Cosine Kuramoto Based Distribution of a Convoy with Limit-Cycle Obstacle Avoidance Through the Use of Simulated Agents

William Howerton

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

Master of Science in Aeronautics & Astronautics

University of Washington

2016

Committee:

Mehran Mesbai

Behcet Acikmese

Program Authorized to Offer Degree:
Department of Aeronautics and Astronautics

University of Washington

Abstract

Cosine Kuramoto Based Distribution of a Convoy with Limit-Cycle Obstacle Avoidance Through the Use of Simulated Agents

William Howerton

Chair of the Supervisory Committee:
Professor Mehran Mesbai
William E. Boeing Department of Aeronautics and Astronautics

This thesis presents a method for the integration of complex network control algorithms with localized agent specific algorithms for maneuvering and obstacle avoidance. This method allows for successful implementation of group and agent specific behaviors. It has proven to be robust and will work for a variety of vehicle platforms. Initially, a review and implementation of two specific algorithms will be detailed. The first, a modified Kuramoto model was developed by Xu [1] which utilizes tools from graph theory to efficiently perform the task of distributing agents. The second algorithm developed by Kim [2] is an effective method for wheeled robots to avoid local obstacles using a limit-cycle navigation method. The results of implementing these methods on a test-bed of wheeled robots will be presented. Control issues related to outside disturbances not anticipated in the original theory are then discussed. A novel method of using simulated agents to separate the task of distributing agents from agent specific velocity and heading commands has been developed and implemented to address these issues. This new method can be used to combine various behaviors and is not limited to a specific control algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

AGENT: vehicle or node, simulated or real

JOHNNY ROBOT: the name given to the hardware platform that the algorithms are implemented on

NODE: another word for Agent or Vehicle

PID: abbreviation for proportional-integral-derivative in reference to a type of controller

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Mehran Mesbahi for his support and guidance. The author also wishes to thank the members of the Robotics, Aerospace and Information Networks (RAIN) Lab at the University of Washington for fostering an environment of learning and collaboration.

DEDICATION

to my family

Chapter 1

INTRODUCTION

Recently, the performance of sensors and micro-controllers has increased significantly, while the implementation cost has been reduced. This has resulted in the incorporation of sensors and controllers into many new applications with greater numbers. Now instead of a single expensive machine or robot it may be more effective to construct or deploy multiple, less expensive machines or robots. This new field is often referred to as "swarm robotics." Swarm robotics has presented new challenges, the major one being how to effectively and efficiently coordinate multiple machines to accomplish a desired task. Control theorists have traditionally developed control methods and algorithms for single agent systems or individual subsystems. However, a field of mathematics referred to as "graph theory" offers tools and methods to control large numbers of agents from either a central controller or as a distributed system.

The Georgia Robotics and Intelligent Systems Labratory (GRITS) is one group using graph theory to develop new methods of control for multi-agent systems. The group presented a paper entitled *Balanced Deployment of Multiple Robots Using a Modified Kuramoto Model* [1] at the 2013 American Control Conference. The authors utilized a cosine Kuramoto model to spatially balance the distribution of agents in a convoy. This algorithm was implemented in simulation and on a hardware test-bed of two wheel steering robots. One issue that was not addressed in [1] was how a disturbance in position or heading of an orbiting agent would affect the convoy. Furthermore, a method for obstacle avoidance was also absent.

Wheeled mobile robots are a favorite platform among control theorists as a means to test and demonstrate new algorithms and methods. As opposed to airborne robots, ground robots can generally operate for longer periods of time given the same amount of battery

power; can be easily confined to a plane; and require less effort to reset between tests. The wheeled mobile robot can also exhibit the dynamics of other vehicles such as the fixed wing aircraft by imposing a minimum forward velocity.

