CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Control and Path Planning of Multi-Agent Systems: Artificial Potential Field Algorithm

A graduate project submitted in partial fulfillment of the requirements

For the degree of Master of Science in Electrical Engineering

By

Patrick Gevargiz

December 2019

The graduate project of Patrick Gevargiz is approved:

_____	_____
Dr. Ali Amini	Date
_____	_____
Dr. Ruting Jia	Date
_____	_____
Dr. Xiaojun Geng, Chair	Date

California State University, Northridge

Table of Contents

LIST OF FIGURES

ABSTRACT

Control and Path Planning of Multi-Agent Systems: Artificial Potential Field Algorithm

By

Patrick Gevargiz

Master of Science in Electrical Engineering

This project proposes a simple, efficient and scalable method for motion planning and control of multi-agent systems (MAS) with collision avoidance capabilities. This report begins with the review of studies and algorithms in the fields of cooperative control of MAS and swarm robotics for collective behavior. In this project, classic Artificial Potential Field (APF) algorithm is utilized to implement control and motion planning for autonomous robots (agents) with obstacle avoidance capabilities. The APF algorithm provides effective and smooth trajectory planning compared with other algorithms that are studied in this project. Furthermore, to address classical APF algorithm limitations such as local minima and narrow channels, a Lyapunov theorem approach, is used to optimize potential fields and resolve those limitations. In addition, a special case of modified APF algorithm is studied to implement leader-follower scenario. The results for each controller are verified using MATLAB simulation. Finally, a survey of existing hardware platform to implement MAS formation control algorithms is presented.

SECTION 1: INTRODUCTION

The control of Multi-Agent Systems (MAS) received a lot of considerations in both academia and industry over last two decades. Recent advancement in computational technologies such as, development of powerful onboard processors, embedded systems improvement, advances in sensor technology and communication systems made commercial use of MAS a real possibility.

The MAS algorithms allow users to control the whole cluster of agents without the need to send commands to each individual agent. This is a critical advantage that is applied in different areas such as, search and rescue missions, satellite formation control, surveillance drones, package delivery robots and military weaponry systems development [1]. In addition, MAS are flexible, and their flexibility allow them to adapt to the dynamic environment. Robustness to failure is another important characteristic of the MAS. If one or few agents failed, the entire system will continue its mission to achieve the collective goal. MAS are also scalable where it allows to apply the control to many agents. Combination of all these features allows the MAS to be considered as a main future technology utilized in swarming spacecraft, terrestrial exploration, satellite formation for planetary surveillance, observation etc. [2]. Recently, new studies demonstrated that MAS algorithms are useful in financial market as well as social behavior forecasting [3].

The first chapter of this report provides a review of the most important collective behavior algorithms that are essential in the development of the MAS. These algorithms have different level of maturity meaning that only few numbers of these algorithms are applied in the real-world applications. All these algorithms are demonstrated in the simulation but small percentage of them have the capability of integrating with current existing hardware technology. Maturity of MAS algorithms is one of the main topics of study for this project. In addition, some important MAS properties such as scalability, robustness are considered to select the best algorithm [2].

The focus of next chapter is on the Artificial Potential Fields (APF) algorithm for the MAS. After a careful study of several important algorithms in the field of MAS and robotic motion planning, APF is an interesting field to investigate. The main advantage of APF algorithm is its simplicity. In addition, APF provides scalability, efficiency, robustness and low

bandwidth in robot communication within a network of agents. This algorithm is among few available algorithms with the highest level of maturity meaning it can be applied to the large group of agents such as robots or UAVs. Another advantage of the APF is in its flexibility to be applied to almost all kinds of collective behaviors.

The APF is a powerful algorithm to be used in implementation of robotic motion control and path planning. The MAS that utilizes APF is used for collective transport, task allocation, area exploration, aggregation and pattern formation. In this project, several of these collective behaviors examined using APF algorithm. APF has some limitations such as Local minima and narrow channels problem. This project will provide a solution to mentioned problems using an optimized APF algorithm.

In chapter four a centralized approach to MAS is studied as well as the modified APF algorithm used to implement leader-follower behavior in the MAS. For each collective behavior that studied in this project, a MATLAB simulation is used to confirm theoretical concepts associated with APF algorithm.

Consequently, in the last chapter of this report, several scalable hardware platforms for MAS is demonstrated. Kilobot and Mona are hardware platforms primarily used by research institutions for development and implementation of collective behavior algorithms. Kilobot is a low-cost MAS platform developed by Harvard University, and Mona is a low-cost open hardware platform developed by Cambridge University, which is used both for robotic as well as MAS researches. A detailed architecture of each of these platforms will be investigated and the advantage of each platform is presented in this chapter.

SECTION 2: SURVEY OF MAS ALGORITHMS

2.1 Overview of MAS Algorithms Classification

Over last two decades' large number of cooperative algorithms for control of MAS were developed. Implementation of collective behavior for a specific project requires an understanding of these algorithms and their mathematical structure. There exist some excellent reviews that compare some of the collective behavior algorithms in control engineering literature. However, these surveys usually limited to a specific collective behavior or MAS field application. In this chapter, the most important collective behavior algorithms reviewed based on the underlying mathematical architecture, application of the algorithm at the same time. In this survey, also robustness and flexibility of each algorithm analyzed. In addition, a survey of maturity of these algorithms presented. Maturity of each algorithm is a very important parameter that indicates if a certain algorithm successfully implemented via hardware in the actual field. In the first section of this chapter few definitions related to MAS collective behavior presented. Some popular classification of MAS algorithms presented as well. Then an overview of each collective behavior algorithm presented. At the last section, these algorithms will be compared based on the criteria developed in this chapter and finally the algorithm of choice Well be selected.

2.2 Classification of Swarm Behavior

The large diversity of definitions and cataloguing in swarm literature usually causes confusion. The best methodology to compare swarm algorithms is to have a clear understanding of the existing classifications of swarm behaviors. In this section, the most popular classification of swarm behaviors will be introduced. Obviously, there are more classes of swarm behavior than the number of behaviors that introduced in this study. However, most of skipped behaviors are redundant or of no significant importance. For example, the hole avoidance, is a swarm behavior that omitted in this section due to the fact that coordinated movement includes that type of behavior.

2.2.1 Pattern Formation

Pattern formation is the creation of the global pattern by the group of swarm agents [15]. This type of behavior can be observed in most of biological phenomenon in nature as well as social behaviors of human societies. It also appears in the formation of astronomical shapes and patterns.

2.2.2 Aggregation

Aggregation is the most elementary behavior in the Multi Agent Systems. This behavior is about the ability of the agents to work together and mobilize themselves to reach a certain location [14]. This behavior is one of the most important collective actions with many examples in nature and a lot of applications in the robotics. There are a lot of solutions proposed for the problem of the aggregation in the robotics. Some of these solutions use innovative methods such as probabilistic controller and genetic algorithm to obtain optimized solution [15]. Aggregation is a natural phenomenon usually occurs in the societies that agents live in close proximity. Aggregation can be interpreted mathematically as a problem of convergence to a singular point [14]. It can be expressed by the following equation:

$$\lim_{t\to\infty}\left\|x_i(t) - x_j(t)\right\| \leq \in \qquad (1)$$

In which the parameter epsilon defines the maximum size of the swarm system.

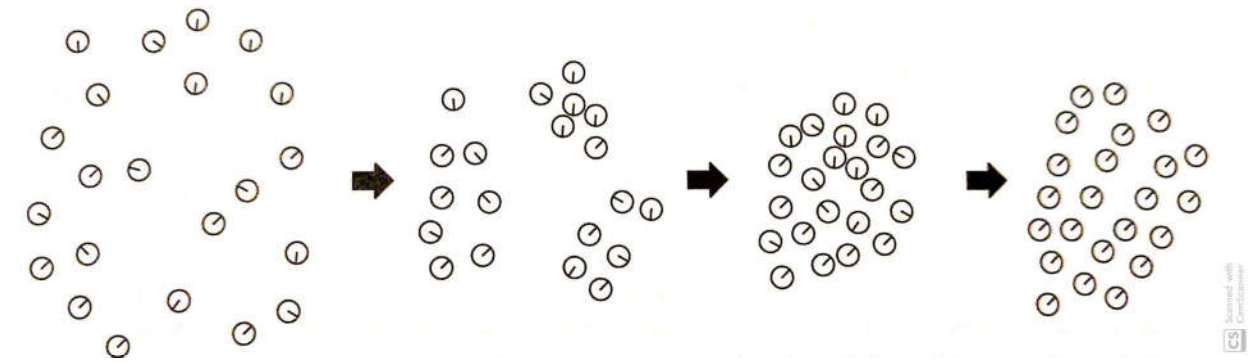

Figure 1: Aggregation of MAS [17]

2.2.3 Social Foraging

Social foraging is another popular collective behavior. It is an indicator of the Aggregation behavior. One of the brilliant solutions to implement social foraging in robotics introduced by Tangamchit [15]. His innovative solution utilizes Monte Carlo learning technique to resolve Social foraging problem. Social foraging behavior is affected greatly by the environment. This behavior observed in many hunter animals in nature.

2.2.4 Flight Formation

Flight formation is a special case of the social foraging. This behavior in particular observed in the foraging of the birds. This behavior in some studies classified independently because of its importance in the study of the algorithms that solve this problem for the Unmanned Aerial vehicles. Flight formation mathematical structure is very similar to the standard aggregation behavior. The only difference is its geometrical configuration in three-dimensional space. It is still trying to converge to a singular point. In the case of UAVs, it can be interpreted as a swarm of the drones trying to reach to a same destination.

$$\lim_{t\to\infty}\left|\left\|x_i(t)-x_j(t)\right\|-d_{ij}\right|\leq\in \tag{2}$$

Flight formation is the combination of the aggregation and foraging behaviors [14]. The steady state characteristic of the formation system in this type of swarm behavior formulated as equation (2). Figure 2 shows one thousand drones that create TIME magazine cover [16]. This is an interesting modern application of flight formation.

Figure 2: Formation of the TIME Magazine logo by group of UAVs [16]

2.2.5 Chain Formation

Chain formation is one of the popular formation behaviors among animals and insects. Ants and elephants' chains are the most well-known type of the chain formation in nature. In the robotic systems, chain formation can be useful when bunch of robots trying to navigate through a narrow

corridor.in the chain formation agents try to find the end of the available chain. They will stay in the chain until the required time to remain in the chain ends [15].

Figure 3: Chain Formation of Robots while passing a corridor [18]

2.2.6 Self-Assembly

Self-assembly is defined as forming complex assemblies from large quantity of fairly simple agents that interact in the vicinity of each other [15]. Self-assembly allows natural phenomena to build sophisticated arrangements, from multi-cellular bacteria to complex animal assemblies such as flocks of birds.

2.2.7 Coordinated Movement (Flocking)

Coordinated movement is one the most important and well-studied swarm behaviors. Coordinated movement has a significant importance among the animal's societies. Flocks of birds and schools of fish are the most well-known swarm behavior in the nature. The goal of this behavior is to keep a global pattern among the agents while they are moving [15]. Collison avoidance is a key factor in coordinate movement. In addition, velocity matching plays an important role in stability of the system. Craig Reynolds in 1986 was the first researcher who studied this behavior and developed an algorithm to solve the problem associated with Flocking.

2.3 Taxonomy of the MAS Popular Algorithms

There are many algorithms developed for the formation control of MAS. Some of these algorithms well-studied and they have the highest level of maturity meaning that they implemented in the industry. for example, distributed feedback control or artificial potential fields algorithms which have high level of maturity and implemented in the fields. Some of these algorithms such as source-searching, convex optimization and behavior composition are in a vet initial phase of development. They are only simulated in the programming environments such as MATLAB. In this section, a brief review of well-known MAS algorithms that have relatively higher level of maturity will be presented.

2.3.1 Consensus Algorithm

Consensus problem first studied in the 1960s with some applications in the management science and statistics theory which later formally named statistical consensus theory [21]. Therefore, this algorithm is one of the oldest in the field of formation control of the MAS. Consensus means a group of independent agents reach to the agreement on the mutual value by cooperating with each other over a communication link [19]. The consensus algorithm is a set of rules that governs the interactions between multiple agents and the information exchange between them [21]. Swarm behaviors such as Aggregation, Pattern Formation and social foraging can easily implemented by this algorithm. This algorithm can implement highly scalable MAS systems with low network bandwidth use. There are various simulations for Consensus algorithm as well as well implemented hardware platforms that demonstrate capabilities of this algorithm. There is still no known field implementation for this algorithm [2].

2.3.2 Artificial potential Functions (APF)

The Artificial Potential Field (APF) is one of the leading algorithms and techniques in the field of Multi Agent Systems (MAS) control. The classic APF algorithm first introduced by O. Khatib in 1986 to make autonomous robots with obstacle avoidance capabilities [4]. This algorithm controls the agents in the environment using the gradients of the potential fields associated with the target and obstacles.

Swarm behaviors such as Aggregation, Pattern Formation and social foraging can easily implemented by this algorithm. This algorithm can implement highly scalable MAS systems with low network bandwidth use. There are various simulations for APF algorithm as well as

well implemented hardware platforms that demonstrate capabilities of this algorithm. This algorithm successfully implemented in the fields and commercial systems [2].

2.3.3 Distributed Feedback Control Algorithm

The distributed feedback algorithm is some control theory literature known as LQG control is a mature algorithm that implemented in the commercial applications. This algorithm uses feedback controller that takes the state of each agent in the MAS system as its input [2]. The biggest advantage of this algorithm is its capability to operate in noisy environments. This algorithm can implement aggregation and pattern formation behaviors. This algorithm implemented in the area exploration robots such as UAVs that are used in search and rescue operations. Distributed Feedback algorithm is a highly scalable algorithm that utilizes low traffic in communication network of the MAS [2].

