



**DUBLIN INSTITUTE  
of TECHNOLOGY**

*Institiúid Teicneolaíochta Bhaile Átha Cliath*

# **Robotic Lecturing Assistant Final Year Project Report**

**DT228**

**BSc in Computer Science**

**Jinwu Li**

**Supervisor: Ken O'Brien**

**Second Reader: Paul Bourke**

School of Computing  
Dublin Institute of Technology

**22/03/2012**

# Abstract

Researchers' passion for automatic robots has never died out in even when AI suffering its hardest times. To name some, NASA's Mars Rover has made a breakthrough in the planet exploring, several library robot guides perform really well doing their job in 1970s. In this project, the author tries to investigate a way to use a robotic lecturing assistant in a college environment.

The objective of this project is to build a robust LEGO robot that can build a map of the environment, find each classroom based on the map and user input, and point the projector correctly on the whiteboard in correct room. Multiple sensors like light, ultrasonic, camera will be used in order to achieve mapping, navigation, obstacle avoidance, and object recognition.

The software development environment will be LEJOS, and ARToolKit library will be used for marker recognition. The program will be primarily written in Java, but the marker recognition part will be written in C.

# Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

---

Jinwu Li

22/03/2011

# Acknowledgements

First, I would like to thank Dr. Brian Mac Namee, this project is based on his idea for Imagine Cup, and he helped me a lot when I was writing my project proposal.

I also would like to express my appreciation to my supervisor, Ken O'Brien, he guides me through the entire development of the project over project management, resource request. And his deep insight in the internal system always push me on the right track.

And John Kelleher and his PhD student Yan Li, they help me a lot in the problems I encountered during the development.

LEJOS Forum is a good place to discuss programming Lego with Java, from where I learn how to implementation advance algorithm for localization and mapping.

## Table of Contents

Table of figures .....	8
Chapter 1.....	9
Introduction .....	9
1.1    Background.....	9
1.2    Objectives.....	10
1.3    Problem Definition .....	10
1.4    Structure of Report .....	10
Chapter 2.....	12
Technologies Researched .....	12
2.1    Introduction.....	12
2.2    A survey on available robots .....	12
2.3    Software development environment(Platform & IDE) .....	12
2.3.1    Comparison of popular LEGO programming environment.....	12
2.3.2    LEJOS overview .....	14
2.4    ARToolKit for AR Marker recognition.....	14
2.4.1    How ARToolKit works.....	14
2.4.2    Program Structure .....	15
2.5    Simultaneous localization and mapping(SLAM).....	15
2.5.1    Fundamental goal .....	15
2.5.2    Available algorithms .....	15
Chapter 3.....	16
Design.....	16
3.1    System Architecture .....	16
3.2    Design Methodology .....	16
3.2.1    Selection of suitable methodology .....	16
3.2.2    Feature-Driven Development .....	17
3.3    Features List .....	17
3.3.1    Robot navigation.....	17
3.3.2    Object recognition .....	18
3.3.3    Projection on the whiteboard.....	18
3.4    Design of Each Feature.....	18
3.4.1    High level overview .....	18
3.4.2    User Interface .....	23
3.4.3    Information flow .....	24

3.5	Design of the robot .....	25
3.6	Source code layout.....	25
3.7	Design Patterns .....	25
3.7.1	Singleton Pattern .....	25
3.7.2	Model View Controller(MVC) Pattern .....	26
Chapter 4.....		28
Development.....		28
4.1	Procedure Description.....	28
4.2	Source code structure .....	31
4.3	Development details .....	32
4.3.1	Fundamental function.....	32
4.3.2	Robot navigation.....	37
4.3.3	Object recognition .....	47
4.3.4	Socket communication .....	50
4.3.5	Projection on the whiteboard.....	51
4.4	Physical development .....	51
Chapter 5.....		52
System Validation .....		52
5.1	Testing method .....	52
5.1.1	Black box testing .....	52
5.1.2	Reason for choosing Black box testing .....	52
5.2	Testing plan .....	53
5.3	Testing and results .....	53
5.3.1	Test build map function .....	53
5.3.2	Test path finding function.....	54
5.3.3	Test color tracking function .....	56
5.3.4	Test marker recognition function .....	56
5.3.5	Test projection on the whiteboard function .....	59
5.3.6	Summary .....	60
Chapter 6.....		61
Project Plan .....		61
6.1	Previous Plan .....	61
6.2	Actual progress.....	61
6.3	Adaption Analysis .....	62
Chapter 7.....		64

Conclusion.....	64
7.1 Key challenge.....	64
7.1.1 Unreliable motion and sensing.....	64
7.1.2 Construct working robot with limited hardware.....	64
7.1.3 Multi-threading, multi-component software design.....	64
7.2 Key learning.....	65
7.2.1 Probabilistic Robotics.....	65
7.2.2 Marker recognition .....	65
7.2.3 Robotic Programming .....	65
7.3 Reflection .....	65
7.3.1 Critical thought .....	65
7.3.2 What if do it next time .....	66
7.4 Future work.....	66
7.4.1 Global localization.....	66
7.4.2 Robot guide in an indoor environment .....	66
7.4.3 Implementation on a more reliable, suitable robot .....	67
7.5 Summary .....	67
References .....	68
Appendix .....	69
User Manual.....	69
Run the code .....	69
The user interface .....	69

## Table of figures

FIGURE: SYSTEM ARCHITECTURE .....	16
FIGURE: MAP BUILDING PC SIDE CLASS DIAGRAM .....	19
FIGURE: MAP BUILDING ROBOT SIDE CLASS DIAGRAM .....	20
FIGURE: PATH FINDING CLASS DIAGRAM .....	21
FIGURE: COLOR TRACKING CLASS DIAGRAM .....	21
FIGURE: MARKER RECOGNITION AND ROBOT POSITION CONTROLLER CLASS DIAGRAM .....	22
FIGURE: COMPLETE CLASS DIAGRAM.....	23
FIGURE: A SAMPLE USER INTERFACE SNAPSHOT.....	24
FIGURE: INFORMATION FLOW DIAGRAM.....	25
FIGURE: PATTERN ON THE SERVER SIDE .....	27
FIGURE: ADAPTED PATTERN ON THE ROBOT SIDE.....	27
FIGURE: USING MOUSE TO CONTROL THE ROBOT MOVEMENT .....	28
FIGURE: TARGET(RED DOT), START POSITION(BLACK DOT) AND A LIST OF WAYPOINTS(BLUE DOTS) .....	29
FIGURE: ARTOOLKIT COORDINATE SYSTEMS(CAMERA AND MARKER) .....	48
FIGURE: THE ROBOT, FINAL VERSION .....	52
FIGURE: BLACK BOX TESTING .....	52
FIGURE: A MAP OF WOODEN-FLOOR CORRIDOR: .....	54
FIGURE: TEST THE PATH FINDING PROGRAM WITH A COMPLICATED PATH .....	55
FIGURE: MARKER RECOGNITION IN A BRIGHT ENVIRONMENT, AND RESULTING COORDINATE VALUES.....	56
FIGURE: RECOGNIZE MARKER SUCCESS, IN A DARK ENVIRONMENT, WITH COORDINATE VALUE .....	57
FIGURE: CANNOT FIND MARKER IN A DARK ENVIRONMENT .....	57
FIGURE: RECOGNIZE A MARKER IN A NORMAL LIGHT CONDITION, AND RESULTING COORDINATE VALUES.....	58
FIGURE: BEFORE AND AFTER ADJUST ANGLE .....	59
FIGURE: BEFORE AND AFTER ADJUST ANGLE .....	59
FIGURE: THE USER INTERFACE .....	69
FIGURE: CONTROLS.....	70
FIGURE: USER INTERFACE AFTER SELECT SOME TARGETS .....	70
FIGURE: A LIST OF WAYPOINTS TO TARGET 1 .....	71
FIGURE: MODE SELECTION AND ANGLE SELECTION .....	71



## Chapter 1

### Introduction

In many educational institutions, mounted projection systems are not always present in the lecture room. This is a challenge for every lecturer, especially for those with a physical disability. This project attempts to develop a mobile robot that can mount a projector on its top, so it can meet a lecturer in a lecture room and position itself at the correct place for displaying the projection image on a whiteboard.

#### 1.1 Background

Lego Mindstorms is a well-known robotics system that provides a wide range of functionality that suits for this project:

- The color sensor enables the robot to recognize the color line, so that it can navigate an entire floor by following a preset instruction line.
- Motors: Give robots ability to move forward and backward. it's essential to this project
- Wireless Transmission: The computing power of Lego Mindstorms is quite limit, so it will be a bottleneck for instant image processing. Luckily we can use Bluetooth to process image remotely and return the resultant instruction stream to the robot.

Image processing techniques enable me to analyze the specific marker images and calculate some mathematic parameters to project properly.

Also, Microsoft Robotics Developer Studio provides a handy .NET like API to control Lego Mindstorms, and LEJOS provides a Java option to manipulate it, so it should be straightforward to program with Lego Mindstorms.

Related research was done by others and can be found in Google/ Microsoft academic search. For example, [1] reviewed various techniques in mobile robot navigation, both indoor and outdoor; [2] discussed how to learn to reach a color object with a color line sensor(even neural network was applied in their publication); [3] explain discussed about object recognition and cognitive map building in a domestic environment using SLAM.

And some similar project can be found on YouTube or other websites which prove that the robot has the ability to do route planning, WIFI connecting, and object tracing.

## 1.2 Objectives

The objective of this project is to build a robust robot that can go to each classroom to setup the projector for a lecturer who will then use cable to connect their laptop with the projector.

So a range of technologies will be researched to achieve the goal:

- Research on multiple sensors' usages to achieve robot mapping and navigation.
- Research on object recognition techniques by using AR marker library.
- Research on robot control, including moving, sensing and rotating robot arm.

## 1.3 Problem Definition

The aim of this project is to build a fully autonomous robot that perform as “an intelligent projector” with ability to navigate all classrooms, avoid obstacles along its way, recognize the whiteboard in the room and focus the projection onto the whiteboard.

According to introduction chapter, it's easy to divide the problem into some sub-problems, and these are main challenges in implementation step.

- Be able to build a map of the whole floor, navigate within the floor, and locate itself. (By particle filters and Markov decision train)
- Can find out the target white board that it needs to project to, and project correctly.(Using camera to detect AR markers)
- Be able to differentiate each classroom on the map that it build by itself.
- Can change the target room number, time to arrive, etc.(Using GUI)
- Its projector can connect to lecturer's laptop by cable.

## 1.4 Structure of Report

The report is organized into seven chapters as follows:

Chapter 2: Discusses the related technologies (software development environment, third party APIs, research papers) that will be related to this project, including some alternative technologies. Also, reasons for choosing them are provided.

Chapter 3: Describes the details of the design methodology chosen, and the reasons for choosing it. A list of simplified version of class diagrams will be given to discuss the design of each component.

Chapter 4: Describes the development details. Separates them in different modules based on the methodology given in Chapter 3. Discusses the problem encountered, and workaround.

Chapter 5: Gives the testing method and testing results.

Chapter 6: Lists the plan shift from interim report, and discusses the reasons.

Chapter 7: Discusses Key challenge, key learning, and future work. And make a conclusion about this project.

Appendix: A user manual about how to use the software.

## **Chapter 2**

### **Technologies Researched**

#### **2.1 Introduction**

In this chapter, all the research the author has done will be listed, together with their relation to achieving the goal of the project, their functions, advantages and disadvantages. Also, reasons for choosing them or not will be provided.

#### **2.2 A survey on available robots**

Robots that are currently available in School of Computing are:

Lego Mindstorms, Evolution scorpion, The Peoplebot, Hitec Robonova. The summary of the features and drawback of each robot is listed below:

- Lego Mindstorms: with multiple sensors, can recognize color, light density, distance detection, and move around by differential drive. It's easy to build a customize a robot with specific functionality. It is widely used in high schools and colleges as an introduction tool to robotics. But it cannot really "see" things, if no additional camera provided, instead, it can "understand" the color change and light density change.
- Evolution Scorpion: This is good at robot navigation(map building, path planning), vision based robot with a video camera. It's good for robot navigation, especially if using SLAM for localization[4]. However, it's API is complaint by many programmers for its difficulty to use, and it is hard to find tutorials online.
- The Peoplebot: The height of it is 104 cm, so it's better for the purpose of projection, as well, it is equipped with multiple sensors, it can follow color, avoid obstacle and so on. But it's too big, and expensive, the author cannot get enough time for using it.
- Hitec Robonova: The humanoid robot provides a stable walk, run, and dance ability, but it's not use for image processing and navigation.

Finally, the LEGO NXT was chosen as the robot being used in this project, because it's cheap, easy to build and program, and many tutorials are available online.

#### **2.3 Software development environment(Platform & IDE)**

##### **2.3.1 Comparison of popular LEGO programming environment**

The robot used in this project is LEGO NXT, it's a mature product of LEGO company with a vast choice of development environments, competitive candidates were listed below[5]:

	NXT Educational	LEJOS	Microsoft Robotics Development Studio(MSRDS)	RobotC
IDE	Yes	Eclipse	Visual Programming Language + Visual studio	Yes
Language	Graphics	Java	C#/Graphics	C
Simulation	No	Yes (with plug-in)	Yes	Yes (Latest version)
Target users	Students, Kids	Java Programmers	.NET programmers	Applications that need Maximum Performance
Cost	Free with NXT Kit	Free	Free for non-commercial purpose	\$49 one year license

Table: Comparison of different software development environment

NXT educational: It's an "official" programming environment provided by LEGO company bundled with the NXT. With only iconic programming interface, it's really good to demonstrate the essence of LEGO robot. But as my project needs some advanced features, the graphical type control might be inadequate.

LEJOS: An excellent cross-platform Java programming environment. It can compile the code and download it into LEGO intelligent brick. With full API documentation and step by step tutorial, it's easy to get started and write complicated code. By using eclipse and its plug-ins, it also can be used for simulation.

MSRDS: Along with Microsoft Visual Studio, it can write code to control different robot easily, i.e., it's cross-robot, service based development environment. Also, it support high concurrency based control, visual simulation environment and C# extended support. But the only drawback is that it can only send command via Bluetooth, but that should not be a problem in this case.

RobotC: It uses a C-Based programming language, so the performance is really reliable. Another good feature about RobotC is its runtime debugging, it allow programmer to set break points and find out problems at runtime.

With all the table and descriptions above, it is evident that NXT educational is too simple to finish some sophisticated tasks, while RobotC's C-like program might be messy for a big project and it is expensive to use. So LEJOS and MSRDS are two most suitable APIs to use.

So, the author has decided to use LEJOS as the development environment. The main reasons are:

- There's some known bugs in MSRDS that lead to sensor no response.
- The MSRDS simulator is not compatible with Windows 64 bit OS.
- LEJOS' simplified API for sophisticated task such as localization and mapping.
- LEJOS supports both downloading code to the robot and remote control.

### 2.3.2 LEJOS overview

LEJOS is a lightweight Java Virtual Machine that can compile java code and uploaded the binary to LEGO robot. And it also supports Bluetooth connection.

According to its official website[6], here are the main features that LEJOS has:

- Pure Java language, include all advantages bring by Java.
- Well-documented Robotics API.
- Step-by-step tutorials.
- Full support of Bluetooth, USB, thus, it supports programs that runs partly on the PC and partly on the NXT. So people can get the efficiency from the NXT local program whilst get more processing power can memory from a PC.
- Simplified robot localization and navigation model.

## 2.4 ARToolKit for AR Marker recognition

ARToolKit is a widely used API for creation of augmented reality(AR) [7], the programming language it's using is C. It uses camera to track markers in real time, the main features of ARToolKit are:

- A multiplatform library(Windows, Linux, Mac, etc.)
- Single camera for marker tracking
- Free and open source
- With graphic and video library
- Many APIs are built based on ARToolKit for enhancement purpose, so understand this one helps to move on using others.

### 2.4.1 How ARToolKit works

The basic idea of ARToolKit application is to make virtual 3D models can overlap over designed markers in live video of the real world. The mechanism can be described in 6 steps:

1. The camera records video and sends it to the computer.
2. The program running on the computer find a specific shape in each video frame.
3. Once the pattern is found, some mathematics were applied to calculate the position of the camera related to the pattern.
4. A 3D model will be drawn in the same position.

5. The model is drawn on top of the video and it will appears stuck on the marker.
6. Output the result to the user.

### 2.4.2 Program Structure

The ARToolKit Documentation specified a standard application code[8]:

1. Initialization: Capture video stream, initialize camera parameters, read in the marker pattern files.
2. Main Loop: Grab a video frame, detect the markers, calculate the camera position related to the marker, and draw the virtual objects on the detected patterns.
3. Shutdown: garbage collection, close the program

## 2.5 Simultaneous localization and mapping(SLAM)

### 2.5.1 Fundamental goal

What SLAM is trying to achieve is, under an unknown noisy environment, given a list of robot motions and sensor measurements, to "build a consistent map of this environment while simultaneously determining its location within this map" [9]. It was regarded as a final solution to make a robot fully automated.