The Robotics, Aerospace and Information Networks (RAIN) Labratory at the University of Washington created the "Johnny" robots, a group of wheeled mobile robots whose positions and headings are sensed by an overhead camera system. The Johnny robots were used to implement the cosine Kuramoto model. Additionally, most ground robots or vehicles require the ability to maneuver around obstacles. A limit-cycle obstacle avoidance algorithm which had been developed for two wheeled mobile robots in *A Real-ime Limit-cycle Navigation Method for Fast Mobile Robots and its Application to Robot Soccer* [2] was chosen for inclusion in the system.

The obstacle avoidance model was combined with the cosine Kuramoto model to provide a more practical solution to the transport problem. In order to combine these two algorithms, a novel approach utilizing simulated agents as a means to decouple individual node behavior from the group was developed and successfully implemented. The use of simulated agents also provided increased resilience to outside disturbances when compared to the original method presented in [1].

This work will begin by reviewing the cosine Kuramoto model and the tools of graph theory that are necessary for its implementation. Then newly formulated bounds on the minimum agent velocity will be detailed. The cosine Kuramoto model will be followed with a presentation of the limit-cycle obstacle avoidance algorithm.

The limit-cycle obstacle avoidance presented in [2] proved to work well when implemented on a test bed of wheeled robots. The robots were able to avoid multiple moving obstacles, even when the positions of those obstacles would have forced other algorithms into a minima. After reviewing the specifics of the high level group coordination and the low level obstacle avoidance, the combination of the two methods utilizing simulated agents will be discussed.

The use of simulated agents allows for the combination of two or more sophisticated behaviors and also provides several additional benefits. The tuning of gains to maintain a

heading and velocity that are unique to the platform can be accomplished independent of higher level behaviors, such as evenly distributing around a radius. This node independent approach is robust and provides for dynamic stability. There were many sources for disturbances when operating the Johnny Robots in the lab. Typically, the location information of a robot from the overhead camera system would be temporarily lost or the wheels of the robot would slip due to foreign object debris. Using simulated agents prevented the robots from becoming unstable individually or collectively. Furthermore, this new method is not confined to the cosine Kuramoto model, other multi-agent methods can be substituted. The results of implementing the cosine Kuramoto model, the limit-cycle obstacle avoidance, and a combination of the two, in both simulation and on Johnny robots are detailed.

Chapter 2

COSINE KURAMOTO MODEL

The Kuramoto model is known for synchronizing oscillators. The phase difference between nodes can be increased by exchanging the sine term for a cosine term. This approach is refered to as the cosine Kuramoto model and was originally presented in [1] as a method to evenly distribute agents around a central node as they orbited on a radius. For completeness this chapter will begin by reviewing the necessary tools from graph theory. The cosine Kuramoto model will then be detailed and new work will be presented that will place bounds on the minimum node velocity.

2.1 Graph Theory

Graph theory has many interesting and useful applications. Graph theory provides a method to coordinate multiple agents using efficient matrix and vector operations. For example the incidence matrix presented in (2.1) allows for the differencing of vectors. Furthermore, graph theory facilitates designing distributed algorithms. A review of basic Graph Theory is necessary to understand the principles and algorithms presented in this chapter.

In Fig. 2.1, the black circles correspond to vertices, nodes or robots and the blue arrows represent directed edges. In this work, a directed edge represents the passing of information from one node or robot to another. Which nodes are passing information and to whom they are passing that information, can be compactly represented by the incidence matrix.

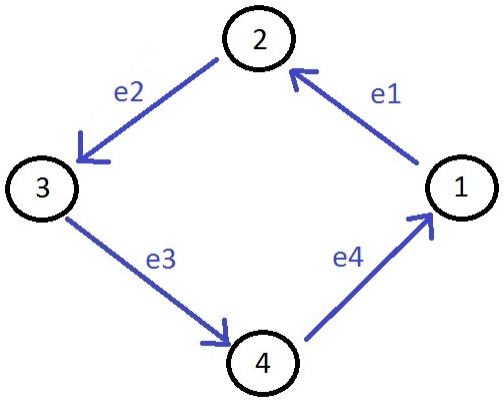


Figure 2.1: A Directed Graph

The indicence matrix D is a two dimensional array that compactly stores the relationship between directed edges and vertices in a graph [3],

$$D = [d_{ij}] \text{ where } d_{ij} = \begin{cases} -1 \text{ if node } i \text{ is the tail of } e_j \\ 1 \text{ if node } i \text{ is the head of } e_j \\ 0 \text{ otherwise} \end{cases} \tag{2.1}$$