2.3.4 Bio-inspired MAS Algorithms

Bio-inspired algorithms are collection of various algorithms that directly inspired from the collective behaviors among the animals and micro-organisms. Algorithms such as Fish-inspired food probing, Kilobot self-assembly, Beeclust foraging, Mergeable modular agents and Gillespie self-assembly agents are just few examples of bio-inspired algorithms. Virtual Pheromone (VP) and Cardinality Algorithms are ant-inspired algorithms that mimic the foraging behavior of the ants. These algorithms utilize a simple local communication link between neighboring agents within small proximity. VP and Cardinality algorithms provide a powerful navigation scheme by marking environment by some agent designator. In these algorithms the agent is either walker or environment designator (beacons) [24]. All of these algorithms are highly scalable. The maturity of this algorithm is either computer simulation or hardware implementation. There is no filed implementation for these algorithms yet [2].

2.3.5 Distributed Optimization Algorithms

This is another family of algorithms in the field of cooperative control of MAS. These set of algorithms include Distributed Linear Programming (DLP), Distributed Convex Optimization (DCO), Distributed Dynamic Programming (DDP) and Sequential Convex Programming (SCP) [2]. These algorithms are generally used for pattern formation. The highest maturity of these algorithms is hardware implementation but still there is no field implementation.

2.3.6 Cooperative Hole Avoidance Algorithm

Cooperative hole avoidance algorithm has its own classification because of the difficulty associated to the hole avoidance problem. In general, Natural holes in the MAS environments are hard to detect. It is very difficult to avoid holes specially if agents move in high speed. Usually holes detected when the agents reach the edge of the hole. These algorithms require data that originated from the utilization of two types of sensors such as fraction and ground clearance sensors [22]. In addition, this algorithm utilizes evolutionary algorithm for the purpose of movement control in high speeds and hole avoidance property. This algorithm implemented using the S-bot swarming development platform [22]. Figure 4 shows the example of hole avoidance algorithm that also utilizes evolutionary algorithm to form a chain of agents to avoid falling into the hole.

Figure 4: Hardware Implementation of Hole Avoidance Algorithm using S-bot Platform [22]

2.3.7 Minimalist Flocking Algorithm

Minimalist flocking algorithm is an algorithm that developed for simple MAS systems. This algorithm inspired by the swarm behaviors in simple microorganisms. The minimalist algorithm designed for agents that are so simple that are not aware of global information about their colony

and have a very simple microcontroller (computational power) on board [23]. This algorithm only utilizes simple IR-sensors in the MAS hardware implementation. The minimalist algorithm doesn't require any global information regarding the heading or position of the agents [23]. This algorithm is built on the regular consensus and flocking algorithms, but it requires less information and it provides low communication traffic on the MAS network. Figure 5 shows the utilization of four IR-sensors in a simple swarm robot. Each decision in this chat represents a certain swarm behavior. For example, the first decision provides collision avoidance. The second and the third decisions provide flocking and aggregation behavior of the robot.

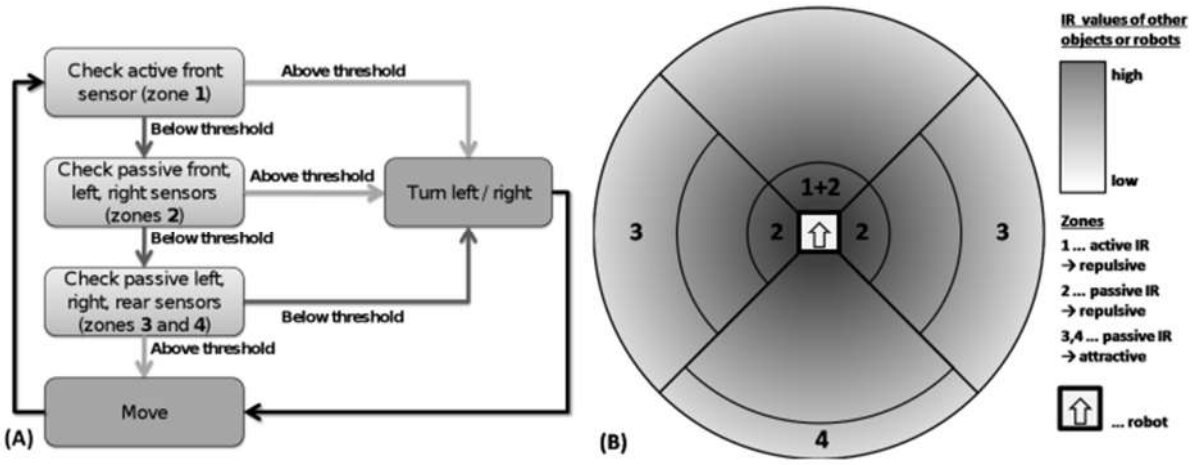

Figure 5: Minimalist Algorithm for a robot with 4 IR-sensors [23]

2.4 The algorithm of choice

The goal of this chapter was a brief review of the available swarm algorithms that can be implemented in the formation control of the MAS with obstacle avoidance capability. Therefore, a lot of literature in the field of MAS reviewed. Finally, after careful evaluation of multiple algorithms, APF algorithm was chosen for the detail review and simulation for this thesis. APF has the highest maturity among all the MAS algorithms that developed over last decades. All swarm behavior can be implemented by the APF algorithm except goal searching capability. A summary of the MAS algorithms assessment is introduced in the figure 6.

	Aggregation	Pattern Formation	Coverage	Area Exploration	Goal Searching	Task Allocation	Collective Transport	Motion Planning	Distributed Estimation	High Scalability	Low Bandwidth Use	Maturity
Consensus	✓	✓	✓						✓	✓	✓	H
Artificial Potential Functions (APF)	✓	✓	✓	✓		✓	✓	✓		✓	✓	F
Distributed Feedback Control	✓	✓							✓	✓	✓	F
Geometric Algorithms												
Voronoi-based Algorithms	✓		✓	✓				✓		✓	✓	H
Circumcenter Algorithms	✓	✓								✓	✓	S
Bearing-only Algorithms	✓	✓								✓	✓	H
Maze Searching Algorithms								✓		✓	✓	S
Leader-Follower (LF) Algorithms		✓								✓	✓	S
Velocity Obstacle (VO) based Algorithms								✓		✓	✓	F
State Machines and Behavior Composition												
Automata-based Algorithms						✓			✓	✓	✓	S
Behavior Composition						✓	✓					H
Petri Networks						✓					-	H
Game Theory based Algorithms						✓					-	S
Resource Allocation Systems								✓		✓	-	S
Bio-Inspired Algorithms												
Kilobot Self-Assembly Algorithm		✓								✓	✓	H
Optimotaxis Source-Searching Algorithm					✓					✓	✓	S
Beeclust Foraging Algorithm				✓						✓	✓	S
Shepherding Algorithm	✓									✓	✓	S
Termite-Inspired Collective Construction Algorithm						✓	✓			✓	✓	H
Fish-inspired Goal Searching Algorithms		✓			✓					✓	✓	H
Gillespie Self-Assembly Algorithm		✓								✓	✓	H
Mergeable Modular Robots		✓								✓	✓	H
Density based Control												
Markov Chain-based Algorithms		✓	✓			✓				✓	✓	H
Smoothed Particle Hydrodynamics (SPH)		✓	✓							✓	✓	H
Optimal Transport based Algorithm		✓						✓		✓	✓	S
Distributed Optimization Algorithms												
Distributed Linear Programming		✓				✓				✓	✓	S
Distributed Convex Optimization		✓				✓			✓	✓	✓	S
Distributed Dynamic Programming						✓		✓				H
Sequential Convex Programming								✓		✓	✓	H
Distributed Auction						✓				✓	✓	H
Local Optimization Algorithms for Global Behavior												
Decentralized Model Predictive Control (DMPC)		✓						✓		✓		H
Formal Methods								✓		✓	✓	S
Sampling-based Motion-Planning Algorithms								✓				H
Centralized Optimization Algorithms												
MILPs and MINLPs		✓				✓		✓	✓		-	H
Linear and Convex Optimization						✓		✓	✓	✓	-	S
Markov Decision Processes (MDP)						✓		✓			-	H
Multi-Agent Traveling Salesman Problems			✓	✓	✓	✓					-	H
Multi-Armed Bandits				✓	✓	✓			✓	✓	-	S
Direct Methods for Optimal Control			✓	✓				✓			-	F
Multiagent Reinforcement Learning				✓		✓					-	H
Frontier Techniques				✓	✓						-	F
Network Flow Algorithms						✓		✓		✓	-	S
Combinatorial Motion Planning								✓		✓	-	S

Figure 6: A Summary of MAS algorithms Assessment Based on Mathematical Structure and Implementation Capabilities [2]

SECTION 3: ARTIFICIAL POTENTIAL FIELDS ALGORITHM

3.1 Overview

The Artificial Potential Field (APF) is one of the leading algorithms and techniques in the field of Multi Agent Systems (MAS) formation control. This algorithm is comprehensively used in the implementation of autonomous robots/UAVs path planning, localization, navigation and mapping. The classic APF algorithm first introduced by O. Khatib in 1986 to make autonomous robots with obstacle avoidance capabilities [4]. Later on, in 1992, Koditschek and Robin introduced navigation functions as a new and powerful approach to address problems within Khatib methodology [5]. Over last two decades a lot of improvement has been done on the classic APF and new optimized potential field has been introduced.

APF is flexible algorithm and can be applied to large number of collective behaviors such as Aggregation, flocking and pattern formation. APF has capability to be applied for area exploration. This is a crucial advantage that make APF as an algorithm of choice for surveillance and rescue robots and UAVs. APF is a suitable algorithm for cooperative decision making. This algorithm is very flexible to implement task allocation, collective transport which is ideal for autonomous warehouse robots.

Motion of the agent in APF algorithm can be compared with the motion of the electric charges in the electric field similar to Figure 7. Agents can be compared with positively charged particle. A destination (target) in this analogy can be compared with the negatively charged particle. Obviously in this scenario obstacles must be considered as positive charges. There are two static and dynamic approach for this scenario. For simplicity we consider a static scenario in which goal and obstacles are stationary. Goal and obstacles create a potential electric field similar to the real electric field. The goal is to move robot/agent to its target/goal within the potential fields created by the obstacles. The force between agent and target is attractive because positive and negative charges attract each other. The force between agent and obstacles are repulsive. The combination of all forces caused by electric fields in the agent environment will determine the total force on the agent/particle. This total force will determine the heading angle of the moving agent. By using appropriate feedback mechanism and control parameter we can guide agent toward its target. Obviously, we can imply same process for the family of agents and

extend this algorithm to a collection of agents within MAS.

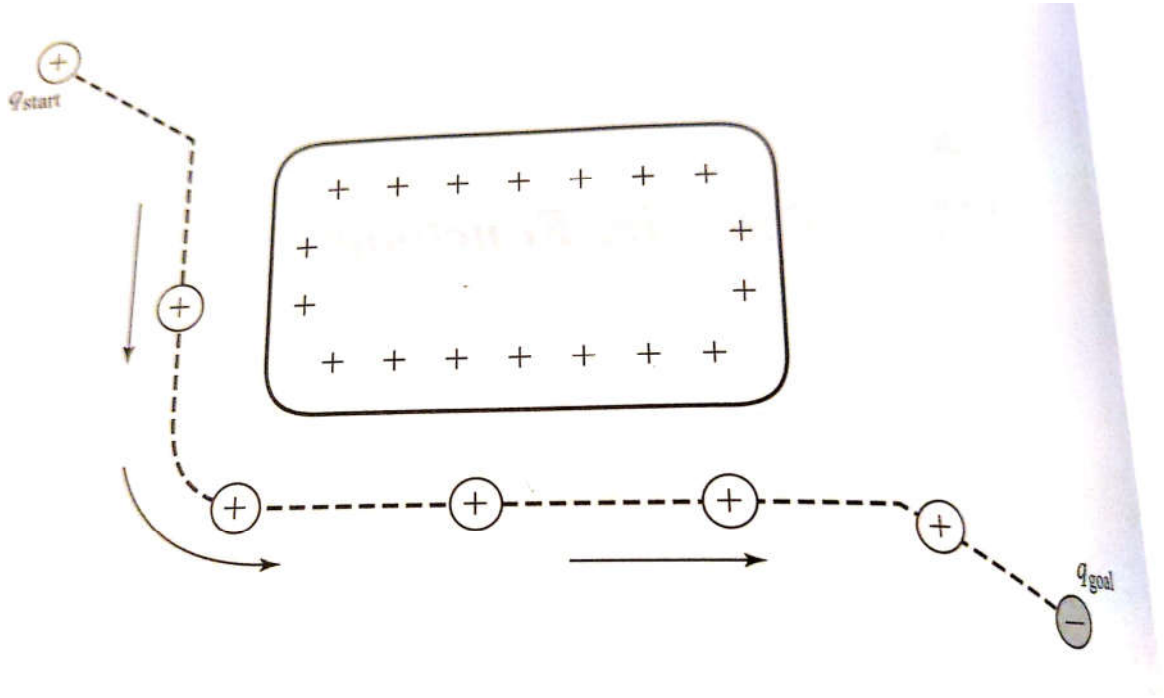

Figure 7: Positive Charge in Positive Potential Field

3.2 Two-dimensional model

Assume an agent/robot is moving in two-dimensional space(R^2). The position of the agent is given by vector $\vec{r}$ at any arbitrary point in the2-dimensional space. The artificial potential field in the moving agent environment is a scalar function that relates the position vector of the agent to a scalar number that represents potential field magnitude at the point. The superposition of potential fields due to target and obstacle at the momentary position of the agent can be written as:

$$U(\vec{r}) = U_{attractive}(\vec{r}) + U_{repulsive}(\vec{r}) \quad (1)$$

Since repulsive potential obtained from the superposition of all obstacle's potential fields therefore the equation (1), can be written in the following form:

$$U(\vec{r}) = U_{attractive}(\vec{r}) + \sum_{i=1}^{n} U_{repulsive\ of\ obstacle\ i}(\vec{r_i}) \quad (2)$$

Since in our scenario there is only one attractive potential filed which is associated with target, we can change the indices of equation (2), as following:

$$U(\vec{r}) = U_{target}(\vec{r}) + \sum_{i=1}^{n} U_{obstacle\,i}(\vec{r_i}) \quad (3)$$

These potential fields produce forces that effect the motion of the agents in the field. We can write a similar equation to equation (3) that represents the total force on the agent in the given point by the position vector.

$$F(\vec{r}) = F_{target}(\vec{r}) + \sum_{i=1}^{n} F_{obstacle\,i}(\vec{r_i}) \quad (4)$$

