUMMARY

This chapter talked about a small SA game done in order to test how an implementation of SA concepts on an interface would be. By elaborating this game it was possible to identify the key points in a development process, the first that the GDTA must be done in the design step, in order to further in the process be able to incorporate all the information gathered into the interface. Also, helping to gain experience in implementing graphical interfaces.

The chapter also presented a description of the design process made for the SA interface, where the GDTA and Activity Modelling were used with the purpose of gathering the requirements for what information should be incorporated in the interface, also a series of prototypes were done, that represent how some elements of the interface should look like.

IV. IMPLEMENTATION

This chapter will talk about the implementation of the SA interface prototype. It is divided into four sections.

The first section will talk about the changes made to the TUIO client protocol 1.1 (Kalttenbrunner) with the aim of enabling the SA interface to communicate with the simulation program.

The second section will describe the implementation of the Pathfinding feature which satisfies the SC11 (see section II.2.3.) point. It covers the creation of the graph, and the search algorithms.

The third section talks about the implementation of the SVG parser that enables the SA interface to know the layout of the fusion power plant, also talks about the classes that hold the information about the objects important to the SA Interface, the map layout, the ATS and the sensors.

The final section of this chapter introduces the implementation of the visual features of the SA interface, as it talks about the sound solutions for warning the operator of any problems occurring. Also talking about the final overall look of the SA interface.

The implementation was done in Microsoft Visual C# 2008 using the TUIO protocol 1.1, the XNA Framework 3.1 and the XPath Library. To see the class diagram of the SA interface go to appendix XV.

IV.1. TUIO PROTOCOL REPURPOSING

This section presents the changes made to the TUIO client protocol 1.1 (Kalttenbrunner) in order to enable the SA interface to successfully communicate with the simulation program. It will discuss the types of objects needed to send through the server to the client and the modifications to the TuioClient class and TuioListener interface.

The necessity to link the interface to the simulation it would run on top of arose soon on the project, because the use of a simulation environment would be a good way to enable the test of the interface and takes most of the critical simulation functionalities out of the interface.

The TUIO communication protocol was chosen since it was fairly easy to modify and it permitted independence of communication from the programming languages implementation, in this case the simulation program (server) would be developed in C++ while the SA interface (client) would be developed in C#.

The first modification done to the TUIO protocol was the type of objects used as data in the communication between the server and client. Instead of using the defined object types TuioObject and TuioCursor three new types of objects were created which extend the TuioContainer class, which is the super class of the TuioObject and TuioCursor. These new object types are TCSObject, SensorObject and WheelObject. The following table shows which information these new types of objects require.

TCSObject	SensorObject	WheelObject
TCS ID	Sensor ID	Wheel ID
Rotation Angle		
Position in the World		
Speed	DataValue	Speed
Weight of TCS		
Battery Level		

Table IV-1: Information within the object types

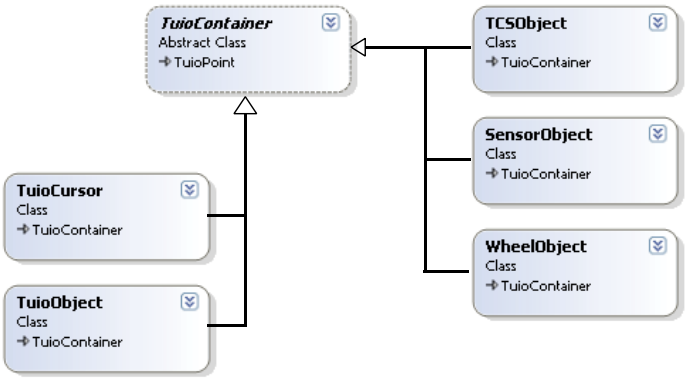

Figure IV-1: Extended classes from TuioContainer

The second modification is to the TuioListener and TuioClient, the TuioListener is an interface which enables the class that implements it to access the data that is received by the TuioClient, the TuioClient is the main class for the TUIO client protocol its function is to connect to the server and receive every communication data sent from the server.

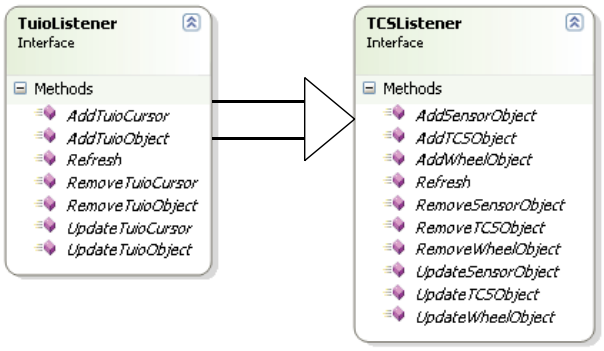

Figure IV-2: Modifications done to the TuioListener in order to accept the three new objects

The TUIOListener modification consists in changing the Add, Update and Remove methods that already exist taking out the methods that refer to the previous types used (TUIOObject and TUIOCursor) and introducing new ones which support the TCSObject, SensorObject and WheelObject.

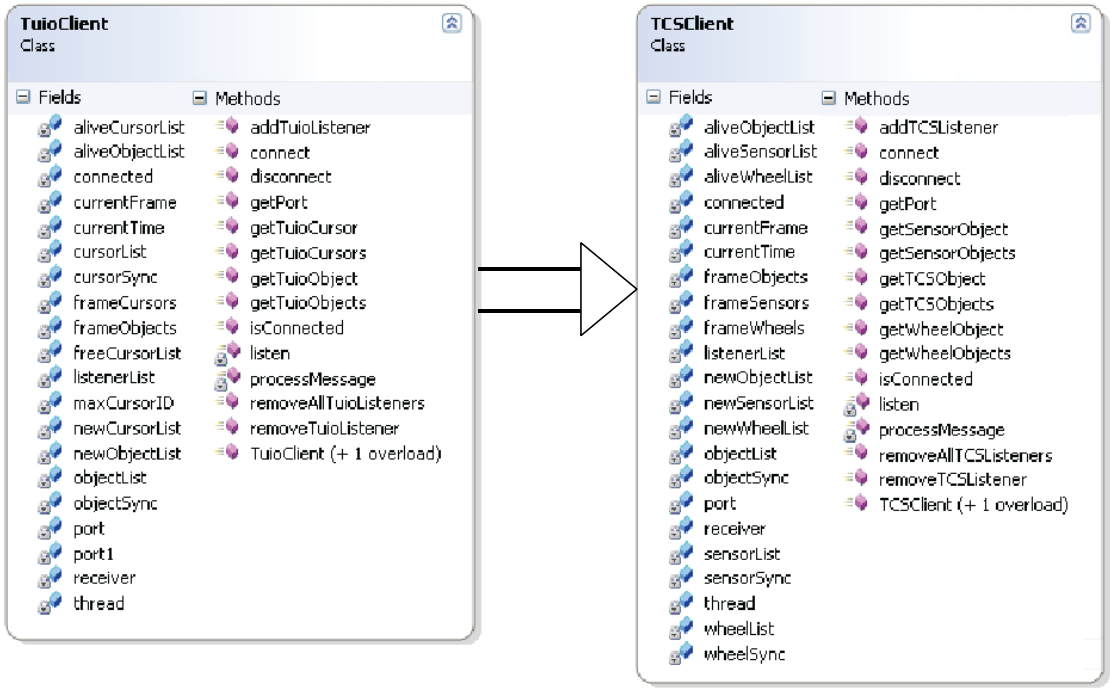

Figure IV-3: Modifications done to the TuioClient in order to accept the three new objects

In the TUIOClient the modifications needed are a little bit more complicated. First a modification to the class's attributes is needed, where for each of the three new types of object the following attributes must be created: synchronization object; a dictionary that will hold the objects of one of the three types and the respective identifying key; a list to hold the identification keys of the active objects of one of the three types; a list to hold the identification keys of all the newly created objects of one of the three types; a list to hold the objects of one of the three types that are belong in the same frame (space of time). The second major modification is done inside the *processMessage*

method, where for each of the three types of objects the address to use for the communication is as follows:

- For the TCSObject: is "/tuio/_sixyaXYAmrPP"
- For the SensorObject is "/tuio/_sixyPaPPP"
- For the WheelObject is "/tuio/_sixyaPP"

Then inside the section for the message type *SET* create variables to assign them the values that are passed through the *args* array variable. Also inside the section for the message type *FSEQ* modify the method calls in the switch cases to be in accordance with the new types of objects. Refer to appendix III for an example in pseudo-code and code in C#.

In this section a modification to the TUIO communication protocol was presented in order to enable the SA interface to communicate with the simulation program it would run on top of. It was talked about the modifications to the types of objects that needed to be sent through the server to the client, and the modifications to the TuioClient class and TuioListener interface to accept the new types of objects that are received.

IV.2. PATHFINDING

This section will talk about the implementation of the pathfinding feature by telling how to create a graph that holds convex polygons (also called navigation mesh) also telling the structure of the XML file which when read creates the navigation mesh. And the implementation of the path searching and path smoothing algorithms will similarly be talked about this use the graph to calculate the optimal path.

According to the SC11 (see section II.2.3.) point the interface needs to present the operator an optimal route for moving the ATS from a given point to another. This is where the pathfinding comes in, it serves to find the best traversable route that connects two points in a map, according to a given criteria and search algorithm. The pathfinding consists of three parts, the graph/tree/matrix this is where all the information about the traversable parts of the map is stored, then there is the search algorithm that searches within the information to calculate the best route between two

points, there are many search algorithms but the two that are used in cases like this are the Dijkstra and the A* algorithms, finally a mechanism to smooth the path.

The code used for the graph, search algorithm on the pathfinding was taken from these web pages (Vicente, Jad Engine) (Vicente, Jad Engine Source Code) and modified to function in this project.

IV.2.1.Navigation Mesh File Reader

The navigation mesh serves to tell the path finding algorithms what parts of the map can be traversed by the ATS, this consists of a series of convex polygons that share one or more sides with other convex polygons to form the area where the ATS can traverse.

The media used to store the information of the navigation mesh is an XML file, which its DTD is has follows:

```
<!ELEMENT navpath (nodes, edges)>
<!ELEMENT nodes (path|rect)+>
<!ELEMENT edges (edge+)>
<!ELEMENT path EMPTY> //This element description is taken from the same element of the SVG file type
    <!ATTLIST path id ID #REQUIRED>
    <!ATTLIST path d CDATA #REQUIRED>
<!ELEMENT rect EMPTY> //This element description is taken from the same element of the SVG file type
    <!ATTLIST rect id ID #REQUIRED>
    <!ATTLIST rect width CDATA #REQUIRED>
    <!ATTLIST rect height CDATA #REQUIRED>
    <!ATTLIST rect x CDATA #REQUIRED>
    <!ATTLIST rect y CDATA #REQUIRED>
<!ELEMENT edge EMPTY>
    <!ATTLIST edge from CDATA #REQUIRED>
    <!ATTLIST edge to CDATA #REQUIRED>
    <!ATTLIST edge  sharedseg CDATA #REQUIRED>
```

The decision to have an XML format has illustrated by the DTD above was because the navigation mesh is represented by nodes and edges, thus the use of a similar way to distinguish the two things was natural. The DTD then represents a list of nodes which in this case can be rectangles or any other type of convex polygons and edges that tell the connection between the nodes and also tell which line segment is shared between nodes.

The created class to parse this XML file and acquire the information needed to create a navigation mesh graph has the following properties: a sorted list to hold all the nodes, where these nodes consist of an identification number, a polygon object, and its position in the map; and a list to hold all the edges, the connections between the

nodes, which consists of the identification values of the two nodes that interconnect as well as the cost to traverse from one node to the other in this case the distance between the centre points of the polygons and the shared line segment between them. Refer to appendix VI for pseudo-code and C# code of the class.

IV.2.2.Connected graph of convex polygons (Navigation mesh)

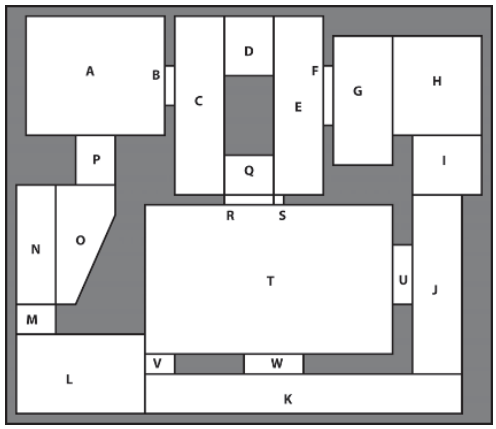

Figure IV-4: Navigation mesh representation of floor plant map

The first step to have a successful pathfinding is to choose the adequate way of abstract representation of the traversable world. (Buckland, 2005) So for this project a connected graph of convex polygons, that is also known as a navigation mesh, was chosen to represent the traversable space for the ATS. The choice of this abstract representation was due to the fact that the resulting graph created by it can have fewer nodes than any other representation and that each node (polygon) enables a definition of the area that is traversable, this means that a safe area can be defined for the ATS to traverse that avoids any undesirable proximity to walls, also it has another useful characteristic that it is possible to go from any point in the polygon to another point in the polygon due to its convexity. So for example in a hallway a convex polygon might be created to represent the traversable floor where the polygon's sides have a safe distance from the wall, enabling the pathfinding search to while finding the best route through that hallway to know that it can use the two most distant points of that polygon to create a line through where the ATS should pass. (Tonks, 2010)

The implementation of the navigation mesh was made by modifying the "Graph" package from the project Jad Engine (Vicente, Jad Engine). First the WeightedEdge class was extended by the use of the CommonSharedEdge which adds the shared segment line between two polygons as a new attribute to the edge object.

