VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi-590018, Karnataka, INDIA

PROJECT REPORT
on

"Charting hazardous indoor spaces using robot swarms"

Submitted in partial fulfillment of the requirements for the VIII Semester

Bachelor of Engineering
IN
COMPUTER SCIENCE AND ENGINEERING

For the Academic year
2017-2018
BY

ABHIJIT JEREMIEL THEOPHILUS	**1PE14CS009**
SURYA PRAKASH M	**1PE14CS154**
VINAY KUMAR ADIGA G	**1PE14CS173**
VIJAYALAXMI HULLATTI	**1PE14CS432**

Under the Guidance of
Prof. Sudarshan T S B
Associate Director &
Professor, Dept. of CSE
PESIT-BSC, Bengaluru-560100

PES INSTITUTIONS

Department of Computer Science and Engineering
PESIT BANGALORE SOUTH CAMPUS
Hosur Road, Bengaluru -560100

PESIT BANGALORE SOUTH CAMPUS

Hosur Road, Bangalore -560100

Department of Computer Science and Engineering

CERTIFICATE

*Certified that the project work entitled **"Charting hazardous indoor spaces using robot swarms"** is a bonafide work carried out by **Abhijit Jeremiel Theophilus, Surya Prakash M, Vinay Kumar Adiga G** and **Vijayalaxmi Hullatti** bearing USN **1PE14CS009, 1PE14CS154, 1PE14CS173** and **1PE14CS432** respectively, students of **PESIT Bangalore South Campus** in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the **Visvesvaraya Technological University**, Belagavi during the year 2017-2018. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated and the project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.*

Signatures:

Project Guide **Dr. Sudarshan T S B** Associate Director & Professor, Dept. of CSE PESIT-BSC, Bengaluru	Head, Dept of CSE **Dr. Sandesh B J** Professor, Dept. of CSE, PESIT-BSC, Bengaluru	Director/Principal **Dr. J. Suryaprasad** PESIT-BSC, Bengaluru

External Viva

Name of the Examiners	**Signature with date**
1. _____	_____
2. _____	_____

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned my effort with success.

We are indebted to our Guide, **Dr. Sudarshan T S B**, Associate Director & Professor, Department of Computer Science and Engineering, PESIT - Bangalore South Campus, who has not only coordinated our work but also given suggestions from time to time.

We are also extremely grateful to our Project Co-ordinators, **Dr. Snehanshu Saha,** Professor**, Prof. Pooja Agarwal,** Associate Professor**, Prof. Jyoti Desai,** Assistant Professors, Department of Computer Science and Engineering, PESIT Bangalore South Campus, for their constant support and advice throughout the course of preparation of this document.

We are greatly thankful to **Dr. Sandesh B J**, Professor and HOD, Department of Computer Science and Engineering, PESIT Bangalore South Campus, for his able guidance, regular source of encouragement and assistance throughout this project.

We would like to express our immense gratitude to **Dr. J. Suryaprasad,** Director and Principal, PESIT Bangalore South Campus, for providing us with excellent infrastructure to complete our project work.

We gratefully acknowledge the help lent out to us by all faculty members of the Department of Computer Science and Engineering, PESIT Bangalore South Campus, at all difficult times. We would also take this opportunity to thank our college management for the facilities provided during the course of the project. Furthermore, we acknowledge the support and feedback of my parents and friends.

Abhijit Jeremiel Theophilus
Surya Prakash M
Vinay Kumar Adiga G
Vijayalaxmi Hullatti

ABSTRACT

Several professional institutions, like the police and fire department, counter terrorism forces, mining organizations, archaeology teams, etc., operate in indoor locations that may prove to be hazardous to personnel and equipment. Prior knowledge of the region, including locations of hazardous elements and routes within the space could mitigate several disasters that occur without preparation. In this project, we propose an architecture for a multi-robot system built around the concept of Swarm Robotics that provides a scalable and robust solution to sweeping and thereby mapping an indoor region. The architecture aims to tackle Simultaneous Localization and Mapping (SLAM) in a bounded region characterized by entrances and indoor walls. We develop an algorithm to ensure coverage during the sweep and adapt it to facilitate dynamically adapting the size of the swarm without any overhead, providing flexibility. We adapt the Physicomimetic Framework for calibrating positions and lattice formations, which includes inter-robot and robot-wall distances. These algorithms are implemented as a simulation on the v-rep simulator and a rudimentary hardware prototype based around an Arduino Uno Rev3 chipset. The simulation is tested in several environments and the results are reported. The hardware was tested primarily for the coverage algorithm.

CONTENTS

Acknowledgement			**i**
Abstract			**ii**
Contents			**iii**
List of figures			**vi**
List of Tables			**vii**
Chapter 1			
1		**Introduction**	**1**
1.1		Purpose	2
1.2		Problem Statement	2
1.3		Literature survey	4
1.4		Proposed system	4
1.5		Summary	4
Chapter 2			
2		**Requirements Specifications**	**5**
2.1		Operating Environment	5
	2.1.1	Hardware Requirements	5
	2.1.2	Software Requirements	6
2.2		Hardware Specification	6
2.3		Software Specification	11
2.4		Functional Requirements	12
2.5		Non-functional requirements	13
2.6		Applications of the System	13
2.7		Summary	14
Chapter 3			
3		**High Level Design**	**15**
3.1		Design Considerations	15

	3.1.1	Assumptions and Dependencies	15
	3.1.2	Goals and Constraints	16
3.2		System Architecture	16
3.3		Data Flow Diagram	17
3.4		Activity Diagram	18
3.5		Sequence Diagram	19
3.6		Summary	20

Chapter 4

4		**Detailed Design**	**22**
4.1		Purpose	22
4.2		Physicomimetic Framework	22
4.3		MiniMapper Design	23
4.4		Blackboard	23
4.5		Sensor and Actuator Drivers	24
4.6		Wall Following Module	24
4.7		Entrance Detection Module	26
4.8		Coordination Module	26
4.9		Summary	28

Chapter 5

5		**Implementation**	**29**
5.1		Programming Language Selection	29
	5.1.1	Python	29
	5.1.2	Lua	29
	5.1.3	C	30
5.2		Platform Selection	30
5.3		Hardware Implementation	30
5.4		Summary	32

Chapter 6

6		**Testing**	**33**

6.1 Software Testing 33
6.2 Testing Process 33
6.2.1 Black Box Testing 33
6.2.2 White Box Testing 34
6.3 Levels of Testing 35
6.3.1 Unit Testing 35
6.3.2 Integration Testing 35
6.3.3 Acceptance Testing 36
6.4 Unit Testing the Main Modules 36
6.4.1 Testing the Physicomimetic Framework 36
6.4.2 Testing the Wall Following Algorithm 37
6.4.3 Testing the Sensors 38
6.4 Integration Testing 39
6.5 Summary 39
Chapter 7
7 Conclusion 40
7.1 Limitations 40
7.2 Future Work 40
Bibliography 41

LIST OF FIGURES

2.1	Microcontrollers for the Hardware Implementation	8
2.2	Sensors for the Hardware Implementation	9
2.3	Actuators for the Hardware Implementation	10
3.1	System Architecture of the MiniMapper	17
3.2	Data Flow Diagram for the MiniMapper	18
3.3	Activity Diagram for the MiniMapper	19
3.6	Sequence Diagram for the MiniMapper	20
4.1	Top View Schematic of the MiniMapper	23
4.2	State Machine for the Entrance Detection Module	27
5.1	Interfacing Sensors to the Arduino Uno	31
5.2	Interfacing Actuators to the Arduino Uno	31
5.3	Final Hardware Implementation	31
6.1	Testing the Physicomimetic Framework	37
6.2	Testing the Wall Following Algorithm	37
6.3	Testing the Ultrasonic Sensor	38
6.4	Running the Simulation	39

LIST OF TABLES

6.1 Unit Tests for the Physicomimetic Framework 36

6.2 Unit Tests for the Wall Following Algorithm 37

6.3 Unit Tests for the Sensors 38

6.4 Integration Tests for the MiniMapper 39

Chapter 1

Introduction

In several professions, there arise situations that involve accessing indoor locations that are either hard to reach, or pose a certain level of risk to human wellbeing. These locations could occur underground, or under extreme circumstances on the surface. Archeology and Mining are examples of professions where such locations are places of exploratory interest, while public services like the Fire department or the Police force would require surveying regions like collapsed structures for search and rescue, or terrorist occupied buildings for counter offensives. Ideally, such regions of interest need to be mapped and studied before personnel and specialized equipment are deployed. Employing robots to map such hazardous and inaccessible regions minimizes human risk and ensures each operation carried out is efficient in cost and effort. Robots designated for such tasks need to be precise, robust and scalable. Precision is the degree of detail in the data collected so as to render a map with a feasible resolution and scale ratio, robustness is the ability of the system to operate in the face of hazards and scalability deals with the ability of the system to increase it's size or capacity to deal with larger areas of interest.

The precision of the data collected by robots, either single or multi-robot, is primarily dictated by the sensors and equipment that they carry. However, the resilience of single or traditional multi-robot systems rests entirely on the durability of individual robots. In a single robot system, if the only agent fails, the system is dead. For traditional multi-robot systems, where each robot has a specific function, if any of the critical robots fail the system is left inoperable. When it comes to scalability, a single robot has an inherent limitation to its reach. Multi-robot systems however can be extended to include more robots. The disadvantage this holds for traditional systems is that the cost of adding an additional robot increases with the complexity of its design. Therefore, for systems with specialized robots, this limits the extent to which the multi-robot system can scale.

Swarm Robotics is a multi-robot paradigm that capitalizes on the advantages that come with using multiple robots while doing away with the complexities involved in them being specialized. Swarm Robotics deals with the organization and operation of a large number of homogeneous autonomous robots with simple functionality and, limited communication and sensing capabilities. The interaction of these robots among themselves and with the environment leads to emergent behavior targeted towards a common goal. Robot swarms are both highly robust and can scale to significant proportions. With large numbers of robots performing repetitive tasks in parallel, the system can still function if a subset of the total

robots is damaged during operation. Further, owing to the simplicity and extreme modularity of a swarm design, it becomes significantly easier to add or drop large numbers of robots from the swarm. These qualities make robot swarms especially suitable for mapping hazardous indoor regions.

