VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

Project Report on

"Investigation and Design of Swarm Intelligence Methodologies Applying Machine Learning for Terrain Mapping"

Submitted in the partial fulfillment for the requirements of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

Mr. Gaurav Vattikuti	USN: 1BY13CS026
Mr. Kishan Gupta K M	USN: 1BY13CS037
Mr. Prajwal D Prakash	USN: 1BY13CS053

Under the guidance of

Dr. Bharathi Malakreddy A
Professor,
Department of CSE, BMSIT&M

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064.
2016 - 2017

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU – 560064

<u>DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING</u>

CERTIFICATE

This is to certify that the Project work entitled **"Investigation and Design of Swarm Intelligence Methodologies Applying Machine Learning for Terrain Mapping"** is a bonafide work carried out by **Mr. Gaurav Vattikuti (1BY13CS026), Mr. Kishan Gupta K M (1BY13CS037), Mr. Prajwal D Prakash (1BY13CS053),** in partial fulfillment for the award of **Bachelor of Engineering Degree in Computer Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2016 - 2017. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report. The project report has been approved as it satisfies the academic requirements in respect of project work for B.E Degree.

Signature of the Guide	**Signature of the HOD**	**Signature of Principal**
Dr. Bharathi Malakreddy A Professor, Dept. of CSE, BMSIT&M	Dr. Thippeswamy G Professor & HOD, Dept. of CSE, BMSIT&M	Dr. Mohan Babu G N Principal, BMSIT&M

External VIVA-VOCE

Name of the Examiners	Signature with Date
1.	
2.	

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and everyone who has helped us make this project a success.

We thank our **Principal, Dr. Mohan Babu G N, BMS Institute of Technology & Management,** for his constant encouragement to take up innovative projects.

We thank our **Head of the Department, Dr. Thippeswamy G, Department of Computer Science and Engineering, BMS Institute of Technology & Management,** for having supported us by provinding all the facilities we needed to complete this project.

We thank our Project Guide, **Dr. Bharathi Malakreddy A, Professor, Department of Computer Science and Engineering,** for her constant support, motivation and guidance.

Special thanks to all the staff members of Computer Science Department for their help and kind co-operation.

Lastly we thank our parents and friends for the support and encouragement given to us in completing this precious work successfully.

Mr. Gaurav Vattikuti (1BY13CS026)
Mr. Kishan Gupta K M (1BY13CS037)
Mr. Prajwal D Prakash (1BY13CS053)

ABSTRACT

Our goal in this project is to build a system of robots that can autonomously generate a map of an unknown location. We use three ground robots with movement and obstacle avoiding capabilities.

We have used and tested various swarm dispersion and exploration algorithms, both in simulations and in physical robots. These algorithms allow the robots to use simple behaviours to efficiently cover a given environment.

Two types of terrain mapping, namely radar and computer vision systems were evaluated. We concluded that a computer vision system that performs visual SLAM (Simultaneous Localization And Mapping) is preferable to a radar-based system.

We have also implemented a machine learning system that is capable of traversing uneven terrain in simulation by solving the mountain car problem from OpenAI gym.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1. INTRODUCTION

1.1 Background

Exploring an unknown environment using a team of autonomous mobile robots is an important task in many real world applications. The exploration and mapping of an unknown environment, which is also defined as simultaneous localization and mapping (SLAM), is a fundamental problem for mobile robots and multirobot systems. Many applications such as rescue, mowing and cleaning require full coverage of the environment, in the fastest time possible (especially in rescue operations).

It is preferable to use several robots for this task instead of a single unit. Doing so allows us to maximise the speed of operation and provides redundancy in case of failure. Additionally, if each robot has certain unique capabilities, the task can be split among the robots in a way that takes advantage of individual strengths. But in order to avoid the exploration of the same surface by more than one robot, there is a need to develop a strategy which allows the robots to coordinate their movements. In the collective exploration of an unknown space, each robot must: maintain communication with other robots to share the progress and the discovered surface over a period T, detect and distinguish between obstacles and free space, avoid obstacles when moving, and choose the best path where a robot can move to discover a maximum space that hasn't been discovered yet. Swarm Intelligence techniques are used to implement such functionalities.

Navigation of autonomous vehicles in unknown environments poses a traversability problem. It may not be possible to hard-code a set of actions the autonomous vehicle should take (e.g., increase speed in one the wheels, revert back by a certain distance and attempt to traverse the path again etc.) for every type of environment terrain surrounding the autonomous vehicle. In such cases, it may be useful to explore the application of machine learning techniques to enable the autonomous vehicle to adapt to its environment.

1.2 Swarm Intelligence (SI)

Swarm intelligent systems typically consist of a population of simple agents that follow simple rules. These agents interact locally with one another and with their environment.

These interactions result in the emergence of intelligent, collective, self-organised, global behaviour unknown to a single agent.

The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave. Local – and to a certain degree, random – interactions between such agents lead to the emergence of “intelligent” global behavior, unknown to the individual agents. Examples in natural systems of SI include ant colonies, bird flocking, animal herding, bacterial growth, fish schooling and microbial intelligence.

To illustrate this, Craig Reynolds developed an artificial life program called Boids in 1986, which simulates the flocking behaviour of birds[1]. The complexity of Boids arises from the interaction of individual agents (the Boids, in this case) adhering to a set of simple rules. The rules applied in the simplest Boids world are as follows:

- **Separation**: steer to avoid crowding local flock mates.
- **Alignment**: steer towards the average heading of local flock mates.
- **Cohesion**: steer to move toward the average position (center of mass) of local flock mates.

More complex rules can be added, such as obstacle avoidance and goal seeking.

1.2.1 Swarm Robotics

The application of swarm principles to robots is called swarm robotics, while ‘swarm intelligence’ refers to the more general set of algorithms. Swarm robotics is an approach to the coordination of multirobot systems which consist of large numbers of mostly simple physical robots. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment. Relatively simple individual rules can produce a large set of complex swarm behaviours. A key component is the communication between the members of the group that builds a system of constant feedback. The swarm behaviour involves constant change of individuals in cooperation with others, as well as the behaviour of the whole group. The two other similar fields of study which more or less have the same team structure and almost the same goals are multi-robot exploration and multi-robot coverage.

Unlike distributed robotic systems in general, swarm robotics emphasizes a large number of robots, and promotes scalability, for instance by using only local communication.

1.3 Machine Learning

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can change when exposed to new data.

The process of machine learning is similar to that of data mining. Both systems search through data to look for patterns. However, instead of extracting data for human comprehension – as is the case in data mining applications – machine learning uses that data to detect patterns in data and adjust program actions accordingly. Machine learning algorithms are often categorized as being supervised or unsupervised. Supervised algorithms can apply what has been learned in the past to new data. Unsupervised algorithms can draw inferences from datasets.

1.3.1 Reinforcement Learning

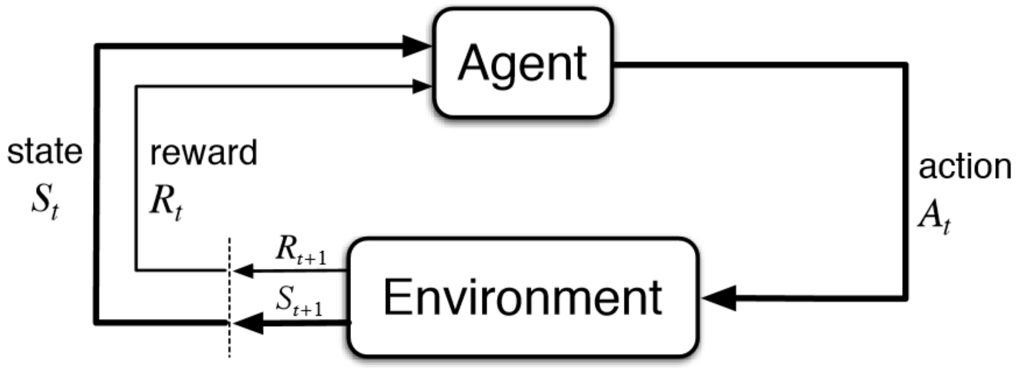


Fig 1.1 Reinforcement Learning

Reinforcement learning (RL) as illustrated by Fig. 1.1, is learning by interacting with an environment. An RL agent learns from the consequences of its actions, rather than from being explicitly taught. It selects its actions on the basis of its past experiences (exploitation) and also by new choices (exploration), which is essentially trial and error learning. The reinforcement signal that the RL-agent receives is a numerical reward, which encodes the success of an action’s outcome, and the agent seeks to learn to select actions that maximize the accumulated reward over time.

Autonomous robots may make feasible exploration of hazardous environments such as the ocean and outer space, using on-line learning to adapt to changing and unforeseen conditions.

1.4 Terrain Mapping

Terrain mapping is a classification system that describes the characteristics and spatial distribution of the terrain.

1.5 Existing Systems

1.5.1 LIDAR

LIDAR (Light Detection and Ranging) is a surveying method that measures distance to a target by illuminating that target with a pulsed laser beam, and measuring the reflected pulses with a sensor. Differences in laser return times and wavelengths can then be used to make digital representations of the target. As opposed to passive sensors that detect energy naturally emitted from an object, lidar uses active sensors, which have their own source of light. Laser beams hit objects and the reflected energy from the objects is detected and measured by sensors.

Distance to the object is determined by the time between transmitted and backscattered pulses. By recording these times and using the speed of light, distance can be calculated.

1.5.2 Radar

Radar is an object-detection system that uses radio waves to determine the range, angle, or velocity of objects. A radar system consists of a transmitter producing electromagnetic waves in the radio or microwaves range, a transmitting antenna, a receiving antenna (often the same antenna is used for transmitting and receiving) and processor to determine properties of the objects. Radio waves (pulsed or continuous) from the transmitter reflect off the object and return to the receiver, giving information about the object’s location and speed.

1.5.3 Stereo photogrammetry from aerial surveys

Photogrammetry is the science of making measurements from photographs, especially for recovering the exact positions of surface points. Such measurements are made on photographs from aerial surveys for generating the map of a given area.

