CLOUD INFRASTRUCTURE AND MORSE POTENTIAL FUNCTION BASED DECENTRALIZED FORMATION CONTROL FOR ROBOT NETWORKS

Final Report for ECE4095 Project B

prepared by

Erwin Mochtar Wijaya

Supervisor: Dr. Ahmet Sekercioglu

Date of Submission: Wednesday, 22 October 2014

Significant Contributions

I. Created the accurate visible LED localization system based on Tony Grubman’s algorithm.

II. Created the software platform to serve as the Cloud Infrastructure of the laboratory’s experimental testbed.

III. Implemented the random-walk algorithm on the lab’s mobile robots as the baseline test case for the Cloud Infrastructure and localization performance.

IV. Created two independent speed controllers (one for translational movement and the other for rotational movement).

V. Created a decentralized control scheme based on a research study on Morse Potential Function.

VI. Evaluated the performance of the Morse Potential Function based control scheme in terms of scalability and robustness and collected empirical data.

MONASH University

Department of Electrical and Computer Systems Engineering

ECE4095 Final Year Project, Semester 2, 2014

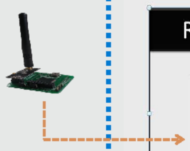
Erwin Mochtar Wijaya

Cloud Infrastructure and Morse Potential Function based Decentralized Formation Control for Robot Networks

Supervisor: Dr. Ahmet Sekercioglu

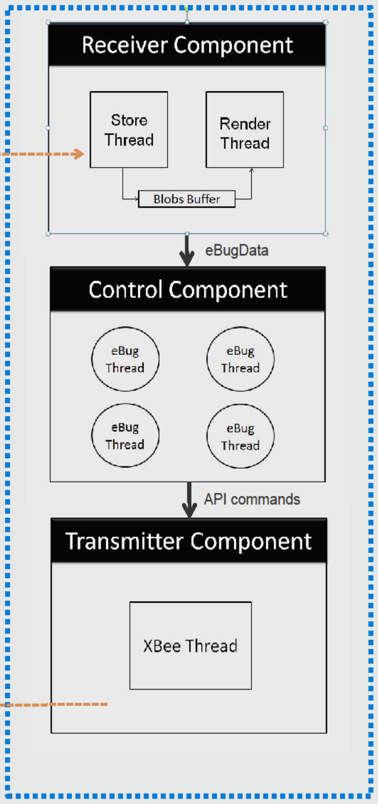
1) PSoC camera tracks the location and colour of LEDs on each eBug. The location and colour of the LEDs are encoded into a 32 bytes packet. This packet is then sent wirelessly through a 2Mb/s NORDIC wireless connection.
➔ *no frame transmission*

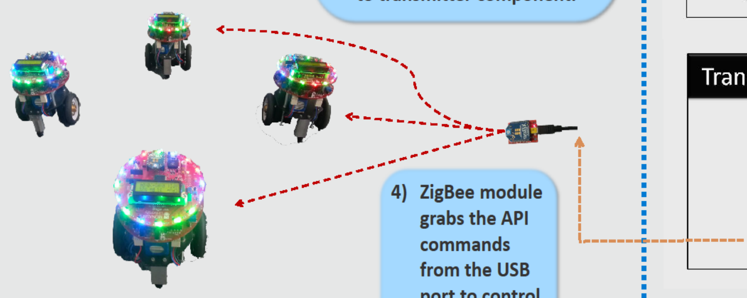
2) The information is received by the receiver and passed to the central base via a USB port.



3) Receiver component converts blobs information into ebugData. eBugData is passed to Control component to perform computation generate API commands which can be translated into eBug movements. These API commands are then passed to transmitter component.

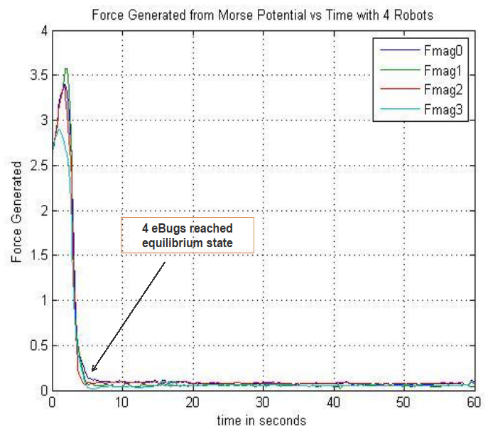
Cloud Architecture





4) ZigBee module grabs the API commands from the USB port to control the eBugs.

EXPERIMENTAL RESULTS



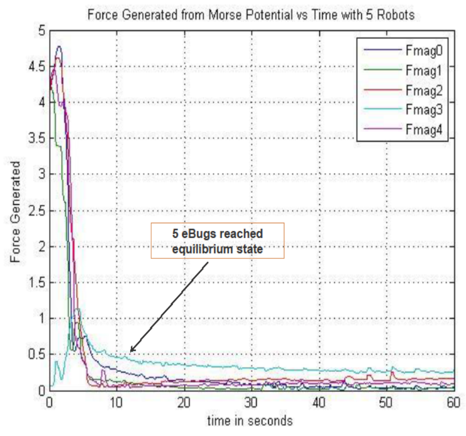


Table of Contents

1 Introduction

1.1 Background

In the past few years, a multi-robot system has become a significant research area in electrical engineering field. This is due to the recent advancement in network robotics as well as increasing demands to address real-world problems such as search and rescue, exploration and surveillance. Clearly, performing a task such as search and rescue with a single robot can be costly and ineffective. A multi-robot system, which refers to a group of autonomous robots performing a task cooperatively, provides higher efficiency in terms of time required to complete the task and robustness to the system should a single robot fail.

One of the main challenges in a multi-robot system is formation control. Formation control, which can be defined as the coordination of multiple robots to form and maintain a certain shape, is significant while solving some of the real-world problems. To illustrate, assigning a team of robot to explore an unknown environment without any formation may result in a team member being separated from the other team members. As the wireless communication module on each robot is limited in terms of its operable range, the missing robot may not be able to regroup once it is outside the wireless communication range.

Apart from that, there are some physical constraints in multi-robot systems. For instance, the on-board CPUs of the team robots are quite limited in terms of their memory space and computing capabilities. This is because CPUs with high computing performance and large memory space are costly in terms of price and energy usage. Hence, installing them on each individual robots may be impractical. Cloud robotics appears as the solution to these problems as it allows offloading high computational tasks from the robots to the cloud.

1.2 Aim

The aim of this project is to implement formation control on a multi-robot system by utilizing a cloud infrastructure, which receives positional information of the robots from the camera. In general, the project can be divided into two stages. The first stage of the project is about designing a cloud infrastructure which performs a random walk with collisions avoidance algorithm on five robots. The next stage focuses on implementing a formation control algorithm on multiple robots which is decentralized, scalable and robust. The general idea of this project is shown in figure 1 below.

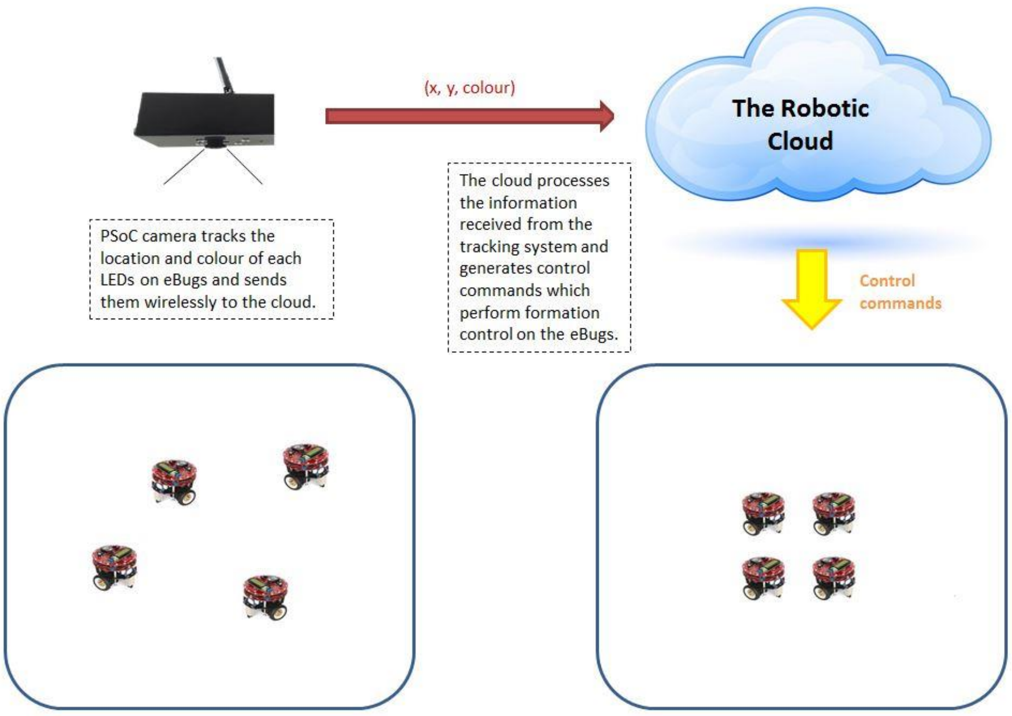

(x, y, colour)
The Robotic Cloud
PSoC camera tracks the location and colour of each LEDs on eBugs and sends them wirelessly to the cloud.
The cloud processes the information received from the tracking system and generates control commands which perform formation control on the eBugs.
Control commands

Figure 1: General overview of the project

1.3 Document Outline

This report presents the design approach on formation control architecture as well as the adopted formation control strategy, which are both built from the literature review on formation control discussed in section three. The hardware specification and software components of the system are presented in section four of the report. Moreover, in section five, cloud infrastructure as the formation control architecture is discussed. Section six presents the random-walk algorithm, which is implemented as the baseline test case for the cloud infrastructure and localization performance. Furthermore, section seven discusses about the proposed decentralised formation control scheme, which is based on Morse potential function. Finally, the performance of the formation control scheme is evaluated in section nine while section ten concludes the whole project.

2 System Requirements

The following are some of the requirements which have been modified from the previous semester requirements specification due to some changes in the project focus.

2.1 Research Requirements

ID	Requirement Description
[CS.01]	Write a literature review on decentralized formation control for robot networks.

2.2 Functional Requirements

ID	Requirement Description
[CS.02]	The eBugs which are within the camera's view boundary should be shown on the screen continuously based on the information regarding their locations and IDs received from the overhead cameras.
[CS.03]	The locations and movements of each eBugs shall be monitored and controlled to ensure that the eBugs never exit the boundary.
[CS.04]	Rendering the image of eBugs on the arena and sending commands to control every eBug shall be done simultaneously.
[CS.05]	The eBugs must be able to detect if collisions are about to occur.
[CO.01]	Each of the eBugs shall perform random walks around the arena.
[CO.02]	The eBugs should be able to avoid collisions after detecting them.
[CO.03]	The eBugs should be able to converge into a formation shape with arbitrary initial positions allocated to each of them.
[OS.01]	The use of ZigBee channel should be optimized in terms of channels usage and throughput.
[OS.02]	The eBugs should be able to determine the best formation to form in terms of the distances they need to travel and the chance of collisions occuring.
[OO.01]	The collision avoidance and formation control algorithm should be able to work with up to 5 eBugs.
[OO.02]	The eBugs formation is automatically chosen based on the number of eBugs and their starting positions.

3 Literature Review

Formation control has been one of the main research areas within multi-robot systems over the past few years. In general, there are two types of approach used to solve formation control problem, namely centralized control and decentralized control. A centralized control scheme incorporates a central node which collects data from individual robots and plans for the next actions to be performed by those robots. As the central node assigns tasks to individual robots based on complete prior knowledge of each robot's state, the overall system's behaviour of a centralized control scheme is predictable [1].

Whilst several studies [2]-[6] in the past proposed a centralized control scheme to address formation control problem, recent research on formation control has focused on inventing a decentralized control scheme [7]-[18]. This trend is mainly due to some technical challenges imposed by a centralized control scheme. For instance, the central node may not be able to obtain all data containing information about state of the individual robots due to the limited communication range. This is a non-trivial problem as the central node plans each robot's action according to their states. In this case, two of the key challenges in centralized formation control are considered, namely scalability and robustness.

In a multi-robot system, scalability refers to performance of the overall system as well as productivity of individual robots with respect to the size of the team [5]. One issue related to the system's performance is communication delay. Research suggests that increasing the number of robots incurs large communication delay [5], [8]. This finding is sensible since all the robots are controlled by a single central node which is only capable of sending commands to one robot at a time. Apart from that, several studies [1, 5, 16] postulate that the system's computational cost is dependent on the number of robots. As all the computation is performed by the central node alone, increasing the number of robots will increase the computational cost. Thus, the postulation seems justified.

Furthermore, system's robustness is also a significant aspect in formation control [8]. In the context of formation control, robustness can be defined as the capability of a system to maintain robots formation when it is subjected to interference, such as agent failures. Agent failures refer to one or more robots not performing the tasks assigned to them [7], [11]. Moreover, it is argued that a centralized control scheme does not provide robustness against agent failures [1]. In multi-robot systems, it is common to elect the robot which is closest to a base station as the central node instead of assigning the base station as the central node itself due to the fact that the base station may not be able to reach all the robots. In that case, it is clear that the whole system will fail once the closest robot to the base station fails to operate. Hence, the argument in [1, 2] can be justified.

Apart from that, numerous approaches to implement decentralized formation control have been proposed in recent years [7]-[18]. These include the use of graph theory, virtual structures, consensus algorithm, market-based algorithm and Lyapunov control theory. Between the aforementioned techniques, there are some similarities. One of them is the fact that there is no central node in the system, as opposed to a centralized control scheme. In addition, all the control schemes in decentralized formation control is based on local information. This means that each robot only needs to know the location of other robots relative to itself while the states of other robots are not required to perform the control algorithm. In this literature review, only some of the approaches to decentralized formation control are discussed.

A study in one-leader constraint formation control has shown that a non-rigid formation control graph can be produced by simplifying the relationships between local leader-follower through adjacency and parameter matrices [12]. However, as the formation moves, this method of non-rigid formation control does not preserve the formation configuration [12]. Apart from that, a decentralized formation control scheme that does not require inter-robot communication has been proposed in [9]. This control scheme, which is based on graph theory, formulates a multi-robot system as a team consisting of one main leader (ML) and multiple sub-leaders (SL) where the movement of the team is guided by the main leader. Through simulations, it has been proved that this control scheme is able generate triangular and diamond formations. However, it has been found that the formation shape cannot be maintained when the leader robot disappears from the follower robots' sights [9]. This can happen in cluttered environments where static obstacles and sensor noises are introduced.

