Leader-Follower Formation Control of ROS Enabled Mobile Robots Subject to Robots Failure

by

Yusuf Abdul-Razaq

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

CONTROL SYSTEMS

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Over the years, control of autonomous vehicles in a defined formation has been the subject of much research. Albeit leader-follower approach being one of the most used in formation control, it suffers a major practical drawback of leader failure while cruising in formation. In this work, we aim to solve this problem by proposing a novel assignment algorithm that assigns a new leader from the follower robots to ensure robots complete their given task when their leader fails. This algorithm also assign role to new robots joining the group, and also to the failed robot when rescued back to the team. We drive robots towards their desired trajectories to achieve formation using a Lyapunov-based time-varying state tracking controller from the literature. Due to role switching amongst member robots, we propose a limit-cycle obstacle avoidance control algorithm to ensure smooth and collision free transition. Simulations and experiments are performed using the Robot Operating System (ROS) framework due to its flexibility to verify the effectiveness and reliability of the proposed algorithms.

To my parents.

Acknowledgments

The completion of this thesis could not have been possible without the generosity and support from a large group of people. Foremost, I'm extremely grateful to my supervisor, Prof. Horacio J Marquez for his priceless advice, support, and patience during my time in his research group. His copious experience and immense knowledge have helped me in all the time of my academic research. Special thanks to Hamid Alian who was extremely generous in providing insights on how to code robots better. I would also like to thank the National Sciences and Engineering Research Council of Canada (NSERC) for their financial support which has made this research possible.

Thanks to Dr. Olufemi Oni for making Canada feel like home to me, and to my friends turned brothers, Temitayo and Mubarak for their emotional support throughout my graduate degree.

I would like to express my gratitude to my parents for their support and encouragement throughout my academic career. Special thanks to my selfless brother, Dr. Abdulgaffar Abdulrazaq, and family at large for their invaluable support during my time away from home. Finally, special thanks to my other half, Nafisat Nasir for her patience and words of encouragement.

Table of Contents

1 Introduction **1**

1.1 Background and motivation 1

1.2 Literature review 3

1.2.1 Formation tracking problems 3

1.2.2 Formation generating problems 5

1.3 Statement of contribution 6

1.4 Synopsis 7

2 Technical preliminaries **9**

2.1 Graph theory 9

2.2 Lyapunov stability 10

2.3 E-puck2 robot 12

2.4 ZED Stereo camera 13

2.5 The Robot Operating System (ROS) 15

2.6 Markers 16

2.7 Drivers 18

3 Formation control with obstacle avoidance and periodic leader switch 21

3.1 Introduction 21

3.2 Robot dynamics 22

3.3 Trajectory tracking 22

3.4 Leader-follower formation control 25

3.5 Obstacle avoidance . . . 26
3.5.1 Theory . . . 26
3.5.2 Algorithm . . . 28
3.6 Hierarchical action selection algorithm . . . 30
3.7 Implementation . . . 31
3.7.1 Simulations . . . 32
3.7.2 Experiment . . . 37

4 Formation control subject to leader failure 44
4.1 Multi-robot algorithms . . . 44
4.2 Assignment algorithm . . . 45
4.3 Implementation . . . 47
4.3.1 Simulations . . . 47
4.3.2 Experiment . . . 52
4.4 Technical difficulties . . . 55

5 Summary and conclusions 60

Bibliography 63

List of Figures

1.1 (a) Amazon Logistic warehouse [1],(b) UGV robots [2], (c) Drone swarms [3] (d) subCULTron Underwater Acoustic Sensor Networks (UASN) concept [4] 2

2.1 (a) Directed graph (b) Undirected graph [23]. 9
2.2 E-puck2 robot [26] 13
2.3 ZED Stereo camera [32] 14
2.4 Basic ROS model 16
2.5 Example of fiducial markers [37] 17
2.6 AR tags [38] 18
2.7 E-puck2 ROS topics [39] 19
2.8 ZED camera RQT Graph Using AR Tags 20

3.1 Robot schematic 23
3.2 Reference and current posture [42] 24
3.3 Phase portrait of the limit-cycle defined by equations 3.13 and 3.14. . 27
3.4 Definition of robot's position relative to an obstacle. 29
3.5 Choice of controller at an instance [21] 31
3.6 Robot-Obstacle setting showing the region of influence of the obstacle. 31
3.7 Robot and obstacle on a ground plane. 33
3.8 (a) Desired and robot's trajectories, (b) Tracking errors and control inputs. 34

3.9 (a) Robot's and reference trajectories, (b)Tracking errors and input velocities. 34
3.10 Trajectory tracking with multiple obstacle avoidance. 35
3.11 (a) Robots at initial position, (b) Robots moving in a formation . . . 36
3.12 Trajectories of (a) Robot0, (b) Robot1, and (c) Robot2. 37
3.13 Trajectories of (a) Robot0, (b) Robot1, (c) Robot2, and (d) Robot3. . 38
3.14 Experimental setup. 39
3.15 Workspace for our experiments. 40
3.16 (a) Robot0 just before the region of influence of robot1, (b) Robot0 inside the region of influence of robot1 and avoiding it, (c) Robot0 back to trajectory tracking after avoiding robot1 completely. 41
3.17 (a) Reference trajectory and robot0 motion trajectory, (b) Robot0 tracking errors and input velocities. 41
3.18 Desired formation shape. 42
3.19 (a) Robot0 leading the group, (b) Robot1 leading the group, (c) Robot2 leading the group, (d) Robot3 leading the group. 43

4.1 Role positions in a square shaped formation with robots occupying positions sequentially. 46
4.2 Desired formation shape. 47
4.3 (a) Robots' trajectories with no robot failure, (b) Trajectories under leader failure. 48
4.4 (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of robot occupying role 1. 49
4.5 (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of robot occupying role 2. 50
4.6 (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of robot occupying role 3. 50

4.7 (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of multiple robots. 51
4.8 (a) Robots' trajectories with no robot failure, (b) Trajectories with failed robot recovery. 52
4.9 Four robots in a formation (a) before leader failure, (b) when the leader failed, (c) when the failed robot is recovered. 53
4.10 Four robots in a formation (a) before leader failure, (b) when the leader failed, (c) when the failed robot is recovered. 54
4.11 Four robots in a formation (a) before role 2 robot failure, (b) when role 2 robot failed, (c) when the failed robot is recovered. 54
4.12 Four robots in a formation (a) before role 3 robot failure, (b) when role 3 robot failed, (c) when the failed robot is recovered. 55
4.13 Test pose estimate. 56
4.14 (a) Test robot tracking a circular trajectory, (b)Plots for tracking errors and input velocities. 57
4.15 non-smooth generated trajectories. 58
4.16 Obstacle avoidance under large pose estimation error. 59

List of Algorithms

1 : Orbital obstacle avoidance 30

2 : Hierarchical action selection 32

3 : Assignment algorithm 46

Abbreviations & Acronyms

AI Artificial Intelligence.

AUV Autonomous Underwater Vehicles.

PSO Particle Swarm Optimization.

ROS Robot Operating System.

UAV Ummanned Ariel Vehicle.

UGV Ummanned Ground Vehicle.

Chapter 1

Introduction

1.1 Background and motivation

The study of multi-robot coordination has grown significantly over the last two decades with advances in robotics and control theory, embedded systems, communication systems, and AI integration. It has attracted enormous attention form different fields such as monitoring and control, search and rescue, transportation, factory floor, homecare, fault diagnosis, just to name a few. This is of course, due to its ability to solve difficult problems that are impossible with a single robot. Loosely speaking, multi-robot systems consist of a group of unmanned ground or aerial vehicles or underwater robots interacting in a certain way to achieve some defined tasks. Some examples are shown in Figure 1.1.

These applications are realized through various cooperative control strategies, this includes cooperative search, formation control, rendezvous, flocking, foraging, just to name a few. With significant research activity in the area of cooperative control, a lot has been published on formation control problem, making it the most actively studied area in multi-robot coordination. Extensive survey in [5] classified formation control strategies into distance-based, position-based, and displacement-based, based on sensing capabilities and interaction topology of agents. In [6], formation control problems are studied through the lens of feedback and communication mechanisms, network topologies, and collective behaviour. In addition, the reference [7] present a

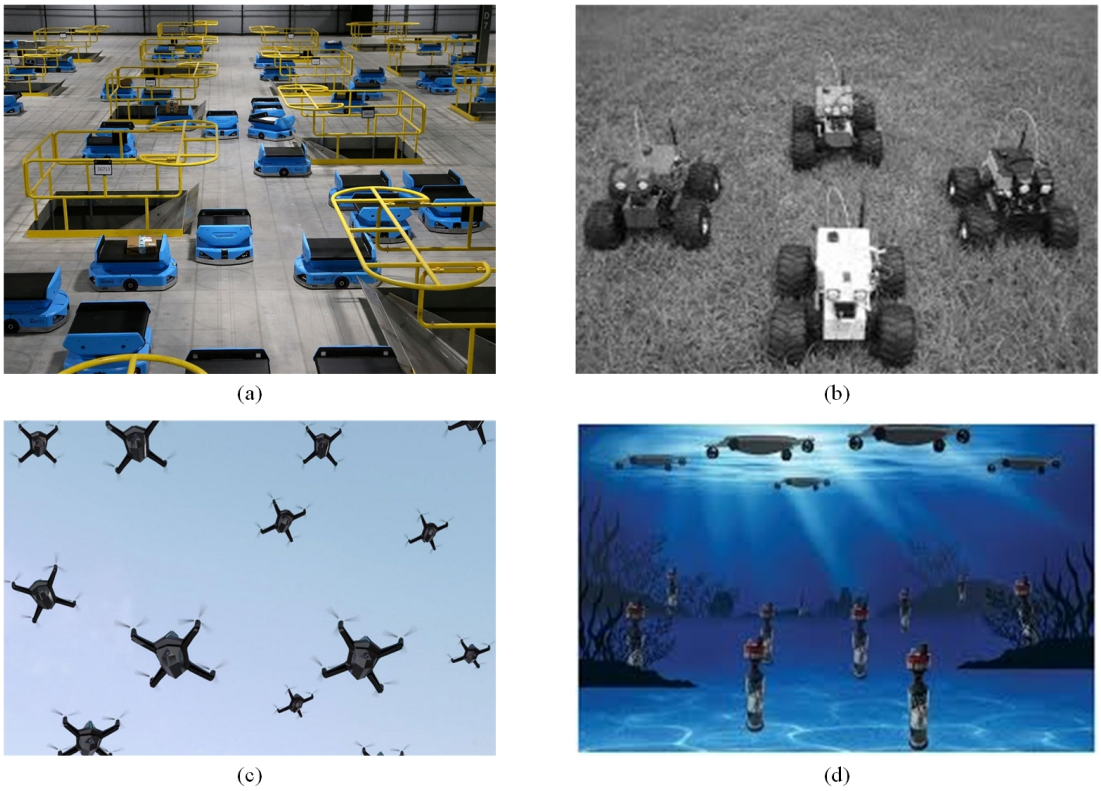

Figure 1.1: (a) Amazon Logistic warehouse [1],(b) UGV robots [2], (c) Drone swarms [3] (d) subCULTron Underwater Acoustic Sensor Networks (UASN) concept [4]

substantial survey, paying attention to methods that adopts flexible formation shape to achieve collision avoidance for multi-vehicle systems. The common control approaches in multi-robot systems are centralized, decentralized, and distributed control approach. This thesis involves the use of distributed formation control, we therefore do not present background knowledge on centralized approach.

According to fundamental ideas in control schemes, references [8, 9] have classified formation control into leader–follower, behavioral, and virtual structure approaches. Leader-follower formation control strategy has been investigated frequently in recent years for its distributed nature and ease of implementation. Basically, one of the robots is designated as the leader and other robots designated as followers, followers track position and orientation of the leader with some prescribed offset. Possible variations include designating multiple leaders, forming a chain, and other tree topologies.

Despite its simplicity and wide range of applications, it suffers a major practical drawback of leader failure. This in turn, result in failure of the whole system to achieve the defined task, which makes the strategy less robust and hence, the need for a lasting solution. While some works as in [10] maneuvered out of this problem by proposing a rather local solution, a flood of papers presented this strategy without addressing this practical problem.

1.2 Literature review

In this section, a brief review on formation control will be provided. We will then briefly review some recent research results relevant to our work, with a more detailed review to be provided in each of the main chapters.

Depending on whether or not desired formations are time varying, formation control problems are generally divided into two, that is, formation generating problems and formation tracking problems. In their review, [5] described briefly how the existing literature addressed each of these problems. They described formation generating problems as problems whose objective of agents is to achieve a prescribed desired formation shape, usually addressed through matrix theory based approach, Lyapunov based approach, graph rigidity based approach, and receding horizon based approach. On the other hand, formation tracking problems as problems where agents are controlled to track some prescribed reference trajectories, usually addressed through matrix theory based approach, potential function based approach, Lyapunov based approach among others.

1.2.1 Formation tracking problems

Formation tracking problems have greater application potential, therefore have received a lot of attention among researchers. An example is in the reference [11], in their work, receding-horizon leader-follower control framework is used to solve the formation problem of multiple non-holonomic mobile robots with a rapid error conver-

gence rate. They proposed two formation schemes; a separation bearing orientation scheme (SBOS) for two robots such that the follower robot follow its leader at a prescribed desired bearing, distance, and orientation deviation. And a separation separation orientation scheme (SSOS) for three robots such that one follower robot follows two leaders by maintaining desired separation with respect to both leaders, and a desired orientation deviation with respect to one of the leaders. They designed in their work, a controller that explicitly control the orientation deviation between leaders and followers to solve what they termed as "formation backward problem", that is, formation problem when robots move backward. Another example is in [12], they propose an optimal formation controller that determines the optimal formation among predefined formations according to the environment. In each of their predefined formation, leader robot move at a constant speed, while follower robots are controlled using a receding-horizon based controller to track leader robot at a prescribed distance. The reference [13] later extend the work in [12] by proposing an automatic formation control method with temporal logic constraints. This is to enable them achieve an optimal formation switching in a cluttered or sufficiently small environment among predetermined formations in real time using receding-horizon based controller.

Lyapunov based approach is used in [10] with leader-follower strategy to solve formation control problem. They treat formation control problem as an extension of trajectory tracking problem. They feed trajectory information to the leader, and the leader has the task of generating a desired pose for each follower robot at some point relative to its pose. Due to nonholonomic constraint of the robot they consider, the desired orientation of each follower is generated such that the reference trajectory will be feasible. They employ the use of an existing Lyapunov based backstepping controller to drive robots to track their respective reference trajectories. In the reference [14] a Lyapunov based tracking controller is designed coupled with an avoidance function for trajectory tracking and obstacle avoidance for a single robot. The result

is then extended to solve formation control problem using the leader-follower strategy. In their approach, leader robot sends its position information to follower robots, then follower robots compute and track a desired posture using the Lyapunov based controller based on the desired distance and bearing sent by a supervisor. The incorporation of avoidance function into their controller ensures robots do not collide with one another when the supervisor decides to switch their formation shape. In [15], multiple autonomous underwater vehicles (AUV) are controlled using a Lyapunov based controller to achieve formation by utilizing the leader-follower formation control strategy in the presence of discrete data transmission between the leader AUV and the follower AUVs. A continuous-discrete extended Kalman filtering (CD-EKF) is designed in this work for follower agents to estimate the position and velocity of the leader when communication constraints occur. Another formation control study of AUV is in [16]. In their work, they designed a novel Lyapunov based tracking control algorithm for the leader robot, and a control law using the Lyapunov theory and feedback linearization techniques to navigate a group of follower robots in a desired formation associated with the leader and follow it simultaneously.

1.2.2 Formation generating problems

Formation generating problem is studied in [17]. In their work, they proposed a decentralized formation control law to achieve desired formation and avoid collisions with obstacles and other robots in the group. Their assumption is that robots get information only form their local neighbours and were able to verify using a set of simulations, the effectiveness of their proposed controller. The reference [18] analyzed formation consensus problem using omnidirectional robots with multiple time delays and noises. Their approach obtained critical stability of the maximum time delay under noisy conditions, then proved that the system can be stabilized and achieve the desired formation when all robot delays are less than the maximum time delay. Relative to traditional Lyapunov method, they proved their solution's lower conser-

vativeness and ease of extension to higher-order system.

Another formation generating problem is studied in [19]. Their work addressed the problem of shape distortions in distance-based formation control when robots are subjected to unknown external disturbances. Performance bounds that constrain the inter-robot distance errors are used to handle connectivity maintenance and collision avoidance among neighboring robots. They established using graph theory, input to state stability, and Lyapunov analysis, a decentralized control scheme that increases formation robustness against shape distortions and formation convergence to undesired shapes under the effect of external disturbances.

1.3 Statement of contribution

Although there is abundance of literature on the leader-follower strategy to solving formation control problems, we have not come across to the best of our knowledge at the time of this work, a research result that explicitly address the practical problem of leader failure in this strategy. A number of research results consider limited communication resources, while a lot have not even considered the possibility of leader failure. Because follower robots rely on their leader to get formation information, its failure means failure of the whole group to complete the given task. We propose a new approach to solve this problem which consist of the following: foremost, the assumption as reported in bulk of the work on leader-follower strategy is that robots in a particular group are homogeneous. This means we can make any of the robots to be a designated leader, so the challenge is what do to do with follower robots when failure condition of their leader is met. We allot ranks to each of the nodes in a particular formation so that robots can know the range of things they can do when on a particular node. We then design a control framework that autonomously assign a leader from the group robots at the beginning of the task and choose a new leader from the group of followers based on the node they occupy when failure occurs to the current leader, other robots then follow the new leader to complete the task. Should

in case we are able to recover the failed robot, our solution is designed such that robots joining a team that already has a leader will join as follower robots, therefore this recovered robot follows in the group complete the task. Our approach provides follower robots with the ability to overcome leader failure and offers the flexibility to easily introduce new robots to the group. This approach solves not only the problem of leader failure, but also, failure of any robot in the group, as it allows other robots to change node where necessary when a robot in the group fails, thanks to pliancy of ROS that made the implementation of this approach easier.

To achieve this, we use an existing Lyapunov-based tracking controller based on the backstepping technique from the literature to drive leader robot to track a reference trajectory, and follower robots to track a desired posture from the leader so they can all move in a defined formation. To ensure active robots do not collide with the failed robot during transitions, we propose avoidance algorithm using the limit-cycle strategy so robots can safely avoid obstacles present along the trajectory. Our avoidance algorithm is different from what is reported in [20–22] in the sense that their work assumes that obstacles are somewhere between robot and goal positions, contrary to trajectory tracking problems where same goal and robot's positions is expected. We also investigate the case of periodically switching the leader autonomously in a practical setting as an alternative, though not a lasting solution to problem of leader failure.

1.4 Synopsis

This thesis is organized as follows: Chapter 1 introduces the research topic, giving a brief background of multi-robot coordination. We briefly introduce formation control as an active research area in multi-robot systems, we stated our research objectives, statement of contribution, and lastly the thesis outline. We introduce some technical preliminaries in chapter 2, discussing the theory needed to achieve our research goals, then an overview of the hardware and software used for simulations and ex-

periment. In chapter 3, a new limit-cycle obstacle avoidance strategy is presented, together with leader-follower formation control with periodic switch. We then present simulations and experimental results to validate the effectiveness of our strategy. Assignment algorithm to fix the problem of leader failure is presented in chapter 4, along with simulations and experimental results to show the effectiveness of our algorithm. Summary and conclusions, and directions for future research come in the last chapter, chapter 5.

Chapter 2

Technical preliminaries

In this section, we present some concepts of graph theory, stability theory of nonlinear systems using the Lyapunov approach, and an overview of the tools and libraries used during our simulations and experiments.