1.1 Purpose

The problem of mapping an indoor region with a robot swarm breaks down to three main tasks. The first two tasks are collectively called SLAM, or *Simultaneous Localization and Mapping*, where each robot must be able to determine its current location with respect to a global frame (localization) while surveying its immediate environment. Surveying the environment severs to construct a local map (mapping) using the current position as a frame of reference. Localization ensures that while coalescing the local maps to form the global structure a single frame of reference is used.

The third task is the control and coordination of the robot swarm. With the problem of mapping an unknown region in mind, there are two key aspects of this task — Aggregation and Task Allocation. Aggregation occurs during the initialization phase that controls how robots come together before initiating a task. Task Allocation is concerned with the administration of subtasks to individual robots.

In this project, we aim to design the structure for a robot swarm capable of mapping an unknown bounded region and develop the control algorithms required to do so. These algorithms are designed to work synergistically to tackle the three tasks of the problem. We propose a design for individual robots of the swarm with the sensory and kinetic capabilities required by the control algorithms. Our purpose is to simulate these designs on a virtual platform and develop a prototype of the robot swarm in hardware. This would serve as a proof of concept to the algorithms designed to achieve a distributed variant of SLAM.

1.2 Literature Survey

Single robot mapping of indoor regions was carried about by Stephen Se, et al., using a vision based approach [6]. Their robot was equipped with a trinocular stereo system which generated Scale Invariant Feature Transform (SIFT) features for each image capture. Stereo matching allowed them to match each image with real world coordinates relative to the robot. Once the SIFT features were matched, they used a least squares procedure to compute better camera ego-motion and thereby more accurate localization.

Thrun and Liu proposed an algorithm for multi-robot Simultaneous Localization And Mapping (SLAM) when the initial positions of the robots are unknown and the environment landmarks are ambiguous [9]. Their algorithm, called Sparse Extended Information Filer (SEIF) represents robot maps using a sparse Gaussian Markov random field which allows constant time updates and linear memory requirement. The key features of SEIF are additivity, for incremental contributions, and locality, to ensure updates performed by a robot are confined to its own pose and detected landmarks. Their algorithm was the first to successfully merge local maps into a single global map without knowledge of robot poses and identifiable landmarks. Howard, et al., presented another approach to indoor mapping for large scale heterogeneous robot teams [1]. Their approach involved three distinct classes of robots, mapping robots, leader robots and sensor robots, to explore the environment, build an occupancy grid, identify the object of interest and monitor it through strategic deployment.

Navarro and Matía [3], as well as Tan and Zheng [8] highlight the key features and characteristics of Swarm Robotics along with advantages of using large number of simple homogeneous robots over a few specialized ones. Their papers are compilations of recent advances in the field.

Rothermich, et al., described and studied the behavior of robot swarms in distributed mapping using both simulations and physical robots [5]. Localization was performed by collecting data relative to multiple landmarks with weights assigned to the reliability of landmarks. Task allocation to each robot was dynamically determined by two components. The first component is determined by whether the robot is allowed to move based on the constraints and quality of localization information. The second component calculates a robot's desire to move through gathered local data. Collaborative mapping was performed by superimposing individual maps generated by these robots to construct a complete representation of the environment.

Physicomimetics is a framework introduced by Spears, et al., to coordinating distributed networks of interacting robots [7]. Inspired by molecular dynamics, the proposed framework functions by simulating a virtual force between every pair of robots. The magnitude of this force is inversely proportional to the distance between robots and is attractive if this distance is beyond a threshold R, repulsive if less than R, or zero. The scale of this force determines whether the system simulates a solid or liquid state of matter. Gaseous is simulated by setting threshold R to infinity.

1.3 Proposed System

We propose a multi-robot system built using the swarm heuristic as a scalable and robust solution to mapping indoor spaces. The system would comprise of a set of homogeneous wheel-based robots, called MiniMappers, that coordinate through simple algorithms to conduct the required task. The MiniMapper will be equipped with ranging and motion sensing units to aid movement and localization and a WiFi module for communication. Assuming that the indoor region begins at a single entrance, the swarm would enter the first entrance and traverse the indoor region. Each robot records its sensor values that would be amalgamated offline to construct a single map of the entire region once the swarm exits the required region.

1.4 Problem Statement

This project aims to develop the control algorithms required to perform aggregation, task allocation, localization and mapping by a robot swarm in an unknown environment. These algorithms will be tested in a simulation and implemented at a small scale on rudimentary hardware.

1.5 Summary

This chapter provides a description of the problem of Simultaneous Localization and Mapping (SLAM) and illustrates why a robot swarm is a suitable solution to it. The chapter specifies scalability, robustness and precision to be the advantages of using robot swarms. Work done previously, referenced by us, either in SLAM or robot swarms is also laid down in this chapter. It finally describes the proposed system and the problem we have tackled to build the system.

Chapter 2

Requirements Specifications

The requirements specification of a system involves detailing the properties and consequences of system behavior that impact the reception and applicability of the system. Requirements identify what the system is expected to perform and what conditions it is subjected to. Requirements are categorized and assigned priorities and often change during the course of development. However, the crucial properties of a system are often established well before the system design is conceived. These requirements could either be functional, pertaining to the primary goals of the system, or non-functional, pertaining to the expected performance of the system.

2.1 Operating Environment

The operating environment is the structure within which the system operates. It consists of the conditions subjected to the system and the external entities affecting the system. These requirements dictate the constraints and the expected interactions of the operating environment towards the system.

2.1.1 Hardware Requirements

The following requirements are expected of the operating environment for hardware compatibility of the simulation,

- at least 4 GB of RAM,
- processor clocked at at least 2 GHz,
- a discrete graphic card,
- 500 MB storage space, and
- (optionally) a mouse with a clickable scroll.

We recommend using a mouse over a touchpad as moving the camera around for dynamic views is inconvenient on a touchpad and offers fewer degrees of freedom. A discrete graphic card is highly recommended as it drastically improves the FPS and real time to simulated time interval ratio yielding smoother, more faithful simulations.

2.1.2 Software Requirements

Considering the operating environment, the only critical software requirement is platform compatibility. The Operating System in use must be capable of running both the simulator and the software required to flash the hardware modules. However since v-rep, used for constructing the robot simulation on a physics engine, matplotlib, for running the simpler simulations, and the Arduino IDE are cross platform, the operating system of choice remains to the user. However, we recommend using at least Windows 8, Ubuntu 16.04 or the Mac OS X. Any other flavors of Linux may be used as long as the required library versions are compatible.

2.2 Hardware Specification

The hardware specifications of the MiniMapper include the choice of processor, sensors and actuators constrained by the algorithm requirements and cost. We found the ***Arduino Uno Rev3*** microcontroller a suitable choice for controlling the MiniMapper, as it is a readily available chipset with a large online community that provides a simple interface to the connected hardware. The ***ESP8266*** microcontroller is used to set up the communication channels among the robots of the swarm.

The hardware components used to set up the MiniMapper's sensory capabilities include,

1. the ***HC-SR04*** Ultrasonic Sensor,
2. the ***MPU-6050*** Inertial Measurement Unit, and
3. IR obstacle sensors.

The kinetic capabilities of the MiniMapper are implemented using,

1. four powered wheels connected to the microcontroller through the L298N H Bridge, and
2. an SG90 Micro Servo Motor to control the sweep of the SONAR module.

Arduino Uno Rev3

The Arduino Uno Rev3 is the third revision of the Arduino Uno microcontroller family based on the ATmega328P microprocessor. The chip comes with a preprogrammed bootloader that allows the Arduino IDE to load code onto the chip without the use of external hardware

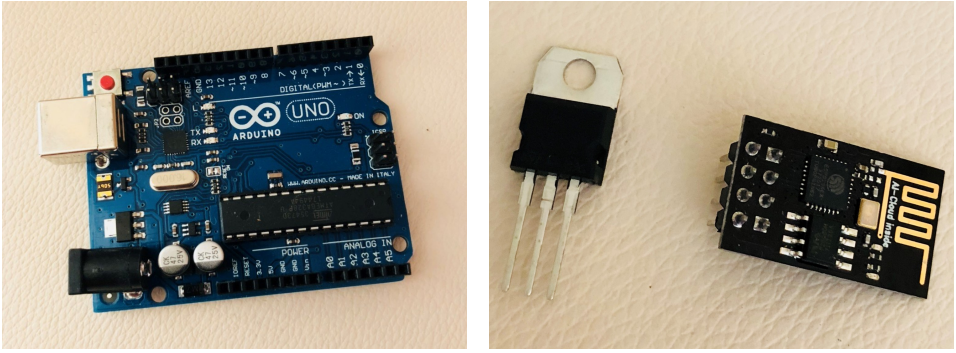
programmers. The bootloader may be bypassed to load code directly using the In-Circuit Serial Programming (ICSP) interface. The chip includes a 16 MHz quartz clock, a USB port and a power jack. The technical specifications of the board are,

- Operating Voltage: 5 V
- Input Voltage: 7 V to 12 V (recommended), 6 V to 20 V (limit)
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current: 20 mA (per I/O Pin), 20 mA (for 3.3 V Pin)
- Flash Memory: 32 kB (0.5 kB used by bootloader)
- Memory: 2 kB (SRAM), 1 kB (EEPROM)
- Clock Speed: 16 MHz
- Physical dimensions: 68.6 mm (length), 53.4 mm (width), 25 g (weight)

ESP8266

The ESP8266 is a low-power integrated solution to Wi-Fi connectivity for a microcontroller. Built around the Tensilica L106 32-bit RISC processor, the ESP8266 comes integrated with the TCP/IP protocol stack and is capable of either hosting an application or offloading the communication module from another microcontroller. This chip can be integrated with other sensors through its GPIO pins and can even be interfaced to an external Bluetooth module. The specification of this chip are,

- Operating Voltage: 2.5 V to 3.6 V
- Operating Current: 80 mA
- Operating Temperature: −40 °C to 125 °C
- Network Protocols: IPv4, TCP/UDP/HTTP/FTP
- Wi-Fi protocols: 802.11 b/g/n
- Security: WPA/WPA2
- Encryption: WEP/TKIP/AES



(a) Arduino Uno Rev3 (b) ESP8266 Wi-Fi Module

Figure 2.1: Microcontrollers for the Hardware Implementation

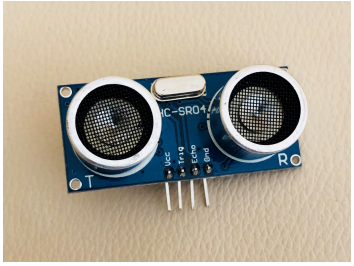
The HC-SR04 Ultrasonic Sensor

The HC-SR04 is a low cost Ultrasonic Ranging Sensor. It uses an audio wave at a frequency beyond the audible range of humans to detect the distance of obstacles. The sensor functions by sending an audio pulse from its transmitter and estimating the distance to an obstacle by the time it takes for the wave to reach its receiver. The control circuitry to operate the transceiver is built into the HC-SR04. The HC-SR04 has an accuracy of 3mm over the range 2cm-4m. Its specifications are,

