University of the West of England
BRISTOL

Bristol Robotics Laboratory

DEPARTMENT OF ENGINEERING

Dynamic coverage of territorial robots in GPS deprivedenvironment

Feifei Rong

Supervisor: Luca Giuggioli

Alan Winfield

A dissertation submitted in partial fulfilment of the requirements of the University of the West of England, Bristol for the Degree of Master of Science September 2017

Abstract

The aim of this project is to apply the area coverage algorithm to E-puck robot in order to find important issues that distinguish experiments in real environment from simulations. In the experiment, individual e- puck robot is able to repel upon detection of obstacles by proximity sensors. The positions of collision are then stored to robots' memory. When a robot is within the impact range of the collision points in its memory, the robot will turn to an angle and escape. Front cameras of e-puck robots are used to distinguish between boundary and other robots. The results of experiments show the high boundary detection error rate with low luminance of the environment. Other issues include efficient sensor data handling, error with drifting and calibration, relationship among impact range, memory time and coverage time.

Acknowledgements

I would like to express my thank to my first supervisor Luca and my second supervisor Alan as well as people in Swarm Robotics group: Lenka, Tom, Simon and Paul, who gave me much support on getting familiar with experiment environment.

Secondly, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The swarm behaviour is a product of long-term evolution in nature. Gregarious creatures obtain this survival skill by adapting themselves to the rule of Nature Selection, such as birds forming different shapes while migrating, bees building up nests and groups of fish swimming in specific directions. These intelligent behaviours of animals all appear to be self-organized, show good group stability and environmental adaptability. However, individuals in swarm groups do not show properties of intelligence. The phenomenon that a group exhibits intelligent properties individuals do not have is named ‘Emergence’. The emergent behaviours of swarm entities are caused by frequent interactions among individuals (Johnson, 2006). Behaviours of individuals can follow an extremely simple pattern.

Figure 1-1 Swarm behaviours in nature

The research field of Swarm Robotics is inspired by the emergent behaviours of creatures in nature. The concept of Swarm Robotics mainly refers to the construction of a multirobot system that emerges intelligence on swarm-level when individual robots interacts with each other or with the environment (Dorigo & Şahin, 2004). The physical structure of individual robots in a swarm robotic system are relatively simple, which has made individual agents cheap and easy to implement. The small size of individual agents has given flexibility of swarm robotic system. Meanwhile, the changeable quantity of individuals has ensured the robustness (where the loss of several individuals has small effect on swarm performance) and expandability. Thus, swarm robotic systems are suitable for conducting tasks such as surveillance and explorations.

Area coverage is one of the core tasks in exploration and surveillance. Under normal circumstances, GPS is used to collect topographic information of target areas in advance. However, when it comes to problems that GPS systems cannot help, such as deep marine

or the complex indoor environment, an efficient method capable of exploring and conducting tasks in such GPS deprived environment is what we need. Example area coverage scenarios include rescue mission after natural disasters, underground exploration for gas, and monitoring forest fires. The area coverage problem has been studied for years, from approaches of traditonal single robot (Kuipers & Byun, 1991) to multi-robot systems (Dasgupta & Taylor Whipple, 2013). The main challenges in sloving area coverage problem using swarm robotic systems exist in finding an efficient intelligent algorithm that can make swarm robots act in a decentralized and self-organized way.

The intelligent algorithm used in this thesis is mainly based on Luca Giuggioli‘s paper in 2016 (Giuggioli, et al., 2016), in which the author proposed a novel approach inspired by territorial behaviours of birds and mammals. This approach has no need of shared coordinate system and global communication. The main concept is to deploy a swarm of robots, where each robot only has local awareness of its surroundings. They move randomly at the beginning and remember where they get collided by other robots to avoid those collision positions. After a period of time, those collision positions in their memory will force them to form territories. The memory time for collision positions is limited considering the dynamics of environment. Simulations have been done in the 2016 paper exploring how changes in parameters can affect the time for a full coverage of a target area.

However, things can be different in practice. The aim of this project is to develop an algorithm that can make a nubmer of e-puck robots do an area coverage task by remembering collision points and repel, in order to find important issues thay only happen in reality other than simulations. To accomplish this goal, objectives are stated below:

- Information on existing algorithms will be checked and the mathematical model of proposed method will be built.
- Getting familiar with the hardware and software setting is essential before programming and debugging e-puck robots.
- The construction of a finite state machine (FSM) is vital for organizing behaviours of swarm robots in the arena.
- Sensors can correctly sense nearby object.
- Cameras on e-puck robots are used to distinguish boundaries and other e-puck robots.
- Position and heading of each robot should be calculated by themselves according to time and robots’ speed.
- The robot memory need to programmed as a list to update in every time step.

- Important issues concerning the actual environment need to be addressed through experiments

1.2 Thesis Overview

The thesis consists of four chapters: literature review, methodology, results and analysis, discussion and conclusion. Literature review provides basic knowledge in understanding the thesis. In this chapter, the definitions and existing features of swarm robotics is introduced. A history of control methods and applications of swarm robotics will point out the importance of this research area. Research domains of swarm robotics such as hardware modelling and searching of efficient algorithm will indicate unresolved problems in this field. The review of several area coverage algorithms leads to the two important concepts: random walk and repelling methods. The reason of choosing E-puck robot platform is clarified by comparison with other platforms. E-puck's hardware structure and software interfaces will be described with knowledge of MCU and SPI.

The part of research methodology will produce three models of finite state machine (FSM) that appropriately describe the software architecture of e-puck robot system. The first FSM only contain the random walk algorithm of repeatedly choosing random angles and distances. The second FSM includes camera detection and repelling function on the basis of the random walk FSM. The last FSM is a further upgraded model with memory and banned area avoidance. Reasons on some probability distribution used for generating random values are given in this chapter. All the states in each FSM will be explained in detail.

Experiments are conducted to test the robot behaviour and reliability. Two repelling methods described in methodology are verified in this section with experiment on single robot and two robots. Five robots will perform area coverage tasks to find out the relationship among impact range, memory time and coverage time. Reliability test consists of boundary misrecognition rate generated by camera under different luminance condition, error caused by drifting or speed change and the ability to deal with dead end. Data will be listed in tables and analysed carefully.

The chapter of discussion will discuss several problems that relates to the issues raised from experiments done in real environment including the defect of the front cameras on e-puck robots, local mapping error caused by motors, sensor data handling issues and how impact range and memory time affect coverage time. Achieve of this thesis and application will also be discussed.

The conclusion part mainly concludes the aim and objectives of this project, along with the actual work being done and the primary findings. Suggestions on further work have been mentioned at the end.

Appendices include the main C++ code that runs on e-puck and some python code for drawing random distributions and plotting graphs.

2 Literature Review

2.1 Swarm robotics

2.1.1 Definitions and Features

So far, there is no precise definition on Swarm Robotics. One suitable definitions produced by Dorigo and Sahin statiing that: ‘Swarm robotics can be loosely defined as the study of how collectively intelligent behaviour can emerge from local interactions of a vast number of relatively simple physically embodied agents.’ (Dorigo & Şahin, 2004). Dorigo and Sahin also gave definitions on essential features a swarm robotics system should possess:

- The system contains large numbers of individual robots.
- The mixing rate of the robot group is low.
- Individual robots are simple and have limited capabilities.
- Individual robots only have personal awareness.
- Loss of several robots has little effect on system performances.

According to the features stated above, a simple design in individual robot has made it cheap and easy to implement physically. The swarm robotics system can also be self-adaptive to complex dynamic environment. Meanwhile, the loss of individuals has small effect on performance. The distributed architecture based on local perception and interaction has significantly reduced the communication volume among individuals, which provides the system with expandability. Therefore, compared to expensive and complex single robotic systems, a swarm robotic system has huge advantages over robustness, expandability, flexibility and economy.

2.1.2 Developments of control methods in Swarm Robotics

The research in robotic swarm system has experienced four stages. At the earliest stage, a lot of researches have been done to discover the theory behind the swarm intelligence among animals and insects. After the establish of relevant theories, many physicists and experts in computer science started to do simulations, even actual experiments, in order to prove that such emerging phenomenon on swarm-level can be achieved by applying simple interaction rules on individual agents. At the third stage, more rigorous mathematical models have been built to support actual applications.

The original control method in a swarm robotic system can trace back to the observation of swarm behaviours among insects, birds and fish (Shaw, 1978) (Ballerini, et al., 2008). In 1940s, when Shannon and Wiener were building a turtle like robot with touch and light sensors (Dorf & Nof, 1990) (Holland, 1997), they have found that the behaviours of the

whole system can be realised by coordination of simple individuals. In 1990s, the number of research on the relevant application of multi-robot system (with only a few individuals) has gradually risen. A famous project named CEBOT in Japan (Fukuda, et al., 1989), which introduced concept on environment reconstruction of a cellular robot; Multi-robot cooperation in the MARTHA project (Alami, et al., 1998) has given a first try on realising coordination capabilities of a large fleet of robots.

With the development of MEMS (Micro-electromechanical Systems), mobile micro robots will have advantages on smaller and cheaper mechanical design. Meanwhile, micro robots have brought a wide prospect to application areas of swarm robotics. Application includes environment inspection on the microscale and handling of tiny objects in the fields of mechanical and biology.

Another emerging research area of swarm robotics is Bio-nano-robot. The area mainly focuses on the functioning of individual nanorobot. The concept of bio-nano-robot was raised from an article by an American scientist called Feynman in 1960 (Feynman, 1960). Nowadays researches tend to follow two main directions in swarm robotics: one is the operation on microscale material using microscale operation parts on microscale device; the other aspect refers to the design, simulation, control and coordination of microscale robots.

2.1.3 Research Domains

Research domains of swarm robotics mainly contain five aspects: construction and analysis of bio-inspired swarm models, design of individual robots, control methods in individual cooperation, communication/interaction modes and swarm system architecture.

Swarm model building and analysis is an important area in the research of swarm intelligence. It has heuristic meanings in swarm cooperation control. This domain mainly refers to how to choose the right behaviour of creatures and build proper models for it. Sufficient analysis of requirements of target robot system is needed, so as to choose the correct behaviour of creatures.

In swarm robot systems, the simpleness of individuals is an essential feature. It reflects in limited ability in perception and computation, along with simple control structures. Simple control structures can enhance the reliability of the whole system and quickly response to changes in the environment. The simpler, the cheaper, which has made a significant number of robots possible. There are two robot control structures: deliberative and reactive (Arkin, 1998). Deliberative structures require strong computation ability. In contrast, reactive structure links perception and reaction, no abstract of the environmental model is needed, which fulfils the nature of swarm robots.

Control methods in individual cooperation require consistency. The applications of swarm robot systems usually need to use information sharing to coordinate swarm behaviours. Thus, the key idea is to design a proper strategy on consistency to force individuals to remain consistent under unstable condition. The problem on consistency has a long history (Lynch, 1996). Consistency means the state of individuals in a swarm system tends to converge as the time goes. Recent years, revolution algorithms have drawn much attention (Nolfi & Floreano, 2000), however, it needs accurate models, which limits the applications (Matarić & Cliff, 1996) (Watson, et al., 2002). Besides, reinforcement learning does not require prior sample training. Its online learning technique would help reinforce the healthy behaviours of swarm robot system (Matarić, 1997).

In a swarm robot system, the interaction and communication among robots are essential for cooperation purpose. When the swarm is executing a specific task, individuals need to know information on the current local scale and the status of other individuals. There are two interaction modes: perception and communication (Arkin, 1998). Perception refers to behaviours of ants leaving pheromone (Greene & Gordon, 2003).Individual robot acquires information indirectly by sensors and information left by other robots. Communication refers to the ability to transfer information globally, which is efficiently but would be limited by complex communication devices and protocols, along with noises and delays in the system.

The difference between individuals in a swarm robot system is an obvious problem. In a complex environment, a system consists of robots with different abilities is a very realistic situation. The swarm architecture can be divided into two: homogeneous and heterogeneous. In nature, by referring to ants and bees, homogeneous architecture is unreasonable (Birk & Belpaeme, 1998). Heterogeneous architecture needs to conquer problems such as the degree of heterogeneous, task allocation, swarm coordinate and cooperation, communication (Parker, 1994) (Jung & Zelinsky, 2000).Task allocation needs to be a self-organized process due to the existence of swarm intelligence (Krieger & Billeter, 2000). Also, human-robot sociology is another research direction aims to gain better performance of swarm robots.

2.2 Area coverage algorithms

2.2.1 Introduction

Area coverage refers to the concept of putting a number of robots in an area and let them move in a specific pattern, such that every point in this area is visited by one or more robots. The minimization of time it takes to get full coverage is usually another optimization problem. In the field of area coverage, there used to have two main approaches. The stigmergic approach (Giuggioli, et al., 2013), which is inspired by ants' behaviour, relies on

environmental marking by the robots to direct their motion towards uncovered territory. One benefit of this approach is the simplicity of control algorithm, since robots only need to move randomly. No localization or memory of markings are needed. However, this approach leads to the redundancy of coverage and relies strongly on duration of the makers' existence. Building robots with actual marking and reading mechanisms is quite difficult in practice. Another approach is the collaborative approach (Kong, et al., 2006). It requires communication among robots no matter how far they are. This approach not only requires localization and navigation, but also memory and communication for task assignment, which refers to a shared coordinate system. To satisfy these requirements in the real world is a big challenge.

A new approach, inspired by bird chirping, was introduced in 2016 (Giuggioli, et al., 2016). When two robots meet, they find each other and remember the location of the meet point in their own coordinate system and treat it as a border where they cannot go beyond. Robots in the system only need to have localization ability and memory. Global communication is not needed. This approach saves a lot of resource in building individual robots, since only localization and memory are considered as abilities. However, the simulation of this approach is done with very simple characteristics. The failure of individual robots and the noise distribution need to be carefully considered in the model itself in order to mimic a more realistic environment.

Algorithm 1 SINGLE-STEP

Require: Current position $\vec{p}$
1: Randomly choose λ, ϑ
2: $\vec{\ell} \leftarrow$RANDOMWALK(λ, ϑ)
3: **if** A robot is detected in position $\vec{b}$ within d
OR
Remembered location $\vec{b}$ is within d **then**
4: $\vec{\ell} \leftarrow \vec{b}$
5: Move to new position $\vec{\ell}$
6: If robot or mark detected, REPEL.

Figure 2-1 Memory-Based Territorial Exclusion Mode (Giuggioli, et al., 2016)

The figure shows the Memory-Based Territorial Exclusion Model proposed in Luca's 2016 paper. It requires a random walk algorithm at the beginning to choose a random angle and a value for random distance. After that, if the robot has detected an object by its onboard sensors, the robot will remember the position of the object and repel. If the robot has moved to a place near one of the positions retrieved from its memory, the robot will avoid memorised position as well. The random walk algorithm and repelling methods are introduced in the next section.

2.2.2 Random Walk

The early random walk theory is inspired by the motion of individual pollen particles studied by Robert Brown in 1828 (Brown, 1828), which is now known as Brownian motion. The random walk theory was then further studied in areas such as random noise and spectral analysis by physicists (Einstein, 1905) (Von Smoluchowski, 1916). Until 1930, a theory concerning mean-reversion process in random walks had been developed (Uhlenbeck & Ornstein, 1930). Based on Edward's paper in 2008 (Codling, et al., 2008), five random walk algorithms will be introduced in this section to get an idea of crucial elements in the random walk theory.

Simple Random Walk (SRW): The simple random walk algorithm is unbiased and uncorrelated, meaning it has no preference in directions, and the location after each step only reply on its previous step. The SRW process is regarded as a Markov process (Kareiva & Shigesada, 1983). However, this approach is completely random and doesn't consider any existing feature of target tasks.

Correlated Random Walk (CRW): The locations selected at each step of correlated random walk process is no longer a Markov process, but has some persistence on a specific direction. This is because the angle selected by CRW at each step is closer to its previous selection. However, the velocity at each step can be a Markov process (Othmer, et al., 1988). CRW is useful in modelling animals' paths, since most of animals also have persistence on moving forwards (Siniff & Jessen, 1969).

Biased Random Walk (BRW): BRW is biased on the probability of selection. A preference of selection indicates a higher probability of selecting a specific direction. When the robot is moving randomly but towards a target, it is a based random walk. BRW can be applied on the modelling of chemo sensitive cells like bacteria, which has some reaction towards drugs (Alt, 1980).