1.5.4 Multi-view stereo applied to aerial photography

Computer vision software can be used which allows general geoscientists to easily create accurate 3D models from field photographs.

1.6 Proposed System

A plethora of options are presently available to generate the map of the terrain. However, there are certain situations which demand robustness and adaptability –

- In case of generating maps of terrains of other planets, aerial imaging might not be possible at all.
- Aerial imaging is also not possible for indoor, underground and underwater environments.
- Lidar scanners are very expensive which makes it risky to deploy them in unknown environments.

The proposed system involves

- Using multiple land robots (can be extended to aerial robots too) to explore the region assigned while using onboard radar and camera sensors to collect spatial data. This can be done in an easily scalable way which allows more robots to be added to the swarm at any time
- Using Simultaneous Localisation and Mapping (SLAM) so the robot will be able to localise itself while performing the mapping without depending solely on filtered GPS data, which is not accurate enough for certain applications and may not be available at all in certain locations.
- Use reinforcement learning to improve robots' techniques to traverse different types of terrains, thereby making the system adaptable.

CHAPTER 2

2. LITERATURE REVIEW

A total of 12 papers were studied of which the 7 most relevant to the project are discussed here. The literature study is organized by the field each paper pertains to.

2.1 Terrain Mapping

Forward looking InSAR based field terrain mapping for unmanned ground vehicle[2], Intelligent Robot Systems (ACIRS), Asia-Pacific Conference on. IEEE, 2016

J Wang, Z Jiang, Q Song, Z Zhou

Unconstructed environments perception is a challenge problem for Unmanned Ground Vehicle (UGV) and is not well solved by traditional sensors, such as: optical camera, lidar and millimeter wave radar. This paper presents a novel forward looking InSAR (Interferometric Synthetic Aperture Radar) sensor for UGVs and focuses on reconstruction the Digital Terrain Model (DTM) of unconstructed environments. DTMs are vital to boost obstacle detection and path planning performances. The forwardlooking InSAR system and its signal processing flow are briefly described. Then practical issues of applying the sensor in field environment are addressed. The impacts of height error and pitch angle error of the InSAR antenna are analyzed and solved by the split spectrum algorithm. Field experiment on a rough meadow has shown advanced perception capability of the forward-looking InSAR in complex environments.

Towards LIDAR-RADAR based terrain mapping[3], Advanced Robotics and its Social Impacts (ARSO), 2015 IEEE International Workshop on. IEEE, 2015

JA Guerrero, M Jaud

This paper addresses the problem of perception for autonomous vehicle navigation in real environments. Integrity safe navigation of autonomous vehicles in unknown environments poses a traversability problem. The authors were interested in the integrity-safe navigation in unknown environments. Safe navigation is a task that depends on the knowledge of the surrounding environment and the vehicle dynamics. Classical navigation approach focus on obstacle avoidance often based on occupancy and elevation maps. They propose to combine an optical sensor and an electromagnetic sensor to build a richer map of the environment which will be used for traversability analysis and path planning. The proposed lidar-radar map encodes the geometry of the environment such that traversability analysis and trajectory planning guarantee the robot’s integrity in a stability sense. A comparative analysis of two mapping algorithms using lidar, radar, IMU and GPS sensors shows the advantages of such bimodal perception system. Results have been validated experimentally.

Straightforward reconstruction of 3D surfaces and topography with a camera: Accuracy and geoscience application[4], Journal of Geophysical Research: Earth Surface

MR James, S Robson

Topographic measurements for detailed studies of processes such as erosion or mass movement are usually acquired by expensive laser scanners or rigorous photogrammetry. Here, they test and use an alternative technique based on freely available computer vision software which allows general geoscientists to easily create accurate 3D models from field photographs taken with a consumer-grade camera. The approach integrates structure-from-motion (SfM) and multiview-stereo (MVS) algorithms and, in contrast to traditional photogrammetry techniques, it requires little expertise and few control measurements, and processing is automated. To assess the precision of the results, they compare SfM-MVS models spanning spatial scales of centimeters (a hand sample) to kilometers (the summit craters of Piton de la Fournaise volcano) with data acquired from laser scanning and formal close-range photogrammetry. The relative precision ratio achieved by SfM-MVS (measurement precision: observation distance) is limited by the straightforward camera calibration model used in the software, but generally exceeds 1:1000 (i.e., centimeter-level precision over measurement distances of 10 s of meters). We apply SfM-MVS at an intermediate scale, to determine erosion rates along a ∼50-m-long coastal cliff. Seven surveys carried out over a year indicate an average retreat rate of 0.70 ± 0.05 m a^{-1}. Sequential erosion maps (at ∼0.05 m grid resolution) highlight the spatiotemporal variability in the retreat, with semivariogram analysis indicating a correlation between volume loss and length scale. Compared with a laser scanner survey of the same site, SfM-MVS produced comparable data and reduced data collection time by ∼80%.

2.2 Swarm Intelligence

Information Sharing in Swarm Intelligence Techniques: A Perspective Application for Natural Terrain Feature Elicitation[5], International Journal of Computer Applications

L Goel, D Gupta, VK Panchal

Particles can interact either directly or indirectly (through the environment). The key to maintain global, self-organized behaviour is social interaction i.e. information sharing between the system's individuals. Hence, information sharing is essential in swarm intelligence. In the paper, the authors highlight how the concept of information sharing in various swarm-based approaches can be utilised as a perspective application towards the elicitation of natural terrain features. The paper provides a mathematical formulation of the concept of information sharing in each of the swarm intelligence techniques of Biogeography based optimization (BBO), Ant Colony Optimization (ACO), Particle Swarm optimization (PSO) and Bee Colony Optimization (BCO) which are the major constituents of the SI techniques that have been used till date for classifying topographical facets over natural terrain.

A Robot System Design for Low-Cost Multi-Robot Manipulation[6], Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on. IEEE, 2014.

J McLurkin, A McMullen, N Robbins

Multi-robot manipulation allows for scalable environmental interaction, which is critical for multi-robot systems to have an impact on our world. A successful manipulation model requires cost-effective robots, robust hardware, and proper system feedback and control. This paper details key sensing and manipulation capabilities of the r-one robot. The r-one robot is an advanced, open source, low-cost platform for multi-robot manipulation and sensing that meets all of these requirements. The r-one has a rich sensor suite, including a flexible IR communication/localization/obstacle detection system, high precision quadrature encoders, gyroscope, accelerometer, integrated bump sensor, and light sensors. This paper presents an overview of the r-one, the r-one manipulator, and basic manipulation experiments to illustrate the efficacy the design.

Kilobot: A Low Cost Scalable Robot System for Collective Behaviors[7], 2012 IEEE International Conference on. IEEE

M Rubenstein, C Ahler, R Nagpal

In current robotics research there is a vast body of work on algorithms and control methods for groups of decentralized cooperating robots, called a swarm or collective. These algorithms are generally meant to control collectives of hundreds or even thousands of robots; however, for reasons of cost, time, or complexity, they are generally validated in simulation only, or on a group of a few tens of robots. To address this issue, this paper presents Kilobot, a low- cost robot designed to make testing collective algorithms on hundreds or thousands of robots accessible to robotics researchers. To enable the possibility of large Kilobot collectives where the number of robots is an order of magnitude larger than the largest that exist today, each robot is made with only $14 worth of parts and takes 5 minutes to assemble. Furthermore, the robot design allows a single user to easily operate a large Kilobot collective, such as programming, powering on, and charging all robots, which would be difficult or impossible to do with many existing robotic systems.

2.3 Machine Learning

Playing Atari with Deep Reinforcement Learning[8], arXiv preprint

Mnih, Volodymyr

The authors introduce a new algorithm called Deep Q Network (DQN for short). It demonstrated how an Artificial Intelligence agent can learn to play games by just observing the screen without any prior information about those games. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. The authors applied their method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. They found that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them. This paper opened the era of what is called ‘deep reinforcement learning’, a mix of deep learning and reinforcement learning.

CHAPTER 3

3. SYSTEM ANALYSIS

3.1 Evolutionary Prototyping

The development model used for the project is called Evolutionary Prototyping.

Evolutionary Prototyping is a lifecycle model in which the system is developed in increments so that it can readily be modified in response to end-user and customer feedback.

It works best for projects where the details of the system are not known and must be decided through the course of development from a vague set of objectives.

The main benefits of this approach are -

- The ability to address risk early in the project
- Early feedback on whether the final system will be acceptance
- Visible progress throughout the project

The main risks of this approach are -

- Unrealistic schedule and budget expectations
- Inefficient use of prototyping
- Unrealistic system performance expectations
- Poor design

We have attempted to minimise these risks by extensively utilising existing literature, previous experiences and expert opinions.

3.2 Problem Definition and Feasibility Study

3.2.1 Problem Definition

Building a dynamic system of robots that can form a swarm to efficiently generate a 3D surface model of a given area, which may pose traversability challenges.

3.2.2 Feasibility Study

Feasibility study is an assessment of the practicality of a proposed project. The feasibility study performed for this project is presented in brief below:

Technical Feasibility

This assessment is based on an outline design of system requirements, to determine whether the team members have the technical expertise to complete the project.

- Swarm algorithms are well explained here[9]
- The team members have previous experience designing sophisticated machine learning systems using frameworks such as Tensorflow.
- The team members have experience using hardware such as various Arduino boards and various sensors and actuators.
- All the programming languages and frameworks that are planned to be used are well-documented and have good community support.

Keeping the above points in mind, the project was considered to be technically feasible.

Economic Feasibility

While we exclusively use Free and Open Source Software which allows us to use and modify code without monetary and legal constraints, the physical robots themselves must be built with hardware components. We have estimated that the cost of these components could add up to approximately Rs. 8,000.

Although the components in general are expensive and the overall cost was higher than expected at first, we, as a team, have decided to bear the full cost of the project to reduce the impact that lack of hardware could have on the feasibility, implementation or functionality of the project. Further, we have tried to select components in a way that allows them to be reused easily for future projects. This made the investment more reasonable.

Operational Feasibility

Robotics itself is very probabilistic in nature. Accurate positioning, sensor noise, etc are all common issues that researchers face while working with robots. However, these issues have been solved before and therefore, can be solved again.

