University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Computer Science	Spring semester, 2011 Open
Writer: Rune Johansen	………………………………………… Rune Johansen
Faculty supervisor: Chunming Rong External supervisor(s): Einar Landre, Statoil	
Titel of thesis: Humen Robot Interaction In Multi-Agent Systems	
Credits (ECTS): 30	
Key words: Intelligent Agent, Human Robot Interaction, HRI, JACK Agent Framework, Lego Mindstorms, LeJos, Java	Pages 83 + enclosure: CD Stavanger, June 8, 2011 Date/year

MIDMAS – Master Thesis

Human Robot Interaction in Multi-Agent Systems

Author:
Rune Johansen

Supervisor
Einar Landre

Supervisor:
Chunming Rong

8. June 2011

Abstract

The oil and gas industry experience an increased dependency on IT and particular software based capabilities to achieve its business objectives. Core business processes such as exploration, well construction, production optimization and operations are all fueled by software and information technology. In coming years we will see that software will fill more and more advanced features, including central control functions in autonomous and collaborative robots and it is believed that agent technology may be of use in this scenario.

The practical benefit from goal oriented systems is a simplification of the human-machine interface. A goal oriented system is able to communicate and react to events in its environment in context of their goals. This is the primary driver for autonomous systems: Simplifying and securing operation of machines in an unstructured / highly dynamic environment.

Human Robot Interaction (HRI) is an important area in development of autonomous robot systems where an operator is present. The design and solution for the HRI will be crucial to the systems performance and robustness. An operator can be relieved of stress as well as have his focus directed to important and critical information at any given time. In this thesis I will look at how intelligent agents can be used to implement a system for controlling autonomous robots and how this can provide a good solution for HRI challenges. To achieve this a multi-agent solution controlling Lego Mindstorms robots has been developed in cooperation with Eirik Nordbø.

The solution is based on three Lego robots operating on a line-based grid. One robot is set to explore the grid, finding objects, and sharing this information (beliefs) with a second robot which is responsible for collecting and delivering these objects to a robot in charge of sorting them according to color. This solution enables investigation of several challenges in relation to intelligent software agents combined with autonomous robot/machine systems, such as human robot interaction and inter-agent communication/coordination.

The agent system is developed using the Prometheus methodology for the design and the JACK Intelligent Agents framework for the implementation. Regular Java is used combined with the LeJos - Java For Mindstorms framework to implement the robot side of the system.

Preface

This master thesis has been written as a continuance of a preliminary project we performed in cooperation with Statoil during the autumn semester of 2010. The preliminary project was a collaboration between Rune Johansen and Eirik Nordbø where the objective was to do a feasibility study of interfacing an agent platform (JACK) with a robot control system (LEGO Mindstorms). The cooperation has continued as we have developed a common technical solution with separate angle of approach for our master thesis. Eirik has focused on inter agent coordination and communication while Rune‘s area of focus has been human robot interaction, both in relation to multi-agent systems controlling robots or machines.

We would like to thank our supervisor Einar Landre at Statoil for all help and support during our work with this thesis and the opportunity to gain insight into the exiting field of intelligent agent technology. We would also like to thank professor Chunming Rong at the University of Stavanger and last but not least Jossi for great coffee and moral support throughout the semester.

Stavanger, 8 June 2011

Rune Johansen

Contents

List of Figures

List of Tables

1 Introduction

This Master's thesis has been written in collaboration with Statoil, an international energy company with operations in 34 countries. Building on more than 35 years of experience from oil and gas production on the Norwegian continental shelf, they are committed to accommodating the world's energy needs in a responsible manner, applying technology and creating innovative business solutions. They are headquartered in Norway with 20,000 employees worldwide, and are listed on the New York and Oslo stock exchanges. [2]

As a technology based energy company, Statoil experience an increased dependency on IT and particular software based capabilities to achieve its business objectives. Core business processes such as exploration, well construction, production optimization and operations are all fueled by software and information technology.

At the same time Statoil, as all other technology based companies, experience that software is becoming the alter ego of their prime technology. Software provides some of the core system functions in airplanes, cars and factories as more functions are automated and optimized.

This development also comes with a dark side, a side described as the paradox of automation. This paradox implies that the more sophisticated a system is, the more difficult it is to manage [3].

Very often a complex system or machine expects a human operator to take control when it moves outside its operational envelope. Experience indicates that this is not a good approach. Therefore we should look into how to construct more systems that are under human control and communicates with its human operator in term of intent and ability to solve a tasked mission. This communication between operator and system is known as Human Robot Interfacing (HRI) and the area of research involving this collaboration is important for the performance and viability of such systems. The Belief-Desire-Intention (BDI) model which is based on human behavior and reasoning can provide a very human like solution to the communication between machine and human operators. Using BDI agents in this context will be beneficial both on system and operator side by relieving operator decision load and stress, as well as improving the cooperation by providing a natural understanding of intentions and desires.

This takes us to autonomous systems and technologies where software is used to implement computer reasoning as well as more traditional automated control functions and the corresponding HRI.

1.1 Motivation

Automation of reactive behavior has its roots in the need to control dynamic systems. Dynamic systems are systems where the system state changes as a function of time, systems that can be described using differential equations.

The scientific platform for controlling dynamic systems is known as control theory or cybernetics, where feedback and/or feed-forward techniques are used to control a systems reactive response to external events for the purpose of keeping the system within its operational envelope. The forces motivating automation are many, but most often falls into one of the categories illustrated in Figure 1.

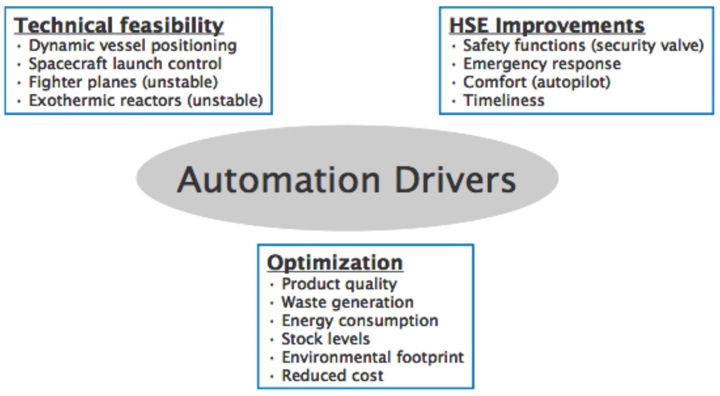


Figure 1: Forces motivating automation

Autonomous systems are motivated by the same forces, but the value from automating decisions, and moving to more goal oriented designs are easier to grasp using an example. The archetype example is human operation of complex processes or vehicles (robots) in unstructured and dynamic environments where time is a key. In these systems three concerns must be managed:

1. Communications loss. Communication links breaks and there is a need for the vehicle to maintain its own integrity as well as the integrity of its operating environment.
2. Communications latency. Remote operation over some distance has latency. In space applications like the Mars rovers, the latency is in minutes. For many earth bound applications latency in the magnitude of seconds might be unacceptable.
3. Operator information overload. Remote control is often more demanding than piloting the vehicle in a more traditional way. When a human is piloting a manned vehicle,vibrations, sounds and vision provide information that is easily lost in a remote control scenario, leading to unnecessary stress and mistakes.

By introducing autonomy, the vehicle (process) becomes able to store its mission objective (goal) and continuously assess its objective (goal) against environmental changes. Assuming the vehicle is an airplane, it will not only be able to detect a thunderstorm ahead, but it will be capable of validating the threat from the thunderstorm in context of its assigned mission.

In such situation the aircraft will recalculate its route, and validate if it has sufficient amount of fuel to pursue its original objective using the new route. In the case the aircraft is not able to accomplish its objective it will request permission from the human operator (pilot) to abort its assigned mission and update its objective to return home safely. The human in charge might reject or acknowledge such request.

Independent of what the human decides the role of the human operator has changed from flying the aircraft to perform mission management in collaboration with the vehicle. As a consequence the abstraction level in the man-machine interaction is raised, and the machine can interact with the human operator in a more human way.

Given industrial challenges such as smaller and more marginal resources (NCS), deeper waters, more demanding operational conditions (arctic), and the need for a reduced environmental footprint (regulations) will require smarter and more lightweight operational concepts. These new operational concepts will drive the development of more sophisticated and automated systems within all Statoil's core business processes (drilling, operations and production optimization), systems that need to be designed for unmanned / remote operations, systems utilizing the power of automated decision making, to enforce and secure prudent operations.

For Statoil to maintain a leading position as a technology based energy company it is important to understand and master autonomous systems including the software engineering challenges that comes with building goal oriented, collaborating physical systems to operate in unstructured environments. [4]

1.2 Problem definition

The following problem has been formulated in cooperation with our teaching supervisor at Statoil:

,,*The students should investigate some of the more complex problems related to autonomous systems, such as inter agent coordination and cooperation as well as agent human interfacing. This should be done through demonstrating how software agents can communicate with each other, with graphical interfaces, with external systems such as robots and human robot interaction (HRI).*"

To achieve this goal, Jack Intelligent Agent development platform [5] will be used together with Lego Mindstorm robots. [6]

Based on this problem definition and further discussion with our supervisors one implementation goal and two hypotheses where defined:

The implementation goal:

,,*Design and implement a proof-of-concept software with a graphical user interface (GUI), robot communication and human interaction. The GUI should implement two way communication with the agents; provide functionality for relevant HRI challenges and display real time information and results provided by the agents. The solution also needs to provide standard interfaces specifying the robot functionality required, as well as a Lego Mindstorms specific implementation of these interfaces.* "

This thesis will discuss external agent communication (GUI, ROBOTS AND HUMANS) and address the following research hypotheses:

Hypothesis 1: Human Robot Interfacing (HRI) can be improved by the use of software agents.

Hypothesis 2: A change in operational environment from more complex (unstructured) to simpler (structured) environments will have significant effect on the human-agent system interaction.

1.3 Report outline

Chapter 2, Software Agents
Agent-oriented software engineering is a rapidly developing area of research. This chapter will present basic agent theory, how they differ from traditional software paradigms and in which contexts they are useful.

Chapter 3, Human Robot Interfacing (HRI)
Gives a brief introduction into HRI followed by theory on how to evaluate and classify a systems HRI. Finally challenges and different approaches for design and implementation of HRI in autonomous systems are presented.

Chapter 4, Methodology and tools
This chapter describes the different methodologies and tools used for modeling and development of the agent-solution and robot application.

Chapter 5, Application
In order to design an application relevant to our problem definition and hypotheses, several approaches where considered. This chapter describes the different solutions.

Chapter 6, System Design
The chapter describes the different phases in our design using the Prometheus methodology. The overall system structure presented in this chapter is probably the most important and useful artifact resulting from the initial two phases of the Prometheus methodology.

Chapter 7, System Development
The different agents and how they communicate are presented in this chapter, including use case scenarios.

Chapter 8, Results
Summarizes the results of our work in light of the thesis hypotheses, as well as the challenges met.

Chapter 9, Conclusion
Based in our original problem definition, we here discuss the further implications of our result.

Chapter 10, Further work
We here present some possible approaches for further work.

2 Software Agents

The notion of AI was first introduced in the 1950's when it went from being fantasy/science fiction to becoming an actual research area. In addition to the design and implementation of robots to model the behavioral activities of humans, AI scientists eventually started to focus on implementing devices (software and hardware) that mimic human behavior and intelligence, Intelligent agents (agents) [7]. As of today no formal definition of an agent exists, but the Wooldridge and Jennigs definition is increasingly adopted.

The following definition is from (Wooldrigde 2002), which in turn is adapted from (Wooldridge and Jennigs 1995):

> „An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives".

Wooldridge distinguishes between an agent and an intelligent agent, which is further required to be reactive, proactive and social (Wooldridge 2002, page 23).

An intelligent agent is characterized as being autonomous, situated, reactive, proactive, flexible, robust and social [8]. These properties for an agent differ from traditional objects in several ways as shown in Table 2.

Property	**Agent**	**Object**
Autonomous	Agents are independent and make their own decisions.	Objects do not exhibit control over their own behavior because an object's method can be invoked by other entities.
Situated in an Environment	Agents tend to be used when the environment is dynamic, unpredictable and unreliable.	Objects tend to be used when the environment is static, predictable and reliable.
Reactive	Agents perceive changes in their environment and will respond to these changes to achieve goals.	Objects can be reactive, but their reactiveness is dependent on how well they manage changes in the environment.
Proactive	Agents are proactive because they persistently pursue goals, i.e. they have goal-directed behavior.	Objects are not proactive because they do not have goal-directed behavior and they lack reasoning ability.
Flexible	Agents are flexible because they can achieve goals in multiple ways.	Objects do not have the ability to choose between different ways to achieve a goal.
Robust	Agents recover from failure and choose another way to reach their current goals.	Objects are not flexible, and as a consequence they are less robust than agents.
Social	Agents have the ability to cooperate, coordinate and negotiate with each other to achieve common or individual goals.	Objects can exchange information and data with each other, but they lack the social aspect of the interaction.

Table 1: Difference between Agents and Objects [1]

2.0.1 Belief-Desire-Intention model

The Belief-Desire-Intention (BDI) model is based on human behavior and reasoning and can therefor provide a control mechanism for intelligent action. It is developed by Michael Bratman [9] to explain future-directed intention.

The Belief-Desire-Intention software model is a software model developed for intelligent agent programming. A BDI agent is a particular type of bounded rational software agent with some specific architectural components.

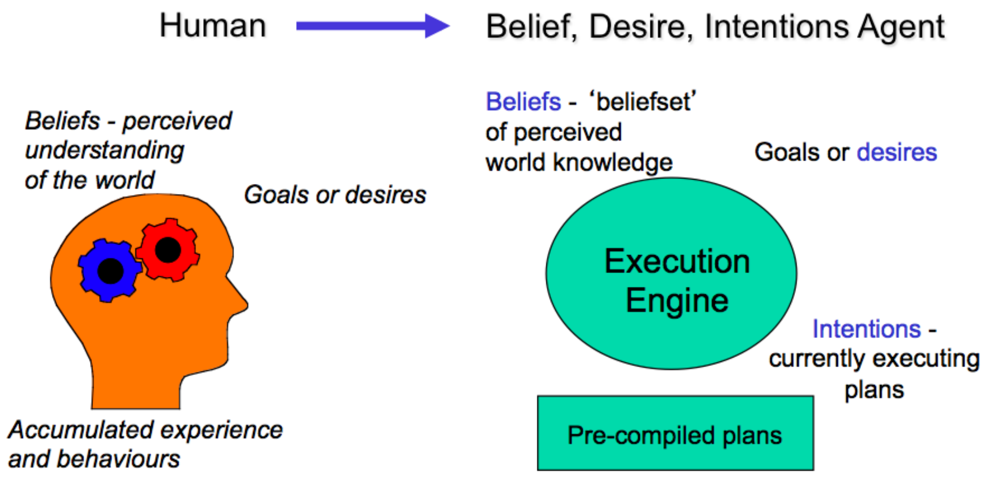

Human
Belief, Desire, Intentions Agent
Beliefs - perceived understanding of the world
Goals or desires
Accumulated experience and behaviours
Beliefs - 'beliefset' of perceived world knowledge
Goals or desires
Execution Engine
Intentions - currently executing plans
Pre-compiled plans

Figure 2: The BDI agent model

- Beliefs: Represent the informational state of an agent, what the agent believes about the world. The term belief is used instead of knowledge as the beliefs may be false although believed true by the agent. Beliefs are organized in Beliefsets.
- Desires: Represents the motivational state of an agent, objectives or situations the agent would like to accomplish or bring about. An agent can have goals which are desires actively pursued by the agent.
- Intentions: Represent the deliberative state of an agent, what an agent has chosen to do to accomplish a goal/desire. Plans are sequences of actions which an agent can use to fulfill it's intentions.
- Events: Events are the triggers for reactive activity by an agent. An event may change beliefs, update goals or trigger plans.

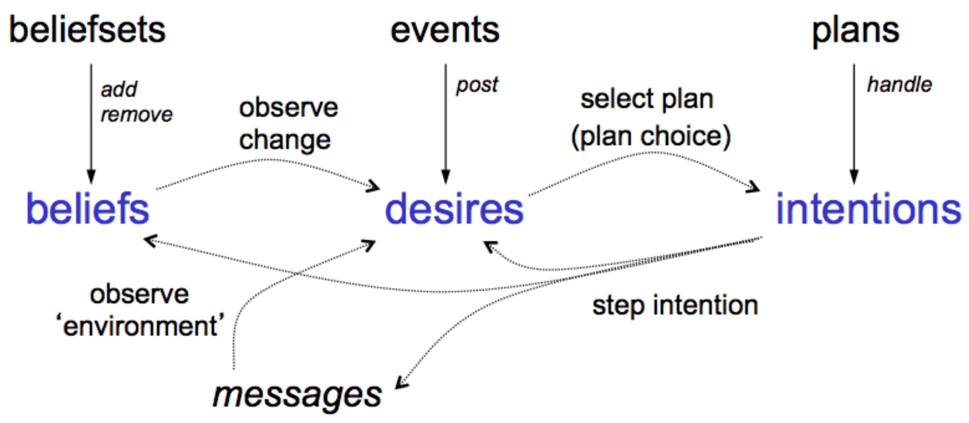


Figure 3: The JACK BDI Execution

2.0.2 Why are agents useful?

An important advantage of agents is that they reduce coupling. The coupling is reduced by encapsulation provided by autonomy, the robustness, reactiveness and pro-activeness of agents [8]. Because of its properties an agent can be relied upon to persist in achieving a given goal by trying alternative approaches depending on environment changes. Being proactive and reactive agents are human-like in the way they deal with problems. This provides a very natural abstraction and decomposition of complex problems. Leading to agents being used in a number of applications such as planning and scheduling, business process systems, exploration of space, military operations/simulation and online social communities.

3 Human Robot Interfacing (HRI)

The presence of robotic technologies and the research being conducted is growing in many fields such as space exploration, military weapons and operations, search and rescue, health care etc. All the different application areas introduce HRI challenges which are unique to its particular field of operation, but several principles and HRI issues are common for all systems where robots are involved. This chapter will present some of the most important issues in robotic operator performance and present some of the well known user interface solutions, both in design and technologies. The content of this chapter is important in order to address the research hypotheses described in section 1.2 and to draw relevant conclusions based on the developed application.