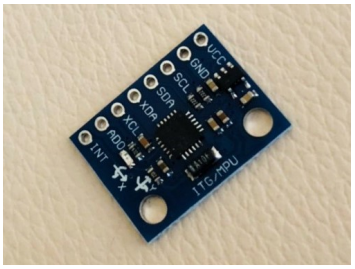
- Operating Voltage: 5 V
- Operating Current: 15 mA
- Operating Frequency: 40 Hz
- Range: 0.02 m to 4 m
- Measuring Angle: 15°

The MPU-6050 Inertial Measurement Unit

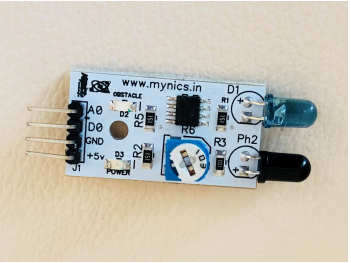
The MPU-6050 is a 6-axis motion tracking device that integrates a 3-axis accelerometer, a 3-axis gyroscope and a Digital Motion Processor (DMP). The device can be extended by connecting additional sensors to its I^2C port. Both the accelerometer and gyroscope have user-programmable ranges. Communication across the registers of the device is performed on the I^2C port. The DMP on the chip offloads motion computation from the system processor minimizing the need for frequent polling. The device also includes a crude temperature sensor. The specifications of this IMU are,



(a) HC-SR04 Sensor



(b) MPU-6050 IMU



(c) IR Obstacle Detector

Figure 2.2: Sensors for the Hardware Implementation

- Operating Voltage: 2.375 V to 3.46 V
- Operating Current: 3.9 mA
- Gyroscope Sensitivity: $\pm 250\,°\mathrm{s}^{-1}$, $\pm 500\,°\mathrm{s}^{-1}$, $\pm 1000\,°\mathrm{s}^{-1}$, $\pm 2000\,°\mathrm{s}^{-1}$
- Accelerometer Sensitivity: ± 2, ± 4, ± 8, ± 16
- Temperature Range: $-40\,°\mathrm{C}$ to $85\,°\mathrm{C}$

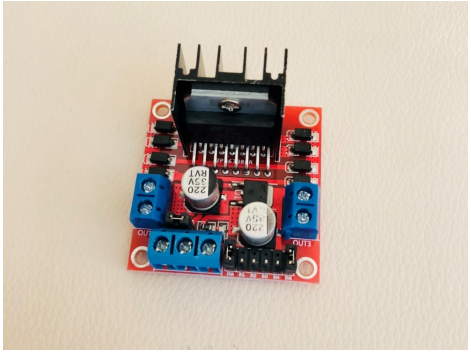
Infrared Obstacle Sensor

The Infrared obstacle sensor consists of an IR transmitter and an IR receiver capable of detecting objects within a short range. The IR transmitter is an LED that emits infrared radiation while the IR receiver is either a photodiode capable of detecting only IR radiations. The detectable distance of the sensor is adjustable through a potentiometer. The sensor works by emitting IR radiation through the transmitter and measuring the received radiation intensity. The specifications of the sensors are,

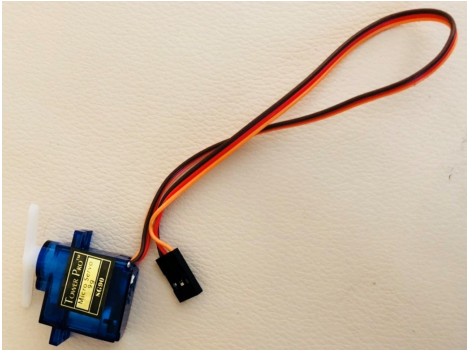
- Operating Voltage: 3.0 V to 5.0 V
- Operating Current: 23 mA to 43 mA (at 3.3 V to 5.0 V respectively)
- Detection Range: 2 cm to 30 cm

L298N H Bridge

To drive a motor in both clockwise and anticlockwise direction, it should be possible to apply the driving voltage in both directions. This is made possible through the use of an H bridge. An H bridge is an electronic circuit, implemented either as an Integrated Circuit or through discrete components, comprising of four switches that determine the direction



(a) L298N H Bridge



(b) SG90 Micro Servo Motor

Figure 2.3: Actuators for the Hardware Implementation

of voltage. The L298N H Bridge is an fully integrated, high voltage, high current H bridge that accepts TTL logic levels as input to drive the switches. The specifications of the device are,

- Driver Voltage: 5 V to 46 V
- Driver Peak Current: 2 A
- Logic Power Output (Vss): 5 V to 7 V
- Logic Current: 0 mA to 36 mA
- Input Voltage Levels: −0.3 V to 1.5 V (low), 2.3 V to Vss (high)
- Max Drive Power: 25 W (at 75 °C)
- Working temperature: −25 °C to 130 °C

SG90 Micro Servo Motor

A Servo Motor is an actuator that controls rotational motion through discrete increments. The motor is usually coupled with a sensor for feedback on the position of the motor. The SG90 Servo Motor has a sweep of 180°, 90° on either side. Its specifications include,

- Operating voltage: 4.8 V
- Operating Speed: 0.1 s/60°
- Stall Torque: 1.8 kgf cm
- Dead Bandwidth: 10 µs
- Temperature Range: 0 °C to 55 °C

2.3 Software Specification

The software specification of the project can be broken down into that pertaining to the simulation and that required for the hardware to function. The simulation of the robot swarm was performed in two phases, the first being a simpler direct simulation of the control algorithms through a plotting software and the other being a more thorough simulation of the model within a physics-engine controlled environment. Designing the algorithms for controlling and coordinating a robot swarm is an arduous and time consuming affair that requires several iterations during both conceptualization and refinement. Modifying these algorithms and reconstructing simulations in a more elaborate simulation takes time. To streamline the development of these algorithms, we construct a simulation of the swarm as an animated scatter plot using a 2D plotting library. The elaborate simulation, built using a robot simulation software utilizing an advanced scripting environment and physics engine, is used to illustrate the final working of the system.

The two kinds of simulations require very distinct software to operate on,

- ***NumPy and SciPy*:** libraries for scientific computing,
- ***Matplotlib*:** a 2D plotting library, and
- ***v-rep*:** a dynamic platform for simulating robots.

Matplotlib is a Python library developed to produce publication quality plots for use in print or interactive environments [2]. It is a part of the SciPy stack and integrates with IPython to provide a powerful interpreted environment for scientific computing. Although Matplotlib excels at 2D plotting, it has been extended to include feature like interactive 3D plots and animations. While Matplotlib functions as the front-end of the first phase of the simulation, the backend is formed over the N-dimensional array object provided by NumPy augmented by the scientific computing routines provided by SciPy.

The second part of the simulation is constructed using the v-rep simulator (virtual robot experimentation platform) built by Coppelia Robotics [4]. v-rep is a versatile and scalable general-purpose robot simulation framework that provides multiple tools and functionalities while abstracting the underlying robotic systems and their complexities. v-rep is cross-platform and provides six different mutually compatible programming approaches over six programming languages,

- Regular API (in C/C++ & Lua),
- Remote API (in C/C++, Java, Python, Matlab, Octave & Lua),

- ROS (Robot Operating System) interfaces,
- BlueZero interface (in C++),
- Plugins (in C/C++), and
- Add-on functions and scripts (in Lua).

We have used the regular API in Lua to program the model. The regular API works fast and is easy to prototype in, and was sufficient for implementing the simulation for this project.

The final software requirement is the integrated development environment for the hardware. The microcontroller used to control each robot is an Arduino Uno Rev3, making the ***Arduino IDE*** the choice of software for implementing the hardware. The IDE is cross platform and provides the necessary drivers to interface the Arduino chipset and load the software onto the board. The choice of programming language for the hardware is C as it provides a native interface to the hardware and has a thriving community for support.

2.4 Functional Requirements

The functional requirements of the system refer to the primary tasks that the system should accomplish. These requirements refer to the functionality the system is expected to provide as a whole. The functional requirements to be adhered to by the MiniMapper Swarm are,

REQ1 The Swarm must be able to completely traverse a bounded region with a single entrance and map every obstacle and wall within that region.

REQ2 The Swarm must be able to aggregate along a wall outside the region boundary and allocate tasks among themselves before initiating mapping of the bounded region.

REQ3 The Swarm must be able to constantly track its position with reference to a global frame while recording all detected obstacles relative to its position.

REQ4 The Swarm must be able to effectively detect entrances to an indoor region and determine whether the entrance has been visited.

REQ5 The Swarm must reduce the number of repeated routes to ensure an efficient route is traversed.

REQ6 The Swarm should be able to detect when the region has been completely traversed.

2.5 Non-Functional Requirements

The non-functional requirements of a system are specifications of the expected emergent behavior of the system during operation. These requirements do not specify what the system should do but rather how it performs in its operating environment. These requirements are critical to acceptable behavior of the system and must be prioritized to provide an optimal trade-off between suitability and cost. For the mapping system, the non-functional requirements are,

- ***Robustness*:** The durability of the system as a whole is a crucial feature when mapping hazardous regions. Even if individual robots are not robust, the sheer number of robots in the system lend it its robustness.
- ***Scalability*:** The system should be easy to extend when the region is expected to be significantly larger than the simulated environment. The control algorithms should be distributed and function independently of the number of robots.
- ***Reliability*:** Sensor readings should be reliable enough to construct a faithful global map. This includes both positioning and obstacle detection sensors as the global map depends heavily on what obstacles are detected where.
- ***Security*:** Communication within the swarm should be secure especially if the region includes possible eavesdroppers.
- ***Usability*:** The system is designed to be usable by any individual. Therefore, most of the initial setup and conduct must be automated.
- ***Maintainability*:** The system should be extensible if additional functionality is required. This extensibility should be in terms of improving both, the individual robot and the swarm itself.
- ***Cost-effectiveness*:** The system should have a clear advantage over single or traditional multi-robot systems in terms of the cost effectiveness. The advantage of robustness and scalability should not come at increased development or production costs.