The modules, such as swarm exploration, robotic reinforcement learning, 3D representation generation have all been implemented individually before. The challenge lies in integrating these fairly distant technologies.

3.3 Requirements Analysis

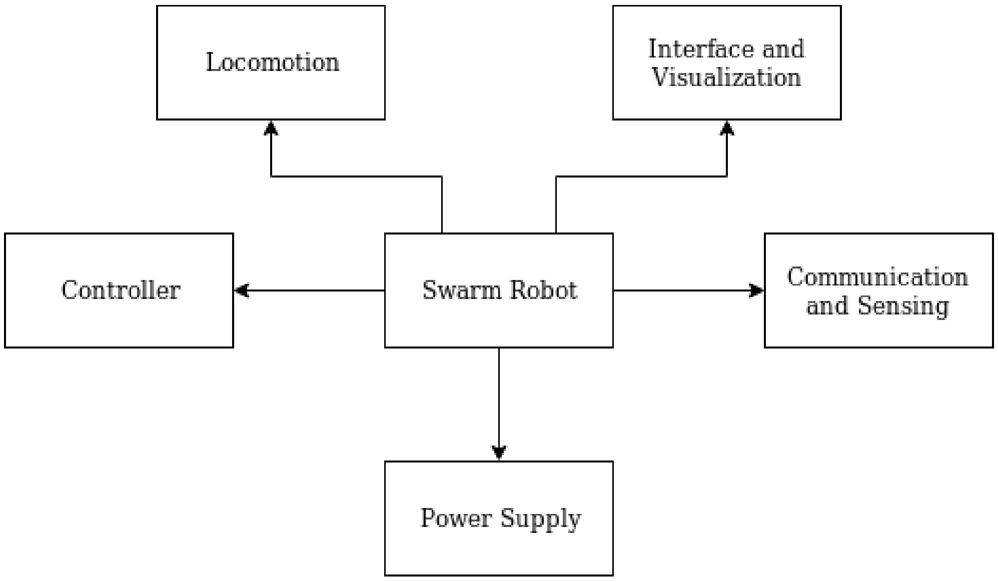


Fig. 3.1 Submodules of the system

Locomotion

One important capability of the robot is to be able to move in its environment. We will be using the most common locomotion strategy for swarm robots, i.e., using a two-wheeled differential drive, where each wheel is powered by an electric gear motor.

Two voltage regulators on a H-Bridge IC can provide power to the motors. Both of these regulators can be switched on and off by the microcontroller.

Communication and Sensing

Each swarm robot must be capable of sensing its immediate neighbours and communicating with them. The most popular implementation is to use an array of infrared LED transmitters and infrared photodiode receivers arranged along the perimeter of the robots' bodies equidistant from each other. Messages are transmitted by pulsing the transmitter according to standard line coding techniques.

During any communication between robots, the receiving robot also measures the intensity of the incoming infrared light. This incoming light intensity is a monotonically decreasing function of the distance between the transmitter and the receiver; therefore the distance to the transmitter can be calculated by the receiver.

Power System

Each swarm robot needs a power source to run on. We will be using a 5V 2.1A USB power bank and a USB cable pinout for each swarm robot. This battery can power the robot for a few days with a single charge.

Controller

The controller for the robot should serve two functions. Firstly, it should interface with all the low-level electronics such as motors, communication components, power circuitry, etc. Secondly, it needs to run a user-defined robot behavior program. The controller used is ATmega328P, which is capable of 20 MIPS (Million Instructions Per Second) has 32 kilobytes of memory, which is sufficient for running the required program.

Web Interface and Visualization

There should be a user-friendly interface to visualize the spatial data obtained.

We will be using a Javascript front-end application running on a browser using data provided by a server program through websockets. This data will be initially sent to the server via a network or through a file transfer from the local storage on the swarm robots.

3.4 Objectives of the Project

3.4.1 Functional Requirements

- The swarm robots should be capable of organising themselves and assigning regions of the terrain among themselves.
- Each robot should be capable of learning to traverse a terrain that it or the other robots have previously not experienced.

- Once a robot has learned to traverse a terrain, it must be capable of storing the knowledge in a centralized location accessible by all the bots.
- A 3D representation of the region should be generated from the above data.

3.4.2 Non-Functional Requirements

- The overall performance of the terrain mapping should be acceptable.
- The system should be scalable to work for larger terrains such that the robots are usable as planetary rovers, etc.
- The swarm robots should be able to recover from failure to traverse a terrain and begin learning to traverse that region again. Also other robots should be least affected by the failure of one robot. This requirement addresses the fault tolerance of the system.
- The system should have minimal maintenance requirements.
- The 3D representation generated by the system should be accurate and reliable.
- The robots should not cause any damage to the surrounding environment.
- The system must be fully dynamic, allowing the addition and removal of robots with minimal effort.

CHAPTER 4

4. SYSTEM DESIGN

The design of the system can be understood through Data Flow Diagrams and its behaviour can be understood through Behavioral State Machine diagrams.

4.1 Data Flow Diagrams

4.1.1 Level 0 Data Flow Diagram

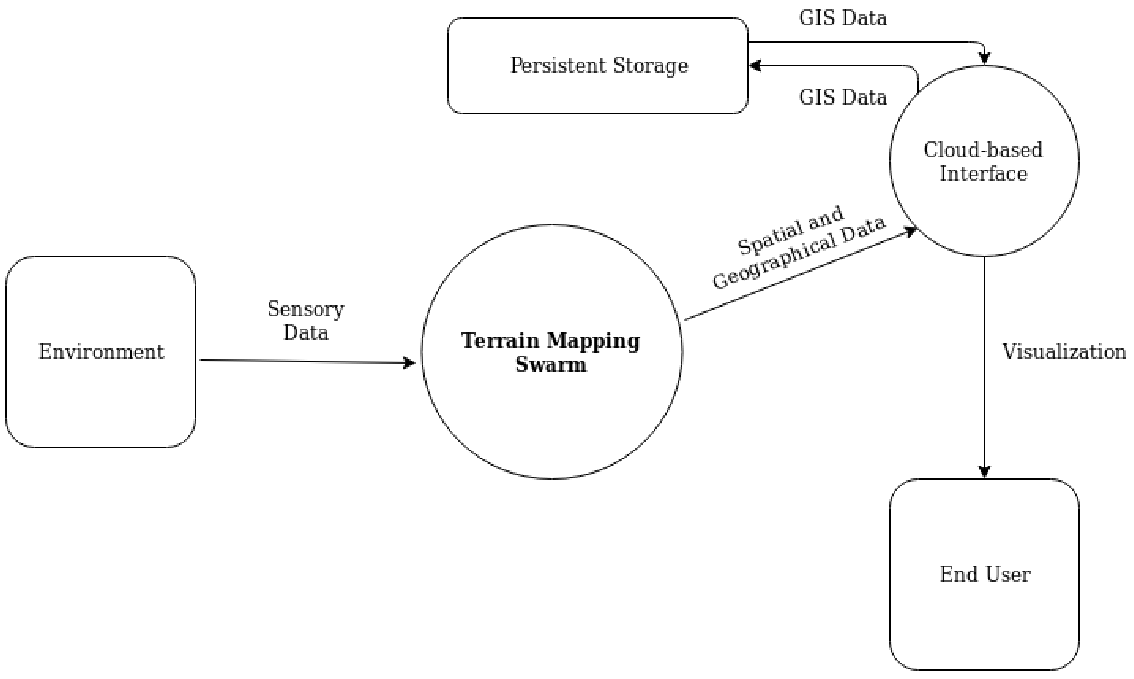


Fig. 4.1 Level 0 Data Flow Diagram

Fig. 4.1 shows the high level overview of the entire system. There exists a swarm of robots that are deployed in an environment. The robots have sensors to sense various parameters of the environment. From the data received through various sensors, the terrain mapping swarm system processes spatial and geographical data. This data is sent over a network to a program running on an external processing system that stores this data in a database and retrieves it when a user wants to visualize the data.

4.1.2 Level 1 Data Flow Diagram

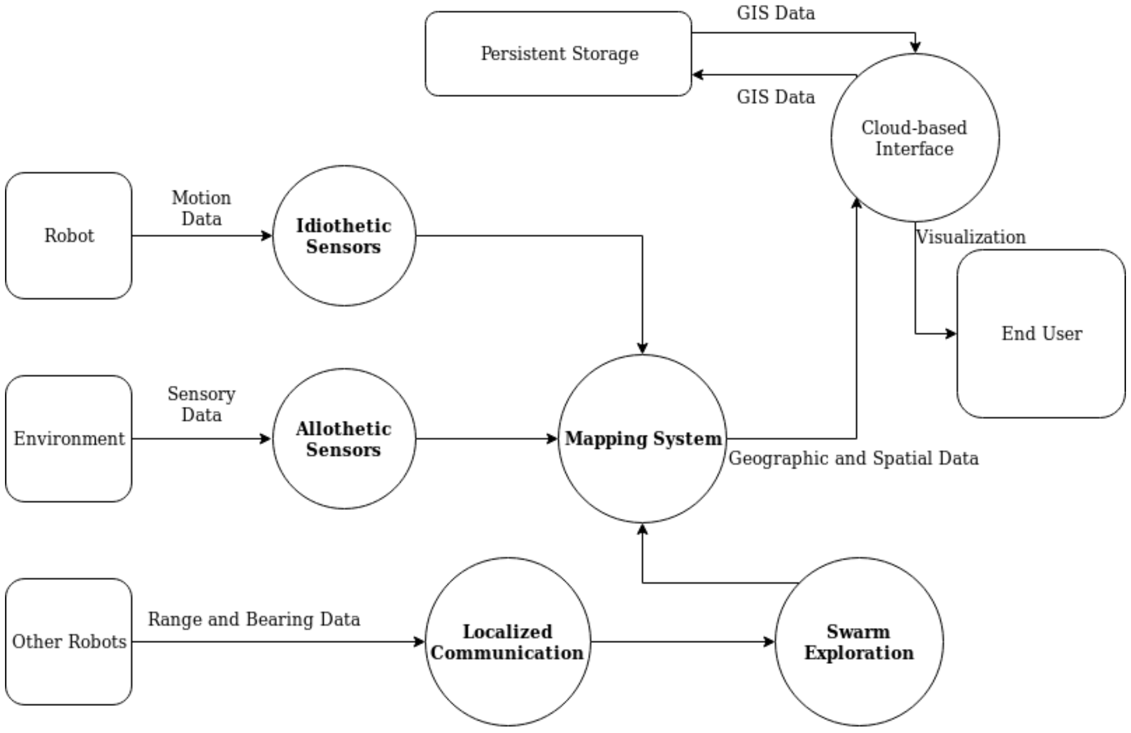


Fig. 4.2 Level 1 Data Flow Diagram

Fig. 4.2 shows the Level 1 Data Flow Diagram of the system.