Biased CRW (BCRW): BCRW combines long-term and short-term goals and maintains stochasticity at the same time (Duchesne, et al., 2015). The BCRW approach is considered to be a flexible and powerful model for animal movements. In this thesis, we only talk about CRW, since long-term goals are not considered in the area coverage behaviour.

Levy walk (LW): This algorithm is unbiased and uncorrelated. But its distribution for step length has an infinite variance. Thus, the distribution is heavy tailed. Many animals have behaviour patterns that like levy walk, such as reindeers and spider monkeys (Benhamou, 2007).

Since many mammals have a persistence to move forward, the correlated random walk is chosen as the movement pattern that followed by swarm robots in this thesis.

2.2.3 Repelling Methods

After a random angle and distance is selected by the random walk algorithm, individuals in a swarm will start to move and meet other individuals in the same area. When two robots meet, or when a robot approaches a boundary area, the robot will have to avoid and move to a new position. This avoiding behaviour is called repelling. Three repelling methods will be introduced in this section.

Repelling upon current heading: In Luca's paper of 2016 (Giuggioli, et al., 2016), he introduced a collision repelling model that can repel the boundary area and other robots efficiently.

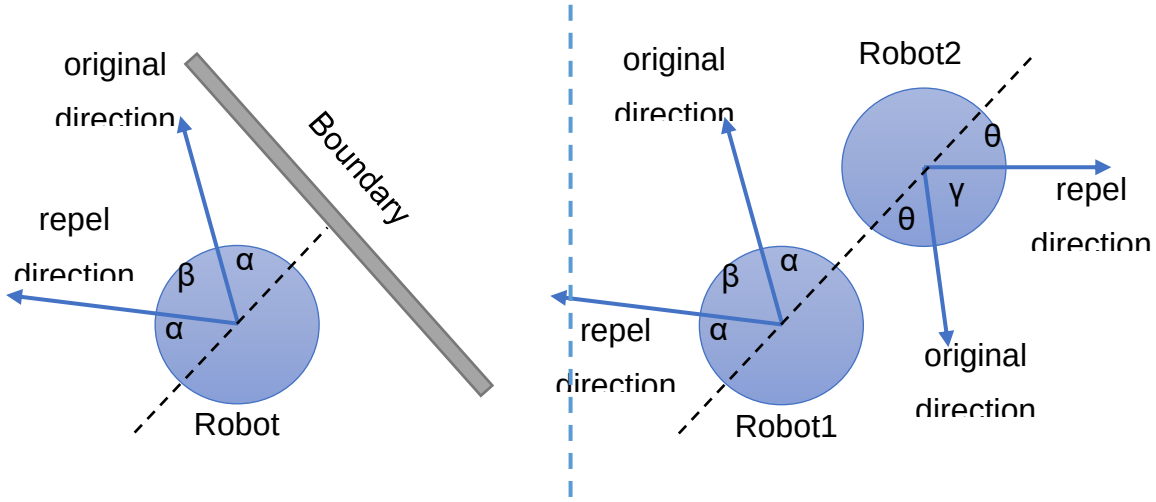

Figure 2-2 Robot repel process on detecting boundaries (left) and another robot (right)

In Figure 2-2, left shows a condition where a robot detects a boundary area on its right. The original direction has an acute angle α towards the boundary area. The robot then turns left with an angle β, where β = π– 2 * α. This algorithm has an advantage that takes the robot's current heading into consideration. When α is large than 90 degrees (absolute value), meaning the robot is not going towards the boundary, no repelling behaviour will be activated under this circumstance. The more the robot direction heads towards the boundary, the more will the robot tries to repel. The right image shows the condition when two robots detect each other on their ways. They will both try to repel an angle with value β and γ according to the other robot's position. This repelling method is efficient since the repelling angle is calculated on the original direction. The method maintains a good persistence of direction on exploring the target area. However, without any randomisation, this method can lead robots to a dead end and make it move in circles when the exit is very narrow.

Braitenberg Algorithm: The Braitenberg algorithm is originally from the concept of the Braitenberg vehicle (Braitenberg, 1986). The motion of the Braitenberg vehicle is directed by

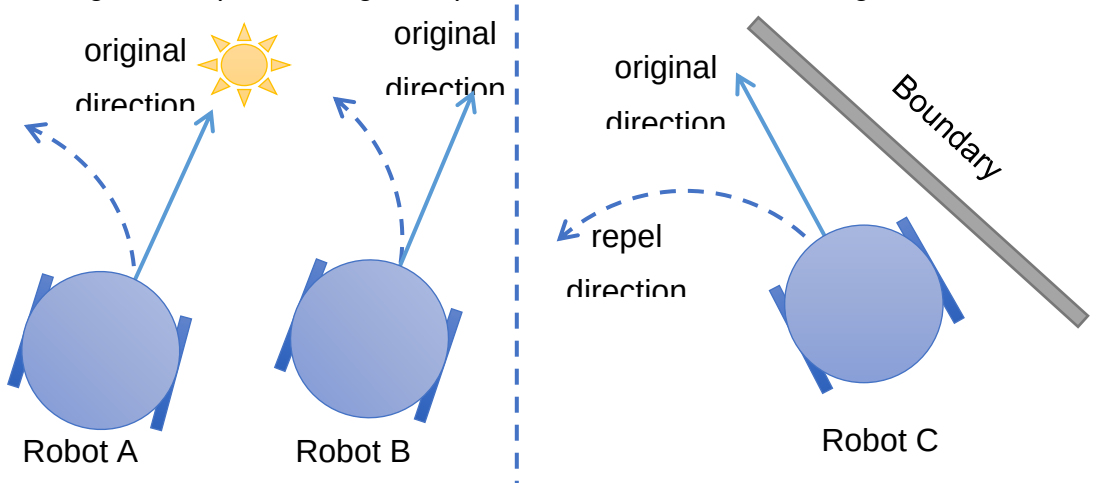

Figure 2-3 Braitenberg vehicles.

Robot A is elusive to light sources. Robot B is applausive to light sources. Robot C is avoiding a boundary area.

sensors around it.

In Figure 2-3, we assume that all three robots have been equipped with differential wheels. Robot A and B have been equipped with light sensors around them and Robot C has been equipped with distance sensors. For robot A, if its light sensors have detected light source on its right, as is shown above, the robot will increase the speed of its right wheel. Thus, the robot will start to turn left and exhibit a movement that is elusive to the light source. On the contrary, Robot B will turn left and head to the light sources, since it is programmed to increase the speed of right wheel upon detecting light sources on its left. When it comes to the obstacle avoidance problem, when distance sensors of Robot C have detected the boundary area on its right, the robot will increase its speed of the left wheel and exhibit an avoiding behaviour escaping the boundary area. The trigger from sensors to actuators can be ipsilateral or contralateral, and excitatory of inhibitory. Ipsilateral and excitatory trigger will lead to repelling behaviours. Contralateral and excitatory trigger will lead to movements that towards the source. The Braitenberg vehicle has a simple reactive architecture. It is efficient and the functioning is purely mechanical. However, this approach has fixed response to a given situation and all the reaction to a specific environment need to be predefined.

Repelling with random angles: Randomisation is a useful process to avoid the disadvantages brought by fixed pattern. This approach is based on Adam's thesis in 2015 (SABRA, 2015).

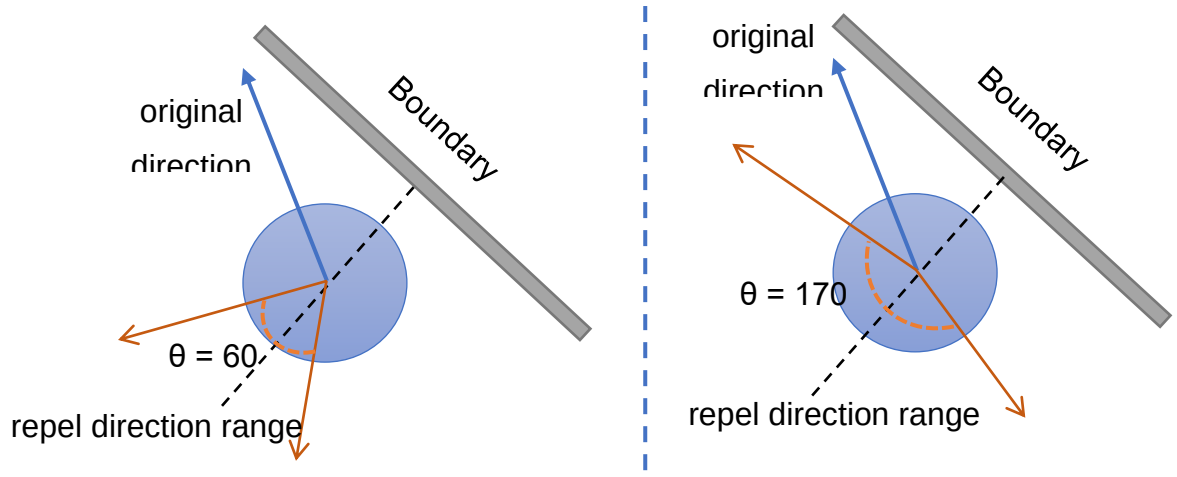

Figure 2-4 Repelling with random angles in a range

In Figure 2 3, when the robot sees a boundary, the robot will turn around with an angle between a range. The left robot has a repel range of 60 degrees and the right robot has a repel range of 170 degrees. This approach doesn't maintain the moving trend of the original direction, but turns the robot according to the relative position of the boundary. This can cause the robot repeatedly passing through the same area and produce redundancy. The lack of efficiency if this approach requires further development or combination with other existing repelling methods.

2.3 Swarm Robot Platforms

Swarm robotics as an emerging research field that has drawn much attention these years, many programmable platforms have been developed to cater to researchers' need in simulation and application. Because of the large number of individuals in a swarm system, the lab research of swarm robotics will require small size but a sufficient number of sensors on individual agents. Meanwhile, the cost is also an important issue in choosing a suitable swarm platform. A comparison among different platforms are indicated in this section as well as the reason for choosing E-puck robots in this thesis. Hardware and software configuration of e-puck robots will also be introduced.

2.3.1 Comparison among platforms

This section focuses on comparison of existing swarm robot platforms, including AMiR (Arvin, et al., 2009), Alice (Garnier, et al., 2008), E-puck (Mondada, et al., 2009), Kilobot (Rubenstein, et al., 2014), Kobot (Turgut, et al., 2007), and SwarmBot (McLurkin, et al., 2006). Comparison is done on size, speed, sensors and endurance.

Table 2-1 Comparison of swarm robot platforms

Robot	Size	Speed	Sensors	Endurance	Open source	Comments
AMiR	6 cm x 7.3 cm x 4.7 cm	10 cm /s wheel	IR distance sensors	2 h	yes	Honeybee aggregation
Alice	2.2 cm cube	4 cm /s wheel	distance, camera	10 h	yes	Embodiment of cockroach
E-puck	7 cm	13 cm/s wheel	distance, camera, accelerometer, microphone	2 h	yes	Educational use
Kilobot	3.3 cm	1 cm/s vibration	distance, light	3 – 24 h	yes	Group charging
Kobot	12 cm	NA	distance, compass, vision,	10 h	No	NA
SwarmBot	13 cm x 13 cm x 13 cm	50 cm /s	distance, camera,	3h	yes	Able to find charging station

Compared to other swarm robot platforms, the size of e-puck robots is medium and is suitable for lab environment with experiment arena of 2 m x 1.5 m. Alice and Kilobot are too small to observe its behaviour. Swarm robot is a bit too large and the maximum speed of 50 cm/s is a waste to the current lab environment. The size of AmiR and Kobot are suitable for the experiment, but no camera is installed on AmiR. Meanwhile, the software and hardware documentation of Kobot are not open source, which means it can be hard to get familiar with it at the beginning. Although the battery of E-puck doesn't last very long, it can travel with maximum speed of 13 cm/s, which compensates the disadvantages on endurance. The combination of distance sensors and cameras is suitable for area coverage tasks.

2.3.2 Hardware structure of e-puck robots

The basic configuration of e-puck robots includes 8 IR proximity (distance) sensors, one 3D accelerometer, 2 green LEDs, 9 red LEDS, 3 microphones, one speaker. It can be connected wirelessly using Bluetooth. Extension ports are provided on the master board to let users connecting other sensors and actuators that suitable for their experiments. Example peripherals include Zigbee unit, Colour LED module and Omnidirectional vision unit. All the information collected by sensor units are procced by A MCU (Micro Processor Unit) dsPIC30F6014A.

However, this MCU has limitations on memory and computation. The 133KB flash and 8KB RAM have limited the amount of information processed by CPU from parallel working sensors. The condition can be even worse for vision detection task. The onboard camera has a resolution of 640x480, but the original board is not able to process the whole frame, only a small fraction of the image is captured. The frame rate is also constrained by the ability of this MCU. As a swarm robot, without any global communication, individuals need to store information data collected by themselves from the environment. Sometimes, real-time data processing is also needed. Therefore, a Linux extension board has been installed on top of the original master board to support more complex experiments. (Liu & Winfield, 2011).

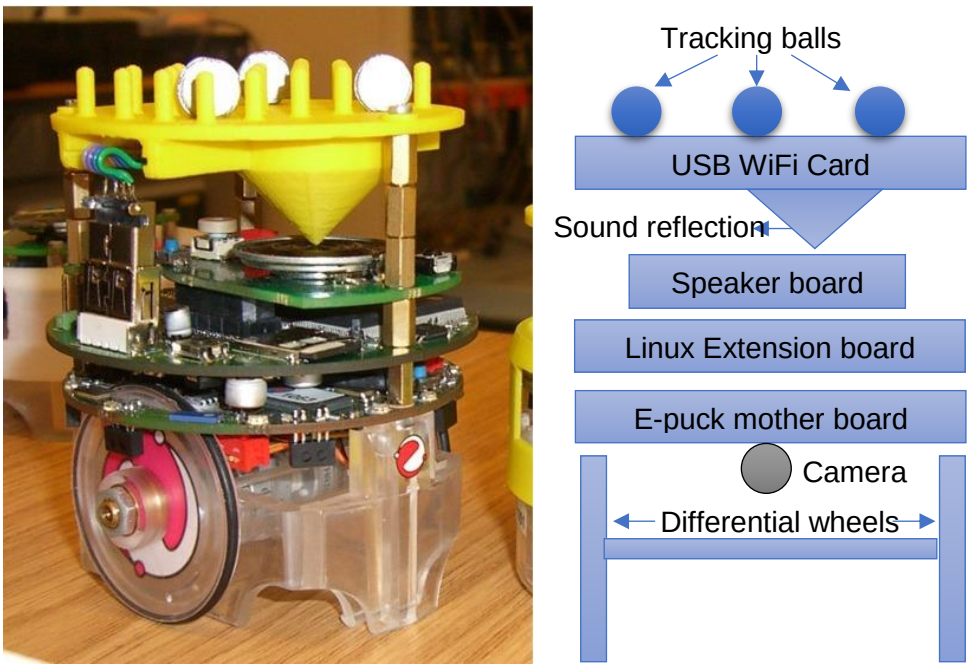

Figure 2-5 E-puck robot with Linux extension board

The architecture of upgraded e-puck is shown in Figure 2-5. At the bottom is the original mother board of e-puck, on top of that is the Linux extension board with ARM 9 microprocessor. The mother board is acting like a slave for Linux board, in charge of

processing low-level sensor data and sending commands to control two motors of differential wheels. The CMOS camera is directed connected to the extension board in order to overcome the problem on storing capture data with only mother board. The yellow plastic part installed on the very top of e-puck robot provides three functions. The part with circular cone shape is an amplifier of sound that emitted by the speaker board. USB WiFi card allows user to connect to e-puck using fast wireless communication methods, which is convenient when debugging the program. The tracking balls are part of the Vicon system in the lab. The Vicon tracing system (http://www.vicon.com) consists of four cameras that installed at the celling of the lab area recording from four different directions. The system then constructs images from these four cameras to build a 3D real-time motion model. The tracking balls on the e-puck is made from reflective material that allows four cameras to recognize. Each e-puck will have different patterns of balls, letting Vicon system distinguish e-pucks from each other.