### 2.5.2 Available algorithms

Various techniques were introduced in [10], here are several most important filters:

- Bayes Filter: a general approach "to estimate the state of an environment", it can be used in a dynamic environment to decide whether a specific position has an obstacle. For more formal definition and implementation see [section 4.3.2](#).
- Kalman Filter: it computes belief for continuous states[11], was used to give best estimation of other vehicles velocity and position. It's an crucial filter in probabilistic robotics, but it might not suitable in this project because we assume the environment is static and no other vehicle involve in the entire process.
- Particle Filter: The key idea of particle filter is to represent the posterior belief state by a set of random generated samples(called particles). Every time the robot takes a new motion and measurement, the algorithm will calculate important factor for each particle based on the measurement. That is, perceive a measurement for each particle, for particles that gets similar measurement as the robot perceived, their important factor will go higher, which means this random particle is more likely be the real time robot pose.

## Chapter 3 Design

### 3.1 System Architecture

Since this project need image processing and automate localization, Bluetooth will be used for remote control the robot so that massive calculation happens on the PC to improve the performance.

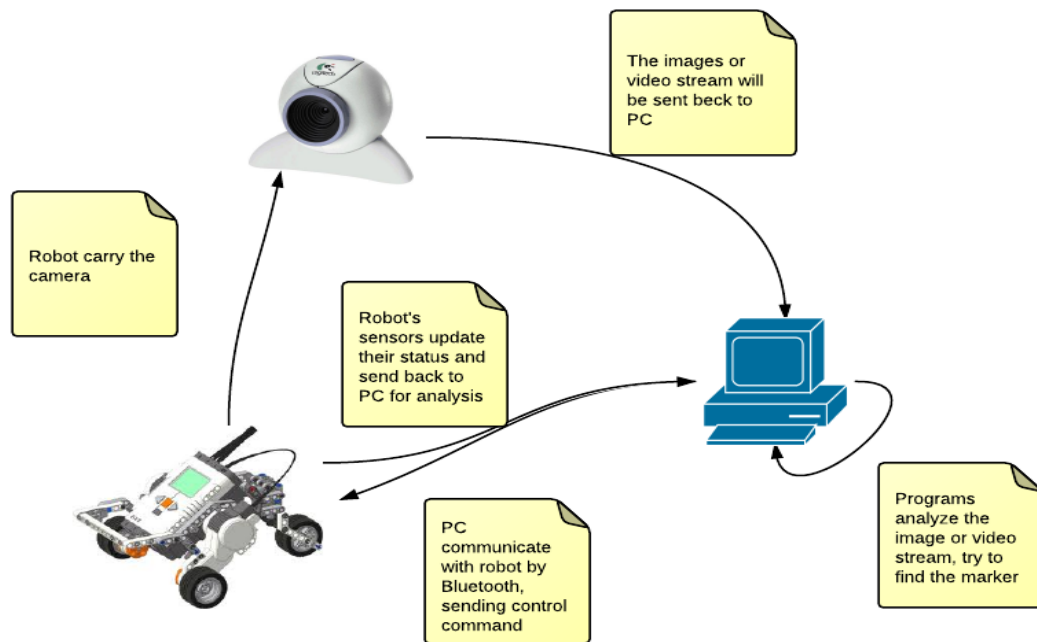


Figure: System Architecture

The architecture being used is a kind of Client/Server Structure. The webcam is mounted on the robot, and send the images it takes to PC via cable(because of the limitation of hardware, the author cannot find a good wireless webcam at the moment). After PC finish processing, then PC will sends control command to the robot. So, not roughly, Webcam, Robot act as “client”, and PC act as server.

### 3.2 Design Methodology

#### 3.2.1 Selection of suitable methodology

Based on the problem definition, it might seem, at first, the Iterative and Incremental development method is needed for fulfilling all the requirements, since the project has a few features that need to be implemented one by one. Therefore, intuitively, the traditional method, the Iterative and incremental development might be a good methodology to choose.

However, if look into the methodology deeply [12], it's easy to find that the methodology is not quite fit well with the project requirement, because the project



can be divided into three modules, but the order of the implementation is not important, and the three modules are sequential. According to the definition of the Iterative and Incremental development, it has to be some intersections in each iterative stage, so it's not the right one for this project.

### 3.2.2 Feature-Driven Development

Generally, FDD has five basic activities[13], but it was used by a small team originally, so the methodology has been adopted to fit the requirement:

1. Develop Overall Model: Firstly, a high-level walkthrough of the scope of the system and its context is required, and then follow by detailed domain walkthrough for each area that need to be modelled. After research and analysis, select one model for each specify domain. Finally, all these models consist of the overall model.
2. Build Feature List: Based on initial knowledge that has been gathered, a list of features will be identified, which is done by functionally partitioning the domain into subject areas. Each area is further broken down into a number of activities. Each step with an activity is called feature.
3. Plan By Feature: As feature list is presented, a development plan is needed to decide the order that the features are to be implemented(base on complexity, dependencies, and so on).
4. Design By Feature: Work out detailed sequence diagrams for each feature, and refine the overall model; Write class and method prologues; Hold a design inspection.
5. Build by Feature: Writing code to implement the feature, after unit test and successful code inspection, the completed code is promoted to the build.

## 3.3 Features List

The system can be divided into three steps, which correspond to three main modules, namely, Robot navigation, Object recognition, and Projection. Each module might be divided into sub-modules based on its internal complexity. In this chapter, the author simply list all the features that have to be done to build the system, [section 4.3](#) will show the detailed implementation.

### 3.3.1 Robot navigation

#### *Robot mapping*

In this project, we need to build a map for the environment, so that the system can use the map to distinguish from corridor and each classroom.

### *Path finding*

Assuming we build a map with static obstacles, we need to find a valid path from start position to the classroom we want the robot go to, so path finding is an important feature that we definitely need to implement.

### *Color tracking*

When the robot enter the classroom, it need to find the whiteboard position. In this project, the author decides to use color tape to guide the robot to the correct place where the robot can project things on the whiteboard.

#### **3.3.2 Object recognition**

##### *Marker recognition & orientation*

When the robot go to the robot position following the color track, it still does not know which orientation it should face. We can tag a marker just below the whiteboard, so when the robot is rotating, it can “know” whether it’s facing correct direction immediately by our marker recognition program.

#### **3.3.3 Projection on the whiteboard**

It’s trivial in theory, but when in robotics, it’s not that easy. We can store information(the height of the whiteboard, etc) in the marker, and once the robot points directly to the marker, we can get the distance to the wall by sonar sensor, and get the height of the whiteboard by analyze the marker, so that the robot can raise the projector properly.

### **3.4 Design of Each Feature**

#### **3.4.1 High level overview**

In this section, the author mix the use of “feature”, “component” and “module”, They are actually referred to the same thing: Feature. A full list of features, can be found at [section 3.3](#).

##### *Robot Navigation Module:*

#### **1. Map building program, sender(PC side):**

Description: At the beginning, the robot starts with an empty map. The user can “drive” the robot via the graphical user interface. When the robot is moving, it also sends back the ultrasonic sensor readings, the program build a map based on these readings.

A class diagram prototype for server side map building functionality:

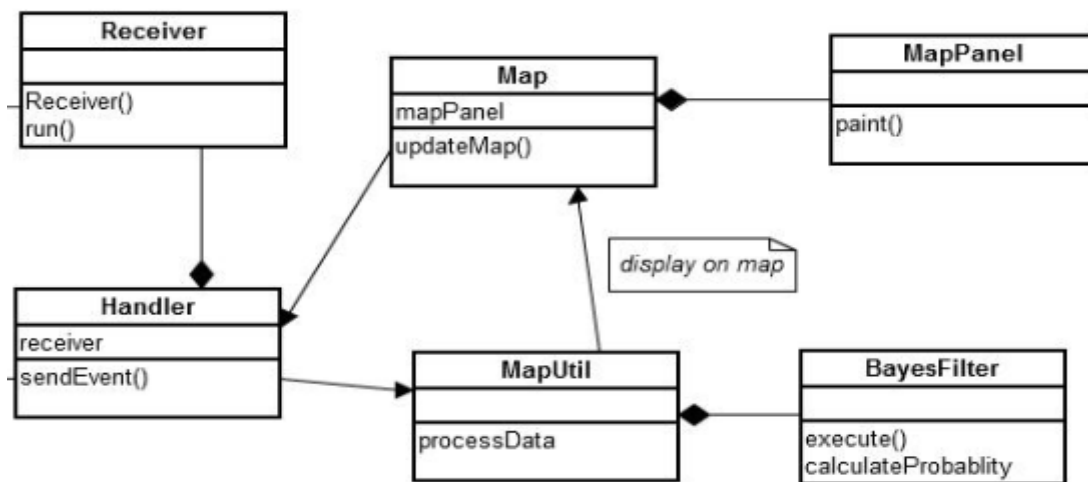


Figure: Map Building PC side Class Diagram

As we can see from the diagram. The program follow the [MVC architecture pattern\(see section 3.7.2\)](#) strictly. The handler is the “Controller”, it’s member, the Receiver takes the responsibility of receiving messages from the client, and pass on to the “Model”, the MapUtil class, the model perform some actions base on the information, which results in the change in the internal model state. After the model state change, the model notify the “View”, which is Map class in this case, to display the information on the screen. Each time the “View” capture an user input, it pass along to the “Controller”, the control will send the control message to the robot client.

## 2. Map building program, receiver(robot side):

Description: The robot receive the message sent by PC server, it has to perform corresponding tasks.

A class diagram prototype for client side map building functionality:

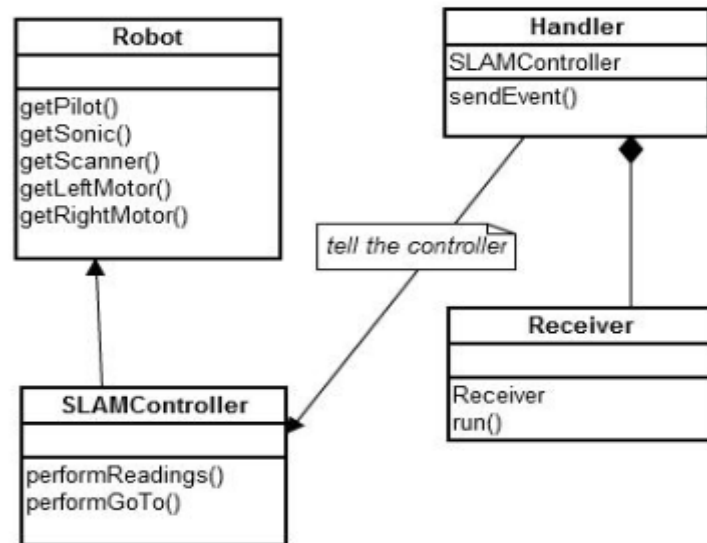


Figure: Map Building Robot side Class Diagram

The robot side program is much simpler than the server(PC) side program, a Receiver class (a member of Handler class) is responsible for receiving command sent by the server, and it calls the SLAMController to perform specific tasks. The SLAMController is an instance that contains algorithm for controlling the robot when building a map and following a path, and it relies on the Robot class to perform actions, since the Robot class encapsulates all the actions that a robot can perform. After the action is finished, the SLAMController tells the Handler to send corresponding message back to the server.

### 3. Path finding(PC side):

Description: With the map we build by programs mentioned above, we need some functions to calculate a list of waypoints that can lead the robot to the destination. When the program performs "follow path" command after the path finding algorithm, we can reuse the "map building program" to control the robot. The difference is that in Map Building stage, we calculate one waypoint at a time from user input, in Path Following stage, we get a list of waypoints that are generated by Path Finding algorithm.

A class diagram prototype for path finding program:

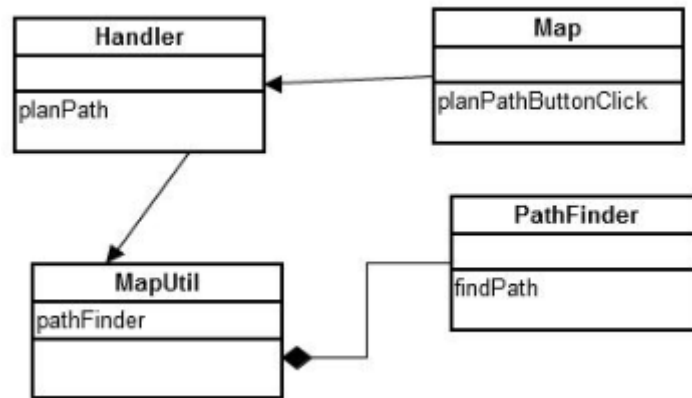


Figure: Path Finding Class Diagram

The server side program follows the MVC paradigm, so when the View(Map class) receive the button click event, it call the Controller(Handler class) first. Then the Control decide to call the Model(MapUtil) class to perform path finding algorithm. Then the result, a list of waypoints was returned to the view for displaying.

#### 4. Color tracking

Description: When the path finding program find a way to the entrance of the room, the color tape that guides the robot to the whiteboard is right there. This program is used to describe the design of the color tracking functionality.

A simplified version of Color Tracking class diagram:

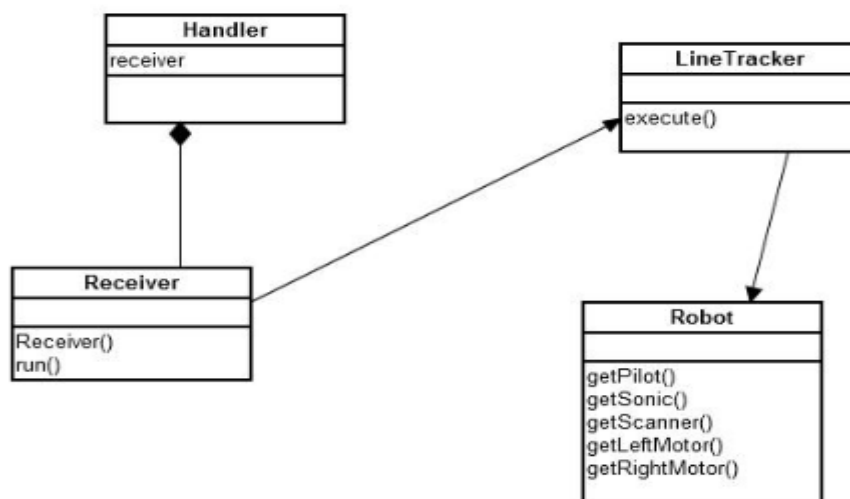


Figure: Color Tracking Class Diagram

When the path finding step finished, the robot side's receiver will receive a notification, then it will call the LineTracker class to track the line. The LineTracker only contains the fundamental logic of the line tracking algorithm, it needs the Robot class to perform basic movement and rotation.

## Object recognition Module:

Marker recognition system(PC):

Description: The key function of the program is a “While-true” loop, it was used for capture video frame from the camera(which mounted on the robot), and try to find marker in the frame. When the marker is found, the program send the marker position value to the Java main program. And then the Java program pass the message to the robot handler, the handler call the corresponding function.

A sequential diagram of the maker recognition program(video processing part)[8]:


<b>Initialization</b>	1. Initialize the video capture and read in the marker pattern files and camera parameters.
<b>Main</b>  <b>Loop</b>	2. Grab a video input frame.
	3. Detect the markers and recognized patterns in the video input frame.
	4. Calculate the camera transformation relative to the detected patterns.
	5. Send the calculated marker's x,y,z coordinate value to Java main program by socket.
<b>Shutdown</b>	6. Draw the virtual objects on the detected patterns.
	7. Close the video capture down.