The robot has two sources of information: idiothetic sources and allothetic sources.

When in motion, a robot can use dead reckoning methods such as tracking the number of revolutions of its wheels. This corresponds to the idiothetic source and can give the absolute position of the robot, but it is subject to cumulative error which can grow quickly.

The allothetic source corresponds to the sensors of the robot, cameras, microphones, lidar, sonar, etc. However, such sources are subject to a problem called perceptual aliasing. This means that two different places can be perceived as the same. For example, in a building, it is nearly impossible to determine a location solely with the visual information, because all the corridors may look the same.

Each robot is also able to sense other robots in its line of sight through their range and bearing sensors. This allows the swarm to communicate among itself in a localized manner and perform simple distributed processes such as exploring the region optimally.

The data from the idiothetic and the allothetic sources and the current relative position of the robots of the swarm are used in the mapping process.

4.1.3 Level 2 Data Flow Diagram

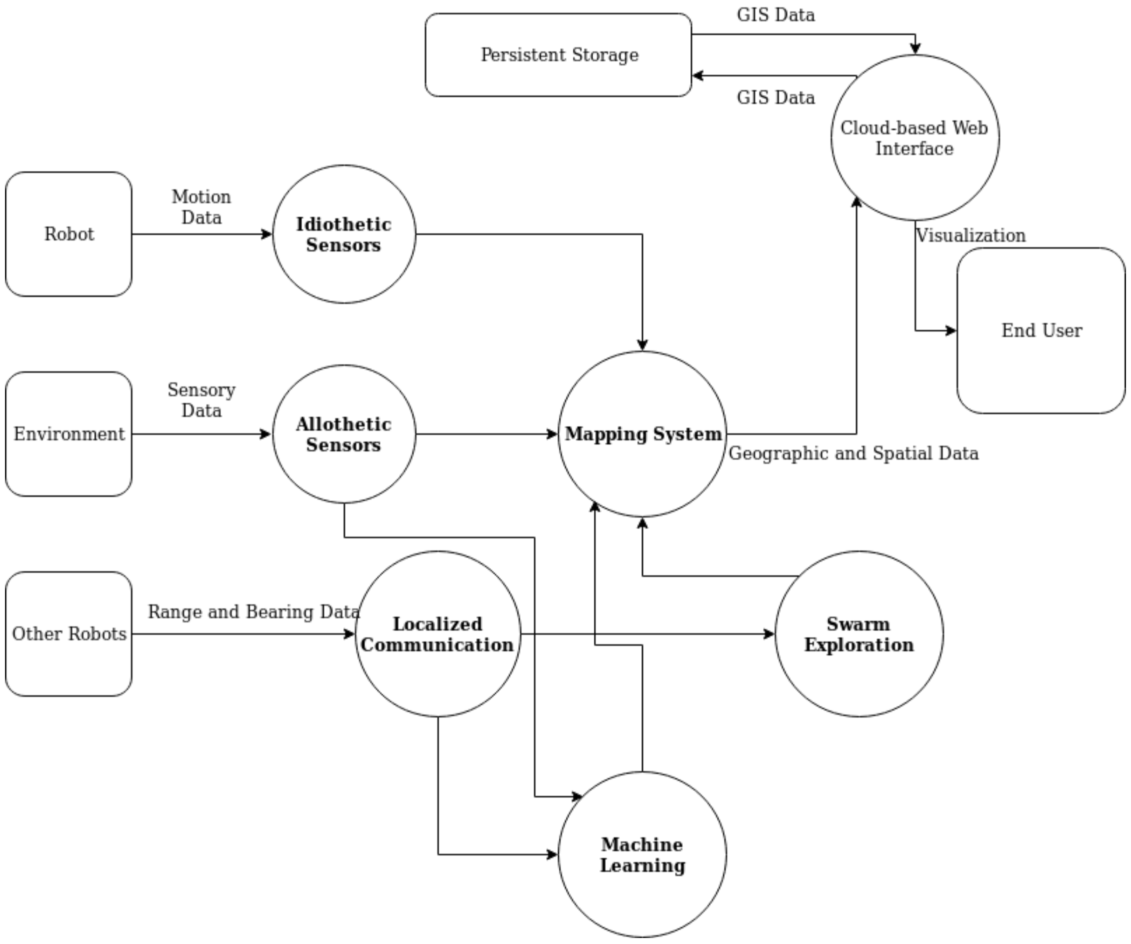


Fig. 4.3 Level 2 Data Flow Diagram

The data sensed from the environment and knowledge about neighbouring robots and their bearing are fed as input to machine learning algorithms that improve the performance of the mapping system.

4.2 Behavioural State Machine Diagram

Behavioral State Machine is specialization of behavior and is used to specify discrete behavior of a part of the designed system through finite state transitions.

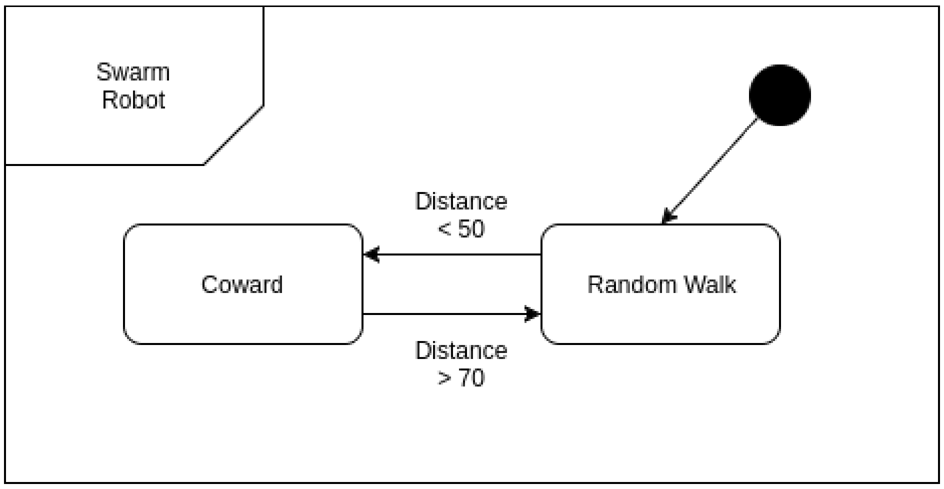


Fig 4.4 Behavioural State Machine Diagram

The swarm robots exhibit behaviour that is based on rules of attraction and repulsion. In our system, there are two major states:

- **Random Walk State**: The swarm robot performs a random walk in the region.
- **Coward State**: The swarm robot is repulsed by the source, strongly when close to it and less to not at all when far from it.

From the Behavioral State Machine diagram, the following behaviour of the swarm can be deduced – the swarm robots move randomly as long as there is a certain inter-robot distance. This allows optimal swarm exploration of the region.

CHAPTER 5

5. IMPLEMENTATION

This section describes the implementation of the project. Most parts of the project were first simulated using standard simulators used widely in research and then ported to a hardware implementation. Adjustments were made with regard to non-availability of sensors by using alternatives. For example, in the simulation, a camera and blinking LEDs were used to identify and communicate locally with other robots, but the hardware implementation uses IR photodiodes and IR LEDs for the same purpose.

5.1 Swarm Exploration

5.1.1 Simulation on ARGoS

In order to simulate both the robots and their environment, we have used ARGoS, a multi-physics robot simulator. It can efficiently simulate large scale swarms of robots. Its accuracy aims to be as close as possible to real life. It is a tool created with research in mind and used all around the world in laboratories. Its constraints and quality make it suitable for use in professional and research environments.

ARGoS takes three things as entry points:

- the robot brain (either C++ code or Lua code)
- an environment code (in C++), either to update the environment's information (like adding objects) or to add more visuals
- a .argos (XML) file describing the experimental conditions.

Controlling Movement

Each robot has two wheels, each of which can be set programmatically to move with a certain speed. Positive values make the wheel move forward while negative values make the wheel move backwards. This arrangement is used as a differential drive allowing a robot to go forward and reverse; and to turn left or right as per the degree of rotation required.

Sensing Obstacles

The robots are equipped with proximity sensors that react to physical object on a 10cm range. There are 24 of them, spread in a ring around the robot body. Each reading is a table composed of an angle in radians and a value between 0 and 1. The angle defines the position of the sensors around the robot and the value is as high as the object is close (0 means nothing is detected). They are all contained in the robot.

Communication

Each robot is equipped with 12 Light Emitting Diodes spread in circle around its body and a last one on top of it (called the beacon). A colour can be set to each of these LEDs.

Sensing Other Robots

There is a semi-spherical mirror on top of the tower on each robot, and under the mirror is a camera facing the front-side of the mirror. The result is an omnidirectional camera and the image is a 360° picture of the surrounding of the robots. To sense other robots, there is an algorithm that will output a list of small light detected (LEDs in the arena, such as the ones on the robot). The resultant blob is defined by its color and position.

Programming the Behaviour of the swarm

To implement coupling and weighting behaviors, an entity called *Force* is used to drive the wheels with a speed and direction that has some direct correspondence with the proximity sensor table. The sensed data from each proximity sensor is transformed into a force. All these forces are summed up and the resultant force vector is returned which influences the speed and direction in which the robot moves.

The behaviour of the swarm robots is based on rules of attraction and repulsion.