2.3.3 Software interfaces

It is known that the mother board and the Linux extension board work in parallel to ensure the normal working state of e-puck robots (Liu & Winfield, 2011). The SPI bus is used to communicate through the two boards using master/slave mode. Since SPI is full duplex, data can be transmitted and received at the same time. On each side of SPI, a buffer is defined to formulate the data type that goes through the communication tunnel.

```
struct txbuf_t
{
    struct cmd_t cmd;           // first two bytes for commands
    int16_t left_motor;         // speed of left motor
    int16_t right_motor;        // speed of right motor
    struct led_cmd_t led_cmd;   // command for leds
    int16_t led_cycle;          // blinking rate of LEDSs
    int16_t reserved[19];       // reserved, to make two buffers the same size
    int16_t dummy;              // leave it empty
};
struct rxbuf_t
{
    int16_t ir[8];              // IR Ranges
    int16_t acc[3];             // Accelerometer x/y/z
    int16_t mic[3];             // microphones 1,2,3
    int16_t amb[8];             // Ambient IR
    int16_t tacl;               // steps made on left motors
    int16_t tacr;               // steps made on right motors
    int16_t batt;               // battery level
};
```

Figure 2-6 Communication buffer on the Linux side (Liu & Winfield, 2011)

```
struct txbuf_t
{
    int ir[8];                  // IR range values
    int acc[3];                 // Accelerometer x/y/z
    int mic[3];                 // microphones 1,2,3
    int amb[8];                 // Ambient IR
    int tacl;                   // steps made on left motors
    int tacr;                   // steps made on right motors
    int batt;                   // battery
};
struct rxbuf_t
{
    int dummy;                  //leave it empty
    struct cmd_t cmd;           //first two bytes for commands
    int left_motor;             //speed of left motor
    int right_motor;            //speed of right motor
    struct led_cmd_t led_cmd;   //command for LEDs
    int led_cycle;
    int reserved[19];           //reserved, to make two buffers the same size
};
```

Figure 2-7 Communication buffer on the mother board side (Liu & Winfield, 2011)

Figure 2-6 and Figure 2-7 show two buffers defined on each side of SPI. It can be seen that the Linux board send commands of motor speed and led blink mode to the e-puck mother board and receive data of IR sensor detection value, accelerometer data, current battery level and steps made by motors. The mother board acts like an intermediate agent that organize commands and feedbacks between Linux board and Peripherals.

C++ is used as a programming language to the Linux board. Compiling of codes is finished on PC by copying the file system of the Linux board and mount it in Linux operating system on PC. After compiling, an executable file will be generated, which can be transmit to the Linux board through WiFi network with SCP (scure copy) protocol.

3 Research Methodology

The aim of this project is to develop an area coverage algorithm that can be applied on e-puck robots. The e-puck robots should have ability to repel upon colliding with other robots and boundary areas. Collision points need to be remembered and be kept away in order to let each robot form a territory. After reading relevant materials on the concept of swarm robotics, area coverage algorithms and e-puck programmable platform, a finite state machine (FSM) should be generated to organize the behaviours of a number of e-puck robots. A list of memory with a limitation on memory time for each e-puck robot needs to be updated in every time step. The on-board cameras of e-puck robots are used to distinguish boundaries and robots. Position and heading of each robot will be updated by itself at each time step. This section will introduce the methodology from a basic finite state machine describing with only random walk, to a very complex model that is able to accomplish the expected task.

3.1 Simple FSM for Random Walk algorithm

The simple finite state machine is a description of a correlated random walk algorithm. It starts with choosing a random angle and turns to the angle. After that, a random distance is chosen and the robot will move to a new position. Once the movement is finished, another loop will begin with choosing a random angle again. The correlated random walk algorithm is a short-term target oriented method. The angle selected at each step is closer to its previous selection. Most mammals' behaviours can me modelled by the correlated random walk algorithm, since mammals have a preference to walk forward.

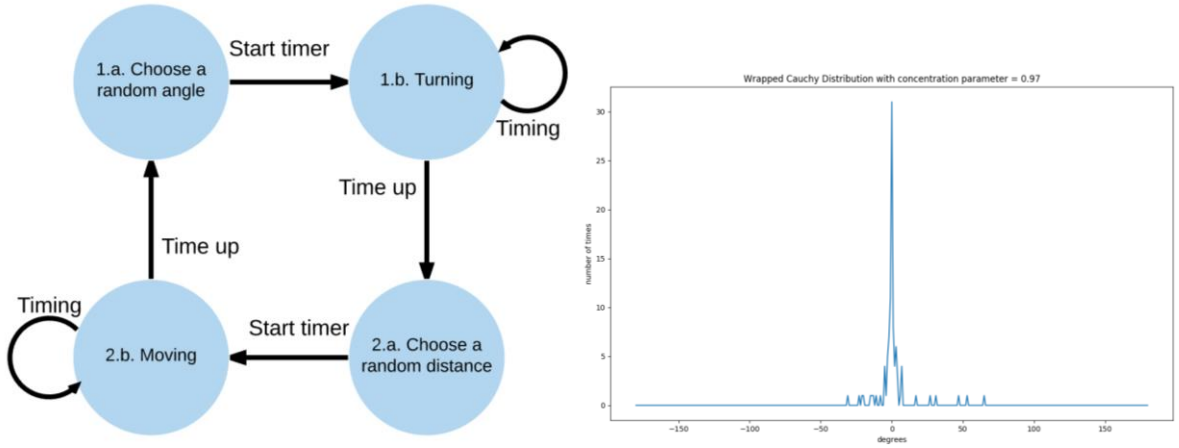

Figure 3-1 Simple FSM for Correlated Random Walk algorithm (left) and Wrapped Cauchy distribution (right)

State 1.a Choosing a random angle: The process of choosing a random angle is based on Wrapped Cauchy distribution.

$$\theta = 2\tan^{-1}\left\{\frac{1-\rho}{1+\rho}\tan\left[\pi\left(U-\frac{1}{2}\right)\right]\right\}$$

Where ρ is the concentration parameter, U is a random number from the standard uniform distribution within interval [0,1]. The concentration parameter used on e-puck in this project is 0.97. The Wrapped Cauchy distribution is shown on the right of Figure 3-1 when concentration parameter equals to 0.97. The distribution graph was drawn by generating 100 random values from Wrapped Cauchy distribution and calculating the number of the same value generated. The graph shows an extremely high concentration around zero degree, which fits in the persistence feature of correlated random walk. The larger the concentration parameter is, the higher the probability will be for the robot to keep the similar direction as the previous step. In Figure 3-2, when concentration equals to 0.1, the graph shows an average probability of choosing angels. Most of the angle values have been chosen one time. A slightly higher probability of choosing angles around zero is shown on the right graph when concentration parameter equals to 0.5.

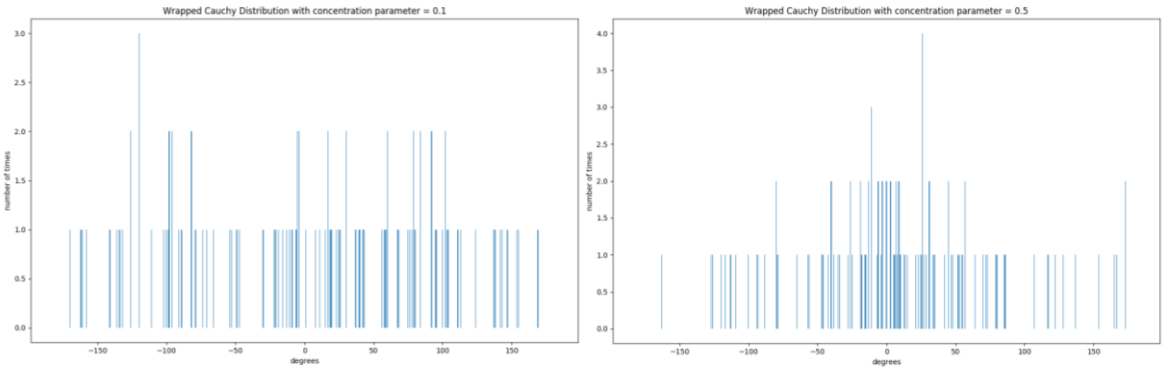

Figure 3-2 Wrapphed Cauchy distribution when concentration parameters equals to 0.1 and 0.5

After an angle is chosen, a timer which equals to the angle value in degree will be set. This is because the movement of the differential wheels is controlled by time. Every single time loop of an e-puck robot is set to be 10ms and a speed of 410 is set as a fixed turning speed in the program. The turning speed value is generated by calibration, making the robot turning 90 degrees with 90 time loops. This is more convenient if the turning angle and the timer have the same value.

State 1.b Tuning: The turning of a robot is conducted by a positive speed applied on one side of wheels and a negative speed applied on the other. The speed applied on two wheels have the same absolute value. This design has an advantage on simplicity of control. Since each e-puck robot is a cylinder, no matter how much it turns by applying opposite direction speed on two wheels, it does not have any effect on the environment. The robot is still at

where it was before turning. It is worth mentioning that if the turning angle is within -5 to 5 degrees, some displacement will be caused and it can result in errors. This is because the present mechanism of sudden stop and sudden move is not easy to change, the realization of a smooth start and stop will take more time. When the turning time is up, indicating the robot has turned to the desired direction, the robot will enter the next state to choose a random distance.

State 2.a Choosing a random distance: The distance is chosen from an exponential decay distribution.

$$Y = e^{\left(\frac{-U}{\beta}\right)}$$

Where, U is a random value that fits with the standard uniform distribution within interval [0,1]. β is a decay constant.

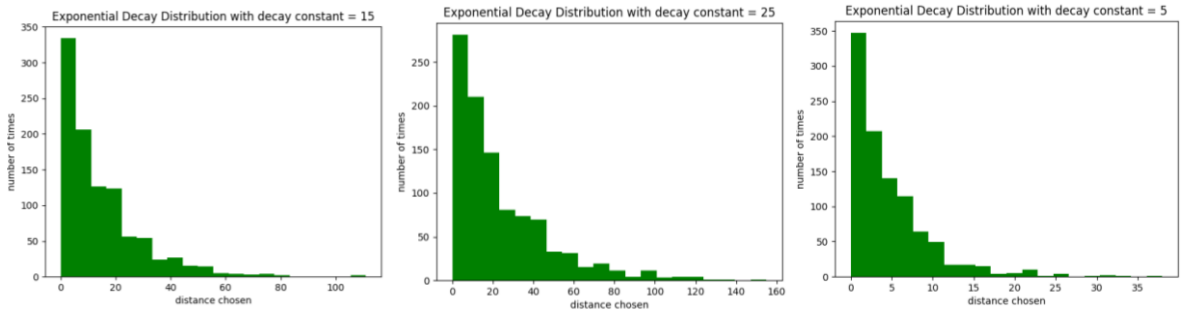

Figure 3-3 Exponential Decay Distribution with decay constant = 15, 25, 5

Figure 3-3 shows histograms of 1000 random values that generated from the exponential decay distribution with different decay constants. All the three figures indicate a tendency that larger number will have lower probability to be enerated compared to smaller values. The probability of value produced near zero is the highest. The left figure has a limitation on value of horizontal axis to be around 100, the middle figure has a limit of around 160 and the right figure has a limit around 35. It can be seen that the larger the decay constant, the higher the limit of value will be. A decay constant of value 15 is chosen in this project and the unit for the produced random nunber is centi-meter. However, due to possibility of choosing a very large value, values that are larger than 50 will be changed to 50. As said in previous section, sudden stop and start will cause slight displacement error. Thus, values under 7 is changed to 7 as a minimum distance of each step. After choosing a proper distance, a timer will be set for next state.

State 2.b Moving: The behaviour of moving forward is generated by applying the same speed on both differential wheels. The value of a particular speed is calibrated to move 10 cm in 1 sec. Since the time for each loop is 10ms, the timer is set to be ten times the

distance chosen by the last state. Although a calibration process can be eliminated and the position of e-puck robot can be represented using a value that is proportional to speed and elapsed time, we still need to take e-puck's diameter into consideration. Sensors are installed at the edge of e-puck robot, therefore, when a sensor detects an object, the position of the object predicted by the robot itself will need to add the radius value. Meanwhile, sensing distance need to be added to the position of the object as well. Thus, actual distance, either than relevant distance is vital in this part. Thus, the speed is set to be 600 to let e-puck move 10 cm in 1 sec. Timer is set to be 10 * speed before moving. When time is up, e-puck will start to choose random angle again.

3.2 FSM with camera detection and repelling methods

Upon the original FSM described in the previous section, a function of obstacle avoidance is added in this section. The additional function takes place after the robot has finished turning and start moving. Since self-rotation of robot does not have any influence on the environment, sensors are only turned on during forward moving. If no nearby object is detected, the e-puck will remain in its original FSM and move to the expected position before starting another loop. Once one of the sensors have detected an object nearby, the robot will turn to the direction of the object and turn on its camera. The camera mounted in the front of the e-puck robot will capture an image frame and store it for post analysis to distinguish between boundary and other robots. Then, different repelling methods are used for boundary and robots. After repelling to a desired position, the robot will start another loop by choosing a random angle. When the e-puck is moving forward when repelling, sensors' values will be checked as well. In other words, as long as the robot is moving forward, sensors will be turned on in case it runs into any obstacles. The whole process is shown in Figure 3-4.

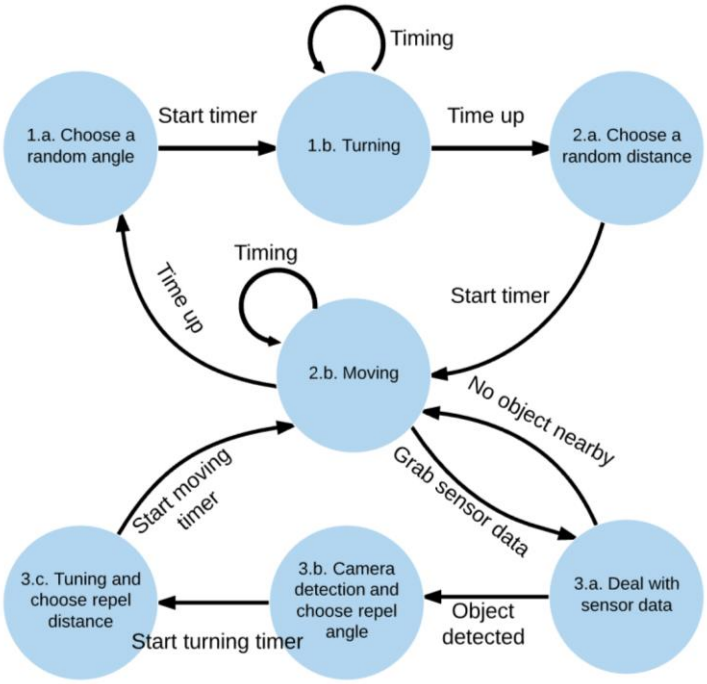

Figure 3-4 Upgraded FSM with camera detection and obstacle avoidance

State 3.a Deal with sensor data: The additional three states can be seen as an inside state of state 2.b, since it is happening after the robot has started moving. Once the robot has started moving forward, it will grab data from the eight sensors every 10ms.