2.1 Graph theory

A graph is an object that consist of a set of non-empty set of vertices and another set of edges. These edges may be directed or undirected depending on whether the edges have orientations or not. Example of a directed and undirected graphs is given in Figure 2.1 (a) and (b) respectively.

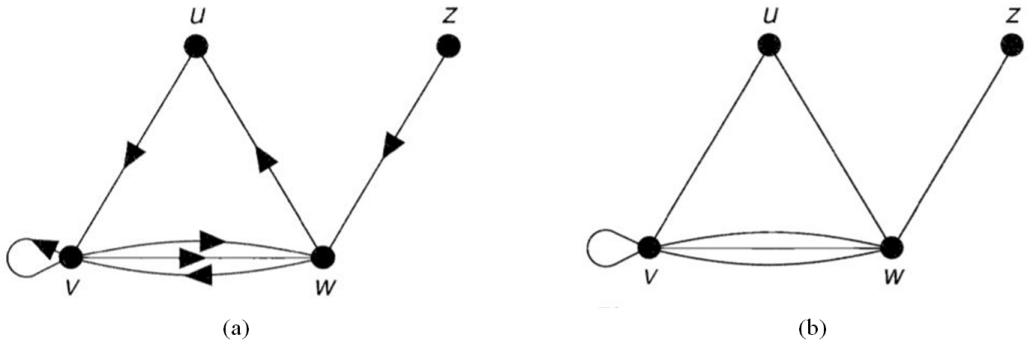

Figure 2.1: (a) Directed graph (b) Undirected graph [23].

With respect to leader-follower formation control technique, [24] describe the formation configuration as a directed graph or diagraph as the leader robot is used to

control the other follower robots in the formation. Also, specific formations lead to the development of unique diagraphs, and a particular set of shape variables associated with each graph structure. The authors added that sometimes in the presence of obstacles, it is necessary to switch from one diagraph to another which leads to internal dynamics of the graph, and hence, resulting to a graph that is non-isomorphic to the original graph.

In reference to [23], a directed graph or diagraph D, consist of a non-empty finite set V(D) of elements called vertices, and a finite family A(D) of ordered pairs of elements of V(D) called arcs. V(D) is the vertex set and A(D) is the arc family of D. An arc (v,w) is usually abbreviated to vw. Thus in Figure 2.1(a), $V(D)$ is the set (u, v, w, z) and $A(D)$ consists of the arcs $uv, vv, vw(twice), wv, wu, andzw,$ the ordering of the vertices in an arc being indicated by an arrow. If D is a digraph, the graph obtained from D by "removing the arrows" (that is, by replacing each arc of the form vw by a corresponding edge vw) is the underlying graph of D (see Figure 2.1(b)).

2.2 Lyapunov stability

Throughout this work, we use two distinct controllers from the literature for trajectory tracking control and obstacle avoidance control. The stability of these controllers is proved using the Lyapunov method, therefore, it is important at the this point to introduce Lyapunov stability in nonlinear systems. Before then, let us first introduce a time-dependent positive definite function. Consider a scalar function $W : D \times \mathbb{R}^+ \to \mathbb{R}$ with variables $x \in D$ and time t. Assuming this function is continuous and has continuous partial derivatives with respect to its arguments, then the function $W(x, t)$ is said to be positive semi definite in D if it satisfies the following conditions (see [25]):

i. $0 \in D$

ii. $W(0, t) = 0 \qquad \forall t \in \mathbb{R}^+$

iii. $W(x,t) \geq 0, \quad \forall x \neq 0, \quad x \in D$

$W(x,t)$ is said to be *positive definite* in D if conditions (i) and (ii) above are satisfied, and there exists a time-invariant positive definite function $V_1(x)$ such that: $V_1(x) \leq W(x,t) \quad \forall x \in D$.

Similarly, W(x,t) is said to be *negative definite (semi definite)* in D if -$W(x,t)$ is *positive definite (semi definite).*

In addition, W(x,t) is said to be *decrescent* in D if there exists a positive definite function $V_2(x)$ such that: $|W(x,t)| \leq V_2(x) \forall x \in D$. More so, $W(x,t)$ is *radially unbounded* if $W(x,t) \to \infty$ as $x \to \infty$ uniformly on t.

Now, consider the system $\dot{x} = f(x,t) \;\; f : D \times \mathbb{R}^+ \to \mathbb{R}^n$ and assume that the origin is an equilibrium state: $f(0,t) = 0 \forall t \in \mathbb{R}$. Then if in a neighborhood D of the equilibrium state $x = 0$ there exist a differentiable function $W(.,.) : D \times [0,\infty) \times \mathbb{R}$ such that:

i. $W(x,t)$ is positive definite.

ii. The derivative of $W(.,.)$ along any solution of $\dot{x} = f(x,t)$ is negative semi definite in D.

then, the equilibrium state is stable. Moreover, if $W(x,t)$ is also decrescent then the origin is uniformly stable.
The equilibrium state is uniformly asymptotically stable if

i. $W(x,t)$ is positive definite and decrescent.

ii. The derivative of $\dot{W}(x,t)$ is negative definite in D

If there exists a differentiable function $W(.,.) : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}$ such that:

i. $W(x,t)$ is positive definite, decrescent, and radially unbounded $\forall x \in \mathbb{R}^n$ and that

ii. The derivative of $\dot{W}(x,t)$ is negative definite in $\forall x \in \mathbb{R}^n$, then

the equilibrium state at $x = 0$ is globally uniformly asymptotically stable.

Suppose that the equilibrium state x=0 is uniformly asymptotically stable, and in addition assume that there exist positive constants K_1, K_2 and K_3 such that:

i. $K_1||x||^p \leq W(x,t) \leq K_2||x||^p$.

ii. $\dot{W}(x,t) \leq -K_3||x||^p$

Then the origin is exponentially stable. And if the conditions hold globally, then the equilibrium state $x = 0$ is globally exponentially stable.

2.3 E-puck2 robot

The e-puck2 is a small (7cm in diameter) differential drive robot developed at the Swiss Federal Institute of Technology in Lausanne (EPFL) in collaboration with GCtronic. The hardware and software of e-puck2 is fully open source, providing low level access to every electronic device and offering unlimited extension possibilities. It is powered by an STM32F4 microcontroller and features a large number of sensors: IR proximity, sound I/O, 9×IMU, ToF distance sensor, camera, and uSD storage. The robot is a full system with USB hub, debugger/programmer, Wi-Fi module [26]. Figure 2.3 shows what the e-puck2 robot is made up, featuring sensors, actuators, microcontroller, and few other components.

Due to its elegant design, flexibility, user friendly, and low-cost, a lot of research work has been implemented using this robot especially in the area of multi-agent systems. It is integrated with the Webots simulation software for programming, simulation, and control of the robot. An early implementation can be found in [27] where swarm of e-pucks are remotely controlled by external users over the internet using Web Services communication protocol. The available support for ROS makes it even more suitable for robotics research in multi-agent systems as it provides easy access

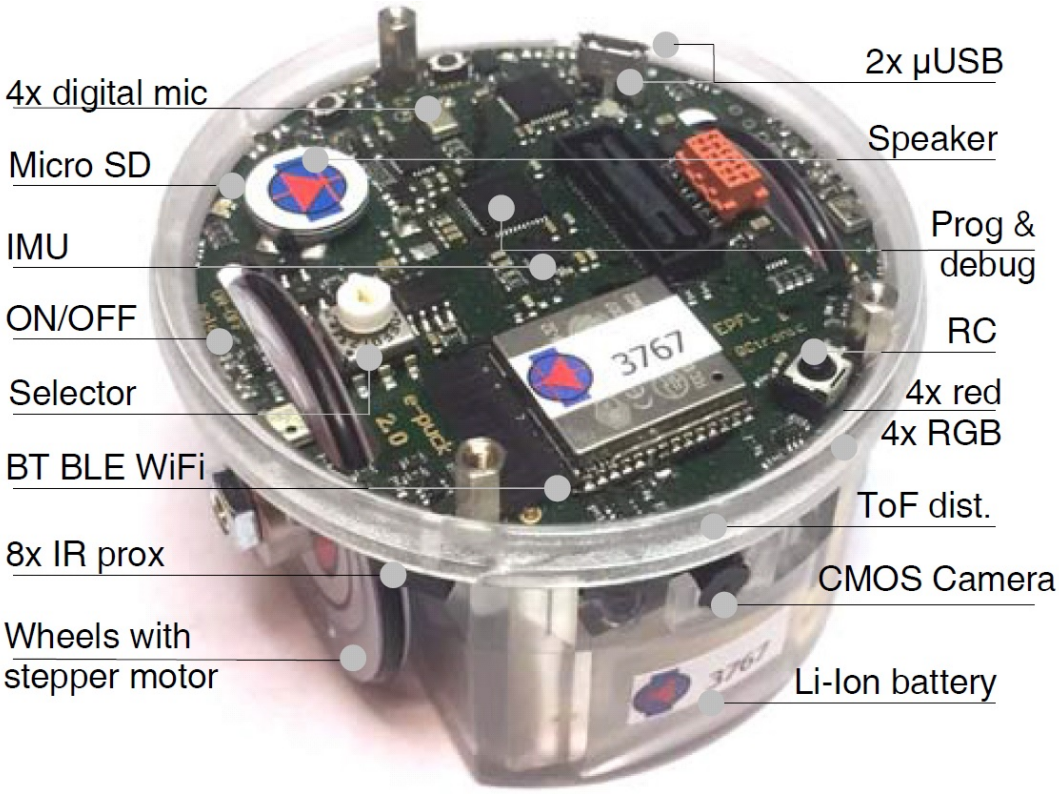

Figure 2.2: E-puck2 robot [26]

to the robot's sensors and actuators, as well as easy communication via Bluetooth or Wi-Fi among robots. Some lately research implementations can be found in [28] where four e-puck2 robots are used for cooperative localization using event-triggered mechanism with minimum communication exchange implemented using the ROS interface. In the reference [29], a Particle Swarm Optimization (PSO) is implemented on three e-puck2 robots with a camera placed on top of the environment to locate the robots using image processing technique and navigate each robot to its next position. This with abundance of references online make e-puck2 our ideal choice for all our experimental implementations in this thesis.

2.4 ZED Stereo camera

In multi-robot applications such as locomotion, path planning, where accurate posture of robots is essential, localization can be challenging especially when using small-sized

low-cost robots such as the e-puck2. One of the common strategies to localizing these robots is by using an overhead camera with some detecting algorithms that can detect markers mounted on the robots. An example is in [30] where an effective vision-based system is proposed to accurately track mobile robots' true pose using multiple overhead cameras. Their system can localize multiple mobile robots simultaneously in a 3m ×6m arena with each robot assigned with a symbol marker for identification. In [12], an overhead camera is used to localize a group of e-puck2 robots to address formation control problem using Model Predictive Control. During the experiment, pose estimates of the robots is sent to the PC in real time via the video camera attached for the computer to calculate optimal inputs to the robots.

In this work, we use the ZED stereo camera to localize our robots due to growing localization error observed from the odometry readings of the robots. ZED is a passive stereo camera that reproduces the way human vision works. Using its two "eyes" and through triangulation, the ZED creates a three-dimensional model of the scene it observes, introducing for the first time indoor and outdoor long range depth perception and positional tracking [31].

Figure 2.3: ZED Stereo camera [32]

Figure 2-3 shows the ZED stereo camera, it has an integrated 2.0m USB3.0 cable, with minimum system requirements of USB3.0 port, CUDA 6.5, NVIDIA GPU with compute capability greater than 2.0, 4GB RAM, Dual-core 2,3GHz, and windows 7 or

Ubuntu 14.04 or later to run the system development toolkit provided by StereoLabs. The camera can be interfaced with multiple third-party libraries and environments such as OpenCV, ROS, PyTorch, TensorFlow, MATLAB, just to name a few. We chose ZED camera to localize our robots because both can be interfaced with ROS, which makes it easier for the camera to communicate pose estimates to individual robots.

2.5 The Robot Operating System (ROS)

Robots need to communicate with the camera to get their respective pose data and communicate with other agents in the group to achieve formation, avoid obstacles, or perform any given task as a group. Therefore, a reliable system is required for robots to interact with the camera and other agents, and one of such systems is the Robot Operating System (ROS).

ROS is a modular open-source framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotics platform [33] . It provides functionality for hardware abstraction, communication between processes over multiple machines, and great tools for visualization. It also provides the flexibility to work with heterogeneous devices in a shared environment. Control of individual robots and communication between robots and other devices is realized using some software processes called "nodes" that can register with the ROS master node. They can execute tasks independently or by communicating with other nodes within the system. The communication mechanism used by ROS is through sending and receiving messages grouped into specific categories called topics. A message is defined by the type of message and data format, a node can send data by publishing it on a defined topic and receive data by subscribing to the topic of interest. Figure 2.4 shows the basic ROS model consisting of nodes registered with the ROS master communicating data via topics.

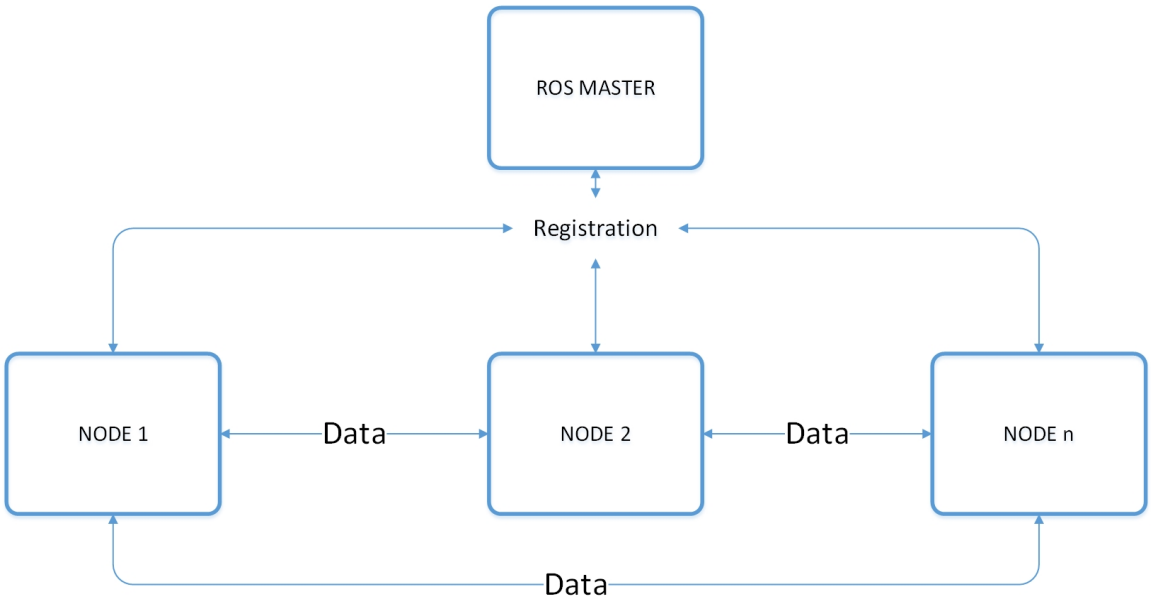

Figure 2.4: Basic ROS model

The modularity of ROS makes working with heterogeneous devices easier as additional functionalities can be easily incorporated to the platform. It also makes code reusability easier in robotics research as one can use packages from other projects to serve some purpose in another project. This along with several other reasons makes ROS the ideal framework to implement our algorithms. We carefully select packages that meets our requirements for implementation.

2.6 Markers

An important step to implementing our work is finding a way to accurately localize our robots. With the ZED camera, we use visual localization to estimate the pose of each robot within the field of view of the camera. Visual localization involves the problem of determining the camera pose of one or multiple query images in a database scene. This problem is highly relevant for a wide range of applications, including autonomous robots, augmented reality (AR), loop closure detection re-localization, SLAM, and Structure-from-Motion (SFM) systems [34]. The complexities associated with detecting our robots can be avoided using patterns designed to be reliably de-

tected by computer vision, such patterns are termed "fiducial markers" (see Figure 2.5). A lot of research papers have employed the use of fiducial markers to localize robots, or to improve localization accuracy. An example is in [35] where AR Tag markers are used for robot localization with an overhead camera viewing the robots from above with a unique marker mounted on each robot. To solve a formation control problem, [36] used overhead camera to determine the pose estimates of a group of e-puck2 robots by mounting patterns on each of the robots.

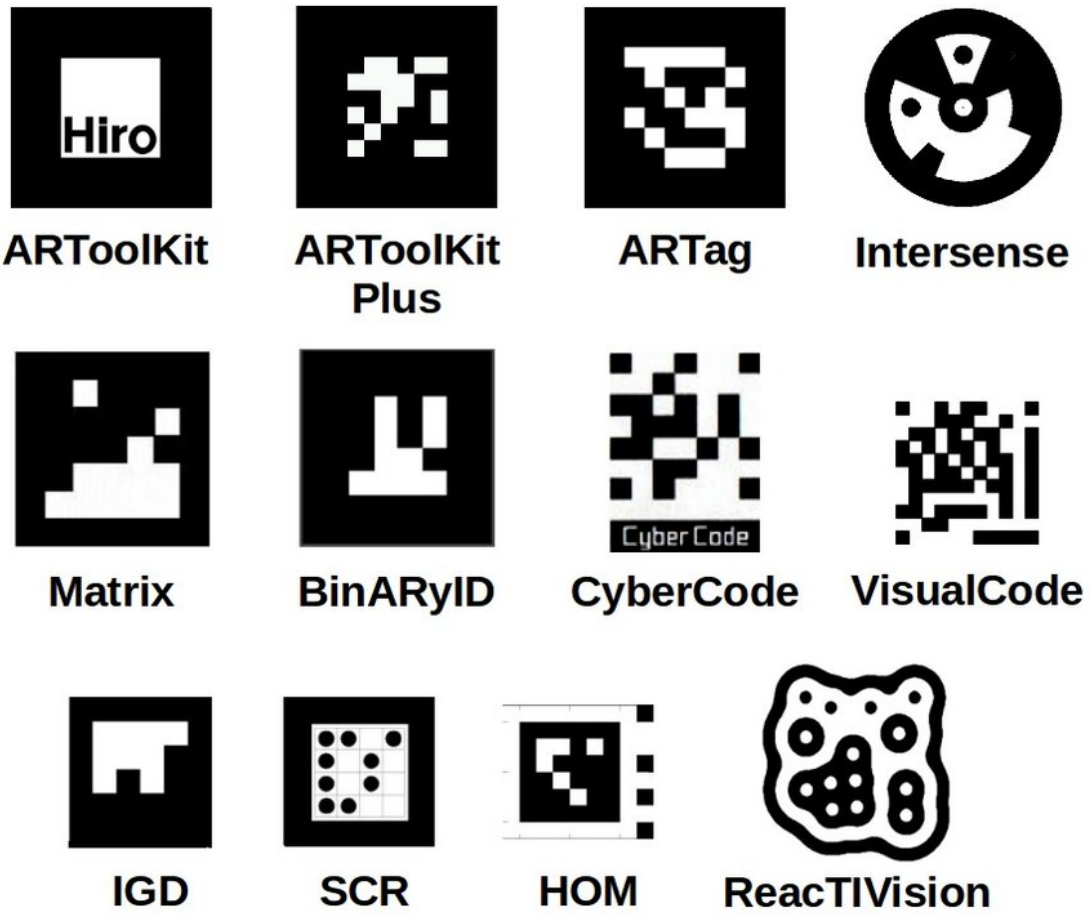

Figure 2.5: Example of fiducial markers [37]

In our work, we employ the use of AR Tag markers, a square pattern of size 5 by 5 printed on a flat surface with black and white patches, and relatively thick solid outer boundary (see Figure 2.6). These tags are provided by the *ar_track_alvar* package, which is a ROS wrapper for *alvar*, an open-source AR tag tracking library.