The total force given by equation (4), determines the heading angle of the agent toward the target. In general, the relationship between potential filed and the force associated with that potential field is given by

$$\vec{F}(\vec{r}) = -\nabla \mathrm{U}(\vec{r}) \quad (5)$$

This means that the component of the force at any point in the plain can be obtained by taking negative of the gradient of potential field at that point. We use this important relation to obtain all necessary equation that describe the motion of the agents in their environment. There is an important requirement for equation (5). Potential functions defined for goal and obstacle should be differentiable. It is very important how to define the scalar function for potential field at the location of goal and each individual obstacle. There is a common method that first defined by Khatib in his paper when he first introduced APF algorithm. The potential function for the goal is defined as a function that have zero value at the target location. The absolute value of the potential filed will increase far away from the target. For the obstacles we follow the opposite criteria. Potential filed at the location of each obstacles is maximum meaning that the maximum repulsive force will be inserted on the object getting closer to the obstacle. The potential filed of the obstacles decreases far away from the obstacle. Using this methodology, we can define variety of potential fields for target and obstacles. To obtain force at any location in the plain we take negative gradient of both sides of the equation (3).

$$-\nabla U(\vec{r}) = -\nabla U_{target}(\vec{r}) + \sum_{i=1}^{n} -\nabla U_{obstacle\,i}(\vec{r_i}) \quad (6)$$

The combination of equations (3) and (5) can be written as:

$$\vec{F}(\vec{r}) = -\nabla U_{target}(\vec{r}) + \sum_{i=1}^{n} -\nabla U_{obstacle\,i}(\vec{r_i}) \tag{7}$$

Gradient vector for the three-dimensional potential function can be written as:

$$\vec{\nabla} U(\vec{r}) = \left[\frac{\partial U}{\partial x} \quad \frac{\partial U}{\partial y} \quad \frac{\partial U}{\partial z}\right]^T = [\, F_x \; F_y \; F_z] \tag{8}$$

The force vector given in above equation will determine the total force at any point in the plain represented by the position vector$\vec{r}$. Figure 8 shows the visualization of the potential fields in the three-dimensional space. Part (a) shows the two-dimensional cut of the space with three obstacles. Part (b) shows the potential functions representation and Part (c) shows the contour representation of the potential fields for the surface energy. Finally, part (d) represents gradient vectors for each of the obstacles potential function [13].

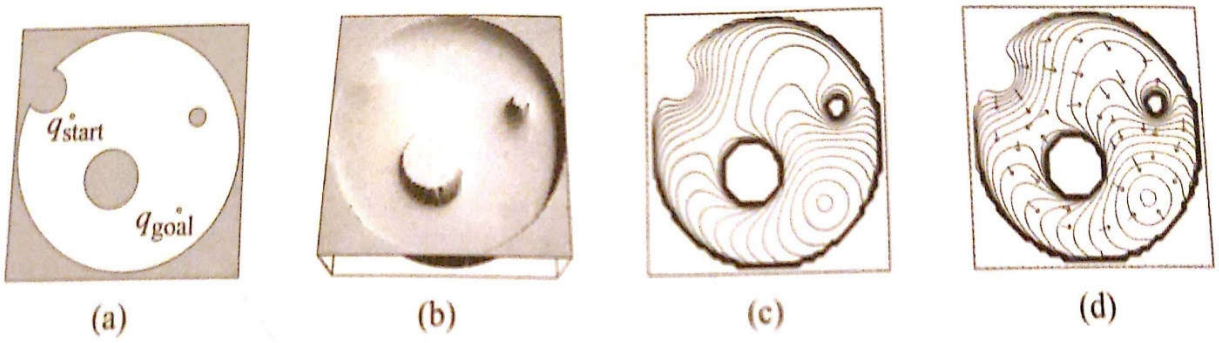

Figure 8: Potential Fields Visualization

3.3 Simple model for potential fields:

We start our analysis by using a very simple method to simulate potential fields associated to the target and obstacles. In this model we will use a linear function that represents the attractive potential field of the target. The potential field of the target in linear model is given by$U(r) = k_{tgt}r$. It worth mentioning that variable r is scalar. In APF theory, we assume potential field as scalar quantities. The k_{tgt} is a gain associated with the potential field of each target. The variable r in this equation represents the distance from agent (robot) position to the target.

Therefore, using distance formula for any given points in the two-dimensional space, we can write the following equation:

$$U(r) = k_{tgt} r = k_{tgt}\sqrt{(x - x_{tgt})^2 + (y - y_{tgt})^2} \tag{9}$$

In above equation, x_{tgt} and y_{tgt} represents the coordination of the target (goal) in two dimensional coordinates. By applying gradient vector and using chain rule we can use equation (8), to obtain force components at any given point (x, y) in the two-dimensional space.

$$F_{tgtx} = -\frac{\partial U}{\partial x} = \frac{-k_{tgt}(x - x_{tgt})}{\sqrt{(x - x_{tgt})^2 + (y - y_{tgt})^2}} \tag{10}$$

$$F_{tgty} = -\frac{\partial U}{\partial y} = \frac{-k_{tgt}(y - y_{tgt})}{\sqrt{(x - x_{tgt})^2 + (y - y_{tgt})^2}} \tag{11}$$

The same method can be applied to determine forces that are associated with the obstacles. A simple repulsive potential field is defined for each of the obstacles. We should note that we can define a separate potential field for each obstacle in the agent's environment or we can define a unified potential field for all obstacles but differentiate them using different repulsive gain. Through this project various types of the potential fields are studied. In this section, we chose a unified simple repulsive potential field to begin the analysis of the MAS.

A simple repulsive potential filed $U(r) = \frac{k_{obs}}{r}$ is defined for obstacles in this model. The k_{obs} is a gain associated with the potential field of the obstacle (object). The variable r in this equation represents the distance from agent (robot) position to the obstacle. Therefore, using distance formula for any given points in the two-dimensional space, we can write the following equation:

$$U(r) = \frac{k_{obs}}{r} = \frac{k_{obs}}{\sqrt{(x - x_{obs})^2 + (y - y_{obs})^2}} \tag{12}$$

In above equation, $x_{obs}\ and\ y_{obs}$ represents the coordination of the obstacle (object) in two dimensional coordinates. By applying gradient vector and using chain rule we can use equation (8), to obtain force components at any given point (x, y) in the two-dimensional space. Using reciprocal derivative rule and chain rule we obtain gradient of the potential filed at each x and y direction.

$$F_{obsx} = -\frac{\partial U}{\partial x} = \frac{-k_{obs}(x - x_{obs})}{[(x - x_{obs})^2 + (y - y_{obs})^2]\sqrt{(x - x_{obs})^2 + (y - y_{obs})^2}} \tag{13}$$

$$F_{obsy} = -\frac{\partial U}{\partial y} = \frac{-k_{obs}(y - y_{obs})}{[(x - x_{obs})^2 + (y - y_{obs})^2]\sqrt{(x - x_{obs})^2 + (y - y_{obs})^2}} \tag{14}$$

Equations (10), (11), (13) and (14) are well describe, the effect of target and obstacles on the movement of the agents in the environment. These equations also provide a relatively powerful tool to navigate agents (robots) toward their predetermined goal. Agent's physical model need to be specified to be able to use force equations given before. In the following section an agent's physical model that is useful in real field applications will be introduced. The combination of the physical model given in the next section and the potential fields introduced in this section will provide a complete and simple APF model that can be simulated using MATLAB or other simulation tools.

3.4 Nonholonomic agent navigation and path planning using APF algorithm

In this section a very useful and practical model for MAS control and path planning will be introduced. In this model agents are considered to be a simple robot. These robots operate similar to a simple car. These systems of simple robots called Nonholonomic system meaning these agents (robots) have limited turn capabilities. These robots can only turn in finite angles and in specific directions.

Nonholonomic is a general concept in mathematics and it is associated with the systems that their state depends on the trajectory taken by the agents to reach that state. The limitations of the nonholonomic system may prevent them to follow the exact path determined by the potential

fields defined in the previous section [7]. To resolve this limitation, a feedback system will be introduced in the next section that will keep agents in the overall trajectory determined by the target and obstacles potential fields.

3.4.1 Agents (robots) dynamics

The dynamics of a nonholonomic agent can be described by three nonlinear differential equations. Figure 9 shows the angles that define the moving direction of the agent. The first angle isθ, which called the heading angle.

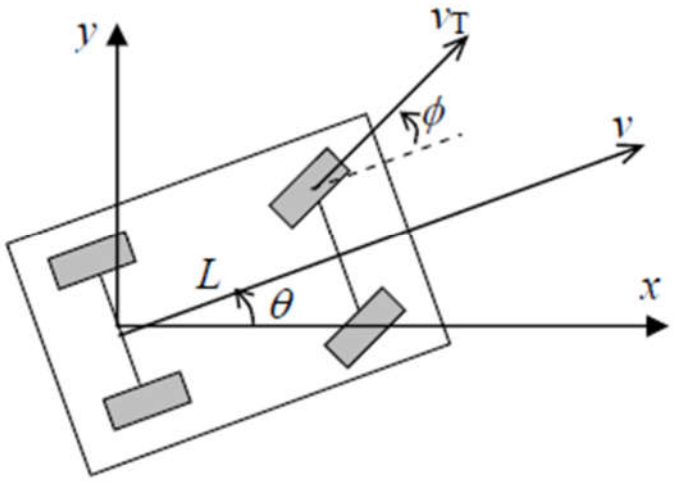

Figure 9: Nonholonomic Robot

It determines the overall heading and direction of the agent in the field. The second important angle is steering angle∅. In this model steering angle will provide a control mechanism for the agent. By changing the steering angle at any point based on the direction of the forces we can navigate robot to the anticipated heading angle which take robot to its goal. In this model, V_{agent} represents the velocity of the agent and L is the wheel base length. The agents are modeled using the subsequent nonlinear differential equations:

$$\dot{x} = \frac{dx}{dt} = V_{agent} \cos\theta \cos\varphi \tag{15}$$

$$\dot{y} = \frac{dy}{dt} = V_{agent} \sin\theta \cos\varphi \tag{16}$$

$$\dot{\theta} = \frac{d\theta}{dt} = \frac{V_{agent}}{L} \sin\varphi \tag{17}$$

In this model $x\ and\ y$ represent the location of the agent in the two-dimensional space. The next step in the APF modeling of MAS, is to solve nonholonomic system presented in the equations (15) through (17). This process requires solving the nonlinear ordinary differential equation. The required step in this process are introduced in the next section.

3.4.2 Solving the nonholonomic model for MAS using APF algorithm

In the previous section a model for nonholonomic agents that have a practical use in simulation and implementation of MAS robots has been introduced. Also, APF model that governs the motion of the MAS in the two-dimensional environment that includes multiple agents has been presented. In this section, multiple scenarios will be offered to examine the presented model. Also, a feedback mechanism that will control agents through their trajectory toward target will be presented.

In the previous section, we considered one agent in the environment consisted of one target and one obstacle. Obviously, the real world is different, and MAS have to deal with large number of obstacles scattered in the field. Therefore, the following technique that encounter various obstacles implemented. We assume an environment that has multiple obstacles scattered in two-dimensional space. The location of the ith obstacles is given by$(x_i\ , y_i)$.

$$F_{obsx} = \sum_i F_{obsix} = \sum_i \frac{-k_{obsi}(x - x_{obsi})}{[(x - x_{obsi})^2 + (y - y_{obsi})^2]\sqrt{(x - x_{obsi})^2 + (y - y_{obsi})^2}} \quad (18)$$

$$F_{obsy} = \sum_i F_{obsiy} = \sum_i \frac{-k_{obsi}(y - y_{obsi})}{[(x - x_{obsi})^2 + (y - y_{obsi})^2]\sqrt{(x - x_{obsi})^2 + (y - y_{obsi})^2}} \quad (19)$$

After the calculation of all the forces associated with each obstacle presented in the environment, the total force excreted on the agent can be calculated using the following equation. It is worth mentioning, that equation of the target force remains the same as equations (10) and (11).

$$F_x = F_{targetx} + \sum_i F_{obsix} \tag{20}$$

$$F_y = F_{targety} + \sum_i F_{obsiy} \tag{21}$$

The new angle is defined that relates these total forces and the direction of the agent all together. It is called force angle and denoted byα. This angle is a key tool that help agents to remain within their trajectory. It is also a guideline for the agent to reach their target while avoiding obstacles in their route toward the goal. The angle alpha is defined by the following equation.

$$\alpha = \tan^{-1}\left(\frac{F_y}{F_x}\right) \tag{22}$$

The alpha angle of force is a key element in the navigation of the agents. In the next section we will define the feedback mechanism that will compare heading angle and force angle to correct trajectory of the robot toward its end point.

3.5 Feedback control of MAS using APF algorithm

This section introduces a feedback mechanism for the control of the agents' motion in their trajectory on the way to the target. The control system of the MAS using APF algorithm is a nonlinear dynamical system. The behavior of this system changes as the agent (robot) moves in its environment. These agents have to deal with external stimulations that generally produced by the obstacles in the environment. Therefore, it is crucial to design a control system that can track all possible changes in the trajectory of the object toward its end point. This control mechanism should be capable of disturbance rejection. This property is crucial when the agents are moving inside dynamical surroundings in which obstacles are moving. A good example of such a case is navigation of the UAVs in the environment that other UAVs are presented. These kinds of systems are super complex. A large number of institutions and corporations are leading research and development projects to implement a practical navigational system in these complex environments [8].

Another important feature in control of MAS is reference tracking. In this project path planning of the MAS is one of the main goals of the algorithm design. It is important that agents

keep around their predefined trajectory and reach their goal without any collision with other agents or obstacles. Considering all these criteria a simple proportional controller designed for this system. The control input to the agent (robot) chose to be a steering angle∅. The goal is to keep heading angle within the proximity of the force angle to keep the agent within the correct trajectory that take agents to their target. Therefore, the control input to the system is defined as following.

$$\emptyset = K_p|(\alpha - \theta)| \tag{23}$$

The control law designed in this section keep the agent in their best route toward the target. The combination of forces implied on the agent will guide agents in their trajectory. The governing equation of the motion and control law are determined using APF algorithm. In the next step the simulation of this control system will be implemented using MATLAB.

