NTNU
Innovation and Creativity

An analysis of the development of a safety-critical system for an Urban Search

Jean Paul Franky Friquin

Master of Science in Computer Science
Submission date: July 2006
Supervisor: Tor Stålhane, IDI
Co-supervisor: Siv Hilde Houmb, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

This report describes the design and implementation of a safety-critical system of a Lego Mindstorms Urban Search and Rescue (USAR) robot prototype.

Assignment given: 23. January 2006
Supervisor: Tor Stålhane, IDI

An analysis of the development of a safety-critical system for an Urban Search and Rescue robot using a Lego Mindstorms robot model.

NTNU
Innovation and Creativity

by Jean Paul Franky Friquin

July 10, 2006

under the supervision of
Professor Tor Stålhane and
NTNU PhD candidate Siv Hilde Houmb.

Submitted to the *Faculty of Information Technology, Mathematics and Electrical Engineering, Department of Computer Science and Information Technology* in partial fulfillment of the requirements for the degree of Master of Technology in Computer Science at the Norwegian University of Science and Technology, Trondheim, Norway.

Abstract

Earthquakes, avalanches, floods, cyclones, tornadoes and more recently tsunamis, or even airplane crashes, explosions like in the recent waves of terror in London [3], the United States [4], Bali,Indonesia [5] or the middle-east [6] are of those events that leave unforgettable scenes of catastrophic disasters. The consequences are such disasters are complicated rescue operations and chaotic information gathering. Today, dogs and humans are used in rescueing victims from disasters involving collapsed buildings, trapped tunnels and so forth. However, in some cases, robots have been deployed to assist the rescue teams. The tragic events of the World Trade Center in 2001 involved Urban Search and Rescue (USAR) robots pioneering rescue search with the help of information technology and robotics engineering. This report describes the design and implementation of a safety-critical system of a USAR robot model which will serve as a basis for analysis and development of future USAR robots. The scenario used in the experiment is a USAR robot prototype searching for survivors in a building which has collapsed. We analyse the scenario on a real prototype made up of Lego Mindstorms parts and equipped with a camera relaying real-time images to a workstation. The paper is a continuation of the report I submitted in December 2005; ‘Identifying the risks involved in the design of a safety-critical system for an Urban Search and Rescue robot’ [12].

Preface

Throughout the course of my studies in computer science, I have been developing software systems that could only be run on the computer screen. With this research work, I have fulfilled my desire to develop a tangible system. I feel that implementation of robotic systems gives developers a priceless, and far more enriching experience because of the interaction between hardware and software. This interaction is both fascinating and appealing to me as it calls upon putting into practice several areas of my education: physics, mathematics, software development, digital design and networking.

This report documents my thesis work, the culmination of a year's research about the implementation and design of a safety-critical system for an Urban Search and Rescue robot model. It has been written using the MiKTeX version of LaTeX 2_ε. The report describes the design, the implementation, the different components of the robot model and provides an insight into how the components are connected together to perform a search and rescue operation. Pseudo codes and algorithms are provided to show the sequence of instructions to perform various tasks. To further enhance understanding about the implementation of this robotic system, the source code files for the robotic system is listed in B two movie files are bundled together with the electronic version of this document. The report is divided into 10 chapters, each of which are described below.

Chapter 1: Introduction. This chapter introduces this report with the motivation, background, definition of the problem and a list of the main actors in the field of search and rescue robotics .

Chapter 2: Problem analysis. This chapter gives an overview of the objectives and software lifecycle used to implement the Lego Mindstorm USAR.

Chapter 3: The software development process. This chapter presents the functional and non-functional requirments for the USAR model. These requirements are a revised version of those from [12].

Chapter 4: System architecture. This chapter presents the architecture of the robotic system.

Chapter 5: Software, hardware and infrastructure. This chapter describes the software and hardware equipment used in this robot model.

Chapter 6: Navigation. This chapter presents the navigation module of the USAR robot giving an insight about the concepts and algorithms used for the movement of the robot.

Chapter 7: Communication. This chapter describes and evaluates the communication mechanism to enable reporting and remote controlling of the robot.

Chapter 8: Object detection, identification and localisation. This chapter presents how the victims are being detected, identified and localised in the arena.

Chapter 9: Presentation of information. This chapter presents the graphical user interface to present the information to the users of the system.

Chapter 10:Discussion and Conclusion. This chapter begins by identifying and discussing some of the problems encountered during the development and implementation of the system. It also presents areas for further work and concludes the project.

Acknowledgements

This research project began in the summer of 2005 with a telephone call from a Phd. candidate, now my adviser on this report, tipping me about the BUCS research project. Since then, the journey has been long and hard, exhausting at times, but yet very enriching and exciting. The research would not have happened if it were not for the help, guidance, and support of a number of individuals. First and foremost, I am thankful to my supervisor Professor Tor Stålhane for his advice and help. He has been very patient and helpful for the numerous times I knocked at his door asking for help and counselling. I wish to express my gratitude to my adviser PhD candidate Siv Hilde Houmb for general guidance, support and invaluable feedback for critically commenting and advising on my draft reports. I would also like to extend my gratitude to PhD candidate Pavel Petrovic. He has been of great help in implementing additional hardware components to the robot, and has been of great support, showing no doubts about the viability of the system.

In addition, I wish to thank Sigbjorn Pareli Lyngdal (Fire department, Trondheim, Norway) for sharing with me his expertise about search and rescue, and the Carpentry section at NTNU for helping me build the search arena.

I dedicate this work to my family: my two children, Frederik and Bianca, who have been the source of my inspiration and my wife, Kathinka for her love, patience and understanding during this arduous journey of graduate study.

- Jean Paul Franky Friquin

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The project 'Identifying the risks involved in the design of a safety-critical system for an Urban Search and Rescue robot' [12], describes the qualitative research to identify the requirements in building a safety-critical system for a USAR robot. The document was submitted in December 2005 and the main focus was to look into the requirements and the associated risks in developing a Lego Mindstorm-based prototype USAR. The report also looked into the development life cycle of a safety-critical system based on the international standard IEC61508 for safety-related systems and CORAS standard platform for risk assessment. In order to identify and analyse the risks, I performed a Preliminary Hazard Analysis (PHA), which is tailored for early stages of development such as requirement analysis. The PHA and HazOp analyses carried out to identify and analyze the risks were presented as well as an insight into the implementation of the robot model. However, the physical design and implementation part were not included. Therefore based on this pre-study project and the results obtained, we will in this report describe the aspects of implementing a low-cost USAR robot model, named Usario[1] . The robot model will serve as a basis for analysis and development of future USAR robots. The scenario used to simulate the operation of the USAR robot prototype is a search and rescue mission within one floor of a collapsed building. The core base of the robot model is built of Lego Mindstorms parts and equipped with a camera relaying real-time images to a workstation.

[1] *The robot model will be referred to as Usario, which, in creole, means small USAR. In creole, a dialect of French, the suffix -io in words means small or diminished. The objective of naming the robot model is to avoid confusion, easier reference and greater convenience when the robot is referenced which could be the robot used as model, a USAR or any other robot.*

1.2 Background

Currently, dogs and humans are used in rescueing people from disaster involving man made structures, like collapsed buildings. However, robots have been deployed in a few cases to assist the rescue teams. The University of South Florida, through its Center for Robot-Assisted Search and Rescue (CRASAR) developed a USAR robot to assist rescue workers by providing sound and images of places unreachable to dogs and humans. The robot was first used during the World Trade Center disaster in 2001, but had unfortunately neither found nor rescued any survivor. However, it has shown some great potential as described in the National Geographic article [25]. In order to have a more accurate picture of the situation in the case of a collapsed building, we had a meeting with the local Fire department in the city of Trondheim. The interview lasted for one hour and is documented in Appendix A. During the event of a building collapse, the police, the fire brigade, the paramedics, the Civil Defence (represented by the Red Cross or the 'Norsk Folkehjelp' in the event of big catastrophies), and the police dog rescue unit are the main actors at the scene. The police represents the legal local authority and is thus called the 'owner' of the accident. However, if the fire brigade arrives at the scene before the police, the former becomes the owner of the scene and passes ownership onto the police as soon as they reach the site. Paramedics do not have the authority at the scene and remain under the command of the police. The standard procedures require that the paramedics and/or firemen enter the scene only after it has been secured as 'no danger of an additional explosion', collapse or fire. From this point on, the objectives (in priority order) of the rescue team are to

1. Safeguard and rescue human lives.

2. Prevent deterioration and preserve assets, material property and infrastruture.

3. Safeguard and rescue animal lives.

In order to achieve these objectives, the rescue team is faced with challenges like determining the number of people trapped inside the building, their location and gaining an extended but adequate knowledge about the environment. Empirical knowledge about the dangers of the scene also helps the owners of the scene to assert the extent of the accidents and hence take appropriate measures.

1.3 Problem definition

After an accident, the conditions at a disaster area are extreme with many unknown parameters. According to the fire department, the safety of any rescuer should in no case be put at risk during a rescue operation. Rather than acting immediately without proper control, they would instead rely on information and intelligence to achieve their objectives. The use of Information technology and robotics can be of great use in such situations. Under such circumstances, eventual robots deployed will be subject to a series of hazards and risks that might affect their performance. Victims may be covered in debris and somehow trapped in between objects, which decreases the robot's chances for detection and rescue. Communication with the server might suddenly drop or even be lost. Hackers might break into the robotics software system thus jeopardizing the entire rescue operation.

In this case, pure gathering information and intelligence about the disaster site will not satisfy the requirements in its entirety. There is an additional need to implement an adequate and comprehensive robotic system to assist the police or the owners of the scene. The system must not only collect, process and present information about the scene, it must also act as a support tool such that the scene owners can use to support their continuous information needs without putting additional lives in danger.

1.4 Related work

Motivated by the positive contribution of robots in the World Trade Center disaster, search and rescue robotics is increasingly gaining interest from both academia and industry. Currently, there are numerous private companies, universities and some government institutions that are actively involved in the development of USAR robots. The following subsections present a list of some of the main actors in the field as well as a highlight of some robotics competitions intended for the general public, together with some groups of interest. The first part presents a list of some of the main actors in the field while the second part highlights some robotics competitions intended for the general public, together with some groups of interest. These are by no means exhaustive lists as there are several other relevant USAR robots available and other ongoing activites .

1.4.1 USAR robot development

USAR robot builders can be divided into three distinct groups; university researchers, private companies (here referred to as the 'industry') and government institutions. The three groups are introduced below.

University research:

1. The University of South Florida, through its Center for Robot-Assisted Search and Rescue (CRASAR) [25] developed a USAR robot to assist first-aid workers by providing sound and images of places where dogs and/or humans cannot reach. The robot was first used during the World Trade Center disaster, but unfortunately did neither find nor rescue any survivor.

2. Carnegie Mellon University's National Robotics Engineering Consortium (NREC) [7], a part of the Robotics Institute in the School of Computer Science, has contributed a lot in the field of robotics. In February 2005, they completed a highly successful prototype development program that validated their technology and which included conducting mobility and scout demonstrations of their system, known as the *Gladiator Tactical Unmanned Ground Vehicle* (GTUGV) [7]. In these demonstrations, the Gladiator prototype met all key system performance parameters, as well as other critical requirements.

3. As early as in 1995, Kobe University from Japan completed the development of their *Utility Vehicle for Search* (UVS) [18], involved during the Hanshin-Awaji earthquake. The UVS consists of several homogeneous small robots linked together and which can climb over obstacles. They have also developed a simulator for Robocup-Rescue, a simulation league that focuses more on coordination rather than victim detection and issues individual robots must solve.

Industry:

1. **iRobots Corporation** [16] is a private-owned company that develops robots for both the consumer market and the military. The *military R-gator* [16] and *packbot* [16] are their two most relevant contributions in USAR robots category.

2. **Inuktun Services Ltd** [15] is a canadian company that manufactures and markets steerable crawler robots. These robots are equipped with cameras and crawlers which provide detailed video output using variable lighting with pan, tilt and zoom controls.

Government institutions:

1. The NASA [22] is a pioneer and key leader in the development of robotics. It has developed, among other robots, *Urbie* to investigate urban environments contaminated with radiation, biological warfare, or chemical spills.

2. The United States Department of Defence [38] is a big player in the robotics market. In many of its key robotics programs, the United Defense Incorporation is the sole-source prime contractor and systems integrator. The department also hires and contracts other companies.

1.4.2 Robotics competitions

1. The **RoboCup Soccer** [29] competition, founded in 1995 is an international research and education effort. Its purpose is to foster Artificial Intelligence (AI) and robotics research by providing a standard problem where a wide range of technologies

can be integrated and examined. The ultimate goal of RoboCup is to develop, by the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall be able to play a soccer game against human world champions. The first edition of the competition took place in 1997 in Japan and has since been held every year.

2. **RoboCup Rescue project** [29] is a contest that focuses on the use of robotics as an aid in rescue operations. Its intention is to promote research and development in disaster rescue by involving multi-agent team work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants, standard simulator and decision support systems, evaluation benchmarks for rescue strategies and robotic systems that are all integrated into a comprehensive system in the future. RoboCup Rescue [30] is a child competition of RoboCup Soccer that started in 2001 and consists of three leagues:

    (a) The **RoboCup Rescue Robot League** is a competition entirely devoted to designing, building and programming of semi-autonomous and autonomous USAR robots.

    (b) The **RoboCup Rescue Simulation League** is a competition that focuses on group coordination during disaster rescue operations. A generic urban disaster simulation environment is constructed on network computers and every participant cooperates with many others, most often from other disciplines, to coordinate and conduct a rescue operation.

    (c) **Robocup Junior** [31] is a project-oriented educational initiative that sponsors local, regional and international robotic events for young students. It is designed to introduce RoboCup to primary and secondary school children, as well as undergraduates, who do not have the resources to get involved in the senior leagues. The focus in the Junior league is on education and it is divided into three categories: soccer, dance and rescue.

Chapter 2

Problem analysis

Having defined the problem in section 1.3, the objective of this chapter is to give an overview of the strategy used to implement a Lego Mindstorm USAR. In the risk assessment of [12], a list of requirements has aready been identified.

2.1 Introduction

This section summarises the identified safety and security requirements (from the report [12]) that need to be satisfied in the development of USAR robots. These requirements have been identified in the risk management process through a PHA and a HaZop, but are not an exhaustive list as further analysis and testing will probably reveal more system hazards. Note that most of the critical hazards (class types I and II) that have been identified, have been treated accordingly, while the other specifications have only been recorded during the conduct of the experiment. They are purely based on observation and are therefore marked as 'Observation'.

The safety requirements identified in [12] are:

1. Robot MUST detect all humans wherever there are humans. (**Detect requirement**)
2. Robot must be able to navigate through the entire site.(**Turn, Move, Follow requirements**)
3. Robot must be able to notify control unit about its own position at any time. (**Report requirement**)
4. Robot must be able to notify control unit about the location of victims within the shortest delay.(**Report requirement**)
5. Communication must be error-free and reliable.(**Communication requirement**)

6. Access to data must be controlled.(**Communication requirement**)
7. There should not be any buttons easily available for unauthorized personnel to tamper with. (**Observation**)
8. Battery/power level must always be monitored. (**Move requirement**)
9. Robot MUST NOT bump into humans. This can injure or deteriorate victims conditions. (**Detect requirement**)
10. Robot must back up on its original track in case of some malfunctions or physical damages.(**Turn, Move, Follow requirement**)
11. Robot must use the least time possible to accomplish its mission. (**Turn, Move, Follow, Report, Locate requirement**)
12. In case robot fails to perform according to established procedures, operator must take control over the system.(**Observation**)
13. Robot must operate independently in case of component/system failure. (**Observation**)

The security requirements identified in [12] are:

1. Only authorized personnel can have access to the system. The use of login and password is recommended.(**Validation requirement**)
2. Access to the system is restricted to authorized personnel holders of a private key or certificate (**Authentication requirement**).
3. Make use of virtual private network so as to ensure an encrypted tunnel of communication shutting down the threat of hackers tapping into the system.(**Reliability, Integrity requirements**)
4. Back up of data must be done as often and quickly as possible.(**Observation**)
5. Back up server must be at a different location, other than on site.(**Observation**)

Grouping these requirements, we get five distinct categories that will be described in detail throughout this report:

1. **Navigation.** Since the robot will be operating in a rapidly-changing and unpredictable environment, the robotic system has to be able to detect and adapt itself to the environment; it has to exhibit the property of context-awareness as it navigates within an unknown environment. Walls, obstacles, victims and unpredictable events have to be anticipated and catered for. The robot should be at all times ready to react to the situation in a safe and secure way. This issue is discussed in more details in Chapter 6.

2. **Communication.** As data are captured by Usario, simple computations are made locally before being sent to the server for further processing and presentation to the user. There is a need to establish a proper communication environment between the robot and the PC server. The different nature of data captured leads us to devise two communication medium which are wireless intranett and Infra Red/Bluetooth signals. This issue of communication and data transport protocol is discussed in more details in chapter 7.

3. **Object detection, identification and localisation.** The detection, identification and localisation of victims are all major aspects of the search mission.The robot needs to look for victims and at the same time identify and locate their positions within the accident site. Chapter 8 describes the technique and technology behind this search.

4. **Presentation of information.** A Graphical User Interface presents the information collected by the robot during its mission. Chapter 9 describes the objective and functionalities of the GUI.

5. **Security.** This category has not been dealt with in the implementation of this sytem because of time constraints. However, we feel that it is an important aspect of the system and it should be composed of a login facility with ID and password, as well as private keys on both client and server sides.

Based on these requirements, we can therefore implement a safety-critical system to help the rescuers in their mission. Our robotic safety-critical system will comprise of a Graphical User Interface(GUI) installed on a remote computing device and a USAR robot which will collect inputs from the environment and present the data on the GUI. In other words, the rescue robot must detect and report the position of victims (either dead or alive) and at the same time gather knowledge about the environment.

2.2 Objectives of the USAR robot model

The objective of the USAR robot model is to serve as a prototype which will eventually help in the building of a USAR robot to assist the owner of the accident. A USAR is a semi-autonomous robot designed to help emergency rescuers in detecting and locating survivors and cadavers in events of crashed buildings. The robotic system will collect data from areas of high risk for rescuers. The data is to be processed and presented as meaningful information to the decision makers. Moreover, the system should also allow a certain degree of freedom in the selection of the data e.g. the robot will be semi-autonomous as it will perform its mission both with and without human intervention. At any time during its search mission, the user must have the option to remotely stop and navigate the robot to any desired location.

2.3 Software Life Cycle

Life cycle models are useful means of describing the various phases of a development project, from the conception of a system to its eventual decommissioning.The software life cycle for this USAR robot model is an iterative one with prototyping and evaluation. The life cycle is described as iterative because at all stages of development, there was the need to cycle through the different development activities. Earlier stages were revisited allowing proper changes to be made when appropriate. Basically, there are four stages:

Stage 1. Requirements specification - Identify and define functional and non functional requirements - Completed in [12].

Stage 2. Design - Incomplete in [12]

Stage 3. Implementation - Incomplete in [12]

Stage 4. Testing - Left as a future assignment

Figure 2.1 shows the iterative process model that incorporates the ability to create prototypes during any development stage in order to explore unknown issues. The figure also shows another necessary step at each stage in the process. Before moving on to the next stage, the results of the current change were evaluated. Stage 1 and part of stage 2 were already completed in [12]. After the completion of the report, some aspects of communication and navigation were investigated further. As a result, new features could be implemented to improve the system. Therefore, amendments were made and the requirements specification has been revised accordingly[2] . Sections 3.1 and 3.2 explore this in more details.

[2] *A new HazOp should have been conducted to take into account these changes but is left for future assignments.*

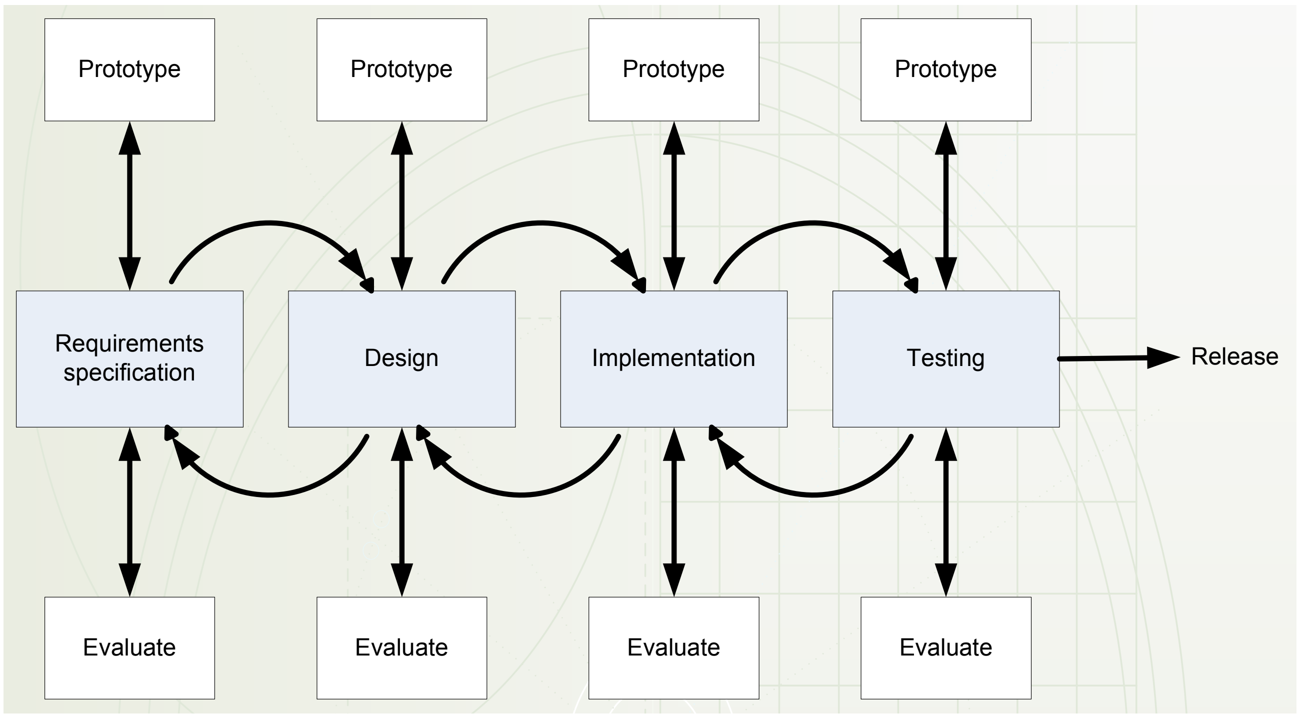


Figure 2.1: Iterative model with protyping and evaluation.