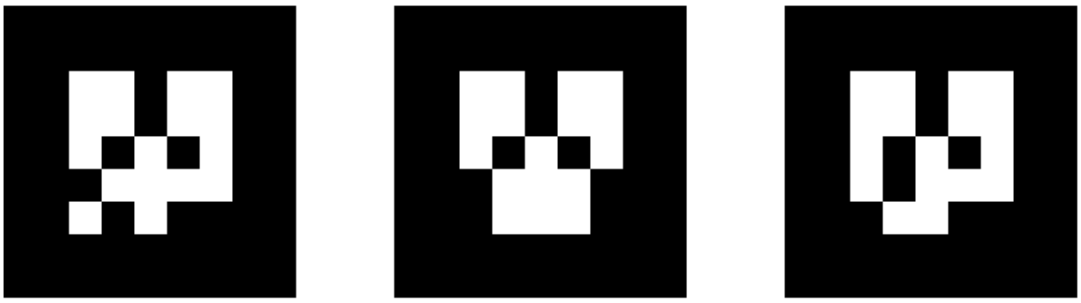

Figure 2.6: AR tags [38]

2.7 Drivers

We use dedicated ROS wrappers for our hardware devices to be able to receive data or send control commands using ROS. The e-puck2 ROS driver maintained by GCtronic, is designed to enable the use of e-puck2 with ROS, with a python and C++ versions available to enable user chose its preferred programming language. Robots can be connected to a computer via either Bluetooth or Wi-Fi, and all sensors and actuators are exposed to ROS so commands can be sent to the robot or receive data from it. Figure 2.7 is an *rqt_graph* showing all the topics published by the e-puck2 Wi-Fi ROS node. Although the node is publishing a lot of different topics for both sensors and actuators, the necessity to involve an external camera for localization mean we will be using just the */mobile_base/cmd_vel* topic to publish velocity command on the robot.

To be able to use the ZED stereo camera with ROS, we use the dedicated ZED ROS wrapper, which outputs the camera left and right images, depth map, point cloud, and pose. It also supports the use of multiple cameras. Since our interest is to localize AR markers (each mounted on a robot) within the field of view of the camera, we intend to use the *zed_ros_examples* repository – a subdivision of the ZED ROS wrapper – which will enable us to use the *ar_track_alvar* package. Figure 2.8 shows the *rqt_graph* of the camera using the *ar_track_alvar* package. In the list of all published topics by the tracking node, our topics of interest are the */zed/visualization_marker* and */zed/ar_pose_marker* which are only updated when a marker is visible. As seen in

the graph, the tracking node also updates the */tf* topic to have the pose of observed markers published in the TF Tree.

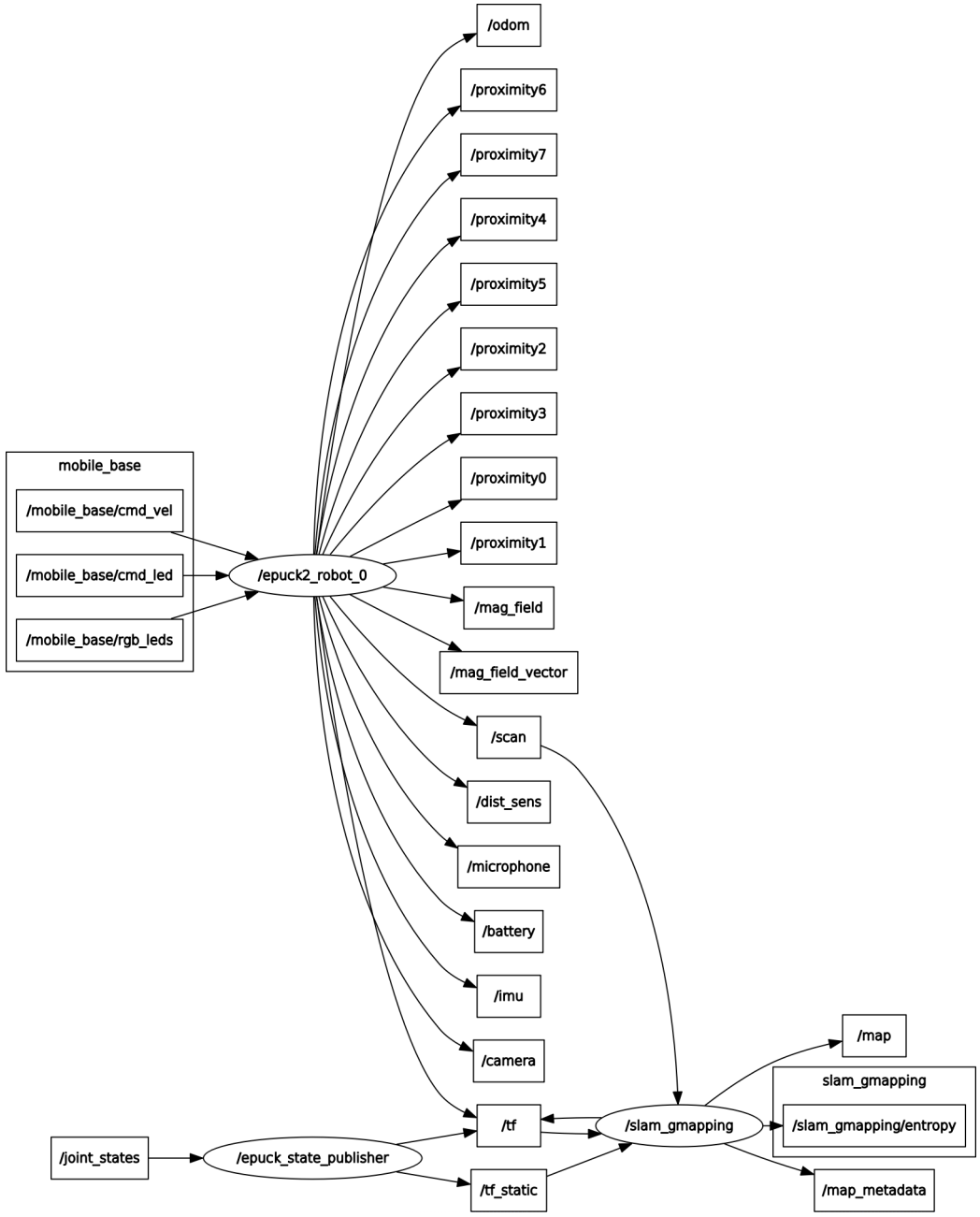

Figure 2.7: E-puck2 ROS topics [39]

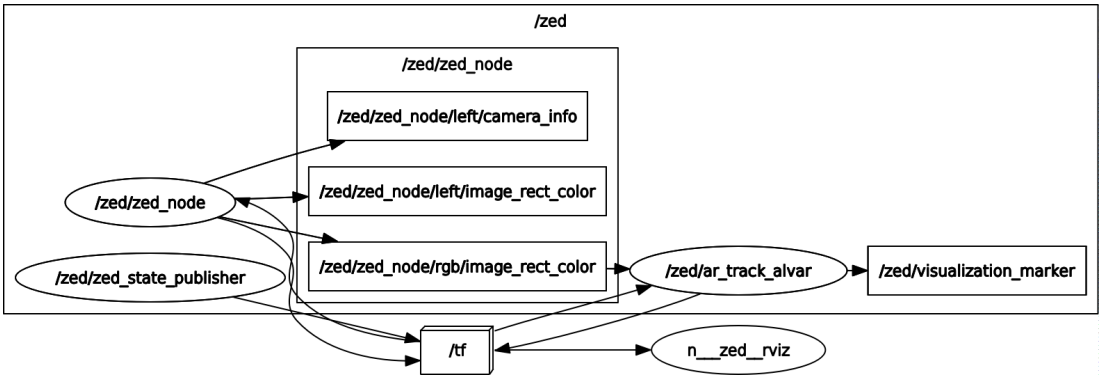

Figure 2.8: ZED camera RQT Graph Using AR Tags

Chapter 3

Formation control with obstacle avoidance and periodic leader switch

3.1 Introduction

This section presents formation tracking problem that involves the use of leader-follower formation control strategy coupled with obstacle avoidance algorithm. Most of the existing leader-follower strategy in the literature involve the use of dedicated leader(s) (see examples in [11–13]), and assumed an obstacle free environment, therefore did not involve obstacle avoidance. The work in [10] consider a case of leader switching, however, the leader is teleoperated using the ROS *teleop_twist_keyboard*. Followers are delayed until the new leader is ready to share formation data with followers, hence, they did not see the need to add obstacle avoidance. In this work however, we use tracking controller to drive all robots in the group to achieve and maintain formation. We also aim to periodically change the leader; hence, robots will break from existing formation to follow the new leader. Therefore, we find it necessary to integrate avoidance control in our strategy to ensure robots do not collide with one another during transition.

3.2 Robot dynamics

Consider a nonholonomic constrained mobile robot in Figure 3-1 whose dynamics are modeled by the following nonlinear ordinary differential equations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{r}{2} \begin{bmatrix} \dot{\varphi}_1 + \dot{\varphi}_2 \\ 0 \\ \frac{(\dot{\varphi}_1 - \dot{\varphi}_2)}{l} \end{bmatrix} \tag{3.1}$$

r represents the radius of the wheel, $\dot{\varphi}_i$ is the angle of rotation of wheel i, and l is the distance between each of the wheels. With $v = \frac{r(\dot{\varphi}_1 + \dot{\varphi}_2)}{2}$ and $\omega = \frac{r(\dot{\varphi}_1 - \dot{\varphi}_2)}{2l}$, we can rewrite equation 3.1 as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.2}$$

Here, $x \in \mathbb{R}$ and $y \in \mathbb{R}$ are the Cartesian coordinates of the center of mass of the vehicle, $\theta \in [0, 2\pi)$ is the heading of the robot with respect to the coordinate axis, v and ω are the control inputs for linear and angular velocity, respectively.

3.3 Trajectory tracking

Trajectory tracking control problem can be solved in a number of ways. Among the most popular techniques is the use of Model Predictive Control (MPC). In this technique, the trajectory tracking task is taken as predictive control problem with multi-constraints by transforming the continuous time system into a discrete state-space mode with fixed sampling period [40]. The references [12, 13] also employ the use of MPC to drive their robots to follow a reference leader. In their comparative study, [40] demonstrate the accuracy of the controller but conclude that it is prone to instability under harsh driving conditions. Another popular method is the Lyapunov based approach, it involves the use of a Lyapunov function to design control inputs to

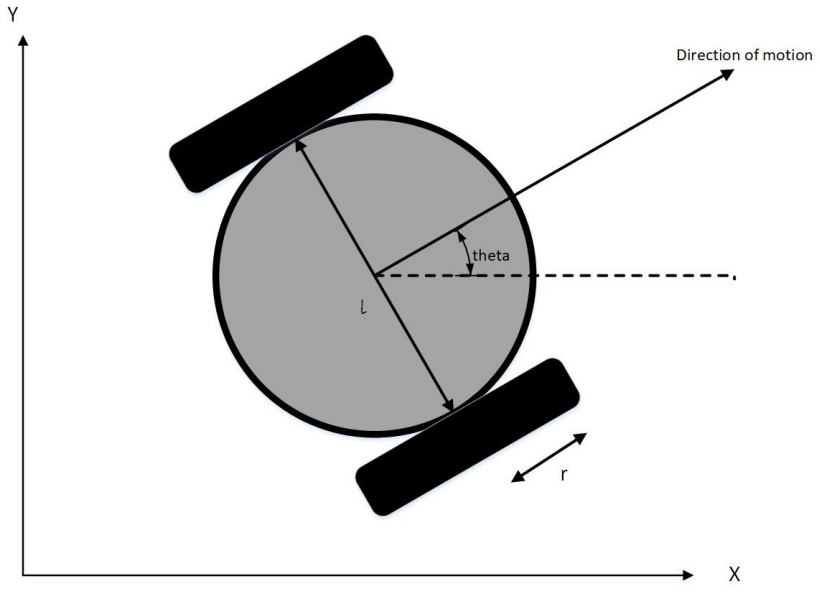

Figure 3.1: Robot schematic

make the zero equilibrium of the system stable. A lot of references (examples include: [10, 17, 41, 42]) have reported the use of a Lyapunov based controller to drive robots to track a reference trajectory. In this work, controller design is not our main focus, we therefore consider using the backstepping controller designed in [42] to drive our robots.

Consider a tracking control problem where a robot is tasked to follow a reference robot with a posture $P_r = (x_r, y_r, \theta_r)$ and reference control inputs v_r and ω_r.

Consider Figure 3.1 where robot's posture (x, y, θ) is given, the error coordinates can be denoted by (see [41]):

$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{3.3}$$

The error dynamics are (see [41]):

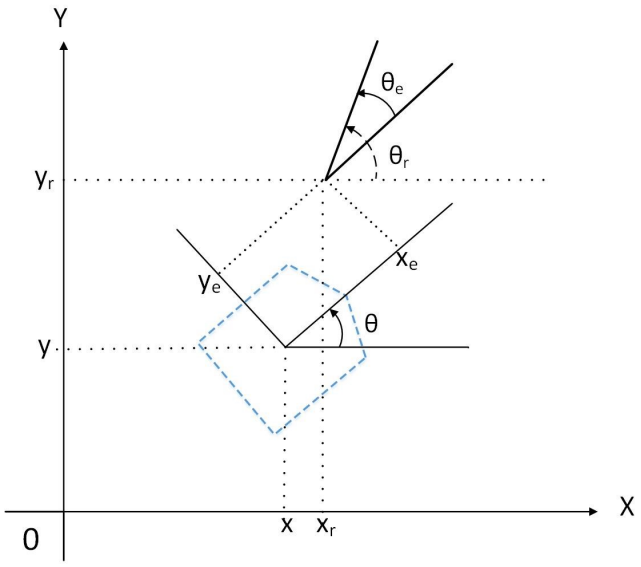

Figure 3.2: Reference and current posture [42]

$$\begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} v_r cose_\theta + \omega e_y - v \\ v_r sine_\theta - \omega e_x \\ \omega_r - \omega \end{bmatrix} \tag{3.4}$$

A Lyapunov stable time-varying state-tracking control law is designed in [42] based on backstepping technique to control robot to track reference trajectory. According to [42], if $\dot{v}_r, \dot{\omega}_r, v_r$,and ω_r and bounded with the assumption that either v_r or ω_r does not converge to zero, the global asymptotic stability of the error dynamics in equation 3.4 is guaranteed, and thus errors converge to zero using the control inputs in equation 3.5 (see [42] for proof of convergence).

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v_r cose_\theta + k_x e_x \\ \omega_r + \frac{k_y e_y sine_\theta}{e_\theta} + k_x e_\theta \end{bmatrix} \tag{3.5}$$

with $a = \sqrt{v_r^2 + b\omega_r^2}$, $k_x = 2\epsilon a$, $k_y = b|v_r|$, $\epsilon > 0$ and $b > 0$.

3.4 Leader-follower formation control

Leader-follower formation control involve the use of a robot called the "leader" to generate trajectory information for other robots in the team called "followers". With each follower robot tracking the leader at a prescribed desired distance and bearing, leader-follower formation control problem can be viewed as a natural extension of trajectory tracking problem. The references [10, 14] formulated formation control problem as a trajectory tracking problem by defining a desired posture $P_d = (x_d, y_d, \theta_d)$ using parameters l and ϕ as desired distance and bearing, respectively, relative to the posture of a leader robot or center of mass of the desired formation. This formulation is defined by equations 3.6-3.8 below (see [10, 14]).

$$x_d = x_l + lcos\,(\phi + \theta_l) \tag{3.6}$$

$$y_d = y_l + lsin\,(\phi + \theta_l) \tag{3.7}$$

$$\theta_d = \text{atan2}(\dot{y}_d, \dot{x}_d) + k\pi, k = 0, 1 \tag{3.8}$$

where $k = 0; 1$ defines the desired drive direction (0 for forward and 1 for reverse motion) and atan2 is the four-quadrant inverse tangent function. As outlined in [10], it is possible for a nonholonomic constrained mobile robot to track a reference trajectory if equations 3.6-3.8 respect the nonholonomic constraint of the robot, meaning it should be consistent with the following form:

$$\begin{bmatrix} \dot{x}_d \\ \dot{y}_d \\ \dot{\theta}_d \end{bmatrix} = \begin{bmatrix} cos\theta_d & 0 \\ sin\theta_d & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} \tag{3.9}$$

Therefore, [10] chose the desired orientation as in equation 3.8 for the desired trajectory to satisfy 3.9.

Given posture of the follower robot $P_f = (x_f, y_f, \theta_f)$ and that of the generated desired posture $P_d = (x_d, y_d, \theta_d)$, formation is achieved if each follower robot tracks its corresponding desired trajectory generated by the leader robot. Mathematically, formation is achieved if $\lim_{t\to\infty}(x_d - x_f) = 0$ and $\lim_{t\to\infty}(y_d - y_f) = 0$, the condition $\lim_{t\to\infty}(\theta_d - \theta_f) = 0$ is desirable but not necessary, however must be bounded. The desired velocities v_d and ω_d are derived from equation 3.9 as:

$$v_d = \pm\sqrt{\dot{x}_d^2 + \dot{y}_d^2} \tag{3.10}$$

$$\omega_d = \dot{\theta}_d = \frac{\dot{x}_d\ddot{y}_d + \dot{y}_d\ddot{x}_d}{\dot{x}_d^2 + \dot{y}_d^2} \tag{3.11}$$

The sign $\pm$ depends on whether the motion is forward or backward. Note that the necessary conditions for reference path design according to (Jiang & Nijmeijer, 1997) is a nonzero desired linear velocity ($v_d \neq 0$) and a smooth twice differentiable path. Similar to equation 3.5, the control inputs in equation 3.12 will drive the follower agents to the desired position P_d to achieve desired formation.

$$\begin{bmatrix} v_f \\ \omega_f \end{bmatrix} = \begin{bmatrix} v_d cose_\theta + k_x e_x \\ \omega_d + \frac{k_y e_y sine_\theta}{e_\theta} + k_x e_\theta \end{bmatrix} \tag{3.12}$$

with $k_x = 2\epsilon a$, $k_y = b|v_d|$, $a = \sqrt{\omega_d^2 + bv_d^2}$, $\epsilon > 0$ and $b > 0$.

3.5 Obstacle avoidance

3.5.1 Theory

We want robots to avoid static obstacles along their respective trajectories and avoid collision with other robots when switching roles in case of failure of leader failure. Several avoidance strategies can be found in the literature. An interesting overview of some of these strategies is given in [43]. References [14, 17] addressed avoidance control using avoidance function defined in [44, 45], which is active only in the bounded

sensing regions of individual robot and do not interfere with robot's individual optimal control law outside of this region. Another notable approach is the limit-cycle avoidance strategy proposed by [20]). This work employs a limit-cycle avoidance strategy to safely avoid obstacles while cruising.

In limit-cycle avoidance strategy, robot needs to follow accurately limit-cycle vector fields defined by the following differential equations (see [20, 21]):

$$\dot{x}_s = (sign)y_s + x_s\left(R_c^2 - x_s^2 - y_s^2\right) \tag{3.13}$$

$$\dot{y}_s = -(sign)x_s + y_s\left(R_c^2 - x_s^2 - y_s^2\right) \tag{3.14}$$

Here, (x_s, y_s) is the position of the robot according to the center of convergence of the limit-cycle, R_c defines the radius of the limit-cycle, and "$sign = \pm 1$" for clockwise and counter-clockwise avoidance, respectively. Figure 3.3 shows the phase portrait of limit-cycle if radius $R_c = 1$ with directions of trajectories - clockwise limit-cycle (left) and counter-clockwise limit-cycle (right) - according to the $x_s and y_s$ axis. The trajectories from all points (x_s, y_s) including inside the circle, move toward the circle.