2.6 Applications of the System

The task of mapping a bounded region has several applications in diverse fields. These applications could either be exploratory in nature or for the purpose of surveillance and monitoring. Examples of such applications include, but are not limited to,

- ***Search and Rescue Operations*:** Disaster management often involves deploying response teams to rescue people trapped in hard to reach surfaces. Such situations often occur in indoor regions due to fires or a collapsing roof. These regions are often hazardous to trained personnel as well, however, knowing the internal structure of the region could greatly improved the relief effort by organizing better evacuation methods.
- ***Urban Counter-Offensives*:** When law enforcement teams need to sweep a building for hostile forces, in situations like a hostage crisis, it is often extremely critical for them to know the internal structure of the building, or even detect a rough location or count of the forces within such a structure.
- ***Mining*:** Mapping out mines is an exploratory example for the given system. Mines could be mapped for hazardous regions and minerals. This would include extending the robot system with modules for detecting the composition of the walls.
- ***Archeology*:** Another exploratory example. Mapping out the interiors of age old buildings with inaccessible regions may be automated using a robot swarm.

2.7 Summary

This chapter discusses the various conditions subject to the system and its expected behavior when interacting with external entities. The system components are discussed and its functional and non-functional requirements laid out. This chapter concludes by describing the various applications of the proposed system.

Chapter 3

High-Level Design

The high-level design of a system includes identifying and describing the key characteristics of the system under development. These characteristics include an overview of the structure of the system, the interactions with the system, the flow of information through the system and the tasks performed by the system. These design decisions form a framework around which the detailed design of components and their implementation is based.

This project involves the development of a simulation and a hardware implementation of the MiniMapper swarm. The swarm involves three main tasks to be performed — Mapping, Localization and Coordination. The high-level design identifies the various modules that tackle these tasks and describes the interactions among them.

3.1 Design Considerations

Before laying out the high-level design of the system, there are a few concerns regarding the system that need to be considered. These concerns outline the direction of the design and the dictate several decisions made throughout the design process.

3.1.1 Assumptions and Dependencies

The first such consideration is to identify the limitations of the system being developed. These include the assumptions on the operating conditions of the system and the extent of its dependencies on external entities.

The assumptions on operating conditions include,

- The environment to be tested in includes only orthogonal edges and right angled corners.
- The surface of the environment is flat, with no significant inclinations.
- The error in sensor readings in the hardware can be simulated in software.
- The swarm is initialized in roughly the direction of the first entrance.

The main dependency of the system is in the simulation which requires the physics engine not introduce a very significant error when simulating actuators. This is essential as most

components in the design assume that setting the velocity in a particular direction once is enough. However, if the errors between actuators is high, this could lead to wayward movements.

3.1.2 Goals and Constraints

The second consideration of the design is the goals and constraints of the system being developed. The primary goal of this project is to verify the control algorithms that coordinate the swarm in a manner that allows it to map a bounded region. These algorithms must be designed to be scalable and robust, and not require external control.

The factors that constrain the design of the swarm include cost and availability of components, simplicity of the design of individual robots, limited processing power on individual robots and the placement of sensors. These constraints stem from the basic definition of a robot swarm, a large number of simple homogeneous robots coordinate to tackle a complex task. These constraints are addressed in the simulation as well as hardware implementation of the system.

3.2 System Architecture

The architecture of a system is a schematic of the various modules of the system, and their relationships, interactions, roles and responsibilities. The system architecture forms the underlying framework for the construction of the system and identifies various aspects of internal behavior at upper layers of abstraction.

The architecture pattern used for the MiniMapper swarm is the Blackboard Architecture. The blackboard design pattern defines three types of components — Knowledge Sources, a Blackboard and a Control component. This pattern is primarily used for problems for which a feasible deterministic solution cannot be constructed. Instead, it relies on several interacting components, called Knowledge Sources, that process raw data individually to generate partial or alternate solutions. These solutions are integrated to find an optimal solution. The Blackboard is a database of information that includes either raw data or partially processed data at multiple levels of abstraction. The Knowledge Sources retrieve information from the blackboard, process it and store the data back into the blackboard. The Control Component coordinates various Knowledge Sources and integrates their solutions.

The Knowledge Sources in the MiniMapper system include,

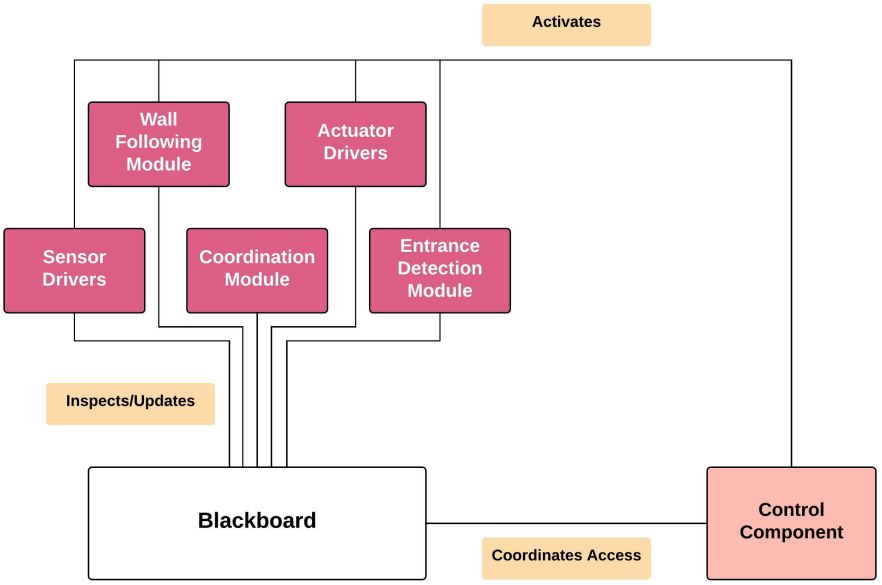


Figure 3.1: System Architecture of the MiniMapper

- ***Sensor Drivers*:** read and record infrared, SONAR and IMU sensor values,
- ***Coordination Module*:** controls forming and maintaining a lattice among individual robots,
- ***Wall Following Module*:** controls movement by ensuring robot follows a wall,
- ***Entrance Detection Module*:** controls detecting, entering or by-passing entrances,
- ***Actuator Drivers*:** read and set joint controller velocities.

3.3 Data Flow Diagram

Data Flow Diagrams (DFD) represent the flow of data within the system, or a subsystem. These diagrams depict processes that transform data, data stores that buffer intermittent results and any external entities that influence data within the system. They serve to illustrate the functioning of the system in terms of data transformations and communication across various subsystems.

The Data Flow Diagram depicted in Figure 3.2 shows the flow of data from the sensors to the motor controllers. The sensor values are read and stored in the Blackboard. The Wall-Following Module, the Coordination Module and the Entrance Detection module use the sensor readings to generate velocities. These velocities are written back to the Blackboard.

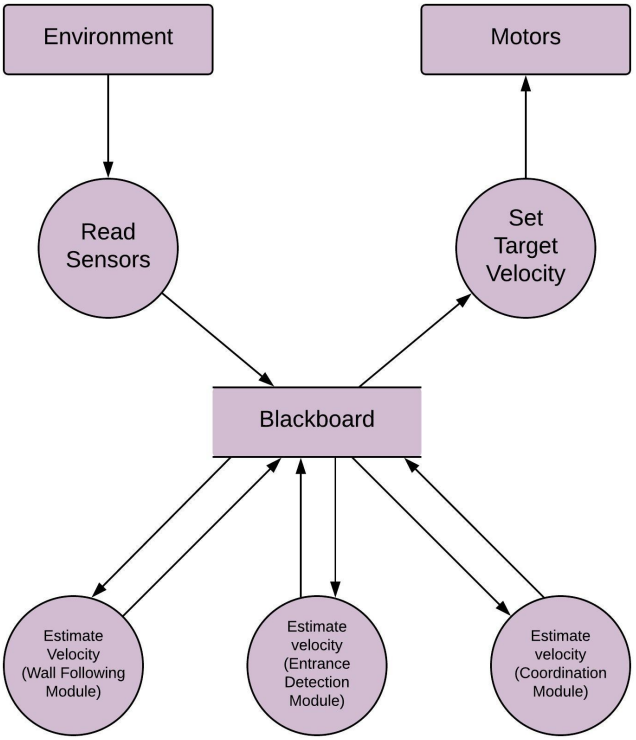


Figure 3.2: Data Flow Diagram for the MiniMapper

The Actuator Drivers read these velocities from the Blackboard, determine a single velocity for each wheel and set the joint controller velocities.

3.4 Activity Diagram

Activity Diagrams are flowcharts that are used to describe the flow of control in an algorithm or computer system. Each node in the activity diagrams represents an action or a decision. Actions depict a single step in the algorithm and decisions depict unconditional or conditional branches, or loops. Activity Diagrams begin at a start node, where data is initialized, and terminated at an end node where data is utilized. They help document complex processes in a clear concise format.