Table: Marker recognition program structure<sup>1</sup>

However, the system requires not only recognize the marker, but also send the marker 3D coordinate to the robot client. So the marker recognition program is just a part of the marker recognition system. Here is a class diagram for the entire system:

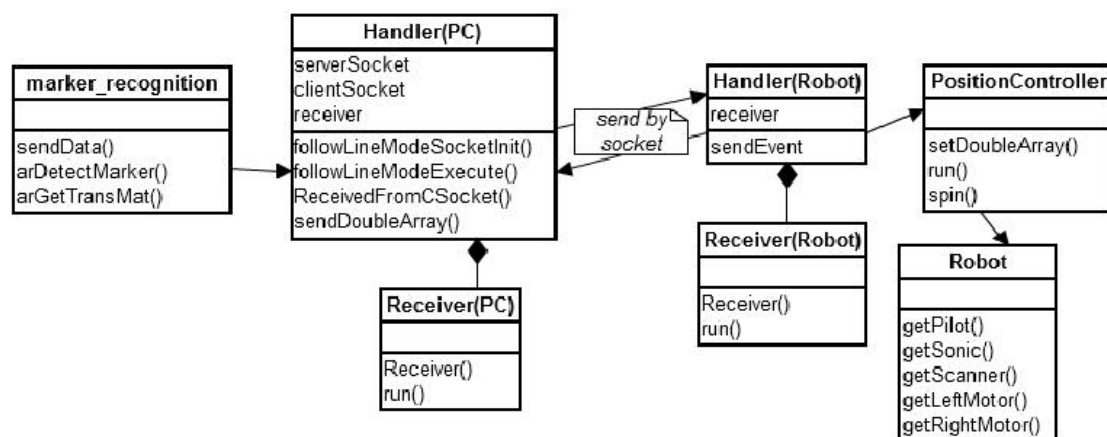


Figure: Marker Recognition and robot position controller class diagram

<sup>1</sup> Source: <http://www.hitl.washington.edu/artoolkit/documentation/devprinciple.htm>, adapted to fit this project



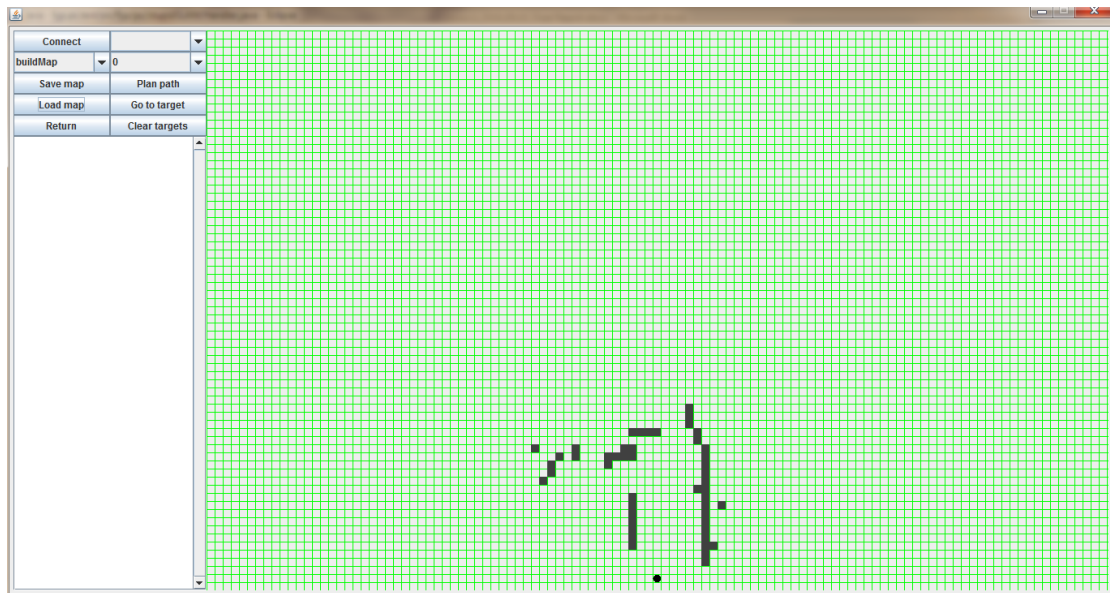


Figure: A sample User Interface snapshot

This system is not a fully automated robot controlling system, so it needs a user to interact with the system. The upper left corner is a list of combo boxes and buttons, they are used to select the mode of the system(build map mode, go to target mode, etc), connect to the robot, and decide which classroom the robot goes into, and save and load a map, etc. For more information see [User Manual](#).

### 3.4.3 Information flow

In this project, the PC will perform as “sender” or “server” to send control commands or data to the LEGO robot, while the robot interpret the information and perform some actions.

Specifically, in the features that will be described in section 3.3, several types of information flow worth to mention:

1. Robot navigation stage: the PC sends events like “GOTO waypoint”, “Finished” to the robot, based on the information, the robot perform actions, and return the sensor readings. The PC program receive those readings and plan the next action. (PC main program -> Robot -> PC main program)
2. Color tracking stage: the stage is only involved the robot, since a simple algorithm with minor calculation can do this job, so there’s no need for sending readings to the PC program. (Robot itself)
3. Marker recognition stage: the information flow at this stage is a little bit complex: The program capture video frame at the speed of 30 frame each second, and then analyse the frame, if a marker is found in that frame, the program will send three parameters(x, y, z coordinate in Camera coordinate system) to our Java main program, then our Java main program pass the data to the robot, finally, robot analyse the direction and distance between robot and the marker.



To summarize, a simple data flow chart of the system can be drawn as follows:

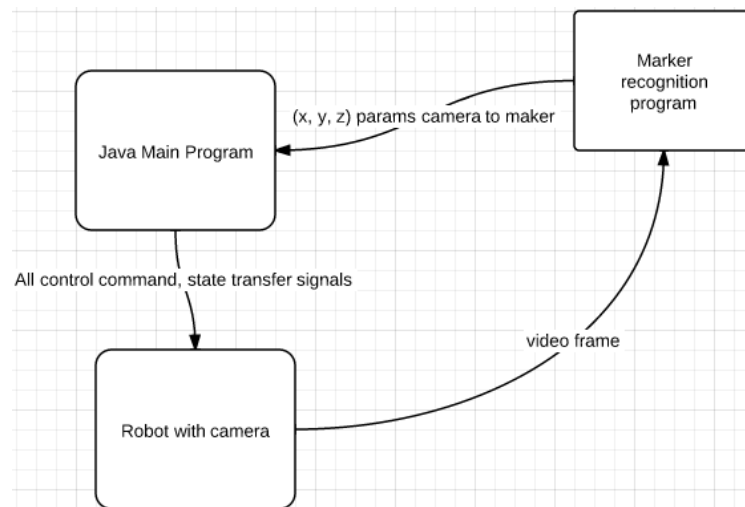


Figure: Information flow diagram

### 3.5 Design of the robot

Finally, a good robot that can perform the tasks described above need to be built. As mentioned in [section 2.2](#), a LEGO Mindstorms robot will be used in this project, plus external camera and projector.

The robot requirements:

- Forward, rotate, backward, with the aid of left and right motor.
- Detect obstacles and hence build a map by using sonic sensor.
- Detect color and follow the color tape by using light sensor readings.
- Recognize marker by using camera.
- Display lecture notes and adjust projection angle by using projector and robot arm.

### 3.6 Source code layout

There're code snippets thought out the [chapter 4](#), all of them will be fully commented and be put into tables. Some lines of code with "(built-in)" postfix mean they're built-in function in some third party libraries or they're written by the other developers, but was used in this project. Full documentation and comment can be found in attached DVD.

### 3.7 Design Patterns

#### 3.7.1 Singleton Pattern

Definition: "The Singleton Pattern ensures a class has only one instance and provides a global point of access to it." [14]

A typical usage of singleton pattern is that there's only one object is available for the given class, which means that the object is global, and can be accessed anywhere. For a simple Java Swing program using MVC model, the advantage of this pattern is significant: there's no need to other classes keep a reference of the object that might be used later. By calling the class's `getInstance()` method, we minimize the object passing function call. Here's a standard implementation of the singleton pattern:

```
public class ClassType{

    private static ClassType object = null; // the only object we use

    private ClassType(){ } // do some initialization

    // global get method
    public static ClassType getInstance(){
        if(object == null) object = new ClassType();
        return object;
    }
}
```

Table: Code snippet for implementing Singleton pattern

### 3.7.2 Model View Controller(MVC) Pattern

#### *What is MVC pattern?*

The MVC pattern separates the domain logic into three separate classes, namely, model, view and controller. They will take care of business logic, UI logic, and input logic respectively[15].

- Model: responds to the requests for information and the change of the state.
- View: display of the information
- Controller: manages user input, responsible to informing the model to change the state.

### *Variation in this project*

This project use the MVC architecture pattern in both client side(the robot) and the server side(the PC). Here is two abstract diagrams that shows how the project use this pattern.

The server side(follows the MVC pattern strictly):

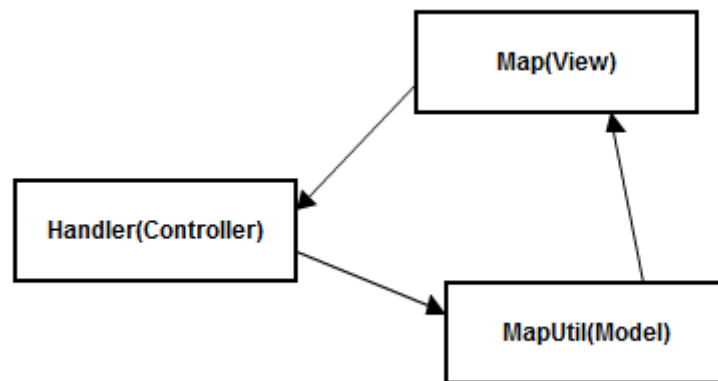


Figure: pattern on the server side

The robot side:

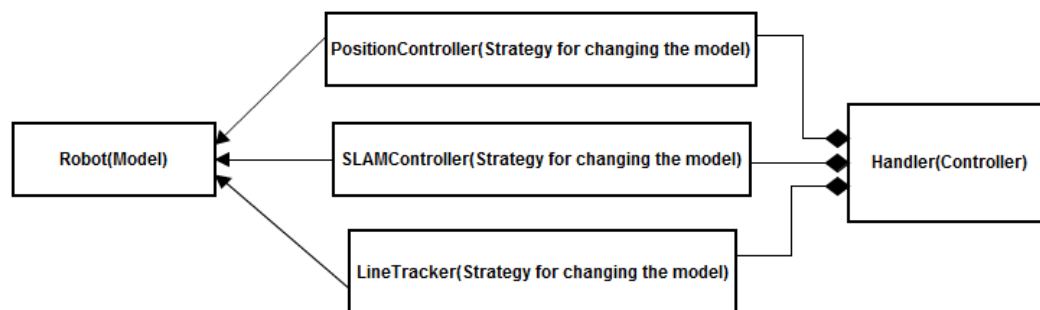


Figure: Adapted pattern on the robot side

Since the robot side does not contains a “View”, the pattern being used is actually an adapted form of “Strategy pattern”[14], but the general idea was the same as the MVC pattern.

## Chapter 4 Development

### 4.1 Procedure Description

The procedure includes three sub-procedures:

1. Map building step: At the beginning, the map on the window is empty, a user can control the robot by “Right Click” on arbitrary place on the map, based on its knowledge about the environment. That is, the reasonable movements are depend on the cooperation of the user, the user shouldn’t try to drive the robot to invalid place, e.g. to hit the wall.

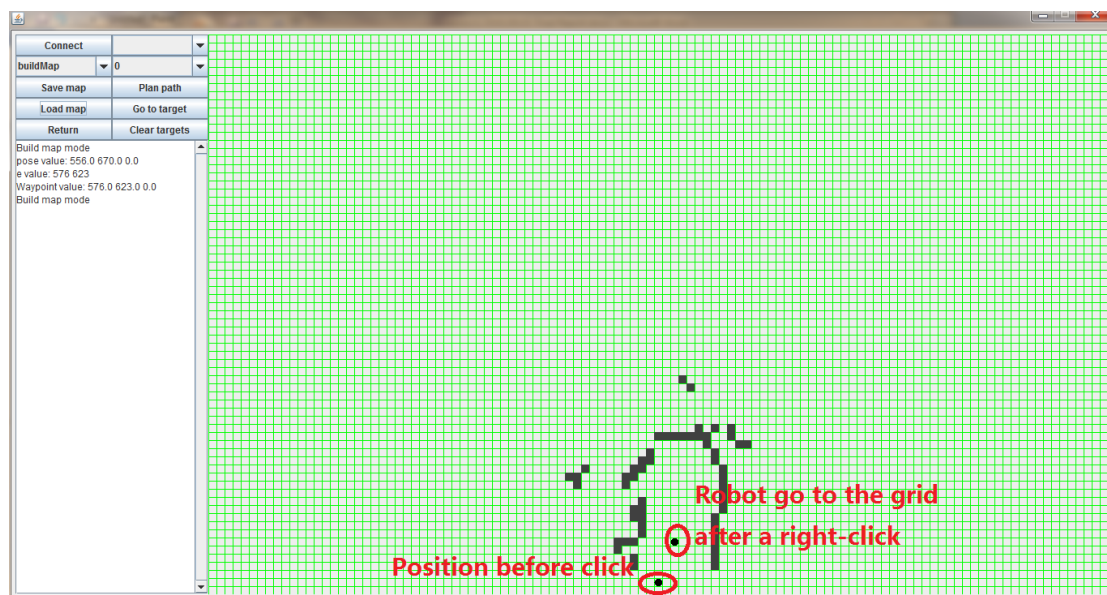


Figure: Using mouse to control the robot movement

2. After the map was built, the user can manually select multiple grids on the map as “targets” by left clicking on the map. After the click, some red dots denote the target will be display on the map.
3. Then the map has been setup. The user can select a room number from a combo box. After the input is finished, the robot calculate a list of waypoints for its current position to the target room, and it perform “particle filter” to update its real state. After the robot comes into the room, a color tracking is available right there. Then the robot will be able to follow the track to a position that not too far away from the whiteboard. At this stage, the camera starts to turn around slowly and send the captured images to the program, and program start to find out where is the tag(the tag use to represent a whiteboard). Once the program find the whiteboard, it will send command to the robot to tell how to project on it.

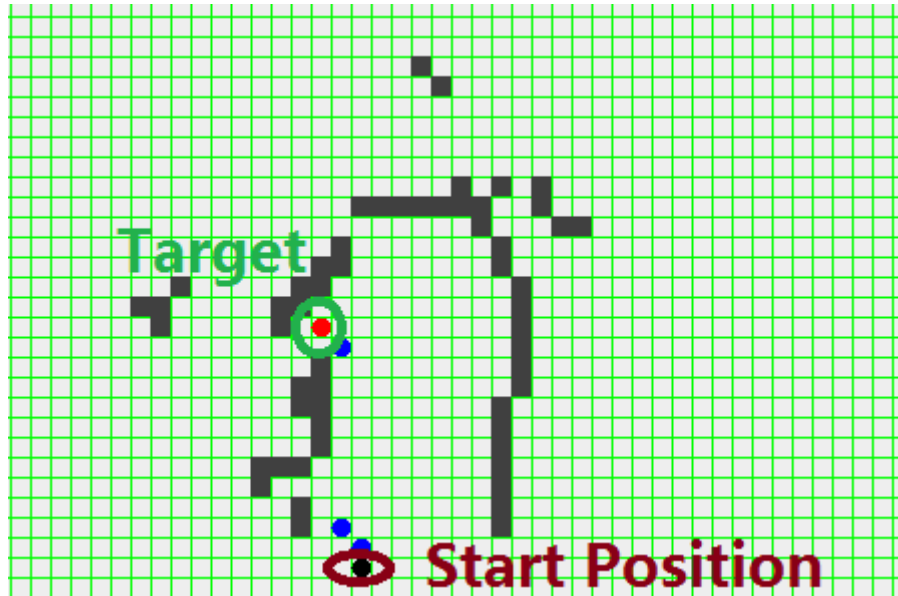


Figure: Target(red dot), start position(black dot) and a list of waypoints(blue dots)

After the lecture is finished, the robot needs to go back to the start point. Whether the lecture is finished is decided by the timer mentioned previously, then the robot need to find a way out from the classroom. Once the robot find the color track again, it will follow the path and go back to the initial point.