The incidence matrix for the graph in Fig. 2.1, where the columns correspond to the edges of the graph, and the rows to the nodes is

$$D = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}. \tag{2.2}$$

In Section 2.2 the incidence matrix will be used to compare the phase angle of nodes confined to a radius, and in Section 2.3 will be used to compare vehicle headings .

Another important matrix used in this work is the Laplacian [3]. There are several ways to define the graph Laplacian, but they all result in the same entries for the elements of the matrix. One definition is, $L = DD^T$, where L denotes the graph Laplacian and D the incidence matrix. The Laplacian matrix for the graph shown in Fig. 2.1 is:

$$L = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix} \tag{2.3}$$

The Laplacian matrix has the useful property of always being a symmetric and positive semi-definite (PSD) matrix.

Additionally, Fig. 2.1 is a special type of graph, called a cycle graph. A cycle graph has the additional property that:

$$L = DD^T = D^T D = -D - D^T \tag{2.4}$$

This property is used in Appendix B

2.2 Cosine Kuramoto Model

If the the sine coupling of the Kuramoto model is replaced with a cosine coupling, a repulsive force is produced between agents. If the repulsive force between agents is confined to a cycle graph, then agents will continue to alter their state until a configuration is reached that allows for balanced repulsive forces.

If N agents or nodes are confined to remain a fixed radial distance from a center position, a control force is applied to the agents angular position as:

$$\dot{\phi} = u, \tag{2.5}$$

where ϕ represents a vector of angular positions of the agents on a circle and u is a vector of control forces. The agents can then be driven to a balanced distribution using a cosine

Kuramoto model:

$$u = KD\cos(D^T\phi), \tag{2.6}$$

where D is the incidence matrix, and K is a positive coupling gain.

Equations (2.5) and (2.6) can be combined as,

$$\dot{\phi} = KD\cos(D^T\phi). \tag{2.7}$$

As a simple example of a system executing the dynamics of (2.7), four nodes are confined to a radius, each with a unique phase angle; see Fig. 2.2.

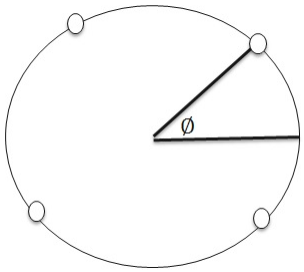


Figure 2.2: Four Nodes Confined to a Radius

The difference in phase angle ϕ can be computed using the transpose of the incidence matrix as,

$$D^T\phi = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{bmatrix} = \begin{bmatrix} \phi_2 - \phi_1 \\ \phi_3 - \phi_2 \\ \phi_4 - \phi_3 \\ \phi_1 - \phi_4 \end{bmatrix}.$$

Furthermore the cosine Kuramoto dynamics of (2.7) applied to the same system are,

$$Dcos(D^T\phi) = \begin{bmatrix} cos(\phi_1 - \phi_4) - cos(\phi_2 - \phi_1) \\ cos(\phi_2 - \phi_1) - cos(\phi_3 - \phi_2) \\ cos(\phi_3 - \phi_2) - cos(\phi_4 - \phi_3) \\ cos(\phi_4 - \phi_3) - cos(\phi_1 - \phi_4) \end{bmatrix}$$

If we examine how node one evolves, $\dot{\phi_1}$ will be positive if $cos(\phi_1 - \phi_4)$ is greater than $cos(\phi_2 - \phi_1)$, moving node one towards node four and away from node two. It is important to note that in order for the system to evenly distribute, the ordering of nodes must be such that $0 \leq \phi_1(0) < \phi_2(0) < \phi_3(0) < \phi_4(0) < 2\pi$. The convergence of nodes executing the cosine Kuramoto dynamics, to an even distribution is stated in Theorem 2.2.1.