Chapter 3

Requirements specification (revised)

The main objective of an USAR robot during a rescue operation is to detect and report the location of survivors as a complementary aid to rescuers. However, during this task, the robots are exposed to uncertain events that may pose risk to the operation of the robot as very few of them operate in a risk free environment. The use of good risk management strategies and tools can aid in preparing the robot for some of these events. By identifying, assessing and controlling the risks involved, one makes what is unknown known and are hence able to incorporate proper mechanism. Therefore risk, safety and security management are important aspects of the development that have to be dealt with to ensure the proper running of the operation. This has been done in Friquin [12]. Here we present an updated requirements specification list.

3.1 Functional requirements

This section presents the functional requirements of the system. This functional requirements list is derived from the use case scenarios on page 35 in [12] and extended to incorporate the results from the risk assessment in [12]. The requirements are numbered, with the abbreviation F preceding the number.

F 1: Navigation
Usario shall be able to:
F 1.1 turn right.
F 1.2 turn left.
F 1.3 move forward.
F 1.4 move backward.
F 1.5 follow a coloured line. (OBSOLETE !!) This requirement is now obsolete as Usario can now move by sensing objects on its path.
F 1.6 avoid obstacles.
F 1.7 travel autonomously.(NEW !!)

F 1.8 be remote controlled.(NEW !!)

F 2: Detection
The camera vision of Usario shall be able to:
F 2.1 detect objects.
F.2.2 detect humans.
Usario shall be able to:
F 2.3 detect walls.
F 2.4 avoid walls.
F 2.5 detect obstacles.
F 2.6 avoid obstacles.

F 3: Identification
The camera vision of Usario shall be able to:
F 3.1 determine the state of the victims (a dead victim is represented by red colour and an alive victim by green colour).
F 3.2 capture image of the victims.
F 3.3 capture sound from the environment.

F 4: Localization
The RCX of Usario shall be able to:
F 4.1 determine its own position.
F 4.2 determine the position of the humans.

F 5: Reporting
The RCX of Usario shall be able to:
F 5.1 report its own position.
F 5.2 report the location of the identified humans.

F 6: Communication
The RCX of Usario shall be able to:
F 6.1 communicate in half-duplex with the PC unit through the IR - Bluettoth interface.(NEW !!)
F 6.2 communicate with the camera through a simple 4-pin flat cable.(OBSOLETE !!)This requirement is now redundant as the RCX can send data to the PC controller through the IR-Bluetooth interface.

The camera unit of Usario shall be able to:
F 6.3 communicate with the PC controller through wireless LAN.
F 6.4 Communicate with the RCX through a flat cable.(OBSOLETE !!) This requirement is now redundant as the RCX can send data to the PC controller through the IR-Bluetooth interface.

F7: Access
The system shall
F 7.1 validate and give access to authorized personel only.

F 8: Power

Usario shall:

F 8.1 be DC battery-driven.

F 8.2 be wireless.

F 9: Speed

Usario shall:

F 9.1 move at a reasonable speed.

F10: Presentation of information

The Graphical User Interface(GUI) shall :

F 10.1 graphically depict the top view of a rectangualr floor.(NEW !!)

F 10.2 display the image captured by Usario.(NEW !!)

F 10.3 graphically plot the path of Usario in operation.(NEW !!)

F 10.4 provide a joystick facility to enable the remote controlling of Usario.(NEW !!)

3.2 Non-functional requirements

The section presents the non-functional requirements of the system. These requirements place restrictions on the robotics system and the development process, as well as specifying external constraints that should be met. There are no changes to the non-functional requirements list from that specified in [12]. The requirements are numbered, with the abbreviation N proceeding the number and are divided into the following categories: environment, reliability, maintainability and upgradability.

N1 Environment

Usario shall not :

N 1.1 bump into objects.

N 1.2 hurt humans.

N 1.3 aggravate condition of victim(s).

N 1.4 damage property.

N2 Reliability

Usario shall not :

N 2.1 be operated/interfered/manipulated by unauthorised personel.

N3 Maintainability

Usario shall :

N 3.1 be easily modified for maintainance purposes.

N4 Upgradability

Usario shall :

N 4.1 be easily upgradable, by use of plug-in components.

Chapter 4

System architecture

The implementation of the safety-critical system has been based on a list of functional and non-functional requirements, which is itself a refinement of the above five categories of requirements. A general description of the system as well as the system design, class diagram and network architecture are discussed in this chapter.

4.1 System description

Usario, the USAR robot prototype, is a mobile robot searching for survivors and cadavers on one floor of a collapsed building. It is a semi-autonomous Lego Mindstorms robot capable of relaying sound and images of the scene to a server machine named PC. The critical issue is that Usario must detect **ALL** bodies or survivors in all the rooms it visits, in other words it must detect humans whenever there are humans in the vicinity. In addition, Usario must be able to correctly report both its own position and the position of the humans identified. Figure 4.1 represents an overview of the robotic system. A java program is downloaded to the Robotics Command Explorer (RCX) by use of a USB tower connected to the server. The program contains instructions in Java code to operate Usario in its search mission. Usario navigates around a one-floored site by avoid collision with the walls. The camera is constantly sending images of the scene (using 802.11g) to the server via the wireless access point. As soon as human survivors are within sight of the camera, the camera's colour recognition function detects the status of the humans (characterized by a green or red toy figure). At the same time, a message is sent by the PC to the RCX, to inform the latter about the identification of the victim. Meanwhile, an authorised user (shown as the operator in the drawing) who is connected to the server, monitors the progress of Usario on his own screen. He also has the possibility to remotely control Usario based on the information presented to him.

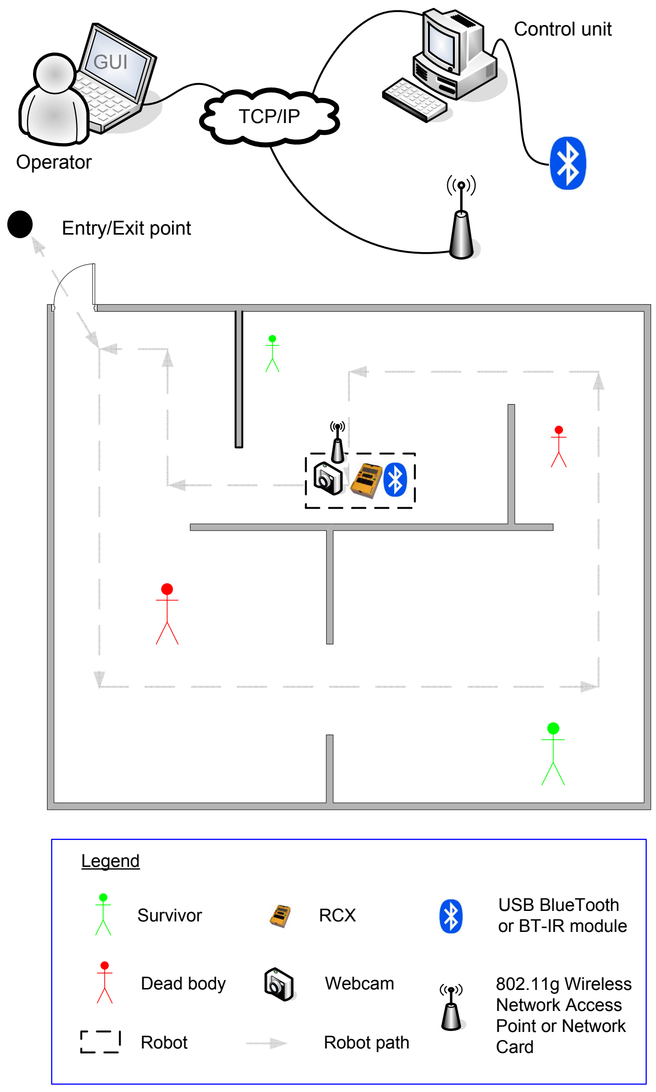


Figure 4.1: Overview of the system.

4.2 System design

The design of our robotic system has been an iterative process. An initial set of objects was chosen based on the requirements (sections 3.1 and 3.2) and general problem description of the system. As behaviours and scenarios were developed for these objects, the need for other object components became apparent. As the design was maturing, the objects were abstracted into classes with common functionalities that were grouped into parent classes with the detailed functions specified in child classes. The initial design effort focused on the overall structure of the program. The individual algorithms for methods were specified using pseudocode and will be presented over the next chapters. Tables 4.1 and 4.2 specify the main classes and functions required, together with a short description.

Package	Class	Method	Short description
PC			Controller class at the PC side
	GUI		Class for the GUI
		Frame	Deployment of the GUI shell
		Path Plot	Drawing of the robot path
		Remote Control	Joystick to remotely control Usario
		misc	Display of miscellaneous items
	commPC		Class controlling the data transport from PC to RCX
		sendData	Sends data
		receiveData	Receives data
	SimpleCommPC		Contains the data communication protocol
	TestDetect		Defines the detection parameters for objects
	Detect		Folder containing C++ files for object detection
	openCV		OpenCV's class for capturing images

Table 4.1: Class description for PC side

Package	Class	Method	Short description
RCX			Controller class at the robot side
	RCXcontroller		Class controlling the operations of the robot
		move	Starts the search mission
		scan	Allows robot to rotate over 360^0
		getPos	Calculates the position of Usario
		checkWall	Verifies the existence of walls
	Robot		Class controlling the basic functions of the robot.
		stop	Stops robot
		turn	Turns robot sideways to scan
		forward	Moves robot forward
		backward	Moves robot backwards
		travel	Moves robot over a specific distance
		rotate	Allows robot to rotate over a specific angle
	commRCX		Class controlling the data transport from RCX to PC
		sendData	Sends data
		receiveData	Receives data
	SimpleCommRCX		Contains the data communication protocol

Table 4.2: Class description for robot side

4.3 Class diagram

Figure 4.2 shows the class diagram in UML 2.0 notation for the server side of the system. The GUI class is the main class and which starts the commPC and Camera threads. The GUI makes use of the listeners in the 'eclipse.org.SWT' library package to detect occuring events within the class. The Camera class contains the settings for the parameters for the Detect DLL's and the Intel's OpenCVimage processing library. The commPC thread repeatedly transfers data packets to and from the GUI by using an instance of the SimpleCommPC class.

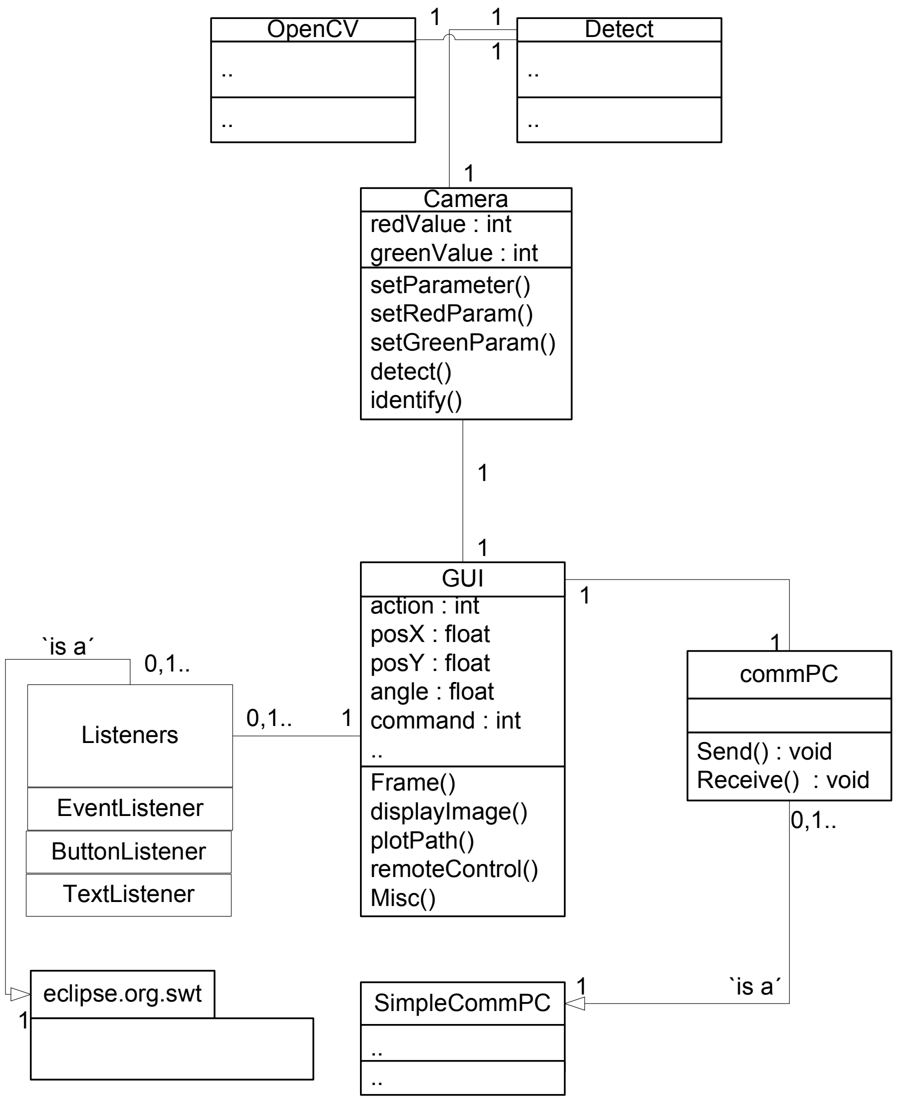


Figure 4.2: Class diagram for PC side in UML 2.0 notation

Figure 4.3 shows the class diagram for Usario's side as UML 2.0 notation. The RCX is the main class and is responsible for the operations of the robot. The Robot class implements the basic functions of the robot by interacting with the inputs (here the sensors) and the outputs (i.e. motors). The RCX retrieves information about the position of the robot to send it to the PC side by use of the commPC thread, which is itself an instance of the SimpleCommRCX class.

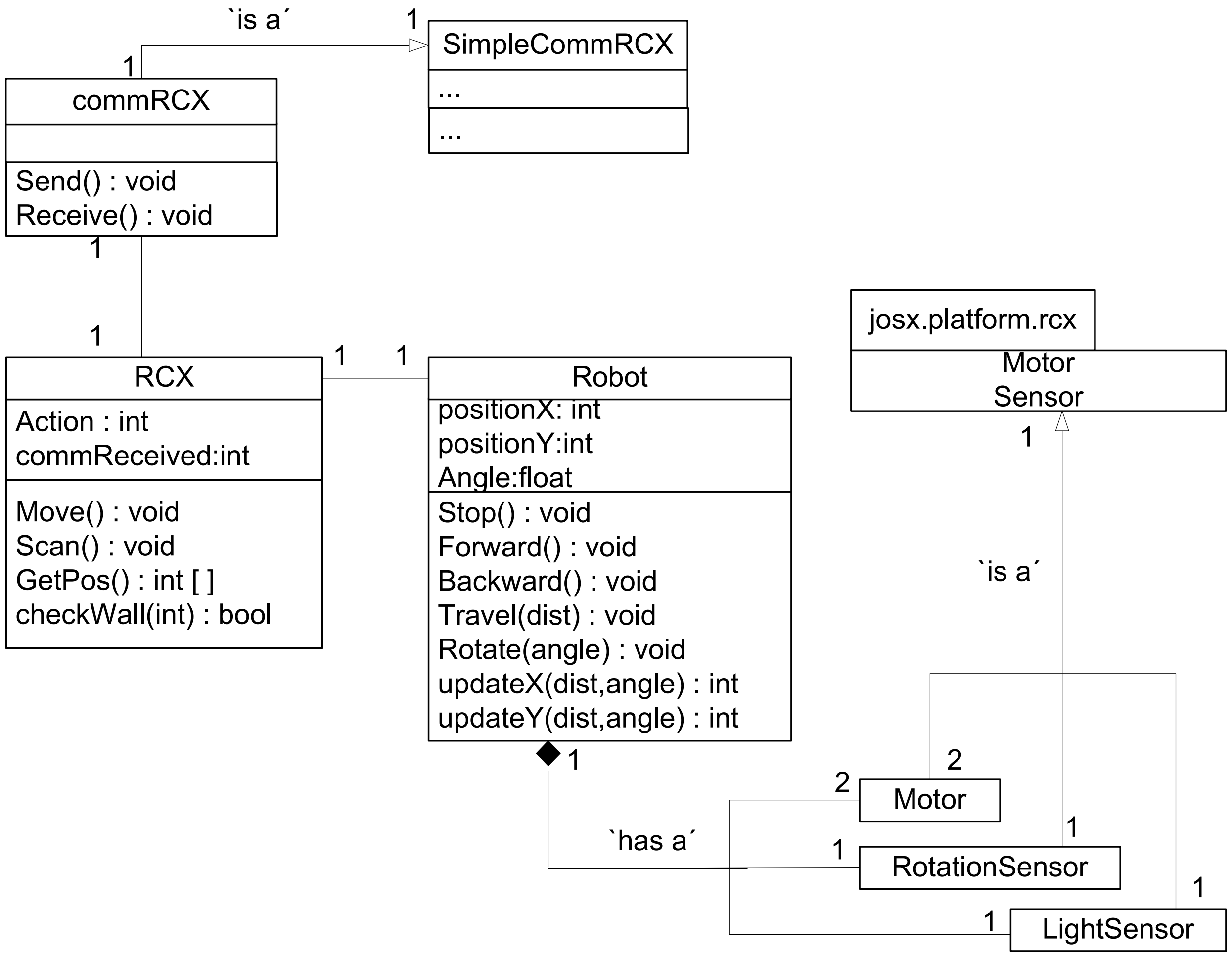


Figure 4.3: Class diagram for Usario side in UML 2.0 notation

4.4 Network architecture

The system is a three-tier architecture model. There are three communication interfaces: 1) between the RCX and the server (referred to as the PC), 2) between the client and the server, and 3) between the camera and the PC. Usario communicates with the PC in two ways: (1) through the InfraRed-Bluetooth interface and (2) via the camera through wireless Ethernet. Figure 4.4 depicts the architecture of the safety-critical robotics system. The first layer of the architecture contains the data link and physical layers of the OSI Model, the hardware components of Usario and the PC. The operating systems of both Usario and the PC are contained in the layer composed of the Network and Transport layers along with the LNP (Layered Networking Protocol) contained in the LeJos package.

The third layer represents the session, presentation and application layers, and contains the applications to enable display and capture the information at the client's side.