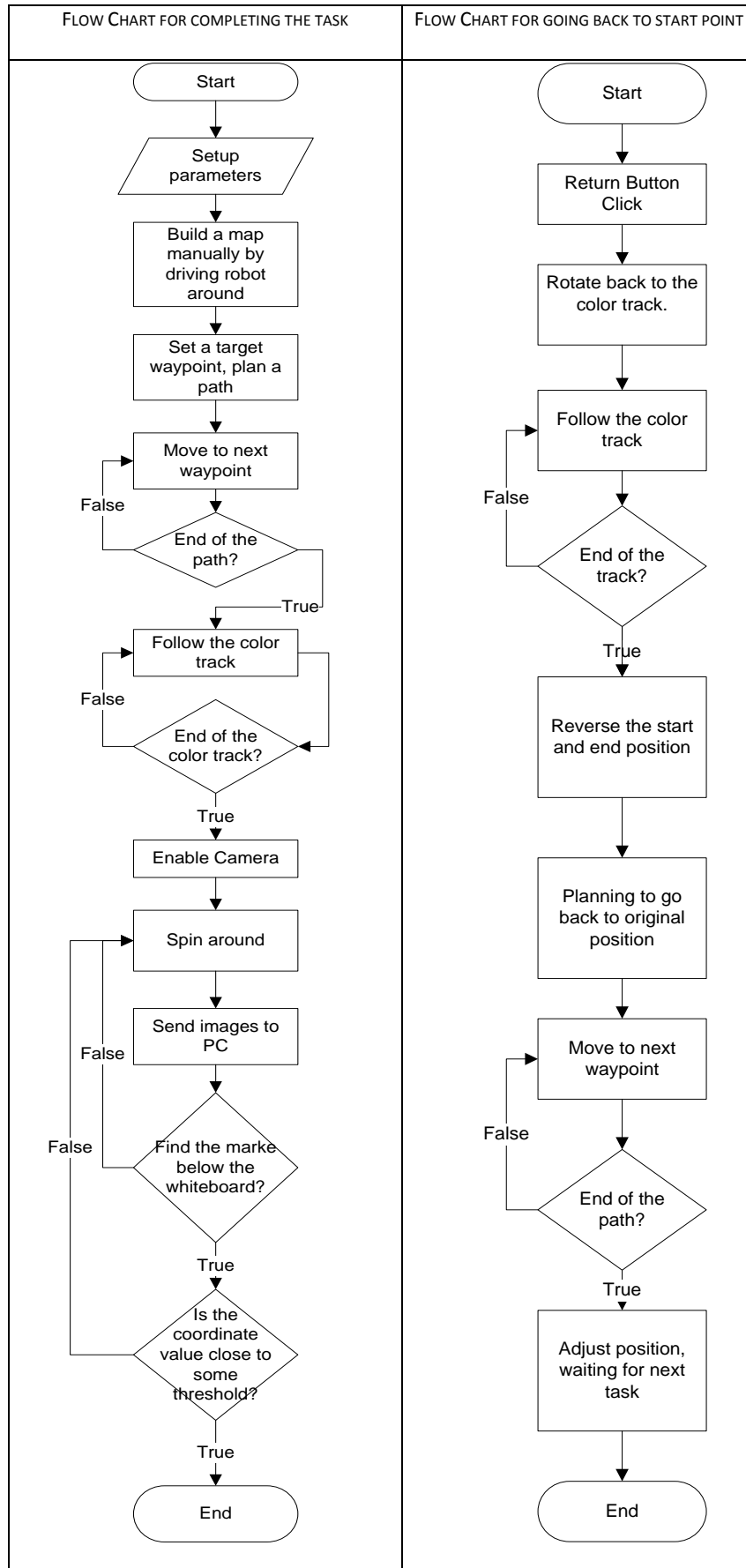


FIGURE 2: PROCEDURE DIAGRAM

## 4.2 Source code structure

Server side source code structure:

Folder: fyp.pc

- package fyp.pc.stupidSLAM
  - BayesFilter.java (Implementation of the Bayes Filter algorithm, use for Robot Mapping, discuss in [section 4.2.2](#))
  - Handler.java (Central controller, responsible for sending and receiving data, and decide perform which function call based on the event receiving)
  - Map.java (A GUI program that visualize the map and interact with the user)
  - MapUtil.java (A set of algorithm that perform updates based on sensor readings)
  - MouseHandler.java (A listener that responsible for receiving mouse event)
  - PathFinder.java (An A\* algorithm implementation that find a path from current position to the target, discuss in [section 4.2.2](#))
  - ClassDiagram.ucls (A class diagram that generate by OpenAid third party reverse engineering tool)
- package fyp.pc.test (all test code, used for test algorithm and functionality during the development)
  - BTInitiator.java (use to test Bluetooth connection, send command to the robot, also use to receive parameter from the marker recognition program)
  - CommObject.java (A message class that use to pass information between the robot and the pc)
  - CommServerTest (use to test whether passing the CommObject is success)
  - HashMapTest.java (Use to test Hashmap method overwriting, used in Path finder program)
  - PriorityQueueTest.java (Use to test PriorityQueue method overwriting, used in Path finder program)
  - Ransac.java (Implementation of RANSAC algorithm, not use in final version of the code, will be discussed in [section 4.2.2](#))
  - RevisedDataUtil.java (Use K-mean to smooth the ultrasonic error, not use in final version of the code, will be discussed in [section 4.2.2](#))
  - RuntimeExecute.java (Use to test calling the marker recognition program at runtime).
  - TestRansac.java (A GUI program that visualize the performance of the RANSAC algorithm, not use in final version of the code)
  - USServerTest.java (A GUI program that visualize the “naive mapping” approach, not use in final version of the code. Will be discussed in [section 4.2.2](#))

Client side source code structure:

Folder: fyp.robot

- package: fyp.robot.stupidSLAM
  - Handler.java (Central controller, responsible for sending and receiving data, and decide perform which function call based on the event receiving)
  - LineTracker.java (Perform line tracking function)
  - PositionController.java (Rotate robot base on marker recognition program's output, use to find the marker)
  - Robot.java (Encapsulate all available robot actions, called by other classes to perform robot motion, etc.)
  - SLAMController.java (Use in map building and path following)
- package: fyp.robot.test (all test code, used for test algorithm and functionality during the development)
  - AdjustPosition.java, Behavior.java, BTReceiver.java, EntryPoint.java, LineTracking.java, PositionMonitor.java (As a prototype of package fyp.robot.stupidSLAM)
  - LightSensorTest.java (Use to collect light sensor threshold)
  - MotorTest.java (Use to evaluate motor motion precision)
  - ProjectionTest.java (Use to collect projection parameter threshold)
  - CommClientTest.java (use to test passing objects via socket)
  - RobotMovement.java (use to test robot mapping and path finding)
  - ScannerTest.java (use to test ultrasonic scanner, which will scan surrounding environment and return readings)
  - USClientTest.java (use to test naïve mapping function, not use in final version of the code. Will be discussed in [section 4.3.2](#))

## 4.3 Development details

According to [section 3.3](#), and [section 3.4](#), the author have to develop all those features to build a entire system, so in this section, the programming details and problems encountered in all the features mentioned above will be discussed.

### 4.3.1 Fundamental function

#### *1. Robot controlling encapsulation*

In [section 3.3.1](#), the project was divided into different components, each component has some overlap functions. For example, the SLAMController has to call the robot to move, rotate; the LineTracker has to call the robot to do the same job, and so does the PositionController. By using “[the Singleton Pattern](#)”, mention before, we can extract a common Robot utility class that has a single global entry point. So all other controller can call the Robot instance directly to perform some tasks.



```

public class Robot {

    private DifferentialPilot pilot; // used to control the robot

    private RangeFinder sonic; // get range reading

    // rotate the robot to get a list of range readings

    private RangeScanner scanner;

    // motors, and also motors' port numbers

    private NXTRegulatedMotor leftMotor = Motor.C;
    private NXTRegulatedMotor rightMotor = Motor.B;
    private NXTRegulatedMotor armMotor = Motor.A;

    // sensor port number

    private SensorPort ultrasonicSensorPort = SensorPort.S4;
    private SensorPort lineSensorPort = SensorPort.S3;

    private LightSensor lightSensor; // light sensor

    private static Robot robot; // global access instance

    private Robot(){    }    // initialize all parameters

    public static Robot getInstance(){ // global access point
        if(robot == null){ robot = new Robot(); }
        return robot;
    }

    public void resetMotor()// stop motor, clear history count

    public void setAngles(float[] angles)//set scanner's scan angle

```

```

// A list of getter

public DifferentialPilot getPilot()

public RangeFinder getSonic()

public RangeScanner getScanner()

public NXTRegulatedMotor getLeftMotor()

public NXTRegulatedMotor getRightMotor()

public NXTRegulatedMotor getArmMotor()

public LightSensor getLightSensor()

}

```

Table: Code snippet for Robot class

This class is relatively simple, it's quite similar to Java Bean, since it just provide a list of getter and setter function to support other classes.

## 2. Processing reading data

Getting reading data from robot and send it back to the PC server is an important step in map building feature. The code snippet below shows how to perform sensor reading in LEJOS and send it back to the PC:

```

public RangeReadings performReadings(){
    Robot robot = Robot.getInstance(); // get the robot instance

    // get a list of angles, in this case, it's from -90 to
    // 90 degree, with step length(incAngle) equals to 5
    angles = getAngles(startAngle, endAngle, incAngle);

    // set the angles
    robot.getScanner().setAngles(angles);

    // then robot rotate to each angle to perform scanning.
    RangeReadings readings = robot.getScanner().getRangeValues();

    return readings; // return a list of reading
}
// then in the Handler class, we call:
public void sendRangeReadings(NavEvent navEvent, RangeReadings readings){
    try{
        // send event type and readings to the server
        dos.write(navEvent.ordinal());
        readings.dumpObject(dos);
        dos.flush();
    }
    catch (Exception e) {
    }
}
}

```

Table: Code for Processing and sending data(Robot side)

On the server side, the receiver will receive the sent data, and pass the data to MapUtil class for further process:

```
while(true) {  
    byte event = dis.readByte(); // read event index  
  
    // get event by index  
    NavEvent navEvent = NavEvent.values()[event];  
  
    switch (navEvent) {  
        case RANGE_READINGS:  
            rangeReadings.loadObject(dis); // load reading data.  
            mapUtil.processData(rangeReadings);  
            break;  
        // skip other cases  
    }  
}
```

Table: Code for Handler(Server side) receiving and passing data to MapUtil

Then mapUtil.processData takes the responsibility of data validation and data conversion, it convert the data into points so that the data can be drawn on the map:

```
public void processData(RangeReadings readings){  
  
    Map map = Map.getInstance();  
  
    // get current robot pose(robot position and heading on map)  
    Pose robotPose = map.getLastPoseIdeal();  
  
    // get current robot heading  
    float robotAngle = robotPose.getHeading();  
  
    // initialize array to store processed data
```

```

Vector<Point> allPoints = new Vector<Point>();

boolean [] isValid = new boolean[readings.size()];

for(int i=0;i<isValid.length;++i){isValid[i] = true;}

// do a lot of calculation here, for-each reading, get its
// position on the screen
for(int i=0;i<readings.size();++i){
    RangeReading reading = readings.get(i);

    // validate the reading, i.e. beyond the maximum
    // range of ultrasonic sensor
    if(!validateReading(reading)){isValid[i]=false;}

    // convert angle in degree to angle in radian
    double angle = convertDegreeToRadian(reading.getAngle()
        + robotAngle;

    Point point = null;
    if(isValid[i])
        // if the point is valid, convert it
        point = pointOnScreen(angle, robotPose,
            reading.getRange());
    else
        // if the point is not valid, regards it as there's
        // an obstacle in maximum range
        point = pointOnScreen(angle, robotPose,
            (float)MAXI_RANGE);

    allPoints.add(point);
}

```

```

    }

    // perform bayes filter, and set the probability value on map,
    // describe later

    BayesFilter.execute(allPoints, isValid, robotPose);

    // re-enable map to capture new user control event
    map.setEvent(true);
}

```

### 4.3.2 Robot navigation

#### 1. Mapping:

General introduction:

##### A. Naive mapping:

Naive mapping stands for drawing obstacles on the map just base on the distances between the robot and the obstacles and the angles between robot's forward vector and scan vector. Here's a simple snapshot of naive mapping technique, this image is the result of the MapUtil class and Map class after execution in the author's apartment:

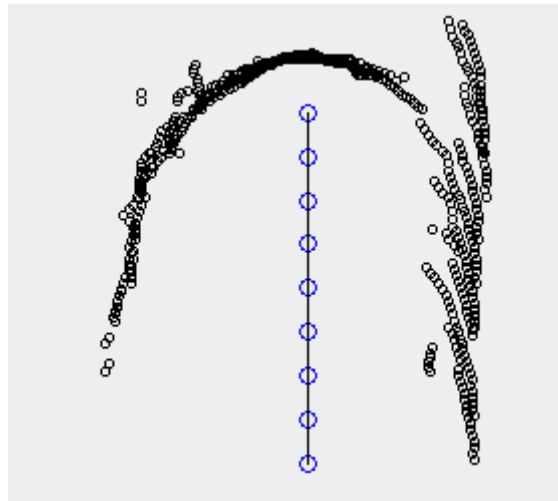


Figure: a naive map(three walls)

In the picture above, the blue circle stands for robot position, the black circles stand for obstacles that the robot "see" using its ultrasonic sensor. In this environment, all the obstacles actually form three walls, but due to the "ultrasonic sensor reading inaccuracy" (which will be discussed in the "problem and solution" section), the data was actually form a "U" style alley.

The advantages and disadvantages of naive mapping:

The main advantage of naive mapping is that it is easy to implement, with the robot current position and the sensor reading(distance and angle pair), we can construct a map without any further calculation.

However, its disadvantages worth noting:

It's computationally hard to find a good data structure to represent "a point's neighbours", that is, when the robot moving and sending back data constantly, we can store all the reading in an array and draw all the obstacles on the map, but we just don't know the relationship on between each point. Technically, it's called "Data Association" problem[16]. For a simple task like finding a wall or finding out whether there's a route from source point to destination point, the calculation time and algorithm is too sophisticated to use.

With the inherent drawback of this representation method, the time complexity of finding a valid path is unacceptable, e.g., we need to enumerate all points to find out which points will stop the robot move to the target, we need to find a threshold so that two neighbour points can be regarded as same object when the distance is under a certain value as well as believe there are separate objects. There's a solution for tackling the drawback, which is called "RANSAC" algorithm, the basic idea of RANSAC algorithm is to find a line or curve that can describe the data under a certain error threshold. Detailed description can be found in[16], [17].

### *B. Occupancy grid map:*

The occupancy grid map is simply transform the real world environment into a set of grids. Normally each grid has equal width and height. According to [10], one of the major problem of grid map localization is the information loss when transform the state of the world into discrete grid. But the advantage of it is also significant: it reduce a large amount of work for finding nearest objects and detecting obstacle that occupied a large space in the environment, which makes path finding become a simple problem to solve: we can easily label each grid as accessible or occupied, therefore classic algorithms like breadth first search, A\* search can be applied into the map.

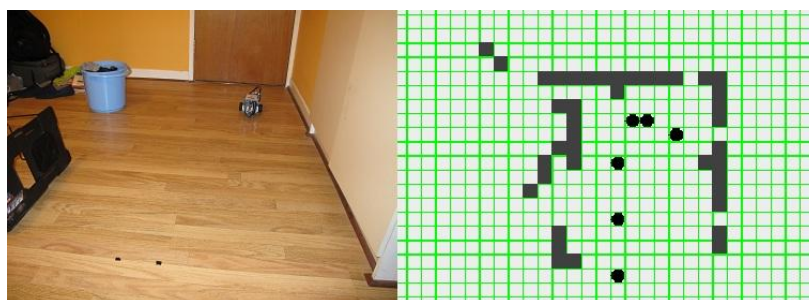


Figure: A sample environment and corresponding grid map

Problems:

As we all know, the ultrasonic sensor is inaccurate in many aspects, the slide below gives a good summary of the effects of the sensor:

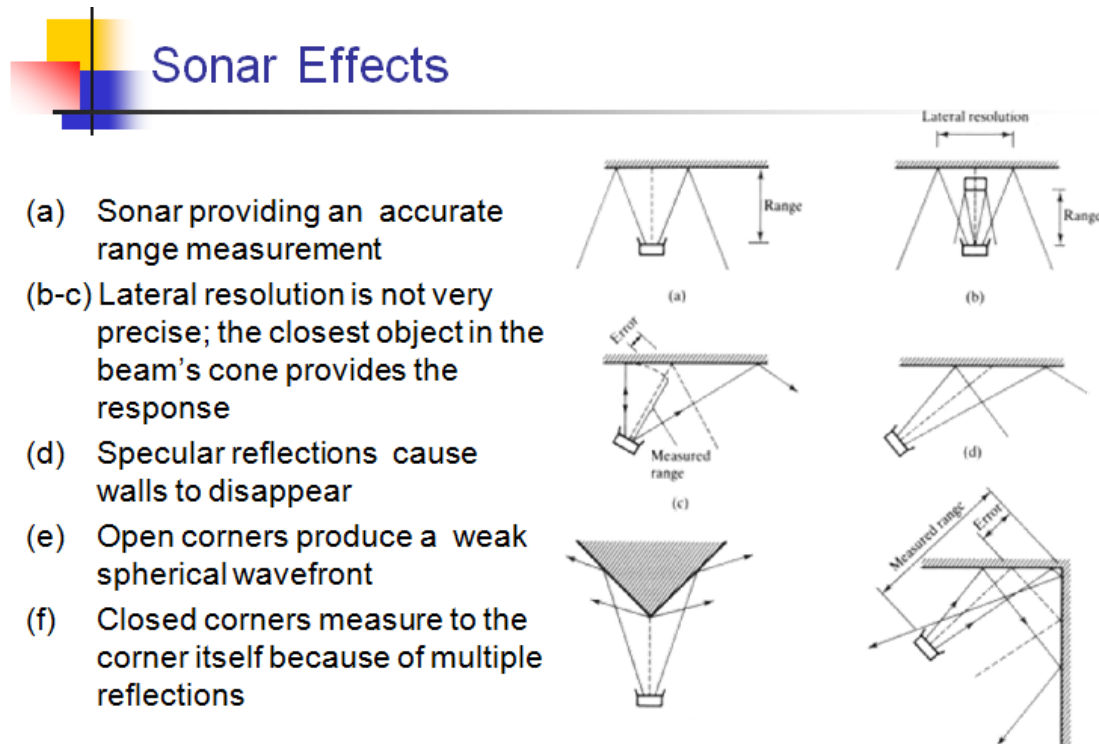


Figure: Problem with ultrasonic sensor<sup>2</sup>

Therefore, the sensor readings can fall into four categories:

- a. Sensor detect object while it is object there in the real world. (True Positive)
- b. Sensor detect object while no object there in the real world. (False Positive)
- c. Sensor detect no object while it is object in the real world. (False negative)
- d. Sensor detect no object while it is no object in the real world. (True negative)

So, if we calculate the occupancy value base on range reading and angle only, we might end up with a very inaccurate map that might even cannot distinguish door from a wall. However, if we use probability model with these four categories approach, we will end up with a really good mapping algorithm(see solution section).