3.1 HRI Metrics

To be able to evaluate task-oriented HRI a set of metrics have been proposed by Fong [10]. These metrics are designed to assess the level of effort required both from the human and the robot in order to accomplish joint tasks. To define a set of task-specific metrics that are applicable to the operation of mobile robots, 5 main general tasks are identified as:

- Navigation from point A to point B
- Perception of remote environment
- Management of robot and human tasks
- Manipulation of remote environment by robot
- Tasks involving social interaction

3.2 Principles

To minimize error and workload within HRI, Goodrich and Olson did a study where they developed a set of principles for designing robot technologies [11]. The basis for these principles are:

1. Neglection time: The amount of time a robot can function efficiently without human interaction
2. Interaction time: the time it takes before a robot's performance is back to maximum after human interaction begins
3. Robot attention demand: How much time is required to operate a robot based on neglection time and interaction time

4. Free time: Time left for secondary tasks during HRI based on neglection time and interaction time
5. Fan out: Number of HRIs that can be performed simultaneously on robots of the same type

These five concepts are the foundation for the seven principles of efficient interface design and these principles are:

1. Switching between different interaction and autonomy modes should require as little time and effort as possible. Knowledge of how to act in each mode should be sufficient for switching modes.
2. If possible, cues provided to the robot should always be natural, for example map-based sketching. The use of naturalistic cues has proven to be an effective mean for conveying intent to robots.
3. This principle addresses the advantages of an operator being able to have as much direct contact with the target environment as possible in order to reduce interfacing with the robot. Providing an as direct link as possible between the operator and target environment will reduce the operator workload as the operator does not need a model of the robot, only the environment, to successfully initiate commands for the robot.
4. Because a direct link as described in principle 3 is not always possible, this principle states that if a direct link is not possible, it is still best to design the interface so that the operator focus remains on the target environment and not on the robot.
5. States that for an interface to be effective, information provided to an operator should be possible to manipulate if needed. For example feedback about the heading of a robot should allow for manipulation of that heading.
6. Is designed to increase the operator's ability to multitask by reducing the cognitive workload. This is achieved by externalizing information which is not immediately relevant but might be necessary later.
7. Finally the last principle aims to ensure that the interface directs the operator's attention towards critical information when needed.

These principles for effective robot interface design have been widely adopted and represent a general way of summarizing information about HRI design concepts.

3.3 Operators workload in autonomous systems

The human role in HRI has been described in many different ways, but in general an operator's workload varies with the level of teleoperation and manual intervention needed if the robot's autonomous actions encounter problems. Simply managing autonomous robots reduces the workload, however this reduction depends greatly on the reliability and robustness of the autonomous system. With 60 - 70 percent system reliability one may fail to achieve any improvement performance wise [12]. In addition to the reliability issues, another important factor associated with workload is the concept of context acquisition. This is when an operator has to switch between tasks, for example from navigation to data analysis based on different sets of sensor input. The interface design itself is important, but this is also an area where software agents can provide a good solution. Software agents run analysis and reasoning on the data/sensor inputs and only involve the operator when needed and in general present results and options in a goal oriented sense, thus reducing the operators workload.

3.4 The paradox of automation

The paradox of automation states that the more efficient the automated system, the more crucial the human contribution of the operator. Humans are less involved, but their involvement becomes more critical and that efficient automation makes humans more important, not less [13].

Due to technological advances and much research, an increasing amount of our vehicles and robots are automated and controlled by software. There has been a lot of effort put into researching the effects of introducing autonomy in domains such as aviation and industrial settings such as nuclear plants, but the effects in robotics are not equally researched. The effect on human performance caused by automation is dependent on the level of automation applied in the system. The level of automation can range from no automated assistance where the operator makes all decisions and takes all actions to fully atonomous systems where human input is essentially disregarded. The main human performance issues which arise with system automation are: mental workload, situation awareness (SA), complacency , and skill degradation. There are several examples where automation decreases the mental workload of an operator, but this is not always the case and many studies show the opposite [14], an increase in mental workload. The SA issue also has positive and negative implications, with automation more information can be provided in a timely manner, but it can also lead to the operator not knowing when changes to the system status occur and thus preventing the human from developing an overall picture of a situation based on processed information received from the computer. This continuous information processing without human intervention can result in complacency on behalf of the human. This becomes a factor when the system malfunctions, and the operator fails to monitor the automated process closely enough and the failure goes undetected. Finally the issue of skill degradation which is the fact that memory and skill decreases over time if not practiced. This also comes into play if a normally automated process fails and a human must perform the task temporarily.

The paradox of automation issues force autonomous systems to be designed to have as little impact as possible on human performance compared to traditional implementations if they are to be favorable.

3.4.1 Interface design for autonomous robots

Many different interfaces for controlling autonomous agents have been developed and tested, each of them with benefits and challenges specific to it. Several studies have been done on the use of various interfaces for controlling autonomous robots, but these are rather specific when it comes to which robot functions they control and the operational environment of the robot. Experts from the Robotics Isntitute at Carnegie Mellon University (CMU) have observed challenges for controlling fully and semi autonomous mobile robots and in a set of interviews come with recommendations and lessons learned [15]. Here is a partial list of the lessons learned:

- With multiple operators, the operator with a direct line of sight of the robot should be given veto.
- Although video and map views are useful it is not required that both are visible at the same time.
- Showing key information with a dashboard layout on the bottom of the screen is useful.
- 3-D interfaces for controlling and navigating is difficult.
- Color changes or pop ups of state information when thresholds are crossed are useful.
- Central error and health summary should be available.
- Integration and color coding information is useful.
- Delay in communication is a factor that must be considered.
- The design should account for potential substandard operator environments and conditions.

Some examples of newer techniques/devices for controlling autonomous robots are cellular phones, PDA, sketch interfaces, natural language and gestures, and haptic/vibrotactile. These different user interface designs have pros and cons presented in table 2:

Display	**Advantages**	**Disadvantages/Limitations**
Cellular phone and PDA	Enhanced portability	Devices of this type have limited screen size and controlling more than one robot per device may be difficult. Software and computing capabilities are also limited due to the size. A touch-based interface needs to provide icons and screen items of adequate size.
Sketch Interface	Uses land-marks for navigation providing a natural and intuitive way to interface with the system. In addition the task representation is based on relative position instead of absolute robot position.	The stylus markings of the user needs to be consistently and correctly interpreted by the system.
Natural Language and Gestures	Reduces learning curve for successful HRI tactics	With gesture-based interfaces lighting conditions and FOV may cause problems for the cameras.
Haptic/Vibrotactile	Better collision avoidance and can be used in environments and settings with bad visibility conditions. The operator's ability to receive, process, and act on sensor input is enhanced.	Limited bandwidth and difficulties duplicating the complexities of vision through tactile interfaces.

Table 2: Advantages and Disadvantages of different techniques/devices [1]

3.4.2 Human agent/robot Teaming

The concept of human-robot teaming is based on the interdependence between the human operator and the robot/agent in carrying out a robot-assisted mission. The term human-robot ratio is important in the design of such teams, this refers to the number of robots that effectively can be controlled by one operator. The team composition is one of the important parts of the system design playing an important role in maximizing performance HRI wise. There are several different options for human-robot team configuration and for each one challenges arise related to the performance of the team. Examples of configurations could be one human-one robot, multiple humans-one robot, and most relevant, and used for this thesis, one human-robot team where one operator sends commands to multiple agents/robots which, in turn, must sort and classify the operator's commands. Several studies show that a common operational picture, shared mental models, and efficient communication flow are the most important factors for human robot-teams.

4 Methodology and tools

This chapter describes the different methodologies and tools used for modeling and development of the agent-solution and robot application.

4.1 Prometheus methodology

Prometheus is intended to be a practical methodology. As such, it aims to be complete: providing everything that is needed to specify and design agent systems. The methodology is widely used in university courses, by industry workshops and the company behind JACK, Agent-Oriented Software [16].

4.1.1 Why a new agent methodology?

Although there are many methodologies for designing software, none of these are well suited for developing agent oriented software systems. Even though there are similarities between agents and objects there are some significant differences justifying the use of the Prometheus methodology over object oriented methodologies. This despite the fact that object oriented methodologies are extensively studied and developed compared to Prometheus.

Some of the main differences between Prometheus and object oriented methodologies are:

1. Prometheus supports the development of intelligent agents which use goals, beliefs, plans, and events. By contrast, many other methodologies treat agents as simple software processes that interact with each other to meet an overall system goal.
2. Prometheus provides explicit modeling of goals which is needed to support proactive agent development. This is generally not a part of object-oriented methodologies.
3. To provide flexibility and robustness a message (or an event) should be allowed to be handled by several plans, not just as a label on arcs which is common for object oriented methodologies.
4. Agents are situated in an environment, thus it is important to define the interface between the agent and its environment.
5. In object oriented programming everything is a passive object, but in agent oriented programming it is needed to distinguish between passive components such as data and beliefs, and active components like agents and plans.

4.1.2 The three phases

The Prometheus methodology consists of three phases, as shown at figure 4.

1. The system specification phase intends to describes the overall goals and basic functionality, including the illustration of the systems operations with use case scenario schemes. The phase is also intended to specify inputs (for example sensor readings) and outputs (actions), namely the interface between the system and its environment.
2. The second phase, called the architectural design phase decides which agent types the system will contain and how they interact based on the previous phase.
3. The detailed design phase looks at each agent individually and describes its internal behavior to fulfill its goals within the overall system.

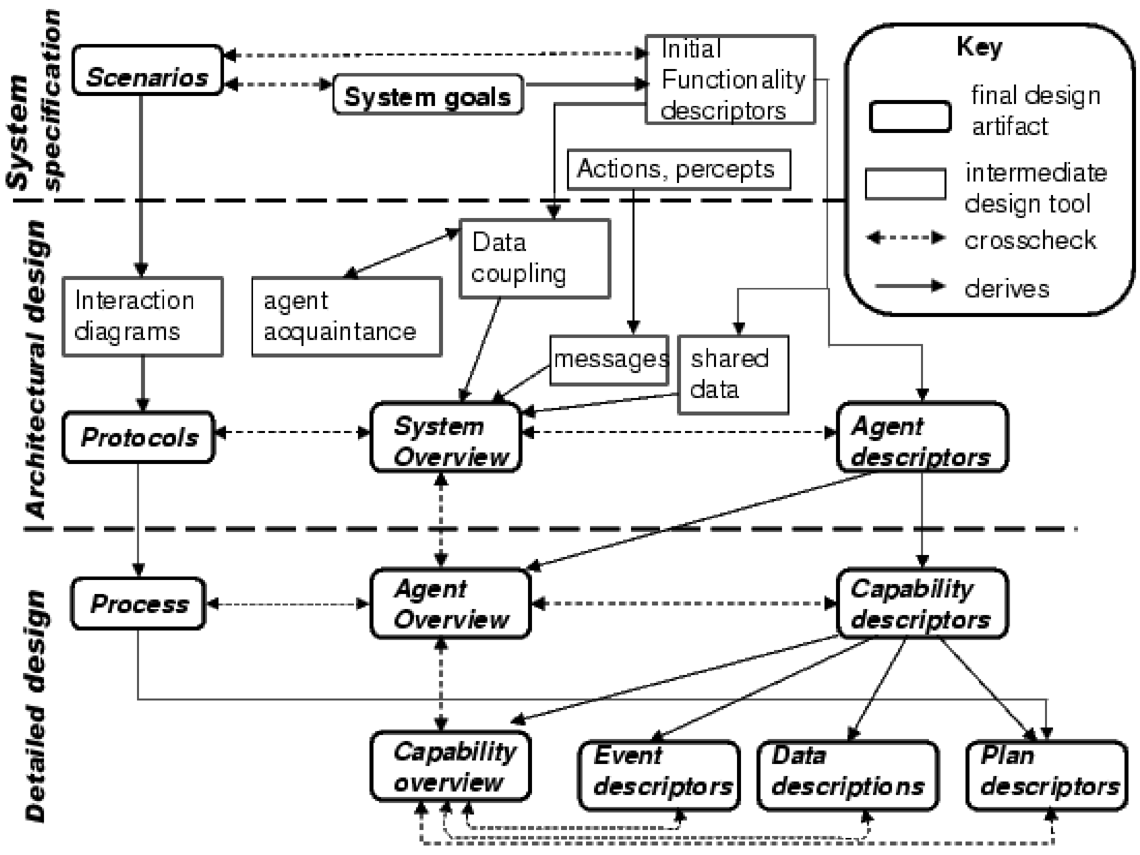


Figure 4: The phases of the Prometheus methodology

System specification As mentioned in the start of Chapter 3.2, the system specification phase focuses on the following:

- **Identifying the system goals.**
  The system goals might be thought of as the overall goals of the system, what the system should be able to achieve. In agent software these goals are important because they control the agents behavior. The system goals are often high-level descriptions; therefor they tend to be less likely to change over time than functionalities.

- **Creating use case scenarios that presents how the system works.**
  Use case scenarios are used to describe how the system operates through a sequence of steps combined with description of the context in which the sequence occurs. These scenarios are useful to understand the structure of the system works.

- **Identify the fundamental features of the system.**
  The fundamental features are groups of related goals, data and input/output that describe the main functionalities of the system. In a ATM system these might be; "Withdraw money", "Check account balance" and "Change card PIN code". As the system is created, the need for new functionality will be introduced. In our ATM system there might be a need to add "Charge cell phone account".

- **Describe the interface connecting the system to its environment, inputs and outputs.**
  An agent is situated in an environment, and we need to specify how the agent affects the environment and what information the agent gets from the environment. Using our ATM system example, the agent gets input in form of credit card data, withdrawal requests and so on. The output might be money or a message on the ATM display.

Architectural design The architectural design phase uses the outputs from the previous phase to determinate which agent types the system will contain and how they will interact. It also captures the system's overall structure using the system overview diagram.

- **Agent Types**
  One of the most important aspects of the Architectural design is to determine which agents are to be implemented and to develop the agent descriptors. The functionalities established in the first phase are grouped into agent types, so that each agent consists of one or more functionalities. The functionalities are grouped together based on coupling and cohesion.

- **System structure**
  Once the agent types are decided upon, the system structure is determined by dividing input and output responsibility among the agents. The major shared data repositories are also specified in this process. These items are modeled in the

system overview diagram, which is perhaps the single most important product of the design process. It ties together agents, data, external input and output, and shows the communication between agents.

- **Interactions**
  The System structure defines who talks to who, while the interactions part defines the timing of communication. This is done trough use case scenarios and is modeled in agent interaction diagrams.

Detailed design The last of the three phases focus on the individual agent's internal design, constructing its capabilities, including plans, events and data so that it can fulfill its responsibilities as outlined in the functionalities it is to provide. It is also important to refine the interaction protocols the agents use for internal and external communication.

4.2 JACK Intelligent Agents

AOS [5] offers a number of products for developing autonomous systems: JACK, JACK-Teams, JACK Sim, C-BDI, CoJACK and Surveilance agent. JACK is the worlds leading autonomous systems developing platform. It is entirely written in Java making it able to run on any system of which Java is available from laptops to high-end multi-CPU enterprise servers. JACK thus has access to all Java features including multiple threads, platform independent GUIs and third party libraries. JACK also provides a JDE (JACK Development Environment) for developing and designing JACK applications.

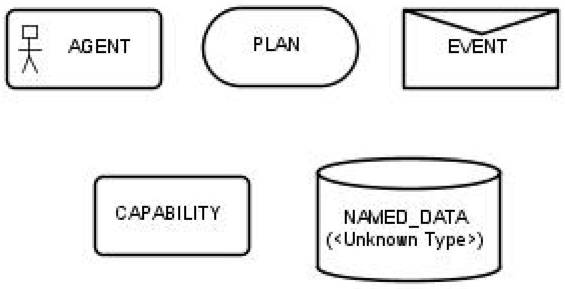


Figure 5: The JACK Components / Agent Model Elements

4.2.1 JDE

Components and links (See Figure 5) can be added/removed in the JDE browser window or graphically using the design tool. These express relationships between agent model, elements and skeleton code is automatically generated for them. The JDE saves in a .prj file and a gcode directory and when you select compile application the corresponding JACK files are generated before the compilation proceeds as it would on the command line.

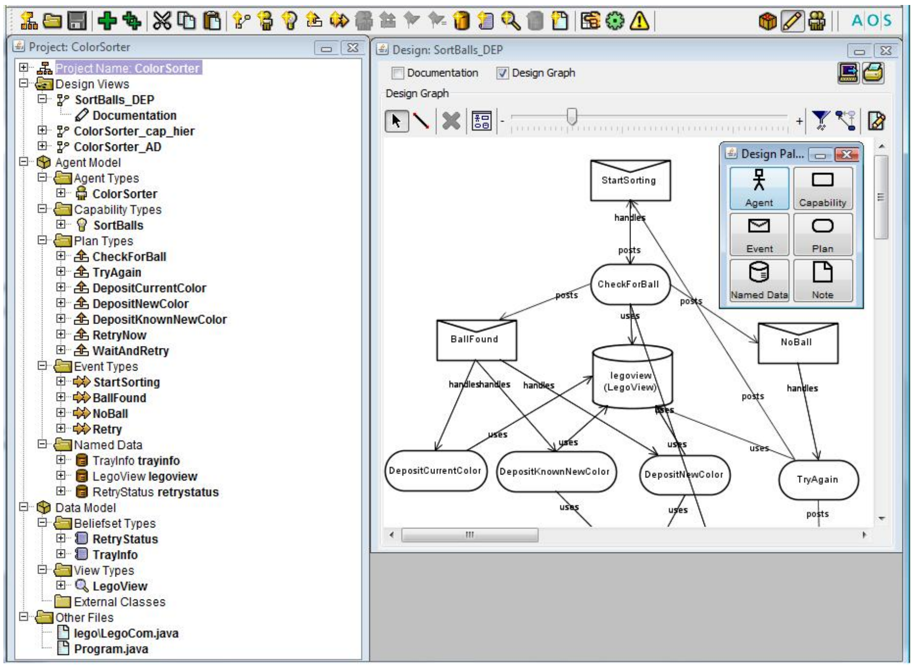


Figure 6: The Jack JDE

4.2.2 DCI

JACK DCI (Distributed Communication Infrastructure) enables agents to communicate within a process, across processes and between different machines. A DCI portal for a process is defined by giving the process a portal name and a port number to identify it. The full name for an agent is agent_name@portal and the DCI will ensure message delivery across portals.

4.2.3 JACOB

Provides machine and language independent object structures that can be stored or transmitted. The object structures are defined using the JACOB Data Definition Language and stored in definition files, which are compiled using JACOB Build.

4.2.4 JACK agent language

The JACK agent language is an extension of Java to support an agent oriented programming paradigm

It introduces new base classes: agent, capability, event, plan, view, beliefset, and extensions to the java syntax to support these e.g. #declarations and @reasoning statements.

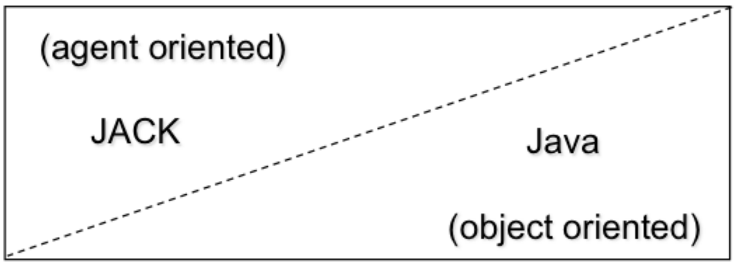


Figure 7: Agent oriented vs Object oriented

It uses the BDI (Belief Desire Intention) agent model

4.2.5 Agent:

The agent type encapsulates knowledge and behavior through beliefsets, events and plans which can be represented as capabilities. It reacts to events and receives messages to perform tasks and services.

4.2.6 Event

All activity in JACK originates from an Event. The event provides the type safe connection between agents and plans as both the agents and plans must declare the events they handle, post and send. JACK supports several different types of events depending on desired plan processing behavior. The different types are:

- Normal:

  A 'Normal' event corresponds to conventional event driven programming. Causes the plan behavior to be that if the plan fails the agent does not try again. There are two base classes for normal events, these are Event, which is the base class for all events and can only be posted internally and MessageEvent which can be sent between agents (a message for the sender, an event for the receiver).