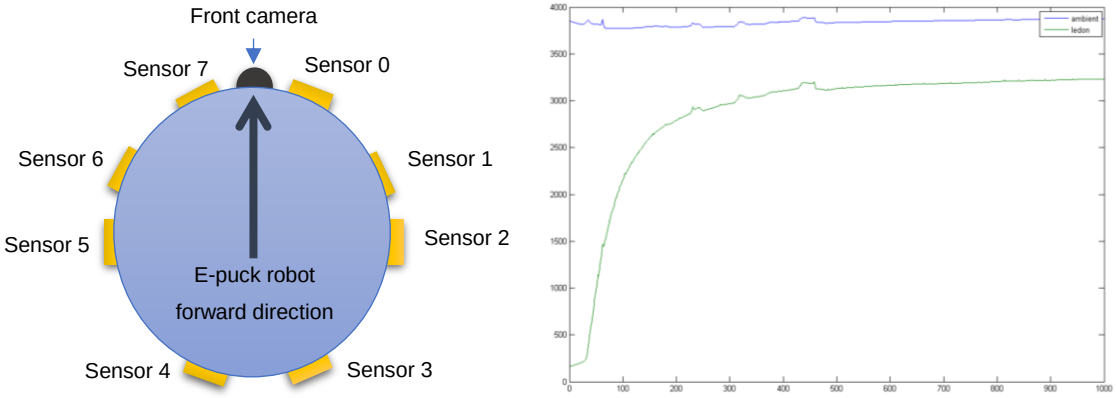

Figure 3-5 Sensor distribution (left) and raw data from sensor 0 when an e-puck robot is moving towards a wall (right)

The distribution of eight proximity sensors are shown in left graph in Figure 3-5. Senor 3 and 4 are not used for detecting obstacle. If a robot is moving forward and another robot is passing by its back, it will not recognize and remain in its movement. Sensor 5 and sensor 2 are responsible for sensing object at directions of -90 and 90 degrees with respect to e-puck's forward direction. Sensor 6 and sensor 1 are in charge of sensing at directions of -60 and 60. Sensor 7 and sensor 0 can be considered to sense object in front of the robot. The graph on the right shows the raw data of sensor 0 when e-puck is moving towards a wall

with condition of different light condition. The sensor's performance can be very poor with insufficient light. Under normal light condition, the closer the robot to the wall, the larger the sensor value will be. The horizontal axis of the graph shows the distance to the wall in steps (1000 steps = 12.8cm). The vertical axis is the raw data value from sensor 0. A limit that confines the concept of 'meeting an object' is set to be 3cm. Although on the right figure, 3cm (234 steps) corresponds to sensor value around 2700, the actual number has been programmed to be 200. If one of the values from eight proximity sensors is larger than or equal to 200, the ID of the sensor will be sent to the next state for further processing. If the values in data sequence are all smaller than 200, the state of the e-puck will become 2.b, meaning it is ok to continue moving.

In practice, data from sensors are not very reliable and the delay in communication is evitable from Linux board to e-puck motherboard. When more than one value is larger than or equal to 200, the program will return the ID of the sensor with the largest value. Another problem is the noise in sensor data. Values of data sequence from sensors can have a sudden rise over 200, which extremely affect the accuracy of result of data processing. This problem is eliminated by letting robots take another four loops, meaning to move another 40ms and compare the grabbed data of every loop. If the ID of the sensor with largest value over 200 has changed or does not exist after 40ms, the program will return the new ID or force e-puck back to state 2.b.

State 3.b Camera detection and choose repel angle: After the sensor ID for obstacles is confirmed, the relative direction of the obstacle is known. The robot will turn to that direction and turn on its front camera. As the sensor distribution shown in Figure 3-5, angles for each sensor ID to turn to face the obstacle is shown in Table 3-1 below. Since sensor 0 and sensor 7 are near the front camera, it is unnecessary for them to turn.

Table 3-1 Angles for each sensor ID to turn if obstacles are detected

Sensor ID	0	1	2	5	6	7
Angle to turn	0	Turn right 60	Turn right 90	Turn left 90	Turn left 60	0

The logic in camera detection of boundary and robots exists in the colour and size of target areas. Debugging of camera settings for recognizing different colours and sizes are essential before applying any camera detection. The debugging interface allows users to record the colour data of target area, as is shown in Figure 3-6. The boundary in the experiment is labelled with blue paper. This is because the original colour of boundary and floor are both white. It is hard to choose values that can distinguish between them. The image on the right

shows the condition that the front camera is facing the boundary. As a result of low luminance at boundary, the colour blue has become very dark, which can be recognized as anything. Therefore, the boundary is not suitable for recognition by cameras.

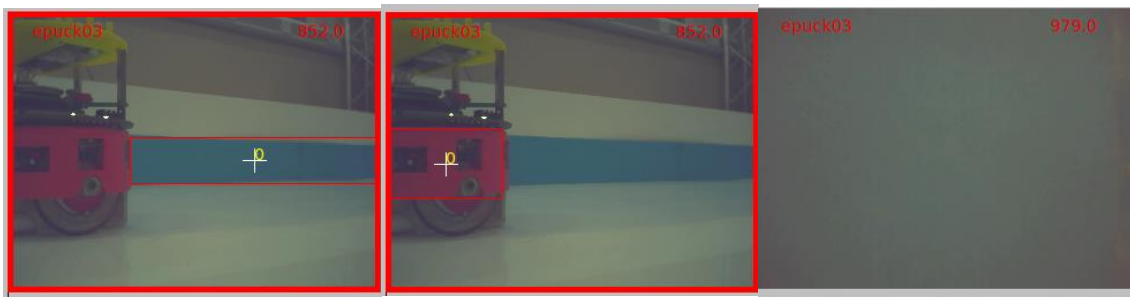

Figure 3-6 Debugging interface for concepts of "Not boundary not robot"(left), "robot"(middle), "boundary"(right)

The method proposed to distinguish between boundary and robots is to define an intermediate condition. When a robot has found an obstacle, it will turn to the direction and turn on camera to see what it is. Assume the obstacle is another robot and the obstacle is in front of the robot, no turning is needed. The camera will see an area of red. The red area can be used to recognize the condition on collision with another robot. However, most of the time, it's the data from the other four side sensors indicating nearby obstacles. Under this condition, the time spent on turning to face the obstacle will make the robot lose the capture of the obstacle (another robot). The front camera can only see either other distant robots or the boundary. Luckily, when the camera sees the boundary with a distance, the colour of boundary is blue, which is different from the condition of close capture stated above. Therefore, the condition of "not boundary not robot" can be detected by the front camera. The overall logic for distinguish between boundary and other robots is: if the condition is either "not boundary not robot" or "robot", the obstacle is a robot; if neither of the two conditions has been fulfilled, it is the boundary.

There are three ways to choose repel angles. Figure 2-2 shows a method concerning the current heading. If the absolute value of angle between current heading and the obstacle direction is less than 90 degrees, the smaller the included angle, the larger the repel angle will be. The second method is the Braitenberg vehicle shown in Figure 2-3. The Battenberg method changes the speed of robots corresponding to data grabbed from sensors. The third method described as choosing random angle around the opposite direction of the obstacle, which is shown in Figure 2-4.

The final method with angle between -30 to 30 is used in the experiment. For the Braitenberg vehicle, the steps of motors are not possible to measure, since the current firmware developed on the e-puck robots does not support this function. It will be challenging to use mathematical approach to model the changes in speed as the error will increase in

practical testing. The first method can waste time. Sensors on robot can only detect 8 directions of obstacles, sometimes sensors are not sensing the shortest distance to the boundary. Therefore, after the first method is used and the robot is planning to move, another sensor may have detected the obstacle (boundary), the robot will have to turn to the direction of obstacle again and repel.

After choosing the repel angle, the robot will set a timer and start tuning.

State 3.c Tuning and choosing a random distance: At the very beginning of the project, a random distance is chosen by the method of state 2.a is used. At a later stage, there is a setting stating that the collision position will only be remembered and repelled after the robot has escaped the impact range of the collision position. Therefore, the distance is set to the value of impact range, in order to leave the impact range efficiently. After starting a timer for moving, the state is then changed to 2.b in case any obstacles detected when the robot is moving forward.

3.3 FSM with memory and banned area avoidance

The robot's behaviour on memory and banned area avoidance are used to form territories. So that each robot will remain in its territory for a period of time until some of the collision positions are forgotten. A memory list is formed with different labels for the boundary and robots. Every time the robot has a collision with the boundary or another robot, the position of the obstacle is recoded and compared with the existing record in its memory. If the obstacle position is not within the impact range of any existing record position, the robot will generate a repelling behaviour. Only positions of collisions with another robot have a limit in memory time, and only these collision positions are repelled. The positions and headings of the robot are updated every 10ms when any moving and rotation behaviour happens.

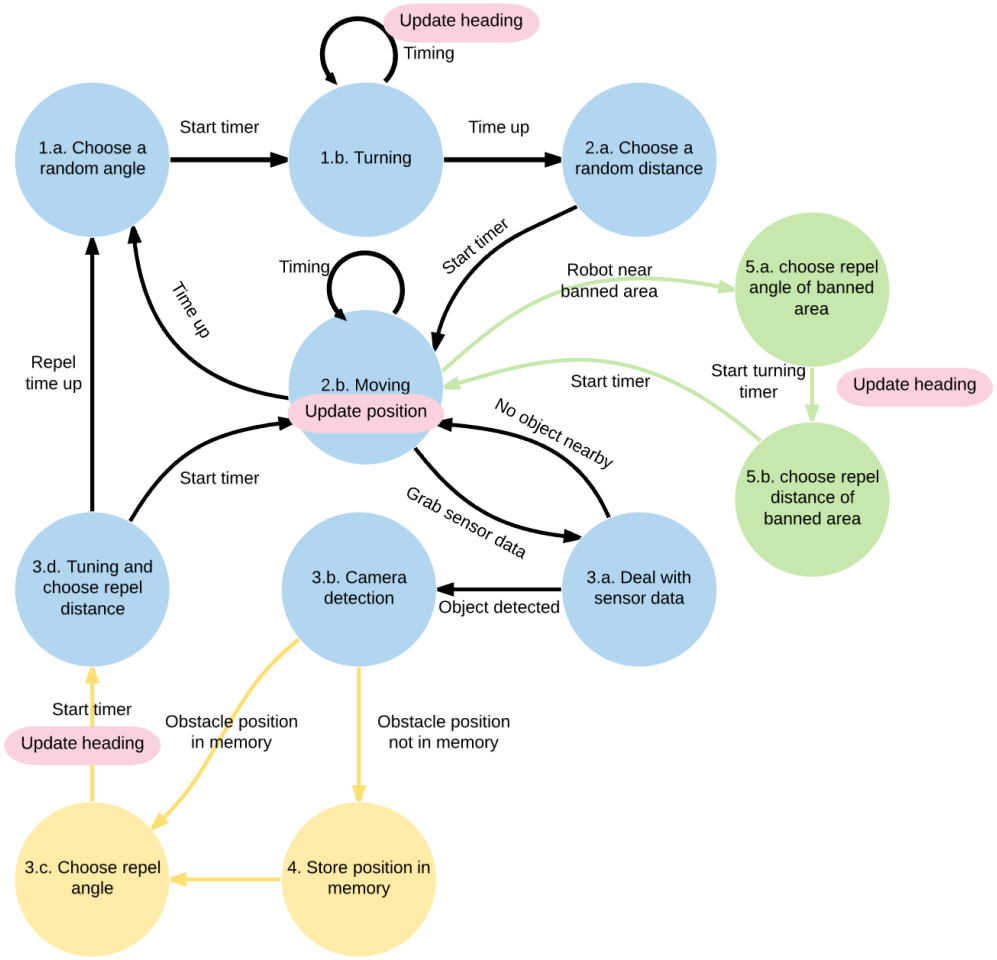

Figure 3-7 Further upgraded FSM with memory and banned area avoidance

State 4. Store position in memory: Positions are updated after every success move without any obstacle detected or banned area nearby. Current headings of e-puck robots are updated after every successful degree it turns. When the robot is turning left, 1 degree will be added to the current heading every 10ms. 1 degree will be subtracted from current heading if the robot is turning right. The current heading is wrapped in the interval $[-180^{\circ}, 180^{\circ}]$. Current position is calculated every 10ms by adding $\cos(\text{current heading})$ to x axis and $\sin(\text{current heading})$ to y axis. The origin point for every robot is different, which is the position where each robot stays before any movement.

In Figure 3-7, the yellow part shows the additional function of collision position storage. The pink bubbles show where an update on current heading and current coordinates of robots should be addressed. The storage format for every collision point contains four elements: coordinate, type (boundary or robot), escape flag and time countdown. The coordinate of collision is defined to be:

$$\mathrm{x} = \text{current x} + (\text{detect range} + \text{robot radius}) * \sin(\text{current heading})$$

$$y = \text{current y} + (\text{detect range} + \text{robot radius}) * \cos(\text{current heading})$$

The escape flag is set to be 1 once the robot has escaped the impact range of collision point for the first time. The time countdown shows how many times left for the collision point to remain in the robot's memory. When time countdown equals to zero, record of this collision point will be removed. After the camera detection has been applied, the memory list will be checked to see if the current collision point is within the impact range of any existing records. If the collision point is found to be new, it will be added to the memory list. Otherwise, the collision point will be ignored and the robot will continue its behaviour by choosing a repel angle.

State 5.a Choose repel angle of banned area: When an e-puck robot is moving, it will take the action of detection and repelling of actual obstacle as its priority. If there is no obstacle around, the robot will consider the distance to the collision points in its memory. Only those collision points with escape flag equals to 1 will be avoid. Otherwise, the robot will go in circles all time. The method chosen for avoiding banned area is stated before in Figure 2-2, the larger the included angle between the robot heading and the obstacle direction, the more the robot will turn to avoid the area. As stated in the description of state 3.b, this repelling method can waste time in terms of repeating the process of turning to obstacle direction. Since there is no still obstacle in the banned area, sensors will not find any obstacle after the robot has turned to the repel angle. This approach is suitable for repelling the banned area.

State 5.b Choose repel distance of banned area: The repel distance is chosen to be the value of impact range. Another way is to use the value of distance left when the robot ran into the banned area. However, this approach can cause redundancy repelling when the distance left is too small. It is better to use longer distance since we need to let the robot escape banned area as soon as possible. When a timer has been started for robot to move forward, the state will change to 2.b. The system will always check obstacles and banned area nearby when the robot is moving forward.

3.4 Conclusion

So far, this section has presented an approach of random walk algorithm with obstacle avoidance and memory repelling. The simple random walk algorithm tends to choose a random angle and a random distance at the beginning to kick start the robot. If no obstacle detected before the robot has moved to expected position, another loop will begin by choosing a random angle. Once the robot has run into an obstacle, it will turn to the direction of obstacle. The front camera will be used to distinguish obstacle between boundary and robot. The collision point will then be stored in the robot's memory with information on coordinates, type (boundary or robot), escape flag and time countdown. There is no time

countdown for collision points with boundary. The escape flag is set to 1 if the robot has escaped the impact range of the collision point. Meanwhile the robot memory will begin to countdown time. If time countdown reaches 0, information of this collision point will disappear in the memory. The obstacle nearby is the first priority to repel when the robot is moving forward. If no obstacle is detected, the robot calculates distance between itself and all the memory position of passed robots. If the robot find itself is within or on the edge of a memory position’s impact range, it will generate an angle to repel.

4 Results and Analysis

The experiment consists of two parts. The part concerning the robot behaviour verifies the efficiency and correctness of robots' behaviours. Behaviour of single robot is demonstrated by graph and the boundary points detected by the robot are pointed out. Two different repelling methods are applied on conditions of single robot and two robots, in order to see which repelling method is more efficient. At the end of first part, five robots are put in the arena with the purpose of measuring the relationship between memory time and coverage time. For the second part, the reliability test concerning the misrecognition of boundary has been conducted with different values of luminance. The error in self-scale with respect to different numbers of robots in the arena is then calculated and analysed. After that, the time for escaping a dead end using different repelling method is tested.

4.1 Robot behaviour with different memory time and repel methods

4.1.1 Single robot with two different repel methods

The arena size in this experiment is 0.75m x 1m. The theory of two repelling method area described in Figure 2-2 and Figure 2-4. The turning back method has a property of quick turning. Since it makes robot turn to the opposite direction of the obstacle and generate a random range between -30 to 30 degrees, it is easy for the robot to start escaping the obstacle immediately after tuning. As long as the two sensors at the back of robot doesn't work or sensors at -90 and 90 of robot's heading do not return any value that smaller than 200, the robot can start moving after turning. Another approach as stated in the previous section can waste time. The behaviours of single robot using different method to repel boundary areas are shown in Figure 4-1. The blue areas are the places where robots have visited. The red dots are the points that belong to boundary areas detected by robot cameras. We can see from the two figures that there is still one dot at the middle of each figure, meaning a misrecognition has happed. For the condition of single robot, misrecognition can result from the sudden error in sensor data.