There are four behaviors which act as personalities for the robots –

- **Lover** is attracted by the source, strongly when far from it, and weekly as it gets closer to it. Force = + distance
- **Aggressor** is attracted too, but goes faster as he gets closer to its prey. Force = + 1 / distance
- **Coward** is repulsed by the source, strongly when close to it and less to not at all

when far from it. Force = - 1 / distance

- **Explorer** is repulsed by the source too, and moves faster when it's far from it. Force = - distance

Each of these behaviours is programmed as a Lua function. As previously described, a random walk behaviour is exhibited by a robot when no other robot is sensed in its proximity. When a robot is in such a random walk state, it moves with a random speed in a random direction. When a robot is detected in its proximity, it moves with a force determined by the behaviour chosen.

Swarm exploration involves each of the robot in the swarm moving away from the other robots. Therefore, to implement swarm exploration from the above set of behaviours, a robot is to be repulsed by the source. Therefore, one of the Coward or Explorer behaviours is to be chosen. In our implementation, the Coward behaviour is chosen.

5.1.2 Implementation on hardware

The behaviour of the swarm was ported from the simulation in ARGoS (written in Lua) to run on the Arduino (C++). Three robots were built, two of which were 4 wheel drives and one was a 2 wheel drive. The design of the robots was influenced by Rice University's R-one[6] and Harvard University's Kilobot[7].

Controlling Movement

Two variants of differential drive - the 4-wheel differential drive and the 2-wheel differential drive were used. In both cases, the same voltage was given to wheels on both sides of the robot body to make the robot move straight, voltages with the same magnitude but opposite polarity was given to motors on different sides of the robot body to make it rotate by a certain degree, while a combination of voltage magnitudes and polarity allows the robots to move by a certain angle while still making translating motion.

Sensing Obstacles and Other Robots

Each robot has 4 HC-SR04 ultrasonic sensors arranged around the body. The sensor has an transducer that emits ultrasonic sound waves. The ultrasonic sound waves has an extremely high pitch that humans cannot hear and is also free from external noises from passive or active sources. The echo is received by the receiver module on the sensor. The time taken between sending the pulse and receiving the echo is measured. The distance to the obstacle is then estimated from the time taken by the sound wave to travel the distance to the obstacle and back.

Inter-robot Communication

Since the project does not require communication between the robots and mere sensing the presence of nearby robots and estimating their range and bearing is sufficient, local sensing of robots is implemented using the same HC-SR04 sensors.

Programming the Behaviour of the Swarm

As in the simulation, the *Force* is calculated (here, from the ultrasonic proximity measurements). The behaviours and the state aforementioned are implemented on the Arduino in C++.

5.2 Machine Learning

Due to the complexities involved in implementing Machine Learning algorithms at the hardware level, such as the lack of robustness of locomotion and the unavailability of different types of terrains, Machine Learning was conceptually implemented as a simulation on OpenAI Gym.

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the structure of the agent, and is compatible with any numerical computation library, such as TensorFlow or Theano.

5.2.1 Simulation using OpenAI Gym and Tensorflow

OpenAI Gym consists of a growing suite of environments (from simulated robots to Atari games). The *MountainCar-v0* environment is used in our simulation due to its similarity with our project (albeit in a single dimension).

The *MountainCar-v0* Environment

The problem definition is as follows:

A car is on a one-dimensional track, positioned between two “mountains.” The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.

Every environment has an action space and an observation space. The action space is a set of actions that the agent can perform. In the case of the *MountainCar* environment, it can perform 3 actions - push left, not push at all and push right.

The observation space is a list of some observations of the environment. In the *MountainCar* environment, the observation space is 2-dimensional *Box* Object. That is, the observation is a point in the X-Y plane.

Implementation

The solution was implemented using a Reinforcement Learning algorithm called Q-Learning. In its simplest implementation, Q-Learning is a table of values for every state (row) and action (column) possible in the environment. Within each cell of the table, the agent learns a value for how good it is to take a given action within a given state. In the case of the *MountainCar* environment, we have many possible states (one for each point the car can be in), and 3 possible actions as mentioned before, giving us a table of Q-values with 3 columns and a specific number of rows (equal to the total number of X-Y points that the car can be in). Initially the table is uniform (all zeros), and then as the agent observes the rewards it obtains for various actions, the table is updated accordingly.

Updates are made to the Q-table using the Bellman equation, which states that the expected long-term reward for a given action is equal to the immediate reward from the current action combined with the expected reward from the best future action taken at the following state. In this way, the agent reuses its Q-table when estimating how to update the table for future actions. In equation form, the rule can be expressed as:

$$Q(s,a)=r+\gamma * max(Q(s',a')) \qquad ...(5.1)$$

Equation 5.1 is the Bellman equation. It says that the Q-value for a given state (s) and action (a) should represent the current reward (r) plus the maximum discounted (γ) future reward expected according to the agents own table for the next state (s') it would end up in. The discount variable allows deciding how important the possible future rewards are compared to the present reward. By updating in this way, the table slowly begins to obtain accurate measures of the expected future reward for a given action in a given state.

Building a Function Approximator

Though the table approach is sufficient for the environment considered, it will not scale to larger environments. For example, if the mountain was larger, the number of states will be more and the length of the table will increase too. The number of states and actions will increase even when the implementation is translated from the simulation to the real world. Therefore, running the same algorithm in the real world will require a generalization mechanism as the number of states can be potentially infinite.

Therefore, there needs to be some way to take a description of the state, and produce Q-values for actions without a table: that is where machine learning techniques such as regression come in. By acting as a function approximator, it can take any number of possible states that can be represented as a vector and learn to map them to Q-values.

In our implementation, the python library *scikit-learn* is used. The implementation performs regression using Stochastic Gradient Decent (SGD).

Regression is a statistical measure that attempts to determine the strength of the relationship between one dependent variable and a series of other changing variables (known as independent variables). Non-linear regression is a form of regression in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables.

For the mountain car task, the action-value function itself is non-linear. (If you increase the value for going right and reaching the top, going left will have the lowest values). Therefore linear approximation doesn’t produce a good action-value function and the car never reaches the top with a linear value function.

In our implementation, an approximate kernel map of the features is generated by applying non-linear transformations of the input. This kernel map can serve as a basis for linear classification or in our case, the SGD Regression algorithm. The advantage of using approximate explicit feature maps compared to the kernel trick, which makes use of feature maps implicitly, is that explicit mappings can be better suited for online learning and can significantly reduce the cost of learning with very large datasets.

The *scikit-learn* class *RBFSampler* is used in our implementation to create a RBFSampler object that constructs an approximate mapping for the radial basis function kernel. The RBF kernel on two samples x and x', represented as feature vectors in some input space, is defined as:

$$k(x, x') = \exp\left(\frac{-\|x - x'\|^2}{2\sigma^2}\right) \quad \text{...(5.2)}$$

Equation 5.2 is the RBF Kernel Equation.

- $-\|x - x'\|^2$ is the squared Euclidean distance between the two feature vectors.
- σ is a free parameter.

Since the value of the RBF kernel decreases with distance and ranges between zero (in the limit) and one (when x = x'), it has a ready interpretation as a similarity measure. The feature space of the kernel has an infinite number of dimensions.

The *FeatureUnion* class from *scikit-learn* concatenates results of multiple transformer objects, *RBFSampler* objects with different variances in our implementation.

This estimator applies a list of transformer objects in parallel to the input data, then concatenates the results. This is useful to combine several feature extraction mechanisms into a single transformer.

The best action is chosen by applying regression using SGD on the transformed features for each action. The action is chosen greedily based on the best predicted reward by the SGD Regressor.

5.2.2 Corresponding implementation on hardware

Implementing a similar 1-dimensional Reinforcement Learning agent on our robots is not very hard. However, such a learning system is of little use in the real world, where the robot must be able to move in 3D space. Hence it can be in many more states than in the simulation. Due to the complexities arising from high dimensionality of state vectors (both input and output) and the limiting nature of the hardware itself (such as the lack of robustness), hardware implementation of the conceptional Machine Learning techniques described in the previous section is part of the enhancements to be performed in the future.

It may however be noted that, though the observation space will not be the same while porting this algorithm to an hardware implementation, the general structure of the agent will be similar. For example, the observation space in all likeliness is a stream of picture frames from the camera mounted on the robot. The agent described previously is flexible enough to adapt to any kind of observations reflecting the state. It is actually agnostic to what the state observations even mean and is just concerned with finding patterns between the observations, actions and rewards.

The rewards in the hardware implementation are determined by sensors such as the accelerometer detecting sudden bumps or slipping while attempting to move etc.

5.3 Terrain Mapping

We use ultrasonic ranging systems and the visual sensor (as part of the visual SLAM) to map the terrain.

5.3.1 Ultrasonic Radar

An ultrasonic ranging system is mounted on the vehicle to provide the vehicle the ranges of the wall boundaries, as well as detecting the presence of obstacles in the vehicle’s path during navigation.

This is done by mounting a HC-SR04 Ultrasonic Sensor on a SG90 micro servo motor that can rotate the ultrasonic sensor to sweep an area of 180°. During exploration, a robot randomly stops to perform an ultrasonic scan of the immediate surroundings. Starting at 0° through 180°, the ultrasonic distances to the wall boundaries every 2° is recorded. The points on the X-Y plane are obtained by transforming this angle and distance data to cartesian coordinates using standard formulae.

The result is a 2-Dimensional map of the immediate terrain. Mapping in 3-dimensions using a pan and tilt arrangement with 2 servo motors was attempted but the process took a long time as the horizontal servo motor had to sweep 180° for every angle of the vertical servo motor from 0° through 80°.

3-Dimensional mapping was performed as mentioned later, using ORB-SLAM. This ultrasonic ranging system is only a fallback system when ORB-SLAM cannot perform as expected, for example, when the lighting is too low, or there are less number of features that can be detected.

5.3.2 Visualization

In order to visualize terrain data obtained by the swarm, a 3D terrain viewer was implemented. The viewer is an interactive application that the user can use to explore collected data at will using intuitive controls. The viewer is capable of showing real time as well as stored datasets.