3.6 MATLAB simulations

In this section, MATLAB simulation for the APF algorithm discussed previously will be presented with the result of each simulation. In the preceding deliberations we determined necessary functions that represent the force of each potential field on the agent at ant arbitrary point in the agent environment. The superposition of the forces in any point in the two-dimensional space can be used to implement formation control of the MAS using APF algorithm.

3.6.1 Potential Fields Simulation

The first step will be the simulation of the MAS system which includes the simulation of the environment of the agents. Most of the simulation for simplicity will be implemented in the two-dimensional space. But to get a better visual understanding of the environment of the agents and to understand the potential fields gradients and related forces it is better to simulate the potential function in the three-dimensional space. Figures 10 and 11 show the simulation of the environment with two obstacles and potential fields representation of each obstacle.in this simulation, the “meshgrid (x, y)” function of MATLAB utilized to simulate artificial potential fields in three-dimensional space. We can change the KG and Ko coefficient values to obtain potential fields with various strength. Also by changing the coordination (xf, yf) we can change the coordination of the potential fields in the three-dimensional space.

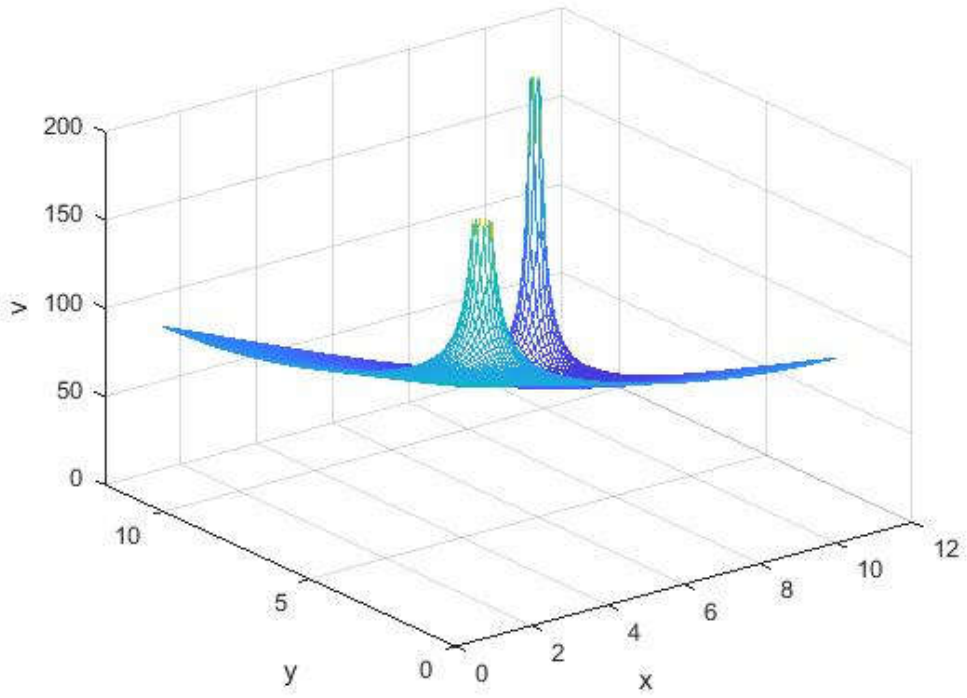

Figure 10: Simulation of the Potential Fields in MATLAB

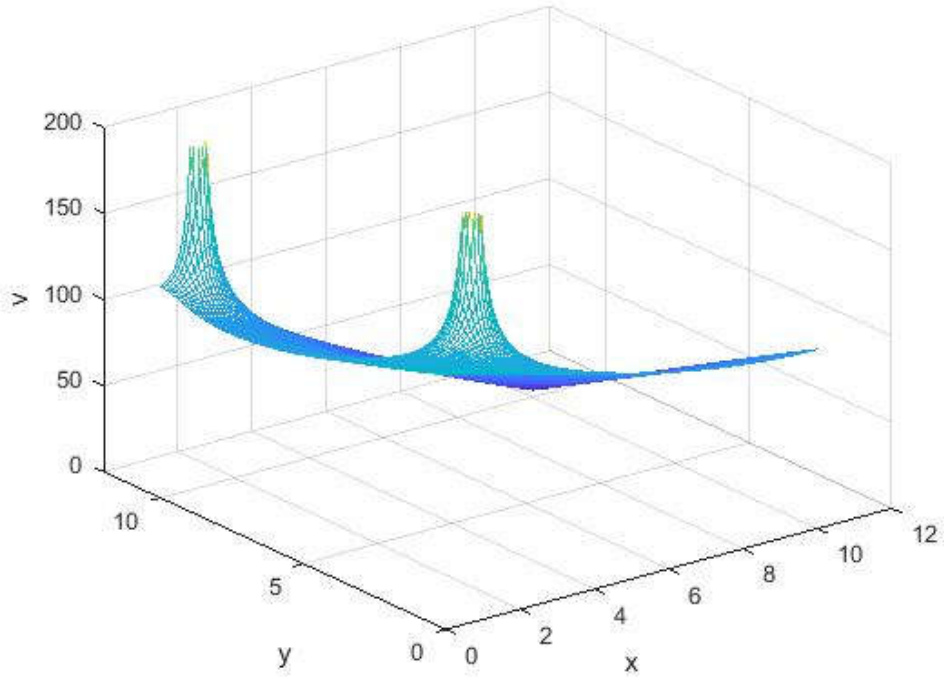

Figure 11: Simulation of Potential Fields, Various Locations

3.6.2 Multiple Agents One Obstacle

In this scenario we will examine the effect of the increasing number of the agents in a simple environment setup that includes only one obstacle. In each simulation run we will add one agent to the environment. We will not change the location of the obstacle and the location of the target. In the real word the effect of the growing number of agents in the environment has an immediate growth in the network traffic and possibility of the interference and increasing level of the noise in the system. Figure 12 shows the trace of the moving agent in the environment with one obstacle. As we can see in the simulation result the effect of the obstacle as a repulsive force causes the trajectory change of the agent.

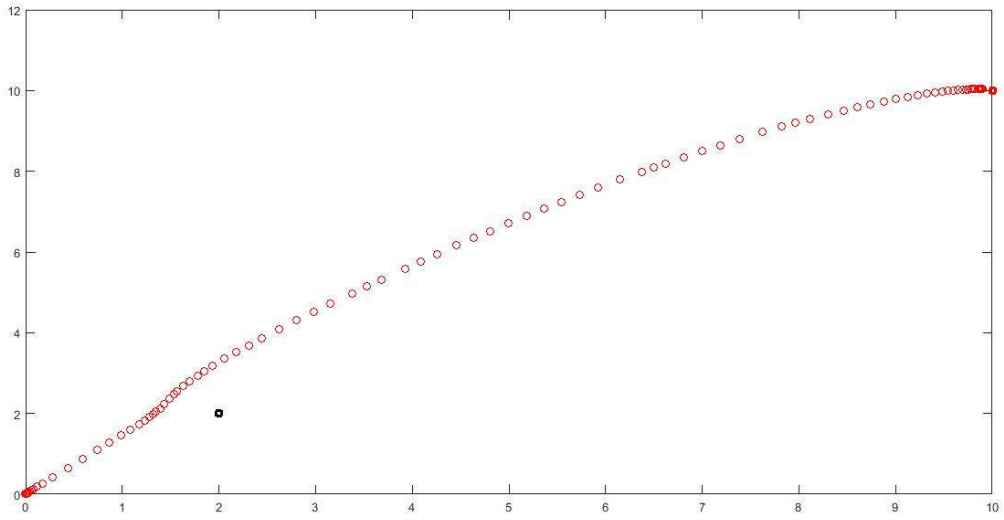

Figure 12: One Agent One Obstacle MATLAB Simulation

In the next simulation, we will add one agent to the system. The second agent represented by green color to distinguish it from the first agent. These agents share very similar trajectory, but they don't collide with each other. We defined a critical neighborhood for each agent that prevents collision with other agents.

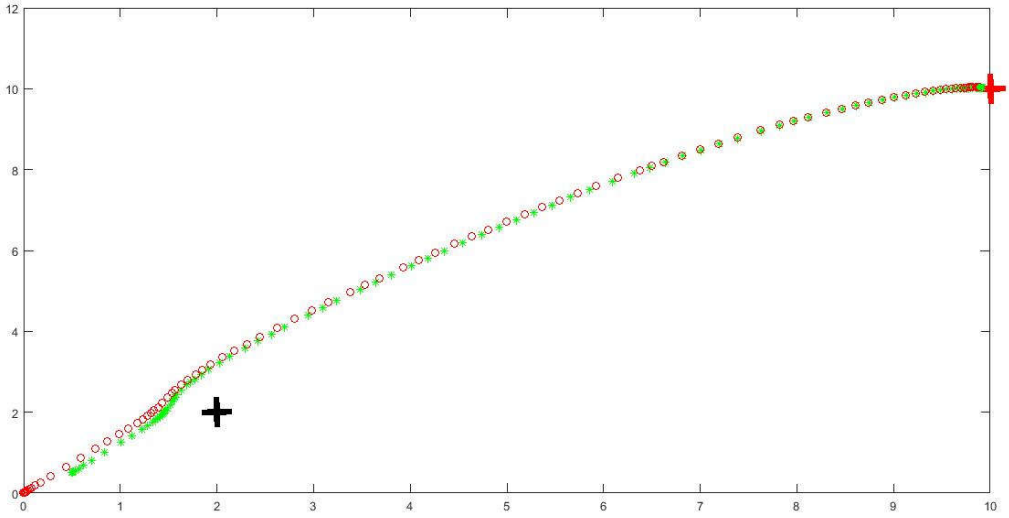

Figure 13: Two Agents One Obstacle MATLAB Simulation

Finally, we add three more agents to the system to increase number of agents to five. This way we create a more complex environment for our agents. Figure 14 shows the result of this simulation.

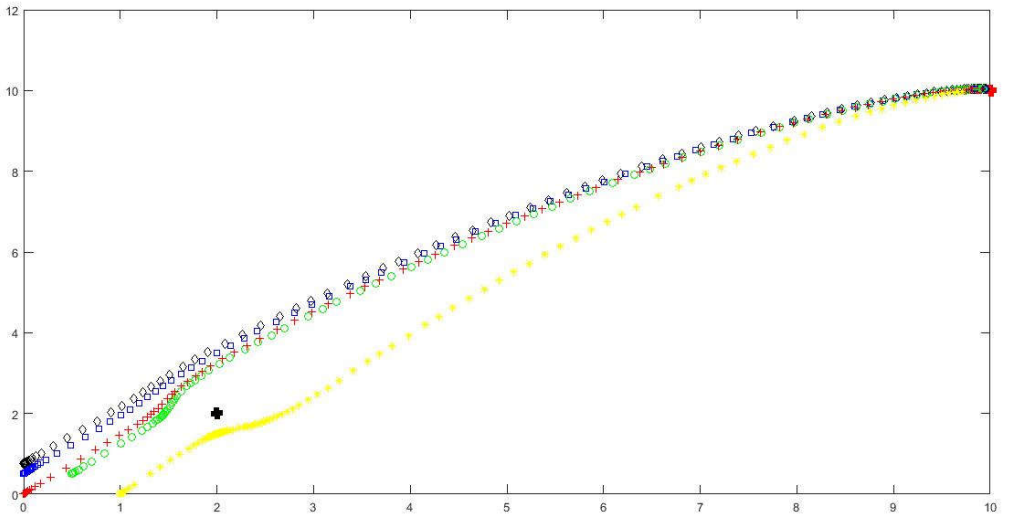

Figure 14: Three Agents One Obstacle MATLAB Simulation

3.6.3 One Agent Multiple Obstacle

In the next phase of the simulation we will consider cases with multiple obstacles but one agent. We will study the behavior of the agent when it is passing through the fields with several potential fields. We investigate the effect of the multiple forces of the repulsive potential fields on the trajectory of the agent. The first case we study includes one agent and two obstacles. As shown in the Figure 15 The obstacles denoted with black dots and the target shown with the red dot with (10, 10) coordination in the two-dimensional space.

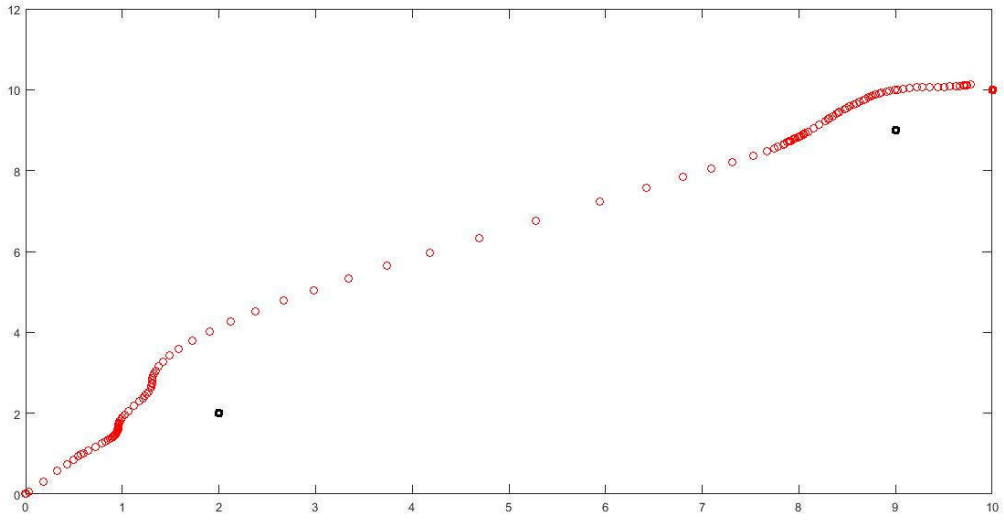

Figure 15: One Agent Two Obstacle MATLAB Simulation

The second case we study includes one agent and two obstacles. As shown in the Figure 10 The obstacles denoted with black dots and the target shown with the red dot with (10, 10) coordination in the two-dimensional space.