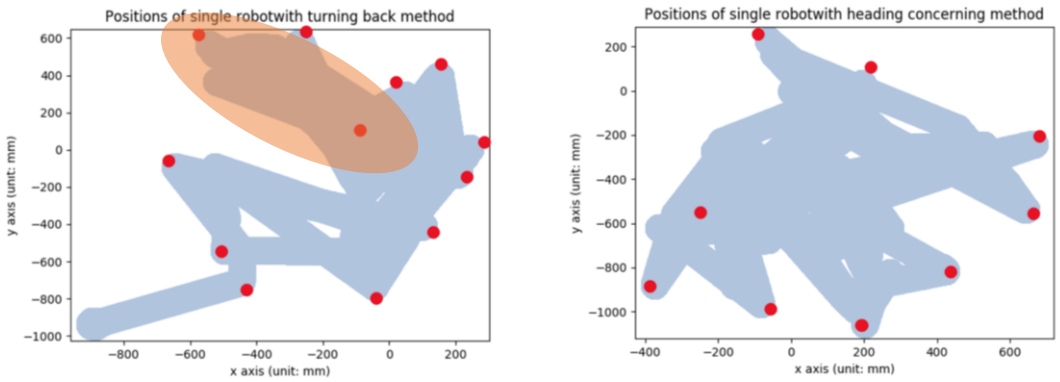

Figure 4-1 Single robot experiment with turning back method (left) and heading concerning method (right)

In Figure 4-1, left figure shows less coverage area than the right figure. This is because the heading concerning method has a larger probability of choosing a smaller repel angle compared to the other method. The orange area on the left shows a typical route repetition problem. The robot can repeatedly pass by these route with hitting the two opposite sides of walls and repelling against them.

4.1.2 Two robots with different repel methods

Two robots are put in an arena with size of 0.75m x 1m. Figure 4-2 shows behaviour of two robots moving together with different repelling size. Several red dots show that are still misrecognition in the middle of each figure. Since this time two robots are in the arena, the misrecognition can be caused by cameras. When the two robots are close to each other, the luminance is not enough for front camera to get a correct data. Therefore, some collision points for meeting other robots are recognized as the boundary area incorrectly.

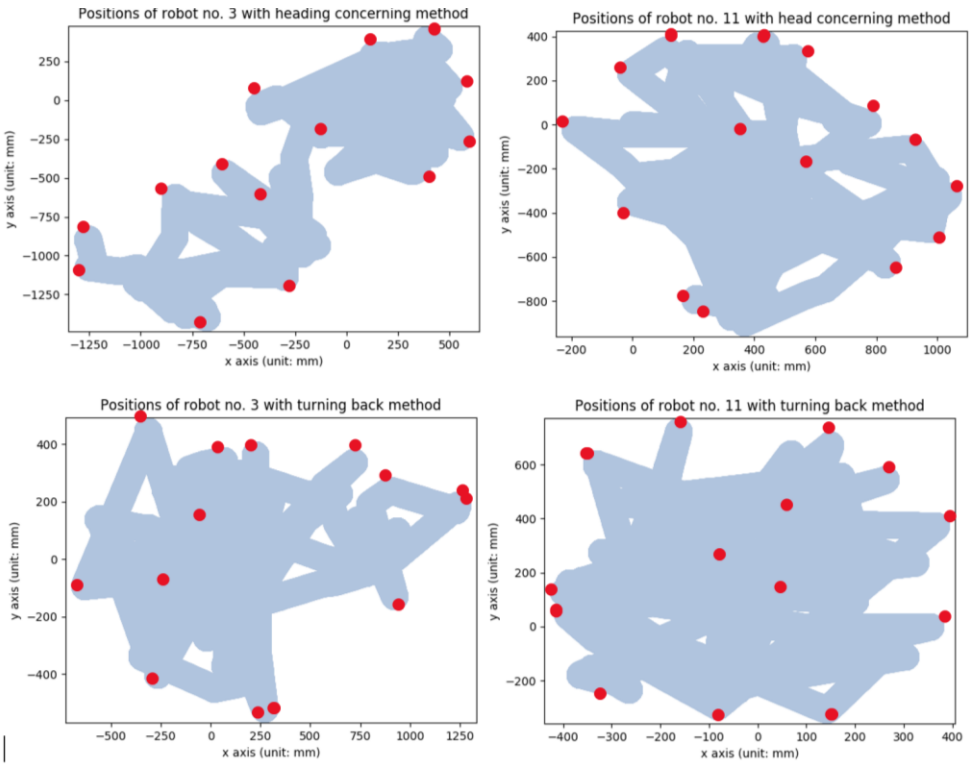

Figure 4-2 Two robots experiment with different repelling methods

As is seen from figures above, the turning back method has more coverage than the heading concerning method. The addition robot should have given the other robot more random repelling. When more than one robot is put in the arena, the turning back repelling method will give more coverage during the same time.

4.1.3 Five robots with different memory time and impact range

The Five robots are put in the arena of 1.5m x 1m. The data below in Table 4-1 shows the time when the coverage area has reached 95% of the arena.

Table 4-1 Coverage time recorded with different impact range and memory time

Memory time Impact range	1min	2 min	3 min
100 mm	423s	492s	523s
150 mm	485s	593s	924s
200 mm	565s	835s	648s

As Luca's paper (Giuggioli, et al., 2016) stated, the coverage time is in proportional of memory time and impact range. Most of the data in the table above fits in this conclusion. Only the match of 2min memory time with 200mm impact range and the match of 3min and

150 do not fit in. This is caused by long time stuck of the robot around the corner area. The reasons of the stuck condition may exist in sensors repeatedly returning data that indicates an obstacle is around, or robot is stuck between the boundary and the banned area.

4.2 Reliability Test

4.2.1 Misrecognition of boundary

Ten experiments are taken under condition of luminance of 310lux and 147lux to find the error rate of misrecognition. One robot is performing normal area coverage task during a time limit of 8min.

Table 4-2 Misrecognition condition of boundary area under different luminance

Experiment no.	Misrecognition under 310 lux	Misrecognition under 147 lux
1	3/15 times = 20%	11/20 times = 55%
2	8/20 times = 40%	14/26 times =55%
3	5/17 times =22%	10/27 times =33%
4	4/18 times =22%	5/18 times =27%
5	7/22 times =31%	15/28 times =53%

The table shows that insufficient luminance can increase error rate of misrecognition on boundary areas. The worst condition can lead up to 55% error rate, which can have a big effect on the result of area coverage task.

4.2.2 Error in self-scale

Due to the firmware problem, the e-puck robot is not able to know the number of steps travelled when the robot is moving. Therefore, time step of 10ms is used to quantize distance travelled and degrees turned. The sense of time is different between the system and the actual world. Error can be caused due to many reasons like drifting and weight.

Table 4-3 Error ratio of self-scale

Robot no. in use	Ideal scale (m)	Actual scale (m)	Error ratio
1	0.75 x 1	1.26 x 1.45	168% x 145% = 243%
2	0.75 x 1	1.17 x 1.56	156% x 156% = 243%
3	0.75 x 1	1.23 x 1.66	164% x 166% = 272%
4	0.75 x 1	0.95 x 1.62	126% x 162% = 204%
5	0.75 x 1	0.98 x 1.43	130% x 143% = 185%

Table 4-3 shows the error ratio with different number of robots in the arena performing area coverage task. When there are 3 robots doing task together, the error rate has reached its peak 272%, which is measured by the area size in robot's head / actual size.

4.2.3 Dead end escape

The experiment is designed to let a robot walk into a dead end with different values of θ, as seen in Figure 4-3 and Table 4-4. The repelling method used is the turning back method Figure 2-4. When θ is equal to 0 and 10 degrees, the robot will turn back immediately after detecting an obstacle ahead. When θ equals to 20 degrees, the robot is confused twice with the position of the boundary. The robot will then turn twice. When θ equals to 40, the robot will be even more confused, it will try to find an angle that its front and side sensors do not detect any obstacle. The robot will be completely not able to escape the dead end if θ equals to 60.

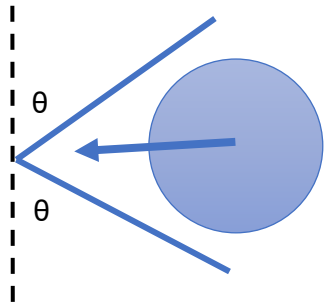

Figure 4-3 Robot is walking towards a dead end

Table 4-4 Time spent on escaping different dead end

θ in degree	0	10	20	40	60
Escape time	5s	5s	35s	62s	NA

5 Discussion

The aim of this project is to apply an area coverage algorithm to e-puck robots, so that the e-puck robots can remember their collision points and repel. A finite state machine with a completely model of expected system has been constructed and implemented.

The camera on the e-puck should be calibrated to recognize the difference between boundary area and other robots it meets. However, the mechanism of the CMOS cameras doesn't allow them to accurately detect colours under conditions of low luminance. A luminance of 147 with error rate of 50% in average has been tested when the robot is performing an area coverage task. When the robot is working in a dark environment, the image frame captured by its camera do not have much difference. Another condition may also affect the working status of camera: the distance between two robots when collision happens. Since the robot should be detected as a small area of red colour, when two robots are very close to each other, a condition of low luminance will happen again. Therefore, it is not very accurate in close image recognition. As matter of fact, the detection of boundary area has just taken use of this special property of the front cameras. As is described in the methodology chapter, if the condition fulfils the detection of a boundary in blue or a detection of a robot in red, the obstacle just nearby is a robot. Otherwise, it is boundary. Close detection of boundary is not recognized as the colour blue.

Positions and headings of the robot itself should be calculated and updated every time step. Since there is no step information of motors returning from the step motor, time is the only measurement to update a position. Due to changes in speed and drifting problem, the result of positions calculated by the system is not accurate as shown in previous chapter. The error ratio is up to 272%. The surface where the robots are moving on is another error source, as well as the weight and the battery type used in the experiment. Different levels of battery left can have significant effects on robots' speed.

Another important objective of this thesis is to ensure the sensors work well on detecting nearby obstacles. The issue of dead end escape has evaluated this function. When the robots reach a place shaped like a corner or a "U", the robot will repeatedly turn in circles near that place. Figure 4-3 shows a typical dead-end escape problem. When the robot turns to an angle and ready to repel, its side sensors will tell the robot there is still an obstacle nearby. Then the robot will get confused by continuously turning to repel angle until its side sensors no longer find any object nearby.

The relationship between impact range, memory time and coverage time is verified in the previous chapter. Impact range and memory time is found to be proportional to coverage time, which is the same as expected in Luca's paper (Giuggioli, et al., 2016).

The main achievement of this thesis is the founding of differences between simulation and experiment results in reality. Experiment results in the real environment can be very different from simulation results. Issues are raised in the aspect of sensor data handling and errors from motors. This type of software on e-puck can be applied on rebuilding the environment, since the e-puck can remember collision points of boundary and remember them in its memory. Other application includes environment monitoring, surveillance and human rescue.

6 Conclusions

The thesis has proposed a way to apply area coverage algorithms on e-puck robots for the purpose of addressing issues concerning about real environment. The function of a single robot has been realised to detect nearby obstacles and remember the collision points. When the robot has detected an obstacle nearby or find it is within the impact range of collision points (only robot), it will turn to a repel angle and escape the area. There is a memory time limit on each memory record stored inside the robot. If the memory time is up, the corresponding collision record will disappear. Only collision points with other robots will be repelled and forgotten. The collision points with boundary will stay in the memory until program ends, in order to rebuild the environment.

The area of swarm robotics has been reviewed in the aspect of definition, developments and research domains. Finite state machines have been designed to help organize the task. Random walk and repelling from area coverage algorithm have been explained in detail. E-puck robot is introduced on hardware structure and software interfaces. After relevant material has been read and discussed, three FSMs that describe functions of e-puck robot from a basic view or Random Walk to a complex view with memory and area avoidance have been proposed. Results of experiment in real environment shows issues concerning the front cameras, proximity sensors, error in step motors and relationship among impact range, memory time and coverage time.

The research has found that it can be difficult to distinguish obstacles between boundary and robot if the luminance of environment is low. Hence this issue can significantly affect the result and performance of the e-puck. Another issue concerning proximity sensors are addressed with smart sensor data handling, since current handling method is easy to be stuck in dead end. Errors in motors can result from drifting or unsmooth surface. These errors will affect the local mapping of the robot and cause scale error. Impact range of a collision point and memory time are found to be in proportional with coverage time.

6.1 Future work

The Vicon system in the arena can be used to record the behaviour on a swarm of e-puck robots. The calibration of every e-puck robot can be completed by comparing the difference in local mapping and global mapping, so that an error distribution can then be generated to guide the local mapping. The dead end escape problem partially results from inaccuracy in local mapping, which is a problem about calibration.An efficient handling approach will be needed to deal with the dead end escape problem, together with calibration. Two side proximity sensors may conditionally detect objects nearby, so as to avoid robots turning in

circles. Most importantly, the firmware on e-puck can be upgraded to return step travelled by wheels from the e-puck mother board. Besides, more trails on finding relationships among impact range, memory time and coverage time can be conducted to form a mathematical model.

7 References

Alami, R. et al., 1998. Multi-robot cooperation in the MARTHA project. *IEEE Robotics \& Automation Magazine,* Volume 5, pp. 36-47.

Alt, W., 1980. Biased random walk models for chemotaxis and related diffusion approximations. *Journal of mathematical biology,* Volume 9, pp. 147-177.

Arkin, R. C., 1998. *Behavior-based robotics.* s.l.:MIT press.

Arvin, F., Samsudin, K. & Ramli, A. R., 2009. Development of a Miniature Robot for SwarmRobotic Application. *International Journal of Computer and Electrical Engineering,* Volume 1, p. 436.

Ballerini, M. et al., 2008. Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. *Proceedings of the national academy of sciences,* Volume 105, pp. 1232-1237.

Benhamou, S., 2007. How many animals really do the Levy walk?. *Ecology,* Volume 88, pp. 1962-1969.

Birk, A. & Belpaeme, T., 1998. A multiagent system based on heterogeneous robots. *Collective Robotics,* pp. 13-24.

Braitenberg, V., 1986. *Vehicles: Experiments in synthetic psychology.* s.l.:MIT press.

Brown, R., 1828. XXVII. A brief account of microscopical observations made in the months of June, July and August 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *Philosophical Magazine Series 2,* Volume 4, pp. 161-173.

Codling, E. A., Plank, M. J. & Benhamou, S., 2008. Random walk models in biology. *Journal of the Royal Society Interface,* Volume 5, pp. 813-834.

Dasgupta, P. & Taylor Whipple, K. C., 2013. Effects of multi-robot team formations on distributed area coverage. In: *Recent Algorithms and Applications in Swarm Intelligence Research.* s.l.:IGI Global, pp. 260-286.

Dorf, R. C. & Nof, S. Y., 1990. *Concise International Encyclopedia of Robotics: Applications and Automation.* s.l.:John Wiley & Sons, Inc..

Dorigo, M. & Şahin, E., 2004. Guest editorial. *Autonomous Robots,* Volume 17, pp. 111-113.

Duchesne, T., Fortin, D. & Rivest, L.-P., 2015. Equivalence between step selection functions and biased correlated random walks for statistical inference on animal movement. *PloS one,* Volume 10, p. e0122947.

Einstein, A., 1905. Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *Annalen der physik,* Volume 322, pp. 549-560.

Feynman, R. P., 1960. There's plenty of room at the bottom. *Engineering and science,* Volume 23, pp. 22-36.

Fukuda, T., Buss, M. & Kawauchi, Y., 1989. *Communication system of cellular robot: Cebot.* s.l., s.n., pp. 634-639.

Garnier, S. et al., 2008. The embodiment of cockroach aggregation behavior in a group of micro-robots. *Artificial life,* Volume 14, pp. 387-408.

Giuggioli, L., Arye, I., Heiblum Robles, A. & Kaminka, G., 2016. From Ants to Birds: A Novel Bio-Inspired Approach to Online Area Coverage. In: *Springer Tracts in Advanced Robotics.* s.l.:Springer.

Giuggioli, L., Potts, J. R., Rubenstein, D. I. & Levin, S. A., 2013. Stigmergy, collective actions, and animal social spacing. *Proceedings of the National Academy of Sciences,* Volume 110, pp. 16904-16909.

Greene, M. J. & Gordon, D. M., 2003. Social insects: cuticular hydrocarbons inform task decisions. *Nature,* Volume 423, pp. 32-32.

Holland, O., 1997. *Grey Walter: the pioneer of real artificial life.* s.l., s.n., pp. 34-44.