- BDI:

  A BDI event represents the desire to achieve a goal and it may cause both meta-level and practical reasoning. This can result in agents trying several different plans and even recalculating the applicable plan set. There are three different base classes for BDI events, BDIGoalEvent, BDIMessageEvent, and BDIFactEvent. The BDIGoalEvent is typically used in @achieve, @insist, @determine etc and will cause an agent to try all applicable plans until one succeeds. The receiver of a BDIMessageEvent uses BDI processing and so does a receiver of a BDIFactEvent but in a non persistent way. The BDI events can be customized to specify how and when to determine the applicable plan set and how to form it, when to do meta-level reasoning, how to choose plan without meta-level reasoning , how to deal with plan failure and how to handle exceptions.

- Rule:

  The event base class for rule events is InferenceGoalEvent. This type of event will cause all plans in the applicable set to be executed regardless of success or failure.

- Meta: The event base class for Meta events is PlanChoiceEvent. This is the mechanism the agent uses to perform meta-level reasoning.

4.2.7 Plan

A plan describes the actions an agent can take when an event occurs. Each plan can only handle a single event and it will either succeed or fail. A plan contains logic to determine if the plan is relevant or not for a given event. It also has at least one reasoning method, which defines the actions of the plan. This method can contain JACK agent language @statements and each of these are handled as a logical condition. These are handled sequentially and if a statement fails the method fails and terminates, only if all statements succeed the plan succeeds.

4.2.8 Capability

Capabilities are used to wrap events, plans and data into reusable components. An agent can 'have' a capability that again can be composed of other capabilities (capability nesting).

4.2.9 Beliefset

JACK Beliefsets are a form of representing an agents belief. A Beliefset is a relational representation where the individual belief representations are propositional. It's like a relational database, but not used for long-term storage or shared between agents. The reason for not sharing Beliefsets amongst agents is to avoid concurrent data updates. A Beliefset may be shared, but there are concurrency issues due to multi-threading and it's therefore normally not done. Technically a Beliefset is a relation which is a set of tuples where each tuple is a belief/fact that can be either true or false. The tuples must have one or more fields, with an unique key field and value field(s). Beliefs can be queried on and changed/added/removed as the agent changes it's beliefs in run time. The change of an agents belief may result in change of behavior and this is invoked by callback methods posting Events that in turn are handled by relevant plans. Beliefsets must be declared in the agents, capabilities and plans that use them.

4.2.10 View

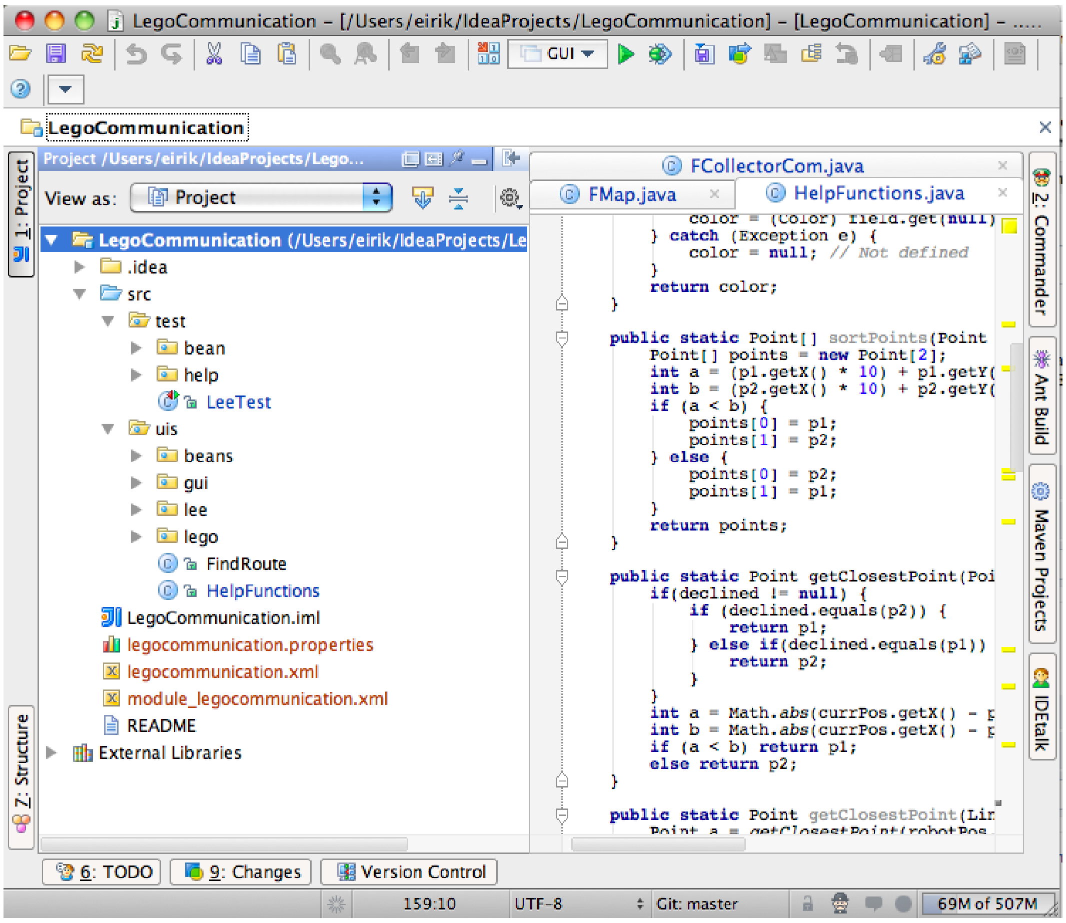
A JACK view is a way to interface between JACK and other systems. Using views it is possible to integrate a range of data sources into the JACK framework like Beliefsets, java data structures and legacy systems. Views must be declared in the agents, capabilities and plans that use them.

4.3 Java

Java [17] is a programming language originally developed by Sun Microsystems. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte-code that can run on any Java virtual machine (JVM) regardless of computer architecture.

4.4 IntelliJ IDEA

IntelliJ IDEA is a commercial Java IDE by JetBrains [18]. It is often simply referred to as "IDEA" or "IntelliJ." IntelliJ IDEA offers smart, type-aware code completion. It knows when you may want to cast to a type and is also aware of the run-time type checks that you made, after which you can perform cast and method invocation in a single action.



```
            color = (Color) field.get(null)
        } catch (Exception e) {
            color = null; // Not defined
        }
        return color;
    }

    public static Point[] sortPoints(Point
        Point[] points = new Point[2];
        int a = (p1.getX() * 10) + p1.getY(
        int b = (p2.getX() * 10) + p2.getY(
        if (a < b) {
            points[0] = p1;
            points[1] = p2;
        } else {
            points[0] = p2;
            points[1] = p1;
        }
        return points;
    }

    public static Point getClosestPoint(Poi
        if(declined != null) {
            if (declined.equals(p2)) {
                return p1;
            } else if(declined.equals(p1))
                return p2;
            }
        }
        int a = Math.abs(currPos.getX() - p
        int b = Math.abs(currPos.getX() - p
        if (a < b) return p1;
        else return p2;
    }

    public static Point getClosestPoint(Lin
        Point a = getClosestPoint(robotPos
```

Figure 8: The IntelliJ IDEA graphical user interface

4.5 LeJOS, Java for Lego Mindstorms

To allow us to program our LEGO robots using Java we used LeJOS NXJ which is a Java programming environment for the Lego Mindstorms NXT. The leJOS NXJ is a complete firmware replacement for the standard Lego Mindstorms firmware that includes a Java Virtual Machine. LeJOS is an open source project and was originally created from the tinyVM project that implemented a Java VM for the older Mindstorms system RCX. The current newest version and the one we used is lejos-NXJ 0.8.5 beta and it is supported by three operating systems: Microsoft Windows, Linux and MAC OS X. It consists of [19]:

- Replacement firmware for the NXT that includes a Java Virtual Machine.
- A library of Java classes (classes.jar) that implement the leJOS NXJ Application Programming Interface (API).
- A linker for linking user Java classes with classes.jar to form a binary file that can be uploaded and run on the NXT.
- PC tools for flashing the firmware, uploading programs, debugging, and many other functions.
- A PC API for writing PC programs that communicate with leJOS NXJ programs using Java streams over Bluetooth or USB, or using the LEGO Communications Protocol (LCP).
- Many sample programs

4.6 LEGO Mindstorms

LEGO Mindstorms is a programmable robotic kit created by LEGO. The LEGO Mindstorms NXT 2.0, which is the newest version, comes with a NXT Intelligent Brick, two touch sensors, a color sensor and an ultrasonic sensor. It also includes three servomotors as well as about 600 LEGO Technic parts.

The NXT Intelligent Brick is the main component of the robot. It can take input from up to four sensors and control up to three motors simultaneously. The brick also has a LCD display, four buttons and a speaker.

Originally the brick comes with software based on National Instruments LabVIEW [20], and can be programmed trough a visual programming language. LEGO has however released the firmware for the brick as open source [6], and several developer kits are available. Due to this, third party firmware has been developed to support different programming language, such as Java, C++, python, Perl, Visual Basic and more.

Figure 9: The NXT 2.0 Intelligent Brick

5 Application

This chapter will present the chosen application scenario and describe the process of defining it before ending up with the final approach. The goal was to have a case where the common implementation goal would be met as well as both the individual parts of the thesis. The HRI part demands a cooperation between system/robots and operator while the agent interaction part implies multiple agents involved. These where both important aspects to take into account when defining the scenario.

5.1 Scenario

Based on the implementation goal specified together with our supervisors we defined a scenario which includes all the desired aspects described in Section 1.2. The scenario is: 3 robots with different properties, which in cooperation are to explore a restricted, unstructured and dynamic operational environment where different types of objects are located randomly. These objects are to be collected and sorted by color. The robots are to coordinate amongst themselves cooperating in achieving a common goal. Each robot is assigned a specific task depending on its abilities, one explores and locates the objects, one collects and deposits the objects found, while the last robot sorts the delivered objects based on object color.

5.2 First approach

We first started out wanting to have the robots operate within a map only specified by a set of boundaries. The robots where to do the navigation an positioning using a sonar sensor measuring distances to the boundary walls and possible obstacles, for example other robots. There are several localization algorithms/techniques used in robotics, but one has proven to be both computationally efficient and accurate making it the most widely used, this is the Monte Carlo Localization (MCL) algorithm [21] [22].

5.2.1 Monte Carlo Localization

The basic idea of this approach is to estimate the robots position using sensor readings. Initially only the map boundaries are known and not the robots position. MCL generates a set of poses distributed randomly within the boundaries all having a weight representing the probability of the pose representing the actual robot position and a heading. Each time the robot moves MCL generate N new samples that approximate the robots position after the move. These samples are generated by randomly drawing a sample from the previous computed sample set with likelihood determined by their previous weight combined with the new sensor reading. This resampling is done each time the robot moves and will eventually determine the robots most likely position with high accuracy.

5.2.2 First approach development

After deciding on this approach we ”built” a simple map and a robot with a sonic sensor shown in Figure 14 and Figure 15. The MCL algorithm was implemented in java with a graphical user interface showing the robots current pose set within the boundaries shown in Figure 10 11 12 13. These figures show a typical scenario where the robot moves several times before its most likely position is determined accurately.

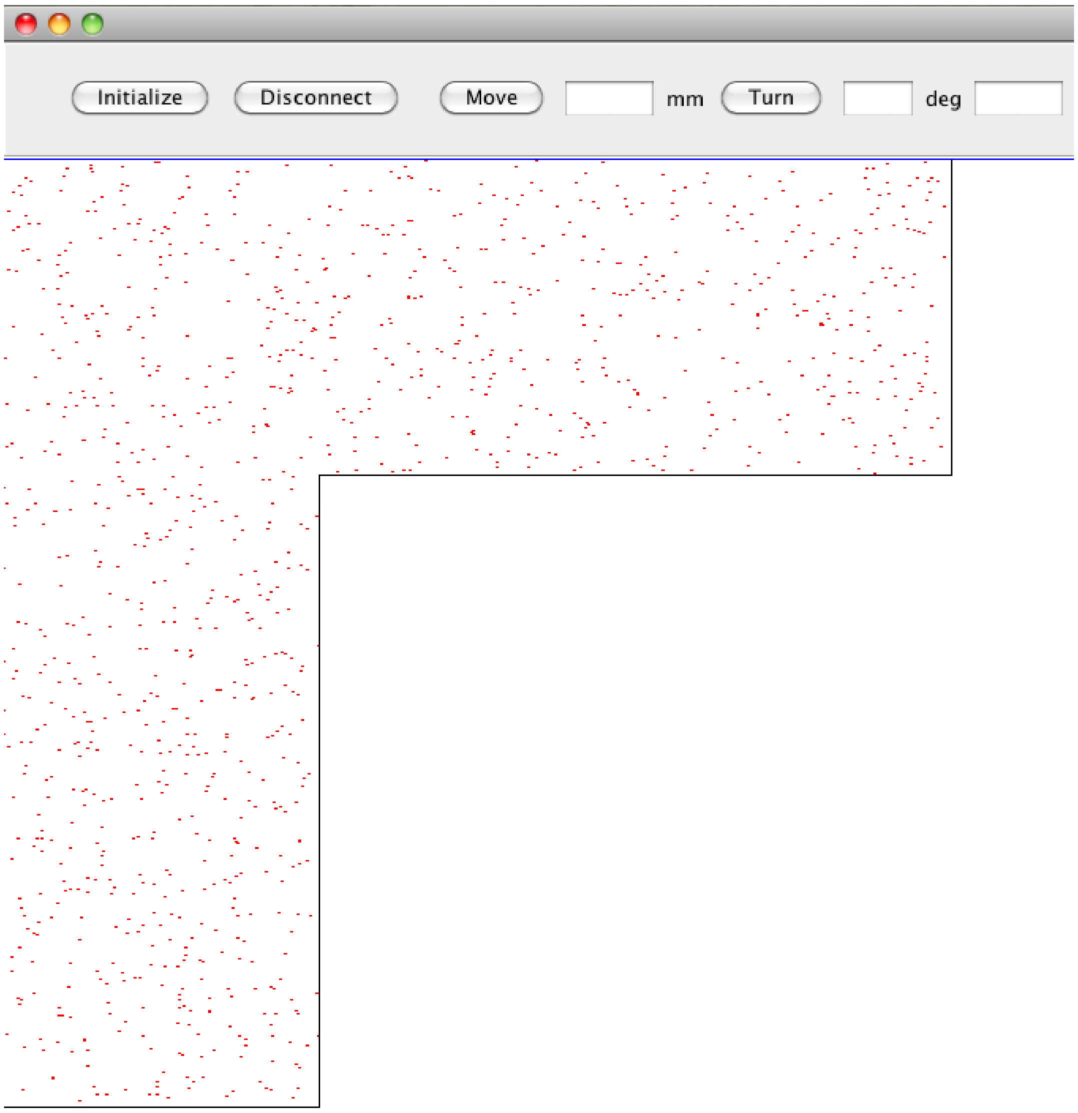


Figure 10: Monte Carlo Localization App initial pose.