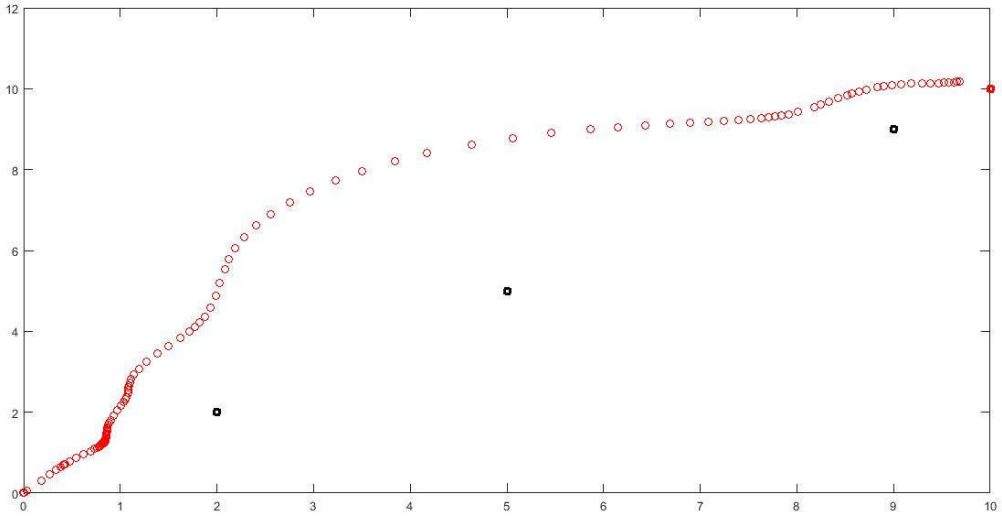

Figure 16: One Agent Three Obstacle MATLAB Simulation

3.6.4 One Agent One Wall

This is the case that multiple obstacles are within the proximity that can be consider as a wall.in the first case we run the simulation with one agent and the obstacles that form the wall (Figure 17).

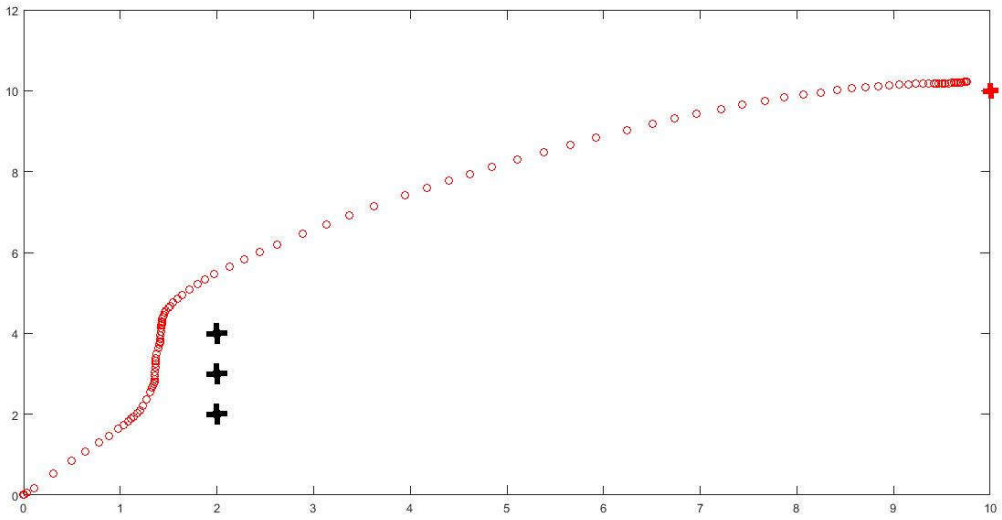

Figure 17: One Agent One Wall MATLAB Simulation

3.6.5 One agent and two walls

This is a complex scenario in which agent should pass over several obstacles in multiple locations. These obstacles are within proximity that form two different walls in the environment of the agent. Figure 18 shows the result of this simulation.

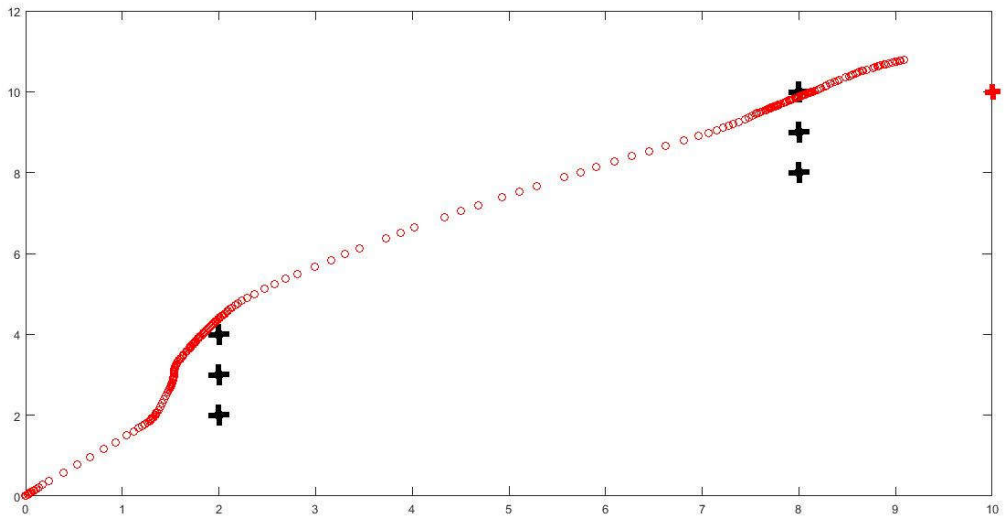

Figure 18: One agent and two walls

3.7 Classification of the problems associated with APF algorithm

The classical APF algorithm is a powerful tool in analysis and implementation of MAS. However, there are several problems associated with this algorithm that caused by APF mathematical structure. Problems such as local minima and target unreachability. Over last decade a number of solutions has been proposed by the researchers to resolve the problems associated with the APF algorithm. This section introduces the classification of well-known APF algorithm limitations and problems.

3.7.1 The problem of unreachable target

Goal unreachability is the most common problem of the APF algorithm. It happens when agents fall into local minima of the potential field on their way toward the target. This case can happen when an agent is on its trajectory to the target and it get very close to an obstacle or a cluster of obstacles. In such a case, the total repulsive force of the obstacles might be greater than

the attraction force of the target. Therefore agent(s) will trap in the neighborhood of the obstacle(s). Agent(s) will oscillate around the obstacle and will be not able to reach their goal [4].

Another case of unreachable goal can be resulted from the principle of superposition of the potential fields (or the superposition of the repulsive and attractive forces) around a trajectory that looks like a line. This is a case that the position of the obstacle is exactly between the trajectory lines that connects agent to the target. In this case, according to the properties of the APF algorithm there must be some points in the plain in which the repulsive force of the obstacle and attractive force of the target neutralized each other. Consequently, the total force at that point will be zero and if the agent enters that point will atop and will trapped there forever [4]. There is another case that observed when performing MATLAB simulation in the previous section. Agent(s) might pass all the obstacles and get close to the target. But the superposition of the repulsive forces of the obstacles prevent agent from getting too close to the target. In this case either agent will oscillate around close neighborhood of the target or eventually depart for the neighborhood and goes away from target.

3.7.2 The narrow channel problem

The narrow channel problem only happens when there are large number of obstacles in the environment. In this case agents try to find a path that looks like a channel to pass the obstacles. When agents passing the channel, it might be locations that the superposition of the attractive and repulsive forces is in a way that will stop agent [4]. Agents might oscillate inside the channel or just stop forever in the point of local minima. Narrow channel problem is the main issue in path planning in three-dimensional space. For example, navigation of the UAV inside cities using APF algorithm is a good case for narrow channel problem [9].

3.7.3 The Jitter problem

Jitter problem becomes a real concern when agents are navigating inside a busy and dynamic environment. Jitter primarily happens because of sudden change in the potential fields associated with the obstacles. It can also occur when agents moving around multiple potential minima. The jitter problem is less aggressive than the first two cases mentioned before. The agent (robot) may finally reach their goal even though that jitter problem occurred but the agent capability of path planning reduced significantly.

As mentioned previously a large number of researches focused on finding suitable solutions to the APF algorithm problems. Most of these optimizations of the APF algorithm are about defining new class of potential fields that prevent the above problems to occur. In the next section, a handful of these optimization techniques and solutions to the APF algorithm problems will be reviewed. Also, a MATLAB simulation of the new technique will be presented to verify the results.

3.8 State space representation of the MAS using APF algorithm

In previous section a model of nonholonomic robot that represents agents in the MAS has been introduced. In this section another approach that utilizes state space representation of the dynamic systems will be presented. It should be noted that the system is modeled using same APF methodology but to solve the system state space representation of the model will be utilized. In the new approach, the dynamics of each individual agents is given by the following differential equations.

$$\ddot{x} = F_x = F_{targetx} + \sum_i F_{obsix} \tag{24}$$

$$\ddot{y} = F_y = F_{targety} + \sum_i F_{obsiy} \tag{25}$$

In this section we take slightly different approach to solve the system of ordinary differential equations given in the equations (24) and (25). To solve this system, we will use state space representation of the system. These representations of the system have the advantage of presenting space variables in the matrix format. Matrix format is an ideal for simulations using MATLAB.

System of differential equations for the control of MAS presented in this section will be presented in state space form by changing the variables. Therefore, new set of variables defined to perform this change of variable as followed.

$$x = x_1 \tag{26}$$

$$\dot{x} = x_2 \tag{27}$$

$$y = y_1 \tag{28}$$

$$\dot{y} = y_2 \tag{29}$$

With this change of variable, we can rewrite the original system of ordinary differential equations directly into the following matrix format:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} F_x \tag{30}$$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} F_y \tag{31}$$

It is worth mentioning that force components in the equations (30) and (31) are the components of the total force obtained by the superposition of the forces associated with the attractive and repulsive potential fields in the environment of the agents. However, this approach still suffers from the general problems associated with the APF algorithm. Therefore, it is necessary to introduce new set of potential fields that will resolve problems such as local minima and narrow channels problem. One pf the problems that occur when agents get very close to the target or passing between obstacles is the oscillation of the agent. To prevent oscillation a set of friction forces introduced to the MAS. This friction force act very similar to the real physical world case and it can prevent agents from oscillation around the target or other objects. The friction forces are defined bellow.

This new design in addition to simplicity can resolve some of the problems that we observed previously. State-space models are suitable for stability analysis. We can perform the stability analysis of the designed system using MATLAB. Simulation result of the new system design using modified algorithms given in the Figure 19 in this simulation there are three targets along with each agent. There are two obstacles presented in the environment. Each agent distinguished with the color codes.

In addition of obstacle avoidance feature and resolving unreachable target problem this algorithm implements formation control as well. In this example the goal is to form a triangle in any given coordinate in the two-dimensional plane.

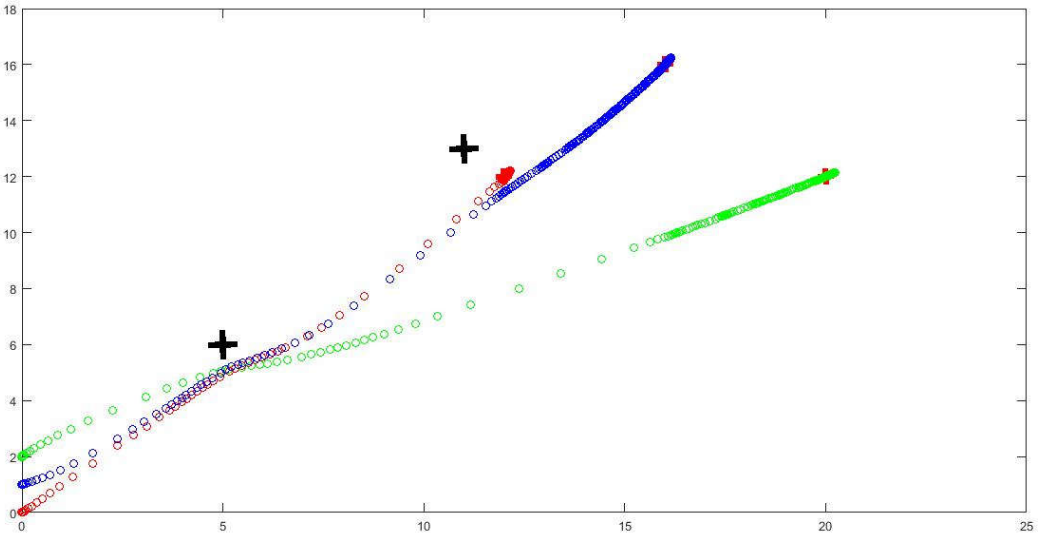

Figure 19 Modified APF Algorithm MATLAB Simulation

SECTION 4: LEADER FOLLOWER ALGORITHM

4.1 Introduction

One of the most important collective behaviors in formation control of multi agent systems (MAS) is leader follower approach (LFA). LFA has the benefit of both centralized and decentralized systems. LFA system can be either leader-follower which has a simple structure, or it can be follower-follower which is slightly more complicated.

One of the most important collective behaviors in nature is leader follower. There are many natural phenomena that leader follower formation control plays a crucial role in its existence and survival of that system. Ants society is a well-known example of swarming behavior. Ants societies utilize leader follower formation control in the relation between the queen ant and the regular worker ants. The queen ant provides commands and directions to the other ants in their society. The worker ants have a very simple and primitive thinking ability. The same pattern that rules ants' world exists in the bee's societies. The queen bee controls their colony and make all the workers in their colony active [11].

Another natural phenomenon regarding leader follower formation control comes from the study of evolutionary biology. The leader follower behavior can describe the genotype mapping in micro organisms' colonies [12]. One of the most exciting but yet surprising application of the leader follower behavior exists in the oscillator circuits used in computer architectures. In this case the leader is the local clock that sets frequency or global phase [12].