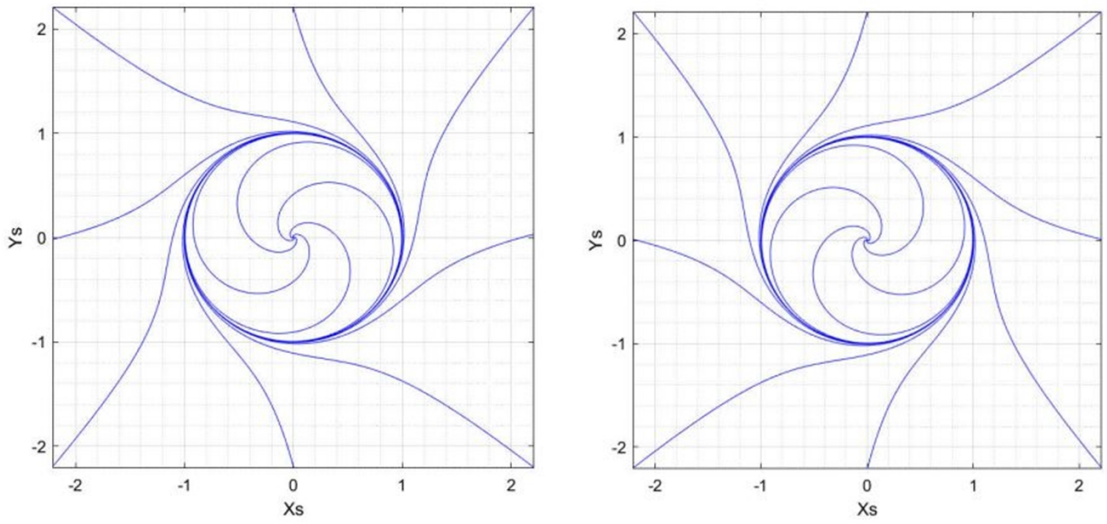

Figure 3.3: Phase portrait of the limit-cycle defined by equations 3.13 and 3.14.

To control robot to follow these trajectories when an obstacle is detected, references [21, 22] used what is called the orientation control. The robot desired orientation is derived from the differential equation of the limit-cycle as:

$$\theta_d = \text{atan2}(\dot{y}_s, \dot{x}_s) \tag{3.15}$$

And the error is given by

$$\theta_e = \theta_d - \theta \tag{3.16}$$

It is proved in [21] that with $k > 0$, the input $\omega = \dot{\theta}_d + k\theta_e$ will drive the robot to the desired orientation, and with a constant nominal speed v, the robot will be able to accurately follow the limit-cycle.

3.5.2 Algorithm

The avoidance algorithm in [21, 22] assumed obstacle to be somewhere between the robot and the target destination. This is not the case in trajectory tracking problem where a near zero tracking error is expected, hence, robot's and goal positions are approximately the same. We therefore propose a new avoidance algorithm to enable the use of this strategy in trajectory tracking applications. To avoid obstacle, the following are necessary:

- Detect obstacle to avoid.
- Define escape criterion which determine if an obstacle is completely avoided or not.

We define D_j, the Euclidean distance between agent and obstacle j. Choose obstacle with the least distance to the robot, this obstacle is characterized by its position (x_o, y_o) and radius R_o. Given position (x, y) of the robot, the obstacle's position (x_o, y_o) is the center of the limit-cycle. Expressing the robot's position (x_s, y_s) with respect to the center of convergence of the limit-cycle, we have: $x_s = (x - x_o)$ and

$y_s = (y - y_o)$. D_j is defined by equation 3.17, the radius of the region of influence R_i and that of the limit-cycle R_c are defined by equations 3.18 and 3.19 respectively:

$$D_j = \sqrt{x_s^2 + y_s^2} \tag{3.17}$$

$$R_i = R_o + R_r + \delta \tag{3.18}$$

$$R_c = R_i - \varepsilon, \varepsilon < \delta \tag{3.19}$$

R_r is the radius of the robot, δ is a safe margin for collision avoidance, and ε relatively small positive number. Because we are dealing with dynamic obstacles (other agents), the sign in equations 3.13 and 3.14 is set to -1 (counterclockwise avoidance). A robot approaching an obstacle from any direction can have coordinate positions (x_s, y_s) to be either positive or negative values (see Figure 3.4). This is used to define escape criterion by keeping track of the signs of x_s and y_s.

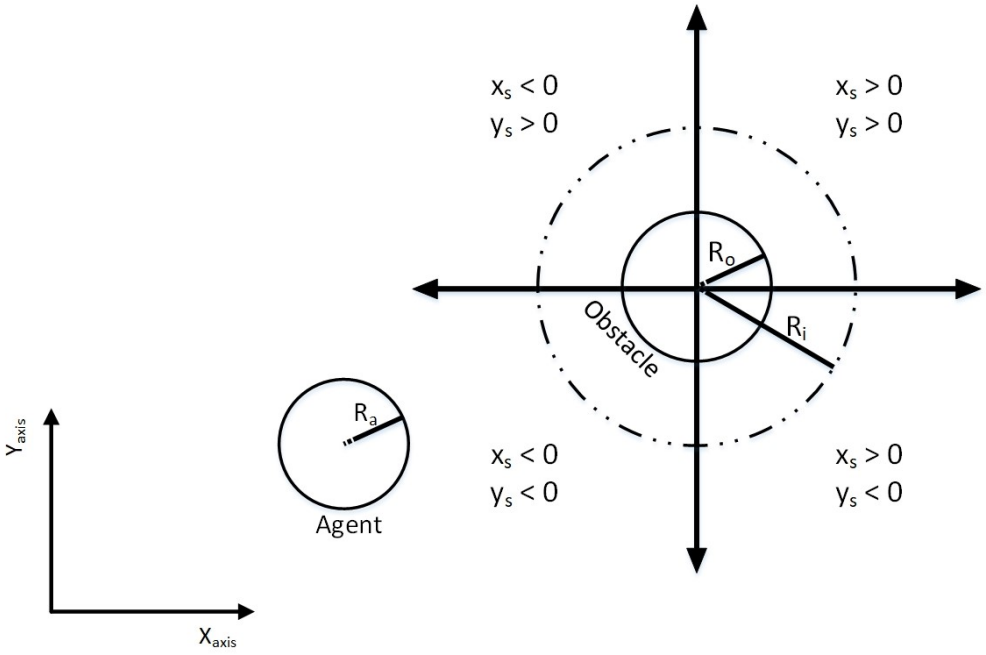

Figure 3.4: Definition of robot's position relative to an obstacle.

Define sgn_{xs} and sgn_{ys} as the signs of x_s and y_s respectively, just after the robot enters the circle of influence of the obstacle, and let n_sgn_{xs} and n_sgn_{ys} be the signs

of x_s and y_s respectively, when the robot is avoiding the obstacle. We compare (sgn_{xs}, sgn_{ys}) and $\left(n\_sgn_{xs}, n\_sgn_{ys}\right)$ and say that an obstacle is completely avoided and switch to tracking controller when the conditions "$sgn_{xs} \neq n\_sgn_{xs}$" and "$sgn_{ys} \neq n\_sgn_{ys}$" are satisfied.. The pseudo code for the orbital obstacle avoidance algorithm is as follows:

Algorithm 1 : Orbital obstacle avoidance

```
1 if D_j ≤ R_i then
2     Avoidance controller
3     if n_sgn_xs ≠ sgn_xs & n_sgn_ys ≠ sgn_ys then
4         Tracking controller
5     else:
6         Avoidance controller
7 else:
8     Tracking controller
```

3.6 Hierarchical action selection algorithm

We have two distinct controllers at our disposal and can activate only one at a time. Tracking controller is the primary while avoidance controller is active only when an obstacle that can obstruct the robot's motion is detected. Figure 3.5 show how these two controllers would be managed while guaranteeing stability of the overall control. While reactive approach in [46] and [47] activate avoidance controller only when the robot is close to an obstacle, the hierarchical action selection approach introduced in [20–22, 48] activate avoidance controller as soon existence of an obstacle that can obstruct motion of the robot is detected, thus reducing the amount to time needed to reach the target. Inspired by [21], we activate the tracking controller as soon as the robot enters the region of influence of radius R_i of the obstacle.

Figure 3.6 depicts robot outside the region of influence of an obstacle of radius R_o and region of influence of radius R_i. Given D_j, the Euclidean distance between a robot and the nearest obstacle j, the pseudo code for the hierarchical action selection algorithm is as follows:

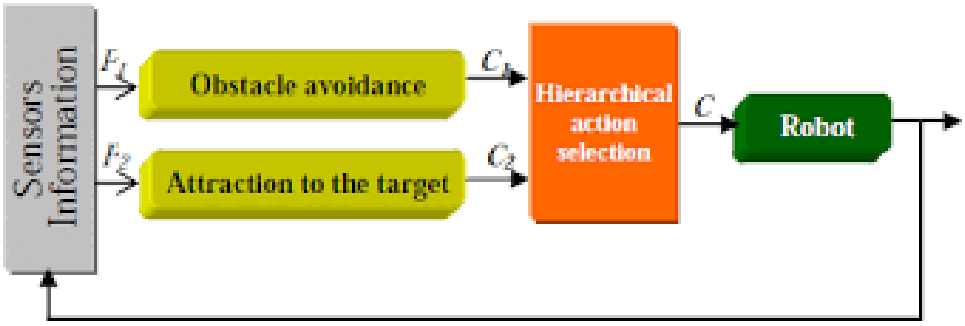

Figure 3.5: Choice of controller at an instance [21]

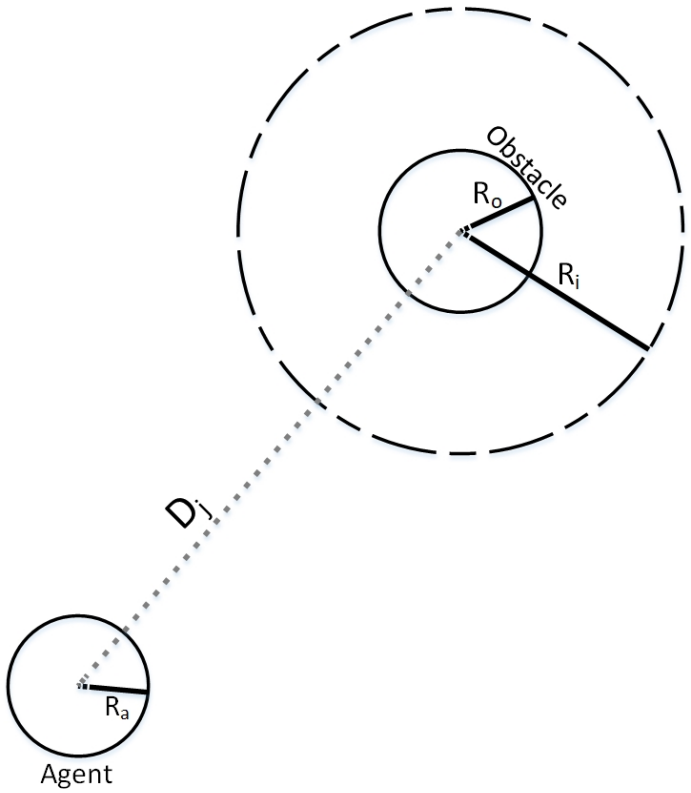

Figure 3.6: Robot-Obstacle setting showing the region of influence of the obstacle.

3.7 Implementation

What follows in this section presents the implementation of our limit-cycle obstacle avoidance algorithm, along with formation control with autonomous periodic leader switch using simulations and experiment.

Algorithm 2 : Hierarchical action selection

```
if D_j ≤ R_i then
    Avoidance controller
else:
    Tracking controller
```

3.7.1 Simulations

In this section, simulations are presented to verify the performance of our avoidance algorithm. In addition, we present results for formation control with periodic leader switch looking at cases regarding number of robots in the group and type of the trajectory. To run our simulations, we use Dell G7 15 laptop with NVIDIA GPU (GeForce RTX 2070 with Max-Q design). We then install and integrate GAZEBO, an open-source 3D robotics simulator to ROS melodic, running on Ubuntu 18.04. It is necessary for robot to be aware of its pose as well as where the rest of the world (obstacles and other robots) is in relation to itself. In our simulations, position of obstacles is provided by the */gazebo/model_states* topic. This topic publishes a list of positions of all models on the ground plane, therefore, we subscribe to each of the model's position by accessing the data of its index in the list. We get the robots' pose estimates through odometry data by subscribing to their respective */odom* topic.

3.7.1.1 Obstacle avoidance

We run a series of simulations to verify the effectiveness of our avoidance algorithm. We consider the following cases:

- **Single obstacle along a circular trajectory:** Consider a nonholonomic constrained mobile robot on a ground plane whose initial position is chosen at random relatively close to the trajectory (see Figure 3.7). The goal is to track a given reference circular trajectory and simultaneously avoid an obstacle present along the trajectory. We define a trajectory in equation 3.20 of radius $r = 4m$ centered at $(x_c, y_c) = (7, 7)$ on the ground plane. We task the robot to complete

the circle over a period of $T = 160s$. The obstacle present along the trajectory is of radius $0.14m$. We set the control parameters for the tracking controller to be $\epsilon = 0.9$ and $b = 2$, while the parameter for the avoidance controller is set for $k = 1.8$

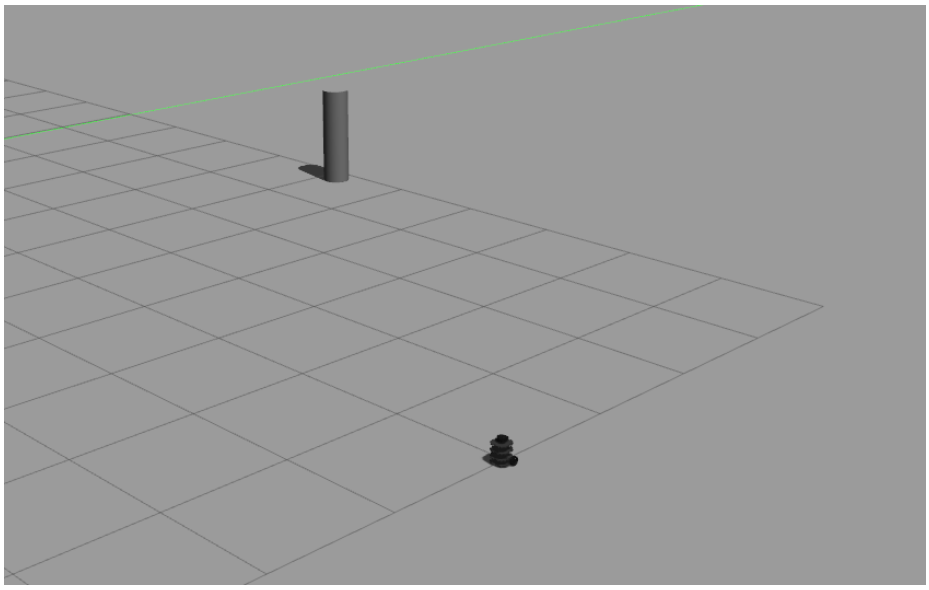

Figure 3.7: Robot and obstacle on a ground plane.

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} + r \begin{bmatrix} cos\alpha t \\ sin\alpha t \end{bmatrix}, \quad \alpha = \frac{2\pi}{T} \tag{3.20}$$

Figure 3.8(a) shows how the robot is able to detect and avoid the obstacle during the trajectory tracking, while 3.8(b) displays tracking errors converging to zero after the obstacle is avoided.

- **Singe obstacle along a linear trajectory:** In this case, we use the same parameters as in case 1 for both controllers. With two obstacles present at positions $(5, 5)$,and $(10, 10)$ respectively, the robot's initial pose is set for $(0.5, 1, \frac{\pi}{4})$. The trajectory is a straight line of gradient 1, defined by equation 3.21, and the robot is to track it over a period of $T = 100s$.

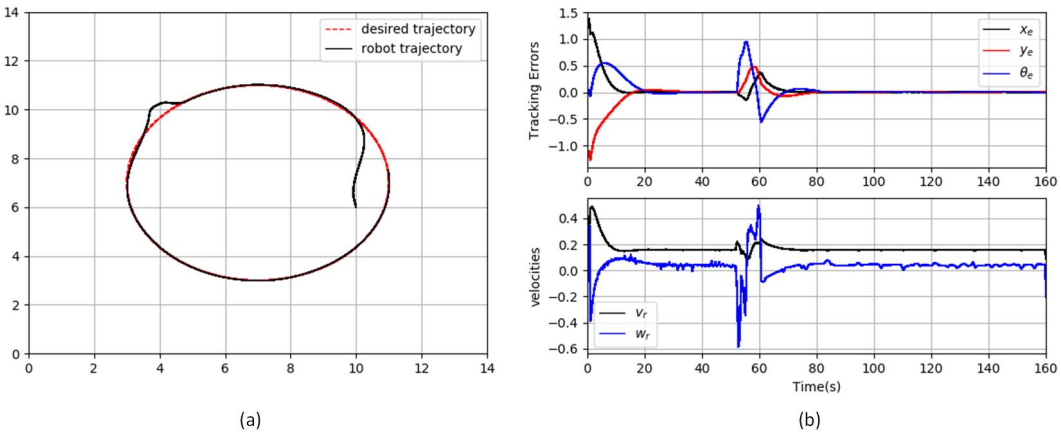

Figure 3.8: (a) Desired and robot's trajectories, (b) Tracking errors and control inputs.

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} \alpha t \\ \alpha t \end{bmatrix}, \quad \alpha = \frac{4.14\pi}{T} \tag{3.21}$$

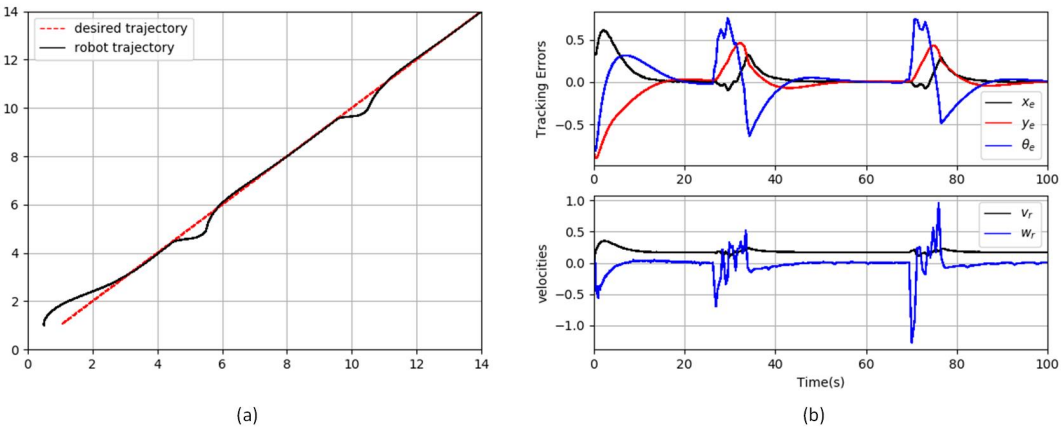

Figure 3.9: (a) Robot's and reference trajectories, (b)Tracking errors and input velocities.

The starting point of the trajectory is set for $(x_i, y_i) = (1, 1)$. Note that the robot's initial posture is randomly chosen somewhere near (x_i, y_i), the obstacles' positions are chosen so that they can obstruct the robot's motion along the trajectory. It can the seen in Figures 3.9(a) and 3.9(b) that the robot avoids collision with both obstacles successfully while maintaining roughly zero tracking

error in the absence of obstacles.