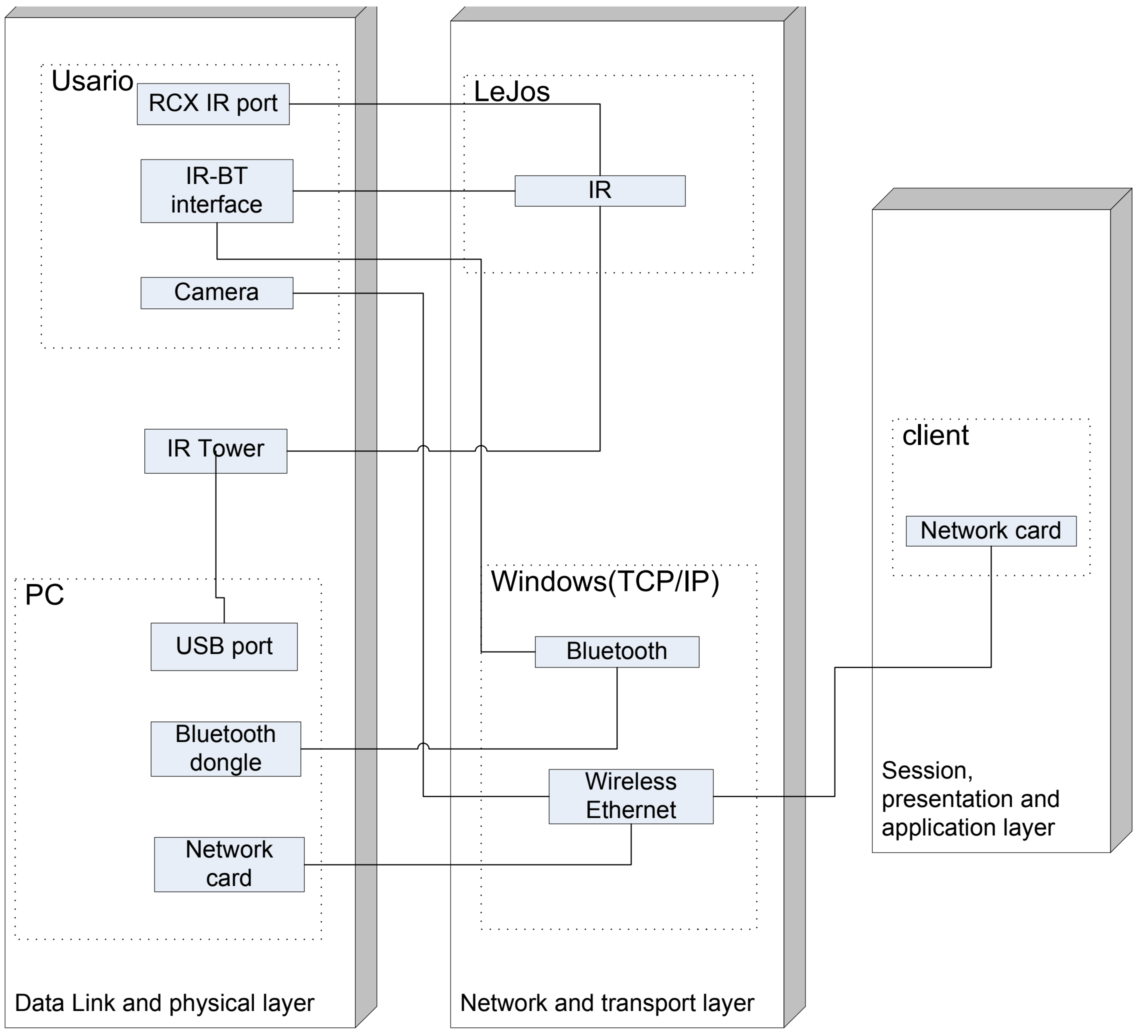


Figure 4.4: System Architecture

4.5 Code implementation and testing

The implementation of the system was carried out through a combination of building the physical robot and coding in paralell. As modular classes were built and incorporated with each other, physical modifications had to be made to the robot. For example, the 'navigation' function worked perfectly fine when tested alone. However, this was not the case when it was integrated with other modules, even though the functions were loosely coupled to each other. The reason was that as more units were added to the robot, its weight and size also increased, which meant that the motors, having a constant power output, needed greater torque to rotate the wheels.

P_{rot} (rotational Power) = τ (Torque)$\times$ ω (angular velocity)

This issue is discussed in more detail in chapter 10.1 and a listing of some of the classes is provided in B.

Chapter 5

Software, hardware and infrastructure

The first section of this chapter describes the software and development platform used. It introduces the various editors and the programming language used in the implementation of Usario. A more comprehensive description of the hardware equipment used is given in the second section.

5.1 Software

5.1.1 Development environment

In [12], *Chapter Evaluation of available programming languages page 68-69*, Java was identified as the most ideal implementation language for the robotic system. Based on the evaluation done and the fact that we are experienced and comfortable in programming with Java, it is still the preferred choice as programming language. We made extensive use of Eclipse and Textpad IDEs (Integrated Development Environments) to write, compile and run Java SE, Java AWT, Java SWT and LeJos codes in our implementation. The development platforms used are:

1. **Java** : an object-oriented programming language developed by James Gosling and colleagues at Sun Microsystems in the early 1990s. Unlike conventional languages which are generally designed to be compiled to native code, Java is compiled to a bytecode which is then run (generally using JIT compilation) by a Java virtual machine. The language itself borrows much syntax from C and C++ but has a much simpler object model and does away with low-level tools like programmer-manipulated pointers [20].

2. **Java Platform, Standard Edition or Java SE** (formerly known as J2SE) : a collection of Java programming language APIs useful to many Java platform programs [20].

3. **Java Abstract Window Toolkit** (AWT) : Java's original platform-independent windowing, graphics, and user-interface widget toolkit. The AWT is part of the Java Foundation Classes (JFC) - the standard API for providing a graphical user interface for a Java program [20].

4. **Standard Widget Toolkit** (SWT) : a Graphical User Interface (GUI) toolkit for Java. It is an open source toolkit and included as part of the Java standard and the Sun Microsystem's proprietary software. SWT accesses native code GUI libraries through the Java Native Interface (JNI) to display its GUI widgets [14].

5. **LeJos** : LeJos [32], is a Java environment for the Lego Mindstorms programmable RCX controller and provides a replacement to the default RCX firmware. It is an open source project and is currently available on both Windows and Unix systems. It includes a Java virtual machine which allows Lego Mindstorms robots to be programmed in the Java Object-Oriented programming language. It is portable, reliable and offers multithreading. However, it is worth to be noted that leJos, unlike Java, does not provide garbage collection facilities.

Whereas the IDEs are:

1. **Eclipse** : an open source platform-independent software framework for delivering what the project calls "rich-client applications", as opposed to "thin client" browser-based applications. Eclipse has been used for editing codes for both the RCX and the GUI.

2. **TextPad** : a text editor for the Microsoft Windows family of operating systems [11]. It has been used mostly for editing and compiling codes for the RCX using LeJos.

5.2 Hardware and infrastructure

The entire robotic system comprises of the following parts:

1. one robot chassis and various lego parts

2. one camera

3. one InfraRed-Bluetooth interface

4. one search arena

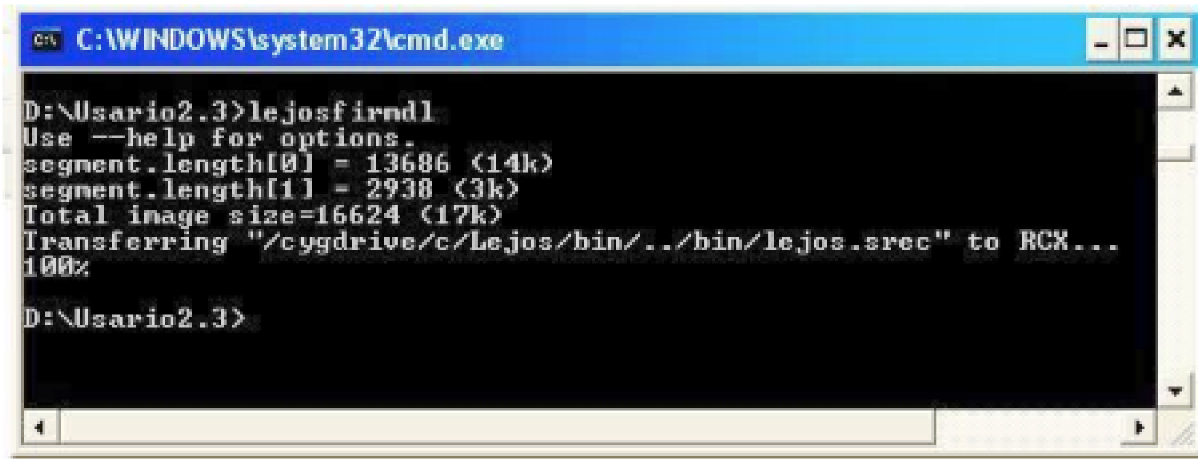
5. one PC server
6. one Bluetooth dongle
7. Three power supply units

They are described in more details in the following subsections.

5.2.1 The robot chassis and Lego parts.

The robot's chassis is made up of various Lego bricks and constitutes its backbone. The six most relevant Lego parts for this work are 1. RCX, 2. InfraRed proximity sensor, 3. motors, 4. IR serial port, 5. rotation sensors, and 6. wheels. A short description of these components, together with their configuration and calibration details are given below.

1. **RCX**
   Description: The RCX(see [12], pg 22) is the Central Processing Unit of the robot and is included as a standard part of the Lego Mindstorms 2.0. The RCX brick, as it is called, encloses a Hitachi H8/3292 series micro-controller. The H8 is capable of running at a clock rate of between 10 MHz and 16 MHz, and the RCX uses the highest clock rate of 16 MHz. This is extremely low compared to modern processors that run at speeds higher than 1000 MHz but it is fast enough for most of the functions of Usario such as turning motors on and off, reading input from sensors and computing the next logical move.
   Installation and configuration: The RCX is first loaded with LeJos, a replacement firmware for the Lego Mindstorms RCX brick. Figure 5.1 shows the downloading of the LeJos firmware, a Java Virtual Machine that fits within the 32KB ROM on the RCX brick.

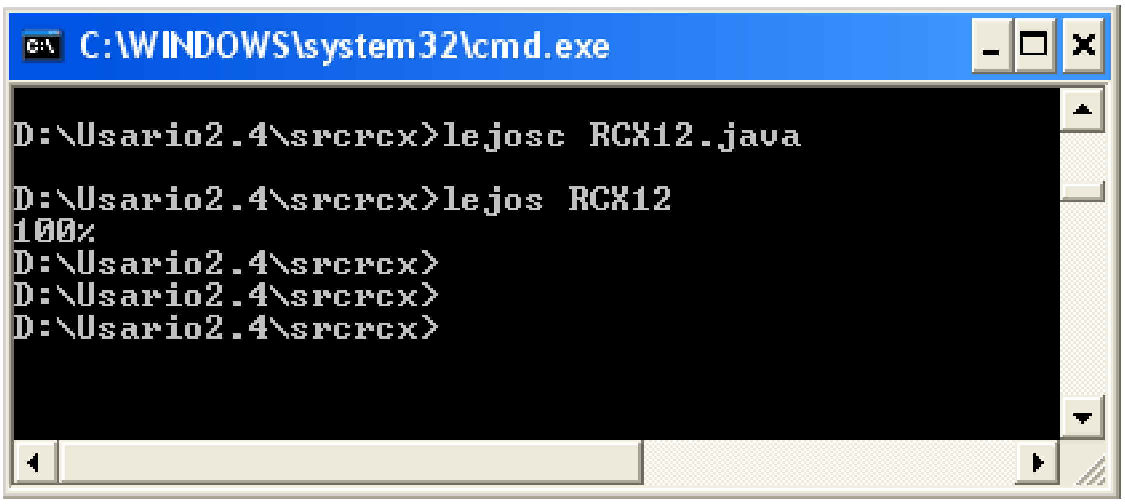


```
D:\Usario2.3>lejosfirmdl
Use --help for options.
segment.length[0] = 13686 (14k)
segment.length[1] = 2938 (3k)
Total image size=16624 (17k)
Transferring "/cygdrive/c/Lejos/bin/../bin/lejos.srec" to RCX...
100%

D:\Usario2.3>
```

Figure 5.1: Loading LeJOS onto the RCX

The RCX brick can store programs in its memory. However, it has been our experience that only 4 programs (storage # s 2-5) can be allocated to user-written programs. The appropriate classes necessary to run the robot are then loaded to the RCX, as shown in figure 5.2. The command "lejosc RCX12.java" compiles the Java class RCX12 in bytecode. The command "lejos RCX12" then triggers the download of the compiled RCX12.class to the RCX brick.



```
D:\Usario2.4\srcrcx>lejosc RCX12.java

D:\Usario2.4\srcrcx>lejos RCX12
100%
D:\Usario2.4\srcrcx>
D:\Usario2.4\srcrcx>
D:\Usario2.4\srcrcx>
```

Figure 5.2: Loading programs onto the RCX

2. **Infrared Proximity Detector (IRPD)**
<u>*Description:*</u> The IRPD sensor is an extra part of the family of Lego Mindstorms products which is used for measuring the intensity of light that enters the tiny lens on the front of the sensor. It can also be used to distinguish dark objects from light. The light sensor has a small red light-emitting diode (LED) that illuminates the scene in front of the sensor. It reads values from 0 to 100, and a 75 value meaning there is no object detected. The basic idea is to send a light towards an object and record the corresponding percentage (or raw value) of light reflected back.
<u>*Calibration:*</u> Results for a small test program for a robot to stop when facing a wall shows an average value of 0 units. Table 5.1 shows the average values in different situations. The IRPD sensor is initialized as 'LIGHT' and 'RAW' in the constructor of the class. The sensor detects objects which are within the 12-18 cm range, which means that we should set Usario to be navigating within this distance range from the walls and objects.

Value	Interpretation
75	No object detected
48	Object on right
22	Object on left
0	Object straight ahead

Table 5.1: Interpretation of the different readings from the IRPD sensor.

3. **Wheels**
   Usario is using two wheels, each attached to a motor on either side of the robot chassis. The wheels have non-pneumatic rubber tyres and have an external diameter of 8.16 cm.

4. **Motors**
   Description: Usario is using two standard Lego motors to power the wheels. Each motor (part number: 71427) is connected to a Lego shaft and a gear to transmit the power to the wheels. The main purpose of the motors are to turn the wheels.

   Calibration: The power of the motors can be adjusted from zero to seven, zero producing the weakest Torque. According to the Lego Mindstorms specifications [11], within the the internals of the motor, there are sixteen gears in the outside of a shaft which means that the motor provides rotations up to an accuracy of a sixteenth of one turn, i.e. 22.5 degrees. Moreover, while assembling the gears on Usario, we noticed a clearance in between the teeth of the gears when they are connected. This leads to a possible $\pm 22.5^0$ error margin while computing the rotation of the wheels.

5. **Serial IR port**
   The serial IR port is located at the top of the RCX. It provides InfraRed communication between the RCX and another device. The IR port sends and receives IR signals at a frequency of 38KHz. The communicating devices must be facing each other, with no obstacles in between as it would impair the signals transfer.

6. **Rotation sensors**
   The rotation sensor is an extra add-on part to the family of Lego products. It is connected to an input of the RCX and is designed to register the number of revolutions of rotating parts like motors and wheels. We are presently using two such rotation sensors, one connected to each wheel of Usario. As a wheel starts to rotate, a rotation sensor registers the number of rotations, but when the direction of rotation changes the number recorded is decremented. In other words, when the robot moves forward a positive value for the number of revolutions is recorded and

when the robot is going backward, that value is decremented accordingly.

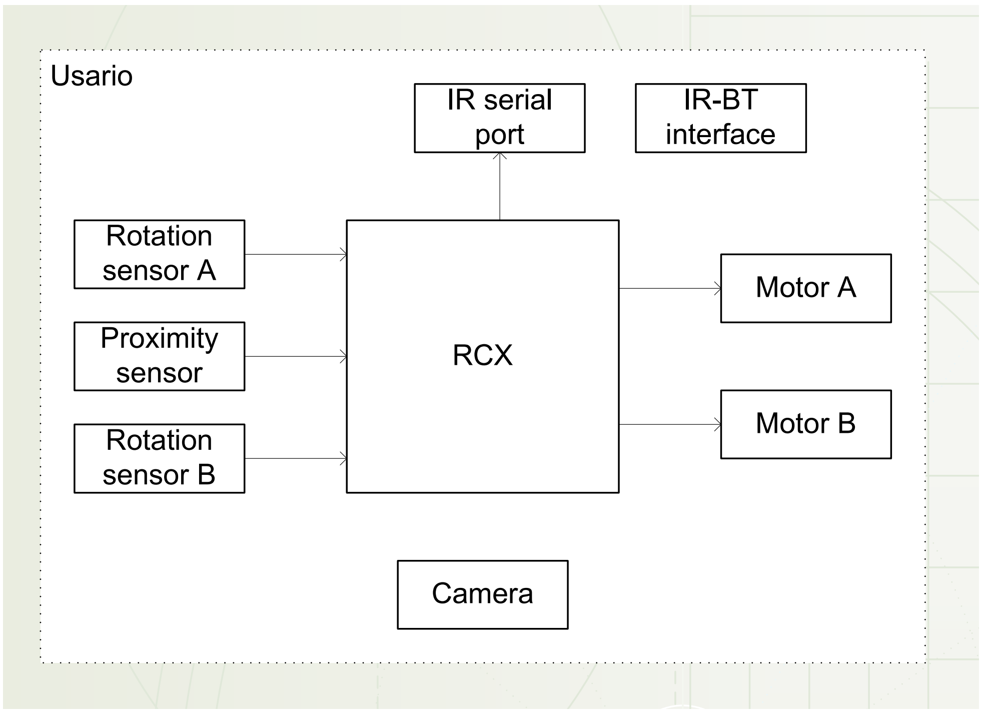


Figure 5.3: High level view of the nodes of Usario

These components are all mounted around the Lego chassis of Usario. The rotation sensors and the IR proximity sensor are the inputs to the RCX, while the outputs are the motors which in turn turn the wheels. The sensors and motors are connected to the RCX through standard Lego cables, while the IR serial port is built in the RCX. Figure 5.3 shows how the physical components of Usario are connected.

5.2.2 Camera

The image and sound capturing device for Usario is a *DCS-5300G Internet camera*, which is a model from D-Link in the wireless internet camera range [9]. It is a digital surveillance system that connects to an Ethernet or wireless broadband network to provide remote video and audio. Using the 802.11g wireless technology, the camera has an built-in webserver and communicates at a maximum wireless signal of up to 54Mbps. Access and control of the DCS-5300G is made using the Internet Explorer browser, telnet and FTP.

5.2.3 InfraRed-Bluetooth interface

Given the present state of the RCX, communication between the RCX and the PC can be very troublesome as the RCX's serial port needs to be placed in a straight line with the IR tower. This is somewhat difficult as the IR tower will have to be mobile, and in addition, the walls of the arena might be obstructing the IR beam. This is not a convenient communication system for Usario as it needs to be mobile and wireless. The next version of Lego Mindstorms to be released in August 2006 does however provide an RCX brick which communicates via Bluetooth. This would largely solve our problem. For this work, we had to develop a customized InfraRed-Bluetooth interface.

The InfraRed-Bluetooth interface developed is a tailor-made transceiver built for this Lego Mindstorm USAR prototype by NTNU PhD candidate Pavel Petrovic. It is a device that enables transmission and reception of data between an RCX and a computing device through a combination of Infra Red and Bluetooth radio protocols. The heart of the interface is a BlueSmirf module that transmits and receives bluetooth signals to and from the bluetooth-enabled PC. The BlueSmirf is connected to a circuit which encapsulates an IR transceiver and which relays the converted IR signal from BlueSmirf to the RCX IR serial port, and back. At the same time the interface transforms the IR signal emitted by the RCX's serial port into a TTL level input further into the BlueSmirf module, which transmits it further over the Bluetooth radio protocol. The communication is half-duplex, robust, reliable and fully adequate for the context of this work. The interface is placed on top of the RCX facing the IR port and consists of three parts:

1. **BlueSmirf chip.** The BlueSmirf module (see [36]) is a 'black box' wireless serial link operating at a frequency of 2.4GHz. The link is composed of two units; a Bluetooth dongle (see 5.7) and a BlueSmirf remote as shown in Figure 5.4. The base unit is a Bluetooth USB dongle5.7 that one attaches to the Windows computer via the USB port. The other unit is a small and unpackaged device that can be powered from 3 to 6V for battery attachment and supports a maximum of eight connections from Bluetooth enabled devices. The device has a built-in antenna and the link can handle full-duplex data rates of up to 115200bps. However, for this project, it has been configured to transfer data at 2400bps in a half-duplex mode. The BlueSmirf firmware buffers incoming and outgoing data while providing for full error checking and packet delivery quarantee.

2. **IR transceiver.** The IR transceiver communicates with the RCX through the serial IR port. It is transmitting analogue signals to the RCX through an IR diode placed next to the RCX port. On the receiving end there is an InfraRed receiver which collects all data sent by the RCX. The analogue signals are emitted and received at the RCX's standard frequency of 38KHz.

3. **IRBT circuit.** The InfraRed-Bluetooth circuit is the device that converts the Bluetooth signals to IR signals and vice versa. On one end, it is exchanging

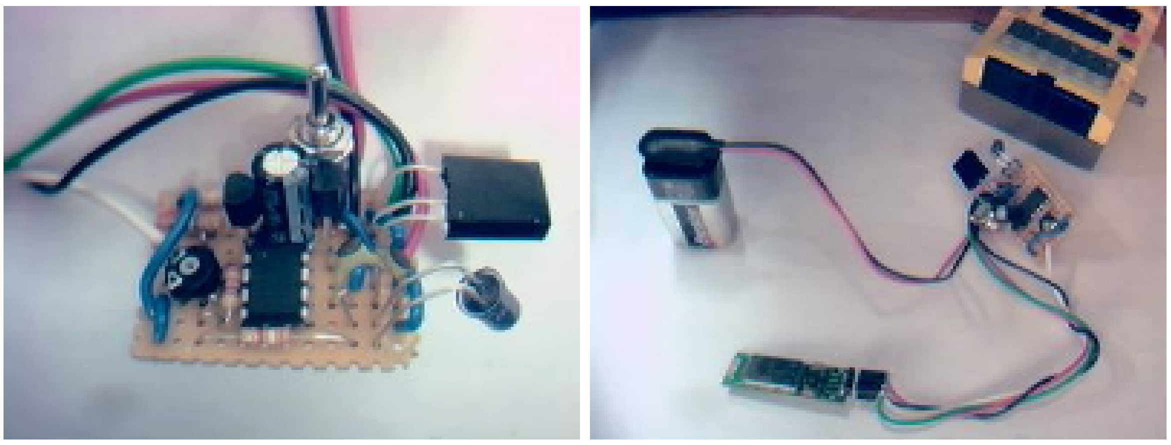


Figure 5.5: The IR transceiver embedded in the InfraRed/Bluetooth circuit.

data using Bluetooth technology with the BlueSmirf chip. On the other end, the InfraRed-Bluetooth circuit is communicating with the RCX through the IR transceiver. Figure 5.5 show the IR transceiver embedded in the InfraRed-Bluetooth circuit.