Johnson, C. W., 2006. *What are emergent properties and how do they affect the engineering of complex systems?.* s.l.:Elsevier.

Jung, D. & Zelinsky, A., 2000. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots,* Volume 8, pp. 269-292.

Kareiva, P. M. & Shigesada, N., 1983. Analyzing insect movement as a correlated random walk. *Oecologia,* Volume 56, pp. 234-238.

Kong, C. S., Peng, N. A. & Rekleitis, I., 2006. *Distributed coverage with multi-robot system.* s.l., s.n., pp. 2423-2429.

Krieger, M. J. B. & Billeter, J.-B., 2000. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems,* Volume 30, pp. 65-84.

Kuipers, B. & Byun, Y.-T., 1991. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems,* Volume 8, pp. 47-63.

Liu, W. & Winfield, A. F. T., 2011. Open-hardware e-puck linux extension board for experimental swarm robotics research. *Microprocessors and Microsystems,* Volume 35, pp. 60-67.

Lynch, N. A., 1996. *Distributed algorithms.* s.l.:Morgan Kaufmann.

Matarić, M. & Cliff, D., 1996. Challenges in evolving controllers for physical robots. *Robotics and autonomous systems,* Volume 19, pp. 67-83.

Matarić, M. J., 1997. Reinforcement learning in the multi-robot domain. *Autonomous Robots,* Volume 4, pp. 73-83.

McLurkin, J. et al., 2006. *Speaking Swarmish: Human-Robot Interface Design for Large Swarms of Autonomous Mobile Robots..* s.l., s.n., pp. 72-75.

Mondada, F. et al., 2009. *The e-puck, a robot designed for education in engineering.* s.l., s.n., pp. 59-65.

Nolfi, S. & Floreano, D., 2000. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines.* s.l.:MIT press.

Othmer, H. G., Dunbar, S. R. & Alt, W., 1988. Models of dispersal in biological systems. *Journal of mathematical biology,* Volume 26, pp. 263-298.

Parker, L. E., 1994. *ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots.* s.l., s.n., pp. 776-783.

Rubenstein, M. et al., 2014. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems,* Volume 62, pp. 966-975.

SABRA, A. A. K., 2015. *Bio inspired Coverage Control in Robotic Swarm ,* Bristol, UK: s.n.

Shaw, E., 1978. Schooling fishes: the school, a truly egalitarian form of organization in which all members of the group are alike in influence, offers substantial benefits to its participants. *American Scientist,* Volume 66, pp. 166-175.

Siniff, D. B. & Jessen, C. R., 1969. A simulation model of animal movement patterns. *Advances in ecological research,* Volume 6, pp. 185-219.

Turgut, A. E. et al., 2007. Kobot: A mobile robot designed specifically for swarm robotics research. *Middle East Technical University, Ankara, Turkey, METUCENG-TR Tech. Rep,* Volume 5, p. 2007.

Uhlenbeck, G. E. & Ornstein, L. S., 1930. On the theory of the Brownian motion. *Physical review,* Volume 36, p. 823.

Von Smoluchowski, M., 1916. Drei vortrage uber diffusion. Brownsche bewegung und koagulation von kolloidteilchen. *Z. Phys.,* Volume 17, pp. 557-585.

Watson, R. A., Ficici, S. G. & Pollack, J. B., 2002. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems,* Volume 39, pp. 1-18.