The leader-follower formation control has two main elements which are the leader and the follower agents. The leader is defined as an agent that its dynamic is independent from all the other agent's dynamic in the system [12]. There are several classifications for the leaders. The first class of the leaders called, power leaders. This type of leaders observed in natural phenomena that demonstrate aggregate motion. In these systems, leader dictated to other agents "where-to-go". Follower agents have no global information about the group destination or the orientation of agents in the system. followers only aware of the information about the adjacent agents. In this scenario power leader synchronize the all the information in the dynamic network. The second type of the leaders called Knowledge leader. Knowledge leaders has fixed dynamics. The follower agents extract information from the leader through the local adjustment apparatus [12]. In these systems, leader dictated to other agents "how-to-go"

4.2 LFA Model Implementation

In this section we are going to implement a leader-follower scenario using APF algorithm. This is one of the powerful advantages of the APF algorithm that can be used in the LFA situations. We use the previously developed model and we chose one agent as a leader. We assign an attractive potential field to the leader agent. Then, an attractive force has been defined between leader and other agents. These attractive functions will define the relations between leader and the other followers in the MAS. In addition, we define a safe neighborhood around the leader to prevent other agents to get too close to the leader. Figure 20 is a MATLAB simulation of leader-follower approach algorithm. The red agent is a leader and the green agent is the follower.

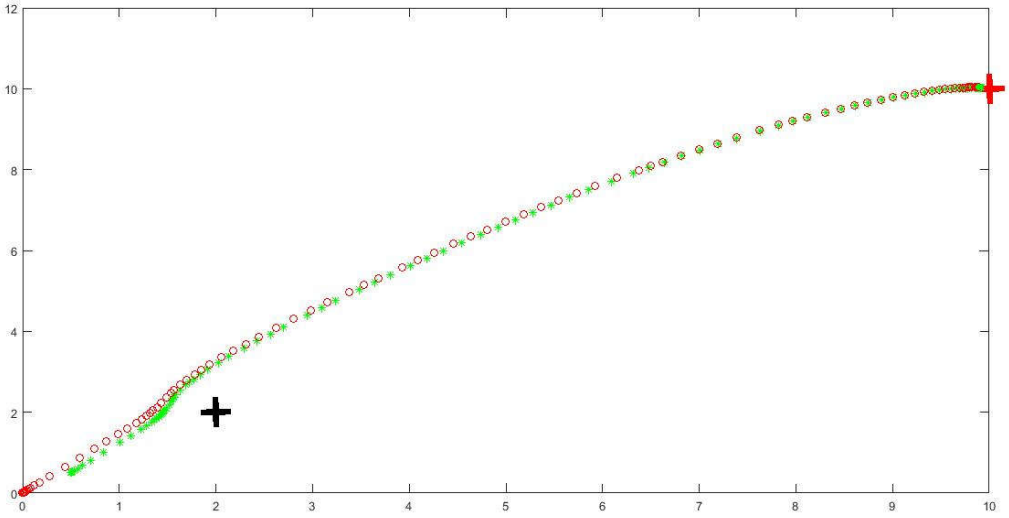

Figure 20: Leader-Follower Approach MATLAB Simulation

SECTION 5: MAS HARDWARE SURVEY

5.1 Overview on the Hardware Platform for MAS systems development

MAS control system design has three main phases. The first phase starts with mathematical structure design that addressed the swarm behaviors. The general approach is to consider each swarm behavior as known problem. All the characteristic of swarming problem has to be determined. This problem formulated as a mathematical problem. Then an algorithm to solve the swarm behavior problem has to be presented. The next step is to verify the ability of the presented algorithm to solve the real problems associated with that behavior. The first two phases are just a theoretical process. In order to implement the developed algorithm, it is necessary to design a hardware platform that can be utilized to test the algorithm in the real field.

The implementation phase of the swarm research requires a large number of agents to be used. Some aspect of swarming such as network traffic can only be tested if large number of agents presented in the environment. In the case of robotic system, the MAS design becomes very expensive. In addition, there is another problem of charging robot when large number of robots exist in the MAS. Therefore, it is crucial to design a MAS that its agents are very less expensive. Also, it is important to have robotic platform that store energy for longer time, and they are easy to power charge.

Over last decade there been a lot of efforts to address these issues. Some successful platforms developed by different research groups around the world. In the review of the hardware platforms, it is important to consider the processing power and communication utilities onboard of each agent. Price of each unit is another important factor that must be considered when implementing swarm hardware platforms. Some of these platforms are open source which can be a very useful for research groups in the universities and for the hobbyist as well. This section will introduce some successful and stable platforms particularly designed for the MAS that can be used for the swarm algorithm development and testing.

5.2 Kilobot Robot

Kilobot is a low-cost platform developed by Harvard university for research and development purposes in the field of swarm robotics. The goal of the Kilobot project is to provide a platform for large scale implementation of MAS. The Kilobot agents are very small about 3.3 cm length. Each individual Kilobot agent (Figure 21) has a 3.4 Volts 160 mAh lithium-ion battery which provides autonomy for six hours [20]. The onboard processor is a simple Atmega328 microcontroller. Kilobot robot agents utilize visible light sensor for navigation and detection of the objects. The communication platform consists of an infrared LED transmitter and photodiode receiver [20]. The Kilobot MAS system has a centralized controller that can be programmed directly from a personal computer. Each agent in this platform costs around 15 dollars. Therefore, Kilobot MAS is a best platform for development of large-scale MAS with more than 1000 agents [20].

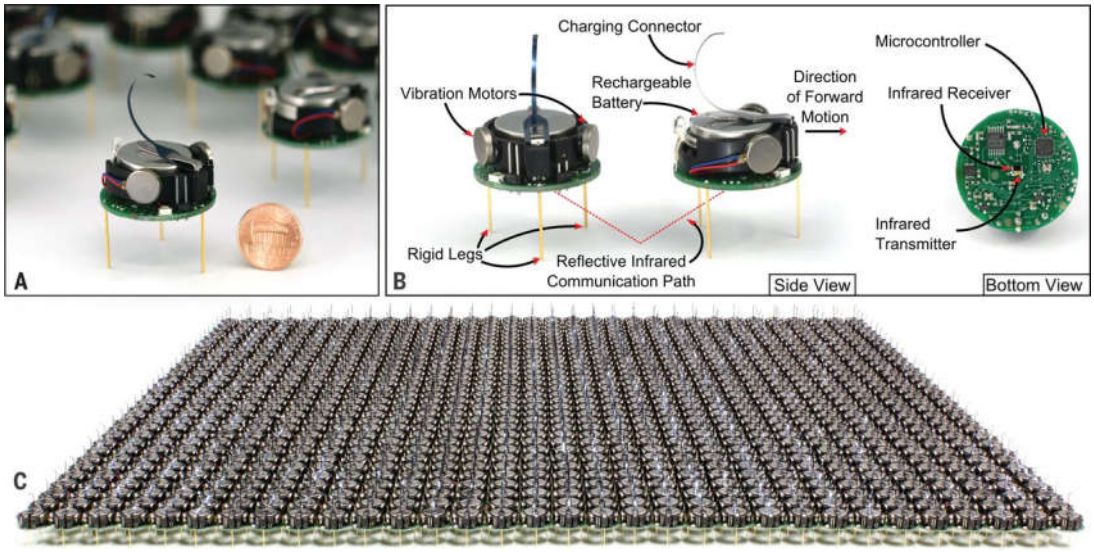

Figure 21: Kilobot Robot [25]

5.3 E-puck Robot

E-puck platform is a miniature robot developed by EPFL [21]. It is one of the oldest commercially available swarm platforms in the world. It is still under development. The latest stable version introduced in 2018. This robot platform utilizes STM32F4 microcontroller. The communication platform utilizes ZigBee. E-puck uses variety of sensors such as 3D accelerometer, IR sensors and camera. (Figure 22)

The locomotion in e-puck is based on wheeled actuators. Its nominal speed rated at 13 cm/s. the e-puck size is about 7.5 cm and its battery pack provide 1-10 hours. of autonomy based on the type of operation and environment that robot operates will vary. E-puck is an advanced robotic platform with a lot of capabilities, but it is relatively expensive for the development of large-scale MAS.

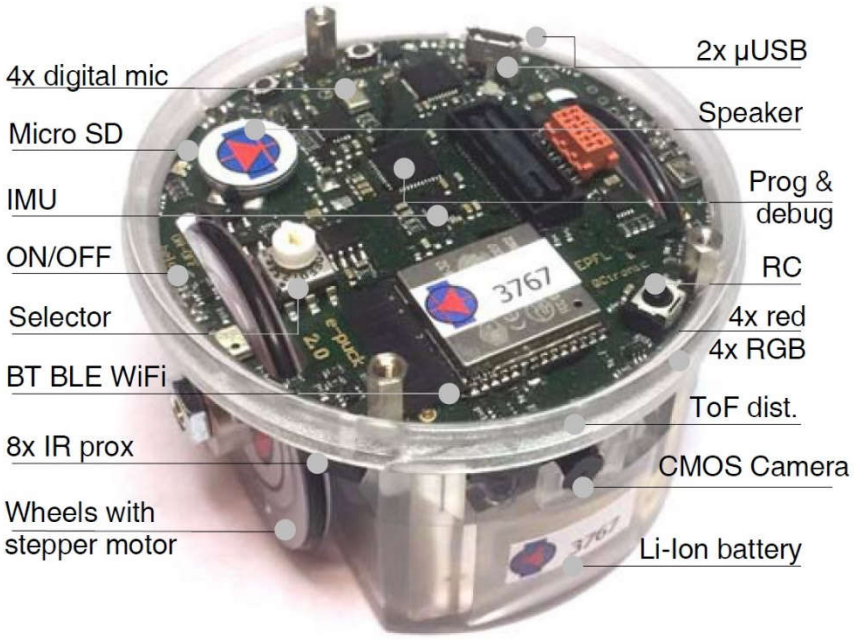

Figure 22: E-puck robot [26]

5.4 AMiR Robot

AMiR is a low-cost, open source swarm development platform developed in Putra University, Malesia. This platform utilizes AVR microcontroller ATMEGA 168. It has variety of sensors including IR-sensors. The locomotion in AMiR is based on wheeled actuators and it has a very smooth controller on board. Its nominal speed rated at 10 cm/s. The AMiR length is about 6.5 cm and its battery pack provide 2 hours of autonomy based on the type of operation and environment that robot operates will vary [27]. AMiR robot uses a 400 mAh Li-Po battery with nominal voltage of 3.7 volts. This battery pack is more reliable and has higher energy density. AMiR is a relatively low-cost platform that can be built with the total cost about 100 dollars per unit. (Figure 23)

Figure 23: AMiR Robot [27]

5.5 R-One Robot

R-One is a relatively low-cost open source platform developed for research and development purposes in the field of swarm robotics. The goal of the R-One project is to provide a platform for large scale implementation of MAS. The onboard computer utilizes ARM Cortex-M3 processor. R-One (Figure 24) robot equipped with 3D accelerometer, wheel encoders and IR-sensors. This robot uses radio frequency and IR for communication. The locomotion in R-One is based on wheeled actuators. Its nominal speed rated at 30 cm/s. The R-One length is about 10 cm and its battery pack provide 6 hours of autonomy based on the type of operation and environment that robot operates will vary [28]. R-One development platform costs about 250 dollars per unit.

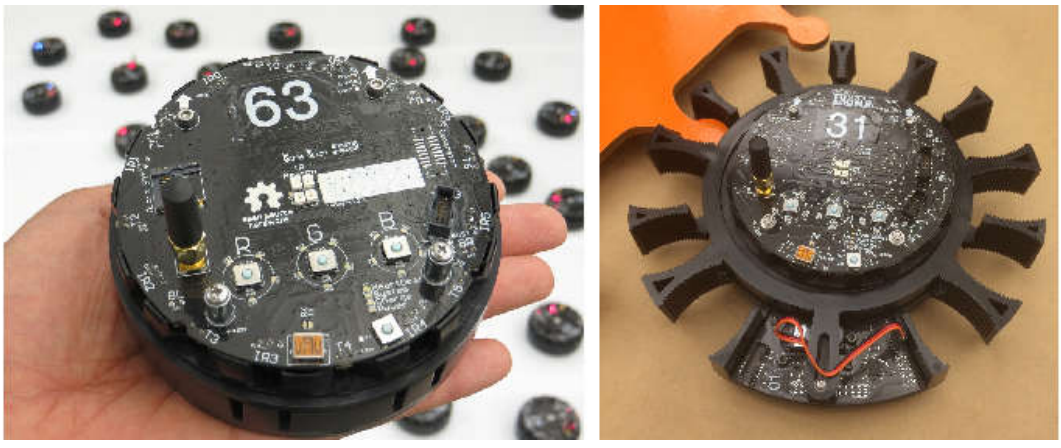

Figure 24: R-One Robot

5.6 Jasmine Robot

Jasmine is one of the oldest open source development platforms for the MAS. Jasmine is a relatively low-cost open source platform developed for research and development purposes in the field of swarm robotics. The goal of the Jasmine project is to provide a platform for large scale implementation of MAS. The onboard computer utilizes two ATmega microcontrollers. Jasmine robot equipped with eight IR-sensors (Figure 25). The locomotion in Jasmine is based on wheeled actuators. Its nominal speed rated at 10 cm/s. The Jasmine length is about 3 cm and its battery pack provide 1-2 hours of autonomy based on the type of operation and environment that robot operates will vary [29]. Jasmine development platform costs about 100 dollars per unit.

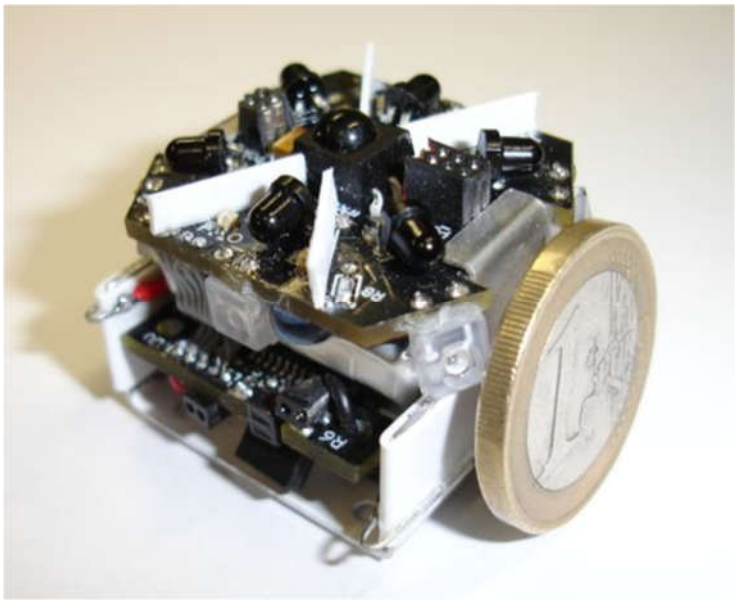

Figure 25: Jasmine Robot

5.7 Mona Robot

Mona (Figure 26) is a low-cost open source platform developed in Manchester university for research and development purposes in the field of MAS and robotics. The goal of the Mona project is to provide a platform for large scale implementation of MAS. This platform has fully integrated with Robot Operating System (ROS). This is a main advantage of this platform beside its low cost. The Mona onboard computational unit is a simple Atmega328 microcontroller that utilizes AVR RISC architecture. Mona robot agents use short range IR-sensors for navigation and motion planning. The locomotion in Mona is based on wheeled actuators. Its nominal speed rated at 12 mm/s [29].