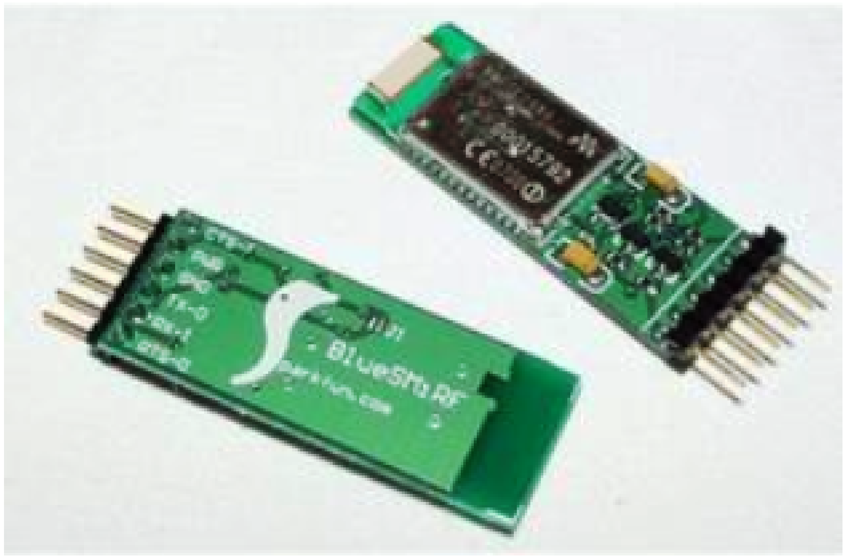


Figure 5.4: The BlueSMiRF V1 module

5.2.4 Search arena

To facilitate testing of Usario, we built a search arena. The arena is a custom made field that simulates a floor of a building. It consists of a 1.40 × 1.60 metre wooden frame with a white cardboard floor. The floor, inner and outer walls are all painted white to allow maximum reflection of light from the proximity sensor. Figure 5.6 shows the search arena.

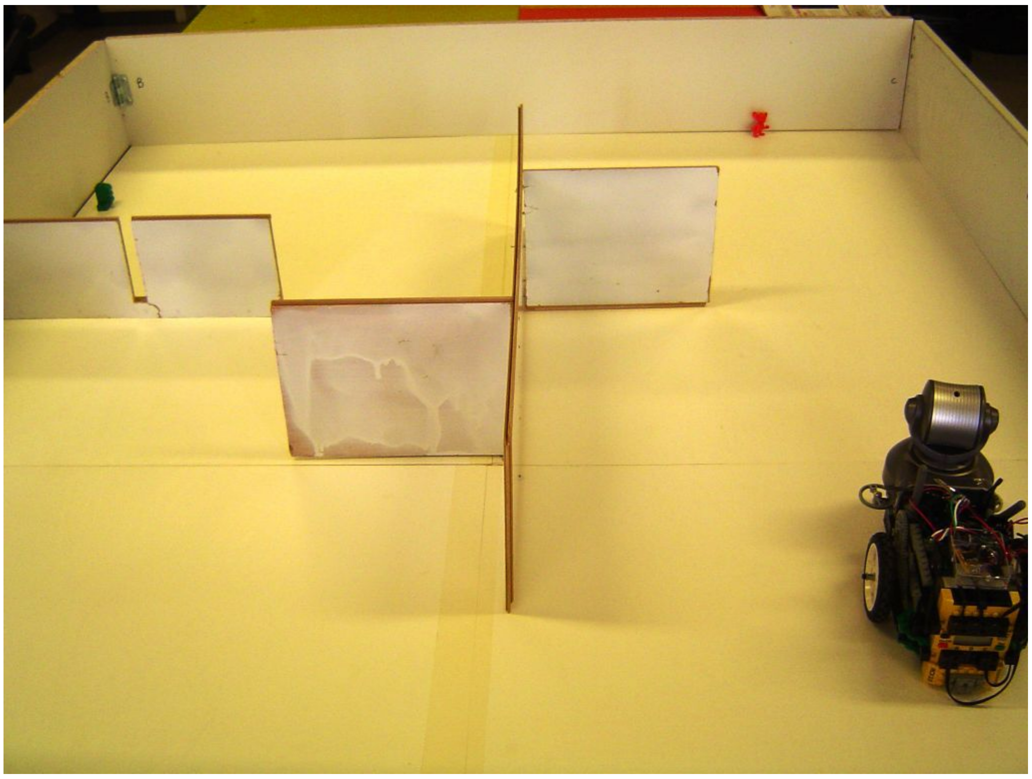


Figure 5.6: The floor model

5.2.5 PC server

The PC unit used is a notebook computer running at a clock speed of 2.0 GHz and with Microsoft Windows XP as operating system. It contains the GUI program, the communication programs and has appropriate sound and graphics card. The PC unit has Java SDE 1.4 and Java Virtual Machine installed. Furthermore, it is connected to the internet and is a Bluetooth-enabled device.

5.2.6 Bluetooth dongle

We used the Trendnet TBW-102UB Bluetooth dongle to exchange data with Usario. Being compliant with USB 1.1 and Bluetooth 1.1 specifications. The dongle is connected via USB to the PC unit to communicate using Bluetooth protocol. It transmits and receives data at a 2.4GHz radio frequency band and has, according to the producer, a maximum data transfer rate of 723Kbps is ensured within a connection range of up to 100 meters. Figure 5.7 shows the Bluetooth dongle.

Figure 5.7: The Bluetooth dongle

5.2.7 Power supply

Requirement F8.1 (see 3.1) of the robotic system stipulates that it has to be free from power cables. It should be powered with battery so as to avoid the danger of the robot being entangled with the wires while it is moving around. The RCX itself is designed to accomodate six 1.5V DC batteries which also provides the power required to run the sensors, motors and IR port. However, the camera (see 5.2.2) does not come equipped with battery facilities and adjustments had to be made for the purpose of this experiment. The camera requires a 12V power supply [10] and which has been achieved by a 10 x 1.2V DC battery pack. In addition, the IR-bluetooth interface requires a power supply of a 9V DC battery. Therefore, the robot is completely battery powered with a total of 16 AA batteries and a 9V LR-61 flat battery.

There are many factors affecting the performance and accuracy of the robot during its mission. A few of these factors are the battery power, physical robot size, lighting conditions, the wifi network and interference in the Bluetooth communication. Therefore, in order to enable Usario fulfilling its mission, the robot needs to be recalibrated each time one of the above factor changes. This is especially important when it comes to multiple runs in the implementation phase, as battery power was found to decrease after every successive run. When we discovered that the robot was not behaving as expected, the batteries were checked and changed/recharged whenever they were found to be below a certain voltage level. Low battery power has also been the cause of problems like truncated or loss of bits in data transfer, inexistant/poor connection, loss of connection, poor quality or no display of images and a drastic reduction in the performance of the RCX. Table 5.2 summarises the minimum and maximum capacity voltage of the powered units of Usario. The minimum voltage is our experienced minimum voltage required to run Usario without any consequent drop in performance, and the maximum voltage is the maximum Voltage when the batteries are fully charged.

Unit	Minimum (Volt)	Maximum (Volt)
RCX	7.6	9.0
Camera	10.0	12.0
IR-BT interface	7.8	9.0

Table 5.2: Maximum/minimum voltage capacity for the powered units.

Chapter 6

Navigation

This chapter describes the techniques and principles behind the robot's manoeuvres. It introduces the concept of dead reckoning in navigation and presents how it has been implemented to satisfy functional requirement F1 Navigation (see section 3.1) for Usario.

6.1 The concept

Early sailors before the 16th century navigated using a method called deduced reckoning or dead reckoning [23]. This type of reasoning helped navigators to find their position by measuring the course and the distance travelled from a known point. For some time now, this method has been adopted for robot navigation. Knowing the starting coordinates, the angle and distance travelled by the robot, one can deduce the current position and destination coordinates as shown in Figure 6.1.

The LeJOS API provides some basic functions to control the motors and read the values of the sensors. The functions are summarised and described in Table 6.1. These

Function	Description
forward	moves motor in a forward direction
backward	moves motor in a backward direction
stop	stops and locks(brakes) the motor
float	stops but does not lock the motor
readRawValue	reads the canonical value of the sensor

Table 6.1: Basic functions in LeJOS.

functions enable basic movements of the robot and a function for reading a sensor. The accuracy of movements for Usario is based on these functions.

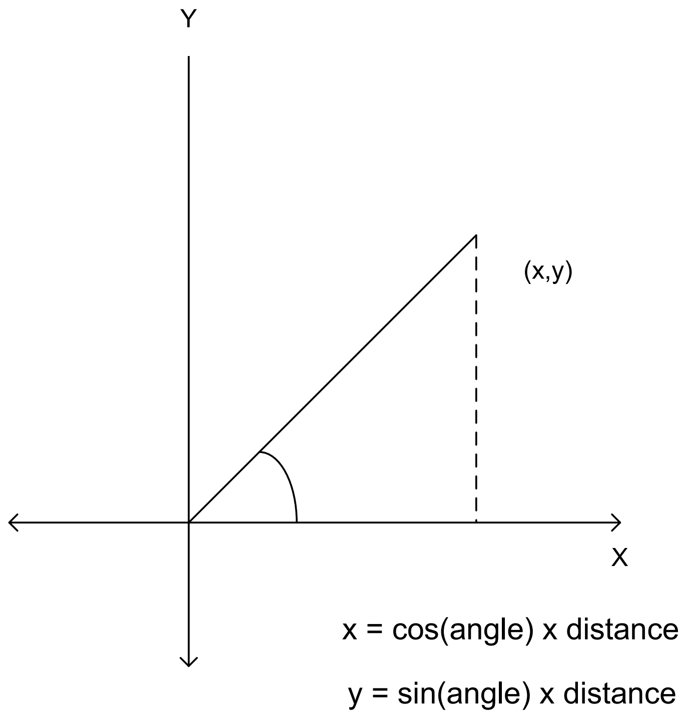


Figure 6.1: Dead Reckoning

6.2 Navigator API

The LeJOS API has a built-in class for navigation called ‘Navigator’ and which is based on dead reckoning navigation. This class caters for robots with differential steering[3] [23]. It provides methods for moving a robot over a particular distance and rotating at a certain angle. The robots current (x, y) coordinates and angle are maintained by the Navigator object, and updated after every stop movement. LeJOS provides two classes that use this interface. The difference between them is the way in which the angle and distance, needed for dead reckoning navigation, are measured. The first one is called ‘TimingNavigator’ and measures the movement in terms of the number of seconds travelled. When instantiating this navigator class, it should be provided with time taken by the robot to move 100 cm and to complete one full rotation. Once instantiated, the class computes the number of seconds it should allow the motors to run before coming to a stop. A second and more accurate class is the ‘RotationNavigator’ class which records the movements of the robot, over a particular distance, in terms of the number of rotations of the robots wheels. A rotation sensor is connected to each of the wheels’ axle. Both classes require that the

[3] *Differential steering is a wheel-based propulsion system commonly used in small robots with two wheels mounted on a single axis. The two wheels are independently powered and controlled, thus providing both drive and steering functions*

robot used is two-wheeled and that the rotation sensors record a positive value when the motors are moving forward. However, while extending these classes to accomodate the needs of the project, we discovered that one had a better control over the sensor and constant values by implementing our own Navigator class which is described further below.

Dead reckoning is less expensive compared to other navigational systems and the computation involved is based on simple calculus and trigonometry. To move the robot forward, both motors are powered in a forward direction. The opposite in the case of moving backward. From simple mathematics calculation, we know that one wheel moves a distance of

$$\pi \times D, \textit{(D being the diameter of a wheel of the robot)}$$

for one wheel revolution. A rotation of Usario, being a two wheeled robot, involves both wheels running at the same time, but in an opposite direction. This rotation about its original axis can be regulated by rotating the wheels for some distance in order to rotate for a certain specific angle. In addition, since Usario is equipped with two rotation sensors, one on each wheel and has a gear ratio of 1:1, we are able to trace and monitor its movement. However, this only holds for a flat and uniform floor surface. Therefore, based on the basic LeJOS functions of Section 6.1 our Navigator API is as follows:

```
**moves the robot forward until stop is called
public void forward(){
Motor1.forward();
Motor2.forward();
}
**moves the robot backward until stop is called
public void backward(){
Motor1.backward();
Motor2.backward();
}
**stops the robot and updates the X and Y coordinates
public void stop() {
Motor1.stop();
Motor2.stop();
}
**moves the robot for the given distance; a positive value indicates a forward
movement and negative value indicates a backward movement
public void travel(int distance) {
int requiredNumberOfRevolutions = $\frac{distance}{2\times\pi\times radiusofwheel} \times 16$;
while(RotationSensors != requiredNumberOfRevolutions) {
if (distance > 0) go forward();
else if (distance < 0) go backward();
} }
```

```
**rotates the robot for the given angle; a positive angle indicates a clockwise rotation and a negative angle indicates a counterclockwise rotation
public void rotate(float angle) {
int distanceToTravel = $\frac{1}{2}\times$ axle length $\times$ angle;
if (angle $>$ 0) then {
Motor1.travel(distanceToTravel);
Motor2.travel(-distanceToTravel);
}
else {
Motor1.travel(-distanceToTravel);
Motor2.travel(distanceToTravel);
}
}
**rotates the robot to face the destination and moves it through the required distance to reach the target point.
public void gotoPoint(float x, float y) {
float angleToRotate = $\tan^{-1}(\frac{y}{x})$;
int distanceToTravel = $\sqrt{x^2 + y^2}$;
rotate(angleToRotate);
travel(distanceToTravel);
}
**returns the current X coordinate of the robot
public float getX() {
return this.Xcoordinate;
} **returns the current Y coordinate of the robot
public float getY() {
return this.Ycoordinate;
} **returns the current angle of the robot
public float getAngle() {
return this.angle;
}
```

For every rotation of 22.5 degrees of the motor shaft, the RCX increments the rotation count by 1. That is, for every full rotation of the wheel, a count of 16 should be recorded by the rotation sensor. Turning the wheels in the reverse direction decrements the count in the same manner as described. The RCX can count up to a speed of 600 rotations per minute. The instance of this navigator is provided with the wheel diameter and the drive length of the axle. The drive length is the distance between the two wheels. When a wheel has completed one complete turn, the robot will have travelled a distance equal to the circumference of that wheel. Knowing the wheel diameter and hence the circumference, distance to be travelled can be broken down into number of rotations of the wheel. Sixteen times this is the count up to which the RCX has to measure before it can stop the motors.

When the robot makes a full 360 degrees turn to the left, the left wheel rotates backward and the right wheel rotates forward for a distance equal to the circumference of a circle whose diameter is equal to the drive length. When the robot has to turn a lesser angle, the wheels will rotate for a fraction of this circumference.

6.3 Navigation and wall detection with collision avoidance

Usario basically moves around in the arena by activating the motors on forward mode. The robot stops when the proximity sensor detects an obstacle object on its path. The robot then rotates about itself[4] for 360 degrees at intervals of 45 degrees and thus obtains values from the sensor at direction angles of 0^0, 90^0 and 270^0. By experimentation, the sensor has been found to read a value of 0 when it is perpendicular to a wall within a distance range of 12-18 cm. Therefore by knowing the values of reflected light to the left and right of the robot, we are able to deduce where the side walls (if any) are in relation to the robot. Our algorithm arbitrarily sets the course of the robot to the free side and in cases where both sides do not have any wall, the course is arbitrarily set to be one of the sides. To avoid the robot ending up in a deadlock situation, the previous arbitrary decision is remembered and every time there is a branching decision to make, the robot is set to move in the direction opposite to the previous. Three possible locations for the walls gives eight possibilities. This decision process is summarised in Table 6.2. The pseudo code is listed in Subsection 6.3.1, while the movie in Figure 6.2 demonstrates the navigation module and algorithm.

Wall at	previous decision	Decision
Front	Left	Right
Front	Right	Left
Front and Left		Right
Front and Right		Left

Table 6.2: Deciding for a direction

6.3.1 Pseudo-code

The pseudo-code for the navigation module is as follows:

[4] *Two rotations to the left and the right would have sufficed to determine the presence of side walls, but since we are looking around for victims, a 360^0 scan is a more reasonable alternatice.*

```
'do the following 8 times:' {
        'At each interval record the sensor value'
        'rotate 45 degrees anticlockwise'
}
if 'wall only in forward direction'
        if 'the PreviousDirection is RIGHT' {
        'turn robot left'
        }
else {
        'turn right'
        }
else if 'wall is ahead and to the left' {
        'turn robot right';
        }
else if 'wall is ahead and to the right' {
        'turn robot left';
        }
else {do nothing but moving forward if wall combinations are
        different from those above }
Keep robot moving forward until
sensor reads a value of > 700;
stop robot;
```

6.3.2 Video demonstrating the navigation module

Figure 6.2 is a video file showing the robot model in action. The video shows Usario navigating around in the arena, when viewed from the top. The video is in 'avi' format and requires at least a version 9.0 of Windows Movie player to be displayed properly.

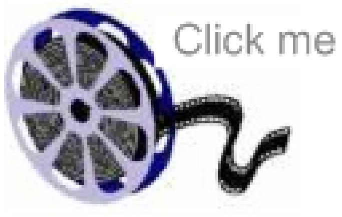


Figure 6.2: Video file showing the navigation module (requires at least a Windows Movie player Version 9)

Chapter 7

Communication

To satisfy the system requirement F5 - Report (Section 3.1) and F6 - Communication (Section 3.1), a communication mechanism has to be established to allow reporting of data. Data captured by the robot inputs can be processed locally by the robot's own processor before being sent to a remote server for further processing and presentation. Two categories of data are distinguishable: image/audio and other input data. They will however have to be sent through different communication channels, as we do not want to congest the communication channel and would like to have a backup alternative to the data stored on the robot itself. Therefore, our communication module is split into two parts; 1. Communcation from the camera unit to the PC class, and 2. Communication between PC and RCX . These are described in the following sections.

7.1 Camera to PC

The camera to PC module is a permanent communication channel which transfers data to the server. This communication module uses wireless internet as transmission protocol. It is a one way data transfer where images and sound from the scene as captured by the camera, are sent to the camera's own web server. To transfer sound and images, the Dlink DCS5300G camera is connected to the faculty's wireless access point. Access and control of the DCS5300G are made using the Internet Explorer version 6 browser, telnet and FTP. However, in order to avoid the fuss of reconfiguring the system each time the robot is switched on, the camera needs to be assigned a static IP address. Hence the camera runs TCP/IP on the faculty's wireless LAN with the assigned IP address 229.241.102.54[5] . The settings for the control unit are configured as shown in table 7.1.

[5] *This is a fictive IP address*

Item	Description	Default setting
Hostname	The name of DCS5300G to be used for wireless LAN communication	DCS5300G
IP address	IP address	229.241.102.54[5]
Subnet mask	Subnet mask	255.255.255.0
IP Gateway	Gateway address	229.241.102.1[5]
SSID	The name of the Wireless LAN network to be used	IDIlab
WEP key	The character string to be used as the key for encrypting data transmission over the wireless LAN	toysrus
Operating mode	Infrastructure mode or ad hoc demo mode	ad hoc
wireless channel	channel between 1 and 11	1

Table 7.1: Internet camera network settings

7.2 PC to Usario and Usario to PC

The communcation between the PC and Usario is implemented as a synchronous half-duplex communication module. The module is a half duplex communication class, as data can be transmitted over both directions but does not offer simultaneous transfer of data. Only one side can initiate data transfer at a time and the receiving end has to wait for this transfer to be completed before it can start sending information. This half duplex communication transfer is done through the IR-Bluetooth transceiver (Section 5.2.3) by a combination of Infra Red beams and Bluetooth signals. Once the BlueSmirf module is paired with the Bluetooth dongle on the server and the GUI is launched, a permanent and synchronous communication exists between the two of them. Data is sent from PC to the BlueSmirf unit, which in turn relays it to the RCX via the IR LED. Figure 7.1 illustrates the communication process between the PC and Usario. The PC generates command data which is sent to the RCX to be executed, while Usario returns information about its whereabouts to the PC.

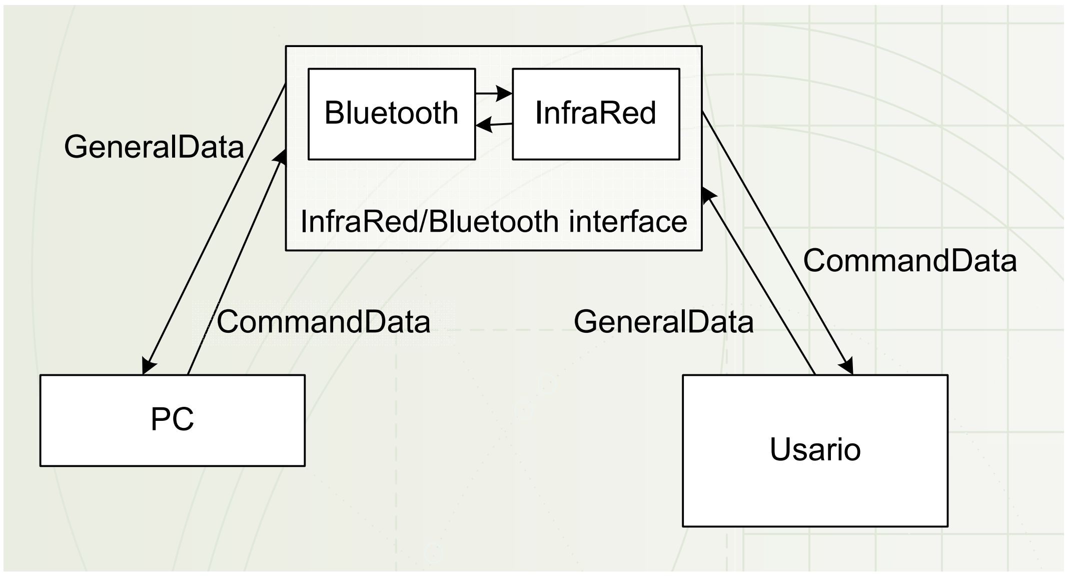


Figure 7.1: Data communication between PC and Usario

Figure 7.2 shows a UML 2.0 sequence diagram depicting communication between the server, represented by the PC, and Usario characterised by the RCX. On the PC side, the GUI initiates the communication by sending a command to Usario. The commPC thread encapsulates the 8-bit command data with the necessary headers (3 bits) and transmits the datapacket to the corresponding thread of communication in the RCX of Usario, commRCX. The packet only contains the action to be performed by Usario.