Furthermore, an extension to the study conducted in [9] has appeared in [12]. The graph theory based control scheme proposed in [12] formulates the problem by first describing inter-robot relationships using directed graphs, where directed edges go from follower robots to their local leaders or the formation leader. This is shown in figure 2. The study in [12] has shown that three robots can be preserved in a formation while moving through environments with obstacles. This finding is verified through real experiments. However, it should be noted that the control scheme is only tested with three robots. Since some of the parameters in the controllers are determined experimentally, one can observe that increasing the number of robots may change the constant parameters. Hence, scalability seems to be a major concern in [12].

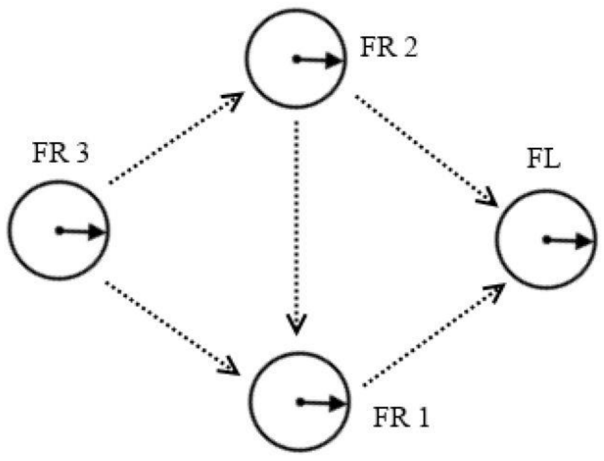


Figure 2: Illustrating inter-robot relationships with directed graphs taken from [12]

In addition to that, a distributed market-based coordination algorithm has been suggested in [11] to implement decentralized formation control. The proposed algorithm is based on the assumptions that each robot knows all the possible tasks which can be assigned to them as well as the maximum number of robots that can be assigned to a given task. In the context of formation control, a task refers to a predefined location in two-dimensional or three-dimensional spaces. Thus, to form a formation, each robot will bid for a unique point on the arena. It should also be emphasized that the task assignment, which is achieved through local auctions between neighbouring robots, may take several iterations before each robot can be assigned to a unique location.

Moreover, the performance of the algorithm in [11] has been evaluated through both simulation and experimentation. It has been found that each robot is assigned to a unique task after one iteration when there is no limitation in network connectivity. On the other hand, when the network connectivity is limited, some of the robots are assigned to the same task for several iterations. It should be emphasized that sharing the same assignment can be a crucial problem in formation control since the assignment refers to a unique point on the arena. As physical robots have certain diameters and lengths, the robots which share the same task may collide before they are being allocated to unique tasks. Apart from that, it should also be noted that this algorithm still requires designing control laws which is able to solve inter-robot collisions and way-point navigation. The study conducted in [11] uses the control laws which are suggested in [7]. Since the proposed control laws in [7] are only tested through simulation of four robots, the scalability of the system is yet to be determined.

Finally, a decentralized formation control scheme based on Morse potential function has been proposed in [18]. This control scheme is based on the creation of attractive and repulsive forces between robots. The forces are generated from the negative gradient of the Morse potential function, which is calculated, based on the distances between robots. The proposed solution seems to perform well with up to one hundred robots reaching a stable formation. However, it has only been tested through simulations without taking into account some practical considerations, such as sensor noises and speed limits of robots. Also, the simulation is conducted by treating each robot as a point agent, which may have reduced the likelihood of collisions between robots. Therefore, it is clear that there is a need to conduct further research on formation control based on Morse potential function in order to determine the scalability and robustness of the control scheme.

4 Hardware and Software Components

4.1 eBugs

eBugs are the physical robots used to form a multi-robot system for this project. The eBugs, which have been previously designed by Nick D'Ademo, is controlled wirelessly via the ZigBee link as each eBug is equipped with an XBee module. The movement of the eBugs is also controlled with the use of stepper motors. It should also be noted that there are sixteen LEDs on the perimeter of each eBug. This is the most important feature of the eBugs for this project as the eBugs are detected by tracking the LEDs on top of them. The LEDs on each eBug are assigned a unique colour sequence which indicates the ID of the corresponding eBug.

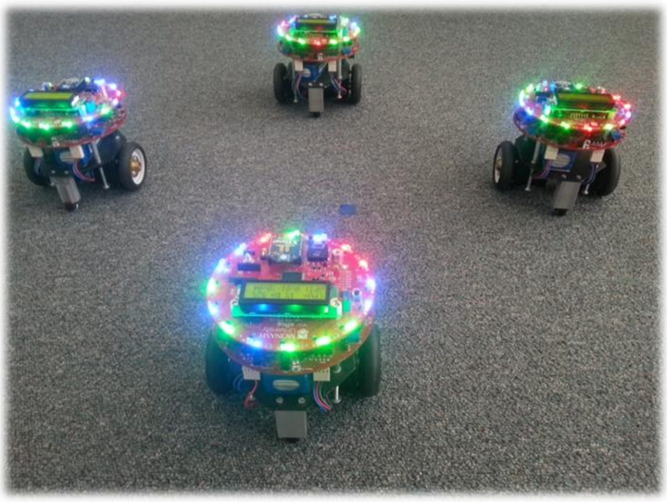


Figure 3: Four eBugs assigned to unique colour sequences

4.2 PSoC Camera

An overhead PSoC camera is used as the main tracking system for the project. It acts like a GPS where the positional information of the eBugs is provided by this camera. However, it should be emphasized here that the PSoC camera only tracks the LEDs on top of each eBug and encodes the information into a packet. This packet is then sent wirelessly to the receiver. Thus, compared to the previous tracking system of the similar project where a webcam camera sends images of eBugs on the arena, this system performs better as there is no frame transmission. The PSoC camera consists of a camera board designed by Leo Xiao Wang, a wireless module and a camera.

Figure 4: PSoC camera used for tracking LEDs on eBugs

4.3 Wireless Receiver Module

The wireless module used at the receiving end is the same as the one installed at the PSoC camera. Also, the board used for the receiver is a PSoC development board designed by Callum Laurenson and Tony Grubman. The main function of this receiver is just to receive and store the blobs information which contains information about the LEDs tracked on the arena.

Figure 5: Wireless receiver module used to receive blobs information

4.4 XBee modules

The XBee module is another significant component in this project. The XBee module provides a wireless communication channel between the robotic cloud and the eBugs. This allows the eBugs to be controlled by the cloud, which acts as the eBugs "brains". Apart from that, the XBee module is mainly used due to its low power consumption. The frequency band of operation of the XBee modules, which operate based on ZigBee protocol, is the ISM 2.4 GHz band. Furthermore, the ZigBee module is also limited by a maximum indoor range of 40 meters in a line-of-sight environment. When used in API modes, the utilization of the ZigBee module only allows a maximum throughput of 35 Kbps.

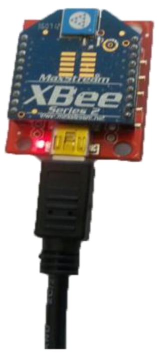


Figure 6: XBee module

4.5 Server Computer

The cloud infrastructure is provided by the server computer in this project. The server computer, which is connected to both the wireless receiver and the XBee module, performs all the required computation and generates control commands to each eBug. The software platform implemented on the server computer is a multi-threaded program, which emulates the on-board processing on each robot. Apart from that, the server computer uses Ubuntu as the main operating system, which is a Linux-based open source operating system. Ubuntu provides some applications and toolkits which help with the development of this project.

4.6 Qt

The software platform implemented on the server computer is called Qt, which is a cross-platform application that uses C++ as the main programming language. The Qt software platform also supports GUI programming. One of the main features in Qt, which is used extensively in this project, is known as SIGNAL and SLOT mechanism. SIGNAL and SLOT mechanism is central when communicating between different objects in Qt.

5 Cloud Architecture

5.1 System Overview

The cloud architecture is implemented using a single Linux machine which runs Qt software as the main platform. The software uses C++ as the main programming language, which adopts an object-oriented programming approach. In general, the system consists of three main software components, namely the receiver, the transmitter and the control unit. Each of this unit consists of multiple threads which run almost in parallel through context switching. The main function of the receiver unit is to process the blobs information on the receiver, which is obtained from the PSoC camera. This information contains the x and y coordinates of the LEDs on top of each eBug as well as their colors. Based on the LEDs information, locations, orientations as well as IDs of each eBug are determined by applying computer vision algorithm which will be explained further in the later section.

Apart from that, the control unit is mainly in charge of planning some tasks which have to be performed by the robots. For our purposes, the tasks performed by each individual robots are either translational movements or rotational movements. It should also be noted that the control unit acts as the “brain” of the robot as it controls all the robots’ actions. Finally, the control commands generated by the control unit are encoded into API packet formats. Furthermore, the transmitter unit, which has been previously designed by Nick D’Ademo, is in charge of sending control commands which have already been converted into API packets. These API packets are sent to individual robots via the ZigBee module. Upon receiving the data, each robot decodes the packet and performs physical actions such as rotating its wheels. The general overview of the system architecture is shown in figure 7.

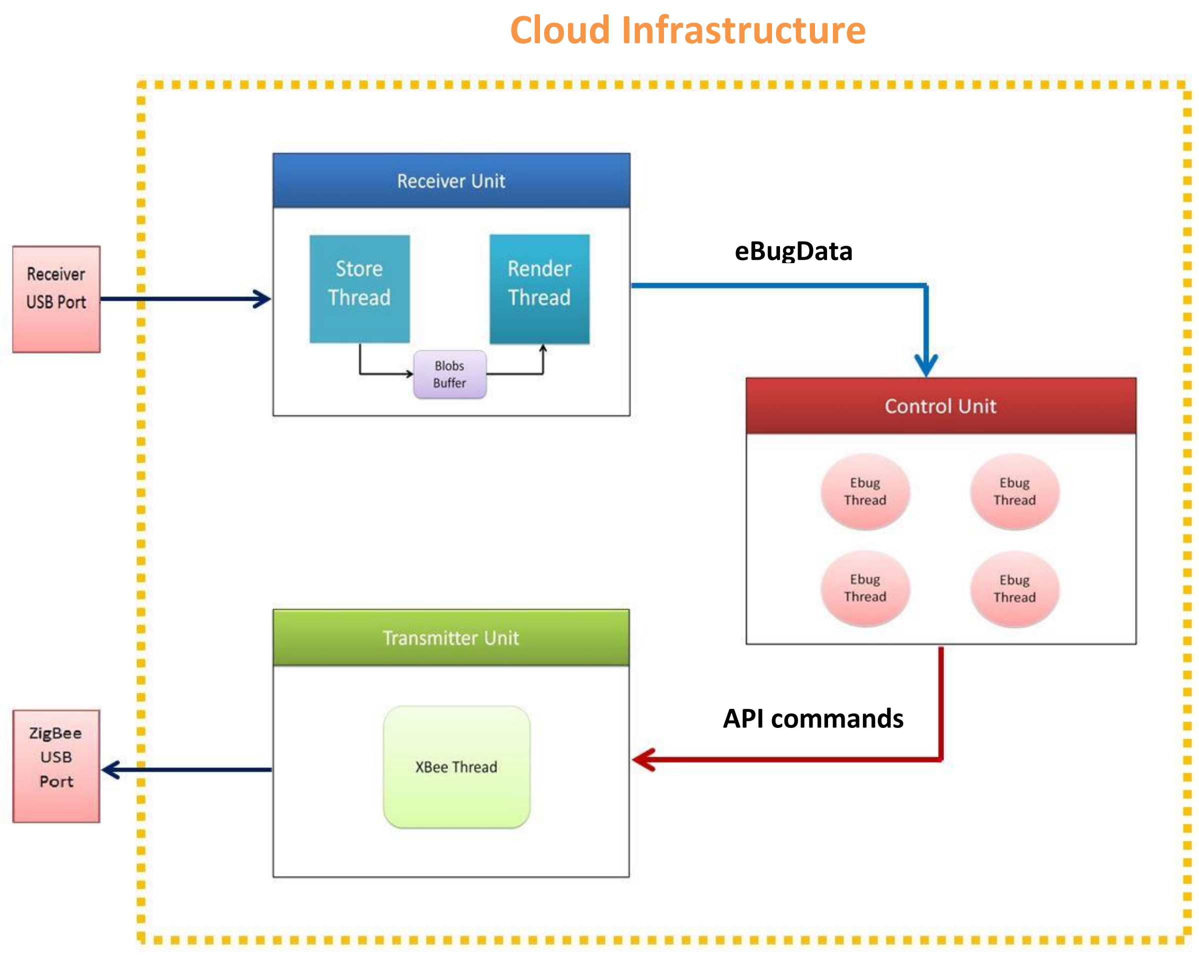


Figure 7: General overview of the Cloud Architecture

From figure three above, it is clear how the cloud architecture works. Initially, the blobs information, which is already being sent by the PSoC camera to the receiver, is grabbed by the receiver unit via a USB port. After that, the receiver unit processes the blobs information and converts it into eBugData which contains robots' positions as well as their IDs. The eBugData is then passed to the Control unit via a SIGNAL and SLOT mechanism, which is one of the main features in Qt platform. Upon receiving the information about the eBugs locations, the control unit generates control commands in API packet formats. After that, the API commands passed to the transmitter unit by storing them into API command buffer, which is accessible by the transmitter unit.

The transmitter unit grabs the commands from the API command buffer and sends them to the ZigBee module via a USB serial port. As the control unit and the transmitter unit are running concurrently, it should be emphasized that a race condition can occur when both the control unit and the transmitter unit are trying to access the buffer at the same time, in which the content of the shared data may alter depending on the order the shared data is accessed. The method used to address this issue is to implement a mutual exclusion (mutex) on the buffer and thus allowing one component to access the shared data at a given time.

5.2 Subsystem Architecture

5.2.1 Receiver Unit

In the receiver unit, there are two main functions which have to be performed. One of them is to grab the blobs information from the receiver via the USB serial port while the other function is to convert the blobs information to eBugData. It is apparent in this case that reading the blobs information may take a much shorter time compared to the conversion process of blobs information into eBugData. This can result in an inefficiency as the receiver unit has to finish converting the blobs information until it can start to receive the new blobs information. Thus, in order to improve the system's efficiency, the receiver unit is divided into two threads which perform the two functions independently, namely the StoreThread and the RenderThread.