<sup>2</sup> Source:

<http://www.mcs.alma.edu/LMICSE/Workshops/longWorkshops/alma2/powerpoints/Sonar%20and%20Local.ppt>

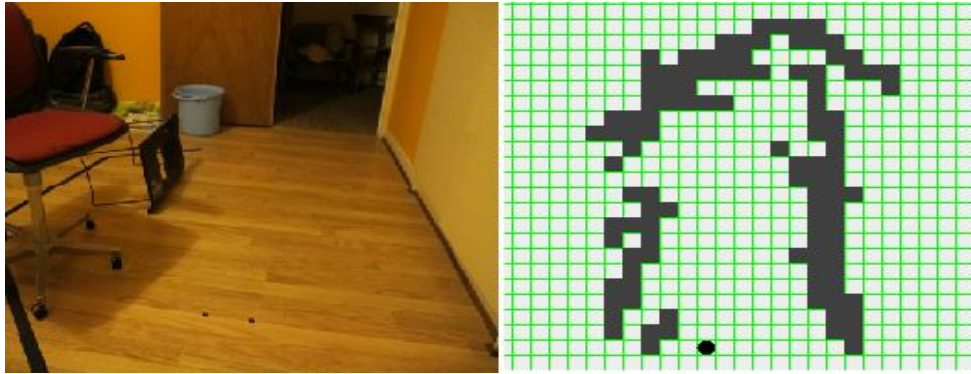


Figure: A map with phantom readings

To address this problem, several algorithms are applicable in this situation. The author will discuss two of them. One is “K nearest neighbour(KNN) with Ransac”, another is “Bayes Filter”. The second one is a probability model. That is, we only accept the reading value with a certain probability. By recursively updating the probability of occupancy of each grid in each measurement step, we can find a better mapping result.

## Solutions:

### *K nearest neighbours(KNN):*

The KNN algorithm is originally used in Machine Learning, it's an effective way to classify a query instance by examples [4]. A simplification introduction of KNN is that the property of a new value will be affected by its nearest neighbours, the neighbour functions that are commonly used are Manhattan distance, Euclidean distance, and Hamming distance. In this project, the author “borrows” the idea of K nearest neighbours algorithm, trying to fix the sonar effect problem mentioned above.

The key idea is that, based on experimental conclusion, the ultrasonic sensor reading will be more precise at 90, -90 and 0 degree. Trying to read range from other degrees that among these degrees will result in inaccurate return.

The algorithm the author uses here is: get K nearest, more-precise values and calculate their mean with the query instance. So it's also called K-Mean algorithm. And it's assuming that 0, 90 and -90 degree are more reliable readings than other readings among these thresholds. Here is a full implementation of K-Mean:

```
// the reading is from -90 to 90, with 5 degree step length.
// K equals to 3 in this case.
private void reviseUltrasonicBeam(Vector<Point> tempData, boolean[] validList){
    int size = tempData.size();
```



```

int startPos = size/2+1;

// revise y, middle to left. That is, we use readings
// starting from 0 degree to revise the data among 0 to -90
// degree. We can think it as: Pull the data up.
for(int i=startPos-2;i>-1;--i){
    if(validList[i] && validList[i+1] && validList[i+2])
        tempData.get(i).y = (tempData.get(i+1).y +
            tempData.get(i+2).y + tempData.get(i).y)/3;
}

// revise y, middle to right. That is, we use readings
// starting from 0 degree to revise the data among 0 to 90
// degree. We can think it as: Pull the data up.
for(int i=startPos+2;i<size;++i){
    if(validList[i] && validList[i-1] && validList[i-2])
        tempData.get(i).y = (tempData.get(i-1).y + tempData.get(i-2).y
            + tempData.get(i).y)/3;
}

// revise x, left to middle. That is, we use readings
// starting from 90 degree to revise the data among 90 to 0
// degree. We can think it as: Pull the data right.
int middle = startPos;
startPos = 0;
for(int i = startPos + 2; i < middle;++i){
    if(validList[i] && validList[i-1] && validList[i-2])
        tempData.get(i).x = (tempData.get(i-1).x + tempData.get(i-2).x
            + tempData.get(i).x)/3;
}

```

```

// revise x, right to middle. That is, we use readings
// starting from -90 degree to revise the data among -90 to 0
// degree. We can think it as: Pull the data left.
for(int i = size - 3; i > middle; --i){
    if(validList[i] && validList[i+1] && validList[i+2])
        tempData.get(i).x = (tempData.get(i+1).x + tempData.get(i+2).x
                               + tempData.get(i).x)/3;
}
}

```

Table: Code snippet for K-Mean

Below is a comparison between a map plotted with no KNN and a map with KNN to revised data:

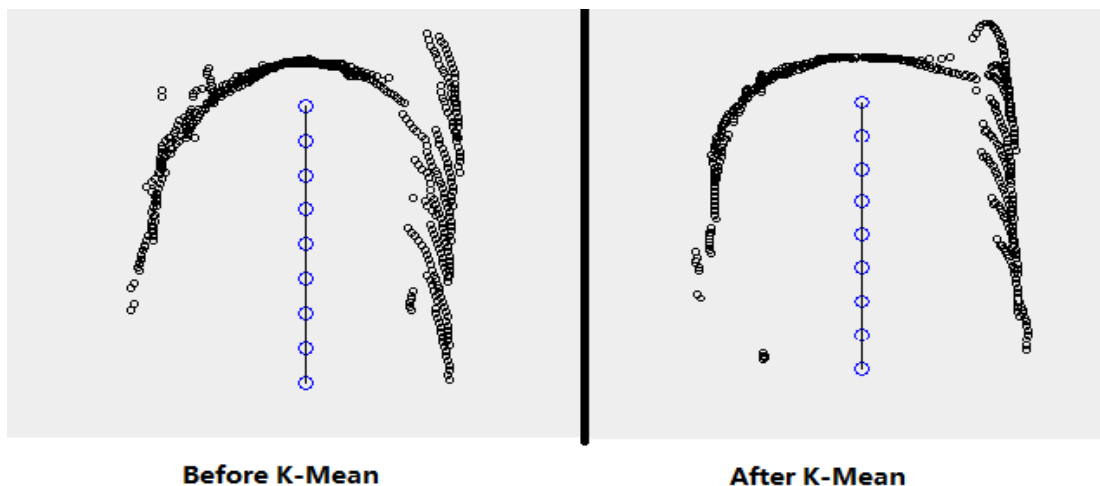


Figure: A naive map before K-Mean and after K-Mean

It shows clearly that after K-mean algorithm, the environment more looks like a three-wall alley.

### *Bayes Filter, A probability approach:*

According to [10], the Bayes Filter algorithm is "the most general algorithm" for calculating posterior belief state. It is a recursive algorithm, which means that the belief state  $bel(x_t)$  at time  $t$  is calculated from the belief state  $bel(x_{t-1})$ , with the most recent control  $u_t$  and the most recent sensor measurement  $Z_t$ . The algorithm's output is the belief state  $bel(x_t)$  at time  $t$ . Below is the Bayes Filters pseudo algorithm:

```

Algorithm Bayes_filter(bel( $x_{t-1}$ ),  $u_t$ ,  $z_t$ )
  for all  $x_t$  do
     $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1})dx$  //prediction step
     $bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t)$  // measurement update
  endfor
  return  $bel(x_t)$ 

```

Table: A general Bayes Filter Algorithm<sup>3</sup>

With the pseudo code, a Bayes Filter in Java can be implemented easily:

```

// probability of sensor say yes, there's an obstacle; and there's
// an obstacle in real world.
private static final double prob_SensorYes_RealYes = 0.8;

// probability of sensor say no, there's no obstacle; and there's
// an obstacle in real world.
private static final double prob_SensorNo_RealYes = 0.2;

// probability of sensor say yes, there's an obstacle; and
// there's no obstacle in real world.
private static final double prob_SensorYes_RealNo = 0.4;

// probability of sensor say no, there's no obstacle; and there's
// no obstacle in real world.
private static final double prob_SensorNo_RealNo = 0.6;

// update belief state based on last belief state
private static double calculateProbability(boolean sensorYes, double
lastValue){
    double prob_yes;

```

<sup>3</sup> Source: [10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*: The MIT Press, 2005., page 36.

```

double prob_no;

if(sensorYes){

    //  $p(x_n) = p(y|y) * p(x_{n-1})$ , x is yes.

    prob_yes = prob_SensorYes_RealYes * lastValue;

    //  $p(x_n) = p(y|n) * p(x_{n-1})$ , x is no.

    prob_no = prob_SensorYes_RealNo * (1.0-lastValue);

}

else{

    //  $p(x_n) = p(n|y) * p(x_{n-1})$ , x is yes.

    prob_yes = prob_SensorNo_RealYes * lastValue;

    //  $p(x_n) = p(n|n) * p(x_{n-1})$ , x is no.

    prob_no = prob_SensorNo_RealNo * (1.0- lastValue);

}

// Normalise the probability to 1.

return (1.0f/(prob_yes+prob_no)) * prob_yes;

}

```

Table: Implementation of Bayes Filter

The If-Else block is corresponding to prediction step in the pseudo-code, the pseudo-code is the generic form, which means that the  $x_t$  can choose more than two values. In our case, the  $x_t$  is in binary form, so the scary formula was changed to two line of code. The normalization in the last line corresponds to the measurement update formula in the pseudo-code, it just normalize the probabilities to let them sum up to 1.

But writing this function is not the end of Bayes Filter in the entire system. Using the sensor readings, all the edge grids can be updated, since we already calculate the x, y map coordinate values of all edge grids. But we also need to update all internal grids, that is, assuming we observe an object at place x, and our robot position is at place y, let's say x is greater than y, we need to update all probabilities from place y+1 to x-1, because an observation of an obstacle at place x also means observations of no obstacle from place y+1 to x-1. It's a non-trivial problem, and it might takes a lot of computation time to update all the internal grids. The Floodfill algorithm is very useful in this case:

```

// Firstly, we update all edge grids, and set their visit status to
// true. Then we start from the robot current position, call the
// flood fill algorithm recursively.
Flood_fill(int start_x, int start_y)
Expand start_x, start_y in four directions(left,right,up,down)
    If (new_x, new_y valid, and map[new_x,new_y] not updated)
        Flood_fill(new_x,new_y)

```

Figure: Flood fill algorithm pseudo-code

At this stage, a probabilistic map has been built up. The robot perceive its surroundings and sends readings back, the above implementation guaranteed the map will be built successfully.

## 2. Path Finding:

When the map has been built, the robot need a list of waypoints that can guide it to the target. As mentioned earlier, the target has to be set manually. After a target has been set, the system has to find a way from the robot's current position to the destination. A\* algorithm is a good choice in this case.

A\* algorithm definition[4]:  $f(n) = g(n) + h(n)$ , where  $f(n)$  is the estimated cost of the best solution at position  $n$ ,  $g(n)$  is the real cost from start position to current position  $n$ ,  $h(n)$  is the expected cost from  $n$  to the goal. It can be proved that A\* is optimal, which means that it will find an optimal path from starting point to the target.

```

A_start(startNode, endNode){
    // store node and distance to the endNode
    // check if the node has been visited
    HashMap<Node,double> hashMap;

    // extract a node base on priority(f(n))
    PriorityQueue<Node> queue;

    // initialize startNode
    startNode.g(n) = 0; startNode.parent = null;
    startNode.h(n) = distance(startNode, endNode);
    startNode.f(n) = startNode.g(n) + startNode.h(n);

    // add the start position into queue
    queue.push(startNode);

    // put the node into hash map
    hashMap.insert(startNode, startNode.f(n));

    Node resultNode;

    while(not queue.empty()){
        Node node = queue.pop();

        if(node's x,y value equals to endNode){
            resultNode = node;
            break;
        }
    }
}

```

```

// expand node in eight direction
// left, right, up, down,
// upper-left, upper-right, down-left, down-right
for each direction {
    x = node.x + direction.x
    y = node.x + direction.y

    if (x,y is not valid) continue;

    if(map(x,y) is occupied) continue;

    // A* heuristic
    // h(n) = distance from (x,y) to endNode's (x,y)
    h(n) = distance((x,y),endNode(x,y));
    // g(n) = distance from (x,y) to node's (x,y)+node's g(n)
    g(n) = distance((x,y), node(x,y)) + node(x,y).g(n)
    f(n) = g(n) + h(n)

    // create a new node
    Node newNode(x,y, g(n), h(n), f(n));
    newNode.direction = direction;
    newNode.parent = node;

    // if the node have been visited
    if(hashMap.contains(newNode)){

        // replace with a good node with less distance than before
        if(hashMap[newNode].f(n) > newNode.f(n)){
            hashMap.insert(newNode, newNode.f(n))
        }
        else{
            continue; }
    }
    else{
        // not in map before, add the node into the map
        hashMap.insert(newNode, newNode.f(n));
    }
    // add the node into priority queue.
    queue.add(newNode);
}
}
Vector<Waypoint> resultWaypoints = new Vector<Waypoint>();

// add the destination node to result set.
if(resultNode != null){
    resultWaypoints.add(Waypoint(resultNode));
}

// only add node where change direction, that is, if the previous node p
// expand the current node c with the same direction as the parent of
// p expand p, skip this node since it wouldn't result in change direction
while(resultNode != null){
    while(resultNode.parent.direction == resultNode.direction
        &&res.parent!=null){
        resultNode = resultNode.parent;
    }

    // change direction
    if(resultNode.parent != null){
        resultWaypoints.add(Waypoint(resultNode.parent));
    }
}

```

```

        resultNode = resultNode.parent;
    }
    return resultWaypoints;
}

```

Table: A\* algorithm for path finding with hash map to test duplicate nodes

In the pseudo code above, the author use the priority queue to speed up searching for the best node to expand by heuristic function  $f(n)$ . Also, the function need to keep track of whether the node has been expanded before and whether the node has a better  $f(n)$  than before. Finally the algorithm will find a path if the path exists.

Then we can optimize the path by removing the middle nodes that in a straight line. Think of a path is constructed by a list of straight lines. What we really need is recording those “turning points”, all other nodes that in the middle of a line can be removed since they didn’t change the direction of the robot movement.

### 3. Color tracking:

LEJOS provides a handy way to control robot movement and sensor update, it’s easy to program a color tracking robot even without any Java programming experience. The basic idea of color tracking can be described in pseudo code as follows:

```

While light_sensor_reading != end_signal
    if light_sensor_reading == a_threshold
        robot moves forward
    else
        robot rotates a certain degree to read sensor value
        increment the degree at each rotation

```

Table: Pseudo code for color tracking

The algorithm is quite simple:

1. The robot will be placed on a black track at the beginning.
2. Move forward and get reading from the sensor.
3. If the reading is “Color is black”, go to 2; If the sensor data is “End signal”, go to 5; else, go to 4.
4. The robot will scan its surrounding by rotating, try to read sensor data at the same time.
5. The end of the color track, start object recognition that will be discussed in next section.

#### 4.3.3 Object recognition

##### marker recognition & orientation

ARToolKit has already have self-contain marker recognition function built-in. At this stage, the problem was transformed to: how to get the related coordinate of a

marker in a camera-coordinate system. Fortunately, the library is using the camera coordinate system to calculate the marker position.

The camera coordinate system can be described as follows[8]: if we move the camera towards the marker, the x value goes down; move the camera to left, z value increases, move the camera up, y value decrease. So, with x and z value, we can find a good project position, with y value, we can find a good project angle.