Theorem 2.2.1. *In the cycle graph G, assume N vehicles' initial orientations satisfy $0 \leq \phi_1(0) < \phi_2(0) < ... < \phi_N(0) < 2\pi, N > 2$. Under the dynamics of the new Kuramoto-like model given in Eqn. (2.6), all orientations will asymptotically converge to the balanced distribution on a circle, i.e. $\phi_{i+1} - \phi_i = \frac{2\pi}{N}$, for $i \in 1, 2, ..., N$, where $\phi_{N+1} = \phi_1$.*

For a proof of Theorem 2.2.1, see Appendix B.

2.3 Cosine Kuramoto Model with Vehicle Dynamics

The single integrator dynamics of (2.7) can be modified to incorporate the dynamics of a two wheel steering car (Ackermann model), or the dynamics of a differential drive vehicle (unicycle model). First the dynamics of the vehicle will be presented and then the cosine Kuramoto dynamics will be incorporated.

2.3.1 Two Wheel Steering Car

The two wheel steering car has the ability to change the angle of its front wheels relative to the rear wheels. This steering angle corresponds to α in Fig. 2.4 and θ corresponds to the current heading of the car.

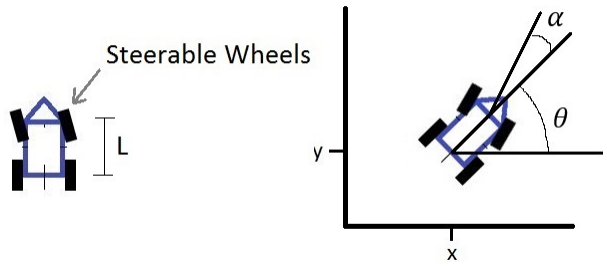


Figure 2.3: A Two Wheel Steering Model

In order for the two wheel steering car to change its heading θ, it must move forward or backward. Unlike the unicycle, it can not rotate in place. The dynamics of the two wheel steering car are

$$\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta(t) \\ \sin\theta(t) \\ (\tan\alpha(t))/L \end{pmatrix} * v(t) \tag{2.8}$$

where L is the wheelbase, and $v(t)$ is the forward velocity of the vehicle.

The two wheel steering dynamics can be incorporated into the cosine Kuramoto model. First, the states of the system are grouped into a row vector

$$q = [x^T, y^T, \theta^T, c^T, r]^T \tag{2.9}$$

where x and y are column vectors containing the Cartesian coordinates, θ is the heading of each orbiting agent, c is the coordinate of the central node, and r is the desired radius of the convoy.

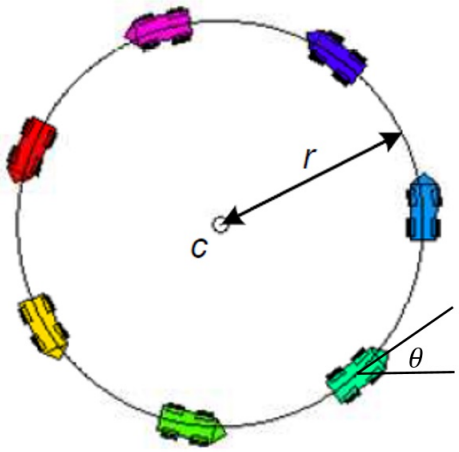


Figure 2.4: A Convoy of Two Wheel Steering Cars

Then the dynamics of the system evolve according to

$$\dot{q} = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta} \\ \dot{c} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} v * \cos\theta(t) \\ v * \sin\theta(t) \\ v * (\tan\alpha(t))/L \\ f_c(t) \\ f_r(t) \end{pmatrix} \tag{2.10}$$

where $f_c(t)$ and $f_r(t)$ are functions that dictate the change in central node position and change in the radius of the convoy. The column vectors of velocity and steering control are defined as,

$$v = w * d \tag{2.11}$$

$$\alpha = atan(\mathbf{1}\frac{l}{r} + k_p * e + k_d * \dot{e}), \tag{2.12}$$

where w incorporates the cosine Kuramoto dynamics, and d is current radial distance of each agent. The distribution of agents affects the velocities of agents v and is determined by