The StoreThread in this case is mainly in charge of grabbing the blobs information from the receiver via the USB port. The blobs information is then stored into a buffer called BlobsBuffer. On the other hand, the main function of the RenderThread is to convert the blobs information which is accessible from the BlobsBuffer and render the image of detected eBugs on the arena by applying computer vision algorithm, which is written by Tony Grubman. As both the StoreThread and the RenderThread are running concurrently, BlobsBuffer is protected with mutual exclusion to prevent a race condition.

The receiver unit is illustrated in figure 8.

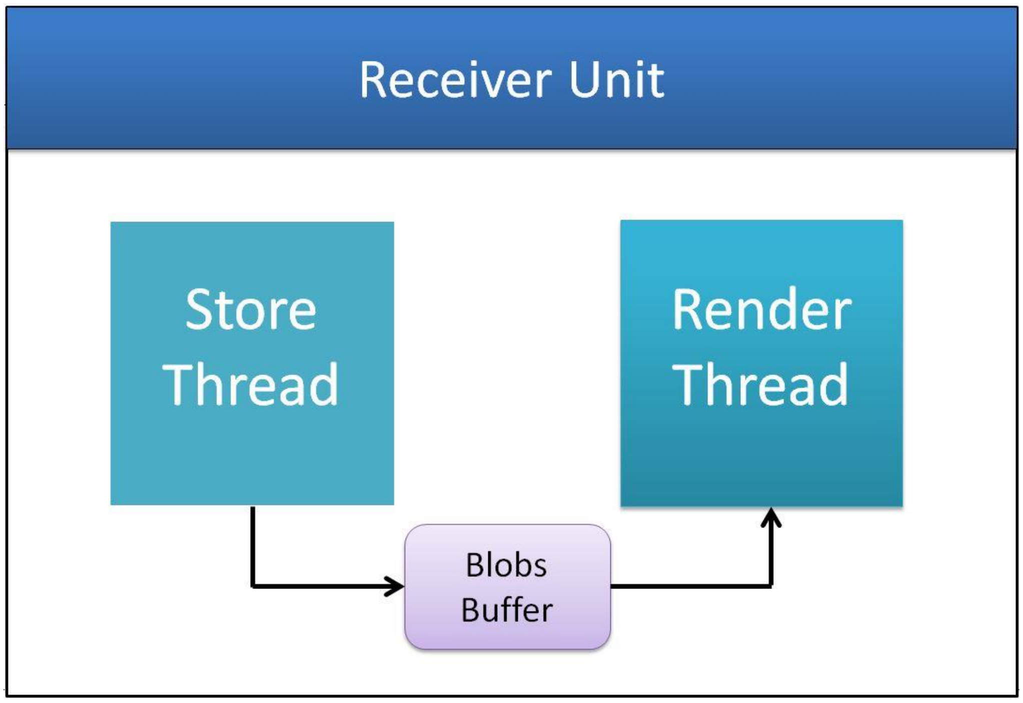


Figure 8: Receiver unit consisting of StoreThread and RenderThread

5.2.2 Control Unit

The main function of the control unit is to plan and control the actions that will be performed by each robot based on the state of the individual robots. It should be noted here that when the system is controlling a large number of eBugs, several problems may arise. One of them is that the time taken for the control unit to process the eBugData and generate control commands increases proportionally with the number of eBugs. To illustrate, when there are twenty eBugs controlled by the control unit, the first eBug that receives a control command has to wait until the control unit finishes processing the positional information of the other nineteen eBugs as well as generating control commands to each of them before it is able to receive the next control command.

Furthermore, as each eBug is controlled sequentially by the control unit, the system becomes asynchronous in which different eBugs may respond to changes in the system's environments at different instants of time. Apart from that, since the control architecture is designed to perform a control scheme in a distributed fashion, controlling the eBugs directly from the control unit, which has access to all the eBugs information, is undesirable. Thus, in order to have a control architecture which is scalable, synchronous and decentralized, the control unit is divided into multiple EbugThreads. In this case, each EbugThread acts as the brain of the physical eBug robot and thus the number of EbugThreads created inside the control unit is the same as the number of eBugs.

Moreover, the eBugData, which contains all the positional information about the eBugs, has to be passed onto EbugThreads as all the computation is performed inside the threads. However, before passing eBugData that contains eBugs positional information to each EbugThread, the data is converted from a global coordinate frame as seen by the PSoC to a local coordinate frame. This is done mainly to allow the EbugThread emulates the real physical eBug where each eBug is only aware of the distance and angle of its neighbouring eBugs. Figure 9 and figure 10 illustrate how the robot appears in global coordinate frame and local coordinate frame respectively.

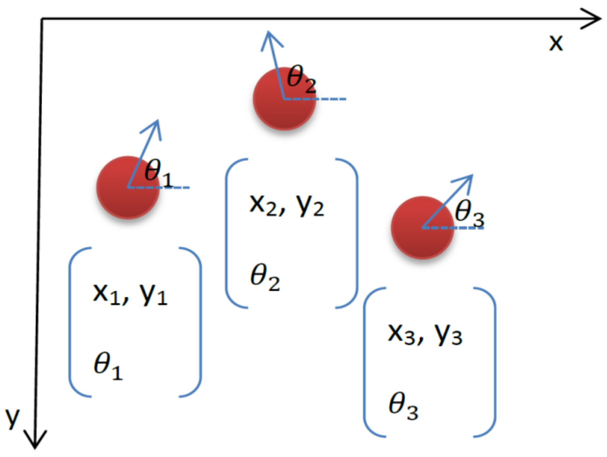

x
y
θ_1
θ_2
θ_3
$\begin{pmatrix} x_1, y_1 \\ \theta_1 \end{pmatrix}$
$\begin{pmatrix} x_2, y_2 \\ \theta_2 \end{pmatrix}$
$\begin{pmatrix} x_3, y_3 \\ \theta_3 \end{pmatrix}$

Figure 9: robots in global coordinate frame

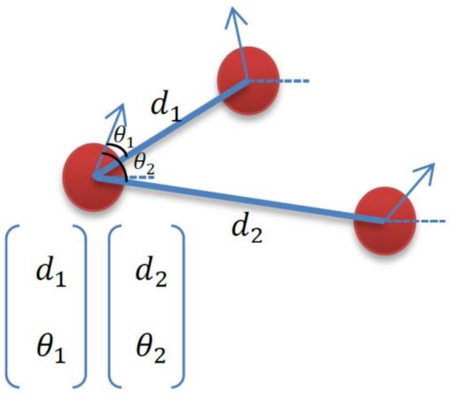

d_1
d_2
θ_1
θ_2
$\begin{pmatrix} d_1 \\ \theta_1 \end{pmatrix}$
$\begin{pmatrix} d_2 \\ \theta_2 \end{pmatrix}$

Figure 10:robots in local coordinate frame

Apart from that, it can be observed that although multiple EbugThreads which run concurrently are created based on the number of eBugs, the system is still not purely synchronous. This is due to the fact that all the generated control commands have to be passed to the transmitter unit, which has a “bottleneck” communication as the transmitter unit is only connected to one XBee module. Hence, although the control commands may have been generated simultaneously by the EbugThreads, the robots may not be able to perform the assigned tasks simultaneously.

It should also be emphasized here that since each EbugThread is sending control commands to the same transmitter unit, a buffer is required due to the fact that the rate at which control commands are generated may be much faster compared to the rate in which the transmitter unit grabs the control commands and sends them to the ZigBee module. This is most likely true when there is a large number of EbugThreads created and therefore resulting in a large number of control commands being generated at every clock cycle. In this case, it seems obvious that the buffer may need to be protected with a mutual exclusion to prevent multiple EbugThreads writing to the buffer simultaneously. The control unit is described clearly in figure 11.

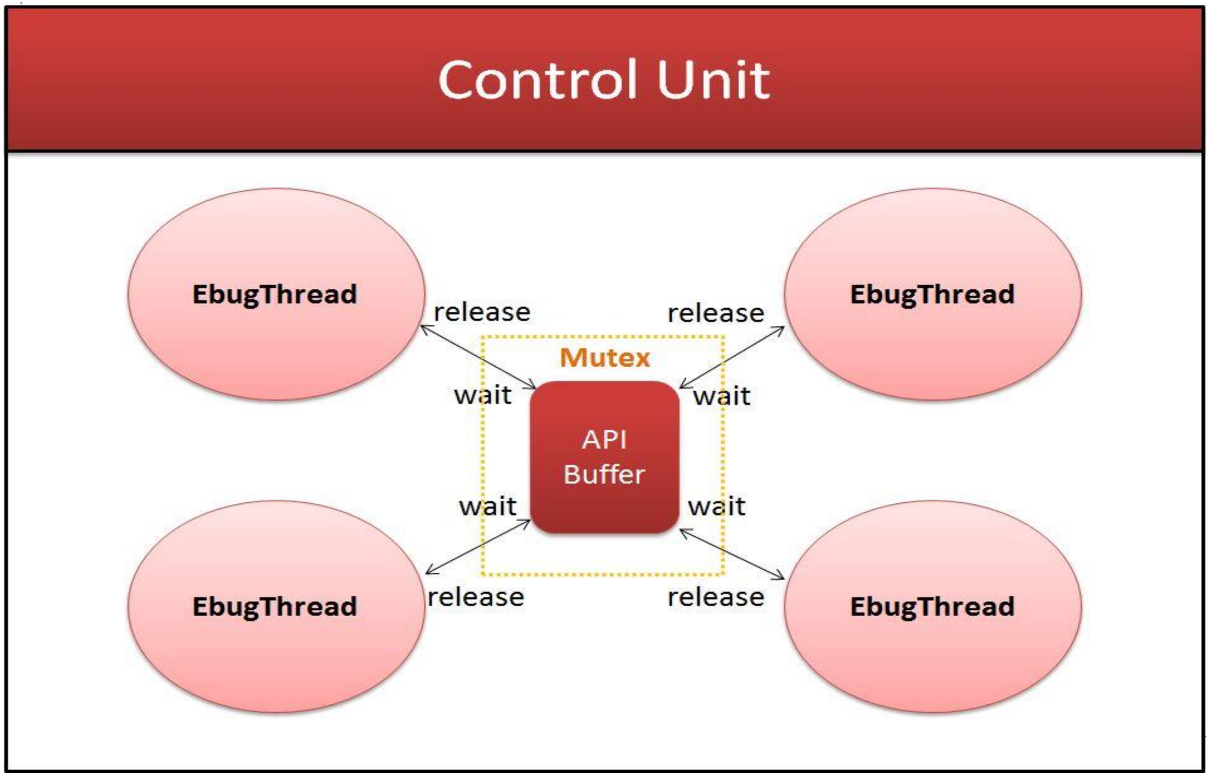


Figure 11: Control unit consisting of multiple EbugThreads

6.2.3 Transmitter Unit

The function of the transmitter is quite simple as it is only in charge of grabbing the control commands which have already been converted into API commands. The transmitter unit only consists a single XbeeThread which checks periodically whether there is data to be sent to the ZigBee module. However, it should also be noted here that the XbeeThread, which accesses the control commands from the API buffer, is blocked when one of the EbugThreads is writing to the buffer as the buffer is protected with a mutex. The transmitter unit is shown in figure 12.

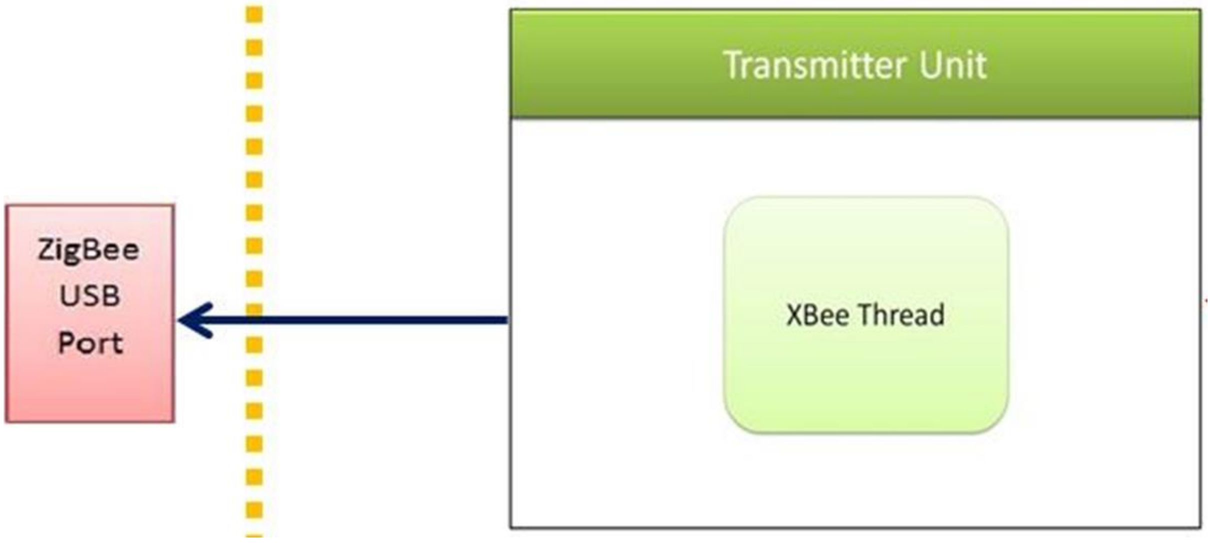


Figure 12: Transmitter unit of the Cloud Architecture

7 Random-Walk

The main purpose of the random-walk algorithm in this case is to verify the cloud architecture proposed in the previous section as well as the visible LED localization algorithm based on Tony Grubman's algorithm. The random-walk algorithm suggested in this section can be considered as distributed, as the same algorithm is being implemented on each EbugThread. Furthermore, the algorithm only requires the distance of neighbouring robots as well as their orientations. These orientations do not refer to the orientations in a global coordinate system seen from the overhead PSoC camera, but instead the orientation in which a robot senses its neighbours.

In the simplest case, where only 1 robot exists on the arena, the random-walk algorithm can be implemented very easily. In this case, the robot is always tasked to move forward for five seconds and rotate for a random amount of time. However, it should be emphasized that the movement of the robot is restricted by a boundary where the area of the boundary is defined by the overhead PSoC camera viewing angle. In this case, the size of the boundary is 640*480 pixels size. Therefore, as robots can only be detected when they are inside the boundary region, the random-walk algorithm has to consider the condition when a robot reaches the boundary.

7.1 Collision Avoidance

It is apparent that there is a possibility of inter-robot collisions when there are two or more robots moving randomly on the arena. Also, with the robot's movements restricted to be inside the boundary, introducing more robots may increase the likelihood of collisions between robots to occur. Therefore, designing a control scheme which prevents collisions between robots is crucial when implementing the random-walk algorithm on two or more eBugs. In this case, the adopted collision avoidance scheme is very simple in which each robot is treated as a point charge that repels its neighbours away.