8 Appendices

8.1 C++ code on e-puck

```
#ifndef IOSTREAM_H
#include <iostream>
#endif


#ifndef MATH_H
#include <math.h>
#endif

#ifndef CAMERA_H
#include <camera.h>
#endif

#ifndef SPICOMMINTERFACE_H
#include <SPICommInterface.h>
#endif

#ifndef IPCInterface_H
#include <IPCInterface.h>
#endif

#ifndef PATHPLANNER_H
#include <PathPlanner.h>
```

```
#endif

#ifndef TASKALLOCATIONMETHOD_H
#include <TaskAllocationMethod.h>
#endif

#ifndef RTM_H
#include <RTM.h>
#endif

#ifndef ARTM_H
#include <ARTM.h>
#endif

#ifndef FSTREAM
#include <fstream>
#endif

#define PI 3.1415926
#define RHO_WC 0.97 //Wrapped Cauchy concentration
#define LAMDA 15    //Distance select parameter
#define IMPACTRANGE 150
#define ROBOTRADIUS 35
#define BOUNDARYDISTANCE 100
#define MEMORYTIME 30000
#define DETECTRANGE 20
```

```
#include "epuck.hh"

using namespace std;

int g_logtofile(0);
int g_turning_speed(10);

ofstream fout("log.txt");

/***************Global Varibles*****************/
extern int timecount;
extern int sensorcount[8];
extern int userQuit;
extern char * g_host;

/***************Internal Lib*******************/
#ifndef CAMERA_H
#include <camera.h>
#endif

#ifndef SPICOMMINTERFACE_H
#include <SPICommInterface.h>
#endif

#ifndef IPCInterface_H
```

```
#include <IPCInterface.h>
#endif

#ifndef PATHPLANNER_H
#include <PathPlanner.h>
#endif

#ifndef TASKALLOCATIONMETHOD_H
#include <TaskAllocationMethod.h>
#endif

#ifndef RTM_H
#include <RTM.h>
#endif

#ifndef ARTM_H
#include <ARTM.h>
#endif


/** Main Robot's class constructor
   \param str Robot's identifier */

Robot::Robot(const char *str) :

  //Avoid Obstacles Constant Arrays
```

```
  //avoid_weightleft {-10,-10,-5, 0, 0, 5, 10, 10},
  //avoid_weightright {10, 10, 5, 0, 0, -5, -10, -10},

  MAXSPEED(800),
  TURNSPEED(410),
  MINSPEED(10),
  CurrentState(0),
  idletimer(0),
  timer(0),
  PreviousState(0),
  currentheading(0),
  turndirection(0),
  currentpos{0,0},
  distanceleft(0),
  sensorcount(0),
  sensorflag(-1)

  //MOVESPEED(500), //Speed at which epuck travel 10cm in 1 sec
                //when timer updates every 10ms

  //TURNSPEED(500), //Speed at which epuck turn 90 degrees in 900ms
                //when timer updates every 10ms

{
  // Initialization for rand()
  // functions
```

```
  srand(time(NULL));

  //Task Allocation Method Initialization
  TaskAllocation = new ARTM(this);
  CurrentActionRunning=1;
  FoodCollected=0;

  cout << "Create robot "<< endl;
  Name = str;

  // Create Robot's Devices
  EmbeddedCamera = new Camera(320,240);
  SPI = new SPICommInterface();
  //IPCROSMONITOR = new IPCInterface("192.168.20.199",10000);
  //IPCROSVICON = new IPCInterface("192.168.20.199",12000);
  //IPCROSNEST = new IPCInterface("192.168.20.199",14000);
  //PATHPLANNER = new PathPlanner(IPCROSVICON);

  // Camera Related Variables
  CameraMargin = 20;
  CameraWidthCenter = EmbeddedCamera->GetCameraWidth()/2;

  // Initialization of Robot Velocities
  this->RightWheelVelocity = 0;
  this->LeftWheelVelocity = 0;
}
```

```
/** Robot's class destructor */
Robot::~Robot(){
  cout<< " Destroy robot "<<endl;
  delete TaskAllocation;
  delete EmbeddedCamera;
  delete SPI;
  //delete IPCROSMONITOR;
  //delete IPCROSVICON;
  //delete IPCROSNEST;
  //delete PATHPLANNER;
}

/** Robot's update process returns true after updating robot's state
    \param TimeStamp TimeStamp */
bool Robot::Update(uint64_t TimeStamp)
{
  //cout << "robot update" << endl;
  timestamp = TimeStamp;
  SPI->Run();
  //cout<< timecount <<endl;
  //cout << "current state: "<<CurrentState <<endl;
  StateSwitch();
  updatememory();
  //PrintProximityValues();
  //PrintTACValues();
```

```
  /*if(IsRobotImage()){
    cout << "is robot" << endl;
    cout << endl;
  }
  else
    cout << "not robot"<< endl;
*/


  //PrintBatteryLevel();

    /*if(CaptureBoundryImage())
      cout << "boundry" << endl;
    else
      cout<<"not boudnry"<<endl;*/
 /* memorycell obs1;
  obs1.type = 1;
  memorycell obs2;
  obs2.type = 0;
  memoryvect.push_back(obs1);
  memoryvect.push_back(obs2);
  memoryvect.push_back(obs1);
  memoryvect.push_back(obs2);
  cout << "----------------------current memory---------------------- " << endl;
          for(int i = 0; i<memoryvect.size(); i++){
            cout << i << ". pos["<< memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "] "
```

```
                    << " escapeflag = " << memoryvect[i].escapeflag
                    << " timecountdown = " << memoryvect[i].timecountdown
                    << " type = " << memoryvect[i].type << endl;
            }
  cout << "----------------------current memory---------------------- " << endl;

  for(int i=0;i<memoryvect.size();i++){
    if(memoryvect[i].type == 0)
      memoryvect.erase(memoryvect.begin()+i);
  }

  cout << "----------------------after memory---------------------- " << endl;
          for(int i = 0; i<memoryvect.size(); i++){
            cout << i << ". pos["<< memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "] "
                    << " escapeflag = " << memoryvect[i].escapeflag
                    << " timecountdown = " << memoryvect[i].timecountdown
                    << " type = " << memoryvect[i].type << endl;
          }
  cout << "----------------------after memory---------------------- " << endl;
*/

    //IPCROSMONITOR->Send();
    //PrintProximityValues();

  /*
    CurrentActionRunning = TaskAllocation->Action();
```

```
    if(CurrentActionRunning==0)
      RandomWalk();
    else if(CurrentActionRunning==1)
      Wait();
    else if(CurrentActionRunning==2)



    // I AM SURE THERE IS A WAY TO CHANGE THIS
    FoodAtNest = IPCROSNEST->GetA)mountOfFoodAtNest();
  CurrentTheta = TaskAllocation->GetCurrentTheta();
  NValue = TaskAllocation->GetNValue();*/



  //SPI->SetSpeeds(LeftWheelVelocity,RightWheelVelocity);
  SPI->SetSpeeds(0,0);
  //cout << timecount<<": set speed "<< LeftWheelVelocity << " " << RightWheelVelocity
<<endl;
  //GetCurrentPos();
  //GetCurrentHeading();
  return true; // run again
}

/** Robot's Initialization procedure.  It basically creates and
   intializes video and communications objects.  */
bool Robot::Init(){
```

```
  cout << " Init ... " << std::endl;

  // Variable for changing the Random Step
  RandomWalkTimeStep = 0;

  // Initialization of the SPICommInterface
  SPI->Init();

  // Initialization of the Embedded Camera
  EmbeddedCamera->Init();

  // Initialization of the IPC Communication Pipe
  //IPCROSMONITOR->Init(this);
  //IPCROSVICON->Init();
  //IPCROSNEST->Init();

  return true;
}



/** Robot resets its the SPIComm process  */
bool Robot::Reset(){
    SPI->Reset();
    SPI->Run();
```

```
    return true;
}

/** Returns a pointer to Robot's Embedded Camera Device  */
Camera* Robot::GetCameraDevice(){
  return EmbeddedCamera;
}

/** Returns a pointer to Robot's SPIComm Interface  */
SPICommInterface* Robot::GetSPICommInterface(){
  return SPI;
}
/**********************Basic Movement Functions**************/

/** Robot Stops */
bool Robot::Stop(){
    LeftWheelVelocity = 0;
    RightWheelVelocity = 0;
    SPI->Run();
    return true;
}

/** Move the Robot Forward */
void Robot::GoForward(){
  RightWheelVelocity = MAXSPEED;
  LeftWheelVelocity = MAXSPEED;
```

```
}

/** Move the Robot Backwards */
void Robot::GoBackwards(){
  RightWheelVelocity = -MAXSPEED;
  LeftWheelVelocity = -MAXSPEED;
}

/** The Robot Turns Left Constantly */
void Robot::TurnLeft(){
  turndirection = 1;
  RightWheelVelocity = TURNSPEED;
  LeftWheelVelocity = -TURNSPEED;
}

/** The Robot Turns Right Constantly */
void Robot::TurnRight(){
  turndirection = -1;
  RightWheelVelocity = -TURNSPEED;
  LeftWheelVelocity = TURNSPEED;
}

/*********************STATES SUPPORTS*******************************/

// Wrapped Cauchy
int Robot::SelectRandomAngle(){
```

```
  float x = rand()%(1000)/(float)(1001);
  float wc = 2 * atan( (1-RHO_WC)/(1+RHO_WC) * tan( PI * (x-0.5) ) );
  int degree = wc * 180/PI;
  //cout << " random degree "<< degree << endl;
  return degree;
}

float Robot::SelectRandomDistance(){
  float x = rand()%(1000)/(float)(1001);
  float dis = -LAMDA *log(x);
  //cout << " logx " << log(x) <<endl;
  //cout<<" x " <<x<<endl;
  //cout<< " random distance " << dis <<endl;
  // limitations
  if(dis >50)
    dis = 50;
  if (dis < 7)
    dis = 7;
  return dis;
}

void Robot::Turn(int ang){
  timer = abs(ang);
  if(ang > 0){
    TurnLeft();
    CalculateHeading();
```

```
    cout << "turn left " << abs(ang) << " degree "<< endl;
    //cout << endl;
  }
  if(ang < 0){
    TurnRight();
    CalculateHeading();
    cout << "turn right " << abs(ang) << " degree "<< endl;
    //cout << endl;

  }
  if(ang == 0){
    //Stop();
    cout << " no turn -->stop" << endl;
    //cout << endl;
  }
}
void Robot::RandomTurn(){
  CurrentState = -1;
  int ang = SelectRandomAngle();
  cout << " select random angle = " << ang << endl;
  if(abs(ang)<5){
    ang = 0;
    cout << "selcted ang < + - 5, change to: " << ang << endl;
  }
  Turn(ang);
   // will start turn in current time slot
```

```
}

void Robot::RandomMove(){
  CurrentState= 2;
  int dis = SelectRandomDistance();
  cout << " select random distance = " << dis << endl;
  timer = dis * 10;
  //cout << "set move timer = " << timer << endl;
  GoForward();
  CalculatePos();
}

void Robot::idle(){
  timer = 0;
  //cout << "previous state = " << PreviousState << endl;
  switch(PreviousState){
    case 0:{
     // cout << "previous state = " << PreviousState << endl;
      CurrentState = 1;
     // cout << "going to state " << CurrentState << endl;
      break;
    }
    case 2:{
     // cout << "previous state = " << PreviousState << endl;
      CurrentState = 0;
     // cout << "going to state " << CurrentState << endl;
```

```
      break;
    }
    case 3:{
      CurrentState = 4;
    }
  }
  if(timer > 0)
    Stop();
}

void Robot::DetectBoundry(int *sensorID){
  vector<int>* IRSENSORSVECTOR = SPI->GetIRDataVector();
  //cout << " sensor reading [ ";
  int maxvalue = 0;
  for(vector<int>::iterator CurrentIR = IRSENSORSVECTOR->begin();
       CurrentIR != IRSENSORSVECTOR->end();
       ++CurrentIR){

    // find sensor with largest value (closest direction to obstable)
    if(*CurrentIR > maxvalue && *CurrentIR > 200 ){
      maxvalue = *CurrentIR;
      *sensorID =  CurrentIR - IRSENSORSVECTOR->begin();
      //sensorcount[*sensorID]++;
    }
  }
}
```

```
void Robot::BoundryAvoidance(){
  int sensorID = -1;
  DetectBoundry(&sensorID);

  // avoid sudden error in sensor
  if(sensorID != -1 && sensorcount < 4){
    sensorID = -1;
    sensorcount++;
  }
  else
    sensorcount = 0;

  PreviousState = 2;
  sensorflag = sensorID;
  //sensorcount[sensorID]++;
  //cout <<"timecountafter " << timecount << endl;

  switch(sensorID){
    case -1:{
      //PrintProximityValues();
      if(avoidbannedarea()){ // run into banned area
        CurrentState = -1;
        PreviousState = 3;
```

```
  }
  break;

}
case 0:{
  PrintProximityValues();
  cout << "object detected at sensor: " << sensorID << endl;
  //cout << "turnleft 180 degree" << endl;
  //cout << "timecount = " << timecount << endl;
  //distanceleft = timer - timecount;
  //distanceleft = BOUNDARYDISTANCE;
  timer = 0;
  timecount = 0;
  CurrentState = 3;
  break;
}
case 1:{
  PrintProximityValues();
  cout << "object detected at sensor: " << sensorID << endl;
  cout << "turnright 60 degree" << endl;
  //cout << "timecount = " << timecount<< endl;
  TurnRight();
  CalculateHeading();
  //distanceleft = timer - timecount;
  //distanceleft = BOUNDARYDISTANCE;
  timer = 60;
```

```
  timecount = 0;
  CurrentState = 3;
  break;
}
case 2:{
  PrintProximityValues();
  cout << "object detected at sensor: " << sensorID << endl;
  cout << "turnright 90 degree" << endl;
  //cout << "timecount = " << timecount<< endl;
  TurnRight();
  CalculateHeading();
  //distanceleft = timer - timecount;
  //distanceleft = BOUNDARYDISTANCE;
  timer = 90;
  timecount = 0;
  CurrentState = 3;
  break;
}
case 3:break;
case 4:break;
case 5:{
  PrintProximityValues();
  cout << "object detected at sensor: " << sensorID << endl;
  cout << "turnleft 90 degree" << endl;
  //cout << "timecount = " << timecount<< endl;
  TurnLeft();
```

```
  CalculateHeading();
  //distanceleft = timer - timecount;
  //distanceleft = BOUNDARYDISTANCE;
  timer = 90;
  timecount = 0;
  CurrentState = 3;
  break;
}
case 6:{
  PrintProximityValues();
  cout << "object detected at sensor: " << sensorID << endl;
  cout << "turnleft 60 degree" << endl;
  //cout << "timecount = " << timecount<< endl;
  TurnLeft();
  CalculateHeading();
  //distanceleft = timer - timecount;
  //distanceleft = BOUNDARYDISTANCE;
  timer = 60;
  timecount = 0;
  CurrentState = 3;
  break;
}
case 7:{
  PrintProximityValues();
  cout << "object detected at sensor: " << sensorID << endl;
  //cout << "turnright 180 degree" << endl;
```

```
      //distanceleft = timer - timecount;
      //distanceleft = BOUNDARYDISTANCE;
      timer = 0;
      timecount = 0;
      CurrentState = 3;
      break;
    }
    default: break;
  }
  //cout << "current heading: " << currentheading << endl;

}


void Robot::StateSwitch(){
  switch(CurrentState){
    case -1:{
      if(timeup()){
        cout << endl;
        GetCurrentPos();
        GetCurrentHeading();
        cout << "current state: "<< CurrentState << endl;
        idle();
        PreviousState = -1;
        break;
      }
```

```
  else{ //turning
    //cout << "current state" << CurrentState << endl;
   // cout << "state -1 CalculateHeading timecount" << timecount << endl;
    CalculateHeading();
    //GetCurrentHeading();
    //cout << "turndirection" << turndirection<<endl;
    //cout << "timecount" << timecount <<endl;
    break;
  }
}
case 0:{
  if(timeup()){
    cout << endl;
    GetCurrentPos();
    GetCurrentHeading();
    cout << "current state: "<< CurrentState << endl;
    RandomTurn();
    PreviousState = 0;
    break;
  }
  else {
    break;
  }
}
case 1:{
  if(timeup()){
```

```
    cout << endl;
    GetCurrentPos();
    GetCurrentHeading();
    cout << "current state: "<< CurrentState << endl;
    RandomMove();
    PreviousState = 1;
    break;
  }
  else{ //idling
    break;
  }
}
case 2:{
  if(timeup()){ // moved the desired distance
    distanceleft = 0;
    cout << endl;
    GetCurrentPos();
    GetCurrentHeading();
    cout << "current state: "<< CurrentState << endl;
    CurrentState = -1;
    PreviousState = 2;
    break;
  }
  else // still moving
    //cout << "state 2 moving....." <<timecount << endl;
    CalculatePos();
```

```
        BoundryAvoidance();
        //GetCurrentPos();
        break;
    }
    case 3:{ // turn to obstacle and see if it is bourndy or not
      if(timeup()){
        cout << " turned to face obstacles" << endl;
        GetCurrentPos();
        GetCurrentHeading();

        // store obstacle position in a temp variable
        double tempobstacle[2];
        tempobstacle[0] = currentpos[0]+(ROBOTRADIUS*2 +
DETECTRANGE)*cos(currentheading);
        tempobstacle[1] = currentpos[1]+(ROBOTRADIUS*2 +
DETECTRANGE)*sin(currentheading);

        // Camera detection: if is robot or not boundry,add pos, and repel
        if(NotBoundryNotRobot() || IsRobotImage() ){ //if robot or not boundry,add pos, and
repel
          cout << "It was a robot" << endl;

          // If it is in the impact range of ROBOT pos that already in the memory, do not store it
          if(!InMemoryRange(tempobstacle,IMPACTRANGE+ROBOTRADIUS)){
            memorycell obstaclepos;
            obstaclepos.pos[0] = tempobstacle[0];
            obstaclepos.pos[1] = tempobstacle[1];
```

```
    obstaclepos.escapeflag = 0;
    obstaclepos.timecountdown = -1;
    obstaclepos.type = 1; //is robot
    memoryvect.push_back(obstaclepos);
    cout << "----------------------current memory---------------------- " << endl;
    for(int i = 0; i<memoryvect.size(); i++){
      cout << i << ". pos["<< memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "] "
                << " escapeflag = " << memoryvect[i].escapeflag
                << " timecountdown = " << memoryvect[i].timecountdown
                << " type = " << memoryvect[i].type << endl;
    }
    cout << "----------------------current memory---------------------- " << endl;
  }
  else
    cout << "InMemoryRange ROBOT " << endl;

  //repel
  PreviousState = 3;
  CurrentState = -1;
  distanceleft = IMPACTRANGE;
  cout << "Collision Repel"<< endl;
  repelgeneral();
  //repelcollision();
  break;
}
else{ // if it is boundry, remember in boundry list forever
```

```
        cout << "It is boundry" << endl;


        // If it is in the impact range of BOUNDRY pos that already in the memory, do not store
it
        if(!InMemoryRange(tempobstacle,BOUNDARYDISTANCE+ROBOTRADIUS)){
          memorycell obstaclepos;
          obstaclepos.pos[0] = currentpos[0]+(DETECTRANGE +
IMPACTRANGE)*cos(currentheading);
          obstaclepos.pos[1] = currentpos[1]+(DETECTRANGE +
IMPACTRANGE)*sin(currentheading);
          obstaclepos.escapeflag = 0;
          obstaclepos.timecountdown = -1;
          obstaclepos.type = 0; //is boundry
          memoryvect.push_back(obstaclepos);
          cout << "----------------------current memory---------------------- " << endl;
          for(int i = 0; i<memoryvect.size(); i++){
            cout << i << ". pos["<< memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "] "
                 << " escapeflag = " << memoryvect[i].escapeflag
                 << " timecountdown = " << memoryvect[i].timecountdown
                 << " type = " << memoryvect[i].type << endl;
          }
          cout << "----------------------current memory---------------------- " << endl;
        }
        else
          cout << "InMemoryRange BOUNDRY " << endl;


        //repel
```

```
      distanceleft = BOUNDARYDISTANCE;
      PreviousState = 3;
      CurrentState = -1;
      cout << "Boundry Repel" << endl;
      repelgeneral();
      //repelboundry();
      break;
    }
  }
  else{ //turning to front
    CalculateHeading();
    break;
  }
}

case 4:{ // Turned to repel angle and start to move forward
  if(timeup()){
    GoForward();
    CalculatePos();
    timer = BOUNDARYDISTANCE;
    cout << "Turned to repel angle and start to move forward " << timer << endl;
    CurrentState = 2;
    break;
  }
  else
    break;
```

```
    }
  }
}


bool Robot::InMemoryRange(double obstaclepos[2],int range){
  int i = 0;
  for(i = 0; i < memoryvect.size();i++){
    double dis = sqrt(pow( obstaclepos[0] - memoryvect[i].pos[0],2) +
                      pow( obstaclepos[1] - memoryvect[i].pos[1],2));
    if(dis < range){
      return 1;
      break;
    }
  }

  if(i == memoryvect.size())
    return 0;
}

void Robot::updatememory(){
  for(int i = 0; i<memoryvect.size(); i++){
    if(memoryvect[i].escapeflag == 0){
      // if not escaped from the range yet, calculate distance between obstacle and robot
      double dis = sqrt(pow( currentpos[0] - memoryvect[i].pos[0],2) +
                        pow( currentpos[1] - memoryvect[i].pos[1],2));
```

```
if(memoryvect[i].type == 0 && dis > (BOUNDARYDISTANCE + ROBOTRADIUS) ){
// escaped boundary range
  memoryvect[i].escapeflag = 1;
  cout << "----------------------current memory---------------------- " << endl;
    for(int i = 0; i<memoryvect.size(); i++){
      cout << i << ". pos["<< memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "] "
                << " escapeflag = " << memoryvect[i].escapeflag
                << " timecountdown = " << memoryvect[i].timecountdown
                << " type = " << memoryvect[i].type << endl;
    }
  cout << "----------------------current memory---------------------- " << endl;
}
else if(memoryvect[i].type == 1 && dis > (IMPACTRANGE + ROBOTRADIUS) ){
// escaped obstacle impact range and set memory time
  memoryvect[i].escapeflag = 1;
  memoryvect[i].timecountdown = MEMORYTIME;

  cout << "----------------------current memory---------------------- " << endl;
    for(int i = 0; i<memoryvect.size(); i++){
      cout << i << ". pos["<< memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "] "
                << " escapeflag = " << memoryvect[i].escapeflag
                << " timecountdown = " << memoryvect[i].timecountdown
                << " type = " << memoryvect[i].type << endl;
    }
 cout << "----------------------current memory---------------------- " << endl;
}
```

```
    }
    else if(memoryvect[i].escapeflag == 1){
      // if alreday escaped from impact range
      if(memoryvect[i].timecountdown > 0)
      // count down memory time
        memoryvect[i].timecountdown--;
      else if(memoryvect[i].timecountdown == 0)
      // if countdown = 0, erase the memory
        memoryvect.erase(memoryvect.begin()+i);
    }
    else
      cout << "escapeflag fault" << endl;

  }
}

int Robot::wraptoPI(double ang){
  if(fabs(ang) > 180){
    if(ang < 0)
      ang = ang + 360;
    else if (ang > 0)
      ang = ang - 360;
  }
  return ang;
```

```
}

bool Robot::avoidbannedarea(){
  int i = 0;
  for(i = 0; i < memoryvect.size(); i++){

    double dis = sqrt(pow( currentpos[0] - memoryvect[i].pos[0],2) +
                      pow( currentpos[1] - memoryvect[i].pos[1],2));

    // Calculate the angle of memory pos with repect to robot current position
    double tempang = atan( (memoryvect[i].pos[1] - currentpos[1]) /
                           (memoryvect[i].pos[0] - currentpos[0]) ) * 180 / PI;
    int AngObWRTRob = wraptoPI(tempang);
    int ang2repel = 0;
    // if run into memory position for boundry
    if(memoryvect[i].type == 0 && dis <= (BOUNDARYDISTANCE + ROBOTRADIUS)
                               && memoryvect[i].escapeflag == 1
                               && abs(currentheading - AngObWRTRob) < 90){

      distanceleft = BOUNDARYDISTANCE;
      cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
<< endl;
      cout << "distance to boundary listed: " << dis << endl;
      cout << " Avoid banned area of boundary at position ["
          << memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "]" << endl;
      GetCurrentPos();
```

```
      cout << "Boundary range is " << BOUNDARYDISTANCE + ROBOTRADIUS << endl;


      //int AngAvoidOb = RandomAngAve(150,210); // Generate random angle for avoiding
with respect to memory pos
      GetCurrentHeading();
      cout << "Obstacle is at " << AngObWRTRob << " degree of myself" << endl;


      //cout << "tempang " << tempang << endl;
      if(AngObWRTRob > 0)
        ang2repel = 180-2*AngObWRTRob;
      else
        ang2repel = -180 - 2*AngObWRTRob;
      //cout << "About to turn to " << AngAvoidOb << " degree with respect to banned
boundary" << endl;
      //int ang = AngAvoidOb - (AngObWRTRob - currentheading);
      ang2repel = wraptoPI(ang2repel);
      Turn(ang2repel);
      cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
<< endl;
      return true;
      break;
    }
    // if run into memory position for collision
    else if(memoryvect[i].type == 1 && dis <= (IMPACTRANGE + ROBOTRADIUS)
                                && memoryvect[i].escapeflag == 1
                                && abs(currentheading - AngObWRTRob) <= 90){
```

```
    distanceleft = IMPACTRANGE;
    cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++" << endl;
    cout << "distance to robot banned area listed: " << dis << endl;
    cout << " Avoid banned area of other robots at position ["
        << memoryvect[i].pos[0] << "," << memoryvect[i].pos[1] << "]" << endl;
    GetCurrentPos();
    cout << "Robot impact range is " << IMPACTRANGE + ROBOTRADIUS << endl;

    //int AngAvoidOb = RandomAngAve(90,270); // Generate random angle for avoiding with respect to memory pos
    GetCurrentHeading();
    cout << "Obstacle is at " << AngObWRTRob << " degree of myself" << endl;

    if(AngObWRTRob > 0)
      ang2repel = 180-2*AngObWRTRob;
    else
      ang2repel = -180 - 2*AngObWRTRob;
    //cout << "tempang " << tempang << endl;
    //cout << "About to turn to " << AngAvoidOb << " degree with respect to banned boundary" << endl;
    //int ang = AngAvoidOb - (AngObWRTRob - currentheading);
    ang2repel = wraptoPI(ang2repel);
    Turn(ang2repel);
    cout << "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++" << endl;
    return true;
```

```
      break;
    }
  }

  if(i == memoryvect.size())
    return false; // the program didn't find it exceed banned area in memory;
}




int Robot::RandomAngAve(int min, int max ){
  int ang = rand() % ( max - min + 1 ) + min;
  ang = wraptoPI(ang);
  return ang;
}

void Robot::repelboundry(){
  int ang = RandomAngAve(150,210); // output ang between -180 ~ 180 degree
  Turn(ang);
}

void Robot::repelcollision(){
  int ang = RandomAngAve(90,270); // output ang between -180 ~ 180 degree
  Turn(ang);
}
```

```
void Robot::repelgeneral(){
  int ang = 0;
  switch(sensorflag){
    case 0: {ang = -90;break;
    }
    case 1: {ang = -120;break;
    }
    case 2: {ang = -90;break;
    }
    case 5: {ang = 90;break;
    }
    case 6: {ang = 120;break;
    }
    case 7: {ang = 90;break;
    }
    default: break;
  }
  sensorflag = -1;
  Turn(ang);

}

bool Robot::timeup(){
  if(timecount < timer)
    return false;
  else{
```

```
    timecount = 0;
    timer = 0;
    return true ;
  }
}

void Robot::CalculateHeading(){
  currentheading = currentheading + turndirection;
  //cout << "+1 heading" << endl;
  if( currentheading > 180)
    currentheading = currentheading - 360;
  if( currentheading < -180)
    currentheading = currentheading + 360;
}

void Robot::CalculatePos(){
  double xdiff = cos(currentheading*PI/180);
  double ydiff = sin(currentheading*PI/180);
 // cout << "current heading" << currentheading << endl;
  currentpos[0] = currentpos[0] + xdiff;
  //cout << "+1 x"<< xdiff << endl;
  currentpos[1] = currentpos[1] + ydiff;
  //cout << "+1 y"<< ydiff << endl;
  //GetCurrentPos();
}
```

```
void Robot::GetCurrentState(){
  cout << "currentstate: " << CurrentState << endl;

}

void Robot::GetCurrentPos(){
  cout.precision(5);
  cout << "currentpos: [ " << currentpos[0] << "," << currentpos[1] << " ]"<<endl;
}

void Robot::GetCurrentHeading(){
  cout << "currentheading: " << currentheading << endl;
}

bool Robot::IsRobotImage(){
  return EmbeddedCamera->RobotDetected();
}
bool Robot::NotBoundryNotRobot(){
  return EmbeddedCamera->NBoundryNRobot();
}
/*********************Print Sensor Data*******************/

/** Prints IR Sensor Vector */
void Robot::PrintProximityValues(){

  cout << this->timestamp\
```

```
        << " : "\
        << this->Name.c_str()\
        << " IR --[";

  vector<int>* IRSENSORSVECTOR = SPI->GetIRDataVector();
  //vector<int>* IRSENSORSVECTOR = SPI->GetIRFilteredDataVector();

  for (vector<int>::iterator CurrentIR = IRSENSORSVECTOR->begin();\
       CurrentIR != IRSENSORSVECTOR->end();\
       ++CurrentIR)
    printf(" %d", *CurrentIR);

  cout << "]" << endl;
}

/** Prints Battery Level */
void Robot::PrintBatteryLevel(){

  int* BatteryLevel  = SPI->GetBatteryLevel();

  cout << this->timestamp\
       << " : "\
       << this->Name.c_str()\
       << " Bat --[ "\
       << *BatteryLevel\
       << " ] " << endl;
```

```
}

/** Prints current robot's state */
/*void Robot::PrintStatus(){

  printf("%d:%s in state [%s] (%d - %d)\n", \
   this->timestamp,     \
   this->Name.c_str());
}*/

/** Returns the Current Action for the Robot  */
int* Robot::GetCurrentActionRunning(){
  return &CurrentActionRunning;
}

/** Returns the Amount of Food Collected by the Robot  */
double* Robot::GetAmountOfFoodAtNest(){
  return &FoodAtNest;
}




/** Returns the Amount of Food Collected by the Robot  */
int* Robot::GetAmountOfFoodCollected(){
  return &FoodCollected;
}
```

```
bool Robot::RobotIsWithinNest(){
  return PATHPLANNER->RobotIsWithinNest();
}

/** Returns the Current Theta for the Task Allocation Model  */
double* Robot::GetCurrentTheta(){
  return &CurrentTheta;
}

/** Returns the Current N Value for the Task Allocation Model  */
int* Robot::GetNValue(){
  return &NValue;
}

/** Robot's Collect Action */
/*void Robot::CollectBlob(){
  if(ThereIsABlobInFront())
    {
      if(!GrabbedSomething())
            MoveTowardsBlob();
    }
  else if(GrabbedSomething())
    {
      if(!RobotIsWithinNest())
            GoToNest();
```

```
      else
           FoodCollected++;
    }
}*/

/** Robot moves towards a Blob that is seen on the front, the
 robot will move forward until it grabs the blob. */
/*void Robot::MoveTowardsBlob(){

  RetrieveBlobCenter();
  DoBlobAlignment();

  if(IsBlobAligned())
    if(!GrabbedSomething())
      GoForward();
}*/



/** Checks if the Robot Grabbed Something */
bool Robot::GrabbedSomething(){
  vector<int>* IRSENSORSVECTOR = SPI->GetIRDataVector();

  // if(IRSENSORSVECTOR->front()>= 50 && IRSENSORSVECTOR->back()>= 200)
  if(IRSENSORSVECTOR->back()>= 50)
    return true;
  else
```

```
    return false;
}
/*void Robot::SetRandomSpeeds(){

  if(RandomWalkTimeStep==15)
    {
      int range = MAXSPEED - MINSPEED + 1;
      RightWheelVelocity = rand() % range + MINSPEED;

      int range2 = MAXSPEED - MINSPEED + 1;
      LeftWheelVelocity = rand() % range2 + MINSPEED;

      RandomWalkTimeStep = 0;
    }
  else
    RandomWalkTimeStep++;
}
/** Robot performs a RandomWalk with Simple Obstacle Avoidance */
/*void Robot::RandomWalk(){
  SetRandomSpeeds();
  AvoidArenaBoundaries();
  ObstacleAvoidance();
}*/
```

```
/** Robot's Colilsion Avoidance function */

/*void Robot::ObstacleAvoidance(){


    vector<int>* IRSENSORSVECTOR = SPI->GetIRDataVector();


    for (vector<int>::iterator CurrentIR = IRSENSORSVECTOR->begin();\

            CurrentIR != IRSENSORSVECTOR->end();\

            ++CurrentIR)

    {

        LeftWheelVelocity +=\

             avoid_weightleft[distance(IRSENSORSVECTOR->begin(),CurrentIR)] *
(*CurrentIR>>3);

        RightWheelVelocity +=\

             avoid_weightright[distance(IRSENSORSVECTOR->begin(),CurrentIR)] *
(*CurrentIR>>3);

    }


    // ??????

    if(timestamp == 100)

      SPI->SetIRPulse(0x1,true);

}


void Robot::AvoidArenaBoundaries(){


  if(PATHPLANNER->RobotIsOutOfBoundaries())

    if(!PATHPLANNER->NestIsInFront())
```

```
      TurnLeft();
}*/

/** Move the Robot Towards the Nest */
/*void Robot::GoToNest(){
  if(PATHPLANNER->NestIsInFront())
    GoForward();
  else
    DoNestAlignment();
}




void Robot::GetOutOfNest(){
  RightWheelVelocity = -MAXSPEED*2;
  LeftWheelVelocity = -MAXSPEED*2;
}*/



/** Get the Robot into Waiting Mode */
/*void Robot::Wait(){

  if(RobotIsWithinNest())
    GetOutOfNest();
  else
```

```
    if(!PATHPLANNER->NestIsBehind())
      TurnLeft();
    else
      Stop();
}*/



/** Make the Robot Align with the Nest
    Otherwise it will keep on turning until it aligns*/
/*void Robot::DoNestAlignment(){

  if(!PATHPLANNER->NestIsInFront())
    TurnRight();
  else
    Stop();
}*/

/** Returns true if the Robot is aligned with the Blob
    False otherwise */
/*bool Robot::IsBlobAligned(){

  if(BlobCenterX<=0)
    return false;

  if (BlobCenterX >= (CameraWidthCenter - CameraMargin)\
```

```
      && BlobCenterX <= (CameraWidthCenter + CameraMargin))
    return true;
  else
    return false;
}
*/
/** Robot Aligns with the Blob in front of it */
/*void Robot::DoBlobAlignment(){

  if(BlobCenterX<=0)
    return;

  //blob to the left of center
  if(BlobCenterX < (CameraWidthCenter - CameraMargin))
    {
      RightWheelVelocity = 0;
      LeftWheelVelocity = MAXSPEED*10/100;
      return;
    }

  //blob to the right of center
  else if(BlobCenterX > (CameraWidthCenter + CameraMargin))
    {
      RightWheelVelocity = MAXSPEED*10/100;
      LeftWheelVelocity = 0;
      return;
```

```
    }
}*/

/** Robot Aligns with the Blob in front of it */
/*void Robot::DoBlobAlignment(){

  if(BlobCenterX<=0)
    return;

  //blob to the left of center
  if(BlobCenterX < (CameraWidthCenter - CameraMargin))
    {
      RightWheelVelocity = 0;
      LeftWheelVelocity = MAXSPEED*10/100;
      return;
    }

  //blob to the right of center
  else if(BlobCenterX > (CameraWidthCenter + CameraMargin))
    {
      RightWheelVelocity = MAXSPEED*10/100;
      LeftWheelVelocity = 0;
      return;
    }
}*/
```

```
void Robot::RetrieveBlobCenter(){
  BlobCenterX = EmbeddedCamera->GetBiggestBlobCenterX();
  BlobCenterY = EmbeddedCamera->GetBiggestBlobCenterY();
}
/** Returns true if there is a Blob in front of the
    Robot. Returns false otherwise. */
bool Robot::ThereIsABlobInFront(){
  if(EmbeddedCamera->GetNumberOfBlobsDetected()>0)
    return true;
  else
    return false;
}




void Robot::PrintTACValues(){

  vector<int>* TACSENSORSVECTOR = SPI->GetTACDataVector();
}

/** Prints MIC Values Vector */
/*oid Robot::PrintMicValues(){

  cout << this->timestamp\
       << " : "\
       << this->Name.c_str()\
```

```
        << " MIC --[";

  vector<int>* MICROPHONESVECTOR = SPI->GetMicDataVector();

  for (vector<int>::iterator CurrentMIC = MICROPHONESVECTOR->begin();\
       CurrentMIC != MICROPHONESVECTOR->end();\
       ++CurrentMIC)
    printf(" %d", *CurrentMIC);

  cout << "]" << endl;
}*/


/** Prints ACC Values Vector */
void Robot::PrintAccelerometersValues(){

  cout << this->timestamp\
       << " : "\
       << this->Name.c_str()\
       << " ACC --[";

  vector<int>* ACCELEROMETERSVECTOR = SPI->GetAccelerometerDataVector();

  for (vector<int>::iterator CurrentACC = ACCELEROMETERSVECTOR->begin();\
       CurrentACC != ACCELEROMETERSVECTOR->end();\
       ++CurrentACC)
```

```
    printf(" %d", *CurrentACC);


  cout << " ]" << endl;
}




/*bool Robot::StepWalked(int *stepl,int *stepr){
  vector<int>* TACSENSORSVECTOR = SPI->GetTACDataVector();
  //cout << "debug " << "left "<< TACSENSORSVECTOR[0] << "right "<<
TACSENSORSVECTOR[1] << endl;
  vector<int>::iterator TACl = TACSENSORSVECTOR->begin();
  vector<int>::iterator TACr = TACSENSORSVECTOR->begin()+1;
  //int diffl = *TACl - laststepl;
  //int diffr = *TACr - laststepr;
  cout << " currentstepl: " << *TACl << " currentstepr: " << *TACr <<endl;
  //cout << " diffstepl: " << diffl << " diffstepr: " << diffr<< " "<<endl;
  //cout << " laststepl: " << laststepl << " laststepr: " << laststepr<< " "<<endl;
  if( *TACl  >= *stepl && *TACr >= *stepr){
    laststepl = *TACl;
    laststepr = *TACr;
    //cout << " after laststepl: " << laststepl << " laststepr: " << laststepr<< " "<<endl;
    return true;
  }
  else{
    laststepl = *TACl;
```

```
    laststepr = *TACr;
    //cout << " after laststepl: " << laststepl << " laststepr: " << laststepr<< " "<<endl;
    return false;
  }
}*/
```