On receipt of the datapacket, commRCX removes the header and trailer bits and transmits the data to the RCX. The data command received by the RCX is then executed by Usario and the relevant attributes of the RCX are updated. The PC is made aware of these changes at the receipt of the next datapacket. The coordinates of the robot are generated by the functions updateX and updateY from the RCX.java class. At the same time of receipt of the packet from the PC, commRCX sends back a message to commPC which contains four bytes of information: 1. coordinates information of the robot (x,y), 2. the angle at which the robot is facing, 3. the command request by the PC and 4. the command being executed. The communication is synchronous as the PC starts the transfer process and for every message sent, there is a corresponding reply to the message. The PC and RCX classes do not explicitly send messages; they rather update their variables and the two threads commPC and commRCX send messages(represented by the recursive dotted arrowhead line) with the updated data without even knowing about updates.

The commPC and commRCX threads do not wait for new data, but keep on continously exchanging data packets (represented by a recursive solid arrowhead line) which are limited to 5 bytes. The communication reliability is somewhat compromised by the inherent unreliability of the IR communication implementation in the LeJos system environment. However, provided that the voltage level of the batteries is sufficient enough, the packets are delivered reliably and in correct order. Longer packets can be sent as well and they are automatically divided into multiple smaller packets due to the limitation of the buffer size for IR communication in the LeJos system environment that optimistically relies on the RCX ROM routines. Section 10.1 and the technical report [27] elaborate more on this issue.

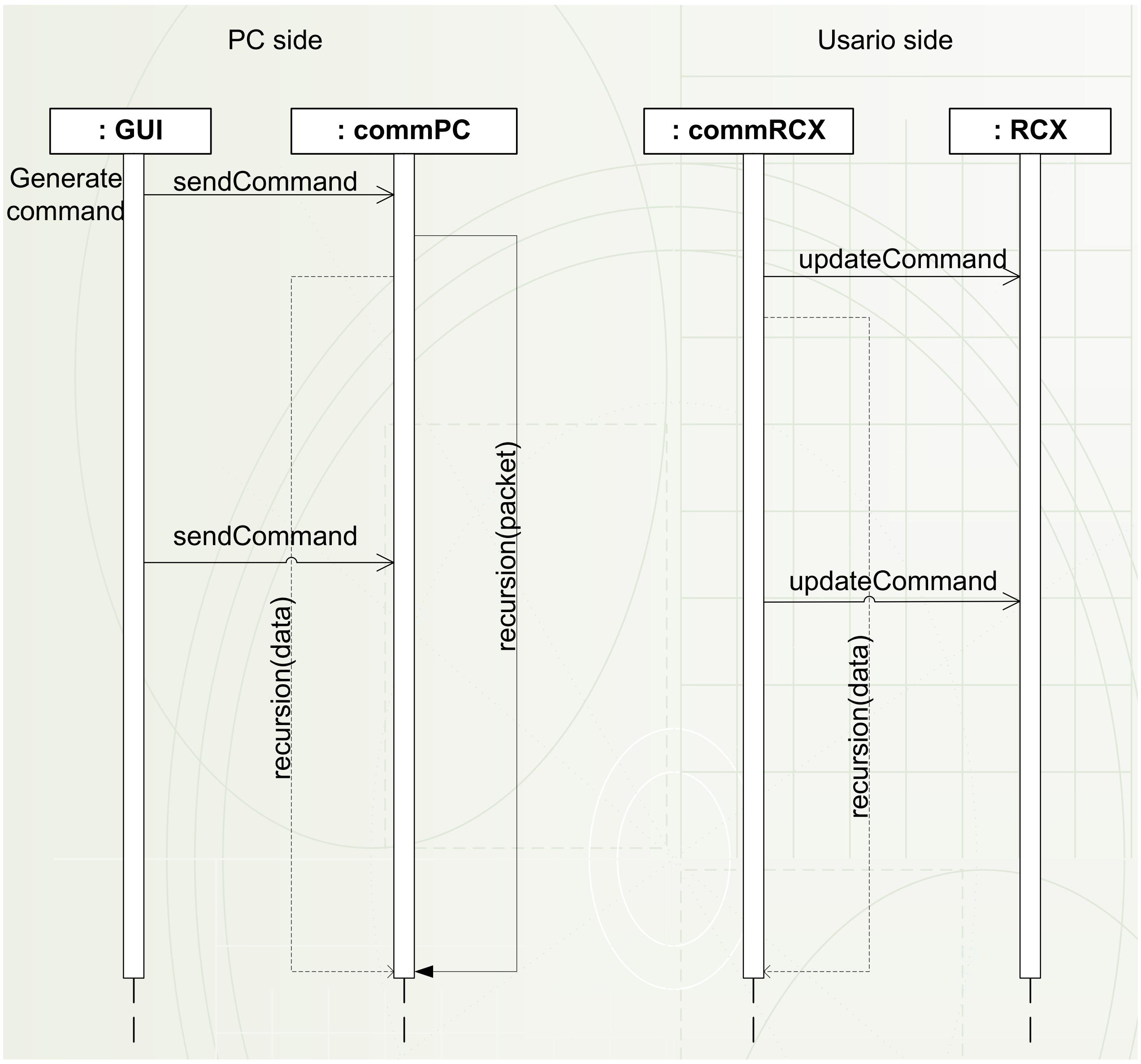


Figure 7.2: Sequence diagram for communication between PC and Usario in UML 2.0 notation

Chapter 8

Object detection, identification and localization

As Usario moves around the arena, it has to detect victims by a coordination of its movements and the camera placed on its top. The robot has the ability to move and rotate about itself while the camera does not; it remains fixed and it only captures images straight ahead of the robot. This chapter describes the concepts used to detect, identify and localize the victims.

8.1 Camera image

Recall that movies are a succession of pictures at a speed higher than what the human eye movement can detect. Also relevant to our discussion is the process of displaying images; any image capturing device has to, at some point in time, store the sequence of images somewhere. The pictures will then be displayed in a browser at a frame rate set by the browser itself and depending on the speed of data transfer. The main idea is to grab the images captured by the camera inside its own webserver. In this implementation, when the camera is activated, it starts capturing images at a rate of 30 frames per second and stores these as pictures in its webserver. Knowing that these pictures are available at the address 'http://229.241.102.54/picture.jpg' in a browser, we somehow needed an application to access this site and copy the pictures from there. We used Telnet (a terminal program and protocol) to trigger the digital output of the camera and detect changes on the digital input of the camera when we were communicating with the RCX through the special cable attached to the camera [12].

Our image capturing device is the Dlink DCS5300G network camera (section 5.2.2). It is running a built-in webserver, which makes the video viewable in a web-browser with a proprietary ActiveX plugin. However, according to the manufacturer, there are two ways to obtain the frames. The first way is to grab the frames from the web-browser screen and the second option is to retrieve the frames stored in jpeg format directly from the camera using ftp protocol - this is what we were doing earlier in [12]. However, the problem with this approach is that the update rate of these captured still images is 1 second, which is not sufficient for a mobile robot application. The only faster method according to the manufacturer is to grab the images directly from the browser's window. In agreement with PhD candidate Pavel Petrovic, we adpoted this recommendation and developed a dynamic loadable library (DLL) in C++ that starts a separate network browser window showing the camera view image, and grabs the frames from its window graphics area using the PrintWindow() Windows API function while running in the background. The main Java GUI application is communicating with the DLL using the provided API based on JNI. The DLL uses the OpenCV library to visualize the grabbed images in a separate window that is placed on top of the Java's GUI. The grabbed images are further processed using the Intel's OpenCV image processing library [35].

8.2 Object detection

8.2.1 Objective

The objective of object detection is to satisfy the functional requirements F2.1 - Detect objects (section 3.1) and F2.2 - Detect humans (section 3.1) , which gives the owner of the accident more detailed information about the existence of victims.

8.2.2 The concept

The basic idea to detect objects is to identify the coloured pixels from each individual picture generated. The OpenCV library displays the frames that are grabbed from the browser window in a separate GUI window. Since the victims have been preassigned red or green colours, groups of pixels that contain a specific number of coloured pixels can thus be declared as objects whenever present. More formally, we test each pixel's color values in the RGB color model with red and green color predicates defined as follows:

$red(r, g, b), iff((r > RED\_C1 * b) \cap (r > RED\_C2 * g)) \cup ((r > RED\_C3) \cap (r > RED\_C4 * b) \cap (r > RED\_C5 * g))$

where r, g, b are the red, green, and blue color values of the pixel, and $RED\_C1 \ldots RED\_C5$ are the detection parameters tuned to fit with the actual light conditions and required sensitivity. In our application, we use the following configuration: ($RED\_C1 = 2.0$, $RED\_C2 = 2.0$, $RED\_C3 = 140$, $RED\_C4 = 1.4$, $RED\_C5 = 1.4$). In other words, the pixel is considered red, when its red color component is much higher than the green and blue component, or alternately, when its red component is slightly higher than the green and blue components, and at the same time, the red component is above certain threshold. When compared to other systems (for example AIBO's built-in color detection, which defines component value intervals for 32 different intensity values),

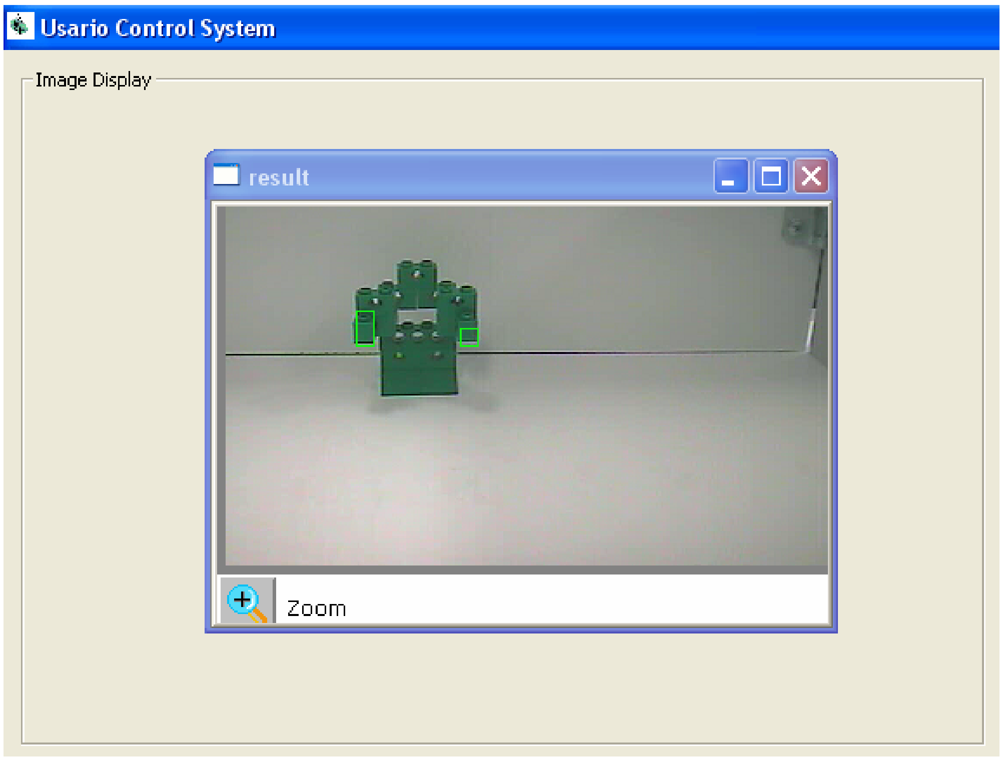


Figure 8.1: Survivor being detected

our model is farily simple. As a consequence, it is simpler to configure and while we are only detecting the basic colors: red and green, we find it strong enough for our purposes.

Green color pixels are detected analogically using the following predicate:

$green(r, g, b), iff((g > GREEN\_C1 * b) \cap (g > GREEN\_C2 * r)) \cup ((g > GREEN\_C3) \cap (g > GREEN\_C4 * b) \cap (g > GREEN\_C5 * r))$.

Once the pixels of the image frame are assigned green and red labels, we consider the pixel rasters - neighbourhoods of the size $N \times N$ (i.e. $N = 10$). If at least δ (i.e. $\delta = 33$)pixels in the neighbourhood are green or red, we label the whole group with the respective color. Consequently, we consider the connected components consisting of rasters labeled with the same color as the candidates for the detected objects. This method allows for tolerance of local errors, holes, reflections, and other noise contained in the object and estimates the borders of the detected objects robustly. Detected victims are shown on the camera image with a surrounding rectangle as shown in figure 8.1.

8.2.3 Pseudo-code for camera

The following java code section represents the implementation of the image grabbing from the browser window and the setting up of parameters for the detection of victims.

Step 1 : create object of Petrovic's Detect function

Step 2 : setup the path to the network browser (such as ”c:
program files
Internet Explorer
iexplore.exe) and the local file that contains reference to the camera image view (such as ”m:
test.html”) as well as the expected title of the browser's window from which the frames are grabbed (such as ”X - Microsoft Internet Explorer”).

Step 3 : set up 4 parameters for object detection in Detect().

parameter 1 - the maximum allowable objects to be detected in one picture is set to 400.

parameter 2 - the length of the raster is set to 10 and since it is a square, the raster is of size 100 pixels.

parameter 3 - the required percentage of color pixels in the raster is set to 33%.

parameter 4 - this parameter sets the minimum number of rasters required to be detected as an object.

Step 4 : set the parameters for the green and red color detection as described in the previous section.

8.2.4 Pseudo-code for Usario

Since the camera is fixed in front of the robot, Usario has to rotate about itself to have a complete view of its surroundings and detect victims. The rotation is divided into eight intervals of 45 degrees, and the objects found in the camera view will be processed by the server. This rotation is executed whenever 1) Usario stops and 2) after Usario has travelled a distance of 50 cm in a straight line. The pseudocode for the object detection module in Usario is:

```
if (robot is stopped OR distance travelled = 50)
rotate 360deg at 45 deg interval.
```

8.3 Object identification

Two colours of the RGB scale, red and green, are used to identify the state of the victim. Red represents a dead victim and green, an alive victim in accordance with requirement F3.1 - Determine the state of the victims(section 3.1) .

8.3.1 Objective

The objective of object identification is to satisfy functional requirement F3.1 which provides more detailed information about the state of the victims to the owner of the accident.

8.3.2 The concept

Red and Green objects, being native colours from the RGB colour scheme, are detected using the proprietary algorithm for simple colour segmentation of the image in detect.dll. In other words, when objects are detected, the RGB values of objects are checked and compared and the corresponding colours will indicate the status of the victims.

8.3.3 Pseudo-code

The following code fragment from Camera.java describes how the parameters (for the colour detection scheme of Detect.dll) are set, based on the concept in section 8.2.

```
{....
line 20. - //relative1B, relative1G, otherwise required R, relative2B,
// relative2G
line 24. - d.redparam((float)2.0, (float)2.0, (float)140.0, (float)1.4,
//(float)1.4);
. - //setup green colour detection:
line 26. - //relative1B, relative1R, otherwise required G, relative2B,
//relative2R
line 28. - d.greenparam((float)2.0, (float)2.0, (float)100.0, (float)1.0,
// (float)1.5);
. - ......}
```

8.4 Object localization

8.4.1 Objective

The objective of object localization is to satisfy functional requirement F4.28 (Section 3.1). It gives the owner of the accident more detailed information about the position of the victims to be rescued.

8.4.2 The concept

Since there is no practical way to determine the angle at which the camera is facing, we decided to keep the camera fixed and instead rotate the robot to determine the angle of an object in relation to the the robot itself. Only objects that lie in the y-axis (i.e. at line x=0) of the camera will be considered relevant. The position of the robot and the angle at which the robot is facing being known at any time, we can deduce the location of the detected object. However, the distance from the camera to the object is not known. To counteract this, we use the idea that when the same object is viewed from two known positions, one can deduce the object's position. Figure 8.2 illustrates this. As Usario detects a victim, a coloured line corresponding to the colour of the victim is drawn at the angle Usario is pointing to. Here the angle is 315^0 and the the victim is alive, hence a green line is drawn at an angle of 315^0. At every scan of the robot, the angle at which it is pointing to, is sent to the server. Meanwhile, the server is processing the image from the camera's webserver to detect objects. On detection (and identification) of the object, the current angle is recorded as this information is always included in the packets sent by the RCX. On receipt of the information, the position of the victim is then deduced to be at the intersection of two lines of the same colour and displayed in the GUI (section 9.1). Figure 8.3 shows the position of a cadavre being localized at the intersection of two red lines. The intersection of the two green lines represents the position of a survivor, detected at a previous stage.