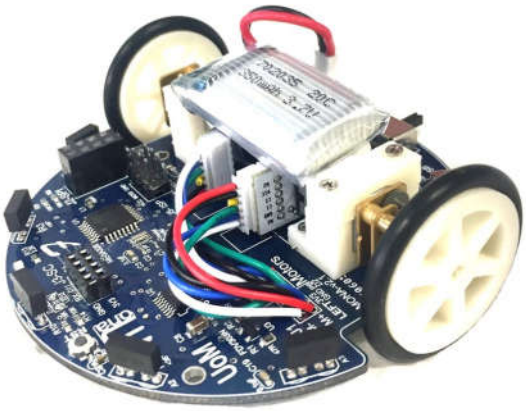

Figure 26: Mona Robot

5.8 WeeMiK Robot

WeeMiK (Figure 27) is a relatively low-cost omnidirectional robotic platform for the implementation of the MAS. This robot is an excellent platform for large-scale deployment of swarm intelligent systems. This robot utilizes AVR Atmega 2560 microcontroller. This unit contains a large diversity of on-board sensors. WeeMike uses eight IR Proximity sensors. It uses 3D accelerometer and 3D gyroscope. WeeMike supports wireless communication using ultra low power transceivers. The large number of available resources on this robotic platform make a good candidate for the development of MAS robots that required complex tasks. For educational purposes this robot has its own ready to use swarming test software [30].

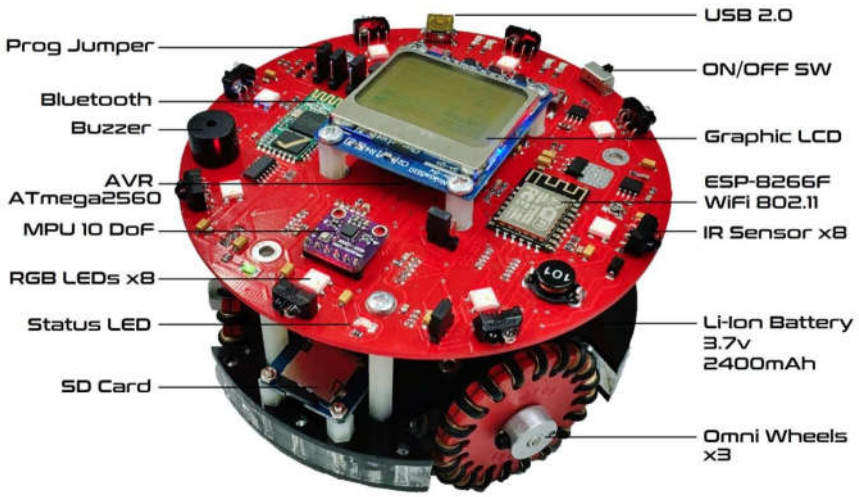

Figure 27: WeeMiK Robot

5.9 Attabot

Attabot (Figure 28) is a low-cost modular robotic platform for the implementation of the MAS and any other robotic research. This robot is an excellent platform for large-scale deployment of swarm intelligent systems. This robotic platform inspired from the Brazilian ants. This robot utilizes Arduino Mega 2560 microcontroller. This unit contains a large diversity of on-board sensors. Attabot uses IR and Ultrasonic sensors. The locomotion in Attabot is based on wheeled actuators with encoders. Its nominal speed rated at 40 rpm/min. The Attabot battery pack provide one hour of autonomy based on the type of operation and environment that robot operates will vary [31].

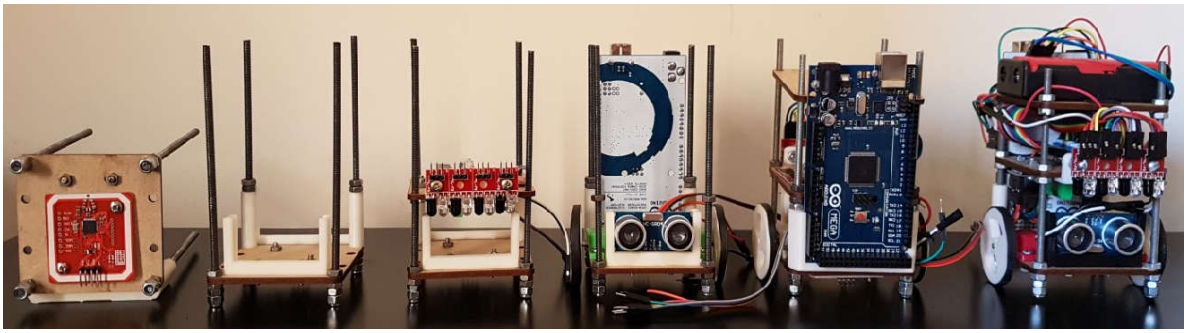

Figure 28: Attabot Robot

CONCLUSION

This work started with the study of the most common algorithms developed over the last decades regarding MAS. These algorithms have been briefly introduced and they are eventually compared in terms of applications in the research development or commercial use in industry. After the study of the MAS algorithm, the Artificial Potential field algorithm selected as an algorithm of choice. This algorithm widely used in academia and it is still under development and improvement by several research groups worldwide. The APF algorithm has been well implemented in several commercial robotic projects such as rescue missions or fire fighter drones.

In this study, the mathematical structure of the APF algorithm investigated. Several simulation scenarios based on the variation among the agents and obstacles has been simulated and results have been captured. While performing the simulations some problems such as unreachable target and narrow channels has been observed. The APF algorithm used in the first part of this study modified to address problems previously observed in simulations. The new approach to APF algorithm and its implementation took place in the new design of the MAS in this study. For simplicity the space-state representation of the MAS system utilized to implement new modified algorithm. The new modified algorithm has been implemented in the MATLAB code using this new approach. To verify the result a new set of simulation with more complicated set-up and variables designed to test if all the previously observed problem has been addressed and fixed properly. Finally, a survey on the hardware platform used in the implementation of the MAS introduced.

REFERENCES

[1] S. Fuady, A. R. Ibrahim, and R. T. Bambang, “Distributed formation control of multi-robot system with obstacle avoidance,” *2013 International Conference on Robotics, Biomimetics, Intelligent Computational Systems*, 2013.

[2] F. Rossi, S. Bandyopadhyay, M. Wolf, and M. Pavone, “Review of Multi-Agent Algorithms for Collective Behavior: a Structural Taxonomy,” *IFAC-Papers On-Line*, vol. 51, no. 12, pp. 112–117, 2018.

[3] J. Rosenbaum, “Forecasting by swarm,” *Forecasting by swarm | Futures Magazine*, 15-Jan-2018. [Online]. Available: http://www.futuresmag.com/2018/01/15/forecasting-swarm. [Accessed: 03-Apr-2019].

[4] J. Sun, J. Tang, and S. Lao, “Collision Avoidance for Cooperative UAVs with Optimized Artificial Potential Field Algorithm,” *IEEE Access*, vol. 5, pp. 18382–18390, 2017.

[5] A. M. Hassan, C. M. Elias, O. M. Shehata, and E. I. Morgan, “A Global Integrated Artificial Potential Field / Virtual Obstacles Path Planning Algorithm for Multi-Robot System Applications,” *Semantic Scholar*, 01-Sep-2017. International Research Journal of Engineering and Technology (IRJET), vol. 4, Issue 9, pp1198-1204.

[6] M. I. Riberio, “Obstacle Avoidance - Institute for Systems and Robotics,” 2005. [Online]. Available: http://users.isr.ist.utl.pt/~mir/pub/ObstacleAvoidance.pdf. [Accessed: 05-Apr-2019]. Isabel Ribeiro is Full Professor and Distinguished Professor at the Electrical and Computer Engineering Department of Instituto Superior Técnico (IST), the School of Engineering of the Universidade de Lisboa.

[7] S. Mastellone, D. M. Stipanovic, and M. W. Spong, “Remote Formation Control and Collision Avoidance for Multi-Agent Nonholonomic Systems,” *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007.

[8] K. J. Åström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton: Princeton University Press, 2008.

[9] H. Choset, K. Lynch, and S. Hutchinson, *Principles of robot motion theory, algorithms, and implementation*. Massachusetts: The MIT Press, 2004.

[10] W. Kowalczyk and K. Kozlowski, “Leader-Follower Control and Collision Avoidance for the Formation of Differentially-Driven Mobile Robots,” *2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR)*, 2018.

[11] B. Ashokkumar and D. Frazer, “implementation of load sharing using swarm robotics ,” *International Research Journal of Engineering and Technology* , vol. 3, no. 3, Mar. 2016.

[12] Wei Wang and Jean Slotine, “A Theoretical study of different leader follower leader roles in networks ,” *Nonlinear Systems Laboratory, MIT*.

[13] Principles of robot motion: theory, algorithms, and implementation Howie Choset-Kevin Lynch-Seth Hutchinson-George Kantor-Wolfram Burgard-Lydia Kavraki-Sebastian Thrun - MIT Press – 2005

[14] A. A. Bandala, E. P. Dadios, R. R. P. Vicerra, and L. A. G. Lim, “Swarming Algorithm for Unmanned Aerial Vehicle (UAV) Quadrotors – Swarm Behavior for Aggregation, Foraging, Formation, and Tracking –,” Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 18, no. 5, pp. 745–751, 2014.

[15] L. Bayinder and E. Sahin, “A Review of Studies in Swarm Robotics,” *Turk Journal of Electrical Engineerng*, vol. 15, 2007.

[16] F. Campaigns, “Swarm of 1,000 drones come together to create TIME magazine cover,” *https://www.famouscampaigns.com/*. [Online]. Available: https://www.famouscampaigns.com/2018/05/1000-drones-come-together-to-create-time-magazine-cover

[17] N. E. Shlyakhov, I. V. Vatamaniuk, and A. L. Ronzhin, “Survey of Methods and Algorithms of Robot Swarm Aggregation,” *Journal of Physics: Conference Series*, vol. 803, p. 012146, 2017.

[18] *ommunication assisted navigation in robotic swarms*. [Online]. Available: http://people.idsia.ch/~frederick/swarm_navigation/online_material.html.

[19] Z. Li and Z. Duan, *Cooperative control of multi-agent systems: a consensus region approach*. Boca Raton: CRC Press, Taylor & Francis Group, 2015.

[20] M. Rubenstein, C. Ahler, and R. Nagpal, “Kilobot: A low cost scalable robot system for collective behaviors,” *2012 IEEE International Conference on Robotics and Automation*, 2012.

[21] R. Olfati-Saber, J. A. Fax, and R. M. Murray, “Consensus and Cooperation in Networked Multi-Agent Systems,” *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[22] D. Kampally, R. Annae, and S. Kumar, “Analysis of Algorithms to Implement Swarm Bots for Effective Swarm Robotics,” *IJCSET*, vol. 1, no. 7.

[23] C. Moeslinger, T. Schmickl, and K. Crailsheim, “A Minimalist Flocking Algorithm for Swarm Robots,” *Advances in Artificial Life. Darwin Meets von Neumann Lecture Notes in Computer Science*, pp. 375–382, 2011.

[24] N. R. Hoff, A. Sagoff, R. J. Wood, and R. Nagpal, “Two foraging algorithms for robot swarms using only local communication,” *2010 IEEE International Conference on Robotics and Biomimetics*, 2010.

[25] M. Rubenstein, A. Cornejo, and R. Nagpal, “Programmable self-assembly in a thousand-robot swarm,” *Science*, 15-Aug-2014. [Online]. Available: https://science.sciencemag.org/content/345/6198/795/tab-figures-data.

[26] G. Ctronic , “e-puck development platform,” *e-puck education robot*. [Online]. Available: http://www.e-puck.org/.

[27] F. Arvin, “Development of a Miniature Robot for Swarm Robotic Application,” *International Journal of Computer and Electrical Engineering*, vol. 1, no. 4, 2009.

[28] J. Mclurkin, A. Mcmullen, N. Robbins, G. Habibi, A. Becker, A. Chou, H. Li, M. John, N. Okeke, J. Rykowski, S. Kim, W. Xie, T. Vaughn, Y. Zhou, J. Shen, N. Chen, Q. Kaseman, L. Langford, J. Hunt, A. Boone, and K. Koch, “A robot system design for low-cost multi-robot manipulation,” *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[29] F. Arvin, J. Espinosa, B. Bird, A. West, S. Watson, and B. Lennox, “Mona: an Affordable Open-Source Mobile Robot for Education and Research,” *Journal of Intelligent & Robotic Systems*, vol. 94, no. 3-4, pp. 761–775, 2018.

[30] M. Karimi, A. Ahmadi, P. Kavandi, and S. S. Ghidary, “WeeMiK: A low-cost omnidirectional swarm platform for outreach, research and education,” *2016 4th International Conference on Robotics and Mechatronics (ICROM)*, 2016.

[31] M. V. Ferreira, J. P. Fonseca, and J. J. Tavares, “Attabot: Open Platform Inspired on Brazilian Ants for Swarm Robots,” *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, 2018.