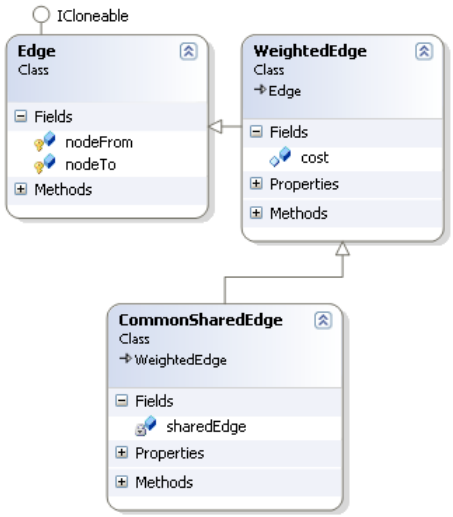

Figure IV-5: CommonSharedEdge specialization

Second a modification was done to the NavigationNode class, here a polygon object attribute was added.

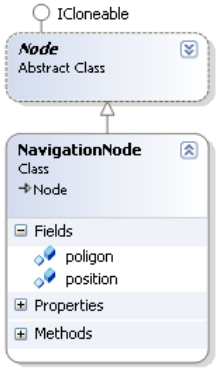

Figure IV-6: NavigationNode specialization

No modification was done to the Graph class. The following figure shows that the Graph class is composed by the CommonSharedEdge and NavigaitonNode.

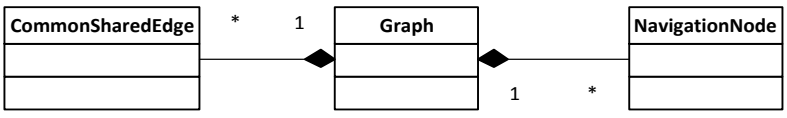

Figure IV-7: Relationship of the Graph class with the Edge and Node

IV.2.3.Path Searching

The search algorithm chosen to do the pathfinding was the A* ("A-star") algorithm, a cost-based search algorithm, this means the search will depend on the *cost* of traversing an edge, in this case the measure of *cost* used for the edges is the distance value between the nodes an edge connects. So the A* algorithm takes into account these costs to make its search for the pathfinding. The A* algorithm functions as

follows, given a graph and two nodes in that graph ("start" node and "goal" node) it generates a path such that the total path cost to traverse from the start to the goal is the minimal cost value among all possible paths from start to goal. To find a path with minimal cost value the A* algorithm, in each iteration of the search, rather than always considering the node that has not been searched yet with the lowest cost so far, it chooses the node that is most likely to lead to the shortest overall path, by using a heuristic to estimate the cost of the path to the goal node from the node being considered. The equation to do this estimate is $F = G + H$ where F is the total estimated cost to reach the goal node, G is the total real cost to reach the current node being examined form the start, H is the heuristic estimate. The heuristic estimate used for this implementation of the A* algorithm is the mean value of the *Euclidean distance* and the *Manhattan distance* $H = \frac{ED+MD}{2}$. The result given by the A* algorithm is a list of edges from the start node to the goal node. (Buckland, 2005)

A Pathplanner class is needed in order for the SA interface to be able to call the pathfinding functionality. The Pathplanner class serves to execute and manage the graph search requests required by the SA interface. Upon a request by the SA interface for a path from the ATS's current location to a target location, the path planer has to find the closest visible unobstructed graph node to the ATS's current location and the same thing to the target location, then it uses the search algorithm to find the path between the two nodes and in the end adds the two positions of the ATS and the target location to the path result, this class is also responsible for calling path smoothing algorithms on the result path. So the Pathplanner class from the "Pathfinding" package of the Jad Engine (Vicente, Jad Engine) was used with some modifications to it, these modifications were made not only to integrate the class into the SA interface implementation but also to create some new lacking functionalities that were needed.

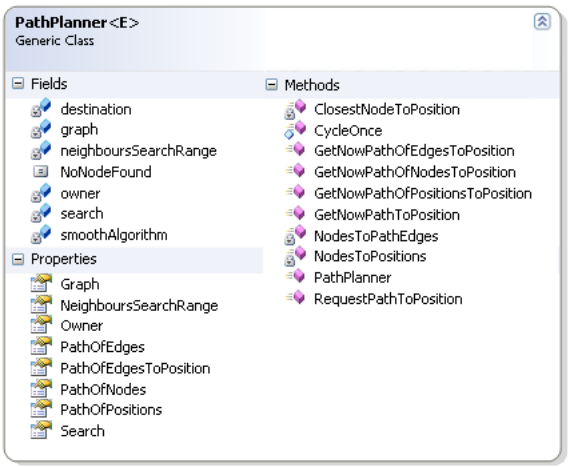

Figure IV-8: Modified Pathplanner class

The modifications consist in the changing the type of the owner attribute to ATSObject, and creating two new methods PathOfNodes, PathOfEdgesToPosition, GetNowPathOfNodesToPosition and GetNowPathToPosition.

The PathOfNodes method consists in getting the list of nodes from the result path given by the search algorithm.

The PathOfEdgesToPosition method gets the list of edges of the result path from the search algorithm, and then transforms the edges into 2D vector positions and adding these into a list. This is done by getting the centre position of each "from" node and shared line segment of the edges and adding these to a list of positions, then on the last edge getting also the centre position of the "to" node adding these positions to the list. The next step in the method is doing the smooth algorithm.

The GetNowPathOfNodesToPosition method makes a call to the search algorithm to find the path from the ATS's position to a target position returning a list of waypoint positions of the path if it exists.

The GetNowPathToPosition method runs the search algorithm to find the path between the ATS's locaiton and the target location it receives a list of edges then calls the PathOfEdgesToPosition method and returns the list of the path's waypoints and the value of the estimated time of arrival.

Refer to appendix VII to see the C# code for these methods.

IV.2.4.Path Smoothing

Figure IV-9: Example of path smoothing. Red line is the computed path, the blue line is the smoothed path

As said previously the Pathplanner class is also responsible for calling the path smoothing algorithms. The smooth functionality in a pathfinding serves to remove unnecessary edges from the computed path giving it a more realistic look, without

sharp turnings or unwanted zigzag effect. To provide the path smoothing functionality the PathMeshSmooth abstract class was created where its methods are called by the Pathplanner class to smooth a path.

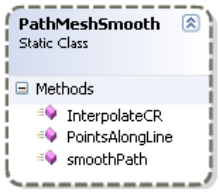

Figure IV-10: PathMeshSmooth static class

The InterpolateCR method takes the path list it calculates a series of points along the curves that exist in the path to make the curves look smoother, this is achieved by using the mathematical equation of Catmull-Rom spline. (Grootjans, 2009)

The PointsAlongLine method takes two points and finds a point along a line created by those two points, this serves to obtain one more point in order for the InterpolateCR method to smooth a curve that is near the first or last points on the path. This is done by using the following algorithm:

Where p_f is the final point in the path list, p_i is the initial point in the path list, l is the length of the line segment formed by p_i and p_f, s is the size of the spacing between two calculated points, st is the increment necessary to get a new point on the line.

1. $(x_d,\ y_d) = (p_f - p_i) = (x_f - x_i,\ y_f - y_i)$
2. $(x_a,\ y_a) = |p_i - p_f| = (|x_i - x_f|, |y_i - y_f|)$
3. $l = \sqrt{x_a^2 + y_a^2}$
4. $st = \frac{l}{s}$
5. $(x_{st},\ y_{st}) = \left(\frac{x_d}{st}, \frac{y_d}{st}\right)$
6. The result can be either a point before the initial point $(x_r, y_r) = (x_i - x_{st},\ y_i - y_{st})$ or after the last point $(x_r, y_r) = (x_f + x_{st},\ y_f + y_{st})$.

The smooth path method takes the calculate path list and checks to see if there are any unnecessary points to be removed, this means if the ATS can move between two non-consecutive points then the points between can be removed smoothing the path. (Buckland, 2005) Refer to appendix VIII for C# code.

The figure below demonstrates a simplified sequence flow of the pathfinding when it is called by the SA interface to provide a path between two positions.

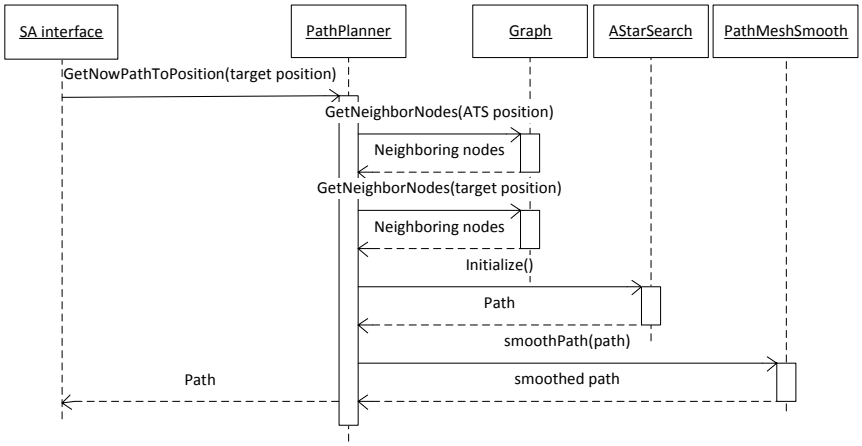

Figure IV-11: Simplefied sequence diagram for pathfinding

IV.2.5.Summary

This section addresses the implementation of the pathfinding feature, which is a requirement identified by the SC11 point (see section III.2.3). It was talked about the implementation of the navigation mesh (the graph to hold convex polygons and their relations), the creation of the XML file which allows creating a new navigation mesh. And the implementation of the path searching and path smoothing algorithms.

IV.3. MAJOR OBJECT CLASSES

This section will talk about the implementation of the classes that will hold information about the fusion power plant, like the ATS, the map layout, and the sensors, as is the implementation of the SVG parser which enables the SA Interface to read the map layout of the fusion power plant.

IV.3.1.SVG file parser

A need arisen to have the same map layouts in both the simulation and the interface a same basic type from where a 2D or 3D model map layout could be represented. It was chosen the SVG file type (World Wide Web Consortium, 2010), because it dealt with points and lines, so any map layout could be created from those points and lines.

In order for the SA interface to be able to read that file type to display the map a SVG parser was created.

The SVG parser only considers the information inside the element tag "svg/g", where in here there is the information about all the polygons that form the map's layout. So it reads the information inside the "path" element tags, for each of this tags it reads the point coordinates that form the vertexes of the polygon saving these into a polygon object. In the end of parsing the entire SVG file the parser returns a list containing all the polygons that form the map's layout. Refer to appendix IV for the pseudo-code.

IV.3.2.Polygon Object class

The creation of the class PolygonObject was necessary in order to create objects that could hold the information about the coordinate points of the vertexes that form a polygon in the map's layout or in the navigation mesh. The properties for this type of object are a list of 2D coordinate vertexes that form the polygon, the area of the polygon that is given by $A = \frac{1}{2}\sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1}y_i$ and the centroid of the polygon that is given by $C = \left\{\frac{1}{6A}\sum_{i=o}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1}y_i)\,;\, \frac{1}{6A}\sum_{i=o}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1}y_i)\right\}$. Refer to appendix V for pseudo-code.

IV.3.3.ATS Object class

The ATSObject class was created in order to create objects that hold all the information of the ATS that were going to be passed by the simulator. The main attributes the ATS object has are: the battery state which is within the values [0,100] in percentage, speed is within the values [0, 0.1] in m/s, mass is within the values [1000, 100000] in Kg, position, facing angle of the ATS in relation to the world in rads and a list of wheels. This class implements the FeedForward method which is the method that has the physics equation to calculate the stoppage distance and position of the ATS at any given time, so then it can be showed to the operator in the SA interface as a feed-forward functionality, in agreement with the SC16 requirement (see section III.2.3). So this method solves the problem in the following way (refer to appendix XI for the c# code):

- Making the supposition that the wheels of the ATS never skid then only the static friction coefficient between the rubber wheel and concrete is considered, so its value is μ_e = 1; also considering that the force of gravity is g = 9,8N, and

the constant that is the percentage of weight that the air-cushion system does not sustain K = 0,05%.

- The deceleration provoked by friction is given by the equation $a_f = -\mu_e * K * g$
- Time value is given by the equation $t = \frac{v}{-a_f}$ where v is the wheel velocity
- The stoppage position of the ATS is given by the equation $position = \left(v * \cos\theta * t + \frac{a_f \cos\theta * t^2}{2},\ v * \sin\theta * t + \frac{a_f \sin\theta * t^2}{2}\right)$ where θ is the steering angle of the wheel plus the orientation angle of the ATS in relation to the world.

IV.3.4.Wheel class

The Wheel class was created to be used by the feed-forward functionality that shows the operator the stoppage distance and position of the ATS at any given time that will be talked about further in this chapter, in concordance to the SC16. The attributes that are part of this object are: the turning angle of the wheel in rads, the position of the wheel in the world, velocity within the values of [0, 0.1] in m/s, and wheel identification.