It is built using plain Javascript. The spatial data (in the form of spherical coordinates) is converted to cartesian coordinates and is received by the Javascript front-end application via websockets from the kemal application serving it. This data point is placed in space within the rendered HTML element.

5.4 Simultaneous Localization and Mapping

Localization refers to determining a robot’s position and orientation relative to a chosen landmark. Mapping is the construction of a map of an unknown environment. Each of these is a major computational problem in robotics, requiring various kinds of input data and processing.

A mapping robot that is placed in an unknown environment has to solve both of these problems to complete its objective. It has to be able to map its surrounding, but to successfully map an environment, the robots own location must be known. Conversely, to identity a robot’s location, some form of a map is required. Thus each problem requires the other to be solved.

To resolve this inter-dependence and make the objective of mapping an unknown environment possible, various techniques have been developed. These techniques are all called Simultaneous Localization And Mapping (SLAM) techniques. Most of these techniques are only approximate, and do not guarantee perfect accuracy. They aim to achieve operational compliance using available resources.

SLAM techniques and algorithms vary in their use of sensors, signal processing techniques, etc. One technique of particular interest is known as Visual SLAM.

5.4.1 Visual SLAM

Visual SLAM is a SLAM technique that uses visual data through various types of cameras to perform localization and mapping. A computer vision system is used to identify and track various points in the visual field. The movements of these points can then be used to perform odometry calculations on the camera, and thereby localize the robot possessing the camera. Since this technique already produces visual imagery of the environments, maps produced by visual SLAM can easily be augmented with imagery.

Visual SLAM techniques vary in their method of identifying features, tracking said features, and in the input sensors they use - grayscale cameras, color cameras, monocular camera, stereo cameras, fisheye cameras, cameras with depth perception, etc.

ORB-SLAM

ORB-SLAM[10] is an implementation of visual slam by Mur-Artal, Raul, Jose Maria Martinez Montiel, and Juan D. Tardos. It is a robust system capable of tracking, mapping, relocalization and loop closing, all without the use of specialised hardware like Graphical Processing Units (GPU) which are common in massive computational tasks.

In addition to being a robust and complete system, ORB-SLAM is also extremely efficient, capable of real-time SLAM on common hardware. It is able to map various environments - small and big, outdoors and indoors, all using a single camera.

ORB-SLAM’s use of Oriented FAST and rotated BRIEF (ORB) features gives it the efficiency and robustness necessary to track features through complex movements and orientation changes. This system is superior in performance and reliability to older ones such as SIFT and SURF.

The ORB-SLAM process consists three parallel tasks running continuously -

1. **Tracking** - localizing the camera in every frame, deciding when to insert a new key frame, etc.

2. **Local mapping** - processing new keyframes to perform local bundle adjustment, removing unneeded keyframes, etc.

3. **Loop closing** - detection of loops and correction of computed trajectory to account

for drifts

By efficient ORB feature identification and coordination among the above tasks, ORB-SLAM is capable of faster and more accurate SLAM than other visual SLAM implementations.

Our ORB-SLAM setup

We have chosen to use ORB-SLAM2, an improved version of ORB-SLAM by the original authors (Mur-Artal, Raul, Jose Maria Martinez Montiel, and Juan D. Tardos). This version includes additional features, namely support for stereo and RGB-D cameras, and is also decoupled from ROS, which allows greater flexibility in how ORB-SLAM2 is integrated with the rest of the system.

For cameras, we have used Android-based smartphones. Each robot has a smartphone mounted on its chassis, in a suitable location and orientation to capture imagery, mainly focusing on the area directly in front of it.

The smartphones run a web application in a suitable web browser (Firefox or Chromium) that is used to capture a video stream and send it to a central processing server. We use the Media Capture and Streams web APIs to capture the camera feed and WebRTC to efficiently transport the stream to the processing unit.

This setup makes our implementation largely platform-independent. Any modern smartphone with a modern web browser can be used. It is even possible to switch smartphones at any time, as long as the robot is recalibrated after such changes.

To calibrate the camera we used a chessboard pattern printed on a sheet of white paper. Using the smartphone, we captured a set of pictures (10-15 in number) of this white sheet at slightly different angles and distances. We then used an OpenCV program to find the pattern and once found, calculate the various camera parameters, including normalized distortion factors. This is done by comparing the captured image to a normalized form.

The processing unit captures the incoming WebRTC stream using Kurento, performs some basic normalization using OpenCV and sends the stream, frame by frame, to an ORB-SLAM2-based processing application which performs SLAM

CHAPTER 6

6. TESTING

6.1 Test Driven Development

Test Driven Development is a testing technique in which the programmer writes failing tests first, and then writes the application code to get the tests to pass. The software part of the project was developed using TDD describing the functionality required (by writing tests) before implementing that piece of functionality.

After the test passed, i.e. the functionality was implemented, the code was refactored in some places for better readability and/or efficiency.

6.2 Unit Testing

Specific units of the project were developed using TDD. Table 6.1 gives an overview of the various units tested, how the units were tested and the results of the test.

Table 6.1 Unit Tests Performed

Sl. No.	**Unit under consideration**	**Test Written**	**Result (Pass/Fail)**
1	Ultrasonic Sensor HC-SR04 Accuracy	UltrasonicAccuracyTest [see Appendix]	Fail
2	HC-SR04 Accuracy (with calibration of +2 cm)	UltrasonicAccuracyTest [see Appendix]	Pass
3	Ultrasonic Readings to Cartesian Coordinates (2D)	UltrasonicMappingTest2D [see Appendix]	Pass
4	Ultrasonic Readings to Cartesian Coordinates (3D)	UltrasonicMappingTest3D [see Appendix]	Pass
5	RL Agent Cumulative Rewards	RLAgentRewardTest [see Appendix]	Pass

6.3 Integration Testing

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. Integration tests provide a great way to perform end-to-end tests that validate the application is performing as expected.

The following manual integration tests were performed:

Table 6.2 Integration Tests Performed

Sl. No.	**Modules under consideration**	**Manual Test Performed**	**Result (Pass/Fail)**
1	HC-SR04 and Differential Drive System	Obstacle Avoidance	Pass (After turning time was increased to 1000ms)
2	ARGoS Swarm Exploration	Multi-Robot Interaction	Pass (Robots avoid each other)
3	Swarm Exploration (Hardware Implementation)	Multi-Robot Interaction	Pass (Robots avoid each other, i.e repel when brought close to each other)
4	Mapping using HC-SR04 on a pan-and-tilt servo arrangement and visualization	Acceptance Testing of the Frontend	Pass (The output is as expected)

6.4 System Testing

Integration testing is a testing in which individual software modules are combined and tested as a group while System testing is a testing conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements.

The following system tests were performed:

Table 6.3 System Tests Performed

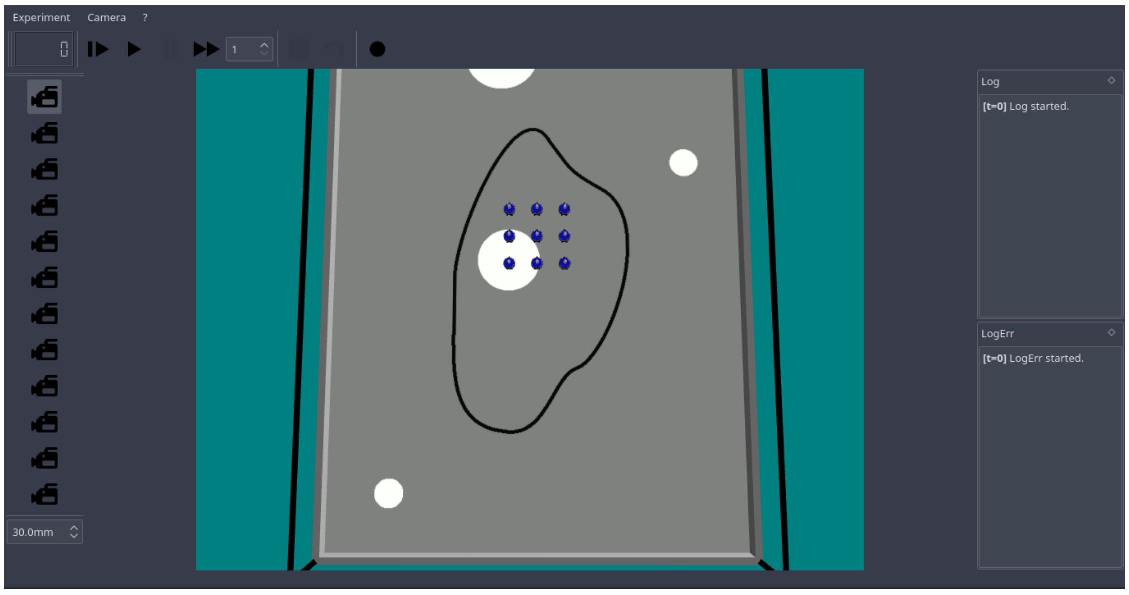
Sl. No.	**Test Performed**	**Observation**	**Result**
1	Map Generation using the multi-robot swarm	A separate map is generated from each robot's feed. Because of synchronisation issues, the feeds are not interpolated and this is marked as future enhancement.	Pass
2	Machine Learning - The robot learns from its environment	A small-scale version (with limited number of environment states and actions) of this was successfully implemented.	Pass
3	Machine Learning - Robots store their learnings for other robots	In Simulation, the pickle package from the Python standard library was used to dump and load learning models. When a pre-trained model was loaded, instead of starting the learning algorithm from scratch, the rewards for the first few episodes was the same as the rewards for the last few episodes when the model was being built.	Pass

CHAPTER 7

7. RESULTS AND DISCUSSION

The following screenshots illustrate the working of the various modules of the system.