(2.13), where θ is the heading of each agent; thus

$$w = u + KDcos(D^T\theta). \tag{2.13}$$

As presented in [1], e is a column vector representing the error between the desired radius and the actual radial position of each agent, $e = d - \mathbf{1}r$, $d = ||p - \mathbf{c}||^r$, $p = [p_1, ..., p_N]^T$, $p_i = [x_i, y_i]^T$, $u = [u_1, ..., u_N]^T$, where N is the number of agents. Furthermore, the derivative of the error, $\dot{e}$, is defined as:

$$\begin{aligned}
\frac{de_i}{dt} &= \frac{d||p_i - c||}{dt} - f_r(t) \\
&= \frac{(p_i - c)^T}{||p_i - c||}(\dot{p}_i - f_c(t) - f_r(t) \\
&= \frac{(p_i - c)^T}{d_i}\begin{pmatrix} v_i cos\theta_i - f_{c1}(t) \\ v_i sin\theta_i - f_{c2}(t) \end{pmatrix} - f_r(t)
\end{aligned}$$

for $i = 1, ..., N$. The system dynamics of (2.10) were implemented in simulation and the results are detailed in Section 4.2. However, the hardware test-bed consisted of unicycle robots and therefore (2.10) needed to be modified accordingly.

2.3.2 Unicycle

The unicycle model effectively models the dynamics of a differential drive robot, or if minimum rate constraints are imposed, models a fixed wing aircraft in two dimensions.

If both the left and right wheels rotate at the same rate and in the same direction, the vehicle will travel forward or backward in a line. A turn can be induced by varying the rate of rotation of one wheel relative to the other. Unlike the two wheel steering car, it is even possible for the unicycle to rotate in place, by rotating each wheel with the same speed but in opposite directions. The current heading of the vehicle is described by θ, the desired heading by θ_d, and the error between the current heading and the desired heading by θ_e; see Fig. 2.5.

The dynamics of the unicycle model can be characterized in terms of the right and left wheel velocities ($V_r,\ V_l$) and are described by

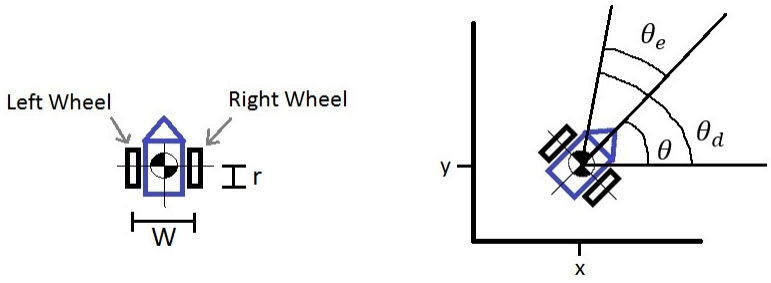


Figure 2.5: A Unicycle Model

$$\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \frac{r}{2}(V_r + V_l)\cos\theta \\ \frac{r}{2}(V_r + V_l)\sin\theta \\ \frac{r}{W}(V_r - V_l) \end{pmatrix} \tag{2.14}$$

where r is the radius of the wheel, and W the width of the robot; see Fig. 2.5. Additionally, the velocity and angular rotation of the vehicle can be characterized in terms of the right and left wheel velocities

$$\begin{aligned} v &= (V_r + V_l) \\ w &= (V_r - V_l) \end{aligned} \tag{2.15}$$