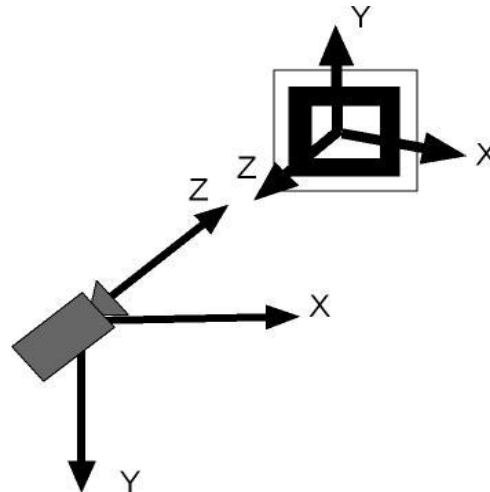


Figure: ARToolkit Coordinate Systems(Camera and Marker)<sup>4</sup>

With the existing library and sample code, writing a new marker recognition program is relatively simple:

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv); // initialize OpenGL Window, UI, (built-in)
    init(); // open video stream, initialize the marker pattern, (built-in)

    // initialize socket to communicate with Java main program
    client_socket = socketInit();

    arVideoCapStart(); // start video capture, (built-in)

    // (built-in), the most important function, mainLoop is a function
    // pointer that will be discussed later
    argMainLoop( NULL, keyEvent, mainLoop );

    socketClose(client_socket); // finished, clean resource
    return (0);
}
```

Table: A sample main function of the AR object recognition program

Above code snippet is just a skeleton of a typical AR program, in this project, we need to find the pattern in the video stream and calculate the markers' coordinate from the camera's perspective:

4. Picture source: <http://www.hitl.washington.edu/artoolkit/documentation/tutorialcamera.htm>



```

// in the mainloop function, just list important function, skip details
/* grab a vide frame, (built-in) */

if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
    return;
}

argDrawMode2D(); // set draw mode, (built-in)
argDispImage( dataPtr, 0,0 ); // draw the image on window, (built-in)

/* detect the markers in the video frame, (built-in) */
if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
    exit(0);
}

arVideoCapNext(); // ready to capture next video frame

/* check for object visibility, (built-in) */
k = -1;
for( j = 0; j < marker_num; j++ ) {
    if( patt_id == marker_info[j].id ) {
        if( k == -1 ) k = j;
    }
}
if( k == -1 ) {
    return;
}

// get the marker's coordinate, put it into patt_trans matrix
arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);

```

```

array d = patt_trans[] // pseudo code to get x,y,z values

sendData(client_socket, d,LEN); // send x,y,z, values by socket

```

Table: core function of the marker recognition program

#### 4.3.4 Socket communication

As mentioned in [section 3.3.2](#), this project requires communication between marker recognition program(written by C) and main program(written by Java), and between main program and the robot. A good thing about socket is that it supports communication cross languages and platforms.

It's worthwhile to discuss the windows version of socket implementation.

Below is a pseudo C style code for a typical WinSock sender program, all the error testing code was removed:

```

WSAStartup(MAKEWORD(2,2),&Ws) // initialize the use of WinSock DLL

// create client socket(sender)

ClientSocket = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

he = gethostbyname(IP_ADDRESS); // get host by name, i.e. localhost
/* fill in server_address here */

ServerAddr.sin_port = htons(PORT); // set port number

// connect to server_address.port

connect(ClientSocket,(struct sockaddr*)&ServerAddr,sizeof(ServerAddr));

char c[150];

/* fill in the char c array here */

send(client_socket,c,(int)strlen(c),0); // send data

closesocket(client_socket); // close the socket

WSACleanup(); // terminate the use of WinSock DLL

```

### 4.3.5 Projection on the whiteboard

```
// xx, yy value is calculated by the marker recognition program
// xx is the distance from the camera to the marker,
// yy is the vertical height from the camera view centre to the marker

public void rotateRobotArm(double xx, double yy){
    // calculate the angle the arm need to rotate.
    double angle = Math.atan((yy-threshold)/xx);