IV.3.5.Sensor Objects Classes

The representation of the real sensors into a programming model had to take into account the possibility of having more than one type of sensors and that the number of sensors in the facility maybe in great number. So three classes were created:

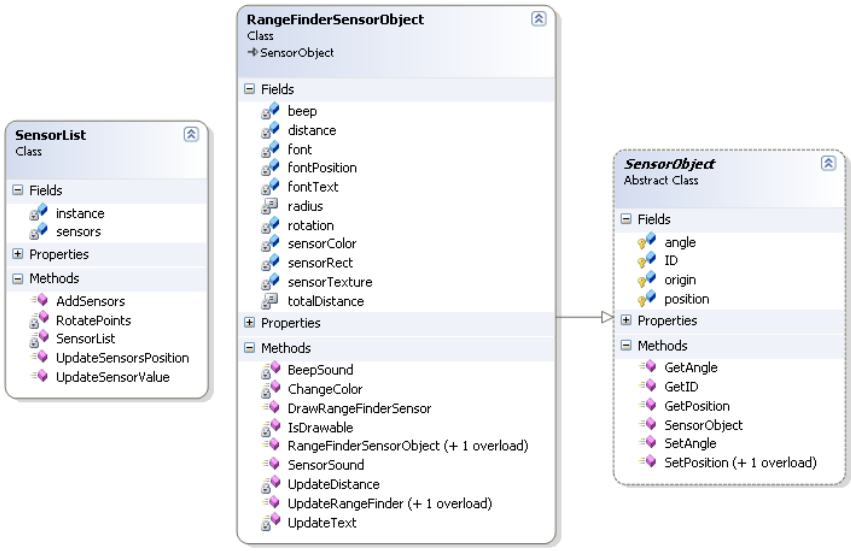

Figure IV-12: Classes related to the programming representation of the sensors

- The SensorObject abstract class which is the generic class for the sensors where it has the common attributes among all the types of sensors, the facing angle in relation to the world in rads, the identification number and the position in the world.

- The SensorList class applies the singleton design pattern, because there is only one set of sensors in the environment thus this enables a creation of a single list object that contains more all the sensors, and stops inadvertently creating more than one list thus duplicating the sensors. The attributes of this class are: the instance that is static and read-only, and has the list that holds all the sensors. This class implements two methods that are important for the visual representation of the sensors on the SA interface they are the RotatePoints and UpdateSensorsPosition. Refer to appendix IX for C# code of these methods.

    - The RotatePoints method serves to make a translation of the sensors position in the map showed in the SA interface when the orientation of the map changes. This is because the sensors need to always be showed in the same position on the map even when its orientation is rotated. This method accomplishes this translation by first creating a rotation matrix in the Z axis with the value of the rotated angle of the map, then makes a vector transform by applying to the sensor's position the rotation matrix and taking into account the centre of the rotation of the map.

    - The UpdateSensorsPosition method serves to update each sensor's visual position when according to the camera movement and rotation. This is accomplish by taking each sensor's real position on the map translating to the interface position according to the camera movement the translation equation used in this case was $tp = (sp_x - cp_x + 400,\ sp_y + cp_y + 400)$, where tp is the translated position, sp is the sensor position and cp is the camera position, then it is called the RotatePoins method to rotate the positions of each sensor according to the cameras rotation.

- The RangeFinderSensorObject class is a specialization of the SensorObject abstract class, thus extending the basic sensor to be like a range finder sensor, where the main attribute extended is the value of distance in cm to which an object is away from the sensor. Then some other attributes were added in order to represent the sensor data on the SA interface according to the SC16

requirement: *font*, *fontPosition*, *fontText*, *radius*, *rotation*, *sensorColor*, *sensorRect*, *sensorTexture* and *totalDistance*. The *radius* and *totalDistance* are constants, where the *radius* as the name suggests is the radius of the circumference done by the textual information of the sensors distance when the maps orientation changes and the *totalDistance* constant tell the default distance reading of the sensors when they do not find any object serving to disable the sensors representation on the SA interface if the distance value that the simulation is giving is the or higher than the this constant's value. The "*font*" attributes serve to display the textual information of the sensors distance value, some attention had to be taken for it work correctly, since the map orientation changes. So when initializing the *fontPosition* variable the following equation must be utilized $fontPosition = (x - radius * \cos(\pi + orientationAngle), y - radius * \sin(\pi + orientationAngle))$ where *orientationAngle* is the angle to where the sensor is facing in the world. And when updating the text position to compensate for the camera position changes the following equation is utilized $fontPosition = (x + 2 * radius * \cos(\pi + orientationAngle), y + 2 * radius * \sin(\pi + orientationAngle))$. The graphical representation of the sensors in the SA interface was accomplished by using a rectangle of a certain fixed width but where its height and colour would change accordingly to the value of distance, there is where the "*sensor*" attributes are used. To make the colour change according to the distance value the value of the distance is verified to see if it is higher than a third of the *totalDistance* if so goes from green to yellow, if not it goes from yellow to red. Refer to Appendix X for C# code of this class.

IV.3.6.Map object

The MapObject Class was created to hold the information about the map layout, in this case the list of polygons that form the layout. Due to the ATS only move through one map this class was implemented with the singleton design pattern in mind. Also this class has a method called RayCast to handle the detection of obstacle collision in a searched path. This method accomplishes this by receiving the segment of the path to check for collisions then for each polygon in the map verifies if any of the polygon's edges intersects the path segment by calling the static method IntersectionPoint. This method receives the path segment and the polygon's edge and checks for a point intersection between them using the following algorithm: where pe_f is the polygon edge final point; pe_i is the polygon edge initial point; ps_f is the path segment final point; ps_i is the path segment initial point. (Bourke, 1989)

$$Ua = \frac{(pe_{xf} - pe_{xi}) * (ps_{yi} - pe_{yi}) - (pe_{yf} - pe_{yi}) * (ps_{xi} - pe_{xi})}{(pe_{yf} - pe_{yi}) * (ps_{xf} - ps_{xi}) - (pe_{xf} - pe_{xi}) * (ps_{yf} - ps_{yi})}$$

$$Ub = \frac{(ps_{xf} - ps_{xi}) * (ps_{yi} - pe_{yi}) - (ps_{yf} - ps_{yi}) * (ps_{xi} - pe_{xi})}{(pe_{yf} - pe_{yi}) * (ps_{xf} - ps_{xi}) - (pe_{xf} - pe_{xi}) * (ps_{yf} - ps_{yi})}$$

Then verifying if 0<=*Ua*<=*1* and 0<=*Ub*<=*1* for there to exist an intersection. Refer to appendix XI for the code on this method.

IV.3.7.Summary

In this section it was seen the implementation of the SVG parser which enables the SA interface to read information about the fusion power plant's layout map, also it was showed the implementation of the classes that retain information about the map layout (MapObject and PolygonObject) as well as the classes that hole information about the ATS (ATSObject and Wheel) and finally the implementation of the classes that hole the information about the sensors (SensorObject, SensorList and RangeFinderSensorObejct).

IV.4. INTERFACE IMPLEMENTATION

This section is going to present the implemented solutions for the visual representations of the interface, based on a widget framework. Where the scales chosen for the visual representations in the SA interfaces were 1meter = 23.06pixels for the main map and ATS icon since the resolution used is 1750 per 1750 pixels. And 1meter = 5,76pixels for the mini-map and mini ATS icon as the resolution used is 400 per 400 pixels. Also it will talk about the auditory warning solutions implemented and the overall look of the SA Interface.

IV.4.1.Widgets

A widget framework consists in each interface's visual element handles its own representation. In this implementation each sensor type handles its own representation as does the ATS icon, the map and mini-map.

IV.4.1.1.ATS representation widget on the partial map

To represent the ATS on the partial map in the SA interface the ATSIcon class was created, this class is also responsible for representing the feed-forward functionality

that shows the operator the stoppage distance and position of the ATS at any given time, in concordance to the SC16 (see section III.2.3). The representation of feed-forward functionality was implemented in two ways, the first one is displaying a single green arrow showing the direction of the movement and the stoppage distance, if the wheels have the same steering angle. The other depiction is showing two green arrows with each arrow showing the direction of the movement and distance stoppage of the back and front of the ATS, if the two wheels have different steering angles.

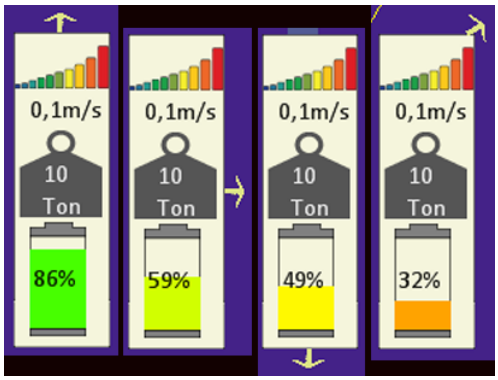

Figure IV-13: The feed-forward visual representation when the steering angle of the wheels coincides only shows one arrow

The implementation of the single arrow was done by creating a rectangle which contains the green arrow image, using has height the stoppage distance with scale applied, which the equation is $height = \left(\frac{\frac{stopdistance}{300}}{0{,}001}\right) * 3{,}545$. And the equation for the rectangles position is given by $Position = \left(x + \frac{Iwidth}{2} * \cos\left(\theta - \frac{\pi}{2}\right), y + \frac{Iheight}{2} * \sin\left(\theta - \frac{\pi}{2}\right)\right)$ where *iwidth* and *iheight* is the width and height of the ATS icon image respectively.

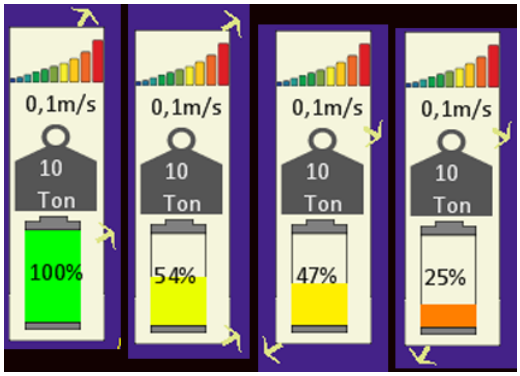

Figure IV-14: the feed-forward visual representation when the steering angle of the wheels is different it shows two arrows

The implementation with the two arrows was done by creating to distinct rectangles where each contains the green arrow image, and the position is given by the equation

$position1 = \left(x + \frac{Iwidth}{2} * \cos\left(\theta_1 - \frac{\pi}{2}\right), y - 0{,}3420412 * iheight + iheight * 0{,}1579588 * \sin\left(\theta_1 - \frac{\pi}{2}\right)\right)$ and $position2 = \left(x + \frac{Iwidth}{2} * \cos\left(\theta_2 - \frac{\pi}{2}\right), y + 0{,}3420412 * iheight + iheight * 0{,}1579588 * \sin\left(\theta_2 - \frac{\pi}{2}\right)\right)$. Refer to appendix XIII for c# code.

IV.4.1.2.Map representation widget

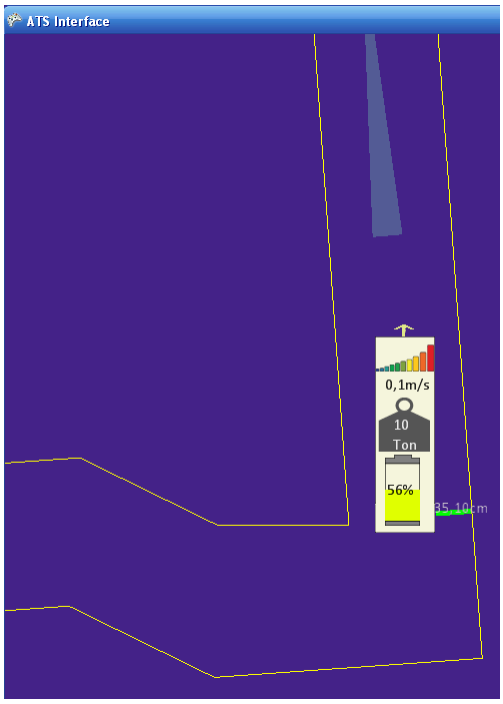

Figure IV-15: Partial map visualization

The partial map representation was accomplished by creating the class MapRenderer, which its responsibility is drawing to the computer screen a panned part of the map where the ATS is currently moving through, and also drawing the path result obtained from the pathfinding.

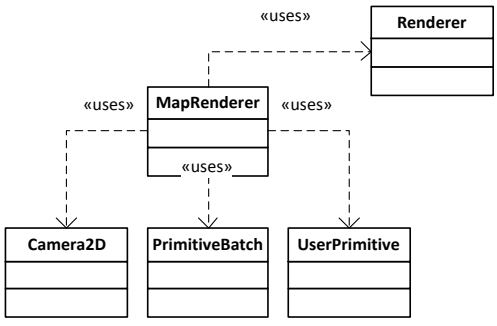

Figure IV-16: Class diagram of MapRenderer class usage relations

To pan the map an instantiation of the Camera2D class (taken from the code found in (Microsoft)) is used by the MapRenderer class that enables to pan just part of the map and rotate the entire map according to the position of the ATS and its orientation. To

draw the partial map an instantiation of the UserPrimitive class (created based on the code from (Microsoft, 2010)) is used, which draws the continuous lines of each polygon of the map's layout to a virtual screen. In order to draw the path result from the pathfinding, first the method GenerateTrackVertices (created based on the code from (Grootjans, 2009), refer to appendix XIV for the c# code) is called to create more vertices that run along each side of the calculated path, all these vertices then are used to create a triangle list which by calling the instance of the PrimitiveBatch class (from (Mircosoft, 2007)) draws triangles alternating then if they are in the even indexes of the list or the odd indexes to give a sense of animation and flow showing the direction the operator should drive the ATS, and if there are only two vertices in the path a black line is drawn instead.