Figure 3.3 depicts the sequence of steps taken during one iteration of the Control Component. Every step requires either a call to a specific module or writing data to the Blackboard. The Control Component gets the sensor readings from the respective drivers. If the Wall-Following module detects a dead-end, velocities are set to turn the MiniMapper 90° right. Otherwise, if the Entrance Detection Module detects an entrance and the entrance is to be entered, the MiniMapper tags the entrance and to enter it sets the velocities to turn

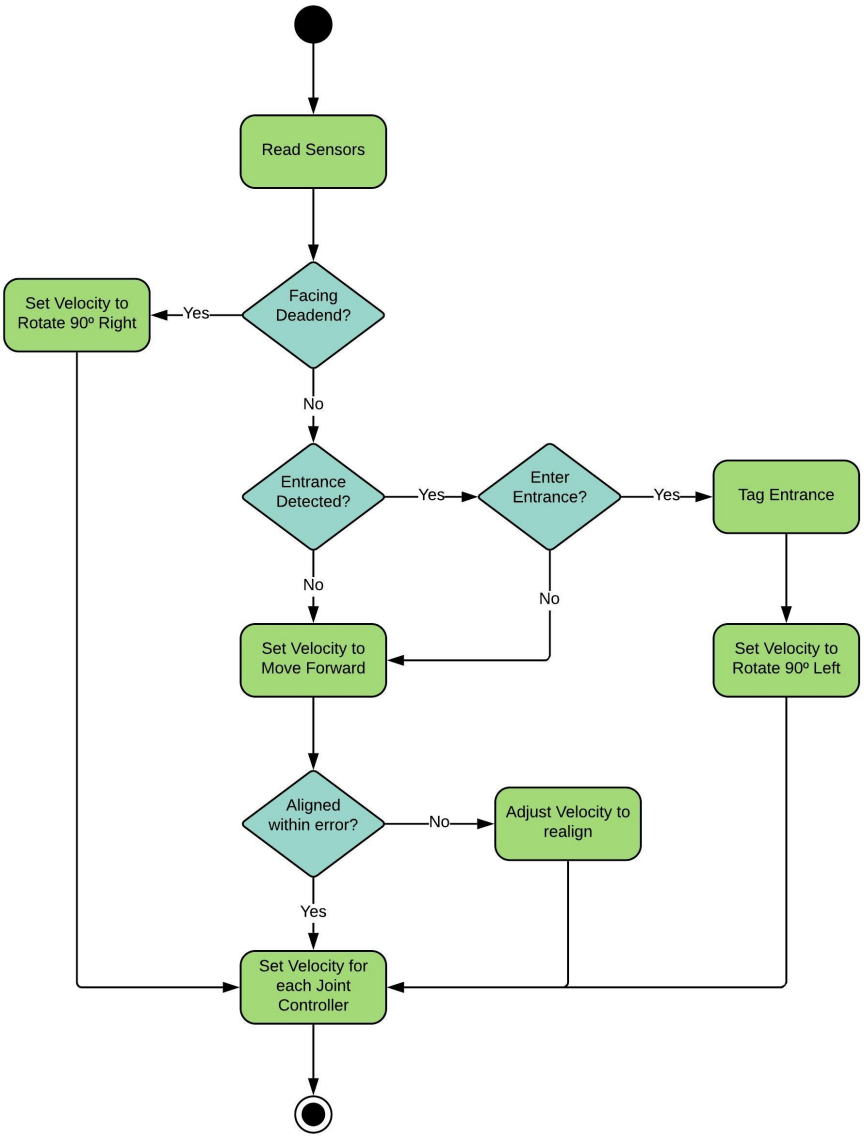


Figure 3.3: Activity Diagram for the MiniMapper

itself 90° left. If either of these do not hold true, the MiniMapper sets velocities to move forward. The Control Component then checks if there is any error in alignment. If there is, the velocity is adjust to account for this error. Once the velocity estimation is completed, these velocities are set for each joint controller by the Actuator Driver.

3.5 Sequence Diagram

Sequence diagrams are used to visualize the flow of logic within a system. This representation can be used for either documentation or validation and is often used to formally analyze

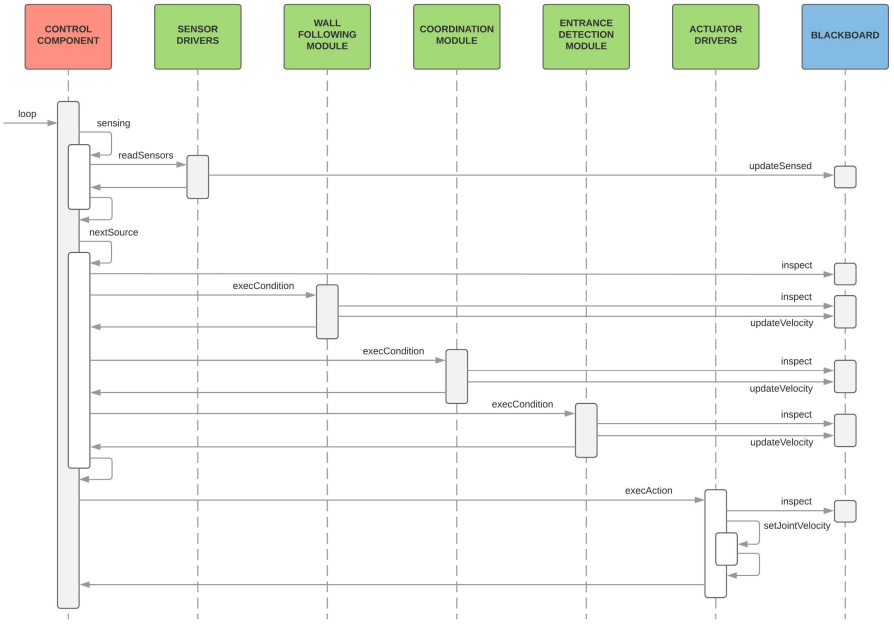


Figure 3.4: Sequence Diagram for the MiniMapper

system specification or design system functions. Sequence diagrams model the dynamic nature of the system and are used to identify dependent behaviors of interacting modules. Sequence diagrams model scenarios, method logic or service logic. Each typical use of the system is modeled as a separate sequence diagram.

The sequence diagram in Figure 3.4 illustrates the sequence of activities undertaken by the system during an iteration of the control loop of the Control Component. The first activity is carried out by the Sensor Drivers. The drivers read the sensor values and update the Blackboard at the start of every iteration. The Control Component then calls `nextSource` to determine which Knowledge Source to invoke. Once a Knowledge Source is chosen, among the Wall Following Module, the coordination Module and the Entrance Detection Module, its corresponding function is called, which checks the sensor values from the database and updates the target velocity. Once the velocity has been updated on the Blackboard, the Control Component invokes the Actuator Drivers that inspects the Blackboard for the updated velocity and sets the joint controllers to this target velocity.

3.6 Summary

This chapter describes the structural and dynamic aspects of the high-level design of the system. It begins by laying out the design considerations to be followed during the development of the system. The chapter identifies the Blackboard architecture as the most

suitable architecture pattern for the problem and describes the various Knowledge Sources to be considered as a part of the Software Architecture. The flow of data and control logic within the system is illustrated through a Data Flow Diagram (DFD) and an Activity Diagram respectively. Interactions among the sub-systems are illustrated through a Sequence Diagram.

Chapter 4

Detailed Design

4.1 Purpose

The detailed design involves describing the structure and behavior of each individual subsystem or module. The design of each system should be coherent with the high-level design in terms of interactions and responsibilities. This phase is the precursor to the implementation phase where most design decisions are finalized to form the groundwork for the implementation.

For the MiniMapper, this phase details the various control algorithms used to coordinate the swarm or control individual robots. We discuss the physicomimetic framework and the algorithm for wall following as part of their individual modules.

4.2 Physicomimetic Framework

The Physicomimetic Framework described by Spears, et al., is an Artificial Physics framework that simulates virtual physics forces between robots [7]. The system acts as a molecular dynamics simulation where the desired formation corresponds to the minimum energy state of the system. The framework treats each robot as a particle i with a position x and velocity v. Consider an interval Δt, the change in position and velocity after an interval is given by,

$$x_{t+1} = x_t + \Delta x \qquad \Delta x = v\Delta t \tag{4.1}$$

$$v_{t+1} = \mu v_t + \Delta v \qquad \Delta v = \frac{F\Delta t}{m} \tag{4.2}$$

The model includes a frictional term, μ for self-stabilization. The force F between particles is given by,

$$F = \frac{Gm_i m_j}{r^p} \tag{4.3}$$

where, m is a virtual mass of the robot, r is the distance between robots i and j and G is a parameter set during initialization. The variable p is a user defined parameter that ranges between -5 to 5. The velocity and force and constrained by v_{max} and F_{max} respectively. The force is attractive if $r > R$ and repulsive if $r < R$, for some threshold R. This setup generates hexagonal lattices. For square lattices, each robot is labeled with one of two colors. Like colors have a separation of $R\sqrt{2}$ instead of R.

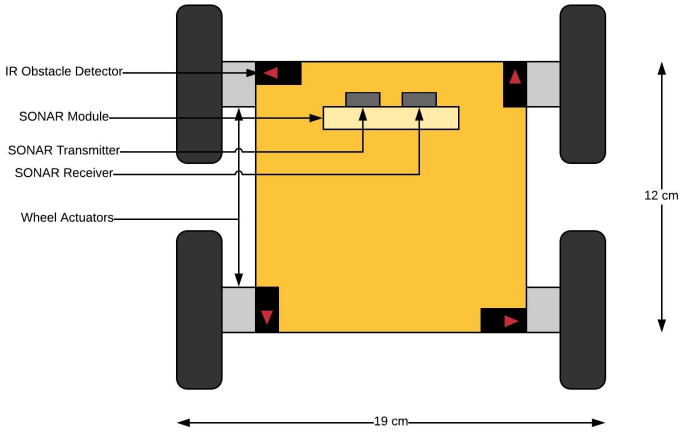


Figure 4.1: Top View Schematic of the MiniMapper

4.3 MiniMapper Design

The MiniMapper is designed as ground-based robot with four wheels, all of which are powered, a SONAR Ranging Module, four IR obstacle detectors and an IMU unit. The top view of the MiniMapper is given in Figure 4.1. The SONAR Module is mounted on top of the MiniMapper on a Servo Motor with a 180° sweep. The four IR sensors, indicated by the black squares with red triangles, are positioned at the corners with the triangles indicating the direction of the ray. Each wheel is connected to a motor controlled through an H-bridge. The wheels may be rotated clockwise or anticlockwise. The movement of the MiniMapper is constrained to either moving forward or back, or turning orthogonally left or right, achieved by rotating the wheels on either side in opposite directions.

4.4 Blackboard

The Blackboard forms the database of information accessed by the Knowledge Bases present in the system. This information may include partially processed data required in stages or data at different levels of abstraction. The Blackboard forms a common ground for aggregating results from various Knowledge Sources and representing results of the system. For the MiniMapper, the Blackboard coordinates the tasks of the Wall Following Module, the Coordination Module and the Entrance Detection Module. The data in the Blackboard is provided by the Sensor Drivers at the start of every iteration and utilized by the Actuator Drivers to drive the joint controllers of the system.

The Blackboard is designed as an associative array, a set of key-value pairs, where

each data store is accessed through its corresponding label. The Blackboard is dynamic in nature and thus extensible if required. For the current implementation of the system, the following data is stored in the Blackboard,

- **`ir`:** Infrared readings stored as an associative array with keys — `front`, `left`, `rear` and `right`, indicating the direction of the ray relative to the robot,
- **`sonar`:** Distance detected by the SONAR.
- **`position`:** Current position as estimated by the DMP on the IMU.
- **`orientation`:** Current orientation as estimated by the DMP on the IMU.
- **`currVelocity`:** Associative array with keys `left` and `right` indicating the current velocities of the left and right side wheels.
- **`nextVelocity`:** Associative array with keys `left` and `right` indicating the updated velocities of the left and right side wheels.
- **`entrStacked`:** Stack consisting of the entrances that have not been exited yet.
- **`entrVisited`:** Set of entrances that have been entered and exited.