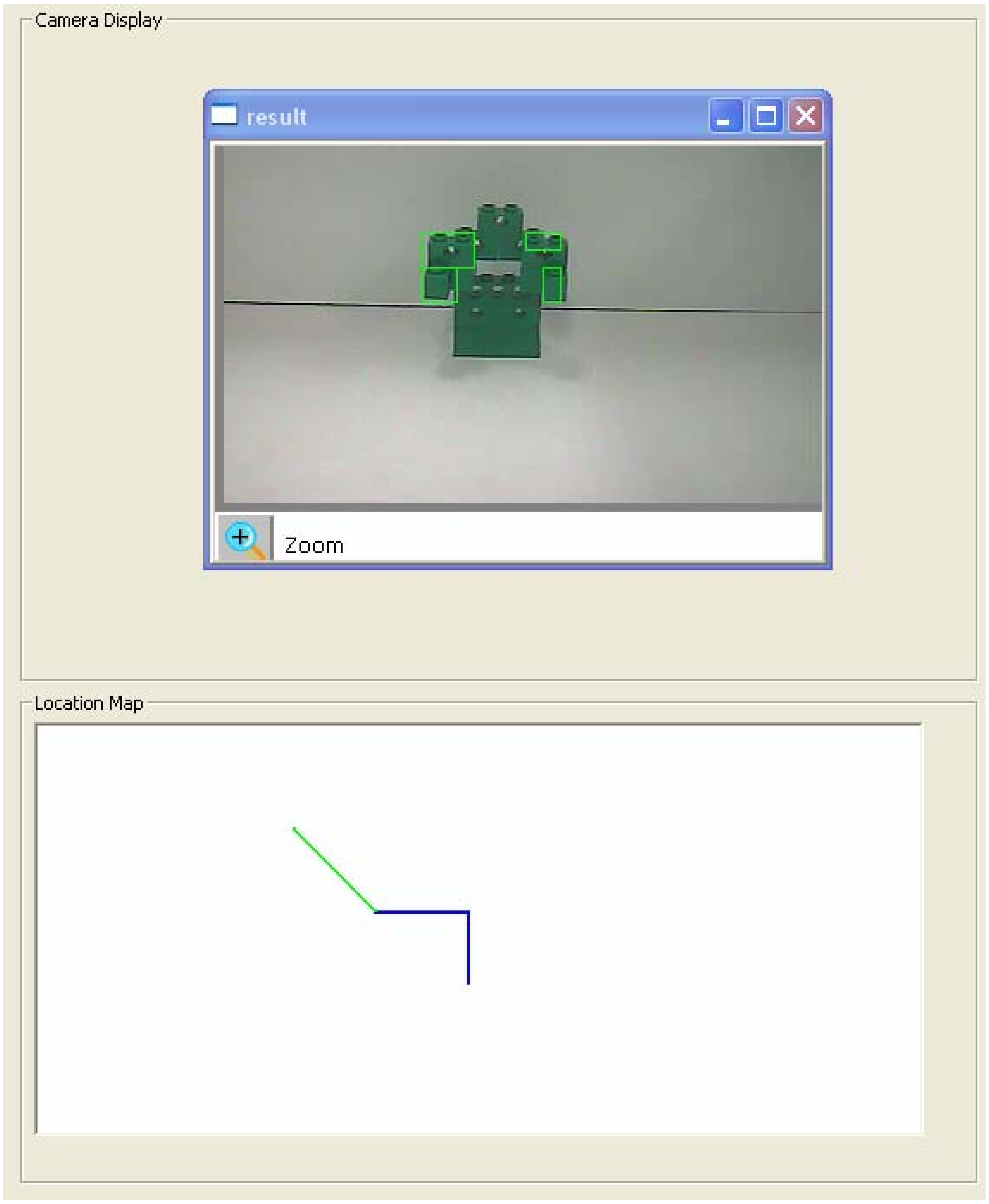


Figure 8.2: Illustration of a survivor being localized

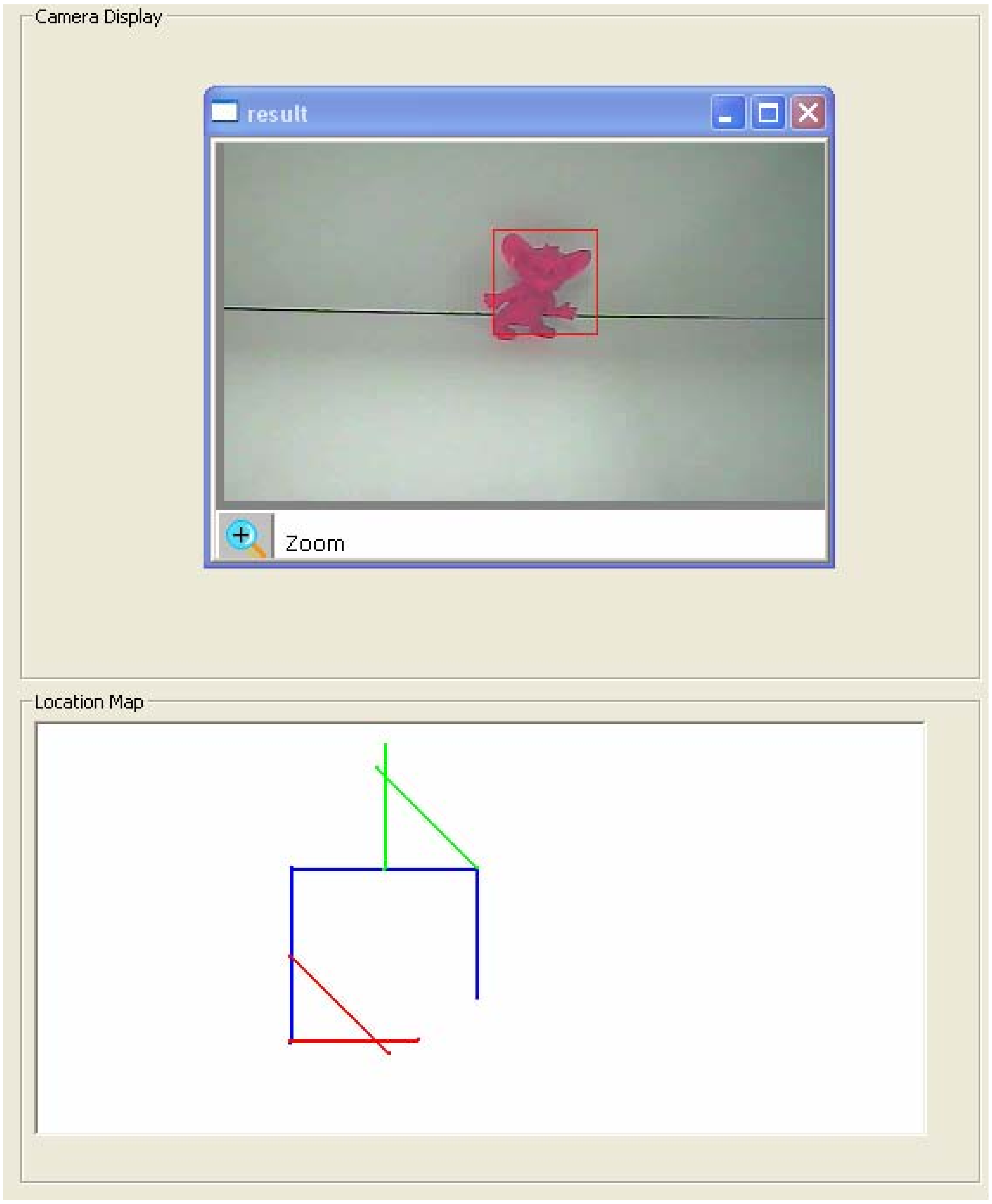


Figure 8.3: Screenshot of a victim being localized

Chapter 9

Presentation of information

In order to display the captured inputs and present information to users of this system as per requirements F10(section 3.1), a Graphical User Interface(GUI) has been implemented. The GUI displays the images of the camera and at the same time provides control facilities to the user. A live feed of the images captured by the camera are preferred to be displayed in order to describe movements because human decision-makers interprete images quicker and in a more efficient way than when presented with a set of statistical data. The amount of information could also be too large or complex to be handled and analyzed without some kind of ordered and systematic visualization of data. This chapter describes the GUI together with its various components including a video demonstrating it.

9.1 The Graphical User Interface

The GUI module has Java 2.0 Standard Edition and Java Standard Widget Toolkit (SWT) as underlying technology. It has been developed using Eclipse and SWT Designer. Java SWT provides a set of classes derived from the *Component* class to construct GUI's. This graphical user interface allows a user to control Usario and at the same time display specific information like a live feed of the floor space from the camera and the movement of the robot represented in a map of the area. As the user presses on graphic buttons of a joystick to move the robot around, the listeners[6] of the underlying program detect that a specific action has occurred and trigger an appropriate response to the event accordingly. The GUI components generate events that indicate that specific actions have occurred and handles them accordingly. Also, the GUI is using an instance of the commPC.java and Detect.java classes to send and receive data, and also to display live images of the scene (see Figure 9.1).

[6]A listener in programming context means special functions that always check the status of another entity.

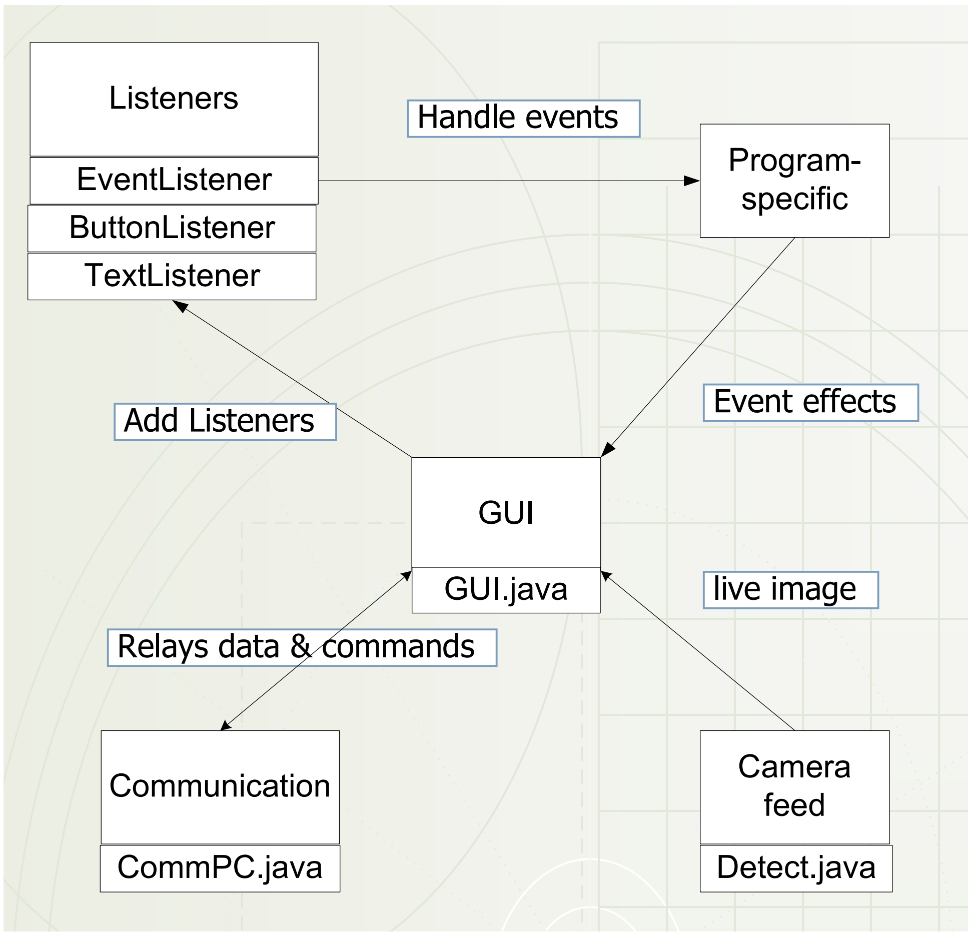


Figure 9.1: The GUI program model

The graphical user interface is composed of four main layers:

1. *Camera display*
2. *Joystick*
3. *Location map*
4. *Miscellaneous*

9.1.1 Camera Display.

This section of the GUI contains the image captured from the scene displayed in a native Windows window. The images displayed are those images grabbed from the camera's webserver and further processed to identify the victims. The 'camera display' section occupies the top left hand corner of the GUI. An example of the couloured victims being displayed surrounded by a rectangular frame in the 'camera display' section is shown in Figure 9.2.

9.1.2 Remote Control.

The remote Control is a component which allows the user to remotely control Usario's whereabouts. It is placed in the top right hand corner of the GUI, next to the 'Camera display' as shown in Figure9.2. The user controls the operation mode of Usario by setting on and off the autonomous or remote control mode . However, on starting the system the robot is under the control of the user who manoeuvres it around the arena. The joystick command, characterized by the arrow and the stop buttons) offers six available options to the user: 1. forward, 2. backward, 3. turn left, 4. turn right, 5. stop and 6. search. The first five commands are self-explanatory and the "search" button will trigger the autonomous mode of operation while the stop button disactivates the autonomous mode, giving control back to the user.

9.1.3 Location map.

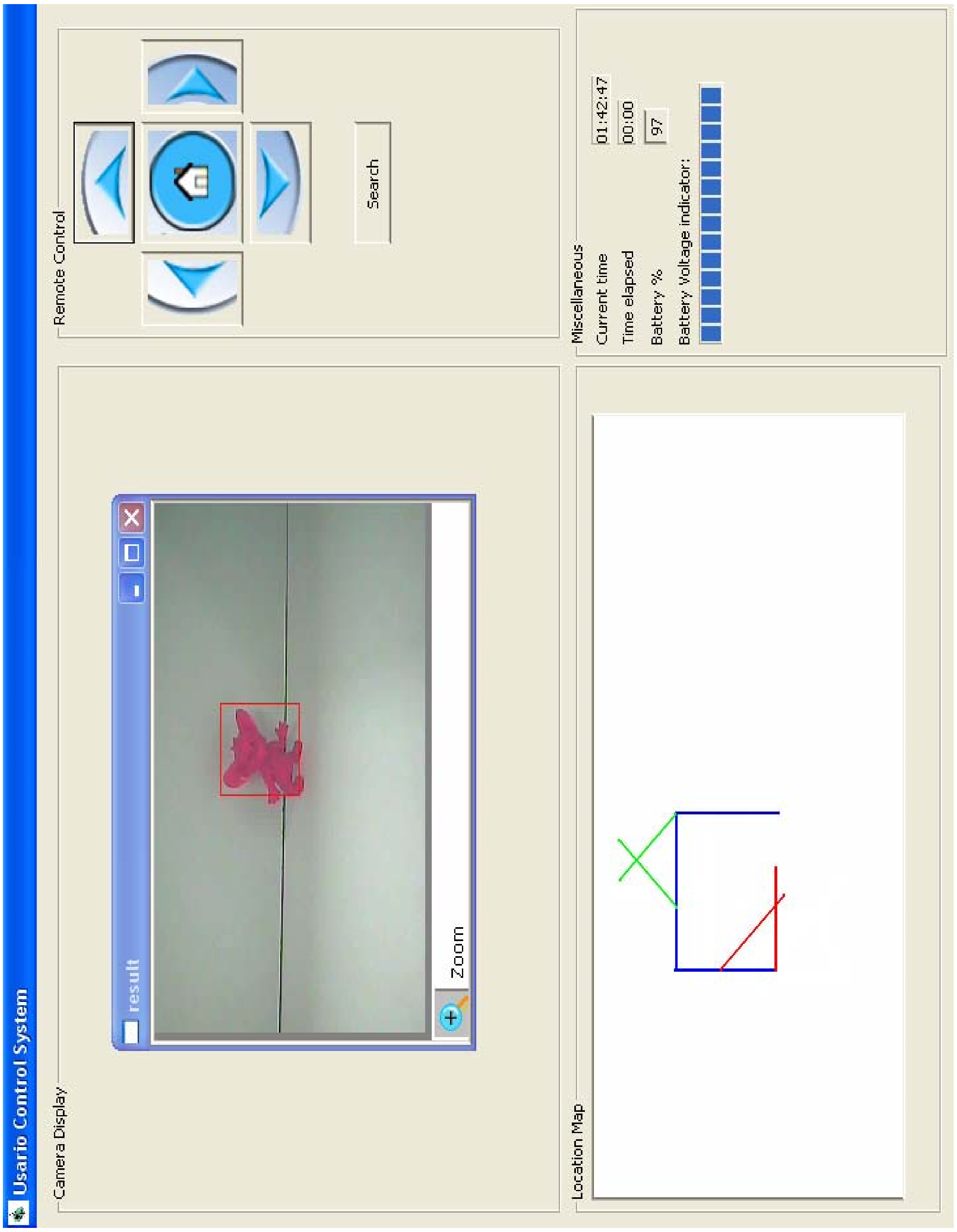
The 'location map' is a window which displays the path taken and the position of the identified victims by Usario. Figure 9.2 illustrates the window. The blue line represents the robot path while a red or green dot represents the postion of a victim.

- **Robot path plot.** The path of the robot is displayed as a series of lines joined together. The path is redrawn whenever Usario is stopped.

- **Victim location.** Knowing the current angle faced by Usario and the status of the detected object(s), a corresponding line is drawn on the canvas. The intersection of the two corresponding lines for the same victim indicates the exact location of the victim.

9.1.4 Miscellaneous

This section contains information about the system itself e.g time elapsed since start of system, current time and battery level indicator for the RCX. It is situated in the bottom right end of the GUI as shown in Figure 9.2.



Current time 01:42:47
Time elapsed 00:00
Battery % 97
Battery Voltage indicator:

Figure 9.2: Screenshot of the GUI

9.2 A video demonstration of the GUI

Figure 9.3 is a video file showing the robot model in action. The video shows the captured output from the screen as seen by a user logged into the system. . The video[7] is in 'avi' format and requires at least a version 9.0 Windows Movie player to be displayed properly.

Figure 9.3: Video file showing robotic system(requires at least a version 9.0 Windows Movie player)

[7]The video shows an outdated version of the 'Miscellaneous' section. It has been replaced by a newer version as in Figure 9.2

Chapter 10

Discussion and Conclusion

10.1 Discussion

This Section describes the problems encountered in the development of Usario, the USAR robot prototype, as well as investigating potential solutions to these problems.

(a) **General.** During the modular development of the robot prototype, things turned out to be working relatively well. However, as we started integrating the different modules together, the robot was getting very heavy and big. As an example, the navigation module for the robot was working satisfactorily but when the camera was added on for detection and localisation of the victims, the robot lost its accuracy. Recalibrating the torque for the motors and sensors did not help much as the Lego chassis for the robot would not tolerate heavy payloads; the robot was carrying a total of 16 AA and one 9 V batteries in addtion to the camera and the InfraRed/Bluetooth interface. Altogether, it summed up to a total weight of 2.8 kg. On epossible solution could be to use a smaller battery but which could provide power to the RCX, the camera and the InfraRed-Bluetooth interface simultaneously.

(b) **Navigation.** Without the DC battery units and the camera, the navigation alogrithm works well as Usario navigates throughout the search arena and did demonstrate that it would be able to exit the maze after exploring all rooms. However, affter being loaded with the camera and the InfraRed-Bluetooth interface, this was not the case. The robot could not complete a 360 degrees rotation when required . It was found that, for a rotation of 360^0 at 45^0 intervals, it would revolve either only about three quarter of a turn or 1.25 revolutions, i.e. the robot was not turning at the intended increments of 45^0, but sometimes less than 45^0 and sometimes more. The first or second 45^0 rotations were observed to be correct, but then for the rest of the rotations, the robot would have an inaccuracy of 10 to 15^0. This would lead to an accumulated inaccuracy of $75^0 \pm 15$ for a 360^0 rotation and cause the robot to turn around the arena without moving in a reasonable direction for further exploration. According to our own experience and observation, we give the following hypotheses as potential causes of the problem:

Hypothesis 1: Before any sidewise turn, the robot's wheel would be rotating perpendicularly to its axle. However, as it turns to one side, there is a greater friction force generated by the ground against the wheels. The robot had also a smaller surface contact area with the floor when it had lesser weight. As more weight was added to the robot, the tyres had a greater surface area contact with the ground and thereby provided greater friction. As described in section 5.2.1-wheels, the tyres being non-pneumatic, there are no means to inflate the tyres to reduce the area of surface contact.

Hypothesis 2: The two DC 9V motors that are being used do not provide necessary Torque to manoeuvre heavy payloads,in addition to what they are designed for. They therefore consume more power in their attempt to rotate the wheels and as a consequence, the batteries get drained quicker than usual. One of the biggest problems in achieving accurate parameters is that as the battery loses power, the output speed of the motors changes. We have noticed several times that when the RCX is turned on, it may have a charge of 7.6 V, and by the time the program terminates the battery power is down to 7.3 V. Then, after shutting down for a few minutes and turning it on, the voltage once again reads 7.6 V. This significant drop in voltage really affetcs the rotations. As the battery charge lessens, the settings become no longer valid. One solution to this problem is to use freshly charged batteries very frequently,such as for every run.

Hypothesis 3: Differences in motor speed between the left motor and the right motor also affect accuracy greatly. One of the noticeable problems encountered in this experiment is the inability of the robot to navigate in a straight line. Just when we thought we had the motors balanced it would start drifting to the right. Then, later in the same run, it would mysteriously straighten up for some stretches. This effect can be attributed to a mix of frictional differences in the structure of the robot, the uneven distribution weight across the robot and the differences in motor power.

Hypothesis 4: The rotation function in the navigation function of the RCX is limited to the rotation capabilities of the motors. Although the rotation sensors enables recording of the number of rotations, there is no known way to control fractions of rotations; i.e. the motors can be set to rotate for some integer units but not in fractions of rotations. In a simple test experiment where we observed the number of rotations of the motors for a 360 degrees rotation of Usario, 26 rotations were recorded. We suspect that when Usario is rotating for a 45^0 interval, the motors will turn to either 3 or 4 rotations, not to 3.25 rotation, which leads to an accumulated error of -13.8 or + 83.1 degrees. We then adjusted our rotation/scanning algorithm for Usario accordingly. The robot was forced to turn 3 rotations for a 45^0 interval when it is in the rotation segments 1, 2, 3, 5, 6 and 7. In the 4^{th} and 8^{th} intervals, the robot is made to turn 4 rotations, which ensures that the total rotation angle is 360^0 and the number of rotations 26.

(c) **Communication.** It has been observed during the exchange of data between the PC and the RCX that data sometimes get distorted or lost. The embedded communication protocol in the IR-Bluetooth interface (i.e. class SimpleCommRCX.java and SimpleCommPC.java) do try, to a certain extent, to correct these imperfections. However, not all data are properly transferred, which leads to some occasional deadlocks and that paralyses the whole system. The entire system gets affected because the communication protocol implemented in the RCX is entirely reliant on a smooth interaction between the sender and receiver. Once the communication link is broken or deadlocked, the next operation does not get executed, hence paralysing the entire system. According to the technical report "Simple Error-Correcting Communication Protocol for RCX" [27] this is an issue that seems to be unavoidable:

> ".. Unfortunatelly, the LeJos/ROM sending routines do not work perfectly, and sometimes emit erroneous IR signals. We believe that this is due to the fact that LeJos firmware does some interrupt-handling for multitasking, or another low-level system activity that interferes with the sending routines, but the outcome is that the messages are sometimes sent with bit errors. Since the IR signal is received with the IR to BT converter, the BT module simply discards those bytes where the parity bit check fails. As a consequence, the message received by the BT receiver on the side of the PC does contain occassional erasures at unknown locations as well as occassional errors - in cases where the erroneous byte still passes the parity-bit checking. Finally, in all our measurements, we noticed that the scrumbled or missing bytes occur seldom - never more often than every fourth byte.." Extracted from page 3 in [27].

(d) **Object detection.** The inaccuracy in navigation leads to some unexplored areas by the camera. Recall that that the robot detects only those objects/victims that are in the middle of the camera view, and since the robot is not accurate in pointing at angles, as it should have been, some objects/victims might be left undetected. The angle of rotation while scanning has been chosen to be 45 degrees to keep the number of turns as low as possible. No tests or experiments have been performed to carefully determine the appropriate angle that would cover all regions. Therefore, this is another possible reason that victimsare sometimes not detected. One potential solution to this problem could be to rotate the camera instead of rotating Usario. Appropriate functions exist within the configuration page (of the camera) to remotely control the camera in such a way.

(e) **Object localization.** Since the robot is not performing as it should, the data collected becomes unreliable. After scanning around at an angle of 360^0 to look for victims, the robot has actually rotated only 270^0 which invalidates the position coordinates of both robot and victim(s). All position coordinates become invalid as the robot is now in conflict with the real situation; it thinks that it is facing the same angle after having rotated 360^0(or 26 rotaions of the motor(s)) while it has, in fact, rotated only an angle of 270^0. Another problem is that Usario does not take the position of the room separations of the search area into consideration at all. This has lead to invalid data and a possible solution could be to draw the position of the walls and room separations on the map.

(f) **Presentation of information.** The problem of incremental inaccuracy while turning leads to some unreliable data coordinates. As the robot's coordinates are sent to the server, the robot is actually at a different place than where the RCX has computed. This lead to an erroneous data map coordinates being plotted on the GUI. These are all errors that must be corrected for the robot to be used.