Multiple obstacles along circular trajectory: Since we will be dealing with multiple robots, we consider the case of having more than just two obstacles along the trajectory. The aim here is to verify that the robot can detect each of the individual obstacles and avoid only the one that can obstruct its motion, and as well with the minimum distance to the robot. Figure 3.10(a) depicts the robot avoiding three different obstacles (each with dimensions the same as the first case) positioned at $(11, 7), (3, 7)$, and $(7, 3)$, respectively. It can be observed from Figure 3.10(b) that the tracking error grows in the presence of an obstacle, which asymptotically converge to approximately zero when the obstacle is completely avoided, that is, when the tracking controller is active.

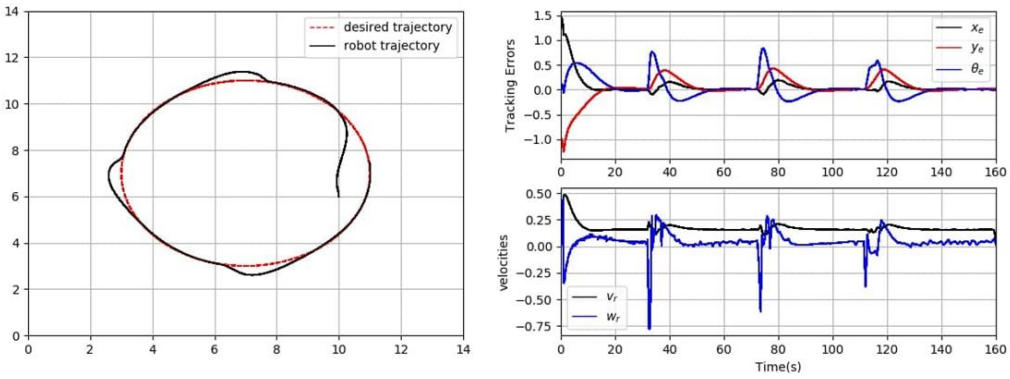

Figure 3.10: Trajectory tracking with multiple obstacle avoidance.

3.7.1.2 Formation control with periodic leader switch

As discussed in the previous sections, leader-follower formation control strategy is easy to implement, but suffers a major practical drawback of leader failure, resulting in followers' inability to complete the given task. One way to solve this problem is by periodically changing the leader, this is different from what the reference [10] presented in the sense that new leader selection is automatic, all robots are controlled using a single workstation, and as well, our leader robot is controlled by tracking and

avoidance controllers. At each transition, the new leader breaks out of the formation to continue with the trajectory tracking, and of course, avoid collision with other robots. Followers and former leader then take new positions with respect to the new leader and move in a formation. To demonstrate this, we consider two cases as follows:

- **Three robots in a circular trajectory:** Consider three robots on a ground plane tasked with tracking a circular trajectory in a defined formation shape. We intend to periodically switch the team leader such that each of the robots leads the group for one-third of the period. The Lyapunov based tracking and avoidance controllers are implemented on each of the robots. The group is required to form an equilateral triangle of length $0.7m$, that is, with one of the robots leading the team, the two follower robots are to follow the leader at a distance of $0.7m$ and bearings $\frac{5\pi}{6}$ and $\frac{7\pi}{6}$ respectively. We use the same trajectory defined by equation 3.21 centered at $(7, 7)$ with radius 4m over a period of $T = 180s$. Figure 3.11 shows the robots at random initial positions and a tapped image of the robots moving in a formation.

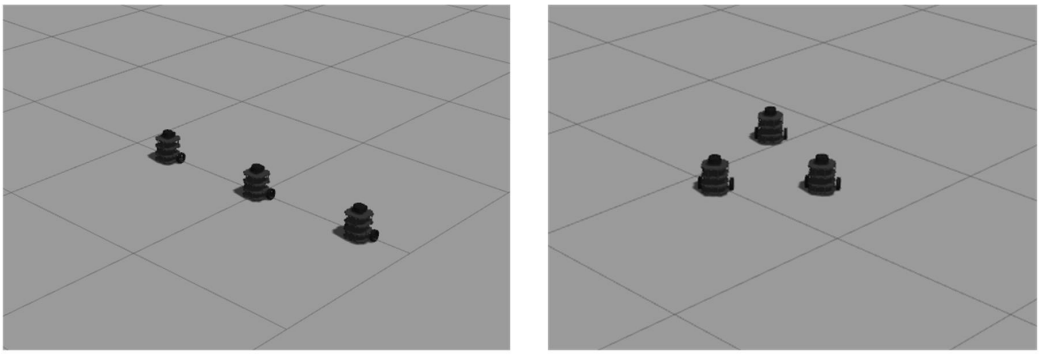

Figure 3.11: (a) Robots at initial position, (b) Robots moving in a formation

As shown in Figure 3.12, robot0 leads the group for the first one-third of the period with robots 1 and 2 following robot0 at bearings $\frac{7\pi}{6}$ and $\frac{5\pi}{6}$, respectively. Robot1 then autonomously takes the leadership in the second one-third with robots 2 and 0 following robot0 at bearings $\frac{7\pi}{6}$ and $\frac{5\pi}{6}$ respectively, and in the

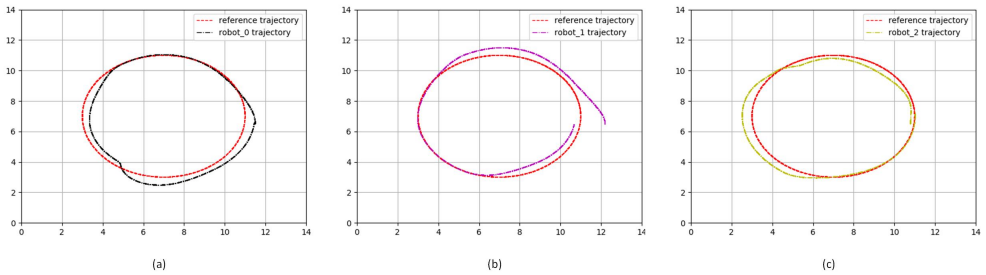

Figure 3.12: Trajectories of (a) Robot0, (b) Robot1, and (c) Robot2.

last one-third robot2 leads while robots 0 and 1 follow. It can be seen from the plots that robots leading the team at a particular time track accurately the trajectory, while follower robots follow the desired trajectory received from the leader.

- **Four robots in a linear trajectory:** This case is a lot similar to the previous case, but in this case, four robots are involved in tracking a linear trajectory defined by equation 3.21 in a square formation shape for a period of 160 seconds, with each robot to leading the team for 40s. The formation is of length 0.6m, and bearings $\frac{5\pi}{6}, \pi$ and $\frac{7\pi}{6}$ from the leader.

  Figure 3.13 shows robot3 following robot0 at bearing of $\frac{5\pi}{6}$, robot1 at a bearing of π, and robot2 at a bearing of $\frac{7\pi}{6}$ during the first, second, and third quarter respectively, and finally leading the team in the last quarter. The case is similar for robots 0, 1, and 2 respectively.

3.7.2 Experiment

Being that uncertainties are marginally handled in simulations, we run experiments using low-cost robots to demonstrate our avoidance algorithm and periodic leader switch working on real robot using four e-puck2 robots. The robot does not have laser sensor, or an inbuild camera that can provide us with point cloud data. It only offers a short laser reading of just about 8cm derived from interpolating data from six of its proximity sensors on the front side. This means we can not use this robot for

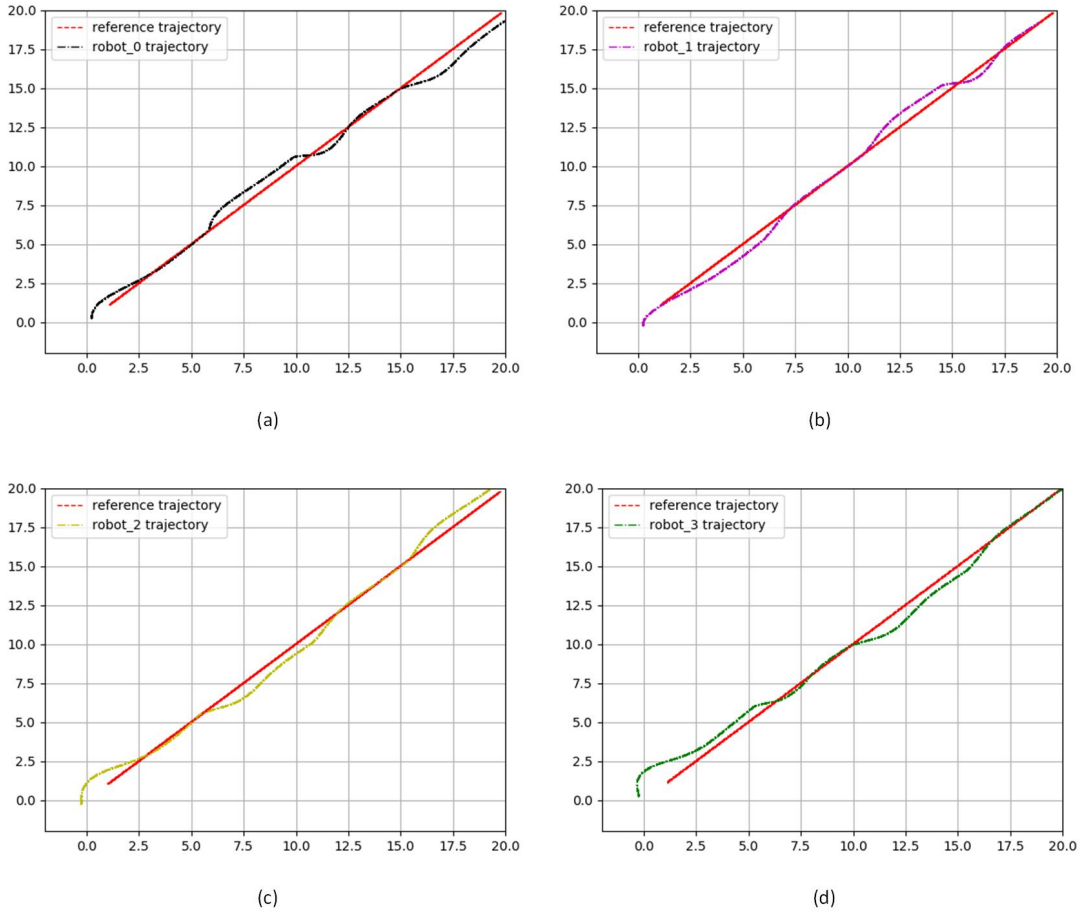

Figure 3.13: Trajectories of (a) Robot0, (b) Robot1, (c) Robot2, and (d) Robot3.

simultaneous localization and mapping (SLAM) in a large environment. We employ the use ZED Stereo camera to provide the pose estimate of each of the four robots. We mount a unique AR Tag on each of the robots so the camera can detect it using the *ar_track_alvar* package. We use *zed_ar_track_alvar* driver to get pose estimates of the AR tags from the camera, and C++ Wi-Fi version of the ROS e-puck2 driver to get sensor readings from the robots as well as ROS topics for the control script to publish commands on actuators.

Figure 3.14 depicts the whole setup consisting of a Dell G7 workstation with Ubuntu 18.04 and ROS Melodic installed on it, a ZED Stereo camera mounted at about 2m above the ground, four e-puck2 robots each with a unique AR tag mounted

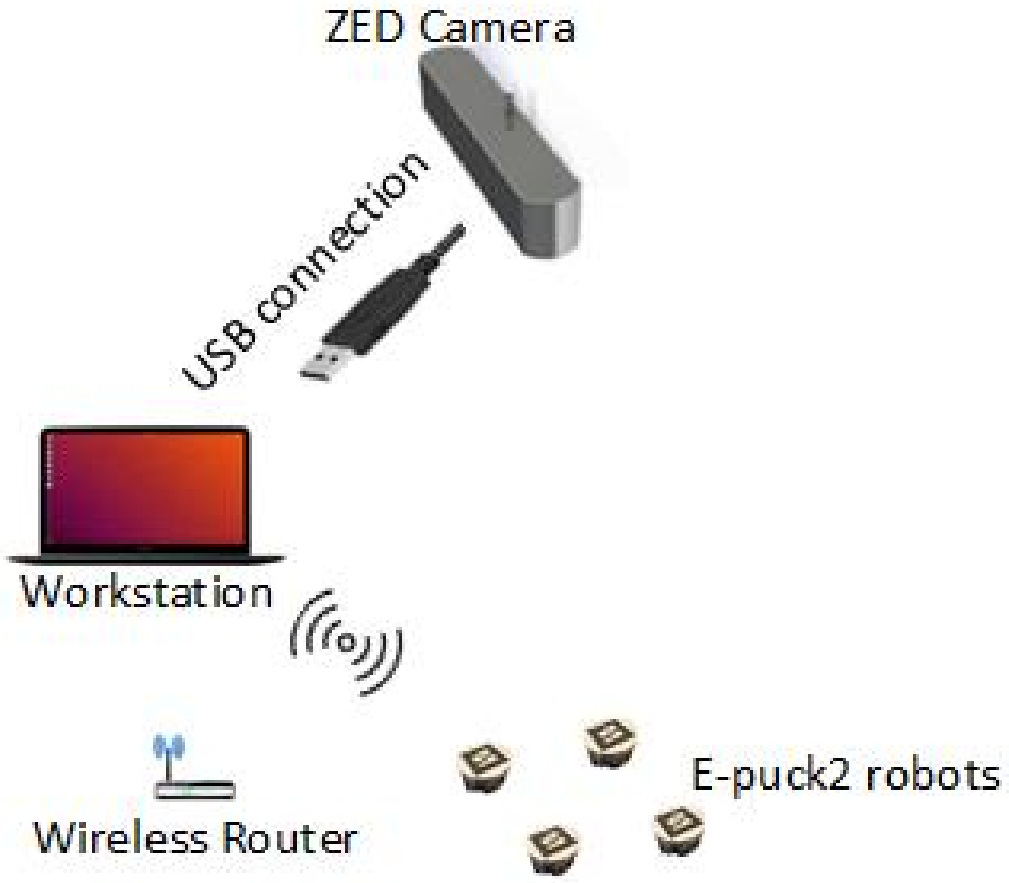

Figure 3.14: Experimental setup.

on top, and a D-Link router that connects the robots and the workstation via wireless communication. Throughout the experiment, the parameters for the tracking controller are set for $\epsilon = 0.9$ and $b = 10$, while the parameter for the avoidance controller is set for $k = 2.2$. Due to the limited view of the camera in the setup, we use circular trajectory defined by equation 3.20 throughout the experiment. The trajectory is centered at $(0, 0)$ with radius $r = 0.4m$ and period $T = 180s$. Figure 3-15 is the physical network setup for all experiments to be reported in this thesis.

3.7.2.1 Obstacle avoidance

To demonstrate obstacle avoidance, we use two e-puck2 robots 0 and 1, with robot0 tracking a circular trajectory and robot1 static at a random position along the tra-

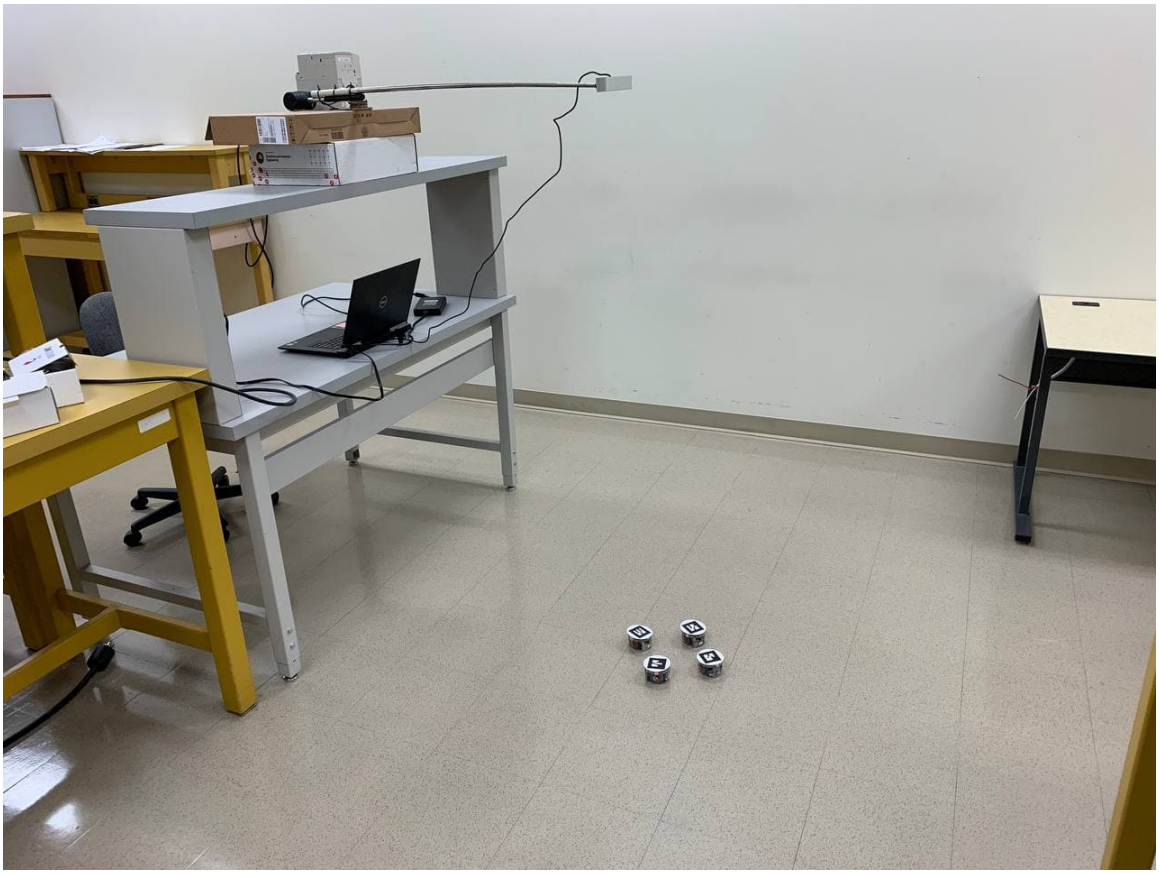

Figure 3.15: Workspace for our experiments.

jectory. As shown in Figure 3.16, robot 0 was able to detect and avoid robot1 located at approximately $(-0.18, -0.36)$ along the trajectory, then switched to tracking after avoidance is complete. Figure 3.17(a) shows the trajectory of the robot tracking the reference trajectory and simultaneously avoiding obstacles present along. Figure 3.17(b) represents the tracking errors and input velocities, it can be observed from the plots that the linear velocity is constant when the avoidance controller is active, confirming the switch between controllers based on the hierarchical action selection algorithm.

3.7.2.2 Formation control with periodic leader switch

For this experiment, we task four e-puck2 robots to track the circular trajectory with three of the robots following the leader, and then automatically switch leader

Figure 3.16: (a) Robot0 just before the region of influence of robot1, (b) Robot0 inside the region of influence of robot1 and avoiding it, (c) Robot0 back to trajectory tracking after avoiding robot1 completely.