4.5 Sensor and Actuator Drivers

The Sensor and Actuator Drivers are modules that abstract the intricacies of interfacing the hardware and allow their access through a single function call. When the Sensor Driver is invoked, it in turn invokes the corresponding lower level driver functions to access the output of each sensor. It writes the resulting IR, SONAR and IMU readings to the Blackboard. The Actuator Driver reads the `nextVelocity` record from the Blackboard, compares it to the current velocity. If there's a change, it updates the current velocity and sets the target velocity of the joint. Both these drivers encapsulate the pin connections, the input voltages to be set on these pins, the detection of output voltages and its translation to appropriate units.

4.6 Wall Following Module

There are two main considerations governing the design of the sweep algorithm,

- An indoor region can be considered to be either a bounded region with a single entrance or comprised of several nested indoor regions. This is ensured by virtually *shutting* an entrance after it has been traversed, as explained in the algorithm.

- The most important criteria for the algorithm to fulfill is coverage. The algorithm should ensure the entire indoor region is covered as much as possible during the sweep, while reducing the number of repeated paths traversed.

The first consideration stresses the importance of detecting and treating entrances distinctly from the rest of the map. By using entrances as reference points, during the course of the sweep, serves the purpose of recalibrating the position and orientation should the error get too severe, and tracking the swarm's progress as it covers the bounded region. The second consideration forms a metric for testing the algorithm. The extent of the region not covered by the algorithm indicates a fallacy in its design.

Algorithm 1 Wall Following algorithm to sweep an indoor region

Require:
maximum IR range IR_{max},
distance from wall $d_{wall} < IR_{max}$,
Blackboard data repository bb.

```
if bb.ir.front < d_wall then
    turnRight()
else if bb.ir.left = IR_max then
    entr ← detectEntrance()
    if entr and entr ∉ entrVisited then
        if entr ∉ entrStacked then
            newtop.position ← entr
            newtop.orientation ← bb.orientation
            push(entrStacked, newtop)
            turnLeft()
        else
            top ← pop(entrStacked)
            if entr ≠ top.position then
                push(entrStacked, top)
                turnLeft()
            else if bb.orientation ≠ top.orientation then
                turnLeft()
            else
                push(entrStacked, top)
            end if
        end if
    end if
end if
moveForward()
```

These considerations led to the development of a Wall-Following algorithm. A wall-following algorithm essentially requires the robot to maintain a fixed distance from a wall

while moving in parallel along a particular direction. Although a simple wall-following algorithm would work for most cases, it fails when a single room has multiple entrances. We devised a modified version of the Wall-Following algorithm, stated in Algorithm 1, to overcome this shortcoming.

The algorithm requires two parameters, the distance to be maintained from the wall (d_{wall}) and the maximum IR range, IR_{max}, and access to the data in the Blackboard. The algorithm begins by checking if the front IR sensor reading drops below the minimum distance from the wall. If it does, the algorithm assumes a dead-end and causes the MiniMapper to turn right. If no entrance is detected, the algorithm checks if an entrance is detected and stores its location in $entr$. If the $entr$ has been visited, no action is taken. If not, it checks if the MiniMapper has come across the entrance before (is in `entrStacked`). If it has not, the entrance location and current orientation is pushed onto the top of the stack. If it has, then the top of `entrStacked` is popped and its position is compared to $entr$. If the positions match, but the orientations differ, then the entrance is being exited. But if the orientations match, then the entrance must be reentered. Finally, regardless of whether a dead-end or entrance was detected the algorithm moves the robot forward.

4.7 Entrance Detection Module

The Entrance Detection module is responsible for identifying entrances as the MiniMapper moves along the edge of the walls. The algorithm is straightforward and relies only on the left IR sensor of the MiniMapper. In order to detect an entrance, the module operates in one of three states, and the value returned by the function call depends on the current state of the module. The first state the module is initialized to indicates that no entrance is detected. In this state, $bb.ir.left = d_{wall}$. The value returned from this state is a boolean false. The second state indicates the possibility of an entrance. Here, $bb.ir.left = IR_{max}$. It also returns false. The third state indicates the confirmation of an entrance. Since the Wall Following algorithm guarantees that the robot is constantly moving forward, all the Entrance Detection Module needs to do is track the distance traveled. If the distance exceeds d_{wall}, the module goes to the Entrance Detected state and returns the current position of the robot.

4.8 Coordination Module

The Coordination Module is responsible for ensuring that the robots maintain their positions and communicate necessary information. This includes notifying neighboring robots when

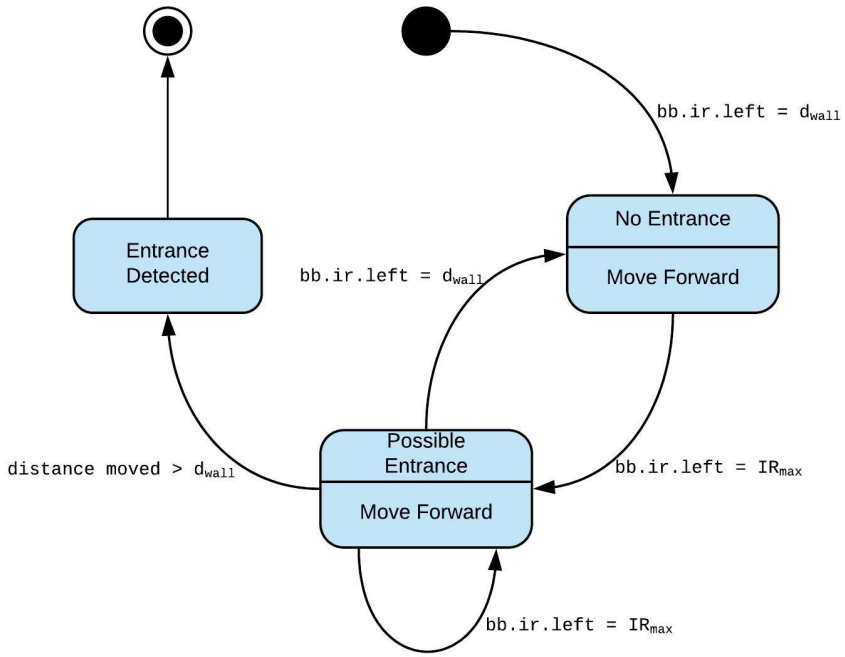


Figure 4.2: State Machine for the Entrance Detection Module

a location has been tagged and ensuring robots do not collide as they traverse the region. The physicomimetic framework, discussed in Section 4.2, forms the groundwork for ensuring robots maintain a required minimum distance between them and the wall, and among themselves. When adjusting the velocity to account for deviation from the wall, R is set to d_{wall}, while a new parameter d_{mim} determines the distance between individual MiniMappers.

Several MiniMappers line up along the outside of the region to be mapped at the start of process. The first entrance is entered regardless of whether it is tagged or not by every robot in the swarm. After the first entrance is visited, each succeeding entrance is tagged by some member of the swarm and pushed to `entrStacked`. This is communicated across the swarm through the Wi-Fi module. Each robot now not only tracks the position of the entrance and its own orientation as it enters, but also the number of robots, $nbots$, relaying tagging the entrance. When it detects an entrance, the MiniMapper determines whether or not it executes lines 14 and 15 in Algorithm 1 based on the probability, $p = 1 - \frac{nbots}{N}$, where N is the total number of robots in the swarm.

4.9 Summary

In this chapter we discuss the detailed design of the various modules in the system. The chapter describes the physicomimetic framework that forms the groundwork for maintaining distances among robots and against the wall. It illustrates the overall design of the MiniMapper, with the placement of sensors and actuators. It establishes the roles and responsibilities of the Blackboard and the Drivers. The chapter further establishes the wall following algorithm, the entrance detection method and the coordination process used in the swarm.

Chapter 5

Implementation

5.1 Programming Language Selection

The three aspects of the project involve demonstrating a basic simulation of the algorithms using a plotting library, developing a more elaborate simulation governed by a physics engine and constructing a rudimentary hardware implementation for the robot swarm. Each of these require different characteristics from the programming tools facilitating their development. We picked Python for the basic simulation, Lua for the final simulation and C as the language of choice for the hardware.

5.1.1 Python

Python is an interpreted, interactive, object-oriented programming language with a focus on a clear concise syntax without sacrificing capability. The language is structured around modules, classes, exceptions and dynamic typing. The language comes with a diverse standard library and interfaces to several system calls. Python was designed to be cross-platform and open-sourced from its inception leading to an extensive community. Python sports a large assortment of third-party libraries built by this community, hosted on a common repository, for a wide range of applications. This streamlines programming in Python leading to higher quality and well tested implementations. For the basic simulation, we use the NumPy and SciPy libraries for number crunching and Matplotlib to render the simulation plots.

5.1.2 Lua

Lua is a powerful yet light-weight language that sports a simple procedural syntax coupled with powerful constructs based on associative arrays and extensible semantics. Lua is interpreted by a fast language engine with a small footprint that can be easily embedded in an external application. Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping. Lua is the language of choice for scripting in the v-rep simulator, that provides a native API to access simulated models and their actuators, a scripting environment and a REPL (Read Print Eval Loop) shell for development.

5.1.3 C

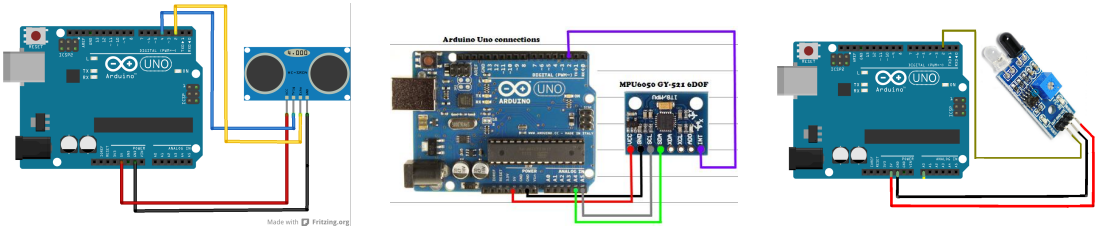
The C programming language was first introduced by Dennis Ritchie, that he demonstrated by providing an implementation to the Unix operating system in C. Since then, the applications and use of C has been as diverse as the reach of computers. It is a general purpose, procedural programming language that compiles directly to assemble code. C defines several constructs that permit low level use of system resources, making it not only possible but also extremely efficient at implementing system programs. Further, being one of the first attempts at a cross platform language, C quickly gained recognition in the application programming domain as well. These reasons coupled with the small memory footprint of a C program makes it ideal for programming the hardware. Supplementing this, the Arduino IDE supports compiling and loading C code to the Arduino Uno natively. The Uno comes preloaded with the firmware necessary to accomplish this task.