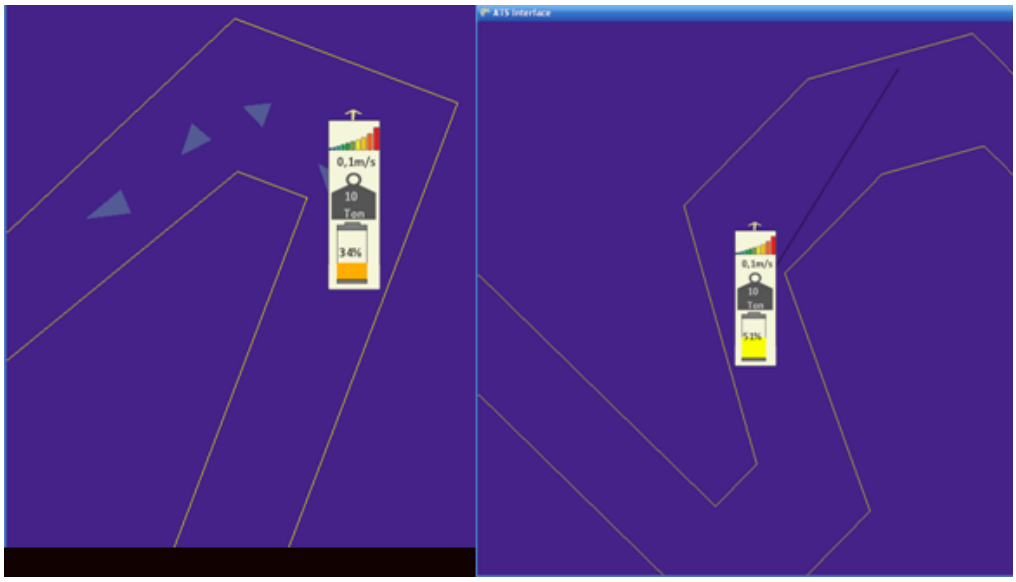

Figure IV-17: Two screens of the partial map displaying the two ways the pathfinding is showed

The virtual screen is created by the Renderer class which implements the singleton design pattern, this virtual screen is needed since it enables to process the contents talked above and puts them in a smaller size than that of the real screen, only then it is taken from the processed content is taken from the real screen and put on the real screen.

IV.4.1.3.Mini-map representation widget

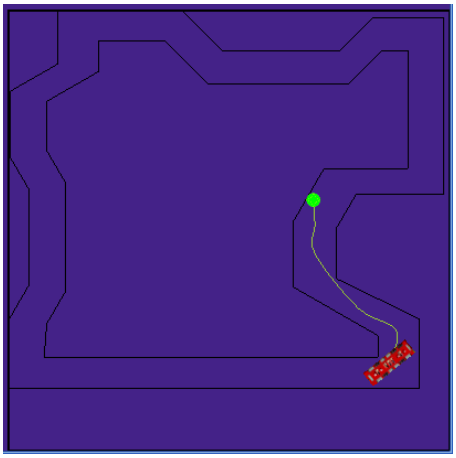

Figure IV-18: Mini-map

The mini-map representation is the responsibility of the Radar class, where it has to draw a small version of the entire map layout and as well as a smaller icon of the ATS and the calculated path from the pathfinding. Here the ATS icon is the one moving around the map. In order to represent the position of the ATS faithfully and the vertices' position of the calculated path a scale equation had to be done:

$position = \left(x * \frac{8}{35} + mpx, -y * \frac{8}{35} + mpy + 400\right)$, where *mpx* and *mpy* are the position coordinates of the mini-map on the screen window.

IV.4.2.Sound

Sound functionality was also implemented to alert the operator when the ATS is to near to the wall and when any component of the ATS is malfunctioning. The sound for the proximity to walls is handled by each individual sensor where the sound used is like the one used by the parking assist feature in cars. It was decided that the beeping sound tempo was going to be as follows:

- No beeping sound if sensor value is higher than 50cm
- If sensor value is within [50; 37.5[then the beeping sound happens with an interval of 1 second
- If sensor value is within [37.5; 25[then the beeping sound happens with an interval of 0.5 seconds
- If sensor value is within [25; 12.5[then the beeping sound happens with an interval of 0.25 seconds

- If sensor value is within [12.5; 0[then the beep sound is continuous

The use of sound to alert the operator for any ATS malfunctioning was accomplished by using the sounds of air-cushions and electric motors working, these alter according to the velocity of the ATS and the weight it is carrying, if any of these components hold stop working the corresponding sounds would stop playing. Thus, these sounds serve has another aid for the operator to know that something is wrong besides the visual representations, therefore addressing the SC7 point (see section III.2.3).

IV.4.3.SA Interface

In the end of the implementation phase a decision was made to have three somewhat distinct SA interfaces, it was intended for conducting the SA evaluation. The first and second interfaces are intended to have the most information integrated within the operators main focus centre with the difference of one being pictorial and textual and the other just textual, the third interface any integrated information and instead shows it in the right side of the screen.

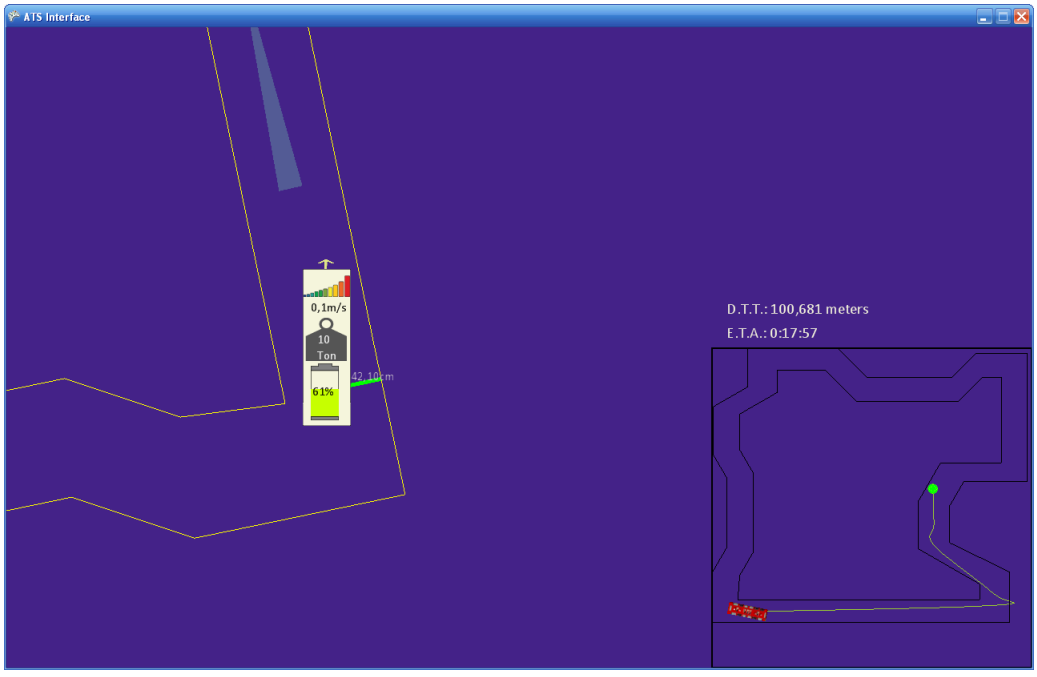

Figure IV-19: First interface with integrated pictorial information

As can be seen on the previous figure showing the interface with integrated pictorial information there are two distinct areas, the left area which shows a partial map that rotates and is panned according to the position and orientation of the ATS with a sensor representation integrated that shows through the means of colour length of the rectangle and text the current distance the ATS is from the wall at that point, also shows the calculated path to follow. There is also showed the icon that represents the ATS where inside it is the battery pictorial representation, the weight pictorial

representation that shows the current weight of cargo the ATS is transporting, this icon disappears when the ATS is not transporting any cargo, there is also showed the pictorial representation of the current speed by means of a bar that grows and changes colour from blue to red and the feed forward arrow. To the right area the mini-map can be seen which shows the entire map layout with a representative ATS icon and the path to follow in green, on top of the mini-map it is showed the distance to target (D.T.T) in meters and the estimated time of arrival (E.T.A).

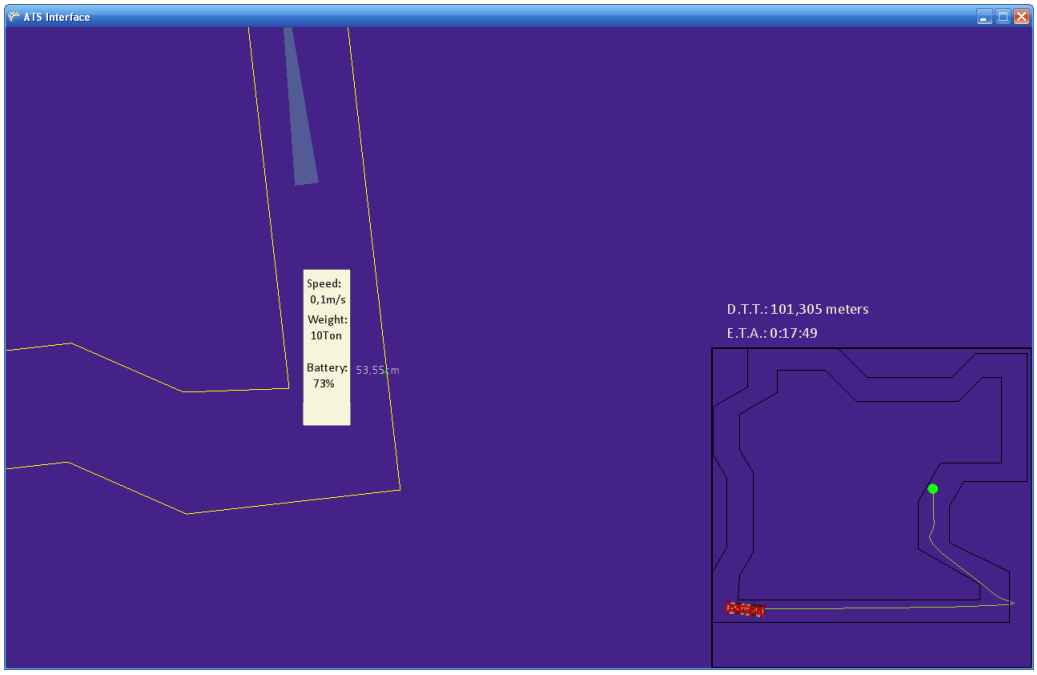

Figure IV-20: Second interface with integrated textual information

The interface with integrated textual information showed in the picture above differs from the previous interface showed in that it only shows the integrated information in textual form even the sensors' display it only shows the textual information of the distant the ATS is from the wall.

Figure IV-21: Third interface with peripheral information

The third interface shows all the information in a peripheral way to the right, where the sensor information is now showed in the mini-map and the velocity, weight and battery is showed above the mini-map.

IV.4.4.Summary

As could be seen this chapter talked about the implementation of the SA interface visual components and auditory components, also the final look of the SA interface is addressed.

IV.5. IMPLEMENTATION SUMMARY

This chapter talked about the implementation of the SA interface prototype. Where it showed the modification need to the TUIO client protocol 1.1 to enable the SA interface to communicate with the simulation program. It also showed the pathfinding implementation and the implementation of major classes for the SA Interface to work. Showing in the end how the SA Interface looks like.

V. EVALUATION

This chapter will discuss the evaluation process for the three SA interfaces. In particular it will cover the method for performing the SA evaluation. Evaluation has a big importance on a project since it allows finding out what interfaces accomplish their objectives and what conclusion can be taking from the problems identified. Due to time constraints and compatibility problems with the simulation program evaluation was not done.

The objective of this evaluation process is to gather information to give some answers about these two topics:

- How the integrated information on the operators focal point can influence SA, by presenting integrated information in pictorial or textual form versus the presenting information in the peripheral field of view of the operator.
- How, if at all the feed-forward information helps in the operator's decision making.

V.1. METHOD

In order to conduct the evaluation some information about the SA interface status needs to be saved during the evaluation process, so it was chosen the CSV file format would be used to save the status information of the SA interface, which would be done automatically by the interface every 5 seconds. The format of the information saved in the file is as follows:

Time: ; <value>
Hovercraft:
Mass=; <value>; Speed=; <value>; Position=; {X: <value> Y: <value>}; Battery=; <value>; Rotation=; <value>
Wheels:
Num=; <value>; Velocity=; <value>; Angle=; <value>0
Sensors:
ID:; <value>; Type:; <type name>; Distance:; <value>; Position:; {X: <value> Y: <value>}; Angle=; <value>

The method to use in the evaluation is the SA measurement SAGAT (Endsley & Garland, 2000). The test participants most have some domain knowledge, or at least some domain knowledge most be taught to them, the knowledge to teach is: how the fusion power plant operates and in what conditions; what is the job of the ATS and a 10 to 15 minutes period might be given to the test participants in order to get used to the peculiar driving controls of the ATS.

The simulated task the test participants are going to do is picking up some cargo and delivering it to a destination. Each test participant will do three tests, each running for 20 minutes, and each test is done with a different SA interface that has a different position for the cargo and the destination. Each of these tests will be done with a time interval of one week to allow the test participants to forget about the in the main aspects of the interface they tested but still remember the domain of the test. The order of the SA interfaces to be tested by the participants will be different for each group of participants. There will be three groups with each group being constituted by at least eight persons and the order to test the interfaces is done so that each group is testing a different interface along the three trials. So the order for the interfaces to be tested by each group is as follows:

- Group 1: SA interface pictorial integrated information, then SA interface textual integrated information and finally SA interface peripheral information
- Group 2: SA interface textual integrated information, then SA interface peripheral information and finally SA interface pictorial integrated information
- Group 3: SA interface peripheral information, then SA interface pictorial integrated information and finally SA interface textual integrated information

The choice of having each group test out each of the interfaces in a different sequence is to see if the results obtained from the test sessions will be consistent independently of which interfaces the test participants already experimented.