However, it should be emphasized here that a robot will only move in the opposite direction of its neighbours when their distance is smaller than a certain threshold in order to prevent them being stuck in one region of the arena. Hence, a robot will only consider the artificial forces generated from neighbouring robots which are within its collision proximity. The direction in which the robot moves to avoid collisions can be computed using vector force addition as illustrated in figure 13.

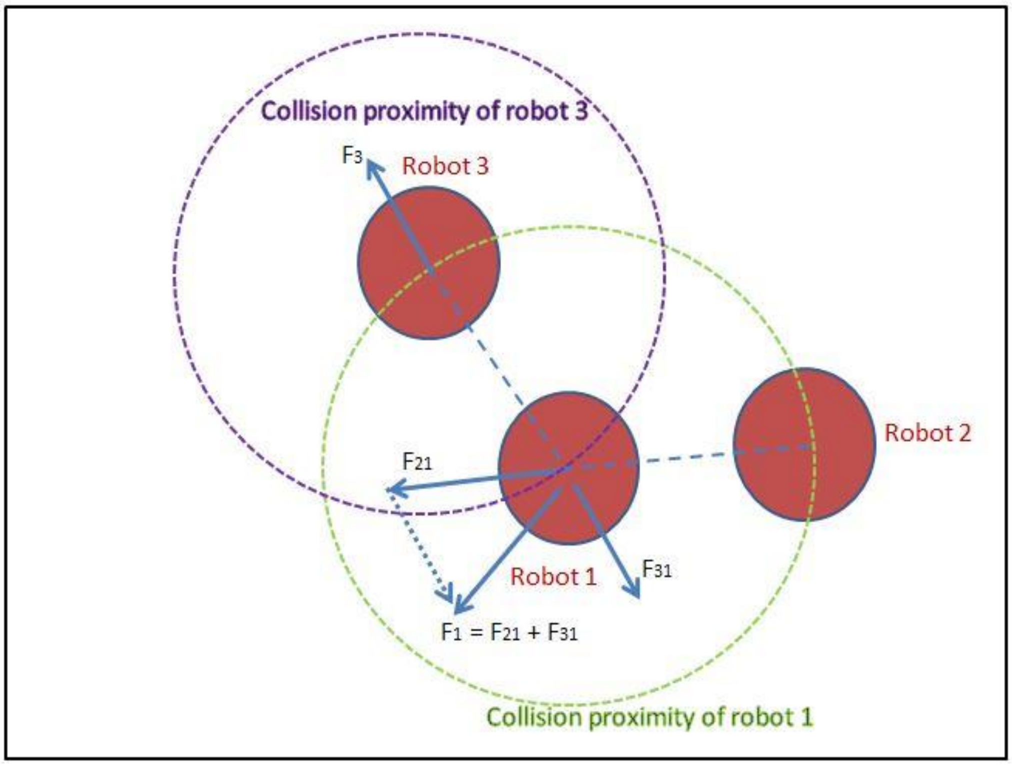


Figure 13: Robot avoids collision by summing artificial vector within its collision proximity

It can be seen from figure 12 that robot 1 is moving away from both robot 3 and robot 2 since they are within its collision proximity. On the other hand, robot 3 only moves away from robot 1 since robot 2 is not within robot 3 collision proximity. Furthermore, it should be noted that this collision avoidance method may not scale well with increasing number of robots as there are several problems which may arise when summing the vector forces, such as aliasing. Therefore, a new conditional loop is introduced in the random-walk algorithm, where a robot stalls its movement if it detects four or more robots within its proximity distance.

7.2 Algorithm

After considering all the possible conditions, the random-walk algorithm can now be implemented inside the EbugThread as follows. Initially, it checks whether the private variable of type *boolean* called eBugtrapped returns a true value. If it does, the EbugThread will generate a control command that will stop the robot's movement. Otherwise, it will jump into the second condition where in this case, another boolean variable named collisionDetected is determined. In the case where the collisionDetected variable returns a true value, the angle error, which is the difference between eBug's orientation and the direction of the summed vector forces, is determined. If this error is larger than 5 degree, control commands to rotate the eBug is generated. Otherwise, this loop will always repeat until the robot is aligned within 5 degree of the summed vector forces.

Furthermore, the distance of the eBug to the boundary is checked where there is no robot within the collision proximity. If the distance is below a certain threshold, the eBug is considered as reached the boundary. In this case, the eBug will be given a new control command only if it is facing the boundary as it will result in eBug exiting the boundary. In the event when the eBug does not detect anything, it will just move forward for five seconds and rotate for a random period to randomize its movement. It should be emphasized here that the order of the conditional loop is crucial. For instance, if the boundary condition is checked before collision state, the robot will never be able to perform collision avoidance once the boundary loop is entered. Hence, the conditional loop should be ordered based on the priority of the robot's tasks. This algorithm is shown in figure 14.

Apart from that, the control unit, which has accesses to all EbugThread, performs supervisory action and changes the state of EbugThread by altering the private variables inside each thread. Changing the state of EbugThread is done through member function calls which is the preferable method since accessing a private variable of an object directly is not desirable from an object-oriented programming point of view. The main function of the algorithm implemented on the control unit is to compute the resultant vector force due to neighboring robots by only taking into account robots which are within the collision proximity. Also, the control unit set the state variables based on the number of robots which are within collision distance. The overall process is described in figure 15.

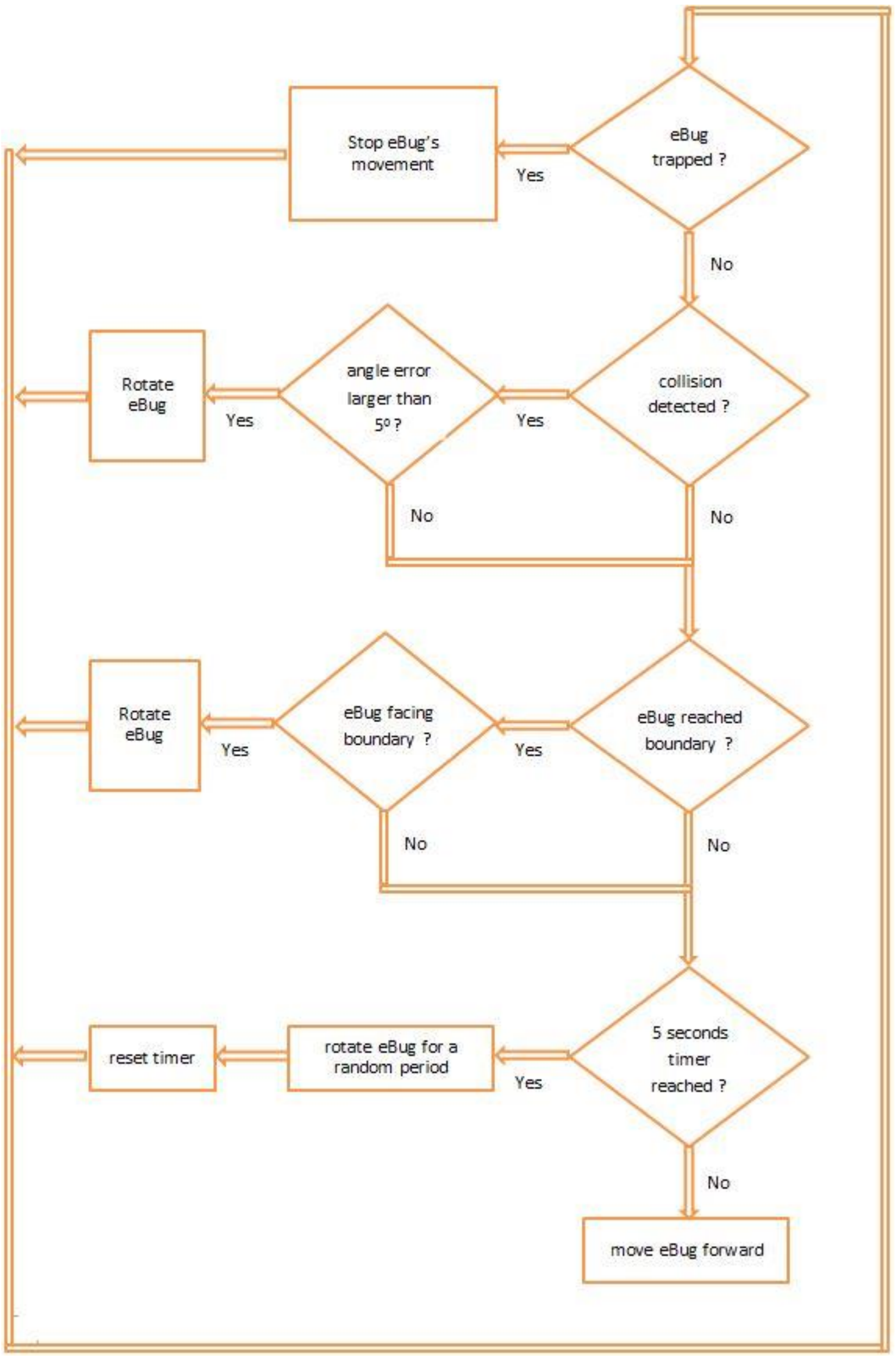
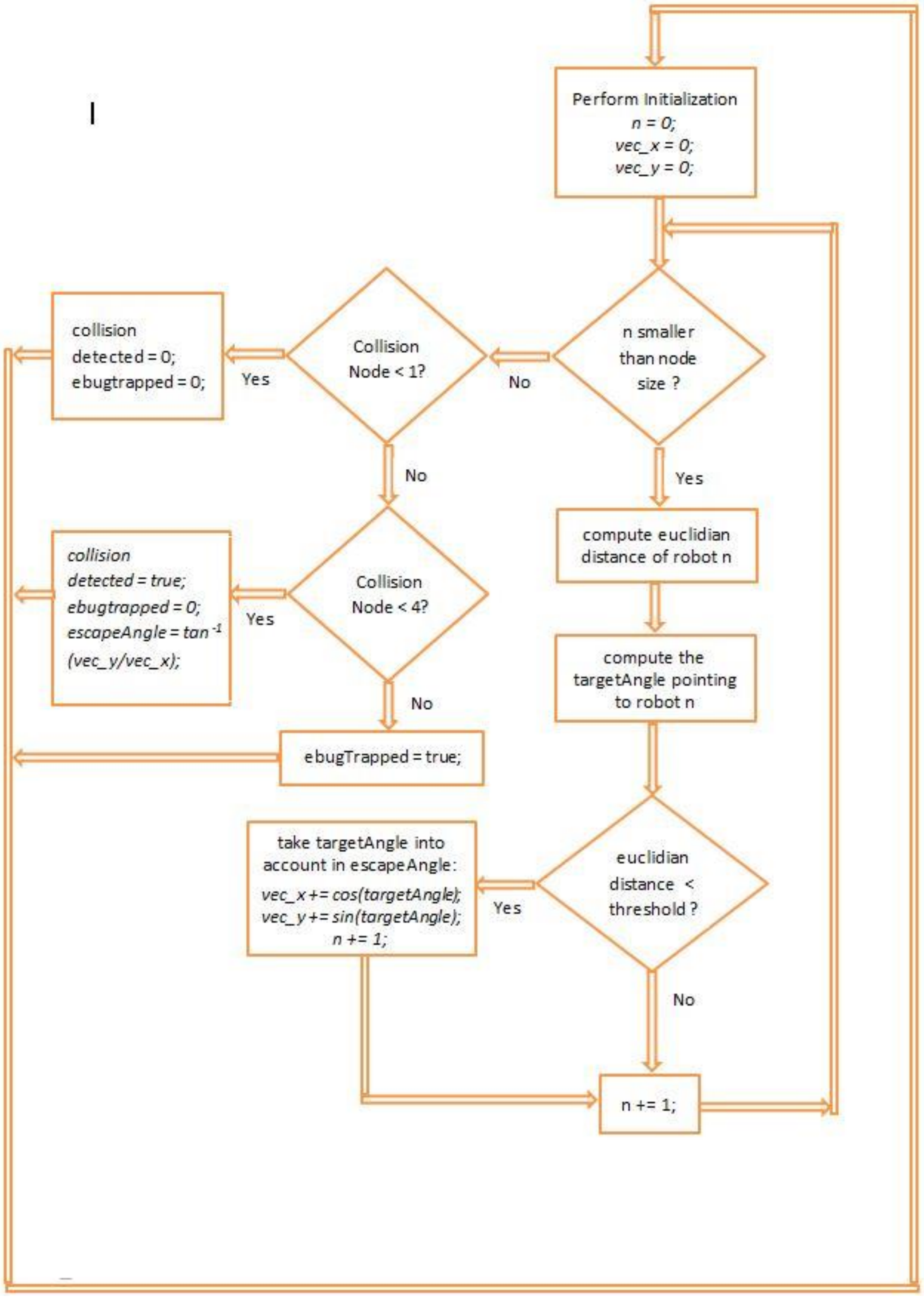


Figure 14: Random-Walk Algorithm implemented on the EbugThread

I



Perform Initialization
n = 0;
vec_x = 0;
vec_y = 0;
n smaller than node size ?
No
Collision Node < 1?
Yes
collision detected = 0; ebugtrapped = 0;
No
Collision Node < 4?
Yes
collision detected = true; ebugtrapped = 0; escapeAngle = tan -1 (vec_y/vec_x);
No
ebugTrapped = true;
Yes
compute euclidian distance of robot n
compute the targetAngle pointing to robot n
euclidian distance < threshold ?
Yes
take targetAngle into account in escapeAngle:
vec_x += cos(targetAngle);
vec_y += sin(targetAngle);
n += 1;
No
n += 1;

Figure 15: Algorithm implemented on the Control Unit which update the state of EbugThread

8 Formation Control

8.1 Morse Potential Function

Morse potential function has been known for its use to represent the potential energy in a diatomic molecule, which consists of two atoms either with the same or different chemical elements. In its general form, the Morse potential function is represented as shown in equation 1. In this case, ***r*** represents the distance between the two atoms while ***r_e*** is the equilibrium bond distance. It is clear from *equation 1* that the potential energy reaches zero at equilibrium distance.

$$V(r) = D_e(1 - e^{-a(r-r_e)})^2 \qquad \text{Eq (1)}$$

Apart from that, the main characteristic of the Morse potential function is that unlike a harmonic oscillator, the energy levels of Morse potential function are not evenly spaced. As the energy approaches the dissociation energy, which is the energy required to break the bond between atoms, the spacing between energy levels of the Morse potential function decreases. However, it should be noted that when the distance between two atoms is less then the equilibrium distance, the energy levels are still evenly spaced. This is illustrated in figure 16.