5.2 Platform Selection

The simulations are implemented primarily on a system sporting an Intel i5-5200U 2.20 GHz Quad Core processor, 8 GB of RAM and an Nvidia GeForce 940M Graphic Processing Unit (GPU), running the Windows 10 operating system. The rendering software for the basic simulations is *Matplotlib*, an advanced 2D plotting library that supports animated plots, redrawn at user specified intervals. *v-rep* was chosen for the final simulation. It provides an IDE like environment for constructing models and scenes, running control scripts and describing views. The hardware is based around the Arduino Uno Rev3 and its components were chosen according to power compatibility and processing requirement with the Uno.

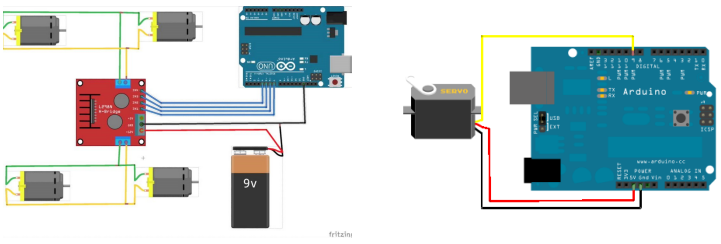
5.3 Hardware Implementation

Hardware programming primarily deals with interfacing the various hardware sensors and actuators to the Arduino Uno Rev3 board. Interfacing these components was straightforward and has been illustrated in Figures 5.1, for the sensors, and 5.2, for the actuators. The connections are made over a breadboard that has not been depicted in the images. Programming these sensors requires setting discrete values on the input pins and interpreting recorded output voltages on the output pins. The frame over which the hardware was constructed uses commonly available materials and was designed to be light-weight and as small as it got. The final hardware implementation is depicted in Figure 5.3.



(a) HC-SR04 Sonar (b) MPU-6050 IMU (c) IR Obstacle Detector

Figure 5.1: Interfacing Sensors to the Arduino Uno



(a) L298N H Bridge (b) SG90 Micro Servo Motor

Figure 5.2: Interfacing Actuators to the Arduino Uno

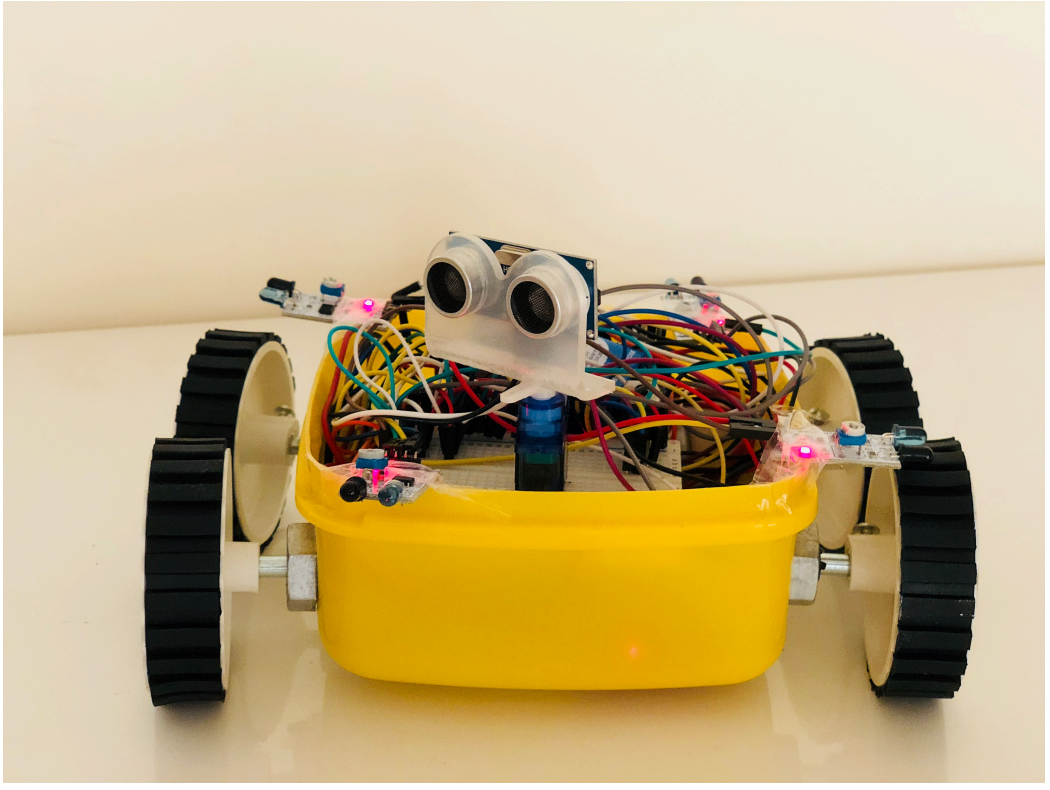


Figure 5.3: Final Hardware Implementation

5.4 Summary

This chapter explains the various implementation peculiarities of the system. It reasons the choice of language and development platform for various aspects of the system. The chapter also describes how each sensor is interfaced to the Arduino Uno for the hardware implementation.

Chapter 6

Testing

6.1 Software Testing

Software testing is the process of executing a program in a simulated environment with the intent of finding faults and failures in the system. Testing includes activities to measure the extent to which the system exhibits its non-functional requirements. Input and output to a system are provided either through simulated or real-world processes. The extent to which the software fails on varying inputs and outputs is improved to a feasible extent, based on the size and complexity of the software system.

6.2 Testing Process

The software testing process is a systematic approach to the testing of the various use cases and scenarios that a software would be subject to. The process involves,

- devising a test strategy, or plan, to guide the rest of the testing process,
- constructing test cases for each scenario or use case of the system,
- executing these test cases to determine where faults lie, whether within a sub-system, at the interfaces, or across sub-systems, and
- generating a report and the closure criteria for the testing process.

At the end of the testing process, a Test Summary and Defect Analysis document may be produced describing the test coverage of the software and the extent to which it exhibits its non-functional requirements.

6.2.1 Black Box Testing

Black box testing focuses on testing the functional requirements of a system. It considers the system to be a black box, where the internal structure is not available to the test designer. Although this technique of testing can be used for non-functional requirements as well, it primarily targets system functions. This method attempts to find,

- incorrect or missing functions,

- interface errors,
- errors in data structures or external database access,
- behavior or performance errors, and
- initialization and termination errors.

The design and development of a system is guided by a set of use cases and scenarios, governing the expected behavior of the system. The expected behavior is described in terms of interactions with external entities called Actors, particularly as the expected output for a given range of inputs. Black box testing aims to validate these expected behaviors of the system and therefore their design involves a say from the stakeholders of the system. The Black box tests for the MiniMapper Swarm involves running the mapping with the swarm in several different environments.

6.2.2 White Box Testing

White-Box testing, also known as structural testing, involves testing the system's internal infrastructure and programming. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. White-box testing targets,

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

At white-box testing, the internal structure of the program is used to determine the test cases. The test cases constructed outline faults at boundaries and internal constraints. The white box tests for the MiniMapper are performed at a module level and include testing the physicomimetic framework, the wall following algorithm and ensuring connectivity among robots in the presence of obstacles.

6.3 Levels of Testing

Testing of a system can occur at three different tiers of the development process. The first, called Unit Testing, is during the development of individual modules. The second, Integration Testing, occurs while interfacing different modules to the final product. The third level of testing occurs just before production begins and is called Acceptance Testing.

6.3.1 Unit Testing

Unit testing is a level of software testing where individual units of a software are tested. The purpose is to validate each unit of the software performs as designed. A unit is the smallest testable part of any software. Its interface is defined by the input it consumes and the output it produces. Units occur at different levels of abstraction. In procedural programming, a unit may be an individual program or function. In object-oriented programming, the smallest unit is a method. Unit testing frameworks, drivers, stubs, and mock/fake objects are used to assist in unit testing.

When it comes to hardware design, each individual sensor or actuator could be a unit in itself. At another level of abstraction, based on how sensors operate they can be grouped together to form a single unit. For example, the H-Bridge and four motors is considered a single unit in the MiniMapper.

6.3.2 Integration Testing

Integration testing is a level of software testing where units are interfaced to each other and tested as a submodule. The purpose of this level of testing is to expose faults due to interacting units. Test drivers and test stubs assist in Integration Testing by abstracting incomplete or resource intensive functionality. The complexity of integration testing grows with the complexity of interaction. Systems with tightly coupled sub-systems demand a greater focus on integration testing than unit testing. Integration testing can be performed incrementally as the various units are attached. For the architecture assumed for the MiniMapper swarm, the main interactions are performed by the Control Unit easing out the process of Integration Testing.

6.3.3 Acceptance Testing

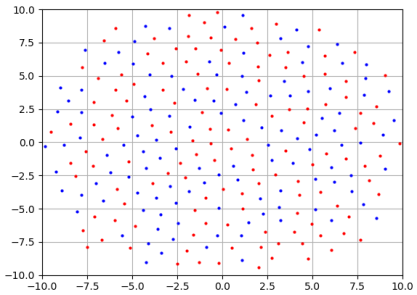
Acceptance testing is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with business requirements and assess whether it is acceptable for delivery. Acceptance testing is performed at the end of the software development lifecycle to indicate the readiness of a software. It is often a variant of the black box testing model, with a focus on evaluating non-functional requirements as opposed to the functional requirements. Acceptance testing for the MiniMapper swarm would involve placing an estimate on its robustness and scalability.