Each 20 minutes test session will have four random system freezes along the session with the SA interface going blank so the test participant can respond to one of 15 randomly created questions sheet that will have the following questions in any order:

- SA level 1 questions (Participant's perception of the elements in the environment):
    - What is the estimate time of arrival?

- What is the distance to target?
- What is the current ATS speed?
- What is the remaining battery power?
- Is the ATS transporting cargo?
- What is the current cargo weight?
- How dangerously close is the ATS to any obstacle?
- Where will the ATS stop?
- Which sensors are giving warnings?
- How many sensors are giving warnings?
- Where is the target location?
- What is the path to follow?

- SA level 2 questions (The participant's comprehension of the perceived elements):
  - Is the path to follow the optimal one?
  - What is the most critical incident issue facing the ATS?
  - Are you taking the shorter route?
  - What is the hardest part of the route you must take?
  - Is the ATS going too fast?
  - Is the remaining power sufficient to get to destination?

- SA level 3 questions (The participant's ability to project the future status of the elements and the prediction of how the participant's actions will affect them):
  - At the current speed and direction is the ATS going to hit any obstacle?
  - If the ATS were to brake where would it stop?

V.2. EVALUATION SUMMARY

This chapter discussed the importance objectives of the evaluation to the interfaces and the method in which these interfaces should be evaluated. Unfortunately it was not possible to perform the evaluation and having results to analyse.

VI. CONCLUSION

This thesis addressed the development process of an interface that enables an operator to control the transportation robot (ATS), which will operate inside the ITER fusion power plant that is being constructed in Cadarache, near Aix-en-Provence in Southern France, if the automation system fails.

To develop the interface, thorough study on teleoperation concepts, such as solutions developed to overcome the unique challenges the operators face when controlling robots remotely, and situation awareness concepts, such as how operators process information and solve problems based on that information, was performed. The knowledge from these two topics helped acquire the information requirements and do the design for the interface prototype. Afterwards a description of the important aspects of the three interfaces implementation was made, where a change to the TUIO communication protocol was described as was the pathfinding feature and major classes. Finally the three interfaces were showed and commented on.

Thus, showing that almost all the objectives proposed on the beginning were accomplished, except for the evaluation which only the description of the method to test the three interfaces was done.

VI.1. GOALS ACHIEVED

Along the elaboration of this thesis the following goals were accomplished:

- A detailed description of the known knowledge in teleoperation and situation awareness concepts applied to a robot remotely controlled with no line-of-sight was presented.
- A plausible list of information requirements that any interface implemented for controlling the ATS needs to satisfy in order to provide the operator with adequate situation awareness.

- A possible design sketch for an interface that allows a human operator to control the ATS safely and successfully was done, based on teleoperation and situation awareness as well as activity modelling.

- Three prototype interfaces were implemented, based on the elements from the previously designed sketch interface. But the three interfaces differ from each other in the position and manner the relevant information for the operator is showed.

    - The first interface shows the information within the operator's focal point of view in a pictorial style.

    - The second interface shows the same information in the same place but only in textual form.

    - The third interface shows the information in a different position in this case the operator's peripheral field of view and in bigger scale.

- A test method was elaborated based on SAGAT to evaluate the three interfaces.

VI.2. CONTRIBUTIONS

The contributions made by this thesis to the scientific community are:

- An information requirements list that shows which information is relevant to show the operators in order to enable them to have a good situation awareness of the activity.

- An example design sketch of an interface following the situation awareness GDTA method and activity modelling.

- The elaboration of an evaluation method based on SAGAT to evaluate the three interfaces that were created.

- Contribution to the ITER project with three viable interface solutions for the manual control of the ATS.

VI.3. LIMITATIONS

Due to time constraints the pathfinding feature calculates paths that seem inadequate, this happens because the graph does not implement the property to enable the polygon vertices to be added as edges enabling a more meticulous search and it has a primitive obstacle detection function that those not take into consideration the area of the polygons that form the traversable area.

Also the feed-forward functionality was intended to show a phantom trail that would represent the full movement the ATS would make to stop at any given time taking into account the unique physical properties of the ATS, but was not possible since a real physics engine was not implemented to deal with the simulation.

The constraints in time and compatibility issues between the interface and the simulation program did not enable the conductions of interface evaluation.

VI.4. FUTURE WORK

For future reference this interface could be improved by optimizing the pathfinding, obstacle detection and feed-forward functionalities to include information about difficult corners or obstacles, highlighting them in order to notify the operator to have caution in those parts. Also, it would be interesting to implement the distance and time remaining of battery power status of the ATS, using it to show when a path is not plausible to travel because the ATS would run out of power and to warn the operator when the distance the ATS could travel is enough to reach the nearest recharging docking zone. As well as implementing warning functionalities to tell the operator when the ATS has any malfunctioning components.

When a realistic simulation of the fusion power plant is created or the real fusion power plant is finished some augmentation should be made to the interface so it can show information about other ATSs working in the same level and which are being controlled by operators, to enable coordination. Moreover information about the current cargo is being transported by the ATS and the next cargo to be transported, as well as information cues for safe docking.

Finally it would also be interesting in the future to prototype and implement the automation monitor part of the interface discussed in the section III.2.

BIBLIOGRAPHY

Albers, M. J. (1999). Information design considerations for improving situation awareness in complex problem-solving. In *Proceedings of the 17th annual international conference on Computer documentation* (pp. 155-158). New Orleans, Louisiana, United States: ACM.

Bejczy, A. K. (1999). Teleoperation, Telerobotics. Jet Propulsion Laboratory, California Institute of Tecnology.

Bourke, P. (1989, April). *Intersection point of two lines (2 dimensions)*. Retrieved July 6, 2010, from http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/

(2005). Pratical Path Planning. In M. Buckland, *Programming Game AI by Example* (pp. 333-336). United States of America: Wordware Publishing, Inc.

Cognitive Processes. (n.d.). Retrieved October 14, 2010, from http://mentalmodels.mitre.org/cog_eng/ce_methods_I.htm

Constantine, L. L. (2008). Human Activity Modelling: Toward a Pragmatic Integration of Activity Theory with Usage-Centered Design. In A. Seffah, J. Vanderdonckt, & M. C. Desmarais, *Human-Centered Software Engineering: Software Engineering Models, Patterns and Architectures for HCI* (Vol. II, pp. 27-50). New York, New York, USA: Springer-Verlag.

Drascic, D. (1991). Skill Acquisition and Task Performance in Teleoperation using Monoscopic and Stereoscopic Video Remote Viewing. *Proceedings of the Human Factors Society 35th Annual Meeting*, (pp. 1367-1371). San Francisco.

Drury, J. L., Riek, L., & Rackliffe, N. (2006). A decomposition of UAV-related situation awareness. *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction* (pp. 88-94). Salt Lake City: ACM.

Durfee, E. H., Kenny, P. G., & Kluge, K. C. (1998, March). Integrated Premission Planning and Execution for Unmanned Ground Vehicles. *Auton. Robots, 5*(1), 97-110.

Endsley, M., & Hoffman, R. R. (2002, November). The Sacagawea Principle. *IEEE Intelligent Systems, 17*(6), 80-85.

Endsley, R. M., & Garland, J. D. (2000). *Situation Awareness Analysis and Measurement.* New Jersey: Lawrence Erlbaum associates.

Ferland, F., Pomerleau, F., Le Dinh, C. T., & Michaud, F. (2009). Egocentric and exocentric teleoperation interface using real-ti 3D video projection. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction* (pp. 37-44). La Jolla, Californa, USA: ACM.

(2009). Getting the Most Out of Vertices. In R. Grootjans, *XNA 3.0 Game Programming Recipes: A Problem-Solution Approach* (pp. 507-520). United States of America: Apress.

ISR Lisbon. (n.d.). *Mobile robot missions in nuclear fusion scenarios of ITER*. Retrieved January 10, 2010, from http://www.ipfn.ist.utl.pt/rh/

ITER Organization. (n.d.). Retrieved January 10, 2010, from ITER Project: http://www.iter.org/

Johns Hopkins University Department of Psychological and Brain Sciences. (2007). *Neurogenetics & Behavior Center*. Retrieved October 14, 2010, from http://nbc.jhu.edu/beh/cognitive.aspx

Jones, D. G., Endsley, M. R., & Bolstad, M. (2004). The Designer's Situation Awareness Toolkit: Support for User-Centered Design. *Proceedings of the Human Factors and Ergonomics Society 48th Annual Meeting*, (pp. 653-657). New Orleans.

Kalttenbrunner, M. (n.d.). *TUIO*. Retrieved March 26, 2010, from http://www.tuio.org

Menchaca-Brandan , M. A., Liu, A. M., Oman, C. M., & Natapoff, A. (2007). Influence of perspective-taking and mental rotation abilities in space teleoperation. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction* (pp. 271-278). Arlington, Virginia, USA: ACM.

Microsoft. (2010). *How To: Draw Points, Lines, and Other 3D Primitives*. Retrieved May 17, 2010, from http://msdn.microsoft.com/en-us/library/bb196414(v=XNAGameStudio.31).aspx

Microsoft. (n.d.). *Tiled Sprites Code Sample*. Retrieved May 17, 2010, from http://create.msdn.com/en-US/education/catalog/sample/tiled_sprites

Mircosoft. (2007, April 27). *Primitives Code Sample*. Retrieved May 17, 2010, from http://create.msdn.com/en-US/education/catalog/sample/primitives

Moyle, S. A. (2005). Parallel processes and situation awareness display design. In *Proceedings of the 17th Australia conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future* (pp. 1-4). Canberra, Australia: Computer-Human Interaction Special Interest Group (CHISIG) of Australia.

Nielsen, C. W., Goodrich, M. A., & Ricks, R. W. (2007, October). Ecological Interfaces for Improving Mobile Robot Teleoperation. *IEEE Transactions on Robotics, 23*(5), 927-941.

Perla, P. P., Markowitz, M., Nofi, A. A., Weuve, C., & Loughran, J. (2000). Gaming and Shared Situation Awareness.

Salmon, P. M., Walker, G. H., Ladva, D., Stanton, N. A., Jenkins, D. P., & Rafferty, L. (2007). Measuring situation awareness in command and control: comparison of methods study. *Proceedings of the 14th European conference on Cognitive ergonomics: invent! explore!* (pp. 27-34). London: ACM.

Shankar, S., Jin, Y., Adams, J. A., & Bodenheimer, B. (2004). Enhancing RoboFlag Users' Situational Awareness. *PROCEEDINGS of the HUMAN FACTORS AND ERGONOMICS SOCIETY 48th ANNUAL MEETING*, (pp. 856-860).

Tetsuo, S., & Yukio, H. (2000, September). Ecological interface enabling human-embodied cognition in mobile robot teleoperation. *Intelligence, 11*(3), 20-32.

Tonks, M. (2010). *Navigation Mesh Reference*. (Epic Games, Inc.) Retrieved July 23, 2010, from http://udn.epicgames.com/Three/NavigationMeshReference.html

Trouvain, B. (2006). Teleoperation of Unmanned Vehicles: The Human Factor. *RTO Human Factors and Medicine Panel (HFM) Workshop held in United States Military Academy* (pp. 11-1 to 11-8). West Point: NATO Research and Technology Organisation.

Vicente. (n.d.). *Jad Engine*. Retrieved May 16, 2010, from http://www.jadengine.com/

Vicente. (n.d.). *Jad Engine Source Code*. Retrieved May 16, 2010, from http://www.koders.com/info.aspx?c=ProjectInfo&pid=39NQQSWCVHDT7NUNT5QWMFABWG

(2003). Cognition. In C. D. Wickens, L. D. John, Y. Liu, & S. Gordon-Becker, *Introduction to Human Factors Engineering* (2nd ed., pp. 120-155). Prentice Hall.

(2003). Displays. In C. D. Wickens, L. D. John, Y. Liu, & S. Gordon-Becker, *Introduction to Human Factors Engineering* (2nd ed., pp. 210-211). Prentice Hall.

World Wide Web Consortium. (2010, June 22). *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. Retrieved June 26, 2010, from http://www.w3.org/TR/SVG11/index.html

Zalud, L. (2007). Augmented Reality User Interface for Reconnaissance Robotic Missions. *The 16th IEEE International Symposium on Robot and Human interactive Communication, 2007. RO-MAN 2007.* (pp. 974-979). Jeju: IEEE.