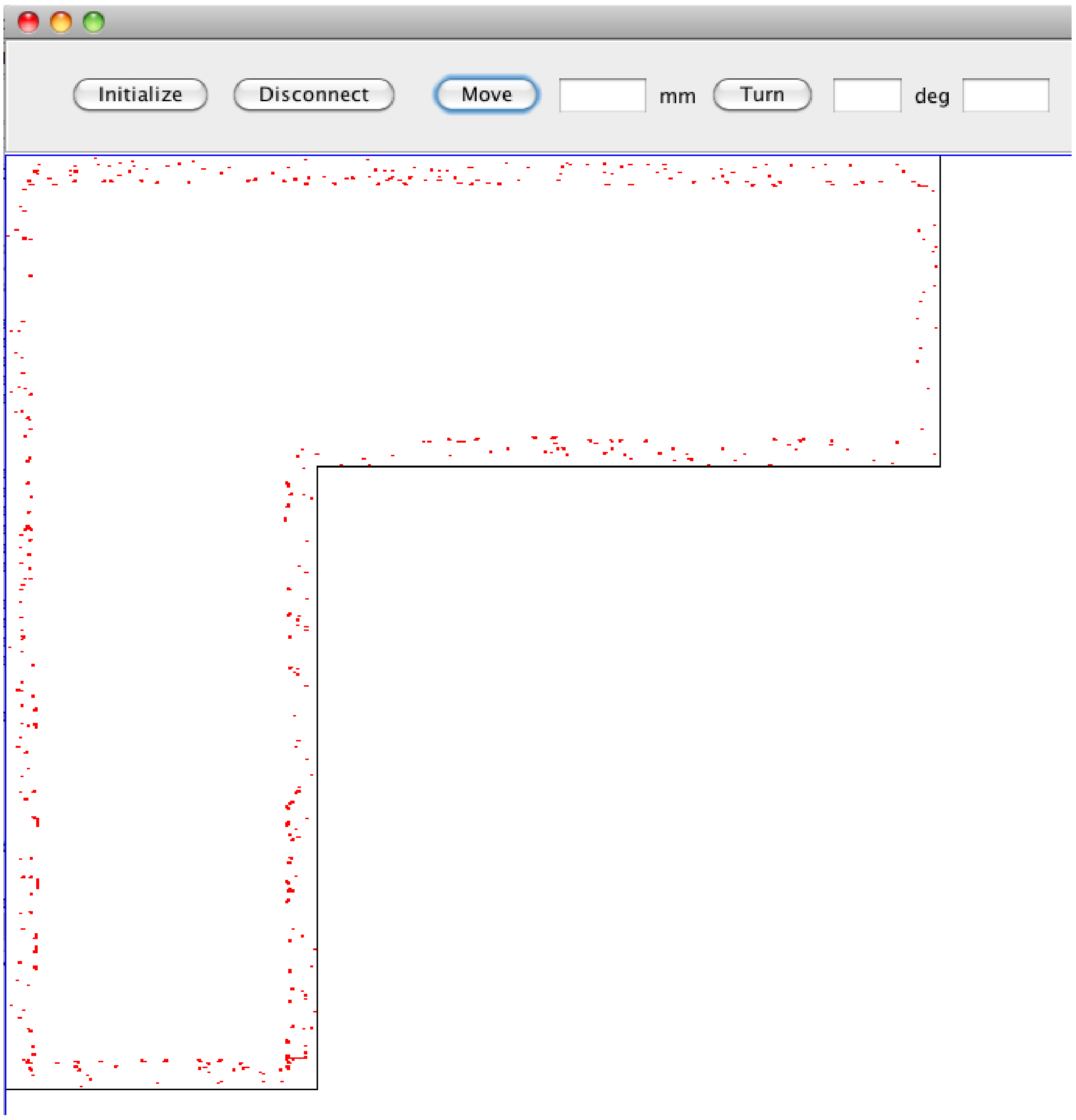


Figure 11: Monte Carlo Localization resampled pose set after first move

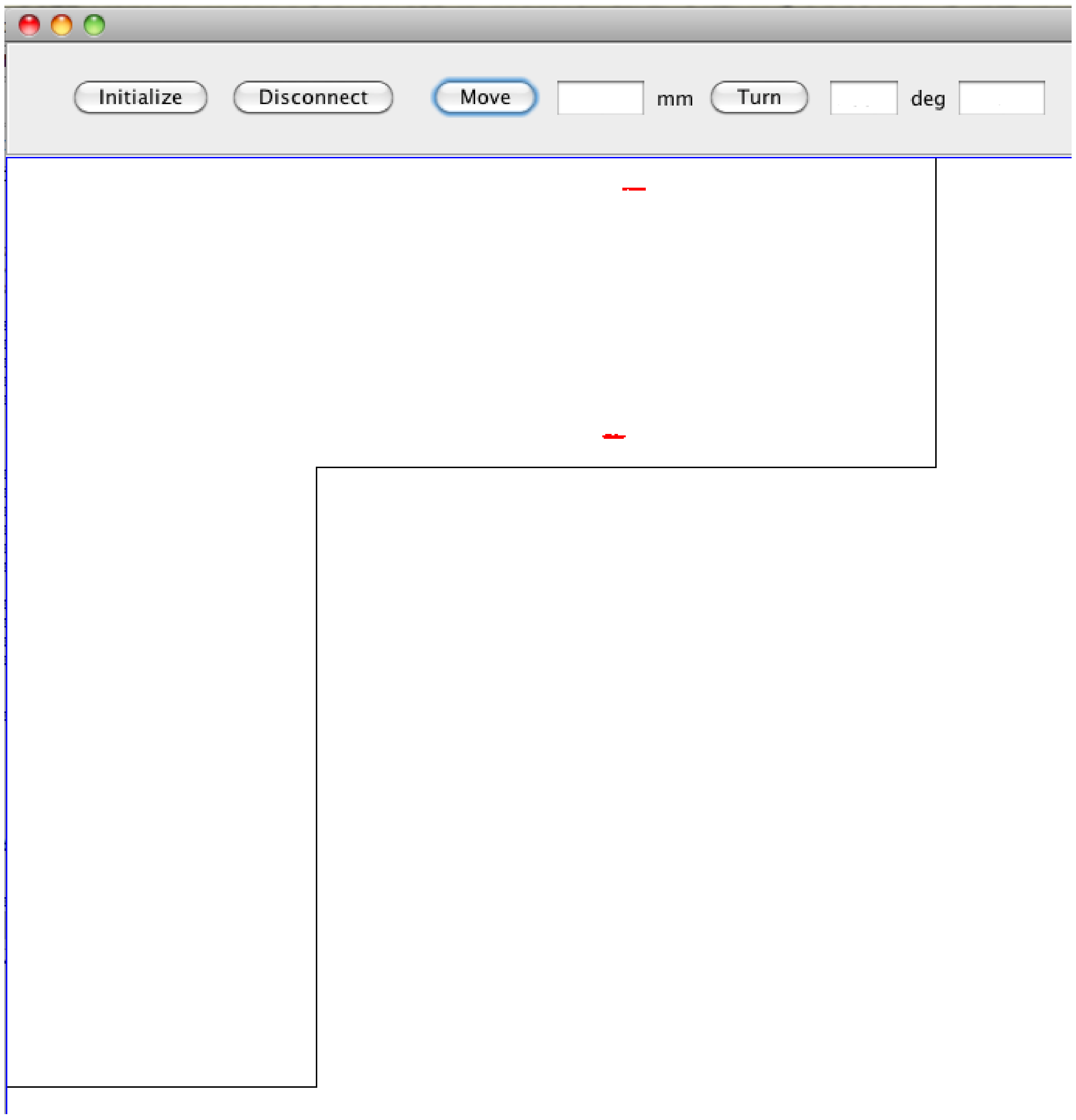


Figure 12: Monte Carlo Localization resampled pose set after several moves

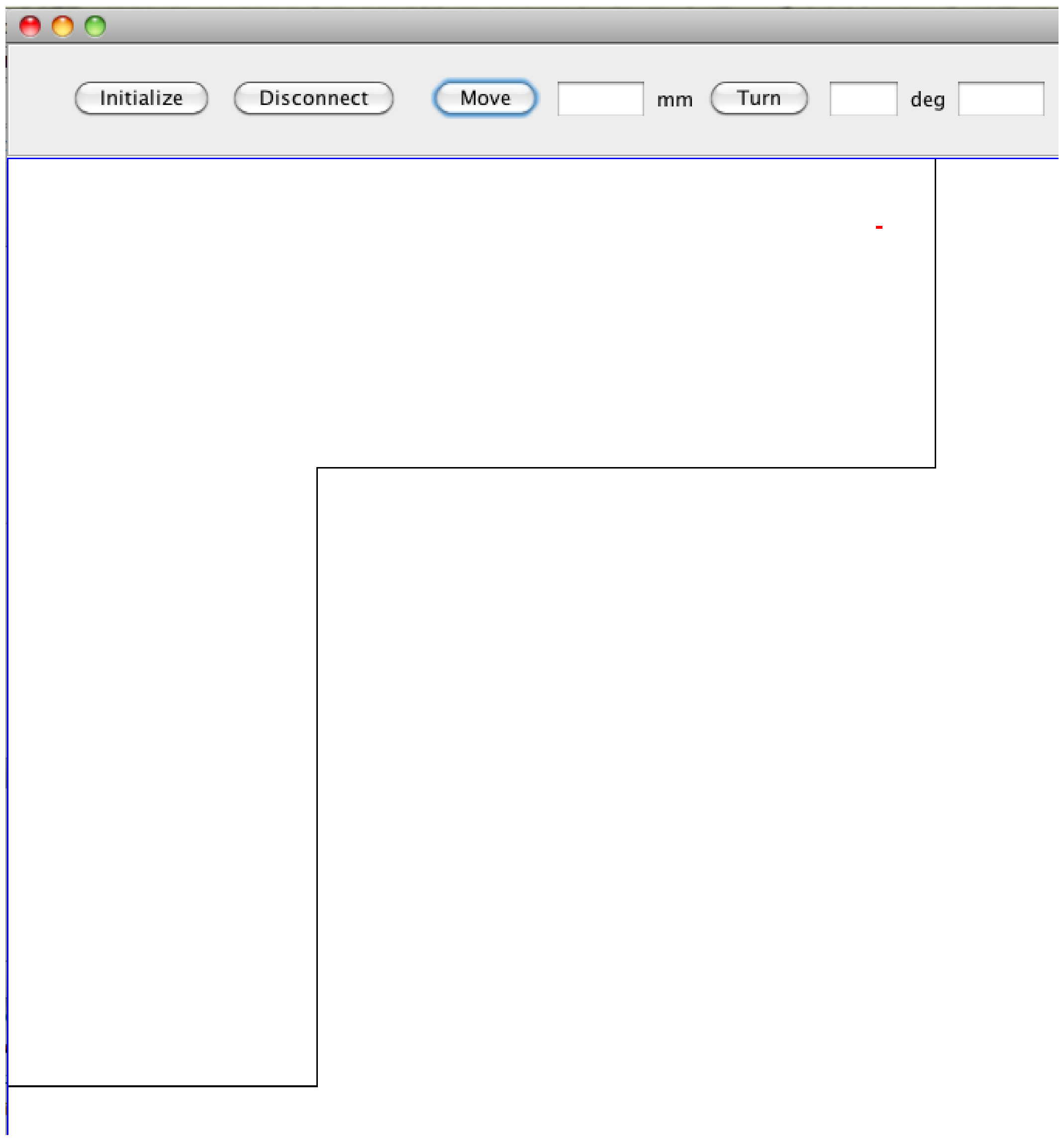


Figure 13: Monte Carlo Localization resampled pose set after location found.

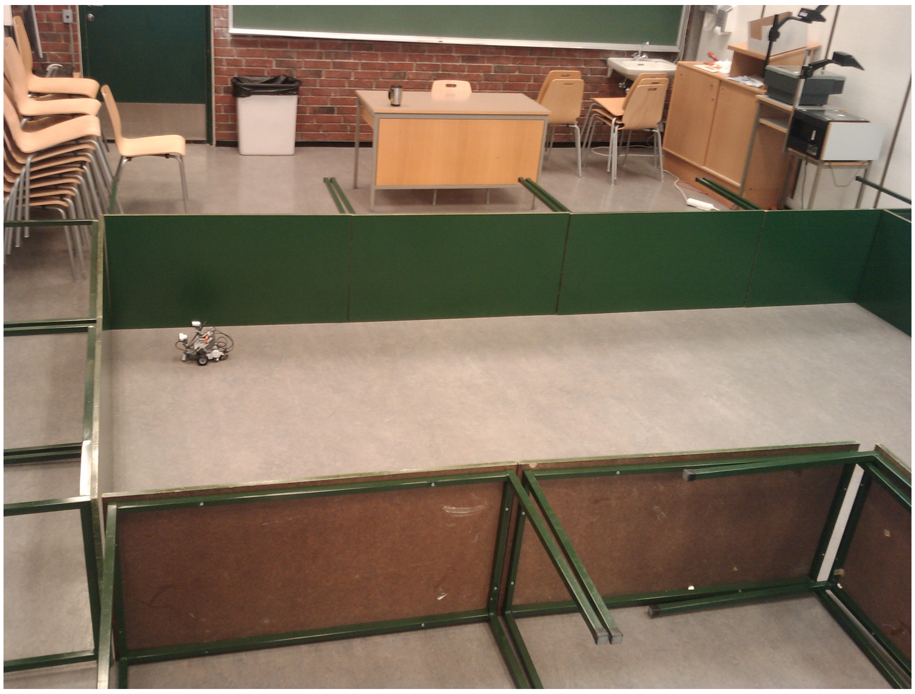


Figure 14: Robot located in MCL map

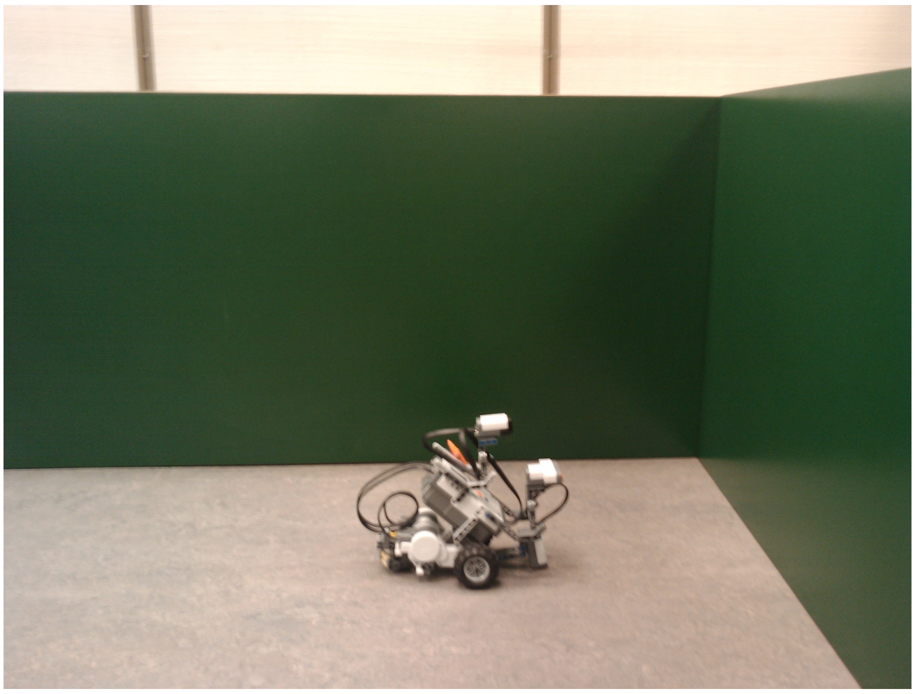


Figure 15: Robot located in MCL map, close up

5.2.3 First approach results

The MCL implementation was satisfactory in terms of accuracy and computational efficiency. Despite this the cons presented during testing heavily outweighed the pros of this approach. The LEGO Mindstorms sonic sensor was unreliable. Uncertainty in exact degrees turned and distance moved where both challenges, and the level of complexity in dealing with these issues increased drastically when more than one robot was introduced into the system. Due to time limitation and the main focus of the thesis being the software agent/HRI challenges we were forced to drop this approach after 1 month of development.

5.3 Final approach

After considering time limitations and the main focus of thesis, the final approach was specified. This approach is based on the robots operating on a line-based map/grid. This approach is preferable as Mindstorms robots have fairly good support for this kind of navigation (line following). There has been done quite a lot of projects on this leaving us to focus on more relevant challenges for the thesis, being the agent implementations and human-agent interfacing. The basic idea of robot setup and common goal remains the same as described in initial approach. A sketch of the overall grid design is presented in Figure 16.

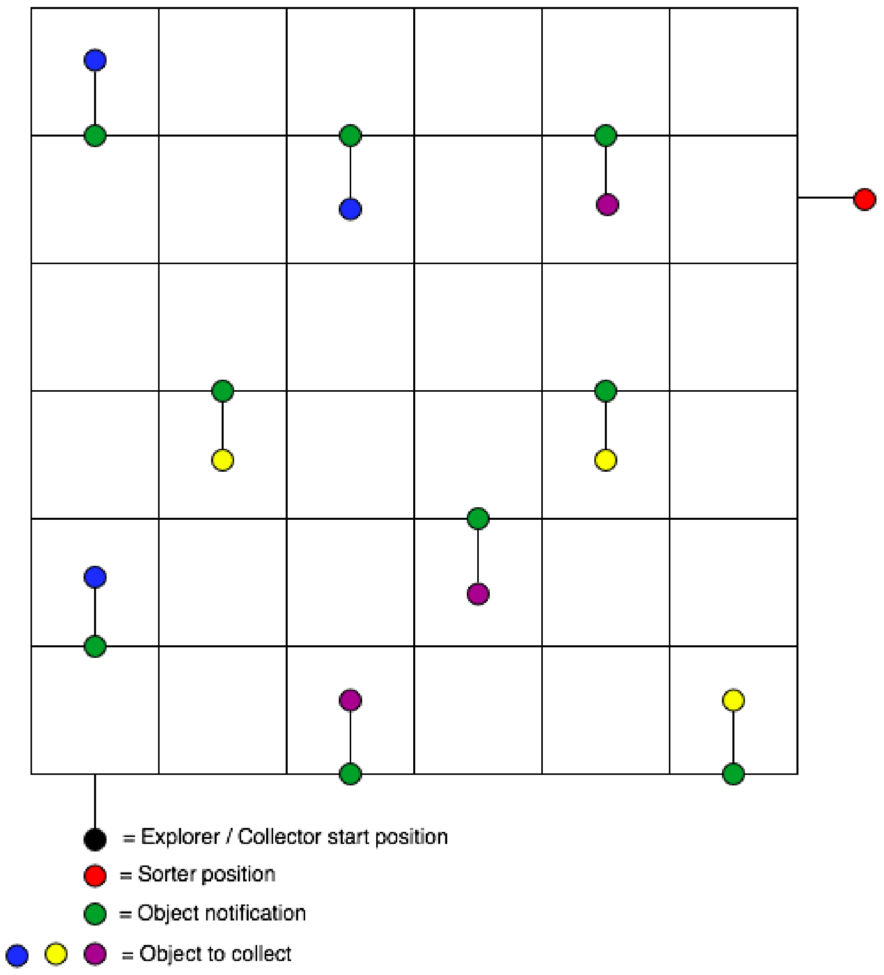


Figure 16: Grid-based map sketch

6 System Design

Our design is developed using the prometheus methodology described in Section 4.1. This chapter will present the main phases of the design process and our design choices in light of the thesis hypotheses.

6.1 System specification

The system goals are derived from the scenario described in previous chapter 5.3. To realize the system a set of main goals and sub goals where defined:

- Explore map
  −Find all drivable lines on the grid.
  −Find all objects located on the grid.
- Collect items
  −Pick up located items.
  −Deliver picked up items to be sorted.
- Sort all items located on the grid.
  −Sort items into trays based on color.
- Collision avoidance
  −Robots yield according to specified priority list. −Determine alternative routes on deadlock.
- GUI design based on best practice approach for successful HRI.
  −Intuitive GUI.
  −Keep operator focus on crucial information.
  −Present results/data in user friendly manor.
  −Ease the load of data analysis for operator.

The required functionalities are defined based on these goals illustrated in Figure 17.

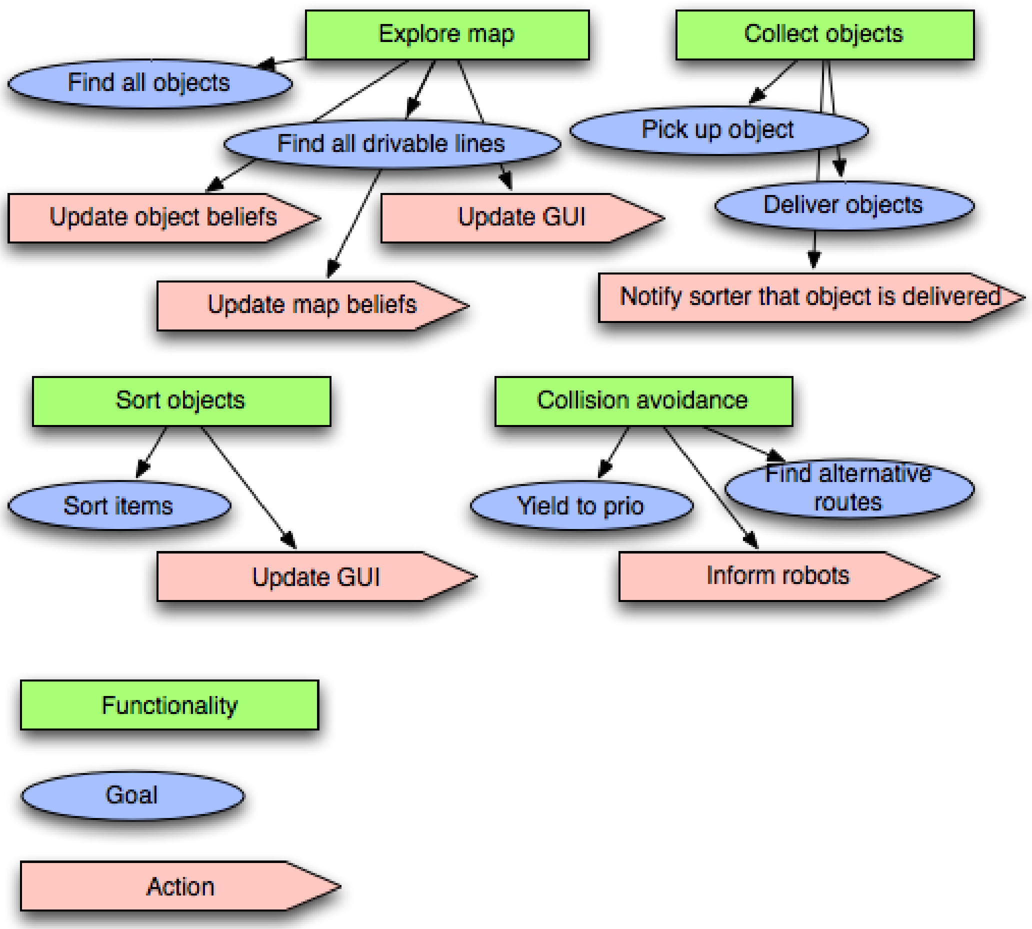


Figure 17: System functionalities based on goals

6.2 Architectural design

After defining goals and functionalities in the previous stage, 5 agents where identified to provide these functionalities and achieve the system goals. The agents and their specifications are shown in Figure 18:

- Agents for controlling the robots.
  −Explorer Agent.
  Agent with plans for controlling the explorer robot according to the defined goals. This agent communicates GUI updates and coordination requests as well as notifying the collector when items are discovered on the grid.
  −Collector Agent.
  Agent with plans for controlling the collector robot according to the defined goals.