APPENDIX: MATLAB CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%This function defines and calculates robot dynamics        %%
%%This program uses Artificial Potential Fields algorithms   %%
%%to implement formation control of multi agent systems (MAS)%%
%%California State University Northridge - Master Project    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function differential=RobotDynamics(t,StateSpaceMatrix)
%potential field gain definitions
Kg1=90; %20   %Target/goal potential field gain
Kg2=90;
Kg3=90;
Ko=30; %30   Obstacles potential field gain. In this code same gain used for
all obstacles
Kij=20; %20   robot i and j repulsive potential field gain.
KLi=100;%leader and robot i  potential field gain.
Kf=10;
Rd=0.5;
%layout of the objects in the 2D plain
target1location=[12;12]; %[x y] coordinate of the target/goal
target2location=[16,16];
target3location=[20,12];
Obstacle1=[5;6]; %3,3 [x y] coordinate of the first obstacle
Obstacle2=[11;13]; % [x y] coordinate of the second obstacle

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Robots Dynamics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%#################################### Leader Dynamics
####################################################
% The relations between leader in the group and the target
R_L_g=sqrt((target1location(1)-StateSpaceMatrix(1))^2+(target1location(2)-
StateSpaceMatrix(3))^2); % distance from leader to the goal
F_L_gx=Kg1*(target1location(1)-StateSpaceMatrix(1))/max([R_L_g 0.01]); % x
component of the force between leader and the goal
F_L_gy=Kg1*(target1location(2)-StateSpaceMatrix(3))/max([R_L_g 0.01]); % y
component of the force between leader and the goal

R_L_obs1=sqrt((Obstacle1(1)-StateSpaceMatrix(1))^2+(Obstacle1(2)-
StateSpaceMatrix(3))^2); % distance from leader to the obstacle 1
F_L_obs1x=-Ko*(Obstacle1(1)-StateSpaceMatrix(1))/R_L_obs1^3; % x component of
the force between leader and the obstacle 1
F_L_obs1y=-Ko*(Obstacle1(2)-StateSpaceMatrix(3))/R_L_obs1^3; % y component of
the force between leader and the obstacle 1

R_L_obs2=sqrt((Obstacle2(1)-StateSpaceMatrix(1))^2+(Obstacle2(2)-
StateSpaceMatrix(3))^2); % distance from leader to the obstacle 2
F_L_obs2x=-Ko*(Obstacle2(1)-StateSpaceMatrix(1))/R_L_obs2^3; % x component of
the force between leader and the obstacle 2
```

```
Fo2y=-Ko*(Obstacle2(2)-StateSpaceMatrix(3))/R_L_obs2^3; % y component of the
force between leader and the obstacle 2

%damping force on leader to prevent oscillations
FrLx=-Kf*StateSpaceMatrix(2);
FrLy=-Kf*StateSpaceMatrix(4);

% Total force x,y components
F_leaderx=F_L_gx+F_L_obs1x+F_L_obs2x+FrLx;
F_leadery=F_L_gy+F_L_obs1y+Fo2y+FrLy;

%Leader dynamics
xdotL=[0 1;0 0]*[StateSpaceMatrix(1);StateSpaceMatrix(2)]+[0;1]*F_leaderx;
ydotL=[0 1;0 0]*[StateSpaceMatrix(3);StateSpaceMatrix(4)]+[0;1]*F_leadery;

%%##################################### Robot2 Dynamics
#####################################################
% The relations between robot2 in the group and the target
R_2_g=sqrt((target2location(1)-StateSpaceMatrix(5))^2+(target2location(2)-
StateSpaceMatrix(7))^2); % distance from robot2 to the goal
F_2_gx=Kg2*(target2location(1)-StateSpaceMatrix(5))/max([R_2_g 0.01]); % x
component of the force between robot2 and the goal
F_2_gy=Kg2*(target2location(2)-StateSpaceMatrix(7))/max([R_2_g 0.01]); % y
component of the force between robot2 and the goal

%Robot 2 Obstacle 1 = follower 1 obstacle 1
R_robot2_obs1=sqrt((Obstacle1(1)-StateSpaceMatrix(5))^2+(Obstacle1(2)-
StateSpaceMatrix(7))^2);
F_robot2_obs1x=-Ko*(Obstacle1(1)-StateSpaceMatrix(5))/R_robot2_obs1^3;
F_robot2_obs1y=-Ko*(Obstacle1(2)-StateSpaceMatrix(7))/R_robot2_obs1^3;

%Robot 2 Obstacle 2 = follower 1 obstacle 2
R_robot2_obs2=sqrt((Obstacle2(1)-StateSpaceMatrix(5))^2+(Obstacle2(2)-
StateSpaceMatrix(7))^2);
F_robot2_obs2x=-Ko*(Obstacle2(1)-StateSpaceMatrix(5))/R_robot2_obs2^3;
F_robot2_obs2y=-Ko*(Obstacle2(2)-StateSpaceMatrix(7))/R_robot2_obs2^3;

%Robot 2 Leader
% R_robot2_L=sqrt((StateSpaceMatrix(1)-
StateSpaceMatrix(5))^2+(StateSpaceMatrix(3)-StateSpaceMatrix(7))^2);
% F_robot2_Lx=KLi*(R_robot2_L-Rd)*(StateSpaceMatrix(1)-
StateSpaceMatrix(5))/max([R_robot2_L 0.01]);
% F_robot2_Ly=KLi*(R_robot2_L-Rd)*(StateSpaceMatrix(3)-
StateSpaceMatrix(7))/max([R_robot2_L 0.01]);
F_robot2_Lx=0;
F_robot2_Ly=0;
%Robot 2 robot 3
R_robot2_robot3=sqrt((StateSpaceMatrix(9)-
StateSpaceMatrix(5))^2+(StateSpaceMatrix(11)-StateSpaceMatrix(7))^2);
F_robot2_robot3x=-2*Kij*(StateSpaceMatrix(9)-
StateSpaceMatrix(5))/max([R_robot2_robot3^4 0.01]);
F_robot2_robot3y=-2*Kij*(StateSpaceMatrix(11)-
StateSpaceMatrix(7))/max([R_robot2_robot3^4 0.01]);

%damping force on robot 2 to prevent oscillations
```

```
F_d_robot2x=-Kf*StateSpaceMatrix(6);
F_d_robot2y=-Kf*StateSpaceMatrix(8);

% Total force on robot 2 x,y components
F_robot2x=F_2_gx
+F_robot2_obs1x+F_robot2_obs2x+F_robot2_Lx+F_robot2_robot3x+F_d_robot2x;
F_robot2y=F_2_gy+F_robot2_obs1y+F_robot2_obs2y+F_robot2_Ly+F_robot2_robot3y+F
_d_robot2y;

%robot2 dynamics - state space equations
xdot1=[0 1;0 0]*[StateSpaceMatrix(5); StateSpaceMatrix(6)]+[0; 1]*F_robot2x;
ydot1=[0 1;0 0]*[StateSpaceMatrix(7); StateSpaceMatrix(8)]+[0; 1]*F_robot2y;

%%##################################### Robot3 Dynamics
#####################################################
R_3_g=sqrt((target3location(1)-StateSpaceMatrix(9))^2+(target3location(2)-
StateSpaceMatrix(11))^2); % distance from robot2 to the goal
F_3_gx=Kg3*(target3location(1)-StateSpaceMatrix(9))/max([R_3_g 0.01]); % x
component of the force between robot2 and the goal
F_3_gy=Kg3*(target3location(2)-StateSpaceMatrix(11))/max([R_3_g 0.01]); % y
component of the force between robot2 and the goal

%Robot 3 Obstacle 1 =follower 2 obs 1
R_robot3_obs1=sqrt((Obstacle1(1)-StateSpaceMatrix(9))^2+(Obstacle1(2)-
StateSpaceMatrix(11))^2);
F_robot3_obs1x=-Ko*(Obstacle1(1)-StateSpaceMatrix(9))/R_robot3_obs1^3;
F_robot3_obs1y=-Ko*(Obstacle1(2)-StateSpaceMatrix(11))/R_robot3_obs1^3;

%Robot 3 Obstacle 2 follower 2 obs 2
R_robot3_obs2=sqrt((Obstacle2(1)-StateSpaceMatrix(9))^2+(Obstacle2(2)-
StateSpaceMatrix(11))^2);
F_robot3_obs2x=-Ko*(Obstacle2(1)-StateSpaceMatrix(9))/R_robot3_obs2^3;
F_robot3_obs2y=-Ko*(Obstacle2(2)-StateSpaceMatrix(11))/R_robot3_obs2^3;

%Robot 3 Leader
% R_robot3_L=sqrt((StateSpaceMatrix(1)-
StateSpaceMatrix(9))^2+(StateSpaceMatrix(3)-StateSpaceMatrix(11))^2);
% F_robot3_Lx=KLi*(R_robot3_L-Rd)*(StateSpaceMatrix(1)-
StateSpaceMatrix(9))/max([R_robot3_L 0.01]);
% F_robot3_Ly=KLi*(R_robot3_L-Rd)*(StateSpaceMatrix(3)-
StateSpaceMatrix(11))/max([R_robot3_L 0.01]);
F_robot3_Lx=0;
F_robot3_Ly=0;

%Robot 3 robot 2
R_robot3_robot2=sqrt((StateSpaceMatrix(5)-
StateSpaceMatrix(9))^2+(StateSpaceMatrix(7)-StateSpaceMatrix(11))^2);
F_robot3_robot2x=-2*Kij*(StateSpaceMatrix(5)-
StateSpaceMatrix(9))/max([R_robot3_robot2^4 0.01]);
F_robot3_robot2y=-2*Kij*(StateSpaceMatrix(7)-
StateSpaceMatrix(11))/max([R_robot3_robot2^4 0.01]);

%damping force on robot 3 to prevent oscillations
F_d_robot3x=-Kf*StateSpaceMatrix(10);
F_d_robot3y=-Kf*StateSpaceMatrix(12);
```

```
% Total force on robot 3 x,y components
F_robot3x=F_3_gx
+F_robot3_obs1x+F_robot3_obs2x+F_robot3_Lx+F_robot3_robot2x+F_d_robot3x;
F_robot3y=F_3_gy
+F_robot3_obs1y+F_robot3_obs2y+F_robot3_Ly+F_robot3_robot2y+F_d_robot3y;

%robot3 dynamics - state space equations
xdot2=[0 1;0 0]*[StateSpaceMatrix(9); StateSpaceMatrix(10)]+[0; 1]*F_robot3x;
ydot2=[0 1;0 0]*[StateSpaceMatrix(11); StateSpaceMatrix(12)]+[0;
1]*F_robot3y;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
differential=[xdotL(1);xdotL(2);ydotL(1);ydotL(2);xdot1(1);xdot1(2);ydot1(1);
ydot1(2);xdot2(1);xdot2(2);ydot2(1);ydot2(2)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if StateSpaceMatrix(2)==0 && StateSpaceMatrix(4)==0
    w=0.001*randn(2,1);
    differential=differential+[0;w(1);0;w(2);0;0;0;0;0;0;0;0];
end

if R_robot3_robot2==0
   w=0.001*randn(4,1);
   differential=differential+[0;0;0;0;0;w(1);0;w(2);0;w(3);0;w(4)];
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




function [t,StateSpaceMatrix]=OCA_ROBOTs()
clear
clc
time=[0 400];
StateSpaceMatrix=[0;0;0;0;0;0;1;0;0;0;2;0];
%potential field gain definitions
Kg1=90; %Target/goal potential field gain
Kg2=90;
Kg3=90;
Ko=30; %Obstacles potential field gain. In this code same gain used for all
obstacles
Kij=20; %robot i and j repulsive potential field gain.
KLi=190;%leader and robot i  potential field gain.
Kf=10;
Rd=0.5;
%target and obstacle locations in 2D plane
target1location=[12;12];
target2location=[16;16];
target3location=[20,12];
Obstacle1=[5;6]; %3,3
Obstacle2=[11;13];
```

```
%Solving Differential equation to determine location of each robot
[t,StateSpaceMatrix]=ode23(@RobotDynamics,time,StateSpaceMatrix);

%robot 1 coordiantes matrix
robot1_xposition=StateSpaceMatrix(:,1);
robot1_yposition=StateSpaceMatrix(:,3);
%robot 2 coordiantes matrix
robot2_xposition=StateSpaceMatrix(:,5);
robot2_yposition=StateSpaceMatrix(:,7);
%robot 3 coordiantes matrix
robot3_xposition=StateSpaceMatrix(:,9);
robot3_yposition=StateSpaceMatrix(:,11);

% plot(x(:,1),x(:,3),x(:,5),x(:,7),x(:,9),x(:,11),'g',...
%     Goal(1),Goal(2),Obs1(1),Obs1(2),'x',Obs2(1),Obs2(2),'x')
% axis([0 12 0 12])
plot (target1location(1),target1location(2),'r-s', 'LineWidth',8)
hold on;
plot (target2location(1),target2location(2),'r-s', 'LineWidth',8)
hold on;
plot (target3location(1),target3location(2),'r-s', 'LineWidth',8)
hold on;
plot (Obstacle1(1),Obstacle1(2),'k-s', 'LineWidth',18)
hold on;
plot (Obstacle2(1),Obstacle2(2),'k-s', 'LineWidth',18)
hold on;

for k=1:1390
    plot(robot1_xposition(k),robot1_yposition(k),'Marker','o','color','red');
    F(k)=getframe;

plot(robot2_xposition(k),robot2_yposition(k),'Marker','o','color','blue');
    F(k)=getframe;

plot(robot3_xposition(k),robot3_yposition(k),'Marker','o','color','green');
    F(k)=getframe;
end
figure;
imshow(F.cdata);

end


%% Simulation of the potential fields for MAS %%
close all
clear all

clc

xf=0:0.1:10;
yf=0:0.1:10;
[X,Y]=meshgrid(xf,yf);

Goal=[10,10];
obs1=[3;3];
```

```
obs2=[9;9];
KG=10;
Ko=20;
%Potential field for the goal:
rG=sqrt((Goal(1)-X).^2+ (Goal(2)-Y).^2);
VG=KG*rG;
%Potential field for the first obstacle
ro1=sqrt((obs1(1)-X).^2 +(obs1(2)-Y).^2)  ;
Vo1= Ko./ro1;
%Potential field for the second obstacle
ro2=sqrt((obs2(1)-X).^2+(obs2(2)-Y).^2);
Vo2=Ko./ro2;
Z=VG+Vo1+Vo2;
mesh(X,Y,Z)
axis([0 12 0 12 0 200])
xlabel('x')
ylabel('y')
zlabel('v')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```