6.4 Unit Testing the Main Modules

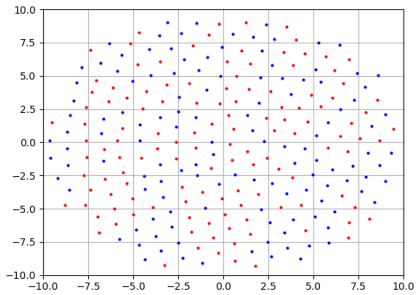
6.4.1 Testing the Physicomimetic Framework

Table 6.1: Unit Tests for the Physicomimetic Framework

Test Case Description	**Parameters**	**Expected Outcome**	**Result**
Hexagonal Lattice with 256 robots	$m = .5$, $R = 1.$, $F_{max} = .2$, $V_{max} = .2$, $p = 3$, $\mu = .01$	Swarm forms a Hexagonal Lattice.	Swarm forms a nearly perfect hexagonal lattice. (Figure 6.1b)
Hexagonal Lattice with 20 robots	$m = .5$, $R = 1.$, $F_{max} = .2$, $V_{max} = .2$, $p = 3$, $\mu = .01$	Swarm forms a Hexagonal Lattice.	Hexagonal patterns can only be observed locally but not globally.
Square Lattice with 256 robots	$m = 1.$, $R = .5$, $F_{max} = .2$, $V_{max} = .1$, $p = 2$, $\mu = .4$	Swarm forms a Square Lattice.	Square lattice noticeable in sections of the swarm. Overall structure slightly hexagonal. (Figure 6.1a)
Square Lattice with 20 robots	$m = 1.$, $R = .5$, $F_{max} = .2$, $V_{max} = .1$, $p = 2$, $\mu = .4$	Swarm forms a Square Lattice.	Square lattice noticeable but further refinement necessary.



(a) Square Lattice formation



(b) Hexagonal Lattice formation

Figure 6.1: Testing the Physicomimetic Framework

6.4.2 Testing the Wall Following Algorithm

Table 6.2: Unit Tests for the Wall Following Algorithm

Test Case Description	**Parameters**	**Expected Outcome**	**Result**
No nested regions.	$d_{wall} = 0.2$	MiniMapper follows every wall.	Expected outcome observed.
Interior room with a single entrance.	$d_{wall} = 0.2$	MiniMapper follows every wall.	Expected outcome observed.
Interior room with two entrances.	$d_{wall} = 0.2$	MiniMapper follows every wall.	The MiniMapper follows a few walls twice. (Figure 6.2)

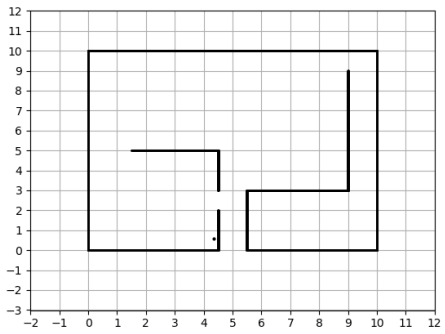


Figure 6.2: Testing the Wall Following Algorithm

6.4.3 Testing the Sensors

Table 6.3: Unit Tests for the Sensors

Test Case Description	**Connections**	**Expected Outcome**	**Result**
Verifying the HC-SR04 Sensor	V_{cc} = 5 V, TRIGGER = Input Pulse, ECHO = Output	Detected range should be 4 cm to 4 m	Sensor operates as expected.
Verifying the IR Sensor	Pin 1 = Output, V_{cc} = 5 V, Pin 3 = GND	Detect object within 3 cm to 20 cm.	Sensor operates as expected.
Verifying the MPU-6050	SDA = Output, SCL = Clock, Pin 7 = GND, V_{cc} = 5 V	Detects accelerometer and gyroscope values for all 6 degrees of freedom	Sensor works as expected.
Verifying the L298N H-Bridge (moving forward)	IN1 = LOW, IN2 = HIGH, IN3 = LOW, IN4 = HIGH	Wheels rotate in the same direction.	Actuator works as expected
Verifying the L298N H-Bridge (turning)	IN1 = LOW, IN2 = HIGH, IN3 = HIGH, IN4 = LOW	Wheels rotate in opposite directions.	Actuator works as expected
Verifying the SG90 Servo Motor	PWM = Input, V_{cc} = 5 V, Pin 3 = GND	Servo Motor moves in increments of 2°.	Actuator works as expected.

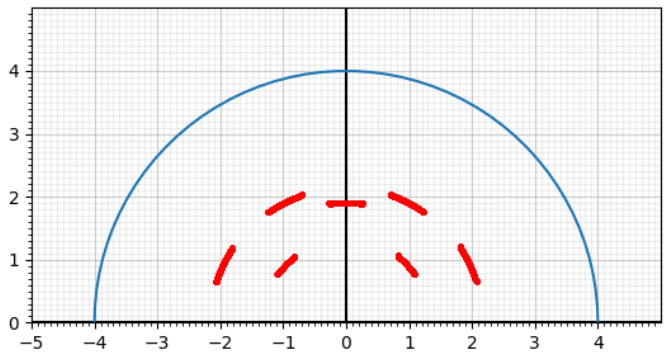
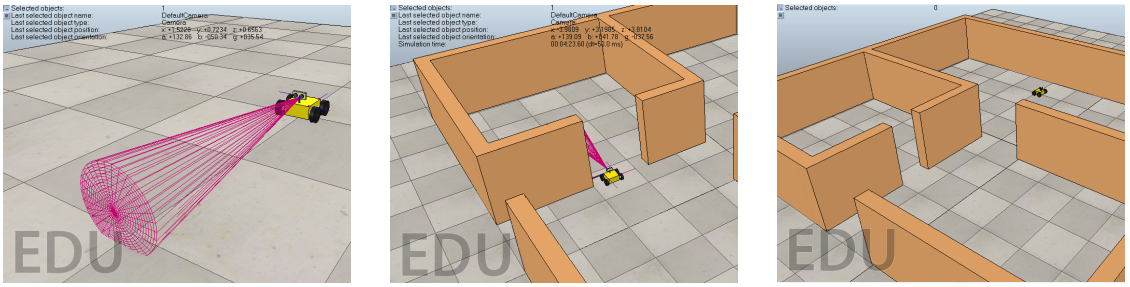


Figure 6.3: Testing the Ultrasonic Sensor

6.5 Integration Testing

Table 6.4: Integration Tests for the MiniMapper

Test Case Description	**Parameters**	**Expected Outcome**	**Result**
Run the Simulation	$d_{wall} = 0.2$, Bullet 2.83 physics engine, nested interiors with multiple entrances	MiniMapper follows every wall.	Although every wall is followed at least once, MiniMapper deviates away from the wall often.
Check Hardware Implementation	$d_{wall} = 0.2$, environment is just an L-shaped wall	MiniMapper approaches corner, turns and continues following the wall.	Expected outcome observed.



(a) Simulation Model (b) Entering an Entrance (c) Following the Wall

Figure 6.4: Running the Simulation

6.6 Summary

Software testing is key to gaining confidence in a software system. It aids weeding out bugs and determining the capability of the system. In this chapter, we describe the various tests carried out on the MiniMapper. These tests include unit tests to verify the functioning of individual modules and integration tests to determine the functioning of the MiniMapper in a simulated and actual environment.

Chapter 7

Conclusion

This project serves to illustrate the need for autonomously mapping an unknown region and describes a system designed to do so. We list out the requirements expected of such a system and develop an architecture and behavior model of such a system. The structural and dynamic aspects of the system are detailed by decomposing the system into modules, each of which tackle one aspect of the process — sweeping the region, tagging recognizable landmarks and cooperating among participants. We devised a Wall Following algorithm to sweep a bounded unknown indoor region and modified it to tackle multiple entrances and cooperation among multiple robots. The working of these algorithms are simulated using the Matplotlib plotting library, assuming individual robots to be points in space. Once the module was finalized, they were integrated and implemented on the v-rep simulation environment as a demonstration. We also build a simple hardware prototype of the simulation to showcase its functioning in a real world setup.

The MiniMapper design is cost-effective and the algorithms are developed to scale to increasing number of robots in the system through simple coordination techniques. Although the algorithms seem simplistic, by using a large number of robots, most indoor regions can be sweeped, provided their entrances are not too large to detect. This goes to show the efficiency of using simple robot architectures to perform complex tasks.

7.1 Limitations

The project was developed with several constraints in mind. This included using fairly cheap sensors with high error probabilities and low resolution. Further, the control algorithms require walls to be orthogonal to each other and the surface of the environment to be flat.

7.2 Future Work

The sensors and actuators on the MiniMapper can be extended to be used in several subtasks while performing a sweep of the region. These subtasks could include detecting people in distressed through visual or audio sensors, detecting hostiles through IR temperature, or identifying minerals in a mine.

Bibliography

[1] Andrew Howard, Lynne E Parker, and Gaurav S Sukhatme. The sdr experience: Experiments with a large-scale heterogeneous mobile robot team. In *Experimental Robotics IX*, pages 121–130. Springer, 2006.

[2] J D Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[3] Iñaki Navarro and Fernando Matía. An introduction to swarm robotics. *ISRN Robotics*, 2013, 2012.

[4] E Rohmer, S P N Singh, and M Freese. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[5] Joseph A Rothermich, M İhsan Ecemiş, and Paolo Gaudiano. Distributed localization and mapping with a robotic swarm. In *International Workshop on Swarm Robotics*, pages 58–69. Springer, 2004.

[6] Stephen Se, David Lowe, and Jim Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 2051–2058. IEEE, 2001.

[7] William M Spears, Diana F Spears, Rodney Heil, Wesley Kerr, and Suranga Hettiarachchi. An overview of physicomimetics. In *International Workshop on Swarm Robotics*, pages 84–97. Springer, 2004.

[8] Ying Tan and Zhong-yang Zheng. Research advance in swarm robotics. *Defence Technology*, 9(1):18–39, 2013.

[9] Sebastian Thrun and Yufeng Liu. Multi-robot slam with sparse extended information filers. *Robotics Research*, pages 254–266, 2005.

TEAM INFORMATION

Abhijit Jeremiel Theophilus

1PE14CS009

+91 78990 79069
abhijit.theo@gmail.com

Aditi 17, Aditi Ahilya Complex, Senior Naval Officers Residential Area, Colaba, Mumbai 400005

Surya Prakash M

1PE14CS154

+91 97319 54155
starsurya96@gmail.com

Flat #901, Pride Picassa, Domlur 2nd stage, Indiranagar, Bangalore 560008

Vinay Kumar Adiga G

1PE14CS173

+91 9916 041562
vinayadiga96@gmail.com

#37/1, Sharadha Nilaya, Masjid Road, Devasandra, Krpura, Bangalore 560036

Vijayalaxmi Hullatti

1PE14CS432

+91 99007 67526
vijayalaxmi.6868@gmail.com

Shree Venkateshwara Nilaya #30/1, Krishnareddy Layout, Konappana Agrahara, Bangalore 560100