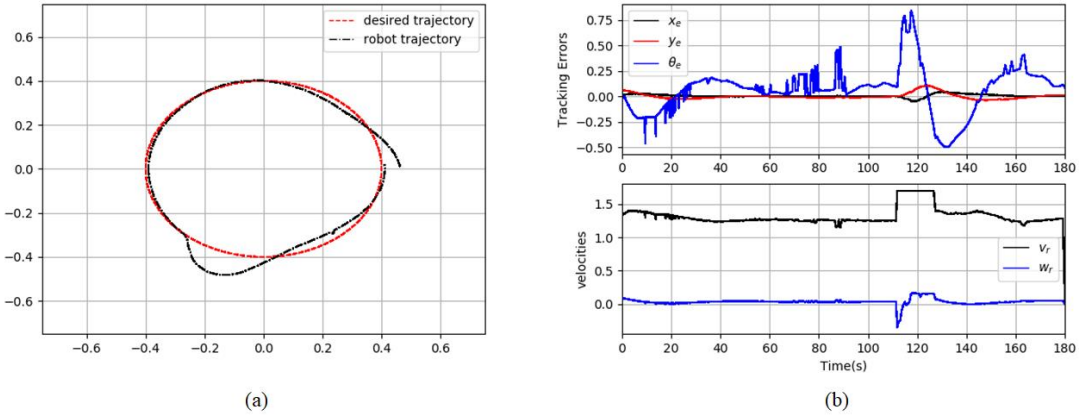

Figure 3.17: (a) Reference trajectory and robot0 motion trajectory, (b) Robot0 tracking errors and input velocities.

each time a circle is complete. We implement it such that robots move in a square formation of size $0.15m$, with follower robots following the leader at bearings of 180°, 225°, and 270° (see Figure 3.18). After completing a circle, robot at a bearing of 270° from the leader becomes the new leader, follower robots at bearings 180° and 225° follow the new leader at bearings 225°, and 270° respectively (meaning the follow new leader at a bearing 45° more than their previous). The former leader then follows the new leader at 180° bearing. Some tapped images from the video of this experiment shown in Figure 3-19 show scenes with each of the robots leading the team, the yellow arrow on each of the scenes points the direction of motion at that moment. The group

tracks the trajectory four times during this experiment with each of the robots leading the team once for a complete circle.

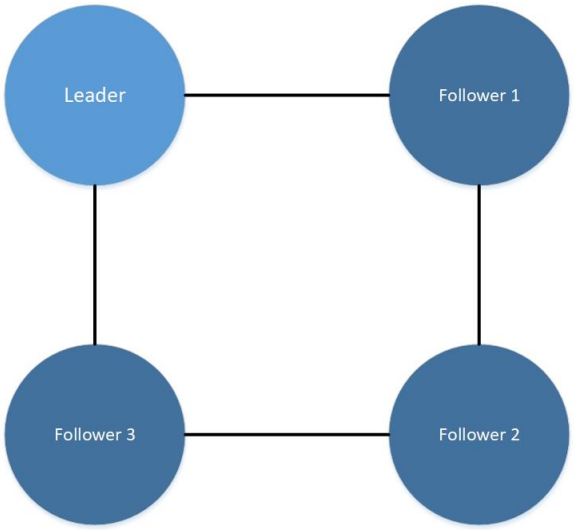

Figure 3.18: Desired formation shape.

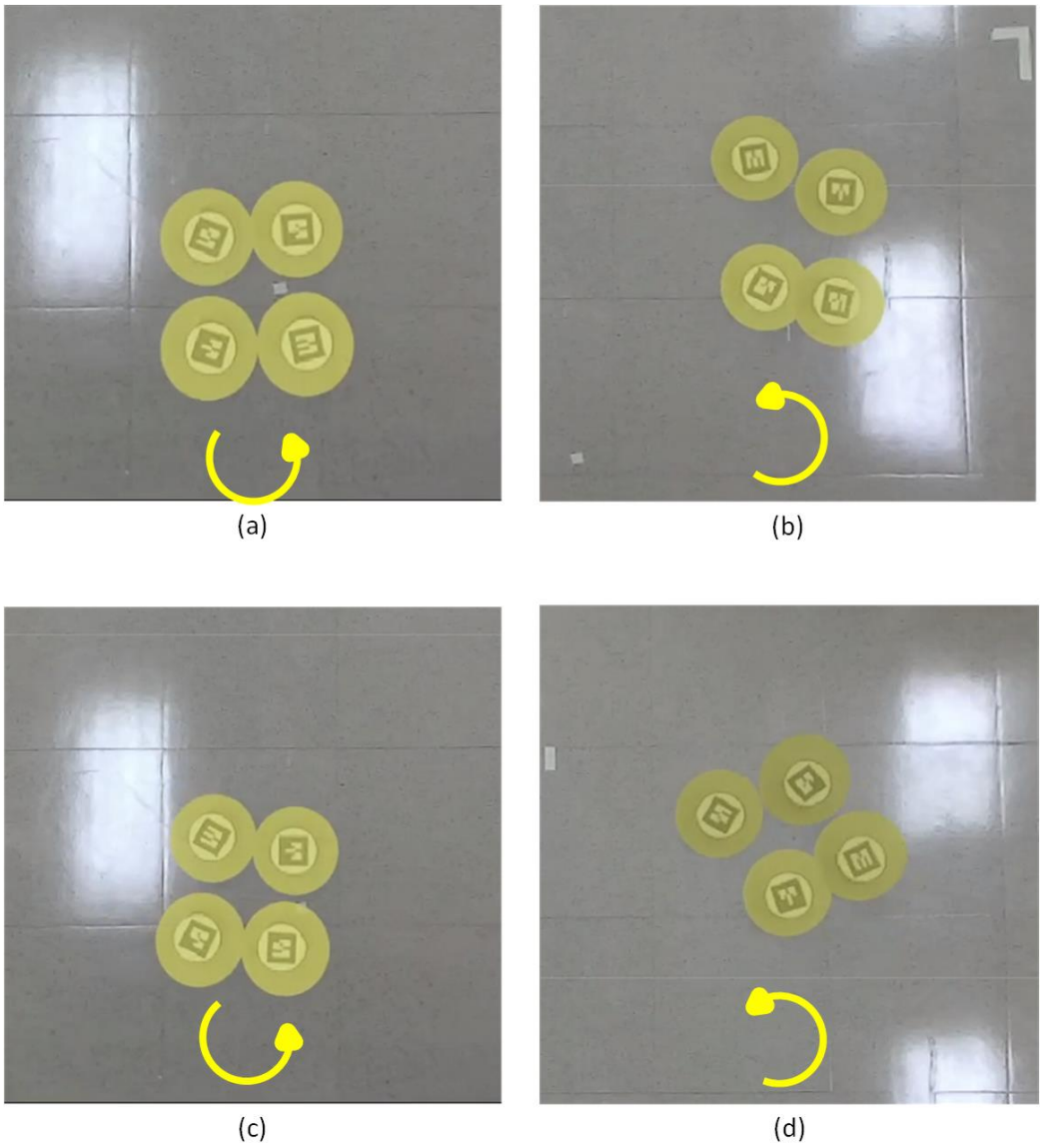

Figure 3.19: (a) Robot0 leading the group, (b) Robot1 leading the group, (c) Robot2 leading the group, (d) Robot3 leading the group.

Chapter 4

Formation control subject to leader failure

4.1 Multi-robot algorithms

Lately, we have seen a surge in the use of multi-robot systems (MRS) in a wide variety of applications. A lot of MRS are being deployed in a variety of domains, examples include: warehouse [1], hospital logistics [49], transport [50], just to name a few. In order to develop and deploy robust MRS in real-world applications, a number of challenging problems needs to be solved. These problems include, but are not limited to, task allocation, group formation, cooperative object detection and tracking, communication relaying and self organization to name just a few [51].

Great number of algorithms are present in the literature to improve task allocation problems in MRS. Multirobot task allocation (MRTA) refers to the assigning of a series of tasks to multiple robots with certain constraints to achieve an objective, such as minimizing the total travel distance of all robots or the average cost of each task and so on [51]. As reported in the survey by [52], consensus-based bundle algorithm, consensus-based auction algorithm, fair subdivision algorithm, learning automata-based probabilistic algorithm, and S+T algorithm allocate tasks to robot with the objective of minimizing time to reach the goal. And to minimize distance travel by robots, the commonly used algorithms include Prim allocation algorithm, SET-MASR algorithm, SIT-MASR algorithm, task-switching algorithm, and incremental

task allocation algorithm.

Our focus in this work is not to minimize time to reach goal or finding minimum distance to the goal. We intend to fix the problem of leader failure in a formation by assigning the task of leading the group to one of the follower robots so the group can continue with the task (in this case, tracking a trajectory in a formation). We want our system to be so robust to withstand the failure of not only the leader, but also the failure of any robot in the group, or even in uncommon cases, failure of multiple robots.

4.2 Assignment algorithm

Leader failure is the major drawback of leader-follower formation control strategy. In [10], this problem is addressed by dynamically changing the leader, choosing a new leader from the follower agents through a dedicated workstation, formation parameters are also (re)defined through this workstation. This seems like a good idea; however, it does not really solve the problem because the newly selected leader could fail as well. The approach also adds the burden of having a dedicated supervisor at the workstation to manually select a new leader. In this work, we develop an algorithm that assigns a role to each robot in the team. We rank roles serially from 0 to $(n-1)$ with "n" the number of robots in the team. Rank "0" is the highest, defined as the leadership-role. All other roles are follower-role with $(n-1)$ having the least rank. Figure 4.1 shows the role positions in a defined formation shape and how we ranked roles serially from 0 to $(n-1)$.

Roles are occupied by agents joining the team sequentially, each agent joining the team takes the role with the highest rank from the vacant roles, meaning, the first agent to join will take the leadership role, and the last to join takes the role with the least rank. In case of a leader failure, this algorithm let agents change their roles to that with a rank one step higher, meaning, agent with a previous role of "*one*" will take the leadership role "*zero*", and all other agents adjust their roles one step higher

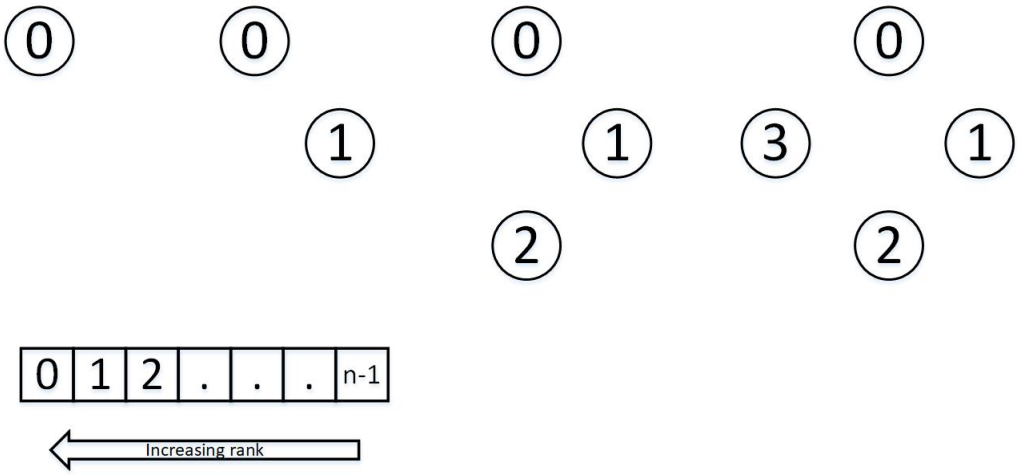

Figure 4.1: Role positions in a square shaped formation with robots occupying positions sequentially.

to follow the new leader, leaving the least role vacant for another robot joining the team to occupy. With this, the problem of leader failure is eliminated completely, as the algorithm will let one of the followers to take leadership automatically, as well, let other followers adjust their roles to follow the new leader. Additionally, any robot in the team could fail without affecting the formation, the active robots only need to automatically adjust their roles if a role ranked higher than their current role become vacant. This also mean that if we can recover the failed robot, we can easily launch it back to the team to occupy the vacant role.

The pseudo code for this algorithm is as follows:

Algorithm 3 : Assignment algorithm

```
1  for Robot in Robots do
2      while Active do
3          get_lead_vacant_role()
4          if robot_role == None then
5              robot_role = lead_vacant_role
6          else:
7              if rank(lead_vacant_role) == rank(robot_role) +1 then
8                  robot_role = lead_vacant_role
9              else:
10                 robot_role = robot_role
```

4.3 Implementation

We run a series of simulations and experiments to describe how our algorithm works. We use the same controllers and controller parameters previously used in Chapter 3 to drive robots for trajectory tracking and obstacle avoidance. The same workstation in Chapter 3 is used for the implementation.

4.3.1 Simulations

Here, we consider a number of possible cases of robot failure that might happen to robots in a formation. Using a maximum of four robots throughout the simulations, we consistently launch the robots sequentially in the order robot0, robot1, robot2, and robot3 so that the roles 0, 1, 2, and 3 are occupied by robots 0, 1, 2, and 3 respectively. It should be noted that any robot can take available role with the highest rank, and we only chose to launch them this way for the sake graphical clarity on how the robots are adjusting their positions in the formation when a robot fails. The desired formation shape as shown in Figure 4-2 is defined by a rhombus of length 1m, with follower robots at bearings 135°, 195°, and 255°. The trajectory used in this case is defined by equation 3.20, centered at the origin $(0, 0)$ with radius $r = 4m$ and period $T = 180s$.

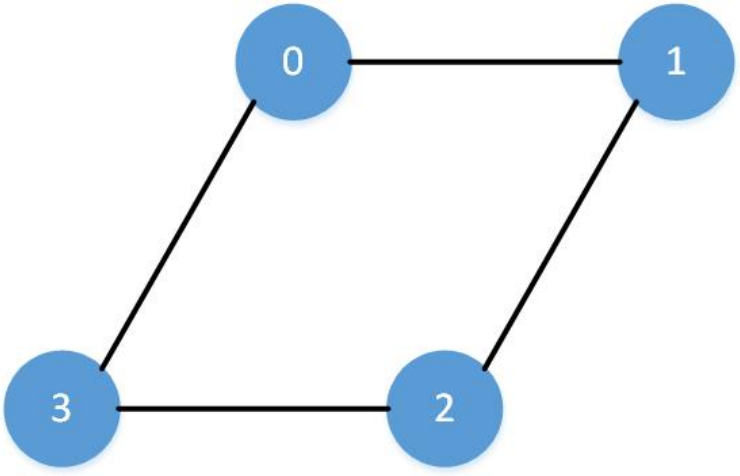

Figure 4.2: Desired formation shape.

The cases considered in this simulation are as follows:

- **Failure of robot occupying role 0:** As stated earlier, the main objective of this algorithm is to enable follower robots to overcome the problem of leader failure. In Figure 4.3(a), robots 0, 1, 2, and 3 successfully tracks the trajectory for a complete circle in a formation by respectively occupying the roles 0, 1, 2, and 3. We then fail robot0 (the leader) by killing its node when it is at about $(-3.8, 1.6)$. It can be seen in Figure 4.3(b), robot1 which was occupying role 1 now takes the vacant role 0 and proceeds with tracking the trajectory. Robots 2 and 3 previously on the roles 2 and 3 now adjusts their roles respectively to 1 and 2 to follow the new leader leaving role 3 vacant.

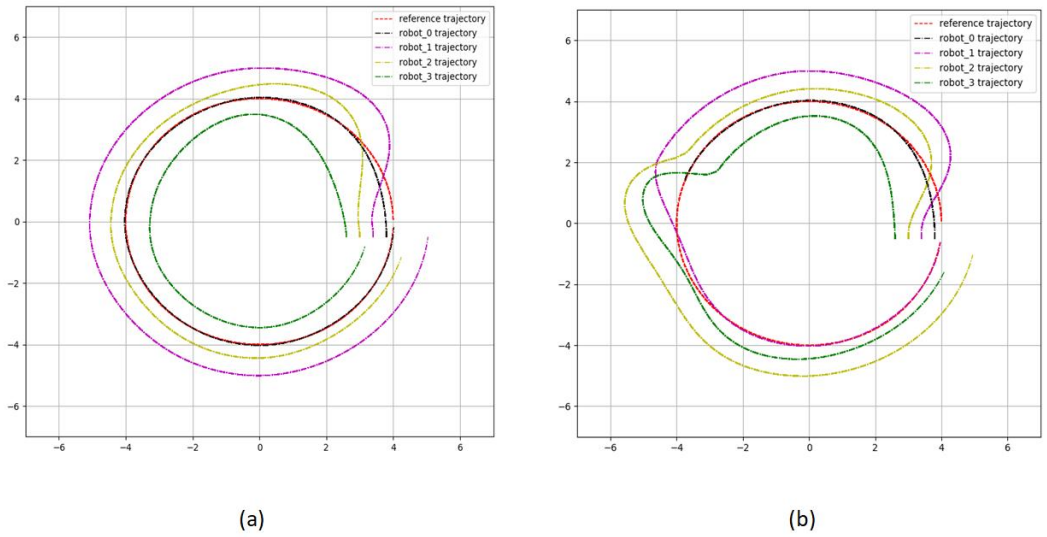

Figure 4.3: (a) Robots' trajectories with no robot failure, (b) Trajectories under leader failure.

- **Failure of robot occupying role 1:** In this case, we consider failing robot1 which is occupying role 1. With reference to Figure 4.4(a), Figure 4.4(b) shows robot0 maintaining its role as the leader (of course because it is on a role higher than that of the failed robot), while robots 2 and 3 update their roles to one with rank one step higher to maintain the formation, leaving role 3 vacant in

case the leader is recovered.

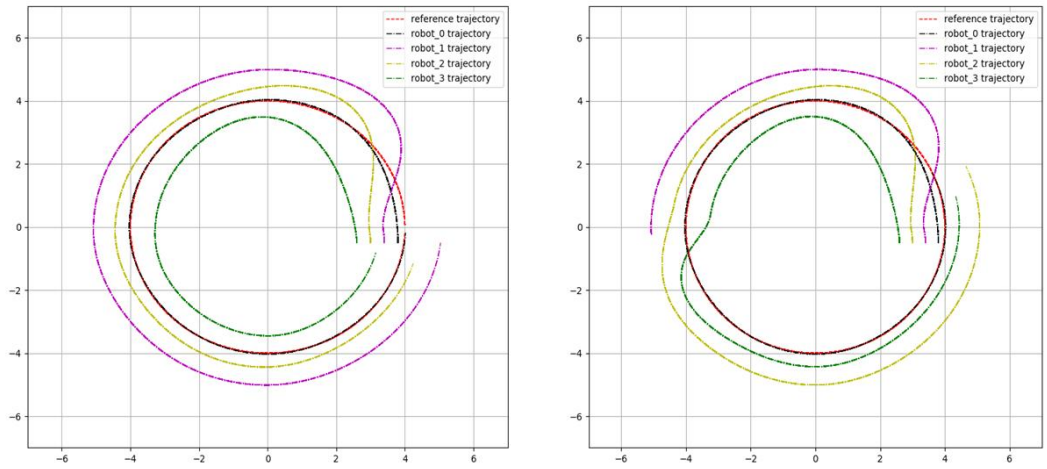

Figure 4.4: (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of robot occupying role 1.

- **Failure of robot occupying role 2:** Similar to the previous cases, we subjected the robot in role 2 to failure to see how our algorithm will handle the case. As shown in figure 4.5(b), the group leader and robot on role 1 maintained their roles, while robot 3 moved to the vacant role one step higher, leaving the least role vacant.

- **Failure of robot occupying role 3:** We want to see if other robots will react if the robot on a least ranked role fails, we therefore killed he robot on role 3. As we expect the algorithm to work, the active robots did not adjust their roles to occupy the vacancy as they are already on a role with a higher rank. Figures 4.6(a) and 4.6(b) respectively shows the robots' trajectories without robot failure and the trajectories when robot on role 3 fails.