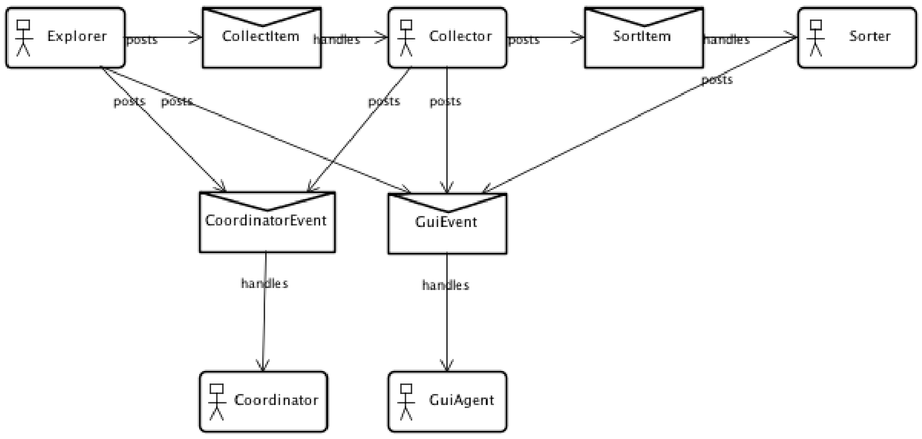


Figure 18: System Agents with basic interaction

Communicates GUI updates and coordination requests as well as notifying the sorter when items are deposited for sorting.
−Sorter Agent.
Agent with plans for controlling the sorter robot according to the defined goals. Communicates GUI updates and coordination, and handles sort requests from collector.

- Coordinator Agent.
  Agent for handling the movement coordination between the 3 robots. Keeps track of robot positions and headings to ensure collision avoidance.

- GUI Agent.
  Handles all communication with the GUI/operator. Updates of the gui as the robots gain more knowledge about their environment and also passes on user input to the robots/robot agents.

The number of agents and their respective tasks give opportunity for investigation of the research hypotheses. The design results in processing of information by the agents before presenting them to the operator, the agents reason on sensor inputs and communicate findings to the operator, and the system takes operator input, all important aspects of HRI. Having this setup of agents an expansion which could handle a complex unstructured environment would involve adding plans for handling the additional scenarios arising in such an environment opposed to the current structured one. The extra interaction needed between operator and system with this complication is relevant in context of hypothesis 2.

6.3 Detailed design

System overview shown in Figure 19:

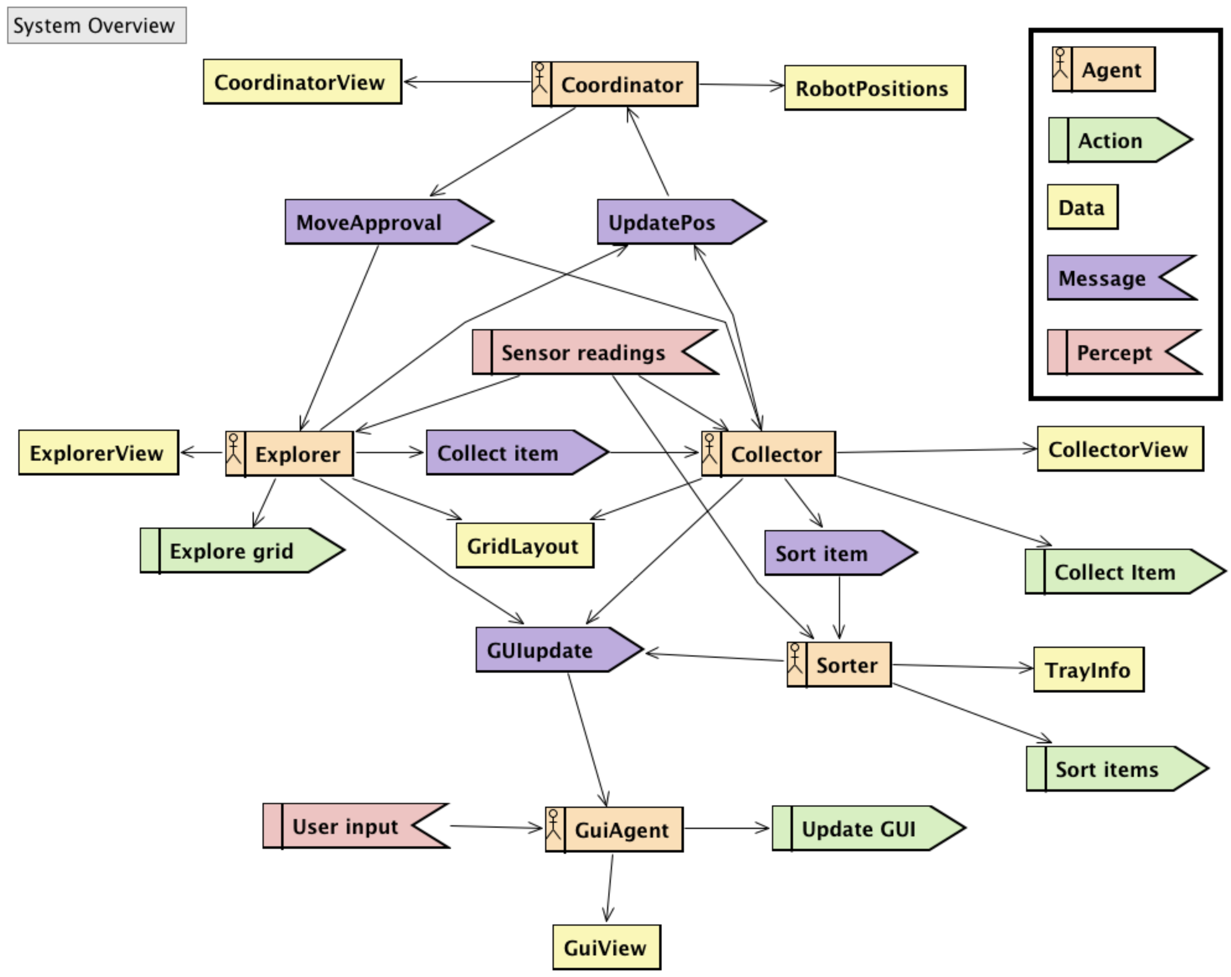


Figure 19: System overview

6.4 System - Robot communication design

The communication between the robots and the system will be done through Bluetooth. Communication classes system side will send commands to the different robots where code for executing these commands will be running. Results and sensor readings sent from the robots will be received and interpreted by the communication classes before being passed on to the agents. An illustration of this design is shown in Figure 20.

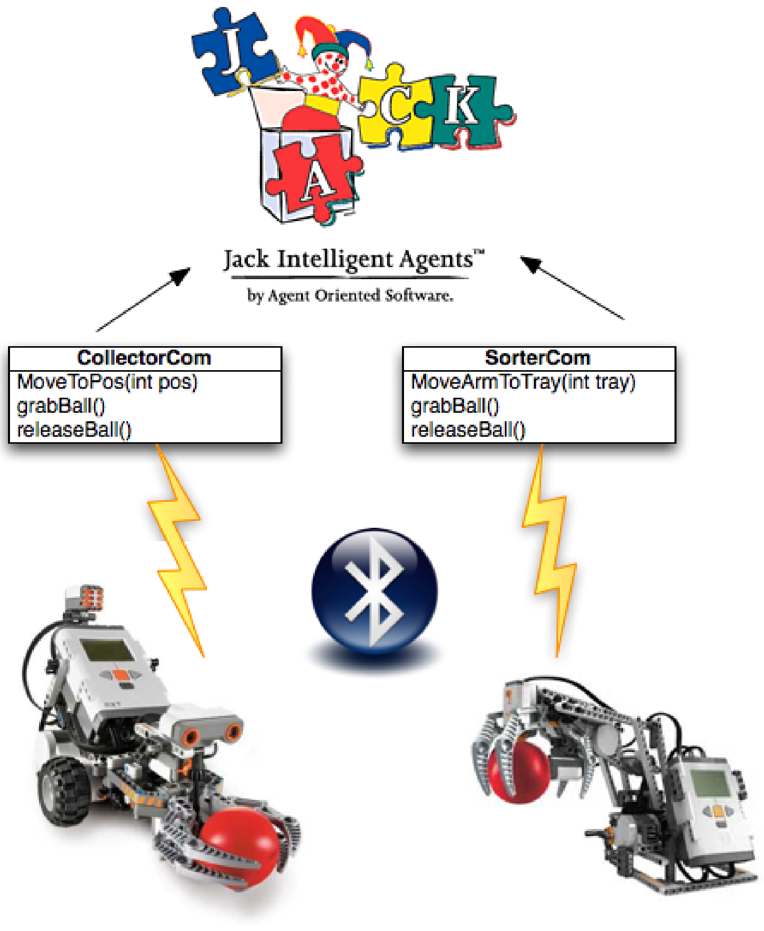


Figure 20: Communication design

6.5 Scenarios

This section will describe the scenarios that take place in the system relevant for this thesis. Figure 21 shows all the system scenarios which will or can occur during a normal running of the system.

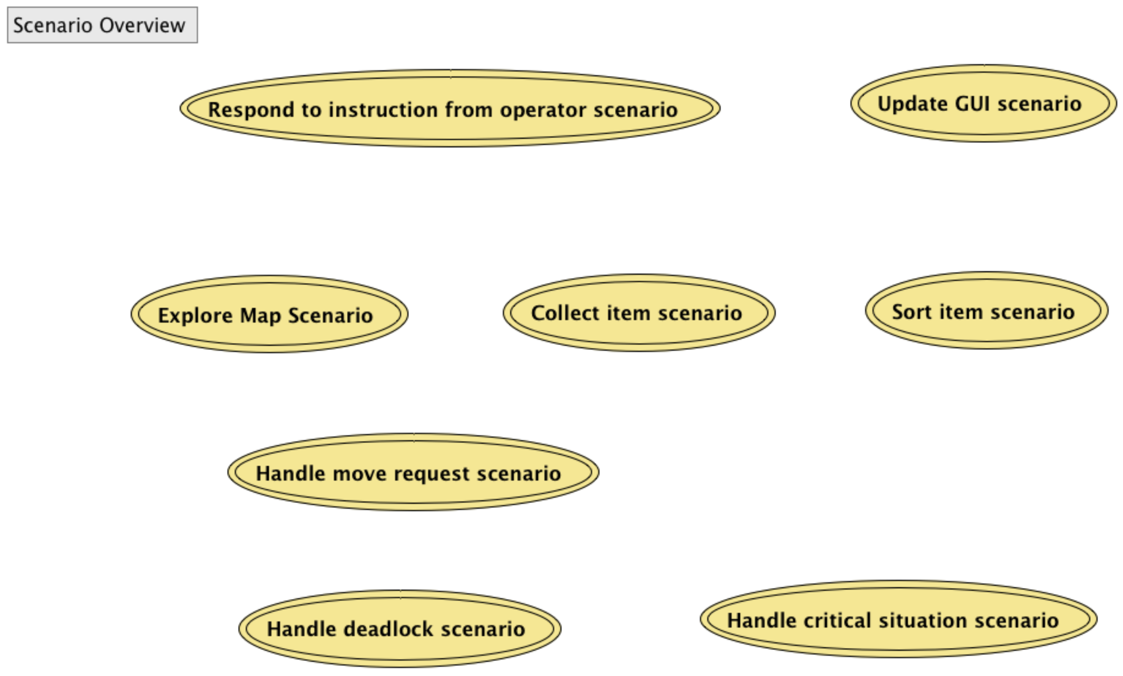


Figure 21: Scenario overview

[S1] Respond to instruction from operator.
Trigger: Command received from operator.
When the operator gives the start exploring command the system must initiate grid exploring.
1. Percept: Operators command.
2. Goal: Initiate execution of given command.
3. Action: Explore Map Scenario.
OR
4. Action: Stop exploring.
OR
5. Action: Initialize connections.

[S2] Update GUI

Trigger: New information has been obtained and needs to be updated in the graphical user interface.
As new information is gathered about the operational environment the GUI must be updated accordingly.
1. Percept: New environment information by sensor input.
2. Goal: Update GUI to correctly illustrate current knowledge about environment.
3. Action: Update GUI with new knowledge.

[S3] Handle critical situation

Trigger: Critical situation has occurred.
If a critical situation occurs which the system cannot handle without human intervention an alarm must be issued to the operator for evaluation and action choice.
1. Goal: Notify operator of critical situation.
2. Action: notify operator.

7 System Development

This chapter describes the implementation, see Figure 5 for symbol explanation.

7.1 Agents

This section will in short present the agents implemented in the system with a description and corresponding figures illustrating the workings of the individual agents.

7.1.1 Explorer

The explorer agent starts exploring when notified by the operator through the GUI. It uses a set of plans to achieve its objective to map out the available grid. It first checks available directions at its current position/intersection and stores this information in a beliefset. Based on available directions it chooses where to move and repeats step one at the next intersection until the entire grid is traversed. In addition to mapping it detects items to collect and notifies the collector agent during the exploration. The information obtained is continuously passed on to the GUI agent so that is can be presented to the operator. An overview of the explorer agent is shown in Figure 22.

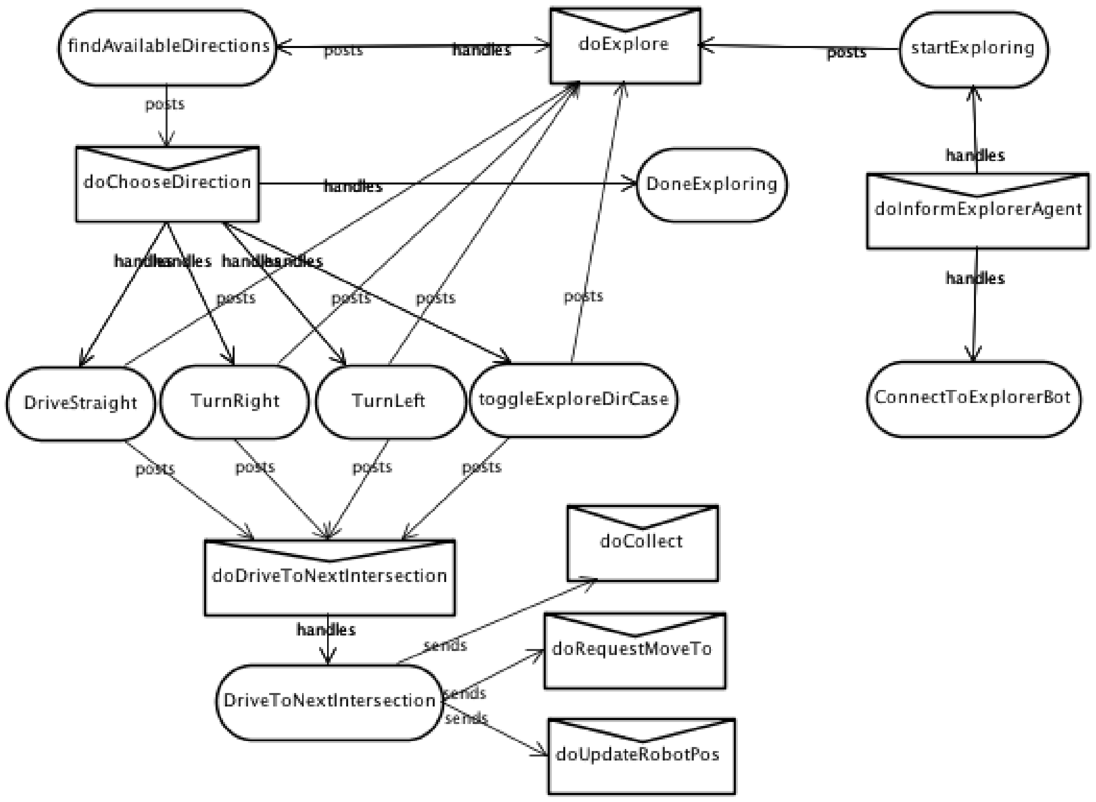


Figure 22: Explorer Agent overview

7.1.2 Collector

After being activated by the explorer, the collector agent first determines the shortest route to the item which is to be collected, then it moves to the item. The item is collected and a new shortest route to the sorter is determined before moving to deliver the item. After depositing the item the collector either repeats this sequence for next object to be collected or waits for a new notification from the explorer with item to collect. The GUI agent is continuously given information representing location and status of collection.

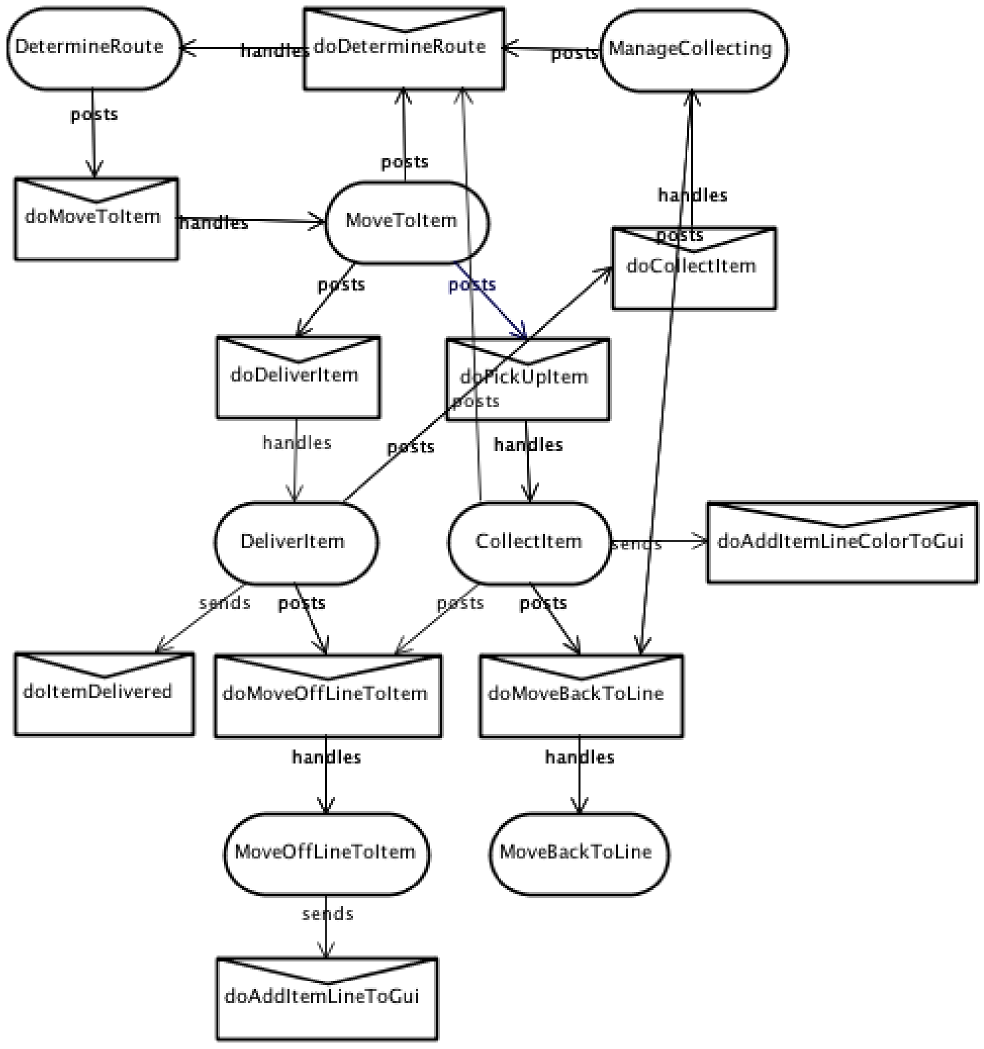


Figure 23: Collector Agent overview

7.1.3 Sorter

The Sorter agent is notified by the collector agent when a new object is ready to be sorted. The sorter then checks the object's color and queries its beliefset to see if the color already has a tray. If it has, the object gets placed in the same tray as the other objects of the same color, if not the object is put in to a new tray. The sorter also notifies the GUI agent that the object is sorted as displayed in Figure 24.