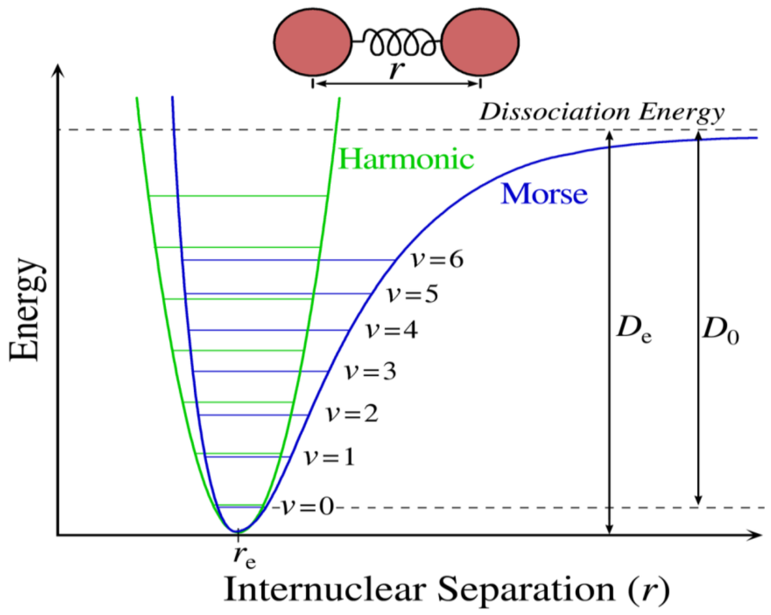


Figure 16: Showing Energy levels of Morse potential function

In this project, Morse potential function is used to implement formation control on multiple eBugs by treating each eBug as an atom. The difference is, however, that the number of atoms is not limited to two in this case. Each eBug experiences forces acting on them due to its atomic bond with the neighbouring eBugs. The forces experienced by each eBug are generated from the negative gradient of the Morse potential function, which is proposed in [18]. Thus, when eBugs are separated at a distance larger than the equilibrium distance $\boldsymbol{r_e}$,they experience an attractive force towards each other. On the other hand, eBugs will experience a repulsive force when the distance between them is smaller than the equilibrium distance.

It should also be emphasized that the repulsive force generated from neighbouring eBugs is much larger compared to the attractive force. This is due to the fact that when the distance between atoms is less than the equilibrium distance, the slope of Morse potential energy is much steeper compared to the case of separation distance being larger than equilibrium distance as shown in figure 16. Therefore, by adopting this method, the chances of collisions to occur between eBugs are reduced. Apart from that, the Morse potential function used for the formation control algorithm is modified as suggested in [18]. The modified Morse potential function is shown in equation 2.

$$U(i) = \sum_{j \neq i} [-C_a e^{-d_{ij}/l_a} + C_r e^{-d_{ij}/l_r}] \qquad \text{Eq (2)}$$

In equation 2 shown above, all the parameters are constant except d_{ij} , which refers to the euclidian distance between robots. The attractive and repulsive strengths are represented by C_a and C_r respectively while l_a and l_r are the scale factor of the attractive and repulsive lengths. It should also be noted that there are two important restrictions in this case. One of them is that the shaping constants must be a positive real numbers. Another restriction is l_a , which is the length scale for the attractive strength, cannot be equal to l_r.

8.2 Simulation

8.2.1 Tuning Constant Parameters

As each eBug has a radius of 6 cm, the distance between eBugs must always be larger than 12 cm to prevent collisions. It should also be emphasized here that the eBug tracking system has a limitation in which the computer vision algorithm may fail to detect the eBugs when the distance between two or more eBugs is less than approximately 18 cm. In that case, one can observe that setting the equilibrium distance to 18 cm will still cause the control scheme to fail as the repulsive force can only be generated if positional information about each eBug is available from the tracking system. As a result, the equilibrium distance of the Morse potential function is chosen to be 36 cm, which is double the minimum allowable distance between eBugs. The size of eBugs in software is represented in pixel size and therefore the equilibrium distance is converted into pixels, which is found to be 125 pixels in size. In addition to that, since there is no precise method to tune the constant parameters of the Morse potential function, tuning is performed manually to produce $\boldsymbol{r_e}$ of 125. The tuning process is done using MATLAB software. Figure 17 shows the Morse potential function with an equilibrium distance of approximately 125.

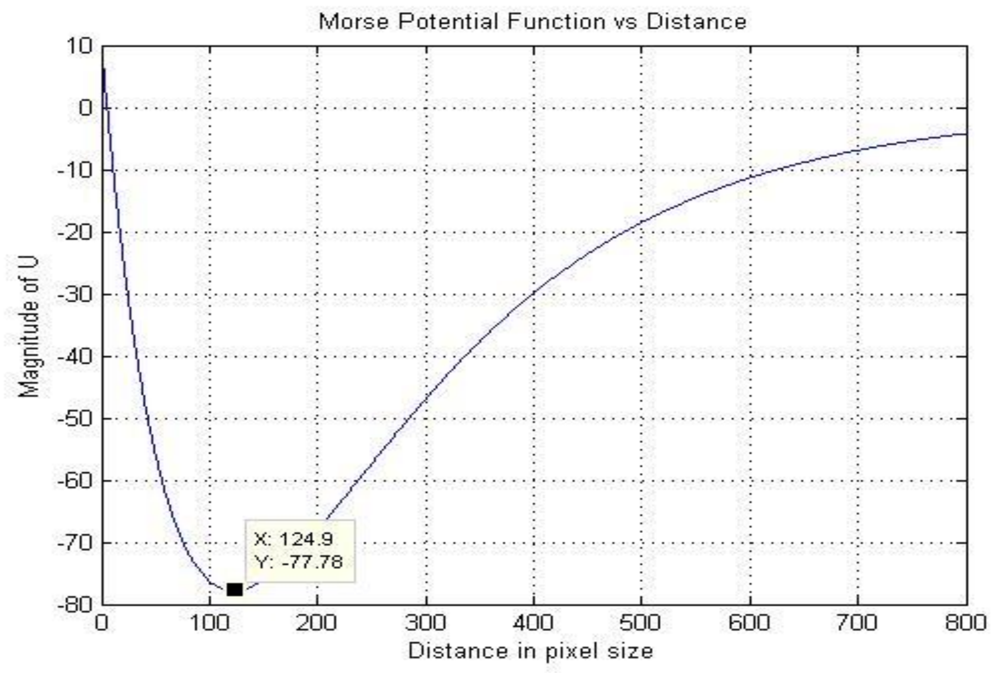


Figure 17: Showing Morse potential function vs Distance

The constant parameters which result in the corresponding equilibrium distance are shown in table 1.

Table 1 Morse Potential Function parameters

Parameter	Value
Ca	230
Cr	240
la	200
lr	75

8.2.2 Generating Artificial Forces

Based on the study conducted in [18], artificial forces, which are used to move the robot, are generated by finding the negative of the gradient of the Morse potential function. For our purpose, the force generated is a two-dimensional vector, since the robot's movement is limited in a two-dimensional space. $\overrightarrow{Fx}$ and $\overrightarrow{Fy}$, which refer to the vector force generated in the x direction and y direction respectively, can then be computed using equation 4 and equation 5.

$$\overrightarrow{Fx_\iota} = -\nabla Ux(i) \qquad \text{Eq (3)}$$

$$\overrightarrow{Fx_\iota} = -\nabla Uy(i) \qquad \text{Eq (4)}$$

Then, the validity of these equations are verified using MATLAB. In the first simulation, it is assumed that one robot exists on the centre of the arena. The centre of the arena refers to location (320,240) in this case, as the size of the arena in terms of pixels size is 640x480. Initially, the energy levels of the Morse potential function are calculated across the entire arena. Figure 18 and figure 19 show the Morse potential function from the top view and 3D view respectively. The vector field generated from the Morse potential function is also shown in figure 20.

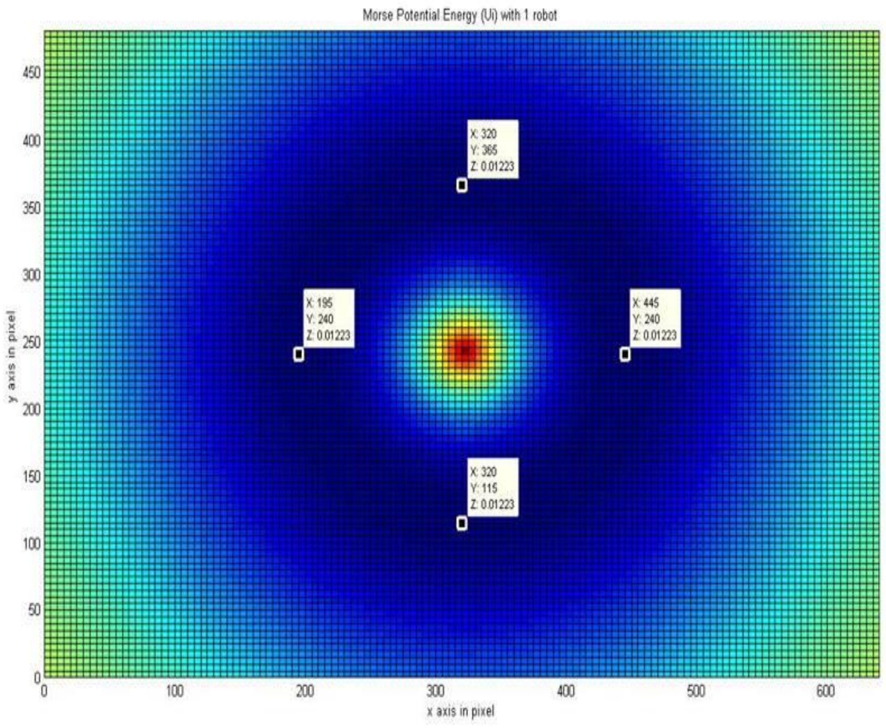


Figure 18: Morse potential function in planar view

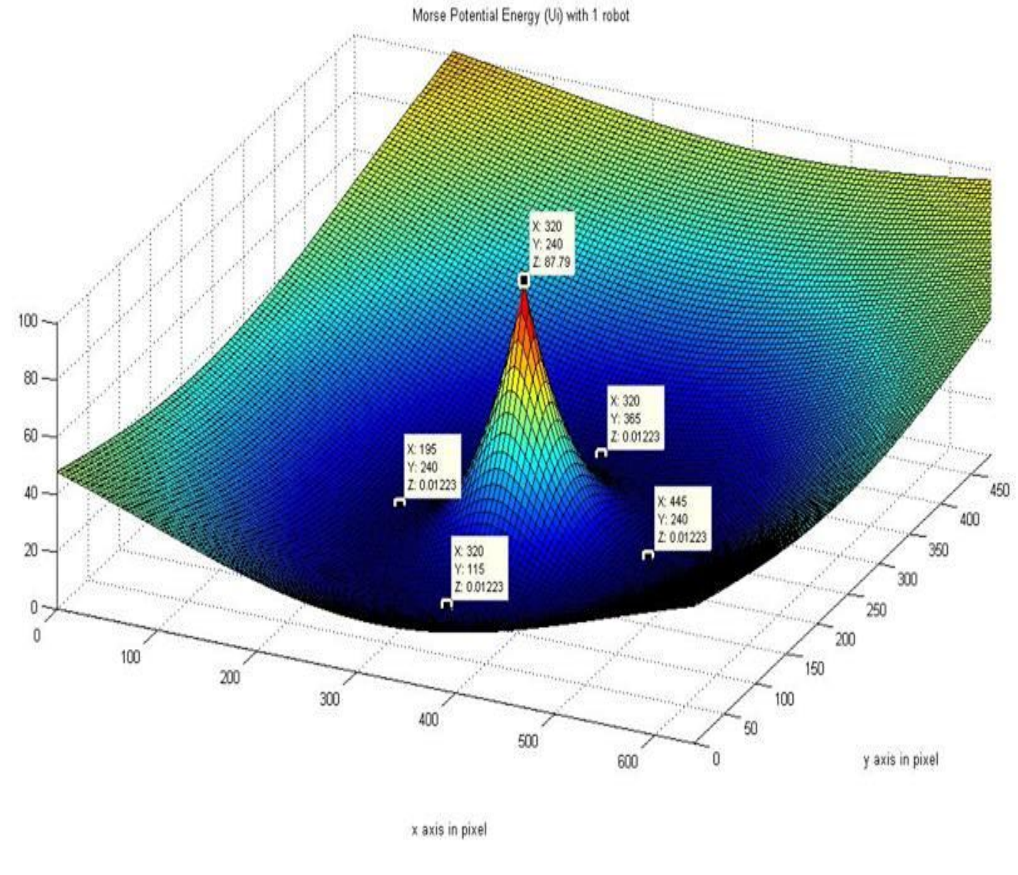


Figure 19: Morse potential function in 2-D space

From figure 18 and 19, it can be seen that the Morse potential function reaches its minimum value at four different locations, which are (195,240), (445,240), (320,115) and (320,365). This is logical because from equation 1, it is clear that the Morse potential function reaches its minimum value at $\boldsymbol{r} = \boldsymbol{r_e}$, which is set to be 125 pixels in this case. Apart from that, it is clear from figure 19 that the Morse potential function reaches its peak at location (320,240), which is assumed to be the robot's location. This is also desirable to prevent collisions as it creates a very strong repulsive force when there is another robot located nearby the centre of the arena.