8.2 Python code on Wrapped Cauchy and Exponential Decay distribution

```
# demo of wrapped cauchy function and exponential decay function

import matplotlib.pyplot as plt
from numpy import *
import math
from pylab import *
import random



# wrapped cauchy function
angle = arange(-180,181,1)
Numcount = zeros(len(angle))
counttotal = 1000
rho = 0.5
i = 0

def find(x, vector):
        for i,element in enumerate(vector):
                if element == x:
                        return i


while i < counttotal:
        ang = 2*arctan( (1-rho)/(1+rho) * tan(pi*(random.uniform(0,1)-0.5) ) )
        degree = int(ang * 180/ pi)
```

```
            #print degree
            no = find(degree,angle)
            #print no
            Numcount[no] = Numcount[no]+1
            #print Numcount[no]
            i=i+1


# exponential decay function
Numcountdecay = zeros(counttotal)
decayconstant = 5
j = 0
while  j < counttotal:
            value = -decayconstant * log(random.uniform(0,1))
            Numcountdecay[jj]= int(value)
            #print value
            j=j+1


plt.hist(Numcountdecay, 20, facecolor='green')
# plt.title("Wrapped Cauchy Distribution with concentration parameter = " + str(rho))
plt.title("Exponential Decay Distribution with decay constant = " + str(decayconstant))
# plt.xlabel("degrees")
# plt.ylabel("number of times")
plt.xlabel("distance chosen")
plt.ylabel("number of times")
# #plt.plot(angle,Numcount)
# plt.bar(left = angle,height = Numcount,width = 0.5,align="center",yerr=0.000001)
plt.show()
```

8.3 Python code of plotting results

```
import matplotlib.pyplot as plt
from numpy import *
import math
from pylab import *
```

```
# load pos data
filename = 'logpos.txt'

posx = []
posy = []
n = 0
with open(filename, 'r') as file_to_read:
  while True:
    lines = file_to_read.readline()
    if not lines:
      break
      pass
    no_tmp, x_tmp, y_tmp = [int(i) for i in lines.split()]
    posx.append(x_tmp)
    posy.append(y_tmp)
    n=n+1
    pass
  coorx = np.array(posx)
  coory = np.array(posy)
  pass


plt.scatter(posx,posy,s =500, marker = 'o',c = 'lightsteelblue')
# load boundary data


filenameb = 'logboundary.txt'
posbx = []
posby = []
tempb = []
```

```
with open(filenameb, 'r') as f:
    first_line = f.readline()
    off = -50
    while True:
        f.seek(off, 2)
        lines = f.readlines()
        if len(lines)>=2:
            last_line = lines[-1]
            break
        off *= 2
    # print 'file' + filenameb + 'first' + first_line
    # print 'file' + filenameb + 'last'+ last_line
tempb = (last_line.split())
posby = tempb[::2]
posbx = tempb[1::2]
del posby[0]
# print tempb
# print posbx
# print posby
plt.title("Positions of robot no. 11 with head concerning method")
plt.xlabel("x axis (unit: mm)")
plt.ylabel("y axis (unit: mm)")
plt.scatter(posbx,posby,c = 'r')
plt.savefig('C:/Users/Feifei Rong/Documents/python/123.png')
plt.show()
```

University of the West of England
BRISTOL

Faculty of Environment & Technology

Faculty Research Ethics Committee (FET FREC)

ETHICAL REVIEW CHECKLIST FOR UNDERGRADUATE AND POSTGRADUATE MODULES

Please provide project details and complete the checklist below.

Project Details:

Module name	**Dissertation(masters)**
Module code	**UFMED4-60-M**
Module leader	
Project Supervisor	**Luca Gioggioli, Alan Winfield**
Proposed project title	**Dynamic coverage of territorial robots in GPS deprivedenvironment**

Applicant Details:

Name of Student	Feifei Rong
Student Number	15042299
Student's email address	Feifei2.rong@live.uwe.ac.uk

	CHECKLIST QUESTIONS	Y/N	Explanation
1.	Does the proposed project involve **human tissue, human participants, environmental damage, the NHS, or data gathered outside the UK**?	N	*If the answer to this is 'N' then no further checks in the list need to be considered.*
2.	Will participants be clearly asked to give consent to take part in the research and informed about how data collected in the research will be used?		
3.	If they choose, can a participant withdraw at any time (prior to a point of "no return" in the use of their data)? Are they told this?		
4.	Are measures in place to provide confidentiality for participants and ensure secure management and disposal of data collected from them?		
5.	Does the study involve people who are particularly vulnerable or unable to give informed consent (eg, children or people with learning difficulties)?		

CHECKLIST QUESTIONS		Y/N	Explanation
6.	Could your research cause stress, physical or psychological harm to anyone, or environmental damage?		
7.	Could any aspects of the research lead to unethical behaviour by participants or researchers (eg, invasion of privacy, deceit, coercion, fraud, abuse)?		
8.	Does the research involve the NHS or collection or storage of human tissue (includes anything containing human cells, such as saliva and urine)?		

Your explanations should indicate briefly for Qs 2-4 how these requirements will be met, and for Qs 5-8 what the pertinent concerns are.

- If Qs 2-4 are answered Yes (Y) and Qs 5-8 are answered No (N), no further reference to the Research Ethics Committee will be required, unless the research plan changes significantly.
- If any of Qs 5-8 are answered Yes (Y), then approval from the Faculty Research Ethics Committee is required *before* the project can start. Approval can take over a month. Please consult with your supervisor about the process.

Your supervisor must check your responses above *before* you submit this form.

Submit this completed form via the *Assignments* area in Blackboard (or elsewhere if so directed by the module leader or your supervisor).

After you have uploaded, your supervisor will confirm that the checklist above has been correctly completed by marking this form as Passed/100% via the *My Grades* link on the Blackboard Welcome tab.

If your submitted answers indicate that further ethical approval is indeed required, then you must also send this completed form to ResearchEthics@uwe.ac.uk

Guidance is available at http://www1.uwe.ac.uk/research/researchethics.
Further guidance can be obtained via the module leader, in the first instance, or the Department's Faculty Research Ethics Committee representatives, including your department's *AHoD for Research*.