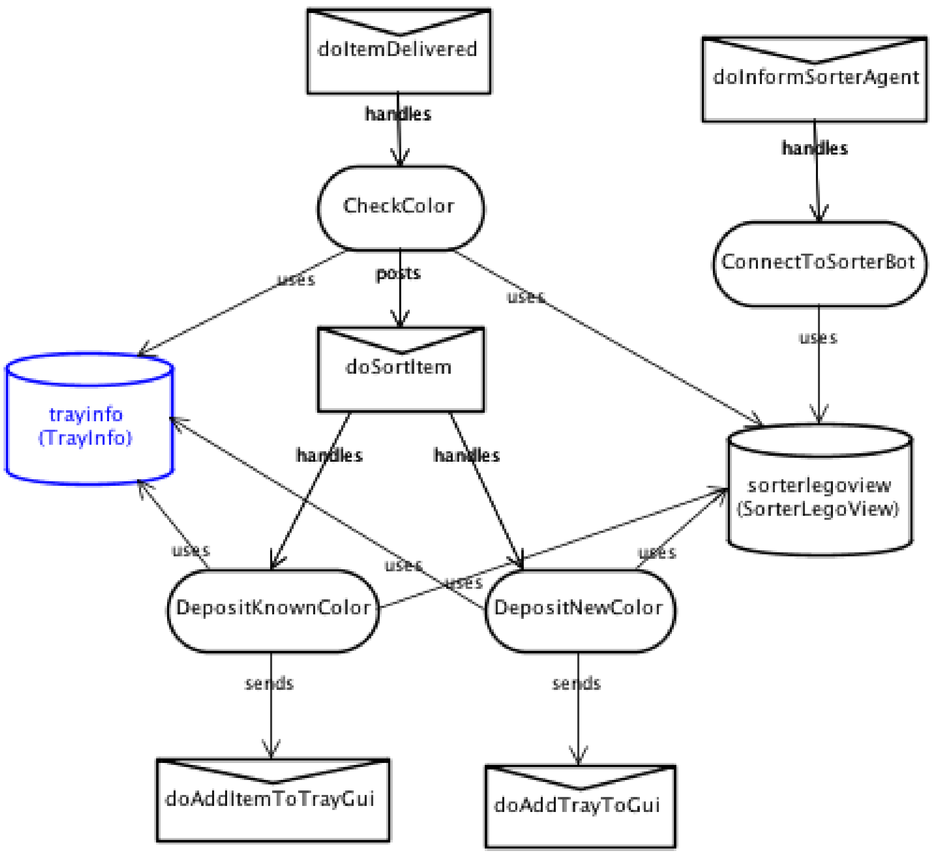


Figure 24: Sorter Agent overview

7.1.4 GUI Agent

The GUI Agent is responsible for handling communication with the external java graphical user interface. It handles events from the other agents and has plans for updating the GUI accordingly to the information received in these events. It also reacts to input from the GUI, and forwards the information to the relevant agents. Figure 25 and Figure 26 illustrate the workings of the GUI agent.

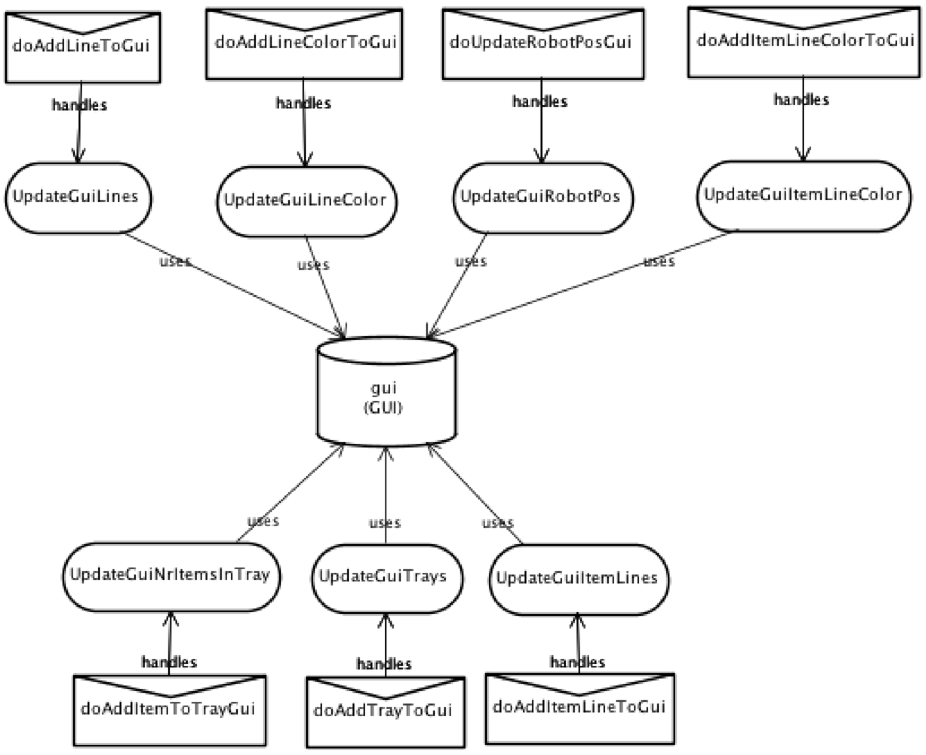


Figure 25: External communication from JACK to the GUI

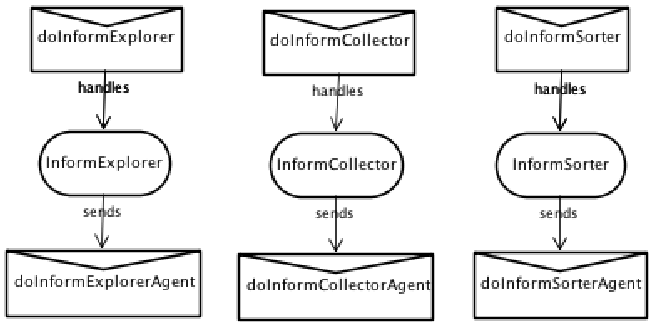


Figure 26: External communication from GUI to JACK

7.1.5 Coordination Agent

The Coordinator Agent is responsible for keeping track of the robots position and avoid deadlocks. The agent is also responsible for informing the GUI Agent about robot movement, as seen in Figure 27.

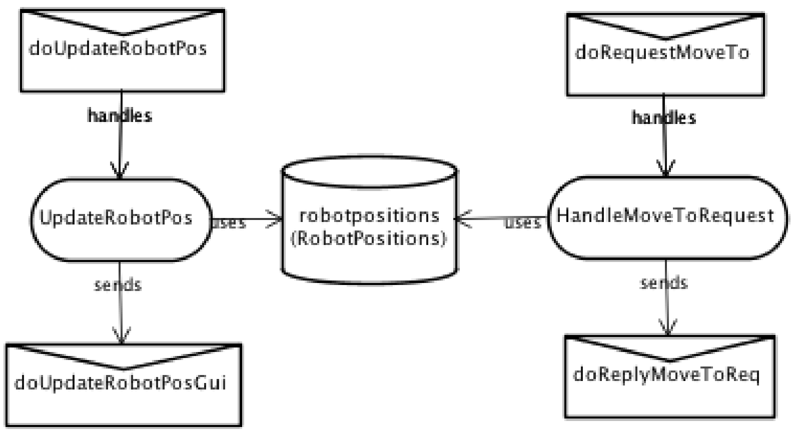


Figure 27: Coordinator Agent overview

7.2 HRI implementation

This section will give an overview of how and what information the different agents present to the operator and how the operator can affect the system. The human robot/agent interaction is implemented using JACK views. The interaction is a two way communication between the GUI agent and the operator, all operator input is through GUI actions and the robots present all results and status updates graphically, both via a JACK view.

GUI View

The gui view is the connection between the agents and the external user interface. The agents invoke methods in the view to update the user interface and the user input given is posted to the agents as message events through the view.

Message: doInformExplorer	
Description	A message event containing some information for the explorer agent given by an operator.
Sender	Gui view
Receiver	Gui agent
Information	The information from an operator to the explorer agent.

Table 3: Information from user to explorer agent.

Message: doInformCollector	
Description	A message event containing some information for the collector agent given by an operator.
Sender	Gui view
Receiver	Gui agent
Information	The information from an operator to the collector agent.

Table 4: Information from user to collector agent.

Message: doInformSorter	
Description	A message event containing some information for the sorter agent given by an operator.
Sender	Gui view
Receiver	Gui agent
Information	The information from an operator to the sorter agent.

Table 5: Information from user to sorter agent.

Explorer agent

Each time the explorer gains new knowledge about its environment this information is stored in a beliefset followed by a GUI update ensuring the display of all available environment data. This update is done by the posting of a message event with the relevant information, by either the beliefset or the agent, which is handled by the GUI agent. The GUI agent in turn invokes methods in the guiview thus updating the external user interface. The messages which result in visual updates for the user sent by the explorer agent or its beliefset are:

Message: doAddLineToGui	
Description	A message event instructing the GUI agent to add a new line to the graphical interface, this is done whenever the explorer discovers a new line on the grid.
Sender	Explorer agent
Receiver	GUI agent
Information	Start and end point of the line and a color defining the lines traversed status and if an object is located on the line.

Table 6: Add a line to the grid in the graphical user interface

Message: doAddLineColorToGui	
Description	A message event instructing the GUI agent to add a new color to a line in the graphical interface, this is done whenever the explorer has traversed a line and detected the color of the line.
Sender	Explorer agent
Receiver	GUI agent
Information	Start and end point of the line and the color defining it.

Table 7: Add a line color to the grid in the graphical user interface

Message: doUpdateRobotPos	
Description	A message event instructing the coordinator agent to register a new position for the robot sender, this is done each time a robot has successfully moved to a new position.
Sender	Explorer agent, Collector agent
Receiver	Coordinator agent
Information	The name of the robot sender together with its new position coordinates and current heading.

Table 8: Update robot pos message event

Collector

The collector uses the mapping provided by the explorer bot to navigate to collectable items. As it moves to and from items the GUI is constantly updated for the operator. The GUI is also updated when items are collected and no longer located on the grid. The updates are done by sending message events directly to the GUI agent or via the coordinator agent. The message events sent by the collector agent resulting in GUI updates are:

Message: doAddItemLineToGui	
Description	A message event instructing the GUI agent to add a new item line to the graphical interface, this is done whenever the collector detects an item line.
Sender	Collector agent
Receiver	GUI agent
Information	Start and end point of the line and the color defining it.

Table 9: Add an item line to the grid in the graphical user interface

Message: doAddItemLineColorToGui	
Description	A message event instructing the GUI agent to add a new item line color to the graphical interface, this is done whenever the collector has collected an item to indicate a successfull pickup.
Sender	Collector agent
Receiver	GUI agent
Information	Start and end point of the line and the color black which indicates that the object has been collected.

Table 10: Add an item line color to the grid in the graphical user interface

Message: doUpdateRobotPos	
Description	A message event instructing the coordinator agent to register a new position for the robot sender, this is done each time a robot has successfully moved to a new position.
Sender	Explorer agent, Collector agent
Receiver	Coordinator agent
Information	The name of the robot sender together with its new position coordinates and current heading.

Table 11: Update robot pos message event

Sorter

The sorter agent is in charge of sorting the objects delivered by the collector and the results of this sorting needs to be presented to the operator. The message events sent by the sorter agent to update the user interface are:

Message: doAddTrayToGui	
Description	A message event instructing the GUI agent to add a new item tray to the user interface, this is done when the sorter finds a new color not already added.
Sender	Sorter agent
Receiver	GUI agent
Information	The tray number of new tray and the color of items assigned to this tray.

Table 12: Add an item line color to the grid in the graphical user interface

Message: doAddItemToTrayGui	
Description	A message event instructing the GUI agent to update the number of items of a given color sorted into a tray.
Sender	Sorter agent
Receiver	GUI agent
Information	The tray number where the item is added.

Table 13: Increment the number of items in a given tray number.

Coordinator agent

The coordinator agent handles the coordination of the robot movements on the grid, as permission is given to move and the robot positions are updated thereafter the user interface must also be updated with the new positions and headings. The message event sent by the coordinator agent causing this update is:

Message: doUpdateRobotPosGui	
Description	A message event instructing the GUI agent to update the robot poistion of a given robot, this is done each time a robot has successfully moved to a new position with permission from the coordinator.
Sender	Coordinator agent
Receiver	GUI agent
Information	The new position of the robot, the robot name, and its current heading.

Table 14: New robot position and heading

GUI agent

The GUI agent handles all the events sent by the different agents for the various user interface updates, but it also sends events to the agents after receiving events generated by the guiview in response to operator input. The events sent to the agents are:

Message: doInformExplorerAgent	
Description	A message event containing some information for the explorer agent.
Sender	Gui agent
Receiver	Explorer agent
Information	The information, for example ”CONNECT” instructing the explorer agent to initiate connection to the explorer robot.

Table 15: Information passed to explorer agent

Message: doInformCollectorAgent	
Description	A message event containing some information for the Collector agent.
Sender	Gui agent
Receiver	Collector agent
Information	The information, for example ”CONNECT” instructing the collector agent to initiate connection to the collector robot.

Table 16: Information passed to collector agent

Message: doInformSorterAgent	
Description	A message event containing some information for the Sorter agent.
Sender	Gui agent
Receiver	Sorter agent
Information	The information, for example ”CONNECT” instructing the sorter agent to initiate connection to the sorter robot.

Table 17: Information passed to sorter agent

7.3 GUI implementation

The graphical user interface displays state information of the system with explored parts of the grid, items discovered and sorted. The different robots are also shown together with their corresponding movements and headings. The GUI implementation does not provide much functionality for operator input/influence as the implementation of the agent system is based on a structured environment due to time and LEGO Mindstorms limitations as stated previously. Currently the only influence an operator has is to initialize the connections between the agents and the robots and start the system with a ”Start” button. Figure 28 shows what the different components not explained in Figure 16 represent. A screenshot of the GUI with connections initialized is shown in Figure 29.

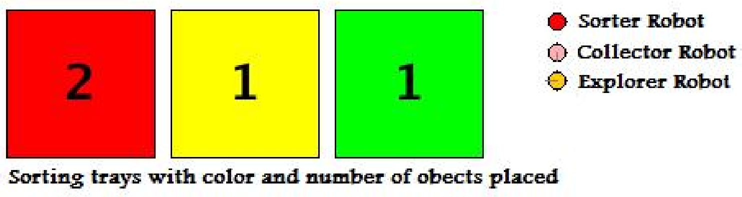


Figure 28: Gui components and what they represent.

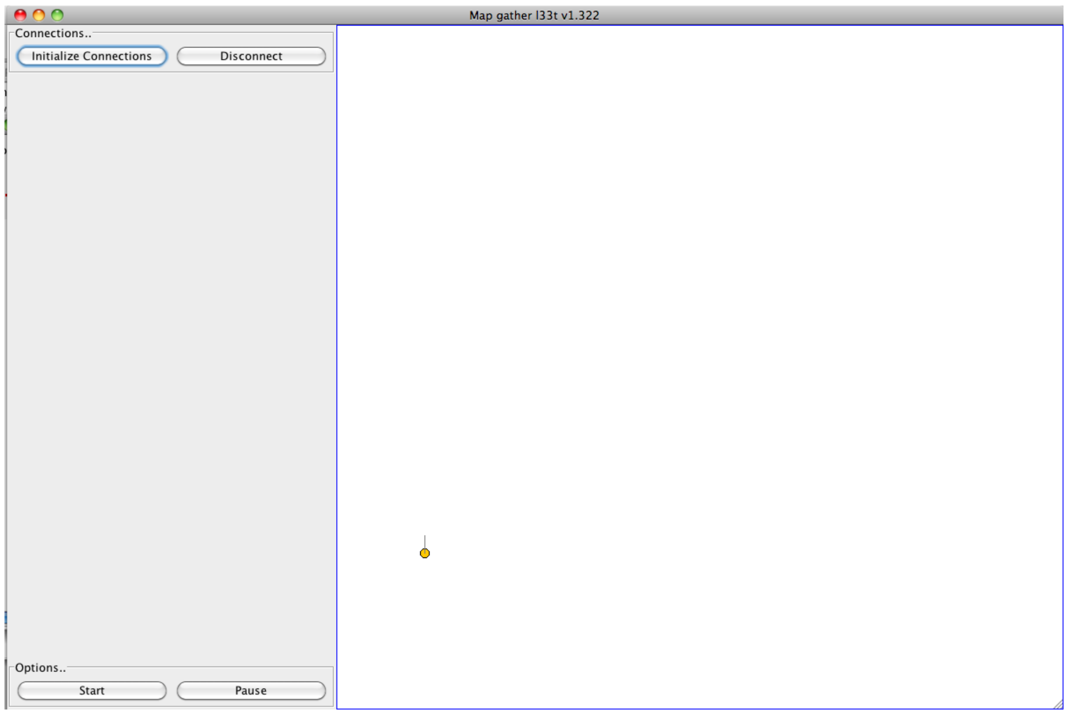


Figure 29: Gui after connections have been initialized.

After initialization of connections the operator can start the system by pressing the start button. Figure 30 shows the system during a normal run.