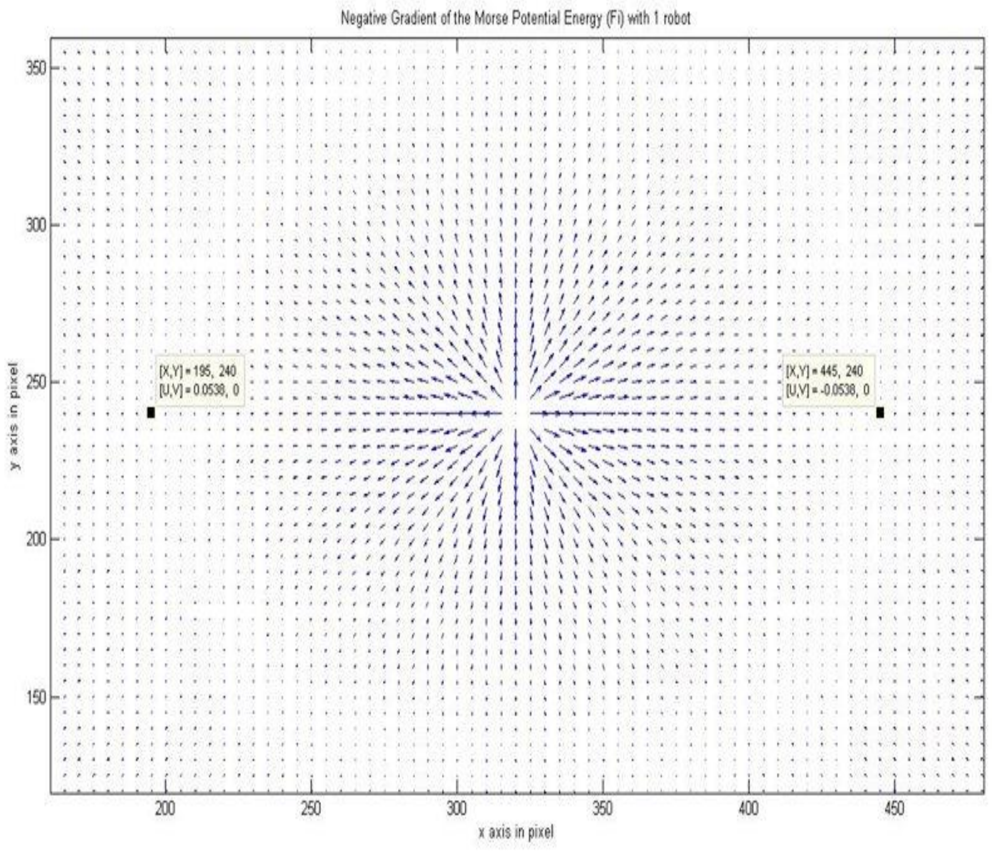


Figure 20: Artificial Forces generated from the Morse potential Function

It can be seen from figure 20 above that the vector fields are pointing away at location (320,240). This is sensible since the robot, which is located at that location, is generating a repulsive force which will push away neighbouring robots and thereby preventing collisions. Apart from that, the equilibrium distance, which is set to be 125 in pixels size, is verified by observing the magnitude of the forces generated at two different locations which are 125 pixels away from location (320,240). It is clear from figure 20 that the forces at location (195,240) and (445,240) are almost zero. Since both of the aforementioned locations are 125 pixels away from the centre of the arena, the vector fields generated are valid. In addition, it can be seen from figure 21 that a robot located at distances larger than the equilibrium distance from the centre of the arena will experience an attractive force. It should also be noted that the density of the vector fields plotted in figure 21 is sparse since the vector field are computed with a step size of 5 pixels size. This is done mainly to reduce the simulation time.

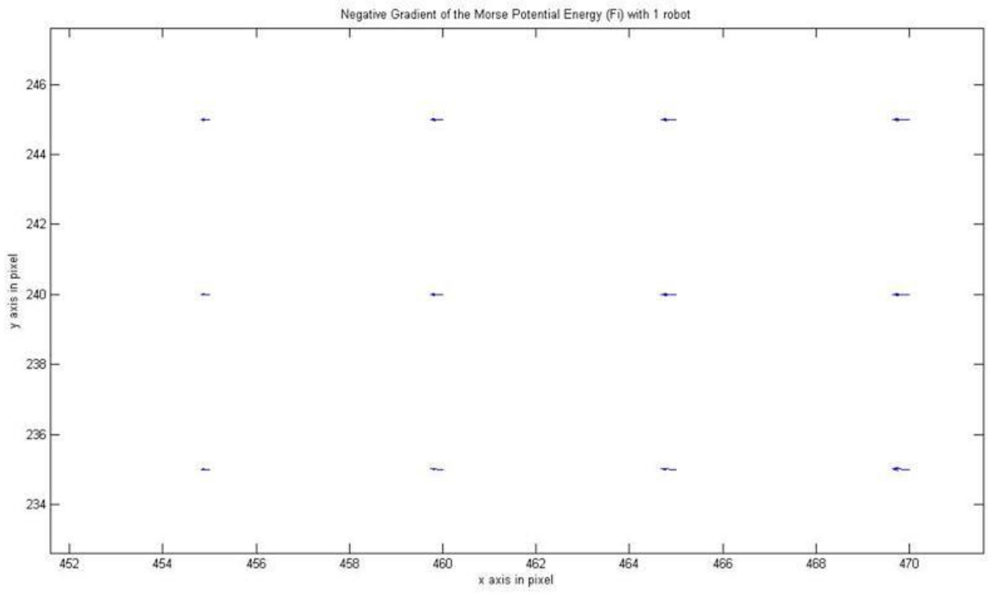


Figure 21: Atrractive forces are generated at distance larger than r_e

8.3 Implementation

Although the force equations used to implement Morse potential based formation control have been verified through MATLAB, they cannot be implemented on the real physical robots yet. The reason behind this is that it does not take into account the non-holonomic constraints imposed by a two-wheeled robot where moving sideways is not possible. To illustrate, consider a force pointing in the positive x direction acting on a two-wheeled robot, which is at ninety degree angle. In this case, it is clear that it is impossible for the robot to instantaneously move in the direction of the acting force. In order to take the nonholonomic constraints into account, a linear speed controller, which is adapted from [18], is introduced in equation 6.

$$v_i = \overrightarrow{Fx_\iota} \cos\theta_i + \overrightarrow{Fy_\iota} \sin\theta_i \qquad \text{Eq (5)}$$

In equation 6, v_i refers to motors speed of the robot while θ_i is the robot's orientation. It is clear how equation 6 addresses the issue with non-holonomic constraint of the robot. In the previous example, where the force is acting horizontally on the robot which is sitting at ninety degree angle, the motors speed of the robot is zero since both $\overrightarrow{Fy_\iota}$ and $\cos\theta_i$ would be zero. Hence, the use of equation 6 to address robot's nonholonomic constraints is justified. Apart from that, it is clear that an additional controller is required in order to allow robots which experience perpendicular forces to move. In this case, an angular speed controller suggested in [18] is adapted to adjust the robot's orientation in the direction of the vector forces. The adapted controller is shown in equation 7.

$$w_i = C_p \, [\overrightarrow{Fy_\iota} \cos\theta_i - \overrightarrow{Fx_\iota} \sin\theta_i] + C_\theta \, [\, arctan2\left(\frac{\sum_{j\in N} \sin\theta_j}{\sum_{j\in N} \cos\theta_j}\right) - \theta_i] \quad \text{Eq (6)}$$

The first term of equation 7 acts as an aligning torque, which takes into account the artificial forces generated from the negative gradient of the Morse Potential Function. In this case, it can be seen that the first term of equation 7 will be zero when the robot is oriented in the same direction as the vector force. On the other hand, the second term takes into account the average orientation of neighbouring robots and the robot's self orientation. One can observe that the magnitude produced by the second term is proportional to the angle error between the robot's self orientation and the average orientation of its neighbouring robots. Also, there are two tuning parameters in equation 7, namely C_p and C_θ. C_p in this case represents the strength of the polar moment while C_θ can be thought as the strength of torsional springs between a robot and its neighbours. It should also be noted that the value of these two tuning parameters is chosen to be the same as the value used in the simulation model proposed in [18]. The value of these parameters is shown in table 2.

Table 2 Constant parameters used for angular speed controller

Parameter	Value
C_p	230
C_θ	240

Apart from that, it has been found that the magnitude of $\overrightarrow{Fx_\iota}$ and $\overrightarrow{Fy_\iota}$, which are computed directly from the gradient descent of the Morse potential function, are too small to be used at the controllers. As the magnitude of $\overrightarrow{Fx_\iota}$ and $\overrightarrow{Fy_\iota}$ are smallest when the distance between robots is large, $\overrightarrow{Fx_\iota}$ and $\overrightarrow{Fy_\iota}$ are scaled according to their distance. Thus, the artificial forces are now calculated using equation 7 and 8.

$$\overrightarrow{Fx_\iota} = -\nabla Ux(i) * [\frac{distance}{40}] \qquad \text{Eq (7)}$$

$$\overrightarrow{Fy_\iota} = -\nabla Uy(i) * [\frac{distance}{40}] \qquad \text{Eq (8)}$$

It should be noted that the distance parameter in equation 8 and 9 is in pixels size. One can observe from the equations that at distance smaller than 40 pixels, the magnitude of the artificial forces is scaled down. However, this may not happen in practice due to the fact that the tracking system is not able to detect two or multiple robots which are separated by a distance of 60 pixels in size.

Finally, as the two controllers designed based on equation 5 and 6 are independent, in which one controls the robot's translational movement while the other outputs the robot's angular speed, they can be combined to produce the left and right wheels speed of the robot. The left and right wheels speed of the robot can be computed using equation 9 and 10.

$$left\ wheel\ speed = v_i - w_i \qquad \text{Eq (9)}$$

$$right\ wheel\ speed = v_i + w_i \qquad \text{Eq (10)}$$

9 Performance Evaluations

In this section, we evaluate the performance of the Morse potential function based decentralized formation control by collecting experimental data. The performance is evaluated from two aspects, namely scalability of the control scheme and robustness of the proposed algorithm to sensor noise.

9.1 Scalability of the Control Scheme

In this case, scalability of the control scheme is investigated by implementing the formation control algorithm using two up to six eBugs. The results obtained are shown in the following figures.

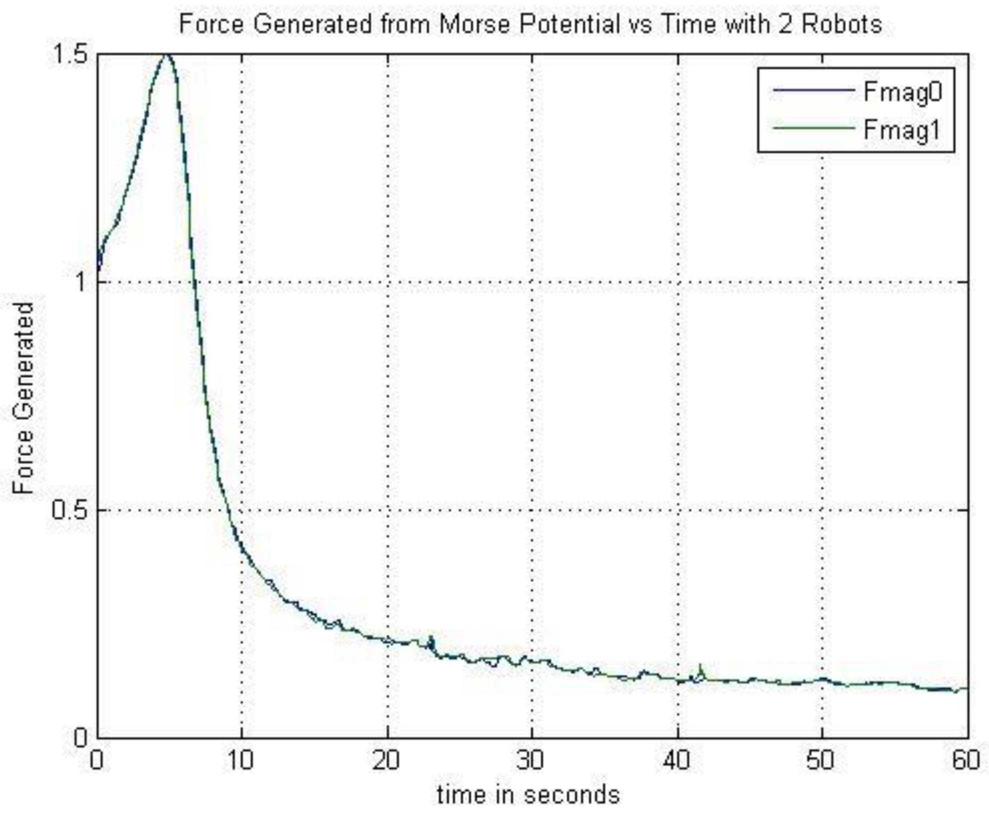


Figure 22: Artificial Forces generated from Morse Potential Function of two robots

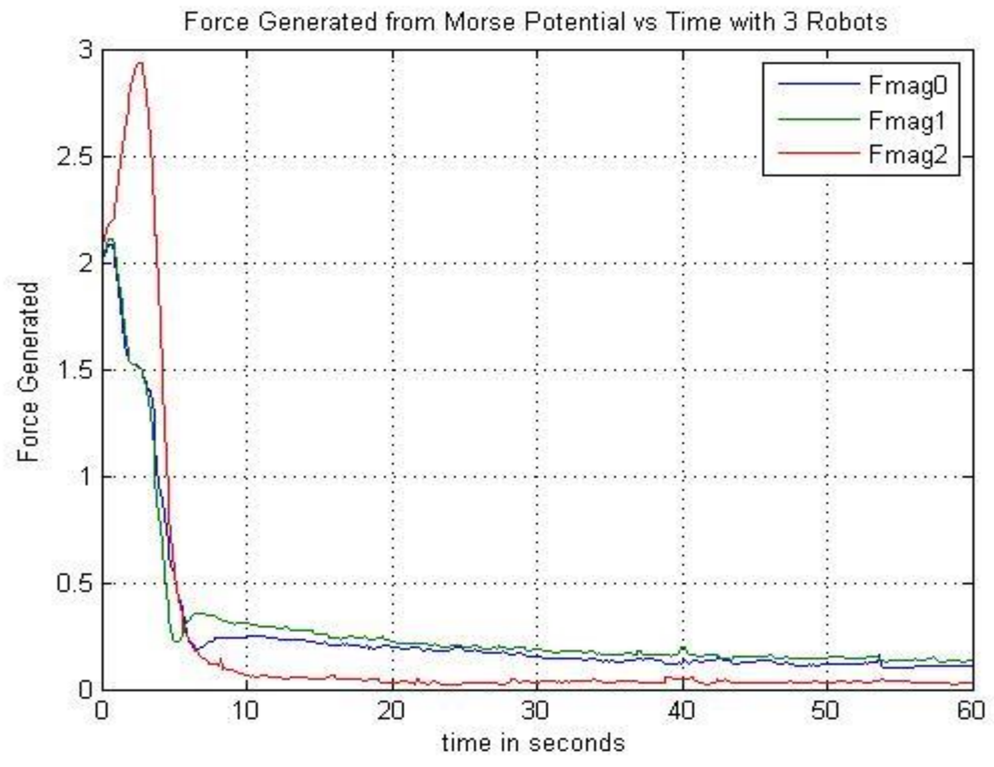


Figure 23: Artificial Forces generated from Morse Potential Function of three robots