10.2 Conclusion

In this experiment we have only dealt with a one-floor crash site but in a real life situation, such crashed sites are far more complex and unpredictable. However, this work represents a step forward in the development of a robotic system since we have managed to build and implement a customized and tailor-made system from relatively cheap equipment. Components like the RCX were not really suited for moving robots but we have found solutions to adapt it to our needs by implementing a new InfraRed-Bluetooth interface. Furthermore, the standard Lego camera was not good enough to capture quality images for further image processing so we improvised and used another camera, originally designed for home and office monitoring. In addition, all the Functional requirements listed in 3.1 have been satisfied, except for F7 - Access, due to time constraints. We hope that this experiment will contribute to the research on robotic search and rescue operations. It could, for instance, be used as a preliminary work for those researching on safety-critical USARs, as we have discussed and tested some of the core issues that need to be with for a safety critical control and monitoring USAR robotic system. It could also be used as a basis for secondary institutions who are comtemplating entering the RoboCup Rescue competition(s).

10.3 Future work

While performing this experiment, some aspects of the robot have been identified as they have the potential to improve the potential of the robot. Improving these aspects could make the robot more accurate and smoothly working. In future assignments, more focus and efforts should be put in the following:

(a) Instead of rotating the robot to detect objects, one should consider turning the camera. This is more practical as turning the camera leads to less energy consumption. It also gives a continuous image as it turns, something which our robot does not.

(b) The Legos core chassis proved to be too weak to support bigger weights. One could either switch to another chassis, other than Lego, or consider lighter equipment parts than the RCX and the camera. In addtition, an alternative to the 17 DC batteries which weighs less would also help.

(c) The ability to hear, see, speak, sense and move is key to the human ability to interact with its environment. However, to perform such human-like tasks, robotics systems must be able to emulate these functions using software. Speech is the only aspect of these that has not been investigated t in this work. A speech-enabled application can be developed alongside the robotics system and give a better interaction with the victims rather than just detecting and/or localising them.

(d) The GUI of the search and rescue robotics system should also be ported to a palmtop. This would enable the owners of the site to be mobile and not be restricted to the constraints of laptops and desktops.

(e) The robot should be able to climb over obstacles and have better wheels for better grip of the ground floor (like the wheels of a tanker or a 'caterpillar'). An alternative could be to acquire the crawler robot of Inuktun services [15] and modify the software system accordingly.

(f) As a safety precaution, an improved mechanism for detection of humans should be considered. In addition to object detection by camera, one could consider detecting humans by Carbon Dioxide, temperature or even by the presence of blood. There are a wide range of sensors on the market that are made to detect specific gases, temperature and even recognition of liquids. An improved detection mechanism could be by detecting the presence of Carbon Dioxide and/or the use of heat sensors.

Appendix A

Meeting with the Fire department in Trondheim

This chapter presents the conversation between J.P.Franky Friquin (F) and Fire Officer Sibjœrn Pareli Lyngdal (L) from the central Fire department in Trondheim, Norway about search and rescue operations. The conversation took place on the 8th of March, 2006 at 1300 hrs. The objective of this meeting is to have a clearer and more accurate picture of the rescue operations during a rescue operation after a building collapse.

F Good afternoon, Mr Lyngdal.

L Good afternoon.

F From a fireman's point of view, can you define the objectives of a search and rescue operation?

L 1. Save human lives 2. Prevent and preserve damages to property and infrastructure. 3. Save animal lives.

F Can you tell me a bit about the process of rescue operation?

L First of all, in the case of an accident or catastrophe, there are a number of groups of people involved; the police, the Fire Search and Rescue department, the paramedics, the Civil Defence and the Dog Rescue unit. In cases of really big accidents , the Red Cross or the Norsk Folkehjel' are also present. The police has the general command of the operations and is so-called the "Owner of the scene". They use the leaders of the different groups as advisers as they co-ordinate the effort. They have the duty and authority to secure the area and call for extra personnel, equipment or expertise.

F What are the duties of the other people at the scene?

L If the firemen arrive first at the scene, they have the legal authority and command until the police arrives. The paramedics do not have any authority and are always under the command of the owner.

F What are the standard procedures?

L The standard procedures, from the owner's point of view, is first to secure the area around the facility, then the facility itself. The owner has to ensure that the accident does not become a danger to the surroundings. Like for example, he has to ensure that there is no danger of additional fire or explosions and that there is no risk of further casualties and injuries, albeit deaths. From a fireman's point of view, the procedure is such that once the go ahead clearance is granted, the main priority is to evacuate the civilians from the scene. However, much emphasis is laid out on safety and security. Securing the site is our absolute priority as the regulations, in Norway, stipulate that personnel should not be sacrificed or put at risk to rescue endangered victims.

F The owner of the scene does take advice from the leader of the different groups involved, what kind of information does he need? What kind of information is he looking for when he is consulting the other leaders?

L

- Where is the exact location of the fire/accident?
- Are there any victims inside? How many of them?
- What is their location?
- What is their status? dead, alive, injured,etc..
- How is the plan of the building/scene?
- What is the extent of the fire?

- What is the stage of the fire? Is it at the beginning, the middle or the end phase?
- Where has the fire spread?
- Are there any other possible dangers? Like gas explosions etc..
- Is there any danger to the structure of the building?
- What is the quality of the air inside? Is there any visibility?
- What is the evaluation of the benefits of output against input? He has to weigh exposing the rescue personnel to danger versus the final outcome of the effort.

F There seems to be a strong need to gather intelligence. Do you think that Information technology can be of a help?

L Yes.

F What are the current problems in gathering intelligence?

L Sometimes, it is very difficult to obtain information like the number of people, their location and their status. In addition, it could be hard to have an extended knowledge of the situation inside the scene itself.

F Do you think that robotics can be of any help in rescue operations?

L Yes. They can attend to places inaccessible or difficult for humans or dogs to reach.

F Any example of such situations?

L In accidents during the transportation of bio-chemical stuffs, in cases where anthrax or other poisonous gases are involved.

F Mr Lyngdal, thank you for answering my questions.

L Vær så god.

-

Appendix B

Source code for the robotic system

The following are the Java source code file for both Usario and PC. They are provided for a better understanding of the system implementation.

Listing B.1: Multi-PageJava code for Usario

```
import josx.platform.rcx.*;
import josx.robotics.*;
/* * This class controls the RCX which performs functions for
    navigation and sending coordinates to the server. */

class RCX12 extends Thread
{
   private int commandReceived;
   private int action;
   private int i=0;
   private float x,y;    //current coordinates
      private int angle; //current angle (0-7) in 45-degree multiplies
   private int movement_started;  //value on the rotation sensor when
       this movement started
   private int movement_started1;  //value on the rotation sensor when
        this movement started
   private int movement_started3;  //same for sensor3
   private int movement_type;  //either +1 if we are moving forward or
       -1 otherwise, 0 if we are turning
   private byte[] packetIn;
   private byte[] packetOut;
   private int len;  //will contain the length of received packet
   private int[] scanArray;
   private SimpleCommRCX commRCX;
   private boolean wallAtFRONT; // indicates if there is wall/obstacle
```

```
        in this direction
    private boolean wallAtRIGHT; // indicates if there is wall/obstacle
        in this direction
    private boolean wallAtLEFT; // indicates if there is wall/obstacle
       in this direction
    private int direction; /* indicates the cardinal direction the
       robot is travelling.
                  NOTE: this direction is with respect to the direction
                      the
                 * robot was facing at the previous move. */
    float lastScan = 0;
    private int previousDirection;
    private final int FRONT = 0;
    private final int LEFT = 2;
//  private final int SOUTH = 6;
    private final int RIGHT = 6;
    private final int AUTONOMOUS = 0;
    private final int STOP = 1;
    private final int MOVE = 2;
    private final int FORWARD = 3;
    private final int BACKWARD = 4;
    private final int ROTATE_RIGHT = 5;
    private final int ROTATE_LEFT = 6;
    private final int DECIDE = 7;
    private final int SCAN = 8;
    private final int RUN = 9;
    private final int ROTATE = 10;
    private final int BACKTOBASE = 11;
    private final int NONE = 999;

      private final float SQRT2 = 1.41421356237f;
      private static final int TURN45DEGREE=3;

    //private static int batteryVoltage;
    //batteryVoltage = Battery.getVoltageMilliVolt();

    public RCX12() {
           x = 0; y = 0; angle = 0; movement_type = 0;

           commandReceived = NONE;

      previousDirection = LEFT;
        commRCX = new SimpleCommRCX();
      packetIn = new byte[1];   // incoming packet
      packetOut = new byte[5];   //outgoing packet
      direction = LEFT;
      scanArray = new int[12];
    }

    public static void main(String args[]) throws Exception {// this
       main allows robot to start in autonomous mode!!!!!
      RCX12 ff = new RCX12();
      ff.startup();
    }

    private void startup() {
      int cnt = 0;
      setup();
      this.start();

      action = RUN;
        while(true) {
           if (commandReceived == AUTONOMOUS) {
             //Sound.buzz();
            try {
              scan();
              decide();
            } catch (InterruptedException e){Sound.beepSequence();}
            finally {
              try {Thread.sleep(1000);} catch (InterruptedException e)
                  {}
              if (commandReceived == AUTONOMOUS) {
                 move();
                 Sound.beepSequence();
                 commandReceived = NONE;
                 }
            }
           }
           else if (commandReceived == STOP) { //comman 1; operator
               takes control of robot, and robot stops to wait for next
               command
              action = STOP;
              Stop();
              commandReceived = NONE;
             }
           /*else if (commandReceived == MOVE) { //
           }*/
           else if (commandReceived == FORWARD) {//command 3; travel
               moves robot forward for 1 sec.
             action = FORWARD;
             forward();
             commandReceived = NONE;
```

```
            //myTravel();
          }
          else if (commandReceived == BACKWARD) {
            action = BACKWARD;
             backward(); //command 4, moves robot backwards until stop
                 () is called
           commandReceived = NONE;
           }
          else if (commandReceived == ROTATE_LEFT) {
             action = ROTATE_LEFT;
             turn45(); //command 5; rotates robot 30 degrees to the
                 left
           commandReceived = NONE;
           }
          else if (commandReceived == ROTATE_RIGHT) {
            action = ROTATE_RIGHT;
             turn45ti(); //command = 6; rotates robot 30 degrees to
                 the right
              commandReceived = NONE;
           }
          else if (commandReceived == SCAN) {
            action = SCAN;
           try{scan();} catch(InterruptedException ie){} //robot scans
                around for victims.
           commandReceived = NONE;
           }
          else if (commandReceived == BACKTOBASE) {
            action = BACKTOBASE;
            commandReceived = NONE;
           }
          else { if (cnt % 10 == 0) update_coordinates(); cnt++;
             adjust_movement(); }
      } // end while(True)
  } // end main();


  private void scan() throws InterruptedException{
    Thread.sleep(1000);
    wallAtFRONT = false;
    wallAtLEFT = false;
    wallAtRIGHT = false;
    action = SCAN;

    //scanning method
      for(i=0;i<8;i++){
          if (commandReceived == AUTONOMOUS) {
             scanArray[i] = Sensor.S2.readValue();
             turn45ti();
             if (i == FRONT ) {
                if (checkWall(scanArray[i])) {
                  wallAtFRONT = true;
                  }
             }
             else if (i == LEFT ) {
                if (checkWall(scanArray[i])) {
                  wallAtLEFT = true;
                  }
             }
             else if (i == RIGHT ) {
                if (checkWall(scanArray[i])) {
                  wallAtRIGHT = true;
                  }
             }
             else{}
             LCD.setNumber(LCDConstants.LCD_SIGNED,(scanArray[i]),
                 LCDConstants.LCD_DECIMAL_0);
           }
           Thread.sleep(1000);
      } // end for
  //return scanArray;
  } // end method scan()


  public boolean checkWall(int sensorValue){
    boolean wall = false;
    if (sensorValue < 62) {
      wall = true;
    }
    return wall;
  }

  // decide where to move
  public void decide() {
    action = DECIDE;

    if ((!wallAtFRONT) && (!wallAtRIGHT) && (!wallAtLEFT)) {
      //move();
    }
    else if (!wallAtFRONT && !wallAtRIGHT && wallAtLEFT){

    }
    else if (!wallAtFRONT && wallAtRIGHT && !wallAtLEFT){
    }
    else if (!wallAtFRONT && wallAtRIGHT && wallAtLEFT){

    }
    else if (wallAtFRONT && !wallAtRIGHT && !wallAtLEFT){
      if (getPreviousDirection() == LEFT) {
        setPreviousDirection(RIGHT);
        direction = RIGHT;
        turn90clockwise();
      }
      else {
        setPreviousDirection(LEFT);
        direction = LEFT;
        turn90anticlock();
        }
    }
    else if (wallAtFRONT && !wallAtRIGHT && wallAtLEFT){
      direction = RIGHT;
      turn90clockwise();
    }
    else if (wallAtFRONT && wallAtRIGHT && !wallAtLEFT){
      direction = LEFT;
      turn90anticlock();
    }
    else if (wallAtFRONT && wallAtRIGHT && wallAtLEFT){

    }
  }

  public int getPreviousDirection() {
    return previousDirection;
  }
  public void setPreviousDirection(int dir) {
    previousDirection = dir;
  }


    private void setup()
    {
    Sensor.S1.setTypeAndMode(4, 0xE0);
    Sensor.S3.setTypeAndMode(4, 0xE0);
    Sensor.S1.activate();
    Sensor.S3.activate();
    Sensor.S1.setPreviousValue(0x4000);
    Sensor.S3.setPreviousValue(0x4000);
    Sensor.S2.setTypeAndMode (SensorConstants.SENSOR_TYPE_LIGHT,
        SensorConstants.SENSOR_MODE_PCT);
    Sensor.S2.activate();
    Button.PRGM.addButtonListener(new myListener());
    fullPower();
  }

    private void fullPower()
    {
    Motor.A.setPower(7);
    Motor.C.setPower(7);
  }

    private void forward()
    {
    fullPower();
    movement_started = Sensor.S1.readValue();
    movement_started1 = Sensor.S1.readValue();
    movement_started3 = Sensor.S3.readValue();
    movement_type = 1;
    Motor.A.forward();
    Motor.C.forward();
  }

    private void backward()
    {
    fullPower();
    movement_started = Sensor.S1.readValue();
    movement_started1 = Sensor.S1.readValue();
    movement_started3 = Sensor.S3.readValue();
    movement_type = -1;
    Motor.A.backward();
    Motor.C.backward();
  }

  private void turnanti()
  {
    Motor.A.forward();
    Motor.C.backward();
  }

    private void turn()
  {
    Motor.A.backward();
    Motor.C.forward();
  }
```

```

    private void adjust_movement()
    {
     if (movement_type == 0) return;
     int s1 = Sensor.S1.readValue() - movement_started1;
     int s3 = Sensor.S3.readValue() - movement_started3;
     if (s1 < 0) s1 = -s1;
     if (s3 < 0) s3 = -s3;
       LCD.showNumber(s1);
       if (s1 > s3 + 1) { Motor.A.setPower(7); if (movement_type == 1)
            Motor.A.forward(); else Motor.A.backward(); Motor.C.flt();
           }
       else if (s3 > s1 + 1) { Motor.C.setPower(7); if (movement_type
           == 1) Motor.C.forward(); else Motor.C.backward(); Motor.A.
           flt(); }
       else { Motor.A.setPower(7); Motor.C.setPower(7); if (
           movement_type == 1) { Motor.A.forward(); Motor.C.forward();
           } else { Motor.A.backward(); Motor.C.backward(); }}
    }

    private void update_coordinates()
    {
      int new_rotation_value = Sensor.S1.readValue();
     int distance = movement_type * Math.abs(new_rotation_value -
        movement_started);
     movement_started = new_rotation_value;
     if (angle == 0) y += distance;
     else if (angle == 1)
     {
       x += distance/SQRT2;
       y += distance/SQRT2;
       }
       else if (angle == 2) x += distance;
     else if (angle == 3)
     {
       x += distance/SQRT2;
       y += distance/SQRT2;
       }
       else if (angle == 4) y -= distance;
       else if (angle == 5)
       {
       x += distance/SQRT2;
       y += distance/SQRT2;
     }
     else if (angle == 6) x -= distance;
     else
```

```
        {
        x += distance/SQRT2;
        y += distance/SQRT2;
      }
      }

    private void Stop()
    {
      int distance;

      Motor.A.stop();
      Motor.C.stop();
      try { Thread.sleep(50); } catch (Exception e) {}
      Motor.A.flt();
      Motor.C.flt();

          update_coordinates();
      movement_type = 0;
    }

    private void turn45()
    {
        fullPower();
            int val1 = Sensor.S1.readValue();
            int limit = TURN45DEGREE;
            turnanti();
        while (Math.abs(Sensor.S1.readValue() - val1) < limit);
        try { Thread.sleep(8); } catch (Exception e) {}
        Stop();
        angle--;
        if (angle < 0) angle = 7;
    }

    private void turn45ti()
    {
            fullPower();
              int val1 = Sensor.S1.readValue();
              int limit = TURN45DEGREE;
              turn();
          while (Math.abs(Sensor.S1.readValue() - val1) < limit);
          try { Thread.sleep(8); } catch (Exception e) {}
          Stop();
          angle++;
          if (angle == 8) angle = 0;
    }

```

```
private void turn90anticlock()
{
    fullPower();
        int val1 = Sensor.S1.readValue();
        int limit = TURN45DEGREE * 2;
        turn();
    while (Math.abs(Sensor.S1.readValue() - val1) < limit);
    try { Thread.sleep(16); } catch (Exception e) {}
    Stop();
    angle = (angle + 6) % 8;
}

private void turn90clockwise()
{
    fullPower();
        int val1 = Sensor.S1.readValue();
        int limit = TURN45DEGREE * 2;
        turnanti();
    while (Math.abs(Sensor.S1.readValue() - val1) < limit);
    try { Thread.sleep(16); } catch (Exception e) {}
    Stop();
    angle = (angle + 2) % 8;
}

private void move()
{
  fullPower();
  action = MOVE;
  forward();
  Sound.beep();
  while (Sensor.S2.readValue() >= 62) { update_coordinates();
     adjust_movement(); }
  Stop();
  Sound.buzz();
}

private void activateReceive() {
  len = 0;
  do {
    len  = commRCX.receive(packetIn);
  } while (len <= 0);
  commandReceived = (int)((byte)packetIn[0]);
  Sound.beep();
}

private void getAndSendCoords() {
      packetOut[0] = (byte)commandReceived;
      packetOut[1] = (byte)x;
      packetOut[2] = (byte)y;
      packetOut[3] = (byte)angle;
      packetOut[4] = (byte)action;
      commRCX.send(packetOut,5);
      Sound.beep();
   }

      public void run()
      {
      while (true)
      {
         activateReceive();
         try { Thread.sleep(8); } catch (Exception e) {}
         getAndSendCoords();
      }
   }


}// end class RCX12
```