The cosine Kuramoto dynamics of (2.10) were modified to work with the unicycle model as,

$$\dot{q} = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta} \\ \dot{c} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \frac{r}{2} * v * \cos\theta(t) \\ \frac{r}{2} * v * \sin\theta(t) \\ \frac{r}{W} * (w + k_p * e + k_d * \dot{e}) \\ f_c(t) \\ f_r(t) \end{pmatrix} \tag{2.16}$$

The dynamics detailed in (2.10) and (2.16) provide a method to evenly distribute a homogeneous convoy of agents around a stationary or moving central node. Before implementing these dynamics in simulation and on a hardware testbed, the question of how quickly can

the central node move relative to the velocity of the orbiting nodes arose. The next section will place bounds on the relative velocities.

2.4 Minimum Node Velocity

"Helen of Geometers" - Boyer referring to the cycloid

This section will determine the minimum velocity that the orbiting exterior nodes must maintain, $V_{n_{min}}$, in order to achieve a balanced configuration over time.

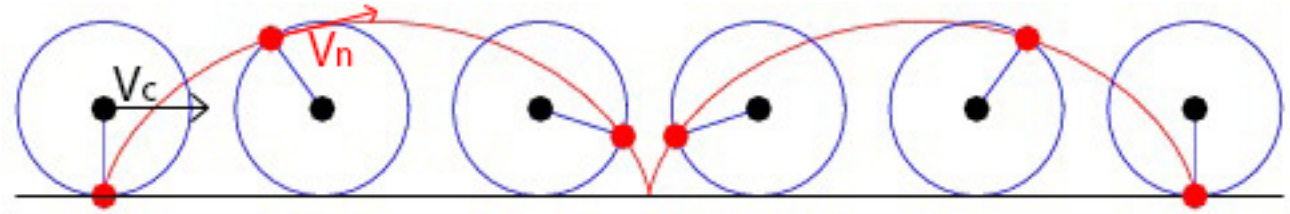


Figure 2.6: Cycloid

A few simplifying assumptions will be made. First, a fixed radius r is assumed. The desired radius of the convoy is not time varying. This leads to the next assumption, that the orbiting nodes stay on the prescribed radius exactly.

Assumption 2.4.1. *Orbiting nodes stay on the predefined radius, i.e.,* $e_r = 0$

If an orbiting node were to deviate from the desired radius, it must travel a greater distance in order to achieve the same balanced final configuration. This would result in a greater time to convergence or an increase to the minimum velocity of the orbiting nodes.

Additionally, the orientation of the orbiting nodes must remain perpendicular to the desired radius of the convoy since the balanced configuration of the agents is reliant upon the relative orientation between agents.

Assumption 2.4.2. *Orbiting nodes maintain an orientation perpendicular to the orbiting radius, i.e.,* $\theta_n \perp r$

The path taken by agents orbiting with minimum velocity that do not violate Assumptions 2.4.1 and 2.4.2 is a cycloid. A cycloid is the path traced out in time by a point fixed to the exterior of a rolling wheel, see Fig. 2.6 [4]. For a single hump of a cycloid, the arc length is $8r$, where r is the radius of the rolling wheel. Furthermore, the center of a cycloid will travel $2\pi r$ for every hump, or a distance of $8r$ as traced by a point on the circumference of the wheel. Nodes orbiting a central node will trace the path of a cycloid. When V_c is the velocity of the central node and V_n is the velocity of the exterior nodes, one has,

$$V_c = \frac{2\pi r}{\Delta t} \tag{2.17}$$

and

$$V_n = \frac{8r}{\Delta t} \tag{2.18}$$

Solving 2.17 for Δt and substituting into 2.18 results in:

$$V_n = \frac{4}{\pi} V_c \tag{2.19}$$

This means that V_n must be only slightly greater than $\frac{4}{\pi}V_c$ in order to approach $(\theta_i - \theta_j = \frac{2\pi}{N})$ as $t \to \infty$.