- **Multiple robots failure:** We have previously considered cases where single robot occupying a certain role fails and verified that the algorithm handles such

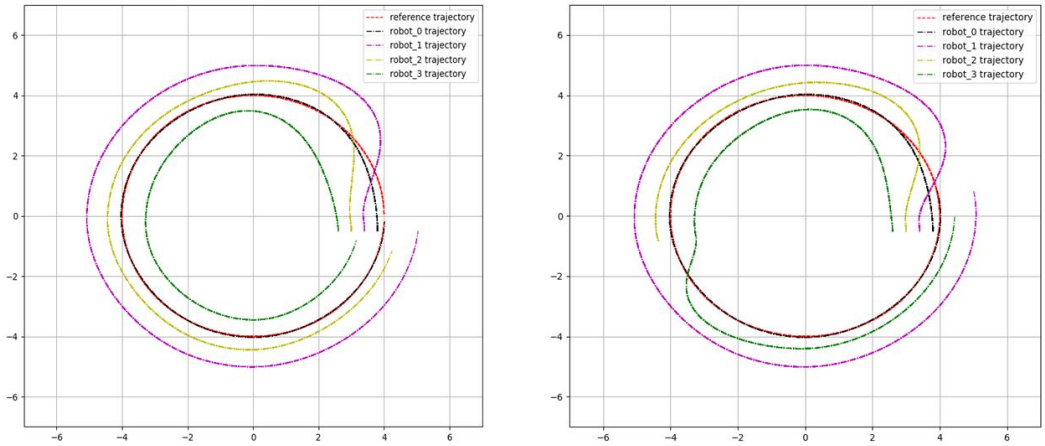

Figure 4.5: (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of robot occupying role 2.

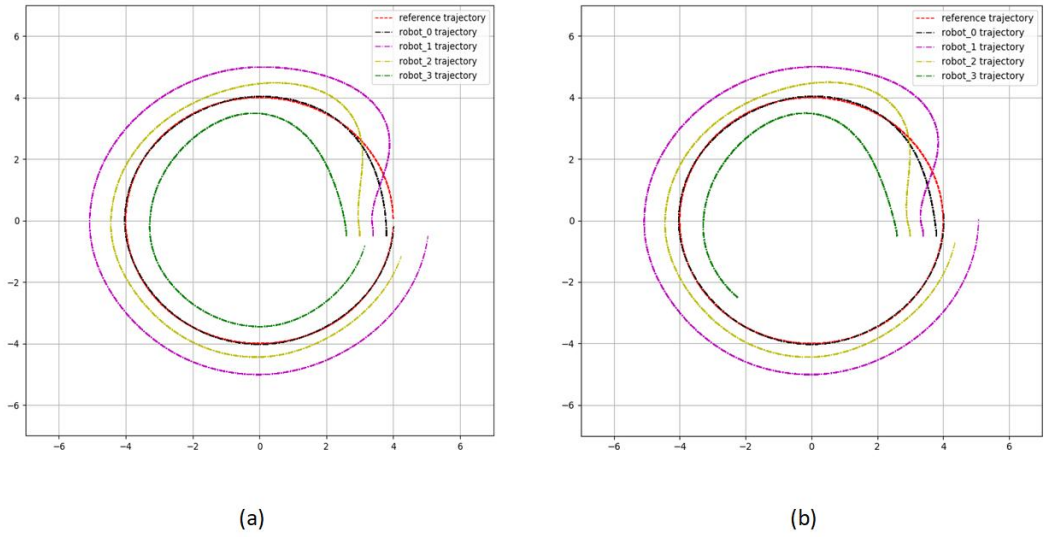

Figure 4.6: (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of robot occupying role 3.

cases well. We now consider testing the algorithm in a rare possibility where multiple robots fail. The aim is to show that unless if every single robot in the team fails, there will be at least one robot that will keep up with the trajectory tracking (or any given task). Consider four robots tracking the trajectory in a

formation (see Figure 4.7(a)) with three of them having the possibility of failure. By respectively killing robots occupying the roles 2, 1, and 0 at different time instances, it can be seen in Figure 4.7(b) how the robot on role 3 adjusted its role to 2, 1, and finally 0. This proves the robustness of our algorithm to robots' failure, and how it offers the flexibility to easily add new robots to the team or remove the failed ones without affecting other robots.

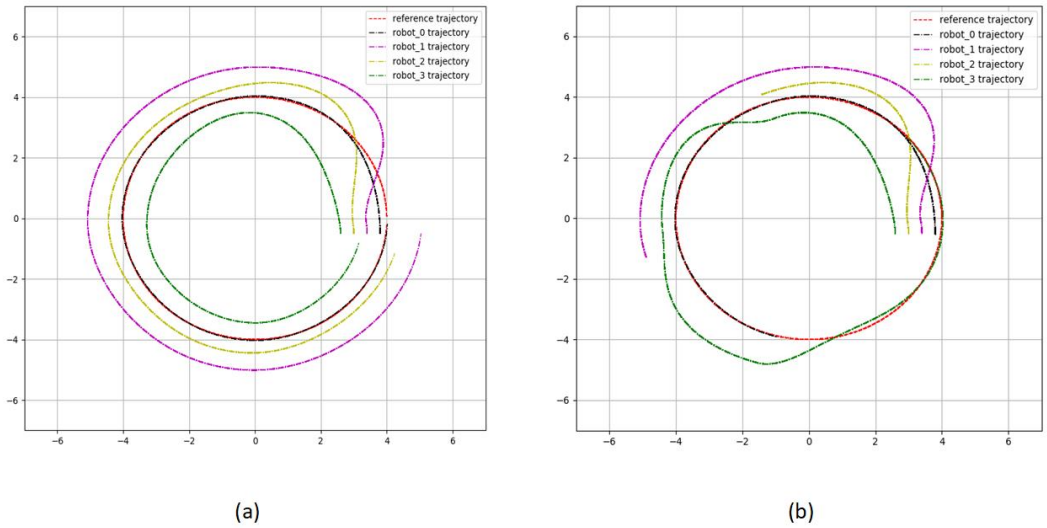

Figure 4.7: (a) Robots' trajectories with no robot failure, (b) Trajectories under failure of multiple robots.

We have seen how the algorithm is able to handle different cases of failure among robots, but what if we are able to recover a failed robot? that is, what will happen if for example, a former leader robot gets back to the team? As mentioned earlier, we designed our algorithm such that robots joining the team will only be assigned role from the list of available roles. We implement it such that all failed robots have their roles reset to "None", so they will be seen as robots with no previous roles. This will enable the algorithm to assign them new roles from the list of vacant roles regardless of what their previous role was. We show this in Figure 4.8(b) where we assumed to have recover a failed leader (robot0 in this case). It can be seen that after the followers have taken over that task of tracking, the failed leader joined back to

occupy the available role 3. There was no role conflict between current leader and the former leader which is now in the group as a regular follower. This shows the effectiveness of our algorithm in assigning appropriate roles to robots so they can overcome the problem of failure from other robots in the group.

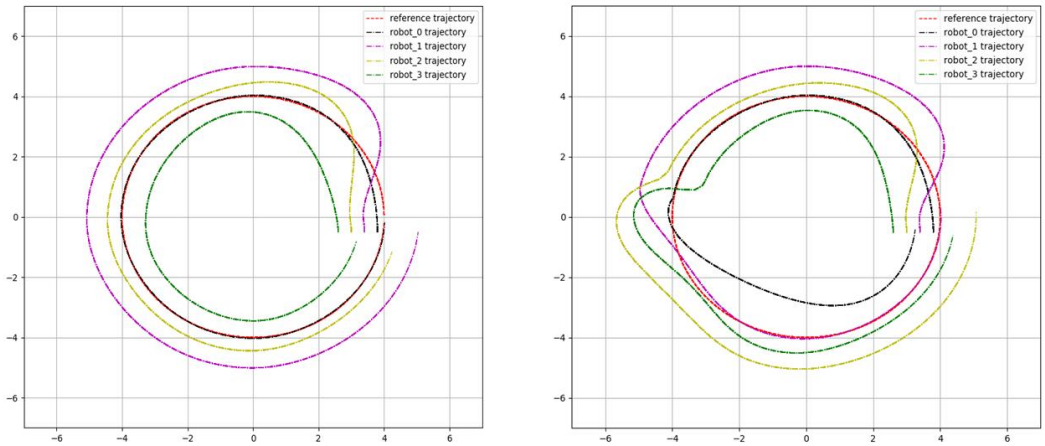

Figure 4.8: (a) Robots' trajectories with no robot failure, (b) Trajectories with failed robot recovery.

4.3.2 Experiment

To demonstrate the effectiveness of our algorithm on real robots, we use the experimental setup in Chapter 3 (see Figure 3.15) for the same reasons stated therein. The packages, driver, trajectory, and formation shape choices remain the same as well. We run a set of experiments considering each of the possible cases of robot failure on a particular role. We present each of the cases in details as follows:

- **Failure of robot occupying role 0:** We begin with demonstrating the case of leader failure in a formation. Consider four e-puck2 robots each with a unique AR tag tracking a circular trajectory in a square formation. Figure 4.9(a) shows the group with the encircled robot leading the team. We made the leader fail in

Figure 4.9(b) by killing its control node, and we can see that the robot on role 1 takes the vacant role 0 with robots on roles 2 and 3 now following the new leader on roles 1 and 2 respectively. Assuming to have recovered the failed robot, we launched back to the group. Figure 4.9(c) shows the robot back in the group right behind the leader occupying role 3. Similar to the result in simulation, the algorithm enables follower robots to overcome leader failure, and the failed robot launched back to the team takes available role not its former role as the leader.

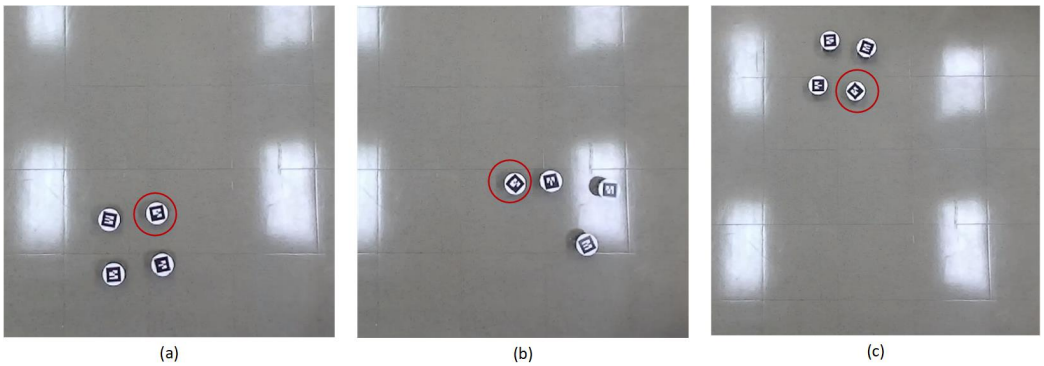

Figure 4.9: Four robots in a formation (a) before leader failure, (b) when the leader failed, (c) when the failed robot is recovered.

- **Failure of robot occupying role 1:** We consider in this case, failure of the successor (robot on role 1) which is at a bearing 270° from the leader. When this robot fails, we expect the leader to maintain its role, and robots on roles 2 and 3 to respectively follow the leader on roles 1 and 2. The encircled robot we see in Figure 4.10(a) is the robot on role 1 shortly before it fails, the leader maintained its role, and other followers followed the leader on a new role as shown in Figure 4.10(b). Figure 4-10(c) depicts the failed robot joining back the team occupying the least ranked vacant role.

- **Failure of robot occupying role 2:** Similarly, the robot on role 2 failed, and only the robot on role 3 adjust its role to fill the vacancy in role 2, while the leader and robot on role 1 maintained their role. Figures 4.11(a) and 4.11(b)

Figure 4.10: Four robots in a formation (a) before leader failure, (b) when the leader failed, (c) when the failed robot is recovered.

depicts the robot before and after failure respectively, while Figure 4.11(c) shows the robot rejoining the team taking the vacant role 3.

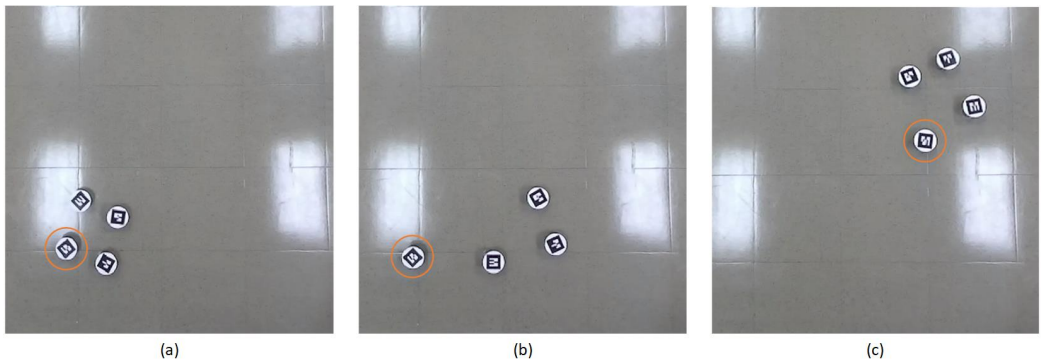

Figure 4.11: Four robots in a formation (a) before role 2 robot failure, (b) when role 2 robot failed, (c) when the failed robot is recovered.

- **Failure of robot occupying role 3:** The last case we looked at is when the robot on the least role fails. Of course, we don't expect other robots to care as they are already on a higher ranked roles, and this is exactly what happen as seen in Figures 4.12(a), 4.12(b), and 4.12(c).

We have seen experimentally, the effectiveness of our algorithm in not only solving the problem of leader failure, but also the failure of any robot in the team. In addition, it also enables followers adjust their positions in the formation when necessary to make room for robots joining the team.

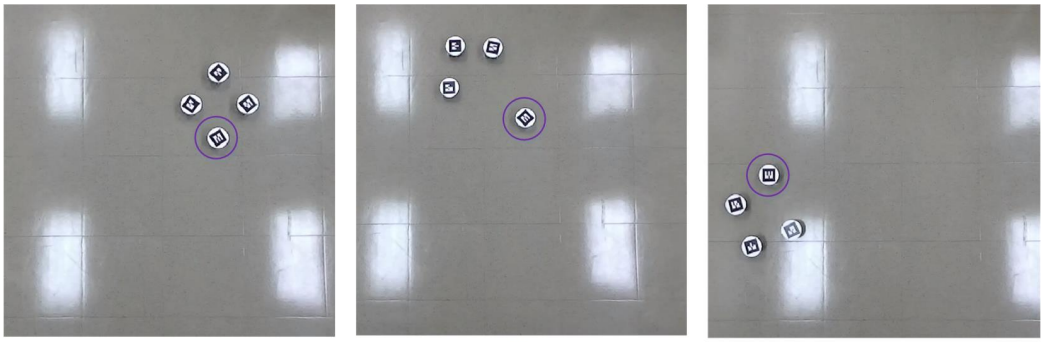

Figure 4.12: Four robots in a formation (a) before role 3 robot failure, (b) when role 3 robot failed, (c) when the failed robot is recovered.

4.4 Technical difficulties

As to any research implementation that has to do with hardware devices, working with mobile robots have its fundamental problems. Developing a good solution to these problems is essential as robots become totally autonomous and continuously expanding their range of application. As outlined in a review by [53], some of the main challenges are navigation, path planning, localization, and obstacle avoidance. Earlier in this thesis, we present algorithms to tackle problems of leader failure in a leader-follower formation control strategy and obstacle avoidance in trajectory tracking problems, it is therefore important to discuss problems encountered when validating our approach using real robots. What follows describes the challenges faced and how we find way out of these problems.

- **Localization:** The challenging part of localization is estimating the robot position and orientation of which this information can be acquired from sensors and other systems [53]. We are dealing specifically with formation control problem where follower robots get formation information from their leader. Since this information is defined relative to the leader's pose, some level of accuracy is needed in estimating the robots' posture so we could achieve the desired formation shape. The e-puck2 robot we use in our experiments provides pose

estimates from the readings by the wheel odometer, and we can of course fuse data form the Inertial Measurement Unit (IMU) using the Extended Kalman Filter (EKF) for a more accurate pose estimation. Unfortunately, this robot does not have laser sensor that will enable us to map our relatively large environment and eventually use the popular Adaptive Monte Carlo Localization (AMCL) ROS package to track the pose of our robots.

With unique AR tag mounted on each robot, we involve the use of external stereo camera to feed each of the robots with its pose estimate using the *ar_track_alvar* ROS package. This provides solution to our localization problem, but also imposed constraints on where the robots can go due to the limited view of the camera. We conducted a test using single robot to determine the accuracy of the pose estimation and observed that at HD1080 resolution, we get good estimates of the position with some jumps in the orientation data as the robot moves away from the center of the camera's field of view (see theta0 in Figure 4.13).

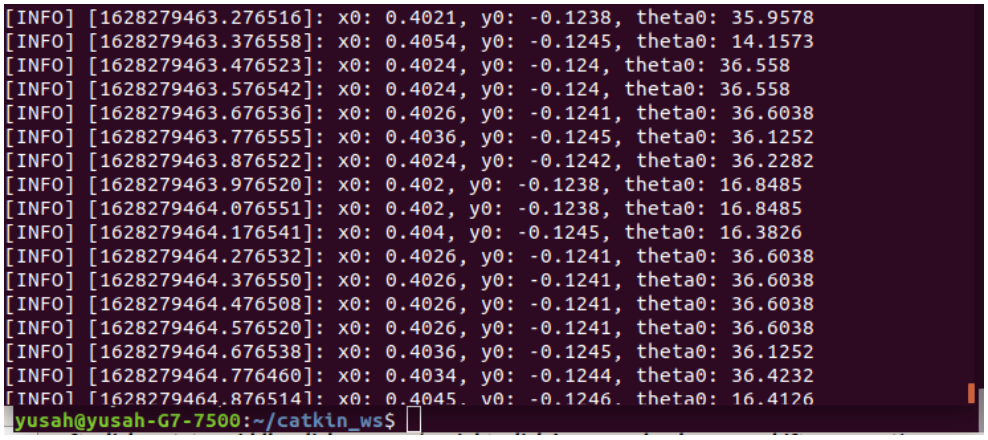

Figure 4.13: Test pose estimate.

Considering how we modeled the dynamics of our robot in equation 3.2 and the desired formation in equations 3.6 to 3.8, these jumps in orientation data creates big problem as it makes it difficult to control the robots to accurately track the trajectory. Figure 4.14(a) depicts an e-puck2 robot tracking a circular

trajectory getting its pose estimate from the camera. In Figure 4.14(b), we can see oscillations in orientation error "θ_e" resulting from the oscillations in the orientation data from the camera.

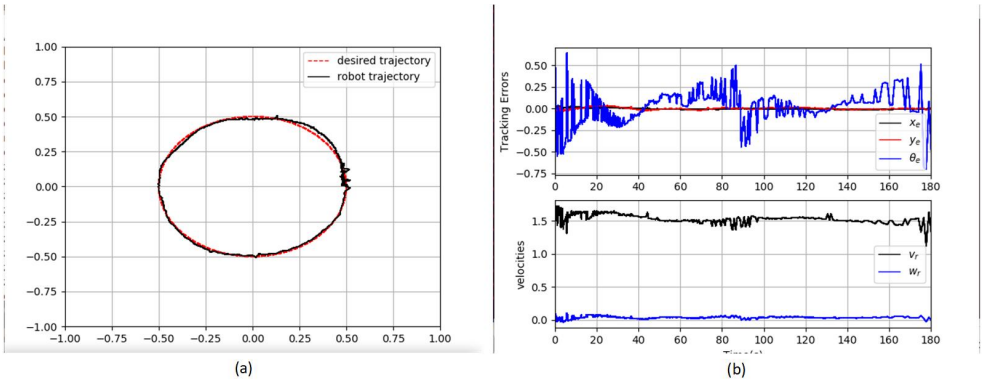

Figure 4.14: (a) Test robot tracking a circular trajectory, (b)Plots for tracking errors and input velocities.