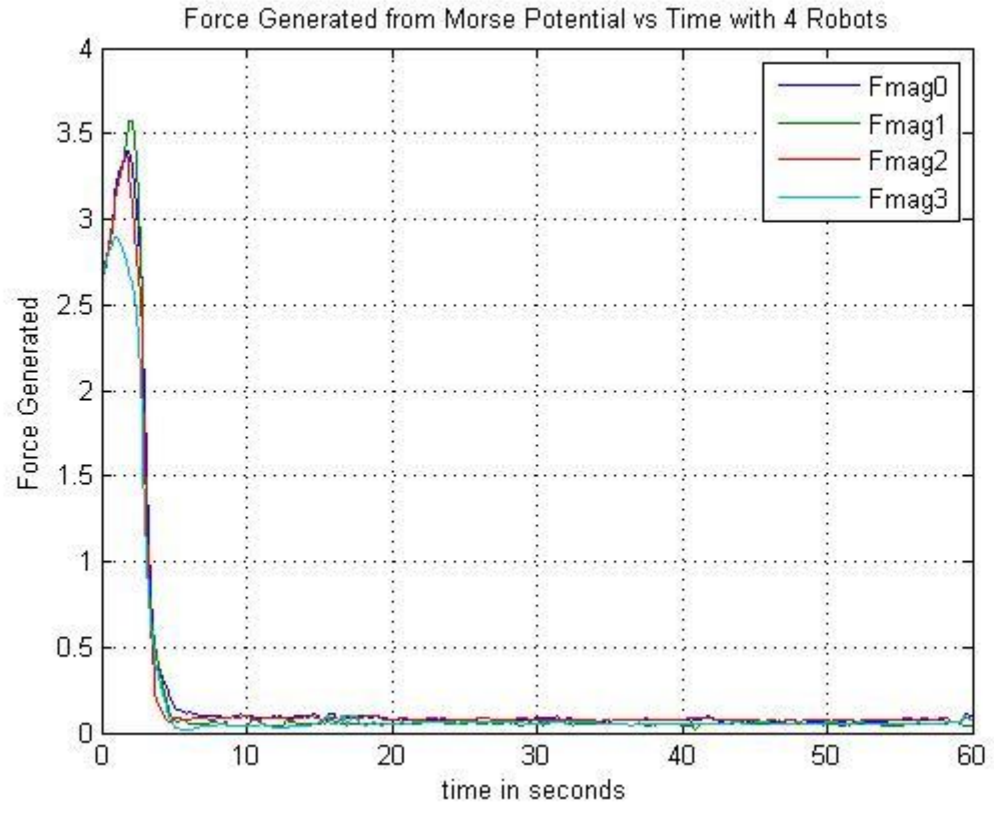


Figure 24: Artificial Forces generated from Morse Potential Function of four robots

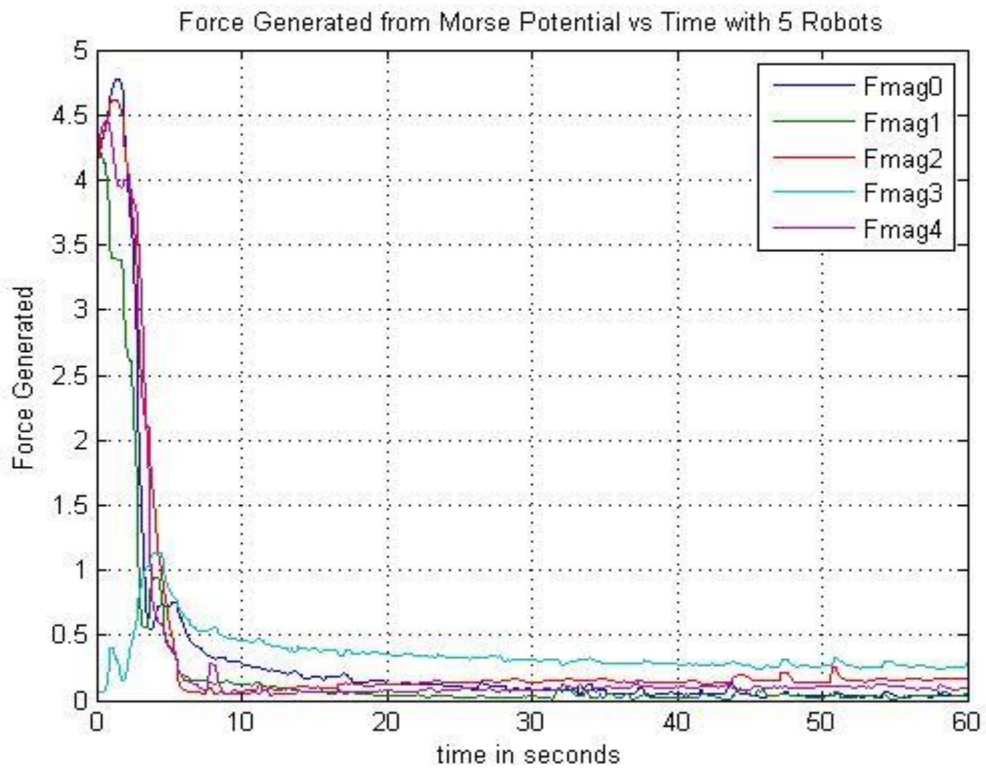


Figure 25: Artificial Forces generated from Morse Potential Function of five robots

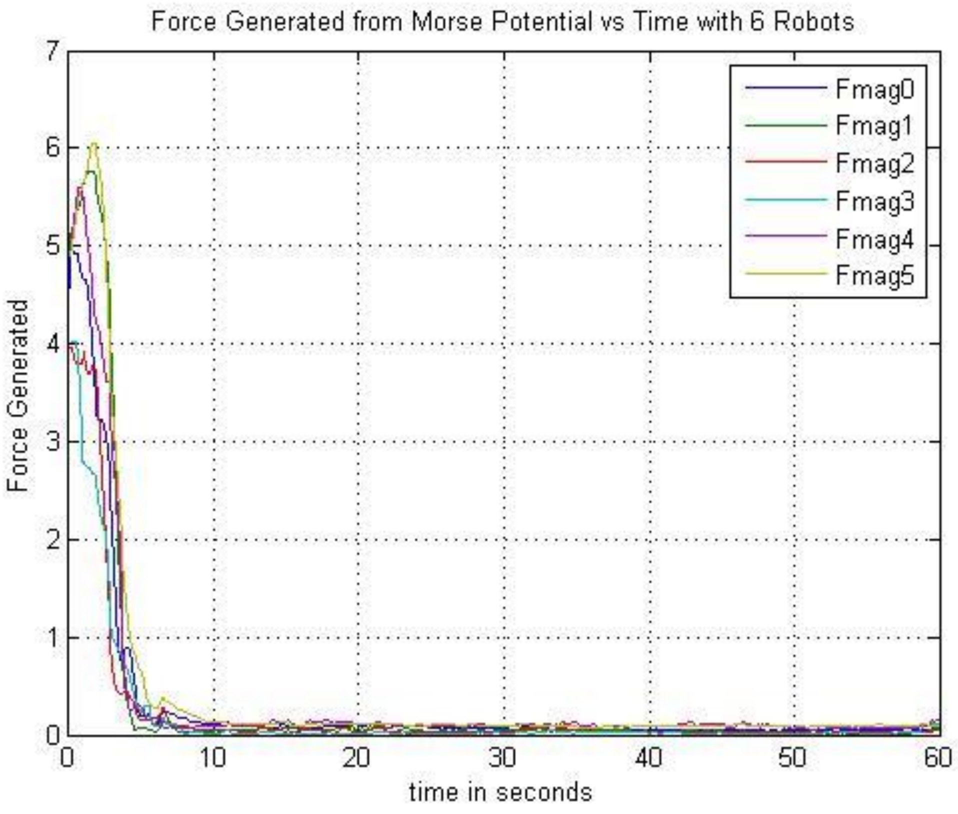


Figure 26: Artificial Forces generated from Morse Potential Function of six robots

From figure 22-26, it can be seen that the Morse potential function based formation control scheme performs reasonably well. In most cases, the artificial forces experienced by each robot are able to reach the minimum value within 10 seconds. It should be noted here that since the vector forces produced by each neighboring robot is summed together to produce a single artificial force, the magnitude of the vector force generated when using 6 robots is much larger compared to when using 2 robots. This may explain why the system with 6 robots reaches convergence at a much shorter time compared to system with 2 robots which can be seen from figure 22 and 26.

On the other hand, it has been found here that increasing the number of robots affects the equilibrium distance experience by each robot. In this case, as the number of robots increases, the separation distance of the robot becomes smaller. As such, at some point, the body of the robots may overlap with each other which may result in collisions. Thus, it seems that the proposed control scheme is not scalable.

9.2 Robustness to Sensor Noise

In this section, the robustness of the algorithm towards sensor noises is tested by using four robots with an artificial noise of up to twenty percent is injected into the system. The following figures show how the artificial forces are affected by sensors noise.