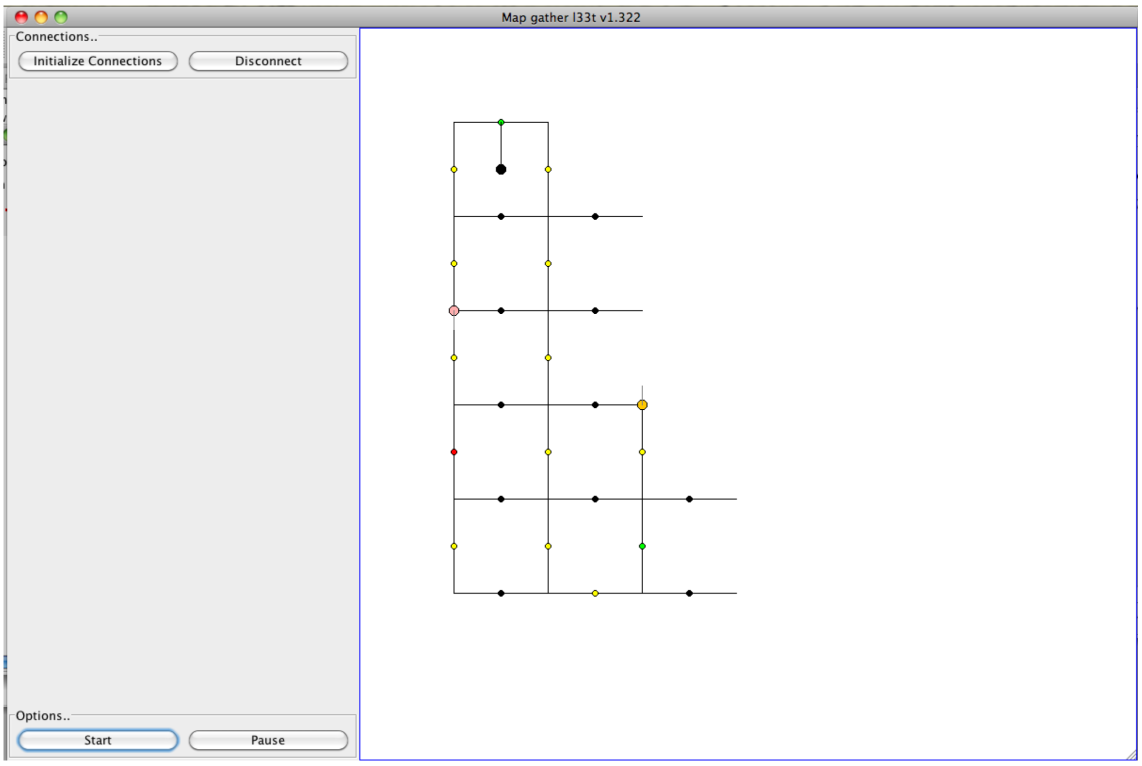


Figure 30: Gui some time after the start command is given.

While the explorer has traversed the entire grid the collector has collected items and delivered them to be sorted. In Figure 31 the entire grid is explored and a set of items have been collected and sorted by color.

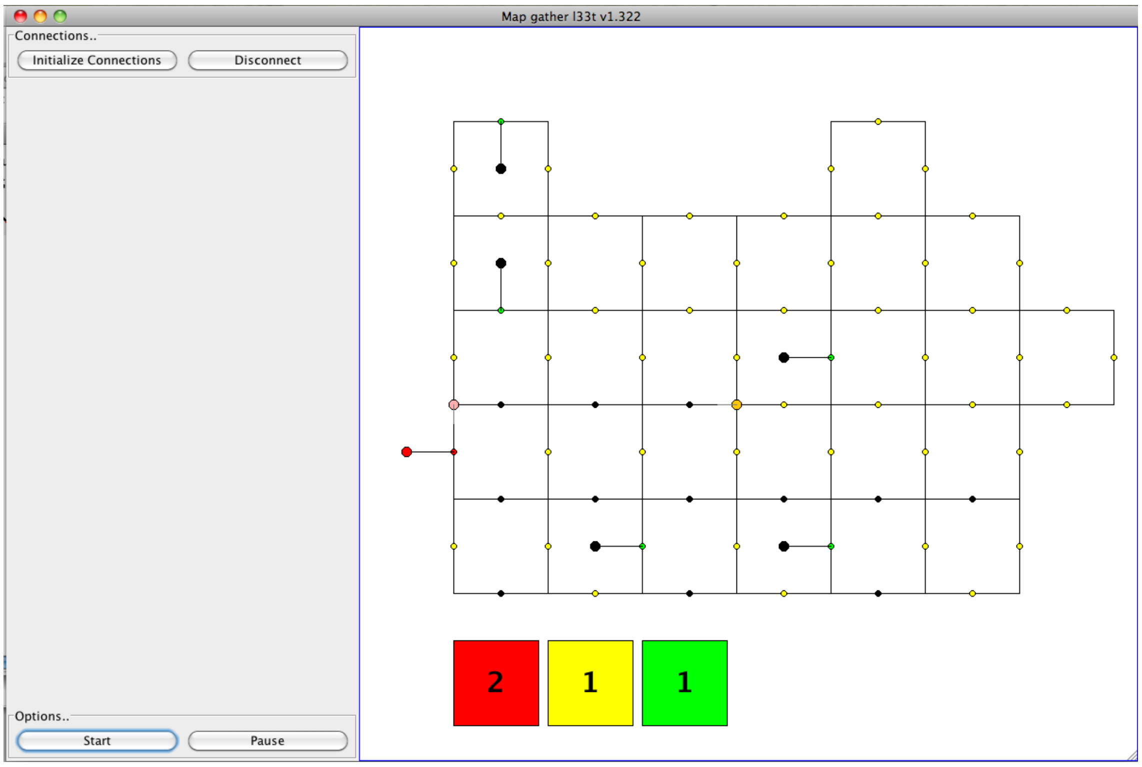


Figure 31: Gui after complete exploration (all objects not yet collected and sorted).

Hypothesis 1 is the main influence of the research hypotheses to the implementation of the GUI. This illustration clearly shows how for example the operator attention is drawn to the items sorted and what color they are as this is important information which is key for good HRI. Sensor readings are not presented to the operator in plaintext but as processed information visually. The agents provide results and findings which in turn are shown in the GUI in a manner according to the theory presented in chapter 3.

7.4 Robot development

The Lego implementation was not a priority during the development due to the limitations discovered relatively early in the process. Because of this the only fully implemented robot code is for the explorer robot. The collector robots code is partially implemented.

7.4.1 Communication protocol

The communication between the robot and system is done by sending commands using Bluetooth. Due to the limitations of Bluetooth technology such as high latency and low bandwidth we want to keep the communication protocol as simple as possible. The server sends its command in the form of three bytes, the first byte is the command it self, and the two following bytes are optional parameters. The robots reply is always 8 bytes which is enough to accommodate the most advances replies needed. For the different robot commands there are several cases to consider shown in tables 18, 19, 20.

Description	**Command**	**Reply**
Battery voltage request	[0,0,0]	[millivoltage,0,0,0,0,0,0,0]
Request to travel a given distance with or without checking the traveled lines color	[1, distance, boolean checkcolor]	[linecolor, 0,0,0,0,0,0,0]
Request to turn given degrees	[2,degrees,0]	[0,0,0,0,0,0,0,0]
read the color at current position	[3,0,0]	[color, 0,0,0,0,0,0,0]
Perform sweep at current location to discover available directions	[4,0,0]	[boolean straight, boolean left, boolean backwards, boolean right, 0, 0, 0, 0]
Disconnect Bluetooth	[5,0,0]	[255, 255, 255, 255, 255, 255, 255, 255]

Table 18: Explorer robot communication protocol

Description	**Command**	**Reply**
Battery voltage request	[0,0,0]	[millivoltage,0,0,0,0,0,0,0]
Request to travel a given distance with or without checking the traveled lines color	[1, distance, boolean checkcolor]	[linecolor, 0,0,0,0,0,0,0]
Request to turn given degrees	[2,degrees,0]	[0,0,0,0,0,0,0,0]
read the color at current position	[3,0,0]	[color, 0,0,0,0,0,0,0]
Perform sweep at current location to discover available directions	[4,0,0]	[boolean straight, boolean left, boolean backwards, boolean right, 0, 0, 0, 0]
Disconnect Bluetooth	[5,0,0]	[255, 255, 255, 255, 255, 255, 255, 255]
Grab object	[6,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]
Release object	[7,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]

Table 19: Collector robot communication protocol

Description	**Command**	**Reply**
Battery voltage request	[0,0,0]	[millivoltage,0,0,0,0,0,0,0]
Move object to tray position	[1, traynumber, 0]	[0, 0, 0, 0, 0, 0, 0, 0]
Read the color of object	[2,0,0]	[color, 0,0,0,0,0,0,0]
Grab object	[3,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]
Release object	[4,0,0]	[0, 0, 0, 0, 0, 0, 0, 0]
Disconnect Bluetooth	[5,0,0]	[255, 255, 255, 255, 255, 255, 255, 255]

Table 20: Sorter robot communication protocol

7.4.2 Internal robot code

The code located on the robot NXT brick is intended to provide as much functionality as possible with minimal amount of data sent using Bluetooth. At first, the robot waits for a Bluetooth connection. Once a connection is made, it waits to receive its three-byte command. Once the command is received, the robot moves or turns, if necessary, and then sends back its eight-byte reply one byte at a time. The robot then waits for its next command. If the robot is commanded to terminate its Bluetooth connection, the robot sends back its acknowledgment, disconnects, and its program terminates on the brick.

The traveling is implemented using a PID algorithm [23] which ensures that the robot stays on the line by constantly reading light values and readjusting accordingly. The code for this is shown as follows:

```java
private void PIDmove(int length) {
    int lightValue;
    int turn;
    int powerA;
    int error;
    int powerC;
    int lastError = 0;
    int derivative;
    resetTacho();
    while (getMM(motorA.getTachoCount()) < length) {
        lightValue = colorLightSensor.readValue();

        error = lightValue - offset;
        derivative = error - lastError;
        turn = (kp * error) + (kd * derivative);
        turn = turn / 100;
        powerA = tp - turn;
        powerC = tp + turn;

        if (powerA > 0) {
            motorA.setPower(powerA);
            motorA.forward();
        } else {
            powerA = powerA * (-1);
            motorA.setPower(powerA);
            motorA.backward();
        }
        if (powerC > 0) {
            motorC.setPower(powerC);
            motorC.forward();
        } else {
            powerC = powerC * (-1);
            motorC.setPower(powerC);
```

```
                motorC.backward();
            }
            lastError = error;
        }
        motorA.stop();
        motorC.stop();
    }
```

7.4.3 System side code

On the system side a communication class is developed for each of the robots interfacing between the robots and the agents. These classes are responsible for sending the commands one byte at a time to the robots and await replies. Once a reply starts being sent, the communication classes read each byte, one at a time, placing them in eight-byte arrays for interpretation before the results in turn are sent to the agents. The communication classes must implement interfaces defining required functionality for the given robot.

7.5 Scenario

Scenario 1 example shown in Figure 32. The operator gives the command to initialize connection to the explorer robot via the GUI. The GUI agent receives this command and passes it on to the explorer agent with a message event containing the information. Plans for handling this event are triggered in the explorer agent which initiates the actual connection, updates the coordinator agent with a start position for the robot. When the coordinator has registered the robots position an event for updating this in the external gui is posted and handled by the GUI agent and plans for updating the interface are executed.

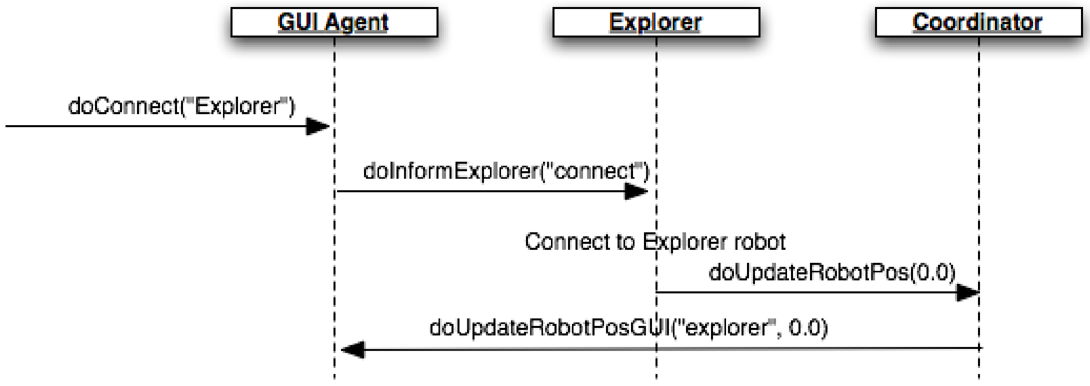

GUI Agent
Explorer
Coordinator
doConnect("Explorer")
doInformExplorer("connect")
Connect to Explorer robot
doUpdateRobotPos(0.0)
doUpdateRobotPosGUI("explorer", 0.0)

Figure 32: Example of scenario 1, initialize connection between explorer agent and explorer robot

Scenario 2 example shown in Figure 33. When the explorer has detected all available directions at an intersection this is updated in the GUI as "untraveled" lines. After traveling a line and checking its color this information is updated to the GUI and the line now has more detailed information then just being available for travel.

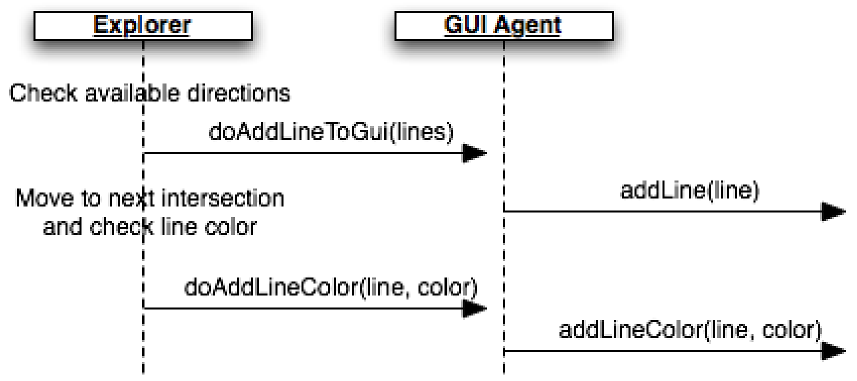

Explorer
GUI Agent
Check available directions
doAddLineToGui(lines)
addLine(line)
Move to next intersection and check line color
doAddLineColor(line, color)
addLineColor(line, color)

Figure 33: Example of scenario 2, update GUI with new environment data

Scenario 3 example shown in Figure 34. An example of a critical situation in the system is the deadlock scenario, where two robots want to travel the same line which would result in a collision. A situation like this is important for the operator to be aware of, thus the gui is updated with a deadlock warning whenever this occurs and a deadlock resolved notification when it is solved. As the system stands at current implementation with two robots in a structured environment, the agents will always be able to solve this situation, but considering multiple robots in unstructured environment, user interaction might be needed and crucial.

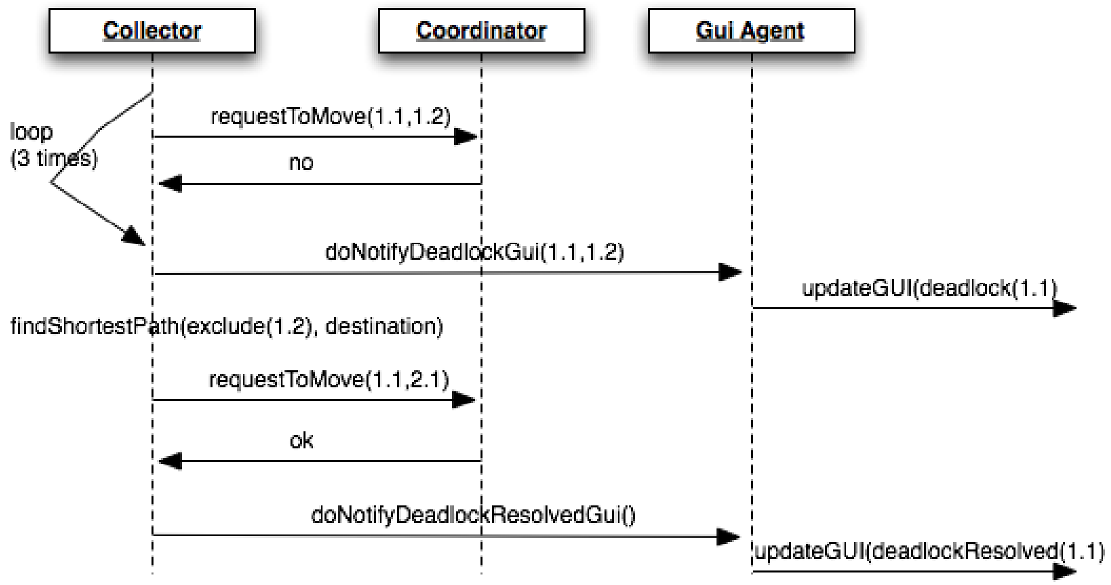

Collector
Coordinator
Gui Agent
loop
(3 times)
requestToMove(1.1,1.2)
no
doNotifyDeadlockGui(1.1,1.2)
updateGUI(deadlock(1.1)
findShortestPath(exclude(1.2), destination)
requestToMove(1.1,2.1)
ok
doNotifyDeadlockResolvedGui()
updateGUI(deadlockResolved(1.1)

Figure 34: Example of scenario 3, updating GUI with deadlock (critical situation) information

8 Results

This chapter presents the final solution with corresponding implementations, an overview of challenges met during the thesis work and an evaluation of the hypotheses presented in chapter 1.2 relative to the final implemented solution.

8.1 Final solution

The implementation goals set for this thesis where achieved with exception of a complete LEGO Mindstorms specific implementation of the defined interfaces. The Lego implementation was not a priority during the development due to the limitations discovered relatively early in the process. This lead to the final solution of implementing a set of java classes (mocks) [24] representing the robots and simulating replies and sensor readings. The downside of this approach is the obvious structured environment in which the agents now operate opposed to the desired unstructured and dynamic environment where the benefits of intelligent agents would be more visible.

8.1.1 LEGO robots and code

Three LEGO Mindstorms robots built according to TriBot [25] and RobotArm [26] schematics with modifications to meet our specific needs. To enable java programming on the Mindstorms intelligent brick the firmware was replaced with LeJOS [19]. Code for continuously receiving user commands and replying with results is implemented for the robots to run on the intelligent brick.

8.1.2 GUI and external java code

A graphical user interface is developed for the operator to interact with the robots. The operator can give input and observe a graphical representation of the robots, sensor readings and results during runtime. The GUI uses both color and placement to direct operator focus towards critical information. The GUI is shown in Figure 35.

Most of the algorithms used are implemented in pure Java, and used as external classes by the agents.