We plot formation data generated by a leader robot to observe the trajectories. Figure 4.15 shows how the three non-smooth trajectories generated by the robot, and how the robot tracks the trajectory with visible tracking errors.

To mitigate this problem, we increased the camera resolution to HD2K, reduced the distance from camera to the robots to just about $2m$, limit our trajectory option to circular with maximum radius of $0.4m$, and defined a square formation of size $0.15m$. This helped us get better pose estimates to implement the experimental results presented in the previous section. Even though we sometimes don't get a perfect square formation, Figures 4.9 to 4.12 show fairly good formation shapes.

- **Obstacle avoidance:** The obstacle avoidance controller used in this thesis controls only angular velocity of the robot with a constant linear velocity to avoid obstacles. The jumps in orientation data which in turn affects the orientation error defined by equation 3.16 makes it challenging to use this avoidance strategy. Prior to arriving at the result for the obstacle avoidance presented in this

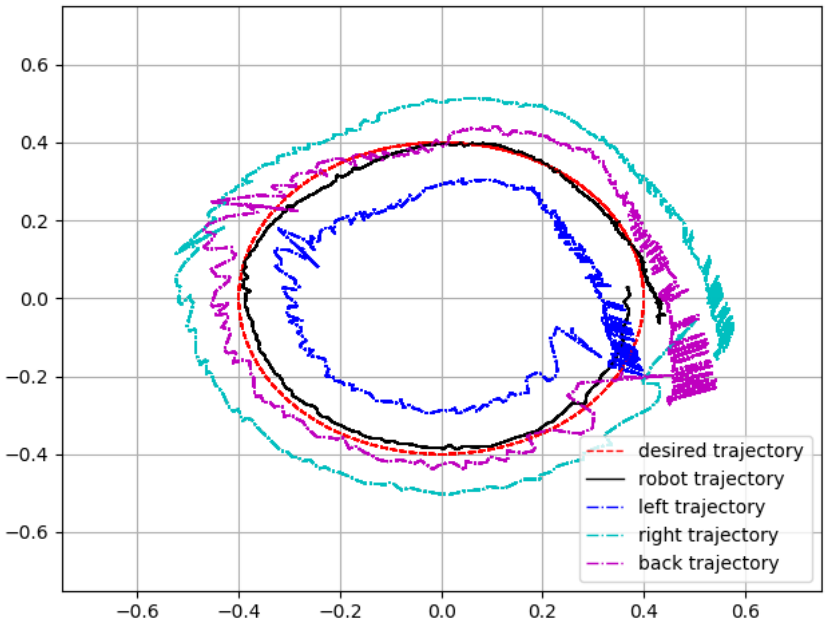

Figure 4.15: non-smooth generated trajectories.

thesis, we obtained results where the robot avoids obstacle off the limit-cycle (see Figure 4.16). This is mainly due to the pose estimation errors from the camera which improve with improved localization. Another problem was with the robot crashing into the obstacle when trying to track the origin instead of the circumference of the limit-cycle. The reason was later found out to be the choice of constant linear velocity and controller parameter. We carefully tuned the parameter and velocity to arrive at the result in Figure 3-17.

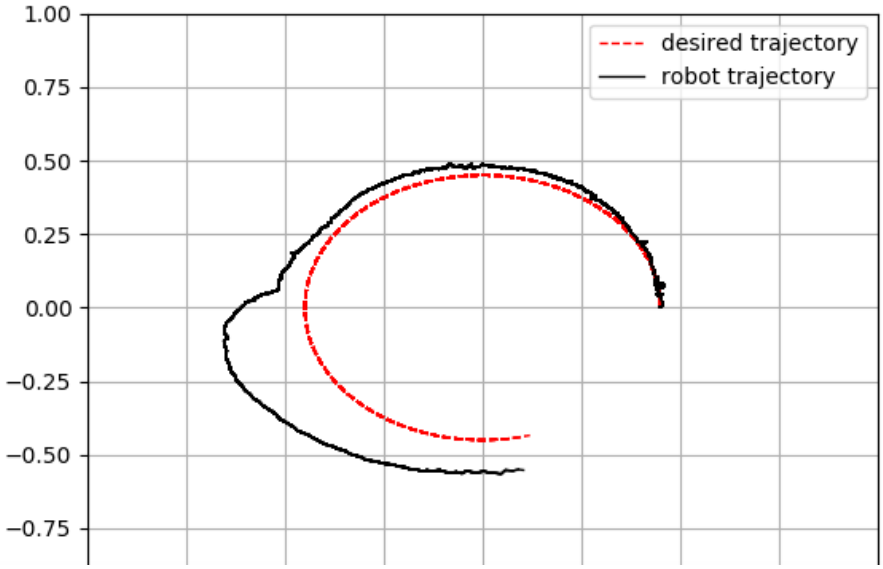

Figure 4.16: Obstacle avoidance under large pose estimation error.

Chapter 5

Summary and conclusions

This thesis presents two novel algorithms: the obstacle avoidance algorithm that enables the use of limit-cycle obstacle avoidance strategy in trajectory tracking applications, and Assignment algorithm that guarantee follower robots overcome the practical problem of leader failure in leader-follower formation control strategy. We introduced briefly in Chapter 2, some technical preliminaries. This includes Graph theory as it applies to leader-follower formation control strategy whose formation configuration is described as a diagraph, and Lyapunov stability theory which is used to design both trajectory tracking and obstacle avoidance controllers from the literature. In addition, we briefly describe the hardware devices used in our experiments, which include the e-puck2 robot (a small differential drive robot) and the ZED Stereo camera for localization. We introduce ROS – a flatform for writing robot software, fiducial markers – to be detected by the camera for robot localization, and ROS drivers used for our hardware devices to round up the second chapter.

We model our robot's dynamics in Chapter 3, then introduced backstepping based controller from the literature for the trajectory tracking problems which we used throughout our implementations to drive our robots. We show from the literature that trajectory tracking can be extended to solve formation control problem using the leader-follower strategy. In leader-follower strategy, the leader having access to the reference trajectory communicates formation information to its followers, this

formation information is a trajectory that must be feasible for the follower robots. The formation information defined in Chapter 3 of this thesis ensure that the trajectory is feasible, and follower robots can successfully follow their leader to move in a desired formation. We also introduced a Lyapunov-based, backstepping controller from the literature, this controller is used to drive both leader and follower robots to their desired posture so they can maintain formation.

Avoiding obstacle is essential for a mobile robot to reach its goal especially in a case where multiple robots are involved or when robots find themselves in a cluttered environment. We present in Chapter 3, a novel algorithm that enable the use of limit-cycle obstacle avoidance strategy in trajectory tracking applications. The stability of the controller used to drive robots to track accurately a limit-cycle is proved using the Lyapunov stability theory. Having two controllers to work with, a hierarchical action selection algorithm from the literature is used to switch between the two controllers. We use simulations and experimental results to validate the effectiveness of our avoidance algorithm. It is presented in the literature that we can switch leader to reduce the leadership burden on single robot, which can locally solve the problem of leader failure. We implement this by adding a layer of automation on the work found in the literature. Our approach uses reduced number of workstations needed to control the robots, and the necessity human intervention at the workstation to switch leader or define new formation parameters.

Assignment algorithm presented in Chapter 4 is aimed at providing lasting solution to the practical problem of leader failure in leader-follower formation control strategy. A mobile robot deployed for rescue missions or surveying can fail, cause of which can be energy or communication failure, this mean a whole group of robots will fail if the failed robot happens to be their leader. Our algorithm solves this problem by letting robots joining the team to choose their role – whether to lead or to follow – from the list of available roles, and for robots in the group to adjust their roles when necessary if they lose a team member. This implies when a leader is lost, robot in the best

position to lead the group automatically becomes the new leader, and other follower robots adjust their roles to follow this new leader. We have shown using simulations and experiments using the ROS framework that unless if every robot in the team fails, there will be at least one robot to continue with the assigned task. We proved in practical setting that any robot in the group can fail without affecting the group formation. We have also seen how the algorithm provides us with the flexibility to easily add new robots to the group or remove failed robots from the group. We lastly describe the technical difficulties we have faced during our experiments and how we overcome such.

In conclusion, we proposed two novel algorithms and verified their effectiveness using a set of simulations and experiments. Future work includes the use of our algorithm in a more difficult task for robots in a formation and cooperative obstacle avoidance. Lastly, we believe our algorithm can be applied in other distributed multi-robot applications to mitigate problems that may arise from member robot failure.

Bibliography

[1] M. O'Brien. (2019). "Independent," [Online]. Available: https://www.independent.co.uk/news/business/robots-amazon-delivery-artificial-intelligence-technology-a9264036.html (visited on 11/02/2021).

[2] L. Barnes, M. Fields, and K. Valavanis, "Unmanned ground vehicle swarm formation control using potential fields," in *2007 Mediterranean Conf. Control Autom.*, 2007, pp. 1–8.

[3] R. Hoag. (2020). "Naval postgraduate school," [Online]. Available: https://nps.edu/-/nps-researchers-developing-the-defensive-playbook-against-large-scale-drone-swarms (visited on 11/05/2021).

[4] I. Lončar, A. Babić, B. Arbanas, G. Vasiljević, T. Petrović, S. Bogdan, and N. Mišković, "A heterogeneous robotic swarm for long-term monitoring of marine environments," *Applied Sciences*, vol. 9, no. 7, 2019.

[5] O. Kwang-Kyo, P. Myoung-Chul, and A. Hyo-Sung, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[6] S. Knorn, Z. Chen, and R. H. Middleton, "Overview: Collective control of multi-agent systems," *IEEE Trans. Control Network Syst.*, vol. 3, no. 4, pp. 334–347, 2016.

[7] Y. Liu and R. Bucknall, "A survey of formation control and motion planning of multiple unmanned vehicles," *Robotica*, vol. 36, no. 7, 1019–1047, 2018.

[8] W. B. Randal, L. Jonathan, and Y. H. Fred, "A coordination architecture for spacecraft formation control," *IEEE Trans. Control Syst. Technol.*, vol. 9, pp. 777–790, Nov. 2001.

[9] P. S. Daniel, Y. H. Fred, and R. P. Scott, "A survey of spacecraft formation flying guidance and control (part ii): Control," in *Proc. 2004 Amer. Control Conf.*, Boston, Massechussetts, 2004.

[10] L. B. Khadir, T. Radosław, K. Wojciech, and P. Jarosław, "Leader-follower formation control for a group of ros-enabled mobile robots," in *2019 6th Int. Conf. Control. Dec. Information Technol. (CoDIT'19)*, Paris, France, Apr. 2019.

[11] J Chen, D Sun, J Yang, and C. H., "Leader-follower formation control of multiple non-holonomic mobile robots incorporating a receding-horizon scheme," *Int. J. Rob. Res.*, vol. 29, pp. 727–747, 2010.

[12] H. Koki, Z. Tadanao, O. Noriaki, and L. Kang-Zhi, "Optimal formation control of two-wheeled vehicles using model predictive control," in *2015 10th Asian Control Conf. (ASCC)*, Chiba, Japan, 2015.

[13] H. Shunta, Z. Tadanao, H. Koki, and L. Kang-Zhi, "Optimal automatic formation control for two-wheeled vehiles using model preditive control with temporal logic constraints," *2018 12th France-Japan and 10th Eur-Asia Congress on Mechatronics*, pp. 161–164, 2018.

[14] M. Silvia, M. S. Dušan, R. G. Christopher, A. I. Koji, and W. S. Mark, "Formation control and collision avoidance for multi-agent non-holonomic systems: Theory and experiments," *Int. J. Rob. Res.*, vol. 27, pp. 107–126, Jan. 2008.

[15] R. Rout and B. Subudhi, "A backstepping approach for the formation control of multiple autonomous underwater vehicles using a leader–follower strategy," *J. Marine Engg. & Technol.*, vol. 15, no. 1, pp. 38–46, 2016.

[16] A. Keymasi Khalaji and R. Zahedifar, "Lyapunov-based formation control of underwater robots," *Robotica*, vol. 38, no. 6, 1105–1122, 2020.

[17] A. B. Siavash, R. Esteban, and W. Herbert, "Formation control of non-holonomic agents with collision avoidance," in *2015 Amer. Control Conf.*, Chicago, IL, USA, Jul. 2015.

[18] H. Wei, Q. Lv, N. Duo, G. Wang, and B. Liang, "Consensus algorithms based multi-robot formation control under noise and time delay conditions," *Appl. Sci.*, vol. 9, no. 5, 2019.

[19] F. Mehdifar, C. P. Bechlioulis, F. Hashemzadeh, and M. Baradarannia, "Prescribed performance distance-based formation control of multi-agent systems," *Automatica*, vol. 119, p. 109 086, 2020.

[20] K. Dong-Han and K. Jong-Hwan, "A real-time limit-cycle navigation method for fast mobile robots and its application to robot soccer," *Robot. Autom. Syst.*, vol. 42, pp. 17–30, 2003.

[21] L. Adouane, "Orbital obstacle avoidance algorithm for reliable and on-line mobile robot navigation," in *9th Conf. on Auton. Robot. Syst. Competitions*, Castelo-Branco, Portugal, May 2009.

[22] B. Ahmed, A. Lounis, and M. Philippe, "Dynamic obstacle avoidance strategies using limit cycle for the navigation of multi-robot system," in *2012 IEEE/RSJ IROS'12, 4th Workshop on Planning, Perception and Navigation for Intelligent Vehicles*, Vilamoura, Algarve, Portugal, Oct. 2012.

[23] R. J. Wilson, *Introduction to Graph Theory*. New York: John Wiley & Sons, Inc., 1986.

[24] J. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Proc. 1998 Int. Conf. Robot. Autom.*, vol. 4, 1998, 2864–2869 vol.4.

[25] H. J. Marquez, *Nonlinear Control Systems: Analysis and Design*. New Jersey: John Wiley & Sons, Inc., 2003.

[26] GCTronic. (2018). "E-puck," [Online]. Available: http://www.e-puck.org/ (visited on 11/12/2021).

[27] M. Sugisaka, H. Tanaka, V. Trifa, C. Cianci, and D. Guinard, "Dynamic control of a robotic swarm using a service-oriented architecture," Jan. 2008.

[28] T. K. Tasooji and H. J. Marquez, "Cooperative localization in mobile robots using event-triggered mechanism: Theory and experiments," *IEEE Trans. Autom. Science. Engg.*, pp. 1–13, 2021.

[29] T. T. Harmanda, M. K. D. Hardhienata, and K. Priandana, "Development of multi-robot systems using particle swarm optimization algorithm for task allocation," in *2021 IEEE Region 10 Symposium (TENSYMP)*, 2021, pp. 1–8.

[30] R. Visvanathan, S. M. Mamduh, K. Kamarudin, A. S. A. Yeon, A. Zakaria, A. Y. M. Shakaff, L. M. Kamarudin, and F. S. A. Saad, "Mobile robot localization system using multiple ceiling mounted cameras," in *2015 IEEE SENSORS*, 2015, pp. 1–4.

[31] StereoLab. (2020). "Stereo lab," [Online]. Available: https://support.stereolabs.com/hc/en-us/articles/207616785-Getting-Started-with-your-ZED-camera (visited on 11/02/2021).

[32] StereoLab. (2020). "Stereo lab," [Online]. Available: https://www.stereolabs.com/zed/ (visited on 11/02/2021).

[33] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: An open-source robot operating system," vol. 3, Jan. 2009.

[34] J. Schönberger, M. Pollefeys, A. Geiger, and T. Sattler, "Semantic visual localization," in *2018 IEEE Conf. Computer Vision and Pattern Recognition*, Jun. 2018.

[35] M. Fiala, "Vision guided control of multiple robots," in *Proc. First Canadian Conf. Computer. Robot. Vision.*, 2004, pp. 241–246.

[36] Q. Van Tran and H.-S. Ahn, "Distributed formation control of mobile agents via global orientation estimation," *IEEE Trans. Control Network Syst.*, vol. 7, no. 4, pp. 1654–1664, 2020.

[37] S Garrido-Jurado, R Muñoz Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, vol. 47, no. 6, pp. 2280–2292, 2014.

[38] S. Isaac. (2016). "Ros.org," [Online]. Available: http://wiki.ros.org/ar_track_alvar (visited on 11/03/2021).

[39] GCtronic. (2020). "E-puck2 pc side development," [Online]. Available: https://www.gctronic.com/doc/index.php?title=e-puck2_PC_side_development#Connecting_to_the_WiFi (visited on 11/06/2021).

[40] K. Yang, X. Tang, Y. Qin, Y. Huang, H. Wang, and H. Pu, "Comparative study of trajectory tracking control for automated vehicles via model predictive control and robust h-infinity state feedback control," *Chin. J. Mech. Eng.*, vol. 34, no. 74, 2021.

[41] K. Yutaka, K. Yoshihiko, M. Fumio, and N. Tetsuo, "A stable tracking control method for an autonomous mobile robot," in *Proc. Int. Conf. Robot. Autom*, Cincinnati, OH, USA, 1990.

[42] J. Zhong-Ping and N. Henk, "Tracking control of mobile robots: A case study in backstepping," *Automatica*, vol. 33, pp. 1393–1399, 1997.

[43] M. Javier, L. Florent, and L. Jean-Paul, "Motion planning and obstacle avoidance," *Springer Handbook of Robotics*, pp. 827–852, 2008.

[44] M. S. Dušan, F. H. Peter, W. S. Mark, and D. Š. Dragoslav, "Cooperative avoidance control for multiagent systems," *J. Dyn. Syst. Meas. Control .*, vol. 129, pp. 699–707, 2007.

[45] G. Skowronski and J. Leitmann, "Avoidance control," *J. Optim. Theory Appl.*, vol. 23, pp. 581–591, 4 Dec. 1977.

[46] H. H. Wesley, R. F. Brett, R. F. Jonathan, and H. W. William, "Visual navigation and obstacle avoidance using a steering potential function," *Robot. Auton. Syst.*, vol. 54, pp. 288–299, Jan. 2006.

[47] Z. Huidi, L. Shirong, and X. Y. Simon, "A hybrid robot navigation approach based on partial planning and emotion-based behavior coordination," in *IEEE Int. Conf. Intell. Robots Syst.*, Beijing, China, Oct. 2006.

[48] M. Mehdi, A. Lounis, K. Djamel, and M. Philippe, "Mobile robot navigation and obstacles avoidance based on planning and re-planning algorithm," in *10th Int. IFAC Symposium on Robot Control (SYROCO'12)*, Dubrovnik, Croatia, Sep. 2012.

[49] S. Jeon, J. Lee, and J. Kim, "Multi-robot task allocation for real-time hospital logistics," in *Proc. Int. Conf. Syst. Man, and Cybernetics (SMC)*, 2017, pp. 2465–2470.

[50] E. David. (2020). "Robotics and automation," [Online]. Available: https://roboticsandautomationnews.com/2020/08/12/logistics-companies-turning-to-robotics-and-automation-as-way-out-of-coronavirus-crisis/35041/ (visited on 11/09/2021).

[51] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Cooperative Robots and Sensor Networks 2015*, A. Koubâa and J. Martínez-de Dios, Eds. Cham: Springer Int. Publishing, 2015, pp. 31–51.

[52] X. Jia and M. Q.-H. Meng, "A survey and analysis of task allocation algorithms in multi-robot systems," in *2013 Int. Conf. Robot. Biomimetics. (ROBIO)*, 2013, pp. 2280–2285.

[53] M. B. Alatise and G. P. Hancke, "A review on challenges of autonomous mobile robot and sensor fusion methods," *IEEE Access*, vol. 8, pp. 39 830–39 846, 2020.