APPENDIXES

APPENDIX I – ROLE PROFILES

ATS Supervisor

Activity	(A11) Supervise the ATS automation task
Background	Experience: Training with the system, qualified System Knowledge: Familiar with the ATS interface system Domain Knowledge: Fully familiar with the fusion reactor and ATS Performance Skills: Begins has intermediate then transitions to expert
Characteristics of Performance	Frequency: Every day Intensity: Minimal to fair amount of workload and in a constant Duration: Full work shift Complexity: Little complexity Predictability: Follows a standard proceeding to do the work Volume: Handles a fair amount of information Direction: Visualizes the information with little to no input of "commands"
Design Implications:	User comprehension User satisfaction User accuracy Reliability of interaction

ATS Driver

Activities	(A12) Control/Drive the ATS, (A13) Delivering loads, (A14) Coordinating multiple ATSs
Background	Experience: Had previous training with the system at least once. System Knowledge: Good or poor, due to not using the ATS interface system over extensive periods of time. Domain Knowledge: Fully familiar with the fusion reator and ATS Performance Skills: May begin has intermediate or expert, but will tend to decline to novice.
Characteristics of Performance	Frequency: One time ranging from one or five years. Regularity: Irregular, it will only happen when the automation system goes down or malfunctions. Intensity: High amount of workload. Duration: Can range from minutes to hours. Complexity: High complexity Predictability: It is unpredictable, has to do whatever is needed at the time. Volume: A fair to great amount of information. Direction: Visualizing information and giving instructions to the system.
Design Implications:	User comprehension Ease of learning User accuracy Reliability of interaction Ease of remembering Efficiency of interaction

APPENDIX II – TASKS DESCRIPTION

(T1) Correcting ATSs tasks assignments	
User Intentions	System Responsibilities
At any time {A4 Asking for information on the reactor's needs, T10 Verifying ATS status, T12 Drive ATS to a Docking Zone, T13 Load/Unload cask, T14 Drive ATS through a specified path, T15 Drive ATS to recharging dock**}**	
Repeat as needed<	
1. Verifying assigned ATSs tasks	
	2. Showing current/future tasks
	3. Offering choices
in any order{	
4. Reassigning task;	
optionally[modifying task**]}**	
	5.Confirm changes**>**

(T6) Correcting ATSs paths assignments	
User Intentions	System Responsibilities
At any time {T10 Verifying ATS status, T12 Drive ATS to a Docking Zone, T13 Load/Unload cask, T14 Drive ATS through a specified path, T15 Drive ATS to recharging dock**}**	
Repeat as needed<	
1. Verifying assigned ATSs paths	
	2. Showing ATSs paths
	3. Offering choices
Repeat as needed<	
4. Changing path layout	
	5. Confirming changes**>>**

(T7) Correcting path to avoid collision	
User Intentions	System Responsibilities
At any time {T10 Verifying ATS status, T12 Drive ATS to a Docking Zone, T13 Load/Unload cask, T14 Drive ATS through a specified path, T15 Drive ATS to recharging dock**}**	
1. Verifying ATSs movements	
	2. Showing all ATSs movements and paths
	3. Offering choices
Repeat as needed<	
4. do (T6) Correcting ATSs paths assignments**>**	

(T12) Drive ATS to a docking zone	
User Intentions	System Responsibilities
At any time {A5 Communicate with other operators to coordinate actions, T10 Verifying ATS status**}**	
Optionally[	
1. Choosing the docking zone**]**	
	2. Informing of which docking zone is to go
	3. Offer guiding information
4. do (T14) Drive ATS through a specified path	
5. do (T13) Load/Unload cask	

(T13) Load/Unload cask	
User Intentions	System Responsibilities
At any time {A5 Communicate with other operators to coordinate actions, T10 Verifying ATS status**}**	
1. Docking the ATS	
	2. Provide information cues for a safe docking
	3. Give information that dock is complete
4. Loading/Unloading cask	
	5. Informing about the operation progress

(T14) Drive ATS through a specified path	
User Intentions	System Responsibilities
At any time {A5 Communicate with other operators to coordinate actions, T10 Verifying ATS status**}**	
	Repeat as needed<
	Optionally[
	1. Offer a path
2. Choose path**]**	
3. Driving ATS to the end of path**>**	
	4. Offer guiding information

(T15) Drive ATS to recharging dock	
User Intentions	System Responsibilities
At any time {A5 Communicate with other operators to coordinate actions, T10 Verifying ATS status**}**	
Optionally[	
1. Choosing the recharging dock**]**	
	2. Inform of which recharging dock is available
	3. Offer guiding information
4. do (T14) Drive ATS through a specified path	
5. Docking the ATS	
	6. Provide information cues for safe docking
	7. Give the information that dock is complete
	8. Begin ATS recharge

APPENDIX III – TUIO PROTOCOL CODE MODIFICATIONS

TuioClient Modification, the following pseudo-code is done for each of the three types of objects:

Create synchronization object
Create a dictionary to hold the created objects and identifying key
Create a list to hold the identification key of the objects that are still active in the system
Create a list to hold the identification key of the newly created objects
Create a list to hold the frame objects

In the "disconnect" method **Clear** all the lists created for each object type

In the "processMessage" method use the following address for each of the object types "/tcs/obj" for the TCSObject; "/tcs/sensor" for the SensorObject and "/tcs/wheel" for the WheelObject.

Inside the conditional to verify **if** the "command" is "set" the following must be done:
Create variables to hold the transmitted value of each of the object's properties, which are passed through an array variable with the name "args".

Then a synchronization **lock** must be done using the synchronization object, this lock serves to not allow more than one thread access to the creation or modification of the object.

Inside the lock a condition is made to see **if** the object already exists in the dictionary if it does not **create** it and add it to the frame list. **Otherwise** see if there are any modifications on the object and **update** the object in the list with the values received.

Inside the conditional to verify **if** the "command" is "fseq" the following must be done:
In the conditional **switch** the corresponding modifications of the object must be done.

The previous modifications in C# are as follows for the TCSObject:

```
private object objectSync = new object();
private Dictionary<long, TCSObject> objectList = new
Dictionary<long, TCSObject>(32);
private List<long> aliveObjectList = new List<long>(32);
private List<long> newObjectList = new List<long>(32);
private List<TCSObject> frameObjects = new List<TCSObject>(32);
```

Inside the proccessMessage method:

```
if (address == "/tuio/_sixyaXYAmrPP") //Code for the TCSObject
{
     if (command == "set")
     {
           long s_id = (int)args[1];
           int t_id = (int)args[2];
           float xpos = (float)args[3];
           float ypos = (float)args[4];
           float angle = (float)args[5];
           float xspeed = (float)args[6];
           float yspeed = (float)args[7];
           float rspeed = (float)args[8];
           float maccel = (float)args[9];
           float raccel = (float)args[10];
           float bvalue = (float)args[11];
           float load = (float)args[12];

           lock(objectSync)
           {
                 if (!objectList.ContainsKey(s_id))
                 {
                       TCSObject addObject = new
TCSObject(s_id,t_id,xpos,ypos,angle);
                       frameObjects.Add(addObject);
                 }
                 else
                 {
                       TCSObject tobj = objectList[s_id];
                       if (tobj == null)
                       {
                             return;
                       }
                       if ((tobj.getX() != xpos) ||
                 (tobj.getY() != ypos) || (tobj.getAngle() !=
                 angle) || (tobj.getXSpeed() != xspeed) ||
                 (tobj.getYSpeed() != yspeed) ||
                 (tobj.getRotationSpeed() != rspeed) ||
                 (tobj.getMotionAccel() != maccel) ||
                 (tobj.getRotationAccel() != raccel) ||
                 (tobj.GetBatteryState() != bvalue) ||
                 (tobj.GetLoad() != load))
                      {
```

```
                                TCSObject updateObject = new
TCSObject(s_id,t_id,xpos,ypos,angle);

     updateObject.update(xpos,ypos,angle,xspeed,yspeed,rspeed,m
accel,raccel,bvalue,load);
                                frameObjects.Add(updateObject);
                                      }
                                }
                           }
                 }

                else if (command=="fseq")
                {

                                if (!lateFrame)
                    {
                                      IEnumerator<TCSObject>
frameEnum = frameObjects.GetEnumerator();
                                      while(frameEnum.MoveNext())
                          {
                                            TCSObject tobj =
frameEnum.Current;

                                            switch
(tobj.getTuioState())
                                {
                                                  case
TCSObject.TUIO_REMOVED:
                                                        TCSObject
removeObject = tobj;

     removeObject.remove(currentTime);

                                                        for (int
i=0;i<listenerList.Count;i++)
                                       {

     TCSListener listener = (TCSListener)listenerList[i];
                                            if (listener != null)
                                            {

listener.RemoveTCSObject(removeObject);
                                            }
                                                        }

     lock(objectSync)
                                       {

     objectList.Remove(removeObject.getSessionID());
                                                        }
                                                        break;
                                                  case
TCSObject.TUIO_ADDED:
                                                        TCSObject
addObject = new
TCSObject(currentTime,tobj.getSessionID(),tobj.getTCSID(),tobj.
getX(),tobj.getY(),tobj.getAngle());

     lock(objectSync)
                                       {

     objectList.Add(addObject.getSessionID(),addObject);
                                                        }
```

```
                                                    for (int
i=0;i<listenerList.Count;i++)
                                    {

     TCSListener listener = (TCSListener)listenerList[i];
                                                        if
(listener!=null) listener.AddTCSObject(addObject);
                                                    }
                                                    break;
                                                default:
                                                    TCSObject
updateObject = getTCSObject(tobj.getSessionID()); //gets from
the list of TCSobjects the object corresponding to the session
ID that is from the received parameters from the server
                                    if ((tobj.getX() !=
updateObject.getX() && tobj.getXSpeed() == 0) || (tobj.getY()
!= updateObject.getY() && tobj.getYSpeed() == 0)) //sees if the
new object X and Y pos (tobj) than the old object
(updateObject) and sees if the new object (tobj) x and y speed
equal 0
                                    {

updateObject.update(currentTime, tobj.getX(), tobj.getY(),
tobj.getAngle(), tobj.GetBatteryState(), tobj.GetLoad());
//this is to do the update just for the position since the
object already stopped
                                    }
                                    else
                                    {

updateObject.update(currentTime, tobj.getX(), tobj.getY(),
tobj.getAngle(), tobj.getXSpeed(), tobj.getYSpeed(),
tobj.getRotationSpeed(), tobj.getMotionAccel(),
tobj.getRotationAccel(), tobj.GetBatteryState(),
tobj.GetLoad());
                                    }

                                                    for (int
i=0;i<listenerList.Count;i++)
                                    {

     TCSListener listener = (TCSListener)listenerList[i];
                                        if (listener != null)
                                        {

listener.UpdateTCSObject(updateObject);
                                        }
                                                    }
                                                    break;
                                                }
                                    }
                }
```

APPENDIX IV – SVG FILE READER PSEUDO-CODE

Create a list to hold each polygon of the map

The **Constructor** method receives as a parameter the path to the file it is supposed to read, inside the constructor there is:

Initialize the list;
Create and **Initialize** an auxiliary list to hold each *string* containing the map's polygon information;
Create and **Initialize** the variable which will hold the content within the file of the path given through the constructor's parameter;
Create and **Initialize** a navigator variable in order to go through the content hold in the variable depicted above.
Using the navigator variable, move to "svg/g[@id='<layer name>']", where <layer name> is the name of the layer in the svg file where the map layout was drawn
While the navigator can move through the information inside that layer **do**
Add to the list, that holds the *strings* containing the map's polygon information, the information inside the attribute "d".
End While
Call helper method to translate the *strings* into actual polygon, passing as a parameter the auxiliary list.

The **Helper** method receives as a parameter list of *strings* containing the map's polygons:

For each *string* inside the list **do**
Create and **Initialize** a list that holds all the points' coordinates of the polygon
Trim the *string* to get a each point's coordinates in an *array*
For each point's coordinates in the *array* **do**
Split the point's coordinates into x component and y component
Add these components to the list that holds the points' coordinates of the polygon as a 2D vector.
End For
Create a new polygon object with the points' coordinates inside the list
Add to the polygon to the list that holds each polygon of the map
End For each

Create a method that returns the list of polygons of the map.

APPENDIX V – POLYGON OBJECT CODE

Next is a description of how to create the class of this object.

The class properties are:

List to hold the points of the polygon
A 2D vector to hold the coordinates of the polygon's centre point
A float to hold the area value

The **Constructor** method receives as a parameter a list of coordinate points. The code inside it is:

Assign the received list of coordinate points to the list holding the points of the polygon

For each point inside the list of coordinate points **do**

Sum += X coordinate of the current point * Y coordinate of the next point – X coordinate of the next point * Y coordinate of the current point;

XCoord += (X coordinate of the current point + X coordinate of the next point) * (X coordinate of the current point * Y coordinate of the next point - X coordinate of the next point * Y coordinate of the current point);

YCoord += (Y coordinate of the current point + Y coordinate of the next point) * (X coordinate of the current point * Y coordinate of the next point - X coordinate of the next point * Y coordinate of the current point);

End For

The area value is the **result** of the following operation Sum / 2;

The centre point 2D vector is the following for the *x* coordinate XCoord / (6 * area) and the *y* coordinate is the YCoord / (6 * area)

Create methods to return the values of the area and centre point, as well as the list of points that form the polygon.

The C# code is has follows:

```
class PolygonObject
    {
        #region Properties
        private List<Vector2> polygon;
        private Vector2 centerPoint;
        private float area;
        #endregion

        #region Constructor Methods
        public PolygonObject(List<Vector2> poli)
        {
            float sumArea = 0, centerX = 0, centerY = 0;

            polygon = poli;
            for(int i=0; i<poli.Count-1;i++)
            {
                sumArea += poli[i].X * poli[i + 1].Y - poli[i +
1].X * poli[i].Y;
            }
            area = sumArea/2;

            for(int i=0; i<poli.Count-1;i++)
            {
                centerX += (poli[i].X + poli[i + 1].X) *
(poli[i].X * poli[i + 1].Y - poli[i + 1].X * poli[i].Y);
                centerY += (poli[i].Y + poli[i + 1].Y) *
(poli[i].X * poli[i + 1].Y - poli[i + 1].X * poli[i].Y);
            }
            centerPoint = new Vector2(centerX / (6 * area),
centerY / (6 * area));
        }
        #endregion

        #region Properties Methods
        public List<Vector2> Polygon
        {
            get { return polygon; }
        }
```

```
        public Vector2 CenterPoint
        {
            get { return centerPoint; }
        }

        public float Area
        {
            get { return area; }
        }
        #endregion
    }
```

APPENDIX VI – NAVIGATION MESH FILE PARSER CODE

It was used C# with the help of the XPath library.