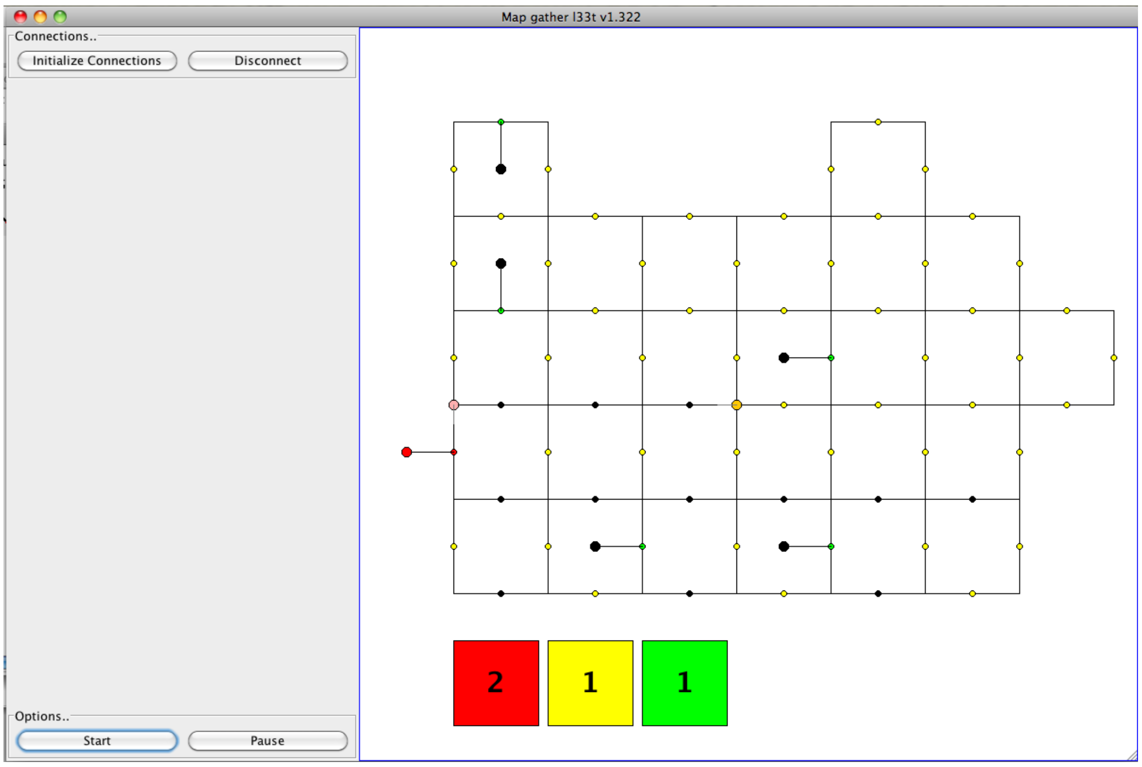


Figure 35: Graphical User Interface

8.1.3 Agent system

A total of 5 agents with respective views, beliefsets, plans and events where implemented, where three represent robots, one interacts with the GUI and the last agent is responsible for robot movement coordination:

- Explorer Agent
- Collector Agent
- Sorter Agent
- GUI Agent
- Coordinator Agent

8.2 Challenges

During the thesis work we have encountered several challenges both practical and technical. A summary of these challenges and how they where solved is presented in this section.

- JACK IDE
  The JACK IDE has several shortcoming compared to other well known IDEs such as Microsoft Visual Studio, Eclipse and IntelliJ. The most apparent being the lack of syntax highlighting, syntax error correction/help and code completion. Shortcomings of this kind in general result in slower development as well as unnecessary frustration as we are used to these features in all other IDEs. No other solution to this problem except just accepting the shortcomings and working with them.
- JACK compiler
  The JACK compiler does not support any Java language features above JDK 1.4 which includes java generics, simplified for statements, optional method arguments etc. This was solved by using these language features in plain Java files and compiling these files separately with javac.
- LEGO Mindstorms
  Generally robotics is a field with many challenges and with Mindstorms being a simple programmable robotics kit the weaknesses are more severe and not easily handled. The weaknesses we have encountered include non accurate sensor readings, limited computational power and poor communication support. These limitations resulted in excessive time usage and thus we where required to give this part of the development less priority especially because this was not the main focus of the thesis. In addition to less priority we where forced to adjust the desired complexity in our implementation goal. Instead of having an unstructured environment as intended a structured grid solution was adopted and implemented.

8.3 Hypotheses

The focus of this thesis has been external agent communication (GUI, ROBOTS AND HUMANS) with hypotheses:

Hypothesis 1: "Human Robot Interfacing (HRI) can be improved by the use of software agents."

Through our work and development of the agent system for controlling the Lego robots JACK intelligent agents have proven to be, in our opinion, a good solution. The properties of BDI intelligent agents (social, reactive, proactive) make agents a viable approach which provide well suited means for communication between system and operator The way agents can do reasoning on data and report goal related results to the operator when needed as well as involve user interaction if necessary is achieved in an intuitive and efficient manner. Plans for handling all operator input and plans for presenting results or involving a user ensure system integrity in regards to HRI.

Based on our results we believe that hypothesis 1 is true, but further research with more complex systems should be done to confirm this conclusion.

Hypothesis 2: "A change in operational environment from more complex (unstructured) to simpler (structured) environments will have significant affects on the human-agent system interaction."

The environment of operation is essential to the level of HRI and autonomy desirable to optimize system performance. With a structured environment as implemented in this thesis the optimal level of HRI is quite low as the agents can operate at a high level of autonomy with excellent results. Introducing a more dynamic unstructured environment would increase the need for operator involvement as the amount of critical situations would drastically increase and scenarios not well suited for autonomous solving would most likely multiply. We do not however believe that this would change the fact that intelligent agents have proven to be a good approach to this type of problems thus confirming hypothesis 2 to a certain degree.

GUI

Due to the structured environment the final GUI has a limited amount of operator interaction functionality. As the agent solution is nearly fully autonomous the only input needed from an operator is to initialize the connections and start the execution. The focus of the GUI implementation has been to visualize the results and beliefs of the agents (robots) during runtime. As well as directing the operators focus towards the important information at all times.

9 Conclusion

The goal of this Master thesis was to investigate the more complex challenges related to autonomous systems with focus on agent-human interfacing. In order to achieve this, we implemented a multi agent system designed for controlling a set of LEGO Mindstorms robots, Lego specific code for realizing the needed robot functionalities as well as classes for interfacing between the two, GUI etc. Two issues of research where formulated; "How can an autonomous systems provide a "good" solution for human-agent system interaction" and "How will a change in the environment of which the system operates in affect the human-agent system interaction, ranging from more complex (unstructured) to simpler (structured) environments".

Although the operational environment of the agents/robots was simplified from unstructured to a structured environment and implemented as more of a simulator rather than actual robots working, the solution still leaves room for investigation of the research issues. A team of robots are given a common goal where they all need to perform different roles to achieve the wanted results. The robots must cooperate and coordinate amongst themselves while constantly updating and reporting results to an operator. Despite the structured nature of the environment and the high level of autonomy implemented there is still some operator interaction present and the way in which results are presented to the operator is an important aspect.

The agent system is capable of controlling the robots and running the scenario for any given grid map using our simulated environment. The agents act according to sensor data and information shared between the agents with some additional operator input. Our test runs show that the agents are able to handle all the defined scenarios regardless of map layout and report accurate results through the user interface. Given the good performance achieved for our specified scenarios, it is important to point out the structured nature of the operational environment as being an important factor. This being the case our experience with the use of software agents to realize operator - multi robot machine systems has been very positive and we believe it to be a good approach.

Our results led to the acceptance of hypothesis 1 and strong implications towards hypothesis 2 being true as well. This being the case, we believe that the goals set for this thesis to investigate the complex challenges related to autonomous systems with various focus areas of approach have been fulfilled.

10 Further Work

The problem definition we started out with turned out to be to excessive and complex due to time limitations. We ended up making several simplifications to the initial implementation goal. The main simplification was degrading from an unstructured and dynamic environment to a structured simulated one. Even though a lot of work was put into the physical robots we also had to abandon this part of the project unfinished allowing us to focus on the more important aspects of the thesis.

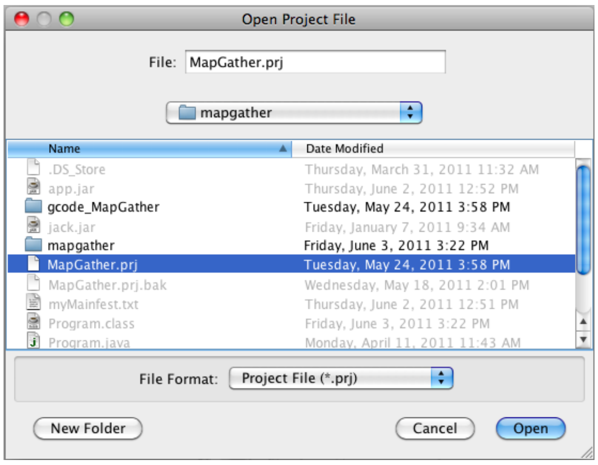
Further work on this project will be to finalize the actual robot implementation and have the physical robots working together with the agent solution as initially intended. This would require refinement of algorithms partly implemented for the different types of sensor input analysis and navigation. The next challenge is to change the operational environment and have the robots function without structured and predictable surroundings. Applying these environmental changes would lead to a greater need for operator involvement in context of critical situations. This involvement will include both input and decision making enabling us to utilize the agents capabilities even better. For example by having the agents provide the operator with a set of suggested solutions for a problem at hand relieving the operators workload.

We find the topic of agent systems very interesting and we would like to spend more time investigating it further both practical and theoretical. Although the agent community is rather small it will be exiting to follow future development within this field.

A JACK installation guide

The JACK framework is available for trial download at the aos group homepage [27]. After downloading the trial version install with the corresponding key you will receive by email after registration.

After installing JACK the thesis project can be opened through the standard file ⇒ open project menu as shown in Figure 36.



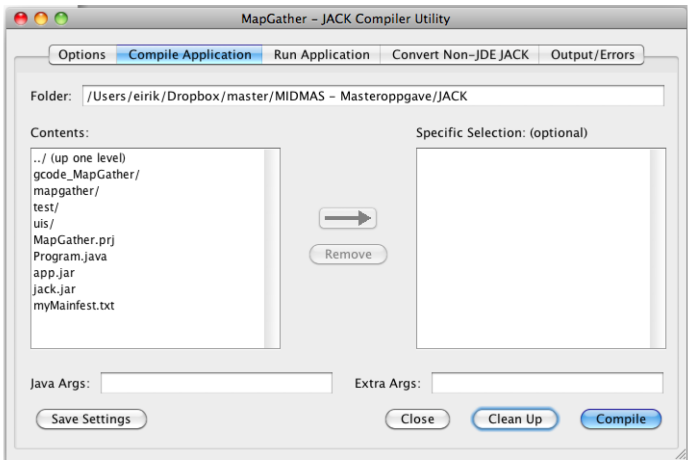
File: MapGather.prj

Name	Date Modified
.DS_Store	Thursday, March 31, 2011 11:32 AM
app.jar	Thursday, June 2, 2011 12:52 PM
gcode_MapGather	Tuesday, May 24, 2011 3:58 PM
jack.jar	Friday, January 7, 2011 9:34 AM
mapgather	Friday, June 3, 2011 3:22 PM
MapGather.prj	Tuesday, May 24, 2011 3:58 PM
MapGather.prj.bak	Wednesday, May 18, 2011 2:01 PM
myMainfest.txt	Thursday, June 2, 2011 12:51 PM
Program.class	Friday, June 3, 2011 3:22 PM
Program.java	Monday, April 11, 2011 11:43 AM

File Format: Project File (*.prj)

Figure 36: Open project in JACK

To compile the program press Tools ⇒ Compiler utility from the menu bar. The compiler window is shown in Figure 37.



Folder: /Users/eirik/Dropbox/master/MIDMAS – Materoppgave/JACK

Contents:
../ (up one level)
gcode_MapGather/
mapgather/
test/
uis/
MapGather.prj
Program.java
app.jar
jack.jar
myMainfest.txt

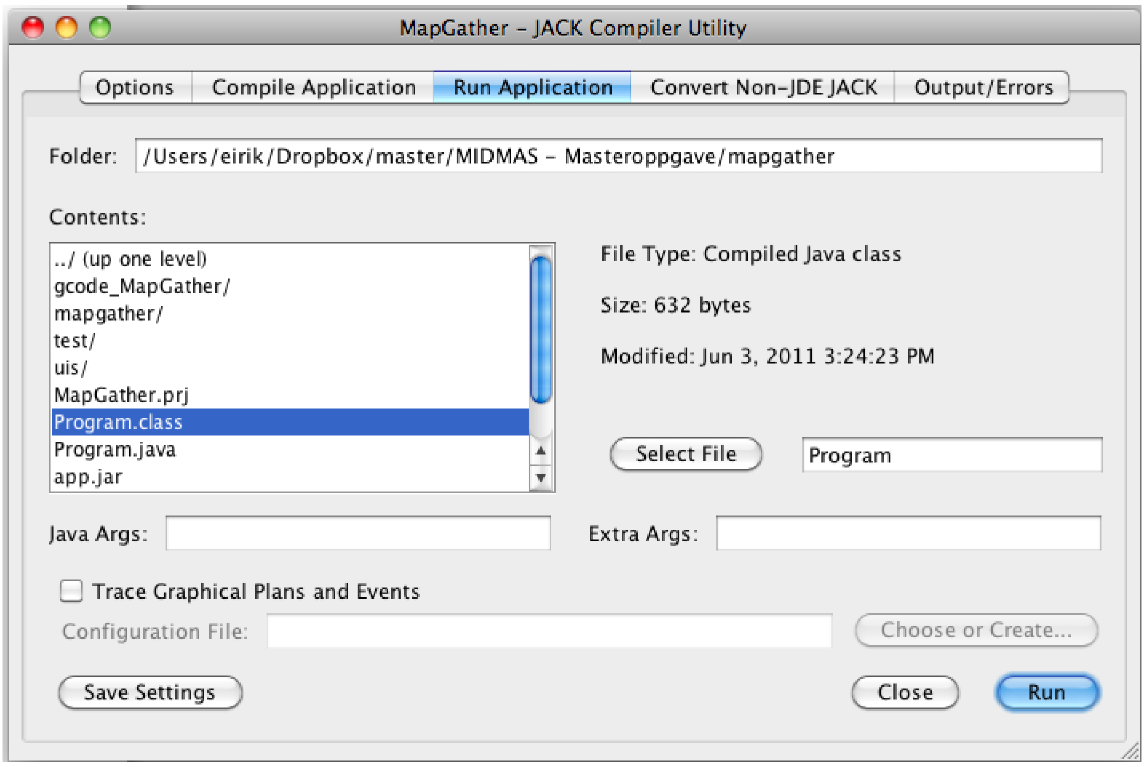
Specific Selection: (optional)

Java Args:

Extra Args:

Figure 37: Compile project in JACK

When the program is compiled successfully it can be run from the "Run Application" tab in the Compiler Utility window as shown in Figure 38.



Folder: /Users/eirik/Dropbox/master/MIDMAS – Masteroppgave/mapgather

Contents:
```
../ (up one level)
gcode_MapGather/
mapgather/
test/
uis/
MapGather.prj
Program.class
Program.java
app.jar
```

File Type: Compiled Java class

Size: 632 bytes

Modified: Jun 3, 2011 3:24:23 PM

Program

Java Args:

Extra Args:

Trace Graphical Plans and Events

Configuration File:

Figure 38: Run compiled project in JACK

For more details see the JACK Development Environment Manual [28].

B User Guide

The system requirements for running the application are Java 1.5 or newer. There is no need to install the JACK framework, as the jack.jar is included on the cd. Remember to have the jack.jar file in the same folder as mapgather.jar for the application to work.

The jar file(mapgather.jar) for running the program is located on the attached CD. After starting the program the connections to the robots need to be established, this is done by pressing the "Initialize Connections" button in the top left corner of the GUI. With the connections up press the "Start" button in the bottom left corner to run the collection scenario. The buttons are shown in Figure 39

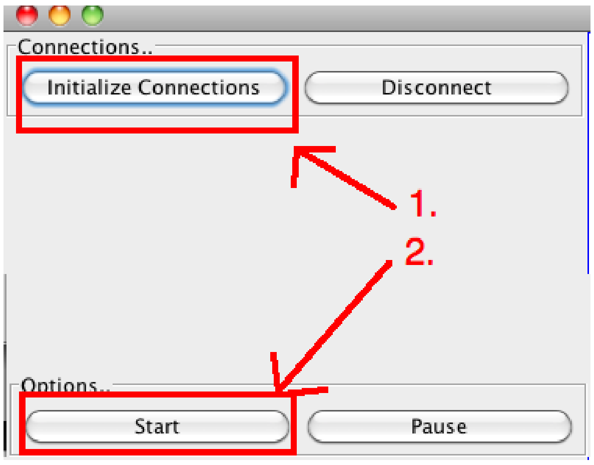


Figure 39: Buttons for running the program

References

[1] Elin Marie Kristensen. Agent technology. Master's thesis, Norwegian University of Science and Technology, 2005.

[2] Statoil ASA. Statoil in brief. http://www.statoil.com/en/About/InBrief/Pages/default.aspx.

[3] Robert N. Charette. Automated to death. *IEEE Spectrum*, 2009.

[4] Einar Landre. Autonomous systems & technologies, research and competence strategy, 2010.

[5] AOS Group. About aos group. http://aosgrp.com/.

[6] Lego. Lego mindstorms. http://mindstorms.lego.com/en-us/Overview/NXTreme.aspx.

[7] Raymond S.T. Lee. *Fuzzy-Neuro Approach to Agent Applications*. Springer, http://www.springeronline.com, 1st, edition, 2006.

[8] Lin Padgham and Michael Winikoff. *Developing intelligent agent systems - a practical guide*. WILEY, http://www.wileyeurope.com, 1st, edition, 2005.

[9] Michael E. Bratman. *Intention, Plans, and Practical Reason*. CSLI Publications, http://http://csli-publications.stanford.edu/, 1st, edition, 1999.

[10] T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Shultz, and A. Steinfeld. Common metrics for human-robot interaction. *Proceedings of IEEE International Conference on Intelligent Robots and Systems (Sendai, Japan)*, 2004.

[11] M. A. Goodrich and D. R. Olsen. Seven principles of efficient human robot interaction. *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics (pp. 3943-3948).*, April 30, 2004.

[12] S. R. Dixon and C. D. Wickens. Automation reliability in unmanned aerial vehicle flight control. *Proceedings of Human Performance, Situational awareness and Automation Conference (pp. 205-209).*, 2004.

[13] Josh Kaufman. *The Personal MBA: Master the Art of Business*. Portfolio Hardcover (December 30, 2010).

[14] T. B.; Wickens C. D. Parasuraman, R.; Sheridan. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics 2000, 30 (3), 286-297.*, 2000.

[15] A. Steinfeld. Interface lessons for fully and semi-autonomous mobile robots. *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA).*, 2003.

[16] Lise Engmo and Lene Hallen. Software agents applied in oil production. Master's thesis, Norwegian University of Science and Technology, 2007.

[17] Oracle. What is java? `http://java.com/en/download/whatis_java.jsp`.

[18] Jetbrains. About intellij idea. `http://www.jetbrains.com/idea/`.

[19] LeJOS. Lejos - java for lego mindstorms introduction. `http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/Intro.htm`.

[20] LabVIEW. About labview. `http://www.ni.com/labview/`.

[21] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

[22] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 2001.

[23] K.H Ang, G.C.Y. Chong, and Y Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology 13*, 2005.

[24] Alexander Chaffee and William Pietri. Unit testing with mock objects. `http://www.ibm.com/developerworks/library/j-mocktest/index.html`.

[25] Ro-botica.com. Tribot building instructions. `http://ro-botica.com/img/NXT/Build-Tribot.pdf`.

[26] Active-Robots.com. Robotarm building instructions. `http://www.active-robots.com/products/mindstorms4schools/building-instructions/Build-RoboArm.pdf`.

[27] AOS Group. Jack dowload site. `http://aosgrp.com/products/jack/index.html`.

[28] AOS Group. Jack development environment manual. `http://www.aosgrp.com/documentation/jack/JDE_Manual_WEB/index.html`.