    angle = Math.toDegrees(angle); // convert radian to degree
    // rotate the arm by degree calculated
    Robot.getInstance().getArmMotor().rotate((int)angle);
}
```

Table: Code snippet for adjusting robot arm's angle

## 4.4 Physical development

The physical robot design was originally from [18], based on the requirements list in [section 3.5](#), the author adapted the rotate ultrasonic scanner to fixed scanner because the LEGO Mindstorms only support three motors, we need motors for left and right wheel and the robot arm. A light sensor was added at the bottom, a webcam and a arm motor was mounted on the left and right hand side respectively. Below is the final version of the robot. The construction of the robot comes all along the software development.

For flexibility purpose, the webcam should be wireless so that the robot can be fully standalone. However, most of the wireless camera are either too heavy for the robot to carry or the angle of the camera lens is too small to capture the marker. So the author decide to use a wired webcam instead. But it won't affect the meaning of this project, because the program logic has not change, we can easily find a bigger robot or a better wireless webcam to reproduce this project.

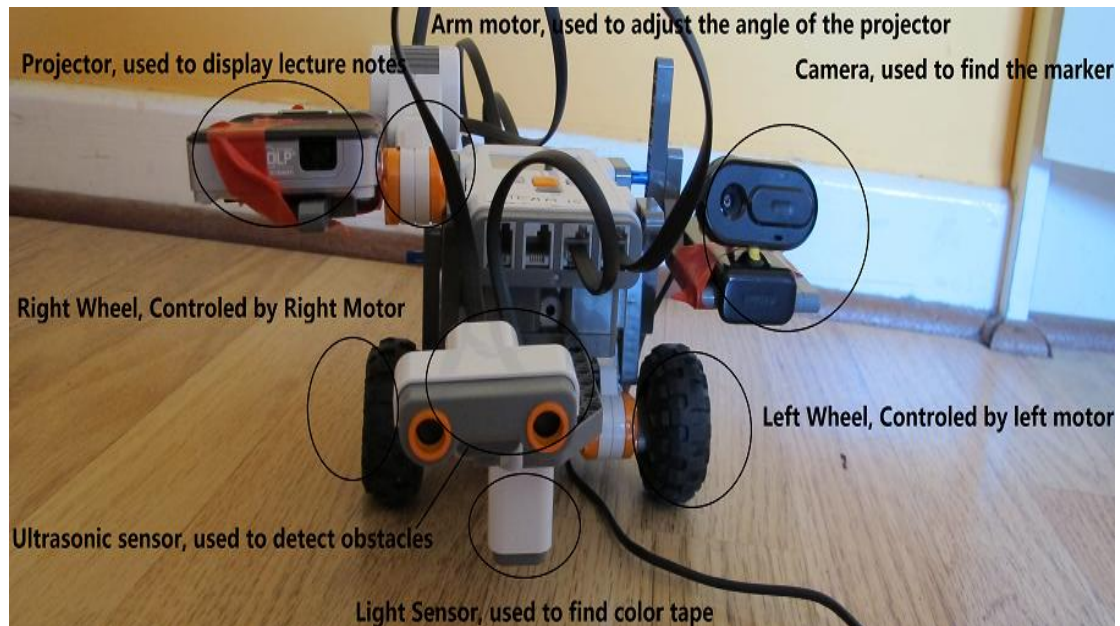


Figure: The robot, final version

## Chapter 5 System Validation

### 5.1 Testing method

#### 5.1.1 Black box testing

Black box testing refers to test the software without seeing any details into the underlying code[19]. Normally, the system is running and the tester perform as user to do some manipulation on the program and see whether the output result is reasonable or correct. Below is a simple diagram shows how Black box testing works:

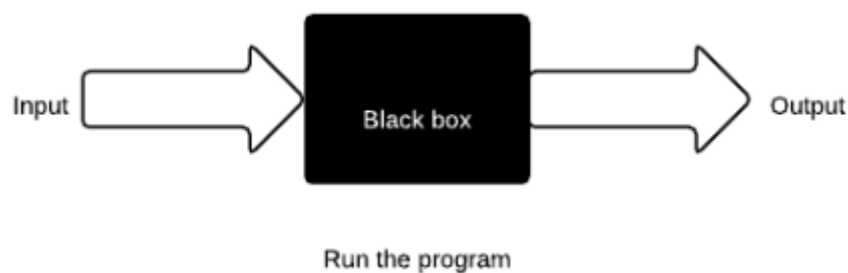


Figure: Black box testing

#### 5.1.2 Reason for choosing Black box testing

In this project, Black box testing is the most appropriate testing method to use to test all the features developed in [Chapter 4](#). Here're the reasons:

- It's hard to visualize the robot state because the limitation of the LEGO Mindstorms' small LCD screen.
- It makes no sense to keep track of the probability value of each grid on a huge map in each measurement step. (Robot Mapping feature)
- It makes no sense to visualize the A\* path finding algorithm internal state when the function is running, the correctness of the algorithm can be easily check by the output waypoints it produced and there're too many states to keep track of. (Path Find feature)
- Seeing a list of light sensor readings and robot rotating degree is equivalent to seeing the robot following a line successfully or not, because the program logic is simple enough. (Line Tracking feature)
- All the information is sent via socket, it's hard to capture a packet to see what's going on in the robot side because the state is consistent so diving into the control flow will not get any benefit in terms of testing the system. (Marker recognition and projection on the whiteboard)

So the author decide to use the Black box testing only in this project, the input can be: user input, component clicked event, or sensor readings; the output will be, the correctness of the robot action, position, etc.

## 5.2 Testing plan

There're three different components(Robot Mapping, Marker recognition and ) in this project, each of them can be divided further into different functionalities. As mentioned in [section 5.1.1](#), this system is mush suitable for Black box testing, so each functionality will be tested based on the evaluation of the output.

Below is a list of sample test case that can be done:

1. Test the Bayes Filter map building function in different environment. E.g. a carpet flooring room, a wood flooring room, a room with complex structure.
2. Test The Path Finding function by choosing different targets. E.g. target room 1, 2, 3, etc. Evaluate whether the path is valid and optimal.
3. Test the Color Tracking function under different light conditions
4. Test the Marker Recognition function under different light conditions.
5. Test the Projection on the whiteboard function by placing the marker at different height.

## 5.3 Testing and results

### 5.3.1 Test build map function

Conditions:

1. A simple wooden floor room:

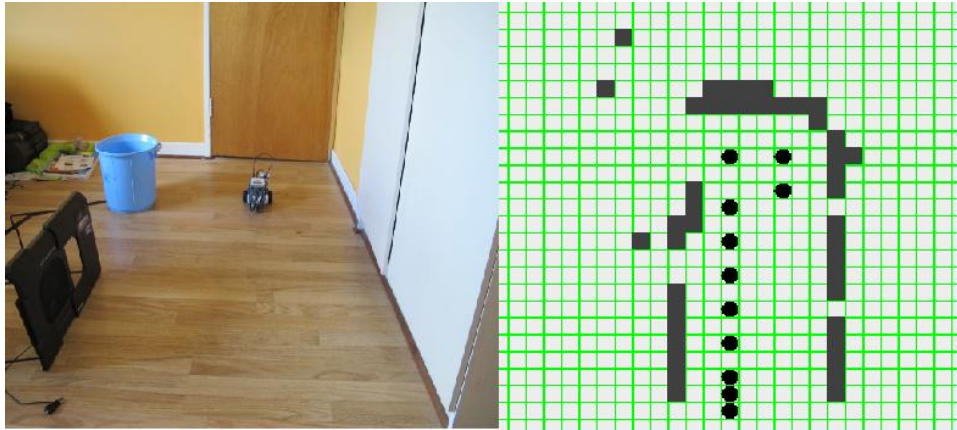


Figure: test map building environment in

As we can see from the photo and the resulting image, the program performs well in this environment. In the image, the black dots denote the robot historical waypoints

## 2. A concrete-floor corridor(simulated):

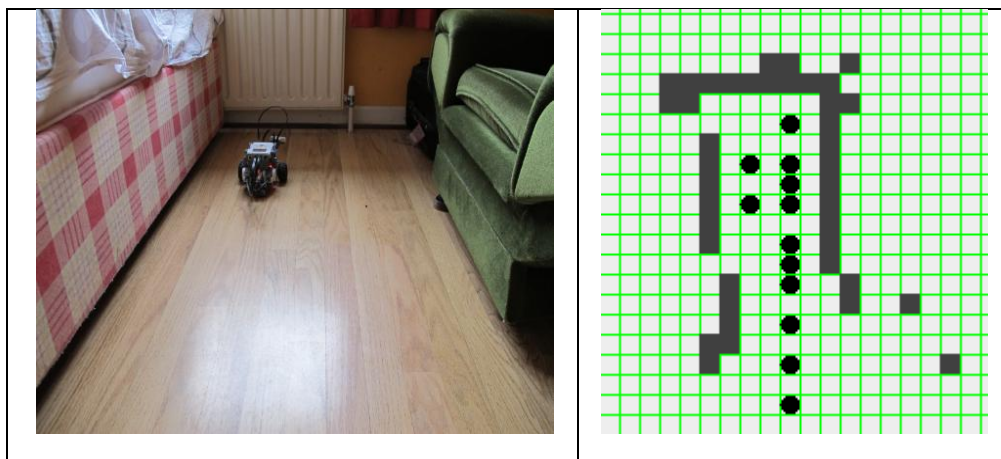


Figure: a map of wooden-floor corridor:

The result is acceptable, even the reading was affected by the material of the surrounding objects, it still generate a reasonable map.

### 5.3.2 Test path finding function

In the following images, the black dot denotes the robot start position and the red dot denotes the robot target position, blue dots denote a list of waypoints that the robot has to achieve.

Conditions:

1. Calculate a path which forms a straight line from start position to the destination

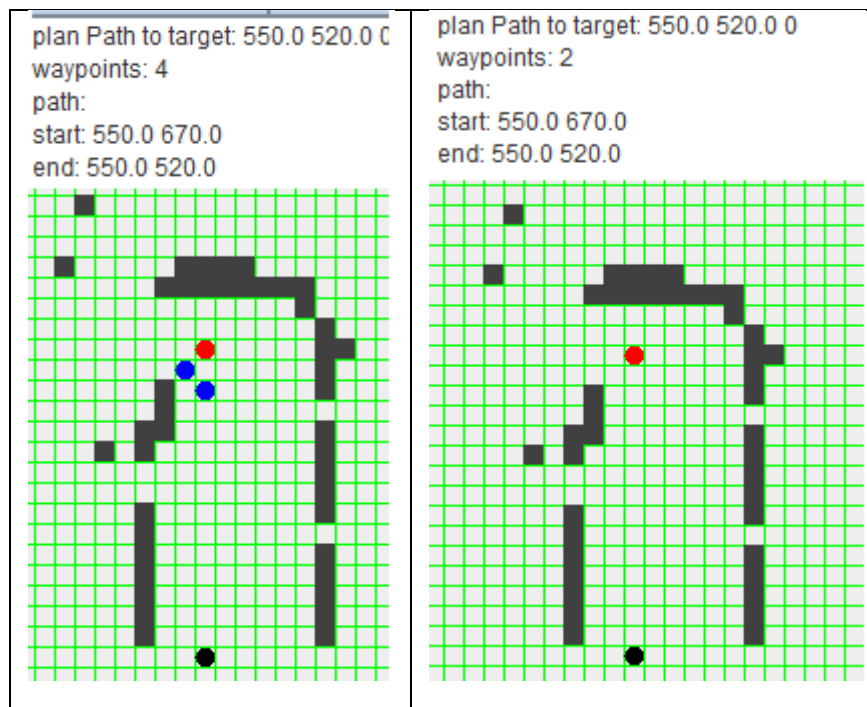


Figure: test path finding program when the path is a straight line, before and after the bug fixed

The text above the grid map is debug text that shows the result of the algorithm, as we can see, before, the program has a tiny bug, so the path includes four waypoints, which means it's not the best solution. After the bug is fixed, the path only includes two waypoints, namely, start position, target position. Pass the test.

2. Calculate a path when the path from start to destination is complicated.

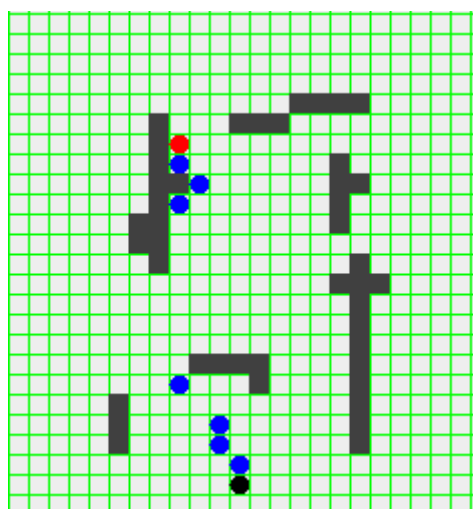


Figure: test the path finding program with a complicated path

The robot find a best path successfully in a complex environment.

### 5.3.3 Test color tracking function

Conditions:

1. Tracking a black tape in a room with normal light condition
2. Tracking a black tape in a bright room
3. Tracking a black tape in a dark room

It's hard to visualize a color tracking program in the report by photo, however, the "bugs" mostly come from the different threshold of "black color", "white color" and other irrelevant color under different light conditions. After the author change the threshold value base on experiment, the function works fine.

### 5.3.4 Test marker recognition function

Conditions:

1. Recognize marker in a bright environment:

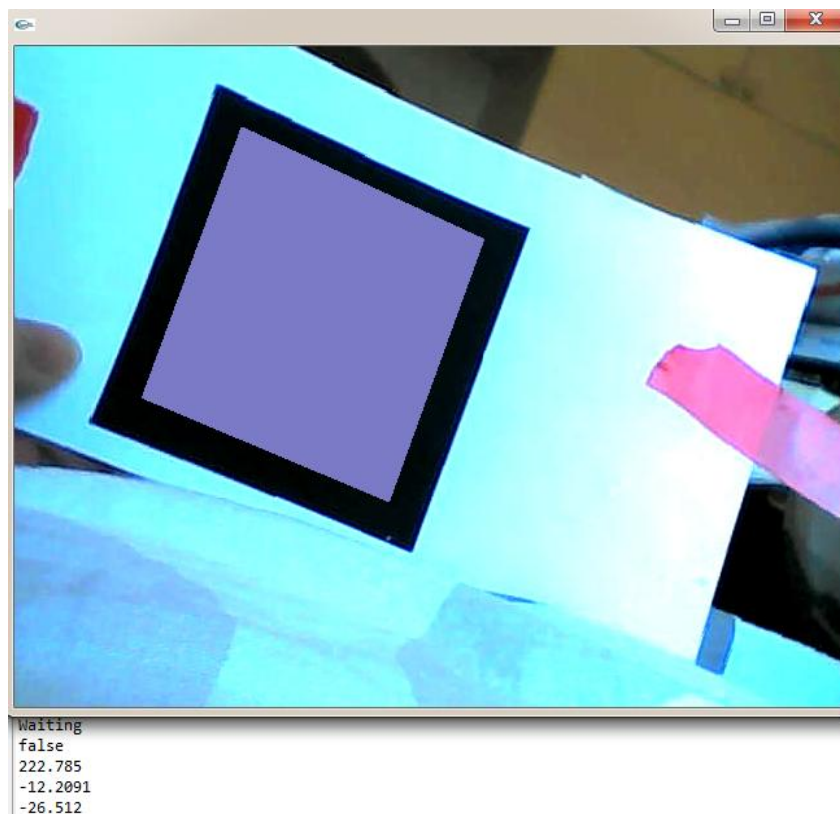


Figure: marker recognition in a bright environment, and resulting coordinate values

Result: The program can recognize a marker without any angle adjustment.

2. Recognize maker in a dark environment



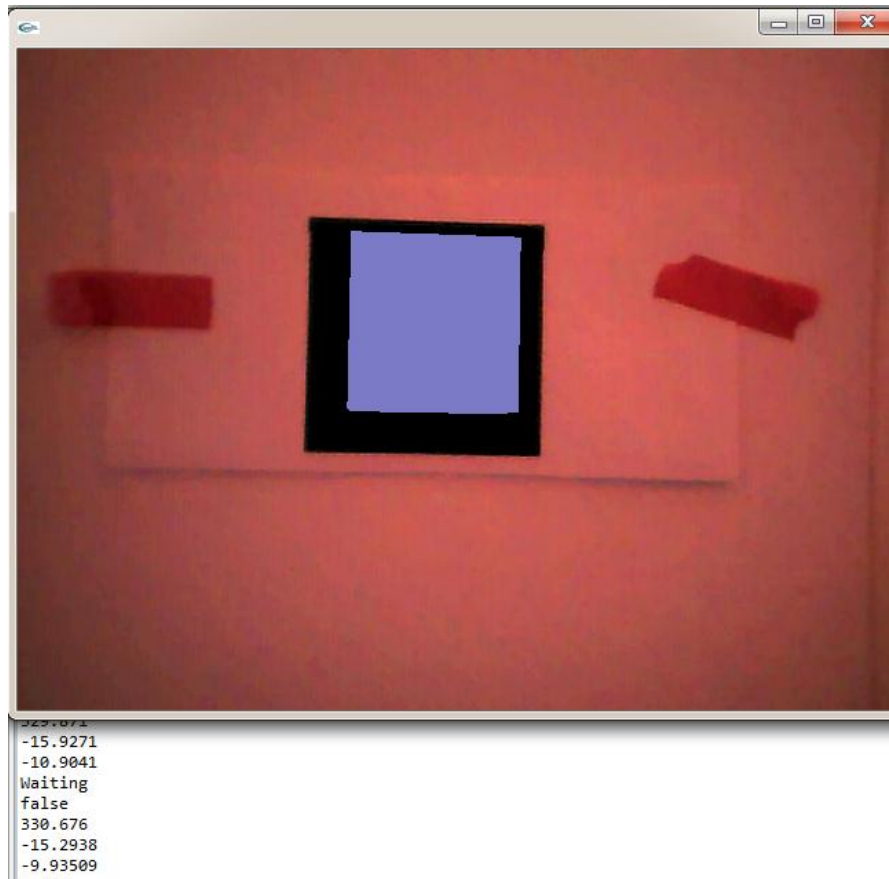


Figure: Recognize marker success, in a dark environment, with coordinate value

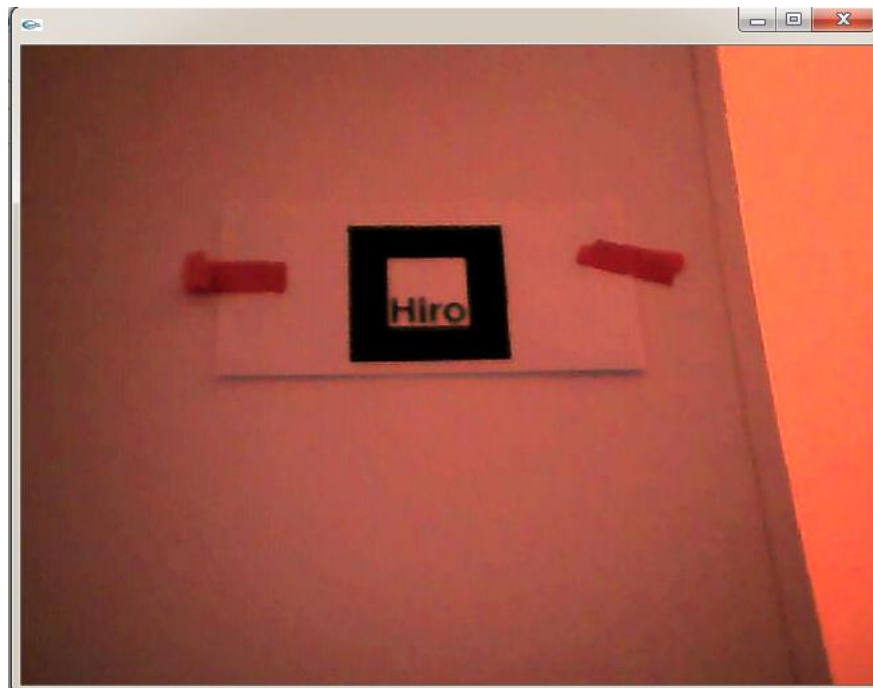


Figure: cannot find marker in a dark environment

The difference is the angle of the camera looks at. If we put the marker right at the centre of the camera view, it will find it, otherwise, it won't.

### 3. Recognize marker in the normal environment:



Figure: Cannot recognize a marker in the normal light condition

The marker cannot be recognized by the program.

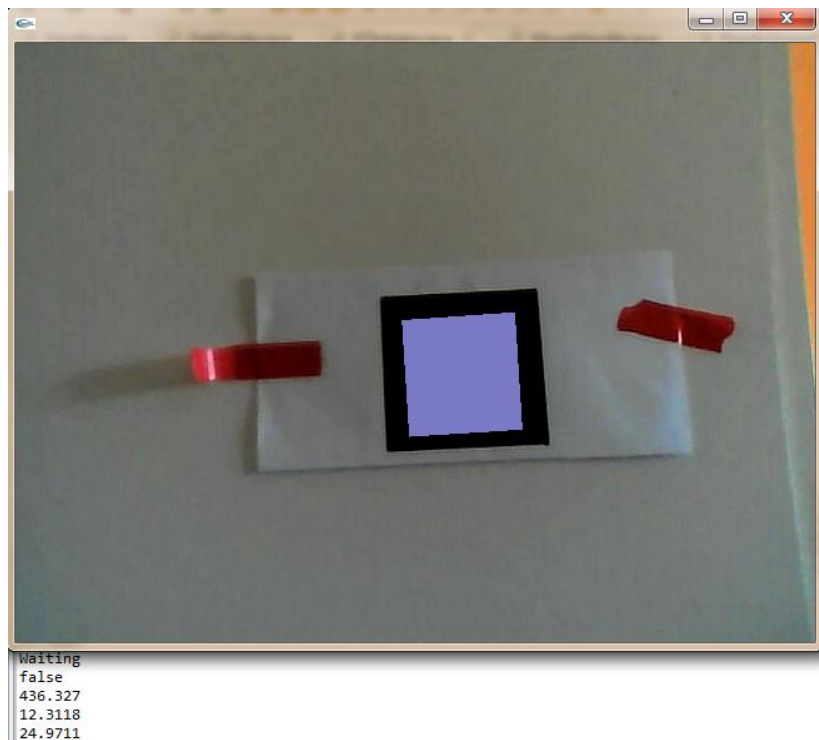


Figure: Recognize a marker in a normal light condition, and resulting coordinate values

After we change the angle of the camera, the marker was eventually found by the program.

These test cases shows that the marker recognition program is not stable if the light is not sufficient.

### 5.3.5 Test projection on the whiteboard function

Put the marker at different height and different distance to the camera

Height 1, distance 1:



Figure: Before and after adjust angle

Height 2, distance 2:



Figure: Before and after adjust angle

The first figure displays the projection perfectly, because we want the project just a little bit higher than the marker position. But the second figure shows a bad result. The cause is the selection of the threshold value that compensate the y coordinate. In the marker recognition program, the x and y coordinate is not in the same scale, x would go to 500 to 1200 normally, whilst the y value only fluctuate between 200 and -200. By carefully setup the environment or re-select the threshold value, this “bug” will be removed.

### 5.3.6 Summary

Here is the summary of the test results:

Test functionality type	Condition	Result
Building map	a wooden floor room	Pass, build a reasonable map
	a wooden-floor corridor	Pass, the result is acceptable
Path finding	Calculate a path which forms a straight line from start position to the destination	Find a bug in the program, fix it immediately.
	Calculate a path when the path from start to destination is complicated. E.g. require many turning	Pass.
Color tracking	Tracking a black tape in a room with normal light condition	Pass.
	Tracking a black tape in a bright room	Pass by change threshold value
	Tracking a black tape in a dark room	Pass by change threshold value
Marker recognition	Recognize marker in a bright environment	Pass.
	Recognize maker in a dark environment	Partial pass.
	Recognize marker in the normal environment	Partial pass.
Projection on the whiteboard	Put the marker at different height	Acceptable, might need to change threshold value to get better result

Table: Test result

## Chapter 6 Project Plan

### 6.1 Previous Plan

Task No.	Task Name	Start	Finish	Duration	Predecessors
1	Feasibility study	Fri 21/10/11	Tue 25/10/11	3 days	
2	Overall model design	Wed 26/10/11	Fri 28/10/11	3 days	1
3	Implementation preparation	Mon 31/10/11	Tue 08/11/11	7 days	
4	Feature 1 Movement	Web 09/11/11	Thu 17/11/11	7 days	3
5	Inspection Feature 1	Fri 18/11/11	Tue 22/11/11	3 days	4
6	Build robot with sensors	Web 18/11/11	Fri 24/11/11	5 days	4
7	Interim report	Fri 25/11/11	Thu 01/12/11	5 days	3
8	Feature 2 Color tracking	Fri 25/11/11	Fri 23/12/11	21 days	6
9	Inspection Feature 2	Thu 26/12/11	Mon 28/12/11	3 days	8
10	Feature 3 AR marker detection	Mon 02/01/12	Thu 02/02/12	24 days	
11	Inspection Feature 3	Fri 03/02/12	Tue 07/02/12	3 days	10
12	Final system development	Web 08/02/12	Thu 15/03/12	27 days	11
13	Final testing	Fri 16/03/12	Web 21/03/12	4 days	12
14	Documentation	Web 26/10/12	Web 21/03/12	106 days	
15	Dissertation	Mon 02/01/12	Web 21/03/12	58 days	7

### 6.2 Actual progress

Some of the stages were not accurate as above “previous plan”, the table below shows the differences(highlighted by red color)

Task No.	Task Name	Start	Finish	Duration	Predecessors
1	Feasibility study	Fri 21/10/11	Tue 25/10/11	3 days	
2	Overall model design	Wed 26/10/11	Fri 28/10/11	3 days	1
3	Implementation preparation	Mon 31/10/11	Tue 08/11/11	7 days	
4	Feature 1 Movement	Wed 09/11/11	Thu 17/11/11	7 days	3
5	Inspection Feature 1	Fri 18/11/11	Tue 22/11/11	3 days	4
6	Build robot with sensors	Wed 18/11/11	Fri 24/11/11	5 days	4
7	Interim report	Fri 25/11/11	Thu 01/12/11	5 days	3
8	Feature 2 Color tracking	Fri 25/11/11	Fri 23/12/11	21 days	6
9	Inspection Feature 2	Thu 26/12/11	Mon 28/12/11	3 days	8
10	Feature 3 AR marker detection	Mon 02/01/12	Fri 20/01/12	15 days	
11	Inspection Feature 3	Fri 23/01/12	Tue 02/02/12	9 days	10
12	Extended feature: Simultaneous localization and Mapping(SLAM)	Wed 08/02/12	Thu 08/03/12	22 days	
13	Final system development (putting them altogether, refactoring, clean-up)	Wed 09/03/12	Fri 16/03/12	6 days	12
14	Final testing	Fri 16/03/12	Wed 21/03/12	4 days	13
15	Documentation	Wed 26/10/12	Wed 21/03/12	106 days	
16	Dissertation	Mon 02/01/12	Wed 21/03/12	58 days	7

### 6.3 Adaption Analysis

As we can see from table above, it actually spends quite small amount of time to implement feature 3: AR marker detection, which means the author overestimates the internal complexity of the feature. After that, the author thought there's plenty of time to try the extended feature: SLAM, but it took a large amount of time to

finish it, and the side effect of the decision leads to the decrease of the final system development time.

It's really a trade-off between a working simple model and an experimental powerful new model. The challenge of the extended feature has gone far beyond the author's current knowledge and skill set, e.g., it requires Kalman filter, statistic sampling, conditional probability, Bayes filter, etc. Even worse is that the author had no idea which kind of filter to use to tackle each of the problem at the beginning.

But the result is quite good, a "stupid SLAM" model was produced by the end. The reason for the author using "stupid SLAM" to name this model is that compare to the general SLAM problem(discussed in [section 2.5](#)) that solve global localization and mapping problem, this project only solve the uncertain sensor reading problem. The development details was shown in [section 4.2.2](#). The robot is able to build a probabilistic map and plan a path based on it. And after the development of the "stupid SLAM" the author in deed learn quite a lot advanced robotics content which are covered in some master's modules. Overall, it was worthwhile to change the plan.

Another point needs to be noticed the time spend on refactoring the code, and clean-up all test info. In development phase, it might be unrealistic to write well structured, clean code as well as produce quick solution. So in the final system development, the author not only has to put all the components together to form a complete system, but also needs to tidy up all debug text and analyse any potential refactoring opportunity.



## **Chapter 7**

### **Conclusion**

#### **7.1 Key challenge**

##### **7.1.1 Unreliable motion and sensing**

In development phase, the most daunting fact that hinder the author's progress is the inherent inaccuracy of motor tacho count, and sensor readings. It's a problem faced by most of the robot systems. In reality, robot localization includes position tracking problem and global localization problem. The author tries to address this problem with particle filters approach, given by [11, 20]. But the time constrains result in fail of full implementation of a probabilistic robotic system. So the author only implement a probabilistic mapping program(Occupancy robot mapping in [section 4.3.2](#)) that demonstrate the understanding in this area and shows how powerful it is.