Listing B.2: Multi-Page Java code for GUI unit

```java
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.Image;
/* Class responsible for the display of the GUI at the PC side */

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import com.swtdesigner.SWTResourceManager;

public class TestFrame{
  public Display display;
  public Canvas canvas;
  public int[] ArrayXcoords;
  public int[] ArrayYcoords;
  public int arrayIndex;
  public double angle;
  public int command;
  //public String batteryValue;
  private final int AUTONOMOUS = 0;
  private final int STOP = 1;
  //private final int MOVE = 2;
  private final int FORWARD = 3;
  private final int BACKWARD = 4;
  private final int ROTATE_RIGHT = 5;
  private final int ROTATE_LEFT = 6;
  private final int NONE = 0;
  private final int RED = 1;
  private final int GREEN = 2;
  //private final int DECIDE = 7;
  //private final int SCAN = 8;
  //private final int RUN = 9;
  //private final int ROTATE = 10;
  private final int BACKTOBASE = 11;
  public int objectDetected;
  private int x1,x2,y1,y2;
  private int m = 0;
  private int n = 0;
  private final int [][] arrayRed = new int[50][4];
  private final int [][] arrayGreen = new int[50][4];
  public Label time2;
  public String myTime;

  public TestFrame() {
    ArrayXcoords = new int[10000];
    ArrayYcoords = new int[10000];
    arrayIndex = 0;
    command = STOP;
  }
  public void run() {
    display = new Display();
    Shell shell = new Shell(SWT.SHELL_TRIM);
    shell.addDisposeListener(new DisposeListener() {
      public void widgetDisposed(DisposeEvent e) {
        System.exit(0);
      }
    });
    shell.setLayout(new GridLayout());
    shell.setMaximized(true);
    shell.setImage(new Image(display,"C:\\EclipseWorkspace\\
       HelloWorld\\src\\pics\\Robot.gif"));
    shell.setText("Usario Control System");
    shell.pack();
    shell.setSize(900, 680);
    createContents(shell);
    shell.open();
    while (!shell.isDisposed())
      if(!display.readAndDispatch())
        display.sleep();
    display.dispose();
  }
  public void createContents(Shell shell){
    {
      final Composite composite = new Composite(shell, SWT.NONE);
      final GridLayout gridLayout = new GridLayout();
      gridLayout.numColumns = 2;
      GridData compositeLData = new GridData(GridData.END, GridData.
         CENTER, false, false);
      compositeLData.widthHint = 800;
      compositeLData.heightHint = 800;
      composite.setLayoutData(compositeLData);
      composite.setLayout(gridLayout);
      {
        final Group ImageDisplay = new Group(composite, SWT.NONE);
        final GridLayout gridLayout_1 = new GridLayout();
        ImageDisplay.setLayout(gridLayout_1);
        GridData ImageDisplayLData = new GridData(GridData.END,
           GridData.CENTER, false, false);
        ImageDisplayLData.widthHint = 550;
        ImageDisplayLData.heightHint = 375;
```

```
            ImageDisplay.setLayoutData(ImageDisplayLData);
            ImageDisplay.setText("Camera Display");

        }
        {
            final Group RemoteControl = new Group(composite, SWT.NONE);
            final GridLayout gridL = new GridLayout();
            gridL.numColumns = 3;
            GridData RemoteControlData = new GridData(GridData.CENTER,
                GridData.FILL, false, false);
            RemoteControl.setLayout(gridL);
            RemoteControl.setLayoutData(RemoteControlData);
            RemoteControl.setText("Remote Control");
            {
                final Button button1 = new Button(RemoteControl, SWT.NONE);
                button1.setVisible(false);
                button1.setEnabled(false);
                button1.setImage(SWTResourceManager.getImage(TestFrame.
                    class,"pics/pic1.gif"));
                button1.setBounds(301, 35, 50, 50);
            }
            {
                final Button button2 = new Button(RemoteControl, SWT.NONE);
                button2.addSelectionListener(new SelectionAdapter() {
                    public void widgetSelected(SelectionEvent e) {
                        command = FORWARD;
                    }
                });
                button2.setLayoutData(new GridData());
                button2.setImage(SWTResourceManager.getImage(TestFrame.
                    class,"pics/pic2.gif"));
                button2.setBounds(355, 35, 50, 50);
            }
            {
                final Button button3 = new Button(RemoteControl, SWT.NONE);
                button3.setVisible(false); //false);
                button3.setEnabled(false); //false);
                button3.setImage(SWTResourceManager.getImage(TestFrame.
                    class,"pics/pic3.gif"));
                button3.setBounds(410, 35, 50, 50);
            }
            {
                final Button button4 = new Button(RemoteControl, SWT.NONE);
                button4.addSelectionListener(new SelectionAdapter() {
                    public void widgetSelected(SelectionEvent e) {
                        command = ROTATE_LEFT;
            }
          });

          button4.setImage(SWTResourceManager.getImage(TestFrame.
              class,"pics/pic4.gif"));
          button4.setBounds(301, 90, 50, 50);
        }
        {
          final Button button5 = new Button(RemoteControl, SWT.NONE);
          button5.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
              command = STOP;
            }
          });
          button5.setLayoutData(new GridData());
          button5.setImage(SWTResourceManager.getImage(TestFrame.
              class,"pics/pic5.gif"));
          button5.setBounds(355, 90, 50, 50);
        }
        {
          final Button button6 = new Button(RemoteControl, SWT.NONE);
          button6.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
              command = ROTATE_RIGHT;
            }
          });
          button6.setImage(SWTResourceManager.getImage(TestFrame.
              class,"pics/pic6.gif"));
          button6.setBounds(410, 90, 50, 50);
        }
        {
          final Button button7 = new Button(RemoteControl, SWT.NONE);
          button7.setVisible(false);
          button7.setEnabled(false);
          button7.setImage(SWTResourceManager.getImage(TestFrame.
              class,"pics/pic7.gif"));
          button7.setBounds(301, 145, 50, 50);
        }
        {
          final Button button8 = new Button(RemoteControl, SWT.NONE);
          button8.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
              command = BACKWARD;
            }
          });
          button8.setLayoutData(new GridData());
          button8.setImage(SWTResourceManager.getImage(TestFrame.
              class,"pics/pic8.gif"));
          button8.setBounds(355, 145, 50, 50);
        }
        {
          final Button button9 = new Button(RemoteControl, SWT.NONE);
          button9.setVisible(false);
          button9.setEnabled(false);
          button9.setImage(SWTResourceManager.getImage(TestFrame.
              class,"pics/pic9.gif"));
          button9.setBounds(410, 145, 50, 50);
        }
        {
          final Button button10 = new Button(RemoteControl, SWT.NONE)
              ;
          button10.setVisible(false);
          button10.setEnabled(false);
          button10.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
              command = BACKTOBASE;
            }
          });
          button10.setLayoutData(new GridData(GridData.FILL, GridData
              .FILL, false, false, 3, 1));
          button10.setText("RESET");
          button10.setBounds(301, 140, 460, 90);
        }
        final Button button11 = new Button(RemoteControl, SWT.NONE);
        button11.setVisible(false);
        button11.setEnabled(false);
        final Button AutonomousOnOff = new Button(RemoteControl, SWT.
            TOGGLE);
        AutonomousOnOff.setLayoutData(new GridData(GridData.FILL,
            GridData.FILL, false, false, 1, 2));
        AutonomousOnOff.setText("Search");
        AutonomousOnOff.setSelection(false);
        AutonomousOnOff.addSelectionListener(new SelectionAdapter() {
          public void widgetSelected(SelectionEvent e) {
            command = AUTONOMOUS;
            if (AutonomousOnOff.getSelection()){
              command = AUTONOMOUS;
              AutonomousOnOff.setText("Grab Control");
              AutonomousOnOff.setSelection(true);
            }
            else {
                command = STOP;
                AutonomousOnOff.setText("Search");
                AutonomousOnOff.setSelection(false);
            }           }
        });

    }
    {
        final Group PathPlot = new Group(composite, SWT.NONE);
        final GridLayout gridLayout_1 = new GridLayout();
        PathPlot.setLayout(gridLayout_1);
        GridData PathPlotLData = new GridData(550, 270);
        PathPlot.setLayoutData(PathPlotLData);
        PathPlot.setText("Location Map");
        {
        }
          canvas = new Canvas(PathPlot,SWT.BORDER);
          canvas.setBackground(SWTResourceManager.getColor(255, 255,
              255));
          final GridData gridData = new GridData(512, 236);
          canvas.setLayoutData(gridData);
        canvas.setVisible(true);
          canvas.addPaintListener(new PaintListener() {
            public void paintControl(PaintEvent e) {
                e.gc.setForeground(e.display.getSystemColor(SWT.
                    COLOR_BLUE));
                e.gc.setLineWidth(2);
            for (int i=1;i<arrayIndex;i++) {
                  e.gc.setForeground(e.display.getSystemColor(SWT.
                      COLOR_BLUE));
                  x1 = ((int)ArrayXcoords[i - 1])+250;
                  x2 = ((int)ArrayXcoords[i])+250;
                  y1 = ((int)ArrayYcoords[i - 1])+100;
                  y2 = ((int)ArrayYcoords[i])+100;
              e.gc.drawLine(x1,y1,x2,y2);
              time2.update();
              System.out.println("state of objectDetected variable is
                  : " + objectDetected);
              if (objectDetected != NONE)
              {
                System.out.println("Detected victim is: " +
                    objectDetected);
                angle *= 45;
                if (y1 > y2)
                {
                  angle += 180;
                }
                if (x1 < x2)
                {
                  angle += 90;
                }
                else if (x1 > x2)
                {
                  angle += 270;
                }
                if (objectDetected == RED)
                {
                  System.out.println("Detected victim is: " +
                      objectDetected);
                  arrayRed[m][0] = x1;
                  arrayRed[m][1] = y1;
                  arrayRed[m][2] = x2+(int)(Math.cos(angle)*30);
                  arrayRed[m][3] = y2+(int)(Math.sin(angle)*30);
                  m++;
                }
                else if (objectDetected == GREEN)
                {
                  arrayGreen[n][0] = x1;
                  arrayGreen[n][1] = y1;
                  arrayGreen[n][2] = x2+(int)(Math.cos(angle)*30);
                  arrayGreen[n][3] = y2+(int)(Math.sin(angle)*30);
                  n++;
                }
              }
              objectDetected = NONE;
            }
                e.gc.setLineWidth(2);
            e.gc.setForeground(e.display.getSystemColor(SWT.COLOR_RED
                ));
            e.gc.drawLine(arrayRed[m][0], arrayRed[m][1], arrayRed[m
                ][2], arrayRed[m][3]);
            e.gc.setForeground(e.display.getSystemColor(SWT.
                COLOR_GREEN));
            e.gc.drawLine(arrayGreen[n][0], arrayGreen[n][1],
                arrayGreen[n][2], arrayGreen[n][3]);
            }
          });

        }
        {
        final Group Misc = new Group(composite, SWT.NONE);
        final GridLayout gridLayout_1 = new GridLayout();
```

```
            gridLayout_1.numColumns = 2;
            Misc.setLayout(gridLayout_1);
            GridData MiscLData = new GridData(GridData.FILL, GridData.
                FILL, true, false, 1, 2);
            Misc.setLayoutData(MiscLData);
            Misc.setText("Miscellaneous");
            final Label currentTime = new Label(Misc,SWT.NONE);
            currentTime.setText("Current time");
            time2 = new Label(Misc,SWT.BORDER);
            time2.setText(myTime);
            final Label timeElapsed = new Label(Misc,SWT.NONE);
            timeElapsed.setText("Time elapsed");
            final Label time3 = new Label(Misc,SWT.BORDER);
            time3.setText("00:00");
            final Label batteryLabel = new Label(Misc, SWT.NONE);
            batteryLabel.setText("Battery %");
            Text text = new Text(Misc, SWT.BORDER);
            text.setLayoutData(new GridData(GridData.BEGINNING, GridData.
                CENTER, true, false));
            text.setText("97");
            text.setEditable(false);
            final Label BatteryVoltageIndicator = new Label(Misc, SWT.
                NONE);
            BatteryVoltageIndicator.setText("Battery Voltage indicator: "
                );
            final ProgressBar progressBar = new ProgressBar(Misc, SWT.
                NONE);
            progressBar.setSelection(95);
            progressBar.setLayoutData(new GridData(GridData.BEGINNING,
                GridData.CENTER, false, false, 3, 1));
        }
      }

    }
    public static void main(String[] args) {
      TestFrame TF = new TestFrame();
      CommPC commpc = new CommPC(TF);
      commpc.start();
      Camera cam = new Camera(TF);
      cam.start();
      myClock mc = new myClock(TF);
      mc.start();
      TF.run();
    }
}
```

Listing B.3: Multi-Page Java code for Camera unit

```
/* Camera.java
   class for detection of victims in the frames obtained from camera
*/
import java.util.Vector;

import detect.Detect;
import detect.DetectedObject;

public class Camera extends Thread {
     TestFrame tf;
  public Camera(TestFrame testframe) {
     tf = testframe;
          }
  public void run()
  {
       int i, j;
       //create detect object
            Detect d = new Detect("\"c:\\program files\\Internet
                Explorer\\iexplore.exe\" D:\\Usario2.4\\support\\detect
                \\camera.html",
                                    "X - Microsoft Internet Explorer",
                                        10, 198, 116, 83);
            //setup detection parameters:
            //max. number of objects, size of raster, required number
                of color pixels in raster, rasters required for object
            d.setparam(400, 10, 33, 10);
            //setup red color detection: relative1B, relative1G,
                otherwise required R, relative2B, relative2G
            d.redparam((float)2.0, (float)2.0, (float)140.0, (float)
                1.4, (float)1.4);
            //setup green color detection: relative1B, relative1R,
                otherwise required G, relative2B, relative2R
            d.greenparam((float)2.0, (float)2.0, (float)100.0, (float)
                1.0, (float)1.5);

            d.visualize(Detect.IMAGE_WITH_FRAME);
            Thread t = new Thread(d);
            t.start();
            int greencandidate = 0;
            int redcandidate = 0;
            for (i = 0; i < 10000; i++)
            {
       Vector a;
       a = d.detect();
```

```
            int seegreen = 0;
            int seered = 0;
        for (j = 0; j < a.size(); j++)
        {
          if ( ((((DetectedObject)a.elementAt(j)).minx) >= 90) && ((((
              DetectedObject)a.elementAt(j)).maxx) <= 250 ) ){

            if (((DetectedObject)a.elementAt(j)).type == DetectedObject
                .GREEN)
              tf.objectDetected = 2;

            else tf.objectDetected = 1;
          }
                if ((((DetectedObject)a.elementAt(j)).type ==
                    DetectedObject.GREEN) && (((DetectedObject)a.
                    elementAt(j)).count > 200))
                {
            seegreen = 1;
          }
          if ((((DetectedObject)a.elementAt(j)).type == DetectedObject.
              RED) && (((DetectedObject)a.elementAt(j)).count > 200))
          {
              seered = 1;
            }
          }
              if (seegreen == 1) greencandidate++;
              else greencandidate=0;
        if (seered == 1) redcandidate++;
        else redcandidate=0;
        }
            System.out.println("finalizing\n");
            //release memory at the end of the program (optional)
        d.destructor();
        d = null;
    }
}
```

Listing B.4: Multi-Page Java Code for commPC

```

/* thread for communication between PC and Usario */

import org.eclipse.swt.widgets.Display;

class CommPC extends Thread{
  TestFrame tf;
  private byte[] packet;  //outgoing packet
  private byte[] answer;  //received packet
  private int len;  //will contain the length of received packet
  //private final int ROTATE = 10;
  private final int STOP = 1;
  private final int FORWARD = 3;
  private final int BACKWARD = 4;
  private final int ROTATE_RIGHT = 5;
  private final int ROTATE_LEFT = 6;

  public CommPC(TestFrame testframe) {
    packet = new byte[1];
    answer = new byte[5];
    tf = testframe;
        }
    public void run() {
    SimpleCommPC comm = new SimpleCommPC("COM7", (byte)0, false);
      try {
            try { Thread.sleep(5000); } catch (Exception e) {}
            while (true)
            {
              //send packet to RCX
                if (answer[4] == FORWARD || answer[4] == BACKWARD ||
                    answer[4] == ROTATE_LEFT || answer[4] ==
                   ROTATE_RIGHT) {
                  tf.command = STOP;
                  packet[0] = (byte)STOP;
                } else {
                  packet[0] = (byte)tf.command;
                }
              System.out.println("Command is: "+tf.command);
              comm.send(packet, 1);
              print("sent command packet:", packet, 1);
              //wait for new packet from RCX and store it to array
                 answer
              //System.out.println("hello");
              do {
                len = comm.receive(answer);
             } while (len <= 0);
                tf.ArrayXcoords[tf.arrayIndex] = (int)(answer[1]*0.75)
                   ;
                tf.ArrayYcoords[tf.arrayIndex] = (int)(50 - (answer
                   [2]*0.75));
                tf.angle = (double)answer[3]*45;
                //here answer[3] contains angle encoded as 0=NORTH, 1=
                   NE, ..., 7=NW
                tf.arrayIndex++;
                print("received packet: ", answer, len);
                //repaint the map on the screen
                Display.getDefault().asyncExec(new Runnable(){
                    public void run() {
                      tf.canvas.redraw();
                    }
                 });
             try { Thread.sleep(1000); } catch (Exception e) {}
          }
       } catch (Exception e) { System.out.println("Exception" + e);
             e.printStackTrace(); }
       comm.close();
   }
   private static void print(String title, byte[] p, int len)
   {
     int counterRotate=-1;
     String action="";
      System.out.print(title);
      for (int i = 0; i < len; i++) {
             System.out.print(p[i] + " ");
             }
      if (title.equalsIgnoreCase("received packet: ")) {
        switch (p[4]) {
         case 0 : action = "AUTONOMOUS";
           break;
         case 1 : action = "STOP";
           break;
        case 2 : action = "MOVE";
           break;
        case 3 : action = "FORWARD";
           break;
        case 4 : action = "BACKWARD";
           break;
        case 5 : action = "ROTATE_RIGHT";
           break;
        case 6 : action = "ROTATE_LEFT";
```

```
                break;
            case 7 : action = "DECIDE";
                break;
            case 8 : action = "SCAN";
                break;
            case 9 : action = "RUN";
                break;
            case 10 : action = "ROTATE";counterRotate++;
                break;
            case 11 : action = "BACKTOBASE";
                break;
             }// end switch
            if (counterRotate == -1){
                 System.out.println(" "+ action);
            }
            else {
                 System.out.println(" "+ action +" (rotation number: "+
                     counterRotate+" )");
            }
         } // end outer if
         System.out.println(" ");
     } // end method print()
}
```

Bibliography

[1] AI group IDI, NTNU. *AI Group - Eval.* Available from http://www.idi.ntnu.no/grupper/ai/eval/software.html Accessed 07.11.05

[2] B.Bagnall. *Core Lego Mindstorms programming* Prentice Hall, 2002.

[3] BBC news. *London attacks* Available from http://news.bbc.co.uk/1/shared/spl/hi/uk/05/london_blasts/html/default.stm Accessed 11.09.2005

[4] BBC news. *11 September* Available from http://news.bbc.co.uk/cbbcnews/hi/find_out/guides/newsid_2209000/2209407.stm Accessed 11.09.2005

[5] BBC news. *Bali bomb attack* Available from http://news.bbc.co.uk/1/hi/world/asia-pacific/4300274.stm Accessed 11.09.2005

[6] BBC news. *Terror attack in Egypt* Available from http://www.somethingjewish.co.uk/articles/1216_terror_attack_in_egy.htm Accessed 11.09.2005

[7] Carnegie Mellon press *Carnegie Mellon press release* Available from http://www.cmu.edu/PR/releases05/050210_marines.html Accessed 30.09.2005

[8] G. Dudek, M. Jenkin *Computational principles of mobile robotics* Cambridge University Press, 2000.

[9] Dlink. *Wireless Security camera-DCS-5300G* Available from http://www.dlink.com/products/?pid=342 Accessed 07.11.05

[10] DLink. *DCS5300G* Available from ftp://ftp10.dlink.com/pdfs/products/DCS-5300G/DCS-5300G_ds.pdf Accessed 11.09.2005

[11] G.Ferrari. *Lego Mindstorms with Java* Syngress Media, 2002.

[12] J.P.F. Friquin.*Identifying the risks involved in the design of a safety-critical system for an Urban Search and Rescue robot* Available from http://www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2005/friquin-fordyp05.pdf. IDI, NTNU. 2005.

[13] B. Gates, N. Myhrvold and P. Rinearson. *Bill Gates: The Road Ahead* Penguin, 1996

[14] T. Hatton. *SWT: A Developer's Notebook* O'reilly, 2004

[15] Inuktun Services Ltd. *Inuktun Services* Available from `http://www.inuktun.com/` Accessed 27.09.2005

[16] iRobots inc. *Robots for the real-world.* Available from `http://www.irobot.com` accessed 27.09.2005

[17] J. Knudsen. *Imaginations run wild with Java Lego robots* Available from `http://www.javaworld.com/javaworld/jw-02-2001/jw-0209-lejos_p.html` accessed 15.09.2005.

[18] Kobe University. *Utility Vehicle for Search version IV* Available from `http://www.rescuesystem.org/robocuprescue/UVS.pdf` Accessed 30.09.2005

[19] Lego inc. *Lego Mindstorms home* Available from `http://mindstorms.lego.com/eng/default.asp` accessed 15.09.2005.

[20] J. Lewis, W. Loftus. *Java Software Solutions; Foundations of program design.* Addison-Wesley, 1998

[21] `http://www.lugnet.com` accessed August 15.09.2005.

[22] NASA. *Urbie- Urban Robot project* Available from `http://robotics.jpl.nasa.gov/tasks/tmr/homepage.html` accessed 27.09.2005

[23] G.W. Lucas*http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html* Accessed, 27.01.2006

[24] R.R. Murphy. *Introduction to AI robotics* MIT press, 2000.

[25] National geographic News. *Search-and-Rescue Robots Tested at New York Disaster Site* Available from `http://news.nationalgeographic.com/news/2001/09/0914_TVdisasterrobot.html` Accessed 27.09.2005

[26] C. Perrow. *Normal Accidents: Living with high-risk technologies*Princeton University Press,1999.

[27] P. Petrovic. *Pavel Petrovic homepage* Available from `http://www.idi.ntnu.no/~petrovic/ecc_tekrep.pdf` Accessed 25.03.2006

[28] P. Petrovic. *Wireless Communication with RCX* Available from `http://www.robotika.sk/projects/rcxbt/` Accessed 14.04.2006

[29] RoboCup. *Robocup official site* Available from `http://www.robocup.org/` Accessed 27.09.2005

[30] RoboCup Rescue. *RoboCup Rescue official page* Available from `http://www.rescuesystem.org/robocuprescue/` Accessed 27.09.2005

[31] RoboCupJunior. *RoboCup Junior official page* Available from `http://www.robocup.org/junior/index.html` 27.09.2005

[32] *leJOS* Available from `http://lejos.sourceforge.net/` accessed August 15.09.2005.

[33] *leJOS API* Available from `http://lejos.sourceforge.net/apidocs/index.html` accessed August 2005

[34] IUI 04 - 2004 International conference on Intelligent User Interface. *Where to look: A study of human-robot Engagement by C.L. Sidner, C.D. Kidd, C. Lee and N.Lesh* Association for computing machinery, 2004

[35] sourceforge.net: OpenCV library *Open computer vision library* Available from `http://sourceforge.net/projects/opencvlibrary/` Accessed 07.10.05

[36] SparkFun electronics. *BlueSMiRF V1* Available from `http://www.sparkfun.com/commerce/product_info.php?products_id=582` Accessed15.03.2006

[37] Trendnet. *Trendnet Bluetooth TBW-102UB* Available from `http://www.trendnet.com/en/products/TBW-102UB_o_1.htm` Accessed 26.03.2006

[38] BAE systems. *BAE systems* Available from `http://www.uniteddefense.com/` Accessed 30.09.5002

Index