Create two lists one that holds all the nodes and one to hold the edge

The **Constructor** method receives as a parameter the path to the XML file containing the nodes and edges of the navigation mesh. Inside the constructor there's the following code:

Initialize the two previously created lists
Create and **Initialize** a variable to hold the content within the file;
Create and **Initialize** a navigator variable in order to go through the content hold in the variable depicted above;
Use the navigator to go to the "navpath/nodes";
Call the **GetNodes** method passing as a parameter the navigator position
Sort the node list by ID
Use the navigator to go to the "navpath/edges";
Call the **GetEdges** method passing as a parameter the navigator position

The **GetNodes** method receives as a parameter the content navigator position, this method serves as a means to parse the *string* contents of the file to get the nodes of the navigation mesh.

While the navigator can move through each node information inside the nodes element **do**
Initialize an identification variable with the value from the attribute "id" of the current node
Create a polygon object
Verify **if** the current node element name is "path" **then**
Assign to the polygon object the result given by **calling** the method **GetPoli** (has was described in the SVG file reader as a helper method)
Else verify **if** the current node element name is "rect" **then**
Create a variable to hold the x coordinate position and **assign** the value from the attribute "x" of the element
Create a variable to hold the y coordinate position and **assign** the value from the attribute "y" of the element
Create a variable to hold the width of the rectangle and **assign** the value from the attribute "width" of the element
Create a variable to hold the height of the rectangle and **assign** the value from the attribute "height" of the element

Assign to the polygon object the result given by **calling** the method **GetRect** by passing as arguments the values from the x and y coordinates, as well as the width and height

Create a node **assigning** to it the identification variable and the polygon object

End While

The **GetRect** method receives as a parameter the values of the x and y coordinates as well as the width and height of the rectangle.

Create and **initialize** a list to hold the vertex points of the rectangle;

Add to the list the vertex point coordinates given by the "x" and "y" values received;

Add to the list the vertex point coordinates given by the "x + width" and "y" values;

Add to the list the vertex point coordinates given by the "x + width" and "y + height" values;

Add to the list the vertex point coordinates given by the "x" and "y + height" values;

Add to the list the vertex point coordinates given by the "x" and "y" values received;

Create a polygon object passing as an argument the list and then **return** the polygon object;

The **GetEdges** method receives as a parameter content from the navigator position, this method serves as a means to parse the *string* contents of the file to get the edges of the navigation mesh;

While the navigator can move through each node's information inside the nodes element **do**

Create a variable to hold the value of "from" and **assign** the value from the attribute "from" of the element

Create a variable to hold the value of "to" and **assign** the value from the attribute "to" of the element

Create a node and **assign** to it the result value from the **search** done on the list of nodes to find the node with the same ID as the "from" value

Create a node and **assign** to it the result value from the **search** done on the list of nodes to find the node with the same ID as the "to" value

Calculate the cost of traversing the two nodes by checking for the distance between the two polygons centre

Create a line segment that holds the values of the two position points and **assign** the values from the attribute "sharedseg" of the element

Create a new edge with the values of "from", "to" "cost" and "line segment" **assigning** the edge to the list of edges

Create methods to return the values of the lists of nodes and edges

Code created in C# with the use of XPath library

```
class ReadXML<N,E>
        where N : NavigationNode
        where E : CommunSharedEdge
    {
        private List<N> nodes;
        private List<E> edges;

        public ReadXML(string path)
        {
            nodes = new List<N>();
            edges = new List<E>();
```

```
            XPathDocument doc = new XPathDocument(path);
            XPathNavigator nav = doc.CreateNavigator();

            //To get the nodes from the XML
            XPathNavigator result =
nav.SelectSingleNode("path/nodes");
            if (result.HasChildren)
            {
                GetNodes(result);
                nodes.Sort(new Comparison<N>(SortFunction));
            }
            result = nav.SelectSingleNode("path/edges");
            if (result.HasChildren)
            {
                GetEdges(result);
            }
        }

        #region Properties Methods
        public List<N> Nodes
        {
            get { return nodes; }
        }

        public List<E> Edges
        {
            get { return edges; }
        }
        #endregion

        #region Helper Methods
        private void GetNodes(XPathNavigator result)
        {
            result.MoveToFirstChild();
            do
            {
                int idx = int.Parse(result.GetAttribute("id",
""));
                PolygonObject poli = null;

                if (result.LocalName == "path")
                {
                    poli = GetPoli(result.GetAttribute("d",
""));
                }
                else if (result.LocalName == "rect")
                {

                    float x =
float.Parse(result.GetAttribute("x", "").Replace('.', ','));
                    float y =
float.Parse(result.GetAttribute("y", "").Replace('.', ','));
                    float w =
float.Parse(result.GetAttribute("width", "").Replace('.',
','));
                    float h =
float.Parse(result.GetAttribute("height", "").Replace('.',
','));
                    poli = GetRect(x, y, w, h);
                }
                N node = (N)(new NavigationNode(idx, poli));
                nodes.Add(node);
            }
```

```
            while (result.MoveToNext());
        }

        private void GetEdges(XPathNavigator result)
        {
            result.MoveToFirstChild();
            do
            {
                int from =
int.Parse(result.GetAttribute("from", ""));
                int to = int.Parse(result.GetAttribute("to",
""));
                N nodeFrom = nodes.Find(
                                                delegate(N n)
                                                {
                                                    return
n.GetIndex() == from;
                                                }
                                            );
                N nodeTo = nodes.Find(
                                                delegate(N n)
                                                {
                                                    return
n.GetIndex() == to;
                                                }
                                            );
                float cost =
Vector2.Distance(nodeFrom.Position, nodeTo.Position);

                Line shareSeg =
GetSharedSegment(result.GetAttribute("sharedseg", ""));
                edges.Add((E)new CommunSharedEdge(from, to,
cost,shareSeg));
            }
            while (result.MoveToNext());
        }

        private PolygonObject GetPoli(string poli)
        {
            List<Vector2> poligon = new List<Vector2>();
            string str = poli.TrimStart('M', 'm').TrimEnd('Z',
'z').Trim();
            string[] temp = str.Split(' '); //Here the string
stays like "12.500,45.601"

            for (int i = 0; i < temp.Count(); i++)
            {
                string[] temp1 = temp[i].Split(',');
                poligon.Add(new Vector2(
                    float.Parse(temp1[0].Replace('.', ',')),
                    float.Parse(temp1[1].Replace('.', ','))));
            }
            return new PolygonObject(poligon);
        }

        private PolygonObject GetRect(float x, float y, float
w, float h)
        {
            List<Vector2> poligon = new List<Vector2>();

            poligon.Add(new Vector2(x, y));
            poligon.Add(new Vector2(x + w, y));
            poligon.Add(new Vector2(x + w, y + h));
```

```
            poligon.Add(new Vector2(x, y + h));
            poligon.Add(new Vector2(x, y));

            return new PolygonObject(poligon);
        }

        private Line GetSharedSegment(string sharedSeg)
        {
            Vector2 p1, p2;
            string[] temp = sharedSeg.Split(';');

            string[] point = temp[0].Split(',');
            p1 = new Vector2(float.Parse(point[0].Replace('.',
','')), float.Parse(point[1].Replace('.', ',')));
            point = temp[1].Split(',');
            p2 = new Vector2(float.Parse(point[0].Replace('.',
','')), float.Parse(point[1].Replace('.', ',')));

            return new Line(p1, p2);
        }

        public int SortFunction(N obj1, N obj2)
        {
            return obj1.GetIndex().CompareTo(obj2.GetIndex());
        }
        #endregion
    }
```

APPENDIX VII – PATHPLANNER CLASS METHODS CODE

PathOfNodes method Code

```
/// <summary>
        /// Gets a list of NavigationNodes from the location of
the entity to the destination
        /// </summary>
        /// <remarks>The path is smoothed according to the
choosen path algorithm.
        /// The path those not contain the starting position,
nor the end position it must
        /// be added.</remarks>
        public List<NavigationNode> PathOfNodes
        {
            get
            {
                List<int> pathIdxNodes;
                List<NavigationNode> pathNodes = new
List<NavigationNode>();

                //Get the path of nodes
                pathIdxNodes = search.PathOfNodes;

                //If there´s at least one node in the path
                if (pathIdxNodes.Count != 0)
                {
                    for (int i = 0; i < pathIdxNodes.Count;
i++)
                    {
```

```
pathNodes.Add(graph.GetNode(pathIdxNodes[i]));
                    }
                }

                return pathNodes;
            }
        }
```

PathOfEdgesToPosition method Code

```
/// <summary>
        /// Gets a list of Vector2 from the location of the
entity to the destination
        /// </summary>
        /// <remarks>The path is smoothed according to the
choosen path algorithm</remarks>
        public List<Vector2> PathOfEdgesToPosition
        {
            get
            {
                List<E> edges = search.PathOfEdges;
                List<Vector2> path = new List<Vector2>();

                //Get the path of Edges and transform it to a
path of positions
                for (int i = 0; i < edges.Count; i++)
                {

path.Add(graph.GetNode(edges[i].GetNodeFrom()).Position);
                    path.Add(Vector2.Lerp(
                        edges[i].SharedEdge.startPos,
                        edges[i].SharedEdge.endPos,
                        0.5f));

                    if (i == edges.Count - 1)
                    {

path.Add(graph.GetNode(edges[i].GetNodeTo()).Position);
                    }
                }

                //If there´s at least one node in the path
                if (path.Count != 0)
                {
                    //Add the posistion of the owner entity
                    path.Insert(0, owner.Position);

                    //Add the edge between the last node of the
path and the destination
                    path.Add(destination);

                    //Smooth the path
                    PathMeshSmooth.smoothPath(owner, path);
                }
                else //The path is a straight line
                {
                    path.Add(owner.Position);
                    path.Add(destination);
                }

                if (path.Count > 3)
                {
```

```
                    int detail = 5;
                    List<Vector2> allPoints = new
List<Vector2>();

                    for (int i = 1; i < path.Count - 2; i++)
                    {
                        List<Vector2> part;
                        if (i == 1)
                        {
                            Vector2 temp =
PathMeshSmooth.PointsAlongLine(path[i - 1], path[i], 1,1);

                            part =
PathMeshSmooth.InterpolateCR(temp, path[i - 1], path[i], path[i
+ 1], detail);
                            allPoints.AddRange(part);
                        }

                        part =
PathMeshSmooth.InterpolateCR(path[i - 1], path[i], path[i + 1],
path[i + 2], detail);
                        allPoints.AddRange(part);

                        if (i == path.Count - 3)
                        {
                            Vector2 temp =
PathMeshSmooth.PointsAlongLine(path[i + 1], path[i+2], 1,2);

                            part =
PathMeshSmooth.InterpolateCR(path[i], path[i + 1], path[i + 2],
temp, detail);
                            allPoints.AddRange(part);
                        }
                    }
                    return allPoints;
                }
                return path;
            }
        }
```

GetNowPathofNodesToPosition method Code

```
/// <summary>
        /// This method performs a full search for a path to
reach the target position and returns
        /// the path of positions to it if it exists
        /// </summary>
        /// <param name="position">The destination of the
search</param>
        /// <returns>The list of positions if the search was
succesful, null if it wasn´t</returns>
        public List<Vector2>
GetNowPathOfNodesToPosition(Vector2 position)
        {
            int closestNodeToEntity, closestNodeToDestination;

            destination = position;

            //If the entity can reach the destination directly,
there´s no need to request a search
            if (owner.CanMoveBetween(owner.Position,
destination) == true)
            {
```

```
                List<Vector2> pathPositions = new
List<Vector2>();
                pathPositions.Add(owner.Position);
                pathPositions.Add(destination);
                return pathPositions;
            }

            //Get the closest node to the entity
            closestNodeToEntity =
ClosestNodeToPosition(owner.Position);
            if (closestNodeToEntity == NoNodeFound)
            {
                return null;
            }

            //Get the cloest node to the destination
            closestNodeToDestination =
ClosestNodeToPosition(destination);
            if (closestNodeToDestination == NoNodeFound)
            {
                return null;
            }

            //Do the full search and if succesful return the
path of positions
            search.Initialize(closestNodeToEntity,
closestNodeToDestination);
            if (search.Search() ==
SearchState.CompletedAndFound)
            {
                List<NavigationNode> nodes = PathOfNodes;
                List<Vector2> pathPositions = new
List<Vector2>();
                pathPositions.Add(owner.Position);
                foreach (NavigationNode n in nodes)
                {
                    pathPositions.Add(n.Position);
                }
                pathPositions.Add(destination);
                return pathPositions;
            }
```

GetNowPathToPosition method code

```
/// <summary>
        /// This method performs a full search for a path to
reach the target position and returns
        /// the path of Vector2 to it if it exists
        /// </summary>
        /// <param name="position">The destination of the
search</param>
        /// <returns>The list of Vector2 if the search was
succesful, null if it wasn´t</returns>
        public List<Vector2> GetNowPathToPosition(Vector2
position, out double ETA)
        {
            int closestNodeToEntity, closestNodeToDestination;
            destination = position;
            //If the entity can reach the destination directly,
there´s no need to request a search
            if (owner.CanMoveBetween(owner.Position,
destination) == true)
            {
```

```
                ETA = ((Vector2.Distance(owner.Position,
destination)*8.5d)/196) / owner.Speed;

                List<Vector2> temp = new List<Vector2>();
                temp.Add(owner.Position);
                temp.Add(destination);
                return temp;
            }

            //Get the closest node to the entity
            closestNodeToEntity =
ClosestNodeToPosition(owner.Position);
            if (closestNodeToEntity == NoNodeFound)
            {
                ETA = double.PositiveInfinity;
                return null;
            }

            //Get the cloest node to the destination
            closestNodeToDestination =
ClosestNodeToPosition(destination);
            if (closestNodeToDestination == NoNodeFound)
            {
                ETA = double.PositiveInfinity;
                return null;
            }

            //Do the full search and if succesful return the
path of positions
            search.Initialize(closestNodeToEntity,
closestNodeToDestination);
            if (search.Search() ==
SearchState.CompletedAndFound)
            {
                float dist=0;
                dist += Vector2.Distance(owner.Position,
graph.GetNode(search.PathOfNodes[0]).Position);
                for (int i = 0; i < search.PathOfEdges.Count;
i++)
                {
                    dist += search.PathOfEdges[i].Cost;
                }
                dist += Vector2.Distance(destination,
graph.GetNode(search.PathOfNodes[search.PathOfEdges.Count]).Pos
ition);

                ETA = (dist*8.5d)/196 / owner.Speed;

                return PathOfEdgesToPosition;
            }
            ETA = double.PositiveInfinity;
            return null;
        }
```