##### **7.1.2 Construct working robot with limited hardware**

Building a LEGO Mindstorms robot is not easy as it first looks like when a certain functionality is expected. The limitation includes: the number and the form of bricks is limited; there're only a small number of ports(3 ports for motor and 4 ports for sensor); the robot requires stable architecture to reduce the motion inaccuracy. The robot has no built-in webcam and projector, the author has to find a way around.

Finally the author built a symmetric robot that with the webcam as its left hand, the robot arm along with the projector as its right hand. And the ultrasonic sensor and the light sensor was plugged as the head of the robot. At the moment, there's no way to get around the cable though, which is use to send real time image to the marker recognition program. A wireless webcam is the solution but when the author is writing the report, they're either too heavy or the resolution of the image is not high enough for the program to analyse.

##### **7.1.3 Multi-threading, multi-component software design**

Multi-threading programming is crucial in this project is because the system requires fast respond in terms of state changed. For example, in the marker recognition stage, it's very inefficient that the robot perform an action only after the program produces an output, it would slow down the system because the marker recognition program process 30 images in a second while the robot cannot respond at that fast. An ideal programming paradigm would be the recognition program sends parameters 30 times in a second, the robot constantly check the data sent to see whether the value reach a threshold, if not, the robot will keep on doing its current job. By doing so, the performance will increase drastically because there's no synchronization happened in the entire phase. So Multi-threading is an important factor to speed up the system performance. And it's relatively hard to develop a multi-threading program because it requires extra work to avoid dead lock and assure system consistency.



Multi-component software design is hard because we have to consider the state transition from one component to another. What makes it more difficult is that different component have different transition logic. So keeping the system in a valid state by restricting the actions that a user can make is critical is the control system. In this project, the author use switch-case block and a large amount of state flags to address this issue.

## **7.2 Key learning**

### **7.2.1 Probabilistic Robotics**

It's the state of the art in robotics. According to [11], this subject is the core of "Google self driving car". Along with the development, the author has to enrol in the online course and reading tons of papers to learn how to implement those filters mentioned in [section 2.5](#). Due to the time limitation, the author was unable to implement all the filters learned in this project, but it will reduce the learning gap in the later robotics development.

### **7.2.2 Marker recognition**

The author was able to write an augment reality application based on the things learned in writing the marker recognition program. As the smart phones become more and more popular and human-computer interaction technique become mature, such a knowledge will help the author in many aspects, like doing further academic research and seeking a career in augment reality industry.

### **7.2.3 Robotic Programming**

An intelligent robot might includes some or all of these subjects[21]: moving, navigation, localization, mapping, path finding, kinematics, vision. In this project, the author was exposed to almost all of these subjects, so have a brief understanding about the challenges existed. A modern robot system requires even more advance techniques like image processing, GPS localization, infrared sensing. They were not covered in this project, however, by understanding the fundamental paradigm of robotic programming, it's relatively simple to make use of the advance techniques.

## **7.3 Reflection**

### **7.3.1 Critical thought**

This project is far away from perfect. It achieves the goal mentioned in the project proposal, that is, a working system was produced at last, but it can be better in many aspects. In the features that the author implemented, there's too much dependence between one feature and another, and too much assumption that makes the system not flexible enough.

For example, the error will become unacceptable if the environment is too large. When the robot following a path in an environment. the motor error can be ignored provide the place is small and only contains two or three rooms. Why was that? Because the error is too small to affect the robot to find a color tape after the path following. Once the color tape is found, the position is automatically “adjust”. For a bigger environment, the motor error will end up with the robot too far away from the color tape, which results in the system fail to perform its job.

Another problem is that the marker recognition program relies on the light condition of a room. So the system might be useless if a room cannot provide required light density.

### **7.3.2 What if do it next time**

If the author have a chance to do it again, here’s something that want to highlight:

1. Implement a localization algorithm, it can be as simple as using multi-wireless signal to calculate the correct position or as hard as using extended Kalman Filter as mentioned in [section 2.5](#).
2. Use a better image processing library for marker recognition to avoid the rigid light density requirement as the current ARToolKit library.
3. Get a experiment environment. In the college the author study in, the author simply cannot have an working environment. So the author can only do experiment in the apartment instead, which really slows down the project progress.

## **7.4 Future work**

### **7.4.1 Global localization**

Global localization is the key to fully autonomous robot. Given an occupancy map implemented in [section 4.3.2](#), implementing a particle filter is relatively simple[10, 11]. If this function was added to the system, then the robot can localize itself even when it was “kidnapped”. So it will drastically improve the reliability of a robot.

### **7.4.2 Robot guide in an indoor environment**

This project is easy to change to a robot guide system that can guide blind people in a indoor environment, provided it can produce sound. In the project, the author showed how to move a robot to different rooms based on requirement. The new system can be controlled by sound, so when a user tells a robot about which room he or she want to go to, the robot will find a path to the room as showed in this project. Then the robot can “beep” when it’s moving, so that the user can follow the robot to the destination.

### 7.4.3 Implementation on a more reliable, suitable robot

LEGO Mindstorms is a good robot to prototype can test algorithm, but it's nearly impractical to put it into production, it was too small, unstable, and do not provide enough ports to support more functionality. So it would be more meaningful if the idea and the algorithm of this project can be used in a bigger, reliable robot.

## 7.5 Summary

Robotics is an area full of challenge, and these challenges come from hardware as well as software. Building a robust robotic system requires more experiment than a typical software development.

Start from October, 2011 till the end of March, the author develops the system under high pressure. The Microsoft Robotics Development Studio is at beta version at the moment, bugs that results in unresponsive sensor is detected just before Christmas, so the author has to change the programming environment. A more challenge bonus feature called "SLAM" takes the author a month to read papers and source code, but the author do not have time to implement them all.

But finally, it turns out that the project meet the objectives in the interim report, that is:

- The author gained experience in multi-sensor usage and known how to take advantage of each of them.
- A marker recognition program is implemented to detect marker, which is a useful technique in further development.
- The project can control the robot do all the tasks that needed.

## References

- [1] G. N. Desouza, and A. C. Kak, "Vision for mobile robot navigation: a survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 2, pp. 237-267, 2002.
- [2] K. Yuki, M. Sugisaka, and K. Shibata, *Learning of Reaching a Colored Object Based on Direct-Vision-Based Reinforcement Learning and Acquired Internal Representation*.
- [3] S. Ekvall, P. Jensfelt, and D. Kragic, "Integrating Active Mobile Robot Object Recognition and SLAM in Natural Environments," in *International Conference on Intelligent Robots and Systems - IROS*, 2006, pp. 5792-5797.
- [4] S. J. Russell, and P. Norvig, *Artificial intelligence : a modern approach*, 3rd ed., Upper Saddle River, N.J.: Prentice Hall, 2010.
- [5] S. Hassenplug. "NXT Programming Software," 27/11/2011; <http://www.teamhassenplug.org/NXT/NXTSoftware.html>.
- [6] L. Developers. "LEJOS – Java for Lego Mindstorms," October, 2011; [lejos.sourceforge.net](http://lejos.sourceforge.net).
- [7] H. Kato, and M. Billinghurst, "Marker tracking and HMD calibration for a video-based augmented reality conferencing system." pp. 85-94.
- [8] p. lamb. "ARToolKit Documentation," 03/12, 2011; <http://www.hitl.washington.edu/artoolkit/documentation/>; <http://sourceforge.net/projects/artoolkit/>.
- [9] H. Durrant-Whyte, and T. Bailey, "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99-110, 2006.
- [10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*: The MIT Press, 2005.
- [11] S. Thrun, "Programming a Robotic Car," Know Labs, Inc., 2012.
- [12] A. Cockburn, "Using Both Incremental and Iterative Development," 2008, pp. 27-30.
- [13] S. R. Palmer, and J. M. Felsing, *A Practical Guide to Feature-Driven Development*, New Jersey: Prentice Hall, 2002.
- [14] E. Freeman, E. Freeman, B. Bates *et al.*, *Head First Design Patterns*: O' Reilly Associates, Inc., 2004.
- [15] Microsoft. "Model-View-Controller," <http://msdn.microsoft.com/en-us/library/ff649643.aspx>.
- [16] S. Riisgaard, and M. R. Blas, "SLAM for Dummies: A tutorial approach to simultaneous localization and mapping."
- [17] M. Zuliani "RANSAC for Dummies," 2012.
- [18] D. Parker. "Explorer Building Instructions," 17/03, 2012; <http://www.nxtprograms.com/NXT2/explorer/steps.html>.
- [19] R. Patton, *Software Testing*: Sams, 2000.
- [20] W. Kun, S. Liying, W. Shucai *et al.*, "Simultaneous localization and map building based on improved particle filter in grid map." pp. 963-966.
- [21] B. Bagnall, *Intelligence Unleashed: Creating LEGO NXT Robots with Java*, Winnipeg, Manitoba: Variant Press, 2011.

## Appendix

### User Manual

#### Run the code

Assuming the user has already set up the environment(For information about how to setup environment, see the instructions.txt in attached DVD). Then the user can use Eclipse to compile and upload the files to the robot by Bluetooth or cable.<sup>5</sup> The folder fyp.robot contains all code that should be running on the robot side, while the folder fyp.pc contains all code that should be running the PC side. The handler class in the fyp.robot.stupidSLAM package and fyp.pc.stupidSLAM package is the entry point for both package. The user should run the Handler on the robot side first, then run another Handler class on the PC side.

Also, the color tape should be pasted on the ground of the test classroom. The color tape should be reached by the robot through a list of waypoints. So the start of the tape should be at the entrance of the door. The end of the tape must parallel to the whiteboard, and the end of the tape and the marker must form a vertical line to the wall.

#### The user interface

Start up:

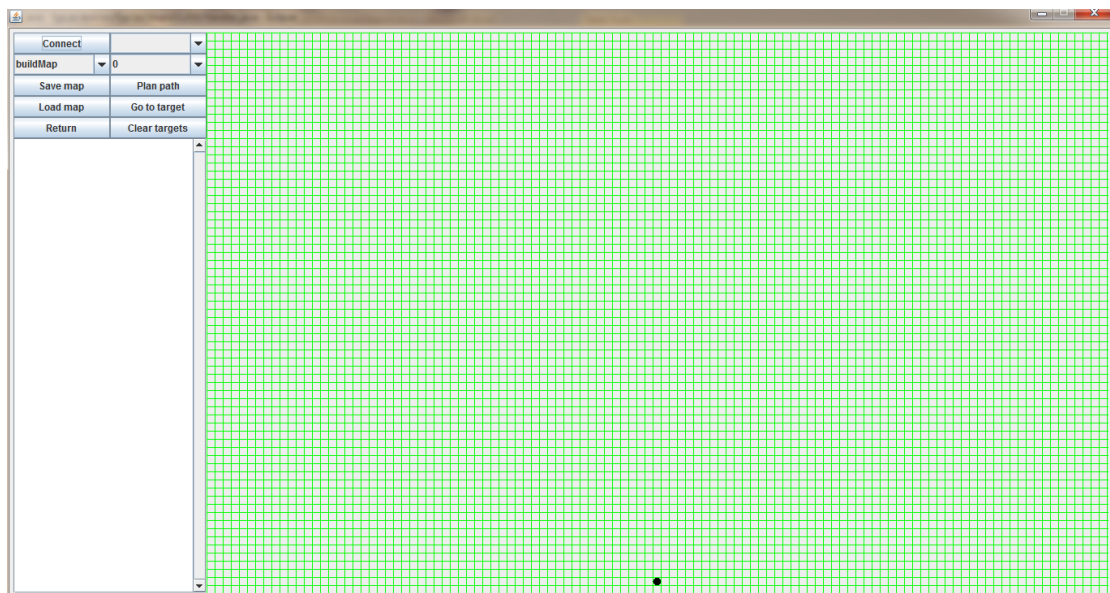


Figure: The user interface

---

<sup>5</sup> For more information about using Eclipse to compile and upload, see <http://lejos.sourceforge.net/nxt/nxi/tutorial/Preliminaries/CompileAndRun.htm>

After the user run the server side Handler successfully, the program start up with an empty map, the black dot denotes the robot current position.

Usage of each button and combo box:

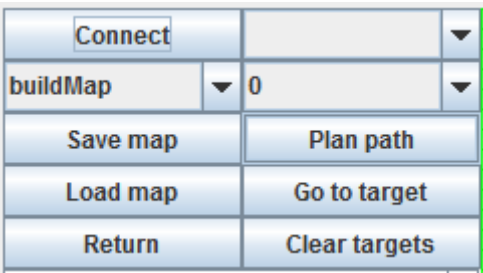


Figure: Controls

First row: the first button is to use to connect to the robot, normally it's not use, just in case the robot miss the connection request at the start. The second control is a combo box, provide a list of targets that for user to choose. It's empty at the beginning, but when the user left click on the map to choose some targets(the target must also be the start of the color tape, otherwise the robot cannot perform following actions), it will contains content like this:

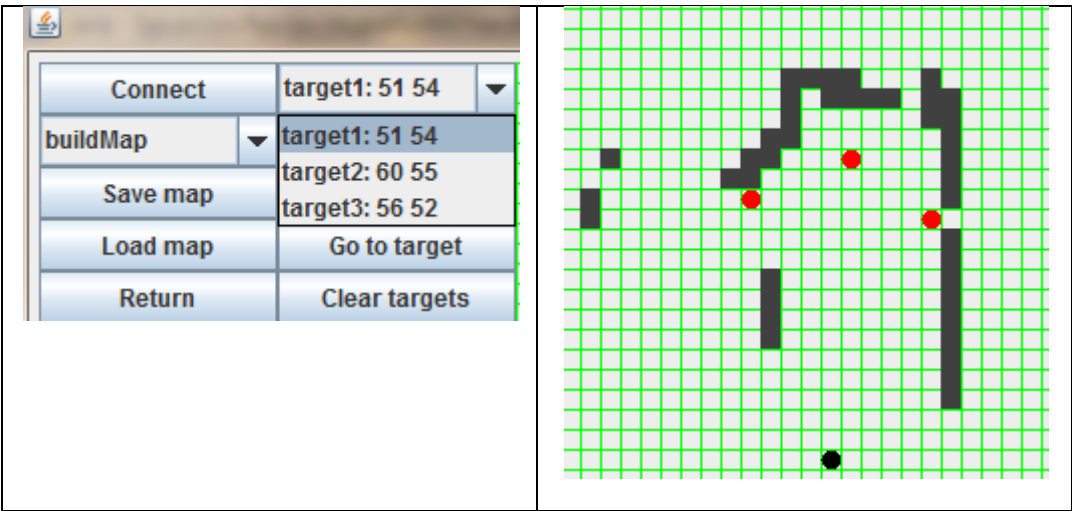


Figure: User interface after select some targets

After a target has been chosen, the user can click the "Plan path", then a list of waypoints will be shown:

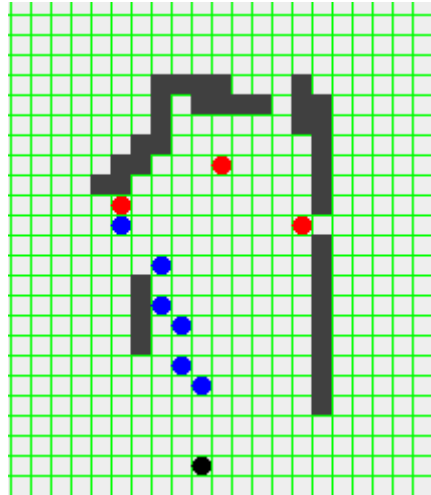


Figure: A list of waypoints to target 1

Then the user can click the “Go to target” button, then the robot will follow the path to the target.

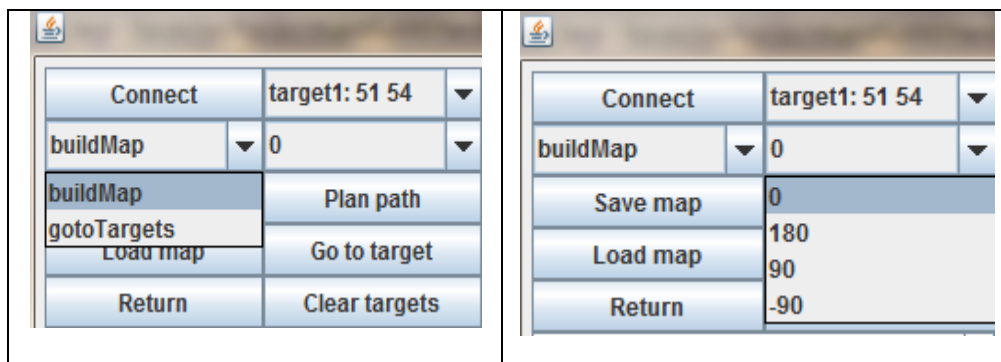


Figure: mode selection and angle selection

The second row:

The first combo box is mode selection, the user can select from two modes, the build map mode is used to guide the robot build a map. The gotoTargets mode is used to ask the robot stop doing ultrasonic reading while moving, which will save time when the map has been built.

The second combo box is angle selection, is was used to rotate the robot when the target reached. Why is that? The robot walk along a corridor has to rotate 90 degree to enter the room, and when the room return, it also have to turn a certain degree to walk along the corridor.

The save map and load map button is easy to understand. Just save a map and build a map.

The return button must be clicked after the projection success, at that time the robot will turn around and keep on following the color track to exit.

The clear targets button is used to clear all targets.