7.1 Swarm Exploration on ARGoS

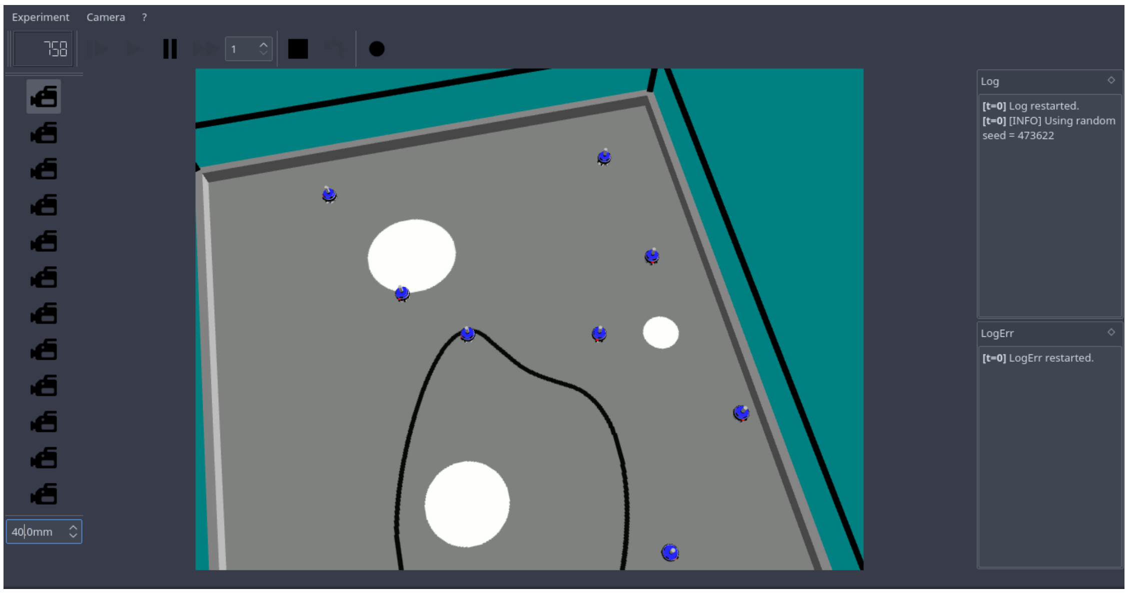


[t=0] Log started.

[t=0] LogErr started.

Fig 7.1 The initial position of the swarm on ARGoS

Fig 7.1 shows the initial positions of the robots before the start of the simulation. The robots are arranged close to each other facing the forward direction.



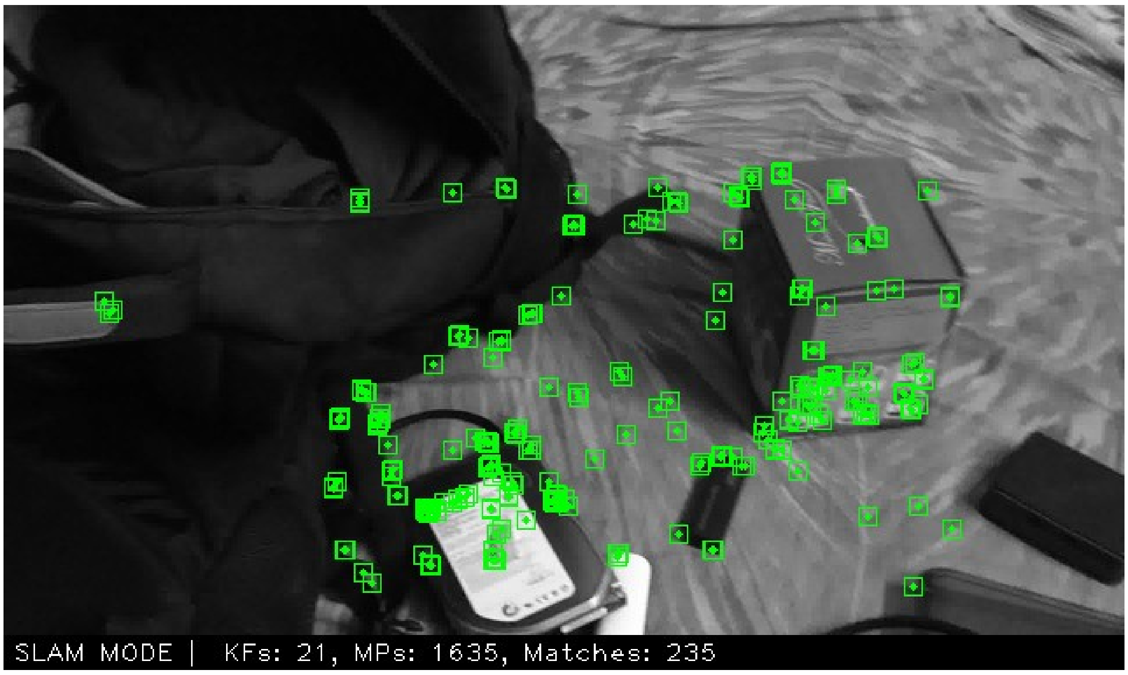
```
[t=0] Log restarted.
[t=0] [INFO] Using random
seed = 473622
```

```
[t=0] LogErr restarted.
```

Fig 7.2 The position of the swarm on ARGoS at the 758th time step

Fig 7.2 shows the positions of the robots some time into the simulation. It can be seen that the robots have dispersed and are repulsed by each other, thereby covering a lot of area in the arena.

7.2 ORB-SLAM



```
SLAM MODE |  KFs: 21, MPs: 1635, Matches: 235
```

Fig 7.3 The ORB-SLAM algorithm working on tracking features on the current frame

Fig 7.3 shows the ORB-SLAM algorithm in operation. The points have already been initialized and they are being tracked frame-by-frame.

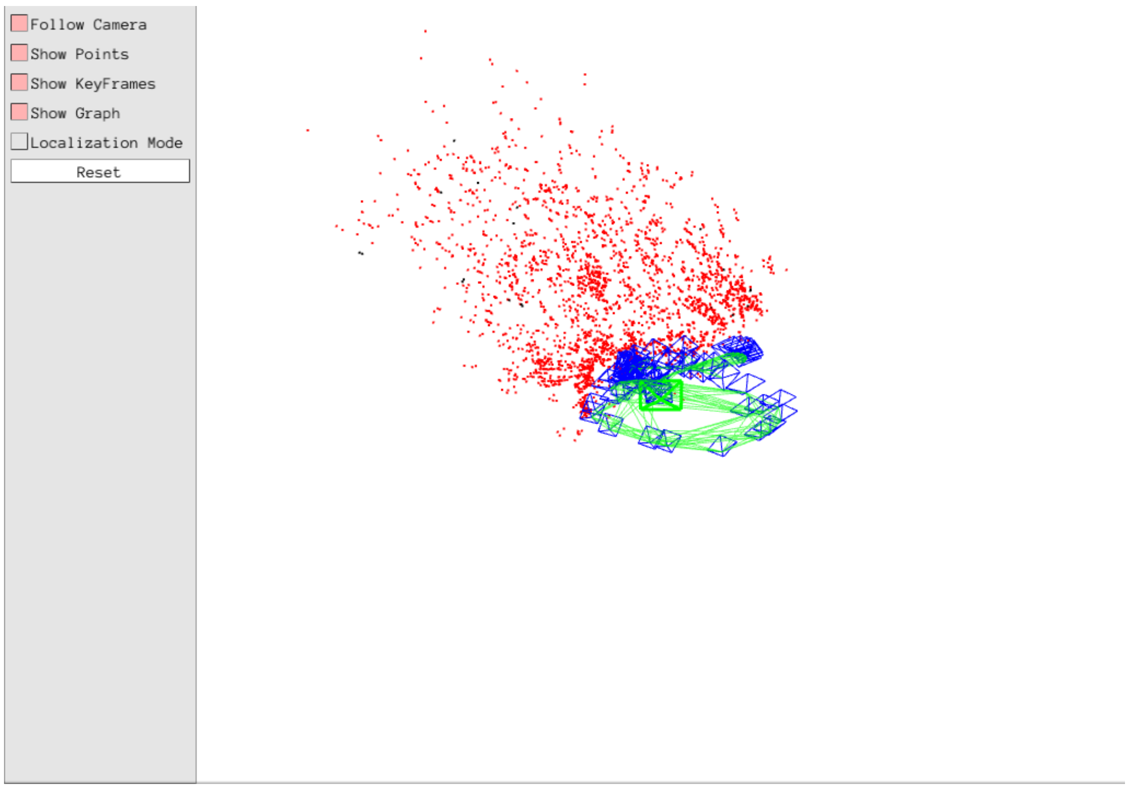


Fig 7.4 The generated map

Fig 7.4 shows the real time map generated by the ORB-SLAM algorithm. It also shows the trajectory and estimate of the current position of the camera and therefore the robot.

7.3 Machine Learning

The following screenshots are that of the mountain-car at various episodes.