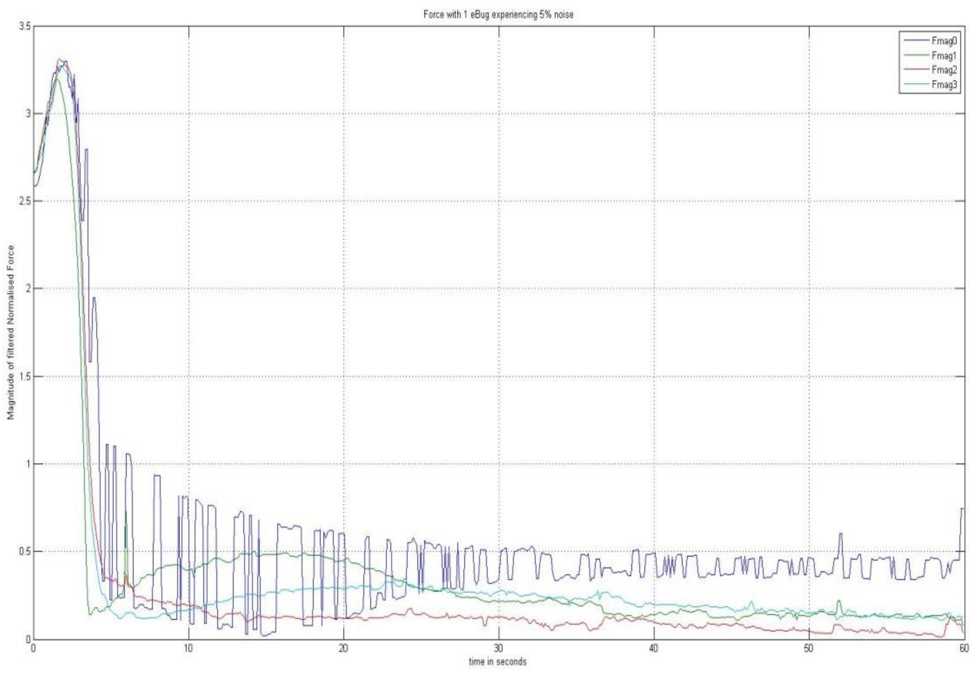


Figure 27: Artificial Forces generated with 1 robot experiencing 5% noise

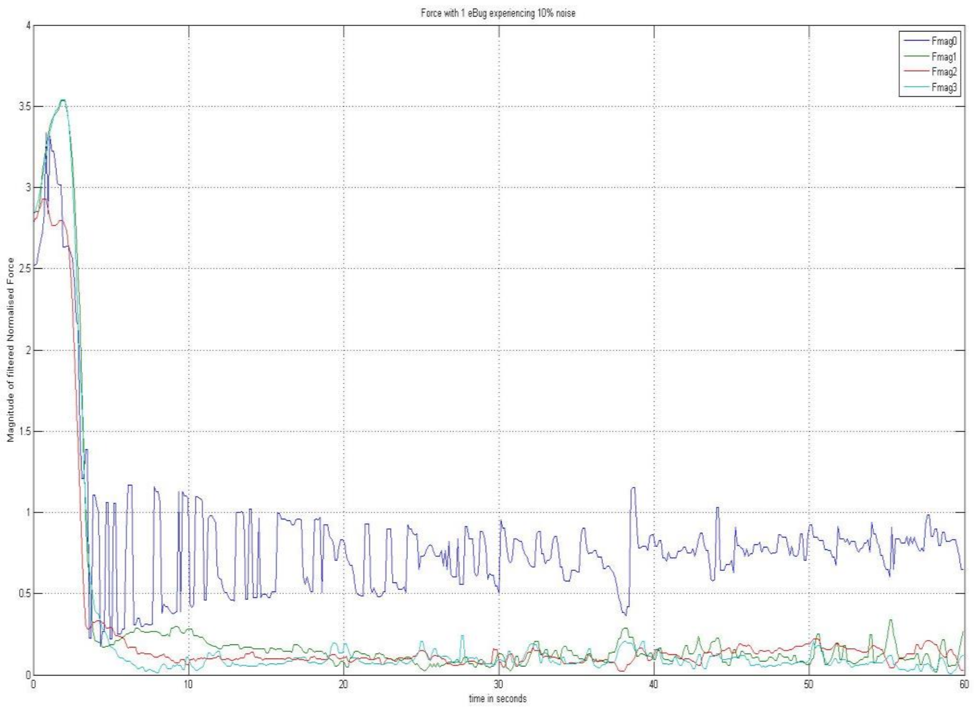


Figure 28: Artificial Forces generated with 1 robot experiencing 10% noise

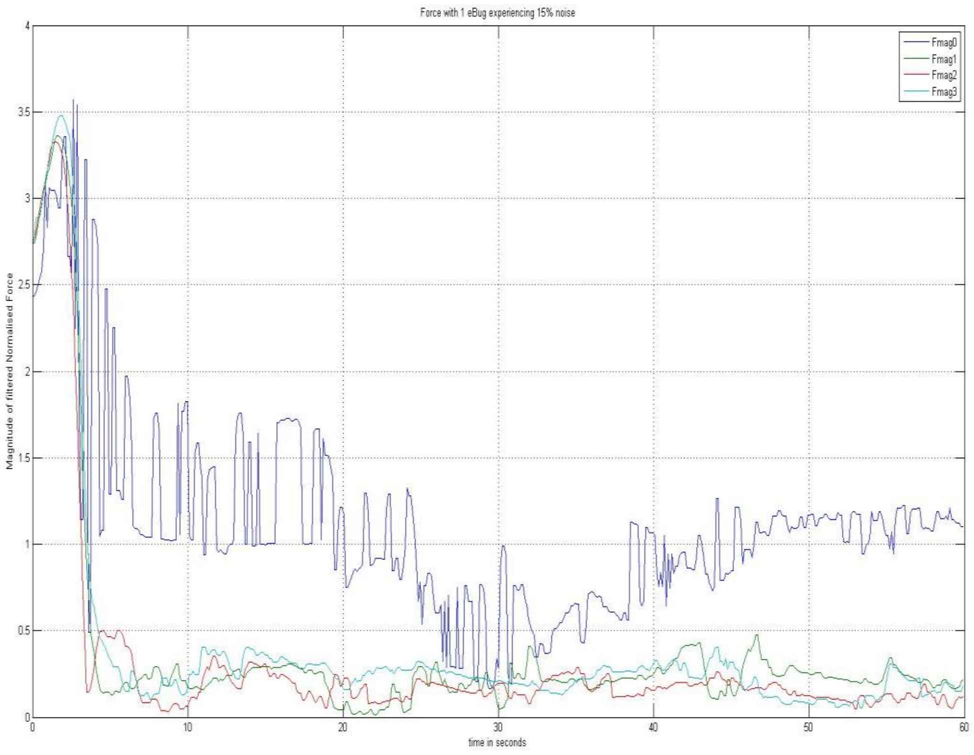


Figure 29: Artificial Forces generated with 1 robot experiencing 15% noise

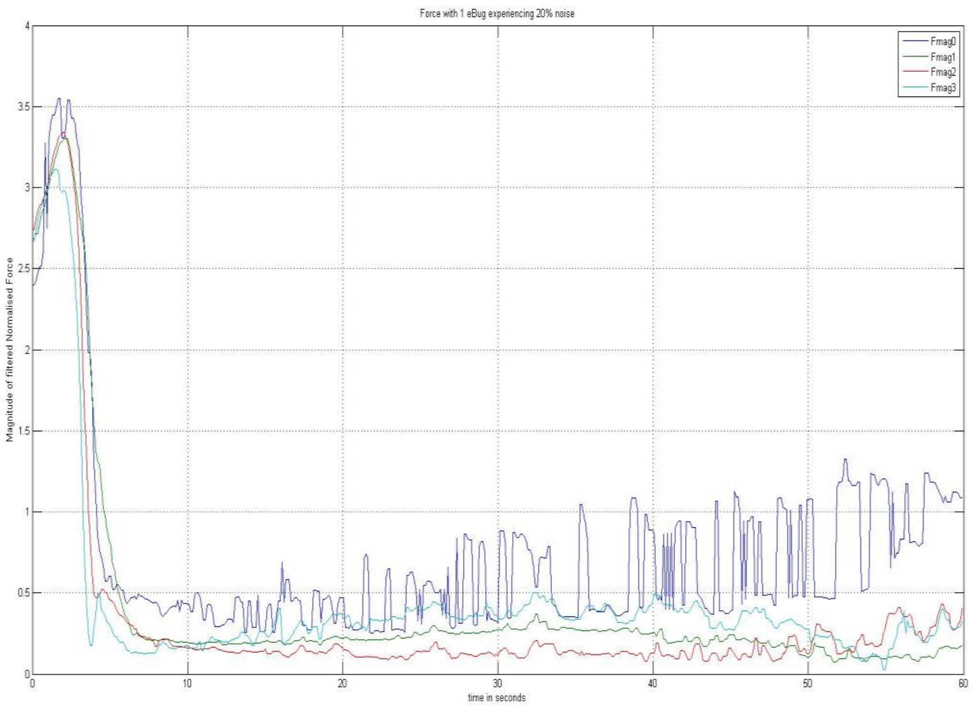


Figure 30: Artificial Forces generated with 1 robot experiencing 20% noise

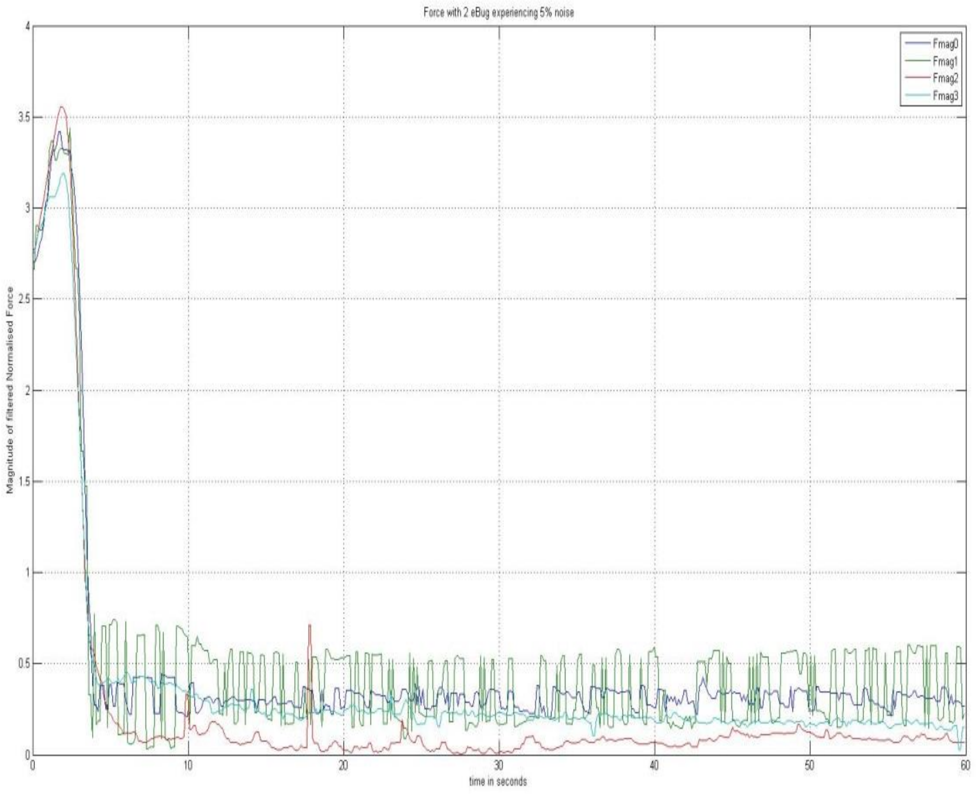


Figure 31: Artificial Forces generated with 2 robots experiencing 5% noise

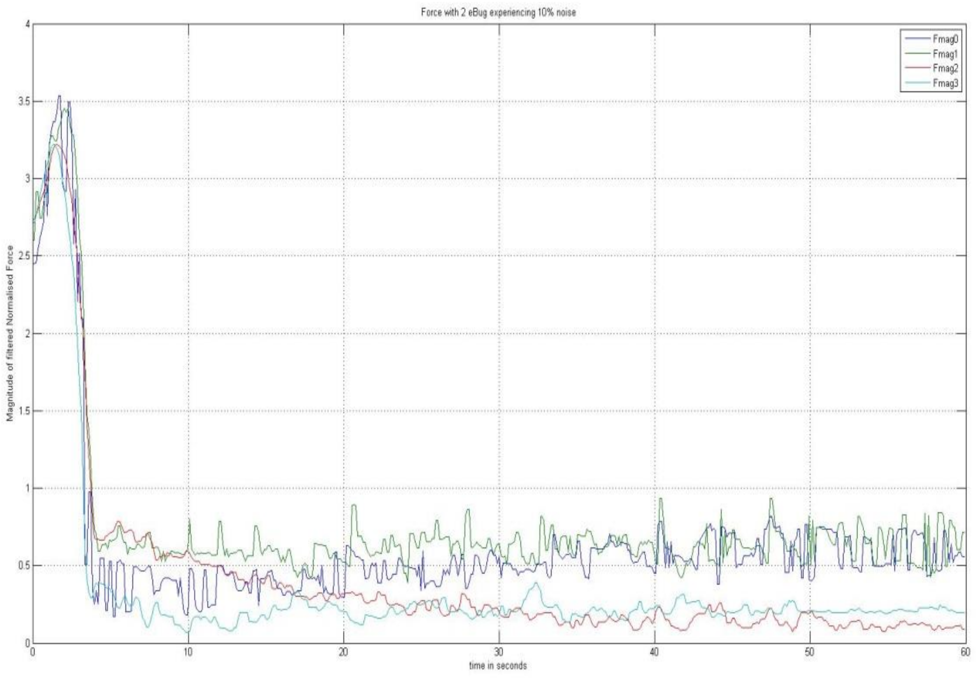


Figure 32: Artificial Forces generated with 2 robots experiencing 10% noise

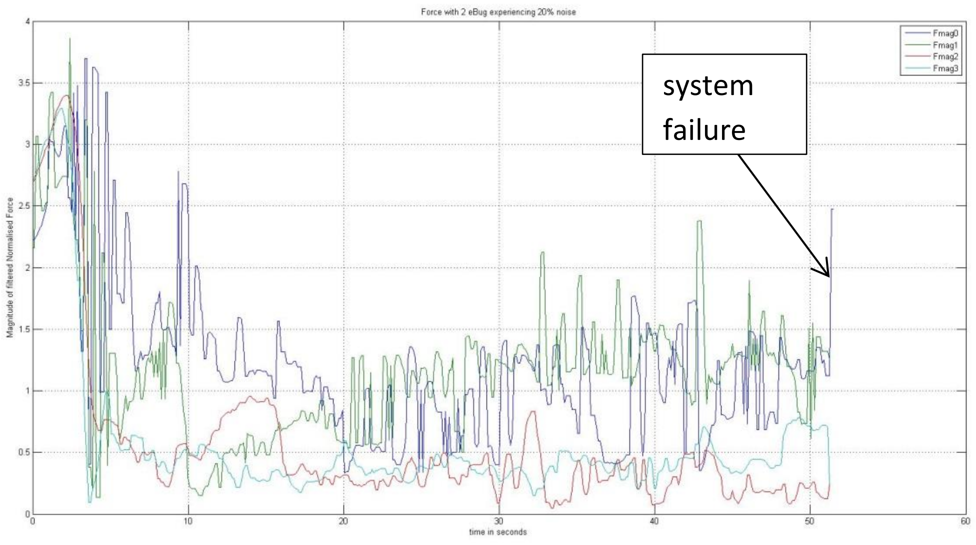


Figure 33: Artificial Forces generated with 2 robots experiencing 15% noise

From figure 27-33, it is clear that the control algorithm suffers from sensor noise. The artificial forces generates oscillatory behaviour and they are not able to reach the minimum point. In figure 33, the system fails just after 50 seconds [3] where collisions between robots occur.

10 Conclusion

In conclusion, the formation control architecture which is based on cloud infrastructure has been successfully implemented. The control architecture has been verified with random-walk algorithm. On the other hand, the proposed control scheme which is based on Morse potential function has several weaknesses. One of them is the fact that the separation distance between robots depends on the number of robots deployed. The other weakness is that the system will never be able to reach a stable equilibrium when sensor noise is introduced. Therefore, future works on this field may focus on improving the scalability and robustness of the control algorithm.

11 Bibliography

[1] J. C. Barca and A. Sekercioglu, "Swarm robotics reviewed," *Robotica,* vol. 31, no. 10.1017/S026357471200032X, pp. 345-359, 2013.

[2] J. C. Barca, Y. A. Sekercioglu and A. Ford, "Controlling Formations of Robots with Graph Theory," in *12th International Conference on Intelligent Autonomous Systems (IAS)*, Jeju Island, 2012, pp. 1-8.

[3] A. Fujimori, T. Fujimoto and G. Bohacs, "Formated Navigation of Mobile Robots using Distributed Leader-Follower Control," Budapest, 2005.

[4] M. M. Zavlanos and G. J. Pappas, "Distributed Formation Control with Permutation Symmetries," in *46th IEEE Conference on Decision and Control*, New Orleans, LA, 2007.

[5] M. A. Lewis and K.-H. Tan, "High Precision Formation Control of Mobile Robots Using Virtual Structures," *Autonomous Robots,* vol. IV, no. 4, pp. 387-403, 1997.

[6] A. K. Das, R. Fierro, V. R. Kumar, J. P. Ostrowski, J. Spletzer and C. J. Taylor, "A vision-based formation control framework," *Robotics and Automation, IEEE Transactions on,* vol. 18, no. 5, pp. 813-825, 2002.

[7] C. D. Cruz and R. Carelli, "Dynamic Modeling and Centralized Formation Control of Mobile Robots," in *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on*, Paris, 2006.

[8] R. Carelli, C. D. Cruz and F. Roberti, "Centralized formation control of non-holonomic mobile robots," *Latin American Applied Research,* vol. 36, no. 2, pp. 63-69, 2006.

[9] H. G. Tanner, G. J. Pappas and V. Kumar, "Leader-to-formation stability," *Robotics and Automation, IEEE Transactions on,* vol. 20, no. 3, pp. 443-455, 2004.

[10] D. V. Dimarogonas and K. J. Kyriakopoulos, "A Feedback Stabilization and Collision Avoidance Scheme for Multiple Independent Nonholonomic Non-Point Agents," in *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, 2005, pp. 820-825.

[11] D. Maithripala, J. Berg, D. Maithripala and S. Jayasuriya, "A geometric virtual structure approach to decentralized formation control," in *American Control Conference (ACC), 2014*, 2014, pp. 5736-5741.

[12] K. M. Lieberman, F. K. Gregory and D. P. Garg, "Decentralized Control of Multi-Agent Escort Formation via Morse Potential Function," in *ASME*, Florida, 2012.

[13] N. Michael, Z. M, K. V and P. G, "Distributed Multi-Robot Task Assignment and Formation

Control," *Autonomous Robot,* vol. 3, no. 4, pp. 50-63, 2013.

[14] L. Seng, J. Barca and A. Sekercioglu, "Distributed Formation Control in Cluttered Environments," Melbourne, 2013.

[15] W. Ren and N. Sorensen, "Distributed coordination architecture for multi-robot formation control," Logan, 2007.

[16] C. Fu, L. Hao, Y. Feng and G. Chong, "Distributed formation control for a multi-agent system with dynamic and static obstacle avoidances," in *Chin. Phys. B*, Shanghai, 2014.

[17] J. Yuan and G. Tang, "Formation Control of Mobile Multiple Robots Based on Hierarchical Virtual Structures," in *8th International Conference on Control and Automation*, Xiamen, China, 2010.

[18] G. Fricke, K. Lieberman and D. Garg, "Swarm Formations under Nonholonomic and Numerosity Constraints," in *ASME 2012 5th Annual Dynamic Systems and Control Conference*, Florida, 2012.