The cosine Kuramoto model enables the even distribution of nodes that are confined to a radius based upon the phase angle of each node. The cosine Kurmoto dynamics can also be used to evenly distribute vehicles in a convoy based on the heading angle of each vehicle. When vehicles are confined to the radius of a circle, and maintain an orientation that is perpendicular to that radius, a control signal can be applied to drive the vehicles around the circle until an even distribution in heading is obtained. Because each vehicle is perpendicular to the radius, an even spacing between agents will also be obtained. The results of implementing the cosine Kurmamoto model in simulation and on a hardware testbed is presented in Sections 4.2 and 5.1.

Chapter 3
OBSTACLE AVOIDANCE

Many different methods for obstacle avoidance have been developed by the robotics community. The principal reason for the large variety is due to the diversity in robotics platforms. The algorithm is generally tailored to the the sensing, actuation and computational capabilities of the platform.

When choosing an algorithm for the Johnny Robots (unicycle dynamics), it was known that the location of obstacles would be determined by an overhead camera system, and that the obstacles would be local and discrete. The Johnny Robots are a platform very similar to the one used in a league of robotic soccer. These similarities made the limit-cycle obstacle avoidance algorithm [2] a natural choice to implement on the Johnny Robots.

This chapter will begin by presenting how to determine if an obstacle resides between a robot and a goal location using coordinate geometry. If an obstacle is present, a method to determine which way to maneuver around an obstacle is discussed. Additionally, if there are multiple obstacles, a method is presented to group obstacles to avoid a local minima.

3.1 Seeking

If the Cartesian coordinates of the goal are (G_x, G_y) and the location of a robot or agent node are (R_x, R_y), then the Euclidean distance from a robot to the goal can be defined as [2],

$$a = G_y - R_y,\ b = R_x - G_x$$

$$\boxed{\text{Robot To Goal}} = \sqrt{a^2 + b^2},$$

and the standard equation of the line from the robot to the goal can be defined by,

$$ax + by + c = 0,$$

where

$$c = (R_y - G_y)G_x + (G_x - R_x)G_y.$$

If the straight line path from the robot to the goal is obstacle free, then the robot should move directly toward the goal, reducing the distance between the two. A method of using coordinate geometry to determine the relative location of obstacles is detailed in the next section.

3.2 Area of triangle using vertices

At each time step, it must be determined if any portion of an obstacle is located between the robot and the goal. One method of determining if an obstacle is present utilizes the area of a triangle using vertices.

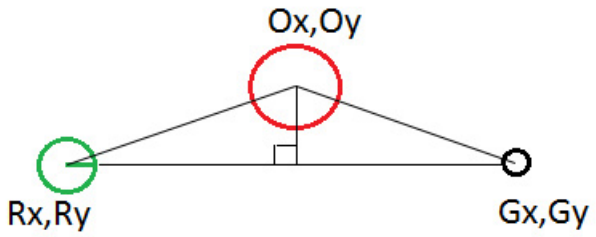


Figure 3.1: Obstacle Geometry

The vertices of the triangle are composed of (R_x, R_y), (G_x, G_y) and (O_x, O_y). If (O_x, O_y) is placed at the top of the triangle, then the height of the triangle represents the distance of the straight line path between (R_x, R_y), (G_x, G_y) and (O_x, O_y).

$$\boxed{\text{Area of Triangle Using Vertices}} = A_V = \frac{1}{2}(aO_x + bO_y + c)$$

$$\boxed{\text{Area of a Triangle}} = A = \frac{1}{2}Base * Height$$

The standard "Area of a Triangle" can be manipulated to solve for the height of a triangle knowing the Area and Base,

$$Height = \frac{2 * A}{Base}.$$

And substituting A_v for A, we can solve for d as:

$$d = Height = \frac{2 * A_V}{Base}$$

$$d = \frac{aO_x + bO_y + c}{\sqrt{a^2 + b^2}} \tag{3.1}$$

If the scalar distance $|d|$ between the center of the obstacle and the straight line path to the goal is less than the radius of the obstacle plus the radius of the robot, then a limit cycle obstacle avoidance maneuver should be executed. If $|d|$ is greater than the radius of the obstacle plus the radius of the robot, then the seeking behavior can be executed. Additionally, the sign of d will determine whether to avoid the obstacle in the clockwise or counterclockwise direction.