APPENDIX VIII – PATHMESHSMOOTH CLASS METHODS CODE

GetNowPathToPosition method code

smoothPath method code

```
public static void smoothPath(ATSObject entity, List<Vector2>
path)
        {
            int i, j;

            i = 0;
            j = 2;

            //Move through all the edges of the path testing if
the entity can move through 2 non-consecutive points
            //and the edge information is the same in the edges
to smooth
            while (j < path.Count)
            {
                if (entity.CanMoveBetween(path[i], path[j]))
                {
                    path[i + 1] = path[j];
                    path.RemoveAt(j);
                }
                else
                {
                    i = j;
                    j += 2;
                }
            }
        }
```

InterpolateCR method code

```
 public static List<Vector2> InterpolateCR(Vector2 v1, Vector2
 v2, Vector2 v3, Vector2 v4, int detail)
         {
             List<Vector2> list = new List<Vector2>();

             for (int i = 0; i < detail; i++)
             {
                 Vector2 newPoint = Vector2.CatmullRom(v1, v2,
 v3, v4, (float)i / (float)detail);
                 list.Add(newPoint);
             }
             return list;
         }
```

PointsAlongLine method code

```
public static Vector2 PointsAlongLine(Vector2 start, Vector2
end, int spacing, int side)
        {
            float xDifference = end.X - start.X;
            float yDifference = end.Y - start.Y;
```

```
            float absoluteXdifference = Math.Abs(start.X -
end.X);
            float absoluteYdifference = Math.Abs(start.Y -
end.Y);

            float lineLength =
(int)Math.Sqrt((Math.Pow(absoluteXdifference, 2) +
Math.Pow(absoluteYdifference, 2))); //pythagoras
            float steps = lineLength / spacing;
            float xStep = xDifference / steps;
            float yStep = yDifference / steps;

            Vector2 result = new Vector2();

            if (side == 1)
            {
                result.X = start.X - (xStep);
                result.Y = start.Y - (yStep);
            }
            else if (side == 2)
            {
                result.X = end.X + (xStep);
                result.Y = end.Y + (yStep);
            }
            return result;
        }
```

APPENDIX IX – SENSORLIST CLASS METHODS CODE

UpdateSensorsPosition method

```
public void UpdateSensorsPosition(Vector2 rotationCenter, float
xPosCam, float yPosCam, float camRotAng)
        {
            for (int i = 0; i < sensors.Count; i++)
            {
                Vector2 origSensorPos =
sensors[i].GetPosition();

                //Function to translate coordinates from the
Map to the window
                Vector2 sensorPos = new Vector2(origSensorPos.X
- xPosCam + 400, -(origSensorPos.Y - yPosCam - 400));

                //Function to make a translation of the sensor
when the map rotates
                float circle = MathHelper.Pi * 2; //to not have
values superior than the 2Pi
                float RotationAngle = camRotAng % circle;
                RotatePoints(ref rotationCenter, -
RotationAngle, ref sensorPos);


sensors[i].UpdateRangeFinder(sensors[0].Distance, sensorPos,
sensors[0].GetAngle() - RotationAngle);
            }
        }
        #endregion
```

RotatePoints method

```
/// <summary>
        /// Method that makes the translation of the position
coordinates for the sensor
        /// </summary>
        /// <param name="origin">The origin point of the
rotation</param>
        /// <param name="radians">the amount of
rotation</param>
        /// <param name="Vectors">The list of the sensors
position vectors to rotate</param>
        private static void RotatePoints(ref Vector2 origin,
float radians, ref Vector2 vector)
        {
            Matrix myRotationMatrix =
Matrix.CreateRotationZ(radians);

            // Rotate relative to origin.
            Vector2 rotatedVector =
                Vector2.Transform(vector - origin,
myRotationMatrix);

            // Add origin to get final location.
            vector = rotatedVector + origin;
        }
```

APPENDIX X – RANGEFINDERSENSOROBJECT CLASS METHODS CODE

UpdateText method

```
        /// <summary>
        /// Method that updates the position and text that is
going to be showed on the screen for the sensor
        /// </summary>
        private void UpdateText()
        {
            fontText = distance.ToString("F2") + "cm";
            fontPosition = new Vector2(sensorRect.X +
(float)(2*radius * Math.Cos(MathHelper.Pi + angle)),
sensorRect.Y + (float)(2*radius * Math.Sin(MathHelper.Pi +
angle)));
        }
```

ChangeColour method

```
/// <summary>
        /// Method that changes the color of the lines for the
sensor on the screen from green to red
        /// </summary>
        private void ChangeColour()
        {
            float amount;
```

```
            if (distance > totalDistance/3)
            {
                amount = (distance - totalDistance/3) /
(totalDistance/3);
                sensorColor = Color.Lerp(new Color(255, 255,
0), new Color(0, 255, 0), amount); //From green to yellow
            }
            else
            {
                amount = distance / (totalDistance/3);
                sensorColor = Color.Lerp(new Color(255, 0, 0),
new Color(255, 255, 0), amount); //From yellow to red
            }
        }
```

APPENDIX XI – INTERSECTIONPOINT METHOD CODE

```
public static bool IntersectionPoint(Line firstLine, Line
secondLine)
        {
            double Ua, Ub;

            // Equations to determine whether lines intersect
            Ua = ((secondLine.endPos.X - secondLine.startPos.X)
* (firstLine.startPos.Y - secondLine.startPos.Y) -
(secondLine.endPos.Y - secondLine.startPos.Y) *
(firstLine.startPos.X - secondLine.startPos.X)) /
                    ((secondLine.endPos.Y -
secondLine.startPos.Y) * (firstLine.endPos.X -
firstLine.startPos.X) - (secondLine.endPos.X -
secondLine.startPos.X) * (firstLine.endPos.Y -
firstLine.startPos.Y));

            Ub = ((firstLine.endPos.X - firstLine.startPos.X) *
(firstLine.startPos.Y - secondLine.startPos.Y) -
(firstLine.endPos.Y - firstLine.startPos.Y) *
(firstLine.startPos.X - secondLine.startPos.X)) /
                    ((secondLine.endPos.Y -
secondLine.startPos.Y) * (firstLine.endPos.X -
firstLine.startPos.X) - (secondLine.endPos.X -
secondLine.startPos.X) * (firstLine.endPos.Y -
firstLine.startPos.Y));

            if (Ua >= 0.0f && Ua <= 1.0f && Ub >= 0.0f && Ub <=
1.0f)
            {
                /*double x = firstLine.startPos.X + Ua *
(firstLine.endPos.X - firstLine.startPos.X);
                double y = firstLine.startPos.Y + Ua *
(firstLine.endPos.Y - firstLine.startPos.Y);*/
                return true;
            }
            else
            {
                return false;
            }
        }
```

APPENDIX XII – FEEDFORWARD METHOD CODE

```
/// <summary>
        /// Method to get the final position of the wheels in
order to give feedforward information
        /// </summary>
        /// <returns>List of coordinates with the final
position of the wheels</returns>
        public List<Vector2> FeedForward()
        {
            float time;
            const float ue = 1, g = 9.8f, k = 0.0005f;
            List<Vector2> list = new List<Vector2>();
            float frictionAcceleration = -ue * k * g;
            float wheel1Angle = wheels[0].Angle, wheel2Angle =
wheels[1].Angle;
            float wheel1Velocity = wheels[0].Velocity,
wheel2Velocity=wheels[1].Velocity;
            Vector2 fPos= new Vector2();


//*****************************Moving/strafing****************
**********
            if (wheel1Angle == wheel2Angle)
            {
                time = currentSpeed / (ue * g * k);

                fPos.X = currentSpeed *
(float)Math.Cos((double)(wheel1Angle)) * time + 0.5f *
frictionAcceleration * (float)Math.Cos((double)(wheel1Angle)) *
(float)Math.Pow((double)time, (double)2);
                fPos.Y = currentSpeed *
(float)Math.Sin((double)(wheel1Angle)) * time + 0.5f *
frictionAcceleration * (float)Math.Sin((double)(wheel1Angle)) *
(float)Math.Pow((double)time, (double)2);

                list.Add(fPos);
                return list;
            }

//************************************************************
**********


//*******************************Turning**********************
**********
            else if (wheel1Angle != wheel2Angle)
            {
                //---------------First wheel-----------------
                time = wheel1Velocity / (ue * g * k);

                fPos.X = currentSpeed *
(float)Math.Cos((double)(rotation + wheel1Angle)) * time + 0.5f
* frictionAcceleration * (float)Math.Cos((double)(rotation +
wheel1Angle)) * (float)Math.Pow((double)time, (double)2);
                fPos.Y = currentSpeed *
(float)Math.Sin((double)(rotation + wheel1Angle)) * time + 0.5f
* frictionAcceleration * (float)Math.Sin((double)(rotation +
wheel1Angle)) * (float)Math.Pow((double)time, (double)2);
```

```
                list.Add(fPos);
                //-----------------------------------------

                //---------------Second wheel-----------------
                time = wheel2Velocity / (ue * g * k);

                fPos.X = currentSpeed *
(float)Math.Cos((double)(rotation + wheel2Angle)) * time + 0.5f
* frictionAcceleration * (float)Math.Cos((double)(rotation +
wheel2Angle)) * (float)Math.Pow((double)time, (double)2);
                fPos.Y = currentSpeed *
(float)Math.Sin((double)(rotation + wheel2Angle)) * time + 0.5f
* frictionAcceleration * (float)Math.Sin((double)(rotation +
wheel2Angle)) * (float)Math.Pow((double)time, (double)2);

                list.Add(fPos);
                //-----------------------------------------

                return list;
            }

            return null;
        }
```

APPENDIX XIII – DRAW METHODS CODE FOR THE ATSICON

```
public void Draw(SpriteBatch sb, float angle, double dist)
        {
            double h = ((dist / 300) / 0.001) * 3.545;
            sb.Draw(spriteATS, position, null, Color.White,
0.0f, new Vector2(width / 2, height / 2), 1.0f,
SpriteEffects.None, 0.0f);

            sb.Draw(arrow, new Rectangle((int)(position.X +
(width / 2) * Math.Cos((double)angle - MathHelper.PiOver2)),
(int)(position.Y + (height / 2) * Math.Sign((double)angle -
MathHelper.PiOver2)), 20, (int)h), null, Color.White, angle,
new Vector2(arrow.Width / 2, arrow.Height), SpriteEffects.None,
1);
        }

        public void Draw(SpriteBatch sb, float angle1, float
dist1, float angle2, float dist2)
        {
            double h1 = ((dist1 / 300) / 0.001) * 3.545;
            double h2 = ((dist2 / 300) / 0.001) * 3.545;

            sb.Draw(spriteATS, position, null, Color.White,
0.0f, new Vector2(width / 2, height / 2), 1.0f,
SpriteEffects.None, 0.0f);
            sb.Draw(arrow, new Rectangle((int)(position.X +
(width / 2) * Math.Cos((double)angle1 - MathHelper.PiOver2)),
(int)((position.Y - (0.3420412 * height)) + (height *
0.1579588) * Math.Sign((double)angle1 - MathHelper.PiOver2)),
20, (int)h1), null, Color.White, angle1, new
Vector2(arrow.Width / 2, arrow.Height), SpriteEffects.None, 1);
```

```
            sb.Draw(arrow, new Rectangle((int)(position.X +
(width / 2) * Math.Cos((double)angle2 - MathHelper.PiOver2)),
(int)((position.Y + (0.3420412 * height)) + (height *
0.1579588) * Math.Sign((double)angle2 - MathHelper.PiOver2)),
20, (int)h2), null, Color.White, angle2, new
Vector2(arrow.Width / 2, arrow.Height), SpriteEffects.None, 1);
        }
```

APPENDIX XIV – GENERATETRACKVERTICES METHOD CODE

```
public static List<Vector2> GenerateTrackVertices(List<Vector2>
basePoints)
        {
            float halfTrackWidth = 15.0f;
            //float textureLength = 0.5f;

            //float distance = 0;
            List<Vector2> verticesList = new List<Vector2>();

            for (int i = 0; i < basePoints.Count - 1; i++)
            {
                Vector3 movDir = new Vector3(basePoints[i+1] -
basePoints[i], 0);
                Vector3 sideDir =
Vector3.Cross(Vector3.Backward, movDir);
                sideDir.Normalize();

                Vector3 outerPoint = new Vector3(basePoints[i],
0) + sideDir * halfTrackWidth;
                Vector3 innerPoint = new Vector3(basePoints[i],
0) - sideDir * halfTrackWidth;

                verticesList.Add(new Vector2(innerPoint.X,
innerPoint.Y));
                verticesList.Add(new Vector2(outerPoint.X,
outerPoint.Y));
                verticesList.Add(basePoints[i + 1]);
            }

            return verticesList;
        }
```

APPENDIX XV – CLASS DIAGRAM OF THE SA INTERFACE