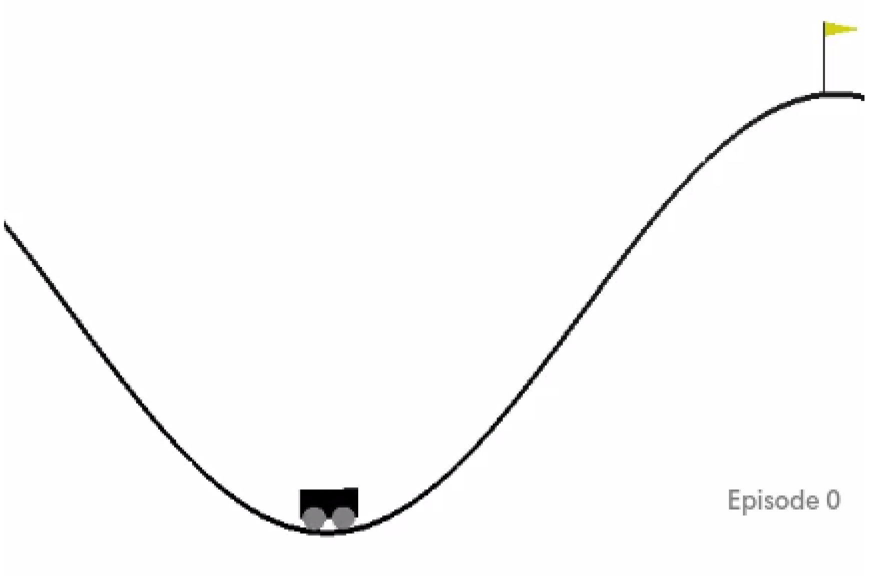


Fig 7.5 Mountain Car - Episode 0

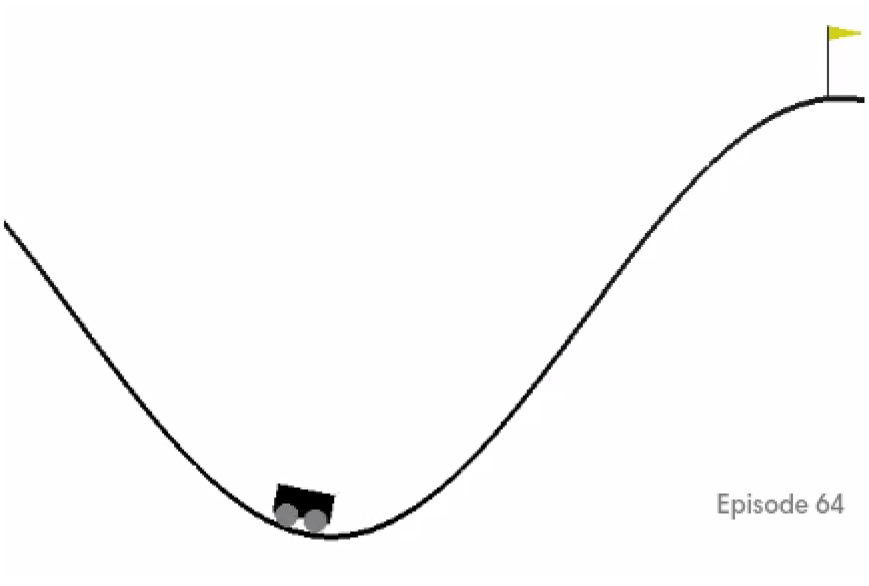


Fig 7.6 Mountain Car - Episode 64

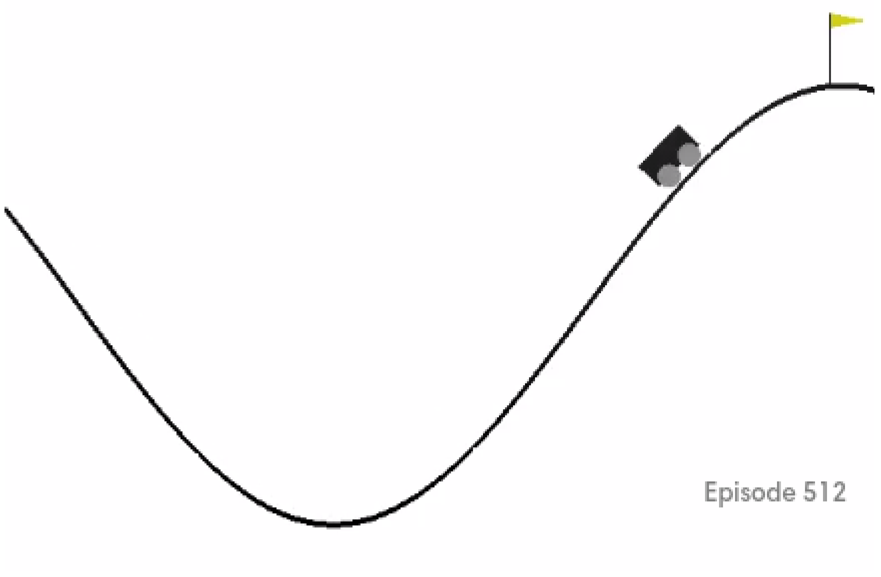


Fig 7.7 Mountain Car - Episode 512

In episodes 0 and 64, the agent does not achieve the goal. But there is more build up in momentum in episode 64 than in episode 0. The agent solves the problem in episode 512.

CONCLUSION AND FUTURE ENHANCEMENTS

By completing the project we intend to, put forward a methodology to solve traversability problems in uncontrolled/unknown environments. Using a multi- robot system as a swarm exploration unit, with each unit also maintaining state information is useful in wide variety of applications such as surveillance, agriculture, search and rescue etc. We could see the advantages of robots that learn from their environment and each other and identified key challenges within the project. The visualization software was performant, user friendly and interactive.

Future Enhancements include the following:

- Synchronizing the data feed from different robots and fusing them to generate one terrain map of the entire region.
- The feature to mark a region and command the swarm to map/explore that region only.
- Using industry-grade sensors such as a LIDAR scanner instead of the pan and tilt arrangement of the ultrasonic sensor. This will result in more accurate and faster readings. Such a system is used presently in driverless cars and will be suitable when the mapping must be done in close to real time.
- The reinforcement learning algorithm could be implemented on the hardware. This could open new means to traverse unknown environments.
- Using other approaches to implement the reinforcement learning agent, such as Deep Q-Learning, as described in one of the papers surveyed or Policy Gradients.

BIBLIOGRAPHY

1. Reynolds, Craig (1987). “Flocks, herds and schools: A distributed behavioral model.”. SIGGRAPH ‘87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques. Association for Computing Machinery: 25–34. doi:10.1145/37401.37406. ISBN 0-89791-227-6.

2. Wang, Jian, et al. “Forward looking InSAR based field terrain mapping for unmanned ground vehicle.” Intelligent Robot Systems (ACIRS), Asia-Pacific Conference on. IEEE, 2016.

3. Guerrero, Jose Alfredo, et al. “Towards LIDAR-RADAR based terrain mapping.” Advanced Robotics and its Social Impacts (ARSO), 2015 IEEE International Workshop on. IEEE, 2015.

4. James, M. R., and Stuart Robson. “Straightforward reconstruction of 3D surfaces and topography with a camera: Accuracy and geoscience application.” Journal of Geophysical Research: Earth Surface 117.F3 (2012).

5. Goel, Lavika, Daya Gupta, and V. K. Panchal. “Information sharing in swarm intelligence techniques: A perspective application for natural terrain feature elicitation.” International Journal of Computer Applications 32.2 (2011): 34-40.

6. McLurkin, James, et al. “A robot system design for low-cost multi-robot manipulation.” Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on. IEEE, 2014.

7. Rubenstein, Michael, Christian Ahler, and Radhika Nagpal. “Kilobot: A low cost scalable robot system for collective behaviors.” Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012.

8. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

9. Lira, Rodrigo. “Introduction | Swarm Robotics”. Rodrigoclira.gitbooks.io. N.p., 2017. Web. 25 May 2017.

10. Mur-Artal, Raul, Jose Maria Martinez Montiel, and Juan D. Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." *IEEE Transactions on Robotics* 31.5 (2015): 1147-1163.

APPENDIX: TESTS

All the tests were written using the Ruby Minitest testing framework. Each of them is given a file consisting of values from observations (if hardware related) or the actual values (if mathematically determinable) and the readings obtained by the unit under consideration.

Ultrasonic Accuracy Test

The test was developed to measure the accuracy of the ultrasonic sensors and find the ideal calibration length to add/subtract. The custom assertion *assert_approx_equal_arrays* is used to assert if each element of the two arrays are within certain delta of each other.

```
class UltrasonicAccuracyTest < Minitest::Test
  def test_the_ultrasonic_sensor_is_accurate_enough
    readings = CSV.parse(File.read("us_readings.csv"))
    readings = readings.transpose
    assert_approx_equal_arrays(readings[0], readings[1])
  end
end
```

HC-SR04 Accuracy (with calibration of +2 cm)

Each reading is incremented by 2 cm to calibrate the ultrasonic sensor.

```
class UltrasonicAccuracyTest < Minitest::Test
  def test_the_ultrasonic_sensor_is_accurate_enough
    readings = CSV.parse(File.read("us_readings.csv"))
    readings = readings.transpose
    readings[1] = readings[1].map { |i| i + 2 }
    assert_approx_equal_arrays(readings[0], readings[1])
  end
end
```

Ultrasonic Mapping Test (2D)

This test was written to develop the conversion of ultrasonic readings which are in polar coordinates to cartesian coordinates. Here only the horizontal scan of the ultrasonic sensor is considered. The custom assertion *assert_approx_equal_vectors* is used to assert if the two vectors are within certain delta of each other. *test_cases* is a dictionary|hash table containing expected cartesian coordinates for a particular angle and distance.

```
class UltrasonicMappingTest2D < Minitest::Test
  test_cases.each do |c|
    angle = c[:readings][0]
    r = c[:readings][1]

    method_name = 
"test_it_returns_correct_cartesian_coordinate_for_angle_
%.3f_distance_#{d}" % [angle, r]

    define_method(method_name) do
      vec = Vector.from_servo_angles(*c[:readings])
      assert_approx_equal_vectors Vector[*c[:expected]], vec
    end
  end
end
```

Ultrasonic Mapping Test (3D)

This test was written to develop the conversion of ultrasonic readings which are in polar coordinates to cartesian coordinates. Here the horizontal and vertical movement of the ultrasonic sensor is considered.

```
class UltrasonicMappingTest3D < Minitest::Test

  test_cases.each do |c|

    h = c[:readings][0]
    v = c[:readings][1]
    d = c[:readings][2]
    method_name = "test_it_returns_correct_vectors_for_angles_
%.3f_%.3f_distance_#{d}" % [h, v]

    define_method(method_name) do
      vec = Vector.from_servo_angles(*c[:readings])
      assert_approx_equal_vectors Vector[*c[:expected]], vec
    end
  end
end
```

RL Agent Cumulative Rewards

MountainCar-v0 is considered “solved” when the agent obtains an average reward of at least -110.0 over 100 consecutive episodes. The reward for each episode is stored on new lines in *rewards.txt*.

```
class RLAgentRewardTest < Minitest::Test
  def test_the_environment_is_solved
    totals = []
    rewards = File.readlines("rewards.txt")
    while rewards.length > 0
      slices = rewards.each_slice(100).to_a
      slices.pop if slices[-1].length < 100
      totals.push slices.map { |a| a.inject(0, :+) / 100.0 }
      rewards.shift
    end
    totals.flatten!
    assert totals.any? { |i| i >= -110 }
  end
end
```