3.3 Limit Cycle

A ground robot moving in a two-dimensional plane exhibits non-linear dynamics as discussed in Section 2.3.2. If a limit cycle can be imposed around a local obstacle, a robot can orbit the obstacle on a path determined by the limit cycle until the obstacle no longer resides on the straight line path to the goal.

From [2], a non-linear two-dimensional system can move with limit cycle dynamics if:

$$\begin{aligned} \dot{x} &= y + x(r^2 - x^2 - y^2) \\ \dot{y} &= -x + y(r^2 - x^2 - y^2) \end{aligned} \tag{3.2}$$

where the radius of the limit cycle is determined by r and (x, y) is the vector difference between the center of the limit cycle and the current location of the system. A Lyapunov function:

$$V(x, y) = x^2 + y^2 \tag{3.3}$$

can be used to prove that a system executing the dynamics of (3.2) will converge to a circle with radius r,

$$\begin{aligned} \dot{V}(x, y) &= 2x\dot{x} + 2y\dot{y} \\ &= 2xy + 2x^2(r^2 - x^2 - y^2) - 2xy + 2y^2(r^2 - x^2 - y^2) \\ &= 2V(x, y)(r^2 - V(x, y)) \end{aligned}$$

where $\dot{V} > 0$ for $V(x) < r^2$ and $\dot{V} < 0$ for $V > r^2$. For more information about how (3.2) is a limit cycle see Appendix C. The second order dynamics of (3.2) will be combined with a method of determining which direction to orbit obstacles in order to determine the appropriate heading of an agent at each time step.

3.4 Desired Heading using Limit-cycle

If an obstacle is present on the straight line path between the agent and the goal, then the agent will adjust its heading in order to maneuver around the obstacle according to (3.4) as provided in [2]. In (3.4), (x, y) is the difference in location between the current Cartesian coordinates of the agent and the coordinates of the obstacle, r is the radius of the obstacle plus the agent, and d is the distance as defined in (3.1), thus

$$\begin{aligned} \dot{x} &= \frac{d}{|d|}y + x(r^2 - x^2 - y^2) \\ \dot{y} &= -\frac{d}{|d|}x + y(r^2 - x^2 - y^2). \end{aligned} \tag{3.4}$$

The desired heading of the agent is then found from (3.5) as

$$\theta = tan^{-1}(\dot{y}, \dot{x}). \tag{3.5}$$

A new desired heading will be calculated each time a location update is obtained. A proportional-integral-derivative (PID) controller is used to determine the appropriate control signal that will move the vehicle from its current heading to the desired heading.

3.4.1 PID

The proportional-integral-derivative (PID) controller is the most widely used controller in industry. Although this type of controller is generally not optimal, a stabilizing control signal can often be obtained without having a model of the system. The PID controller has three tunable gains that act on the error that results from feedback.

The control signal is of the form [5]

$$u(t) = K_P e(t) + K_I \int_0^t e(t)\, dt + K_D \frac{de(t)}{dt} \tag{3.6}$$

where

1. K_P contributes to the magnitude of $u(t)$, proportional to the size of the error. This gain also contributes to stability of the system and has a medium rate of response.

2. K_I eliminates errors in the system that are persistent after the application of proportional control. This gain has a slow rate of response and if chosen too large, may lead to undesirable oscillations.

3. K_D is the derivative gain that can reduce rates of oscillation. It has a fast response rate but is sensitive to noise.

The PID controller can be used to orient a unicycle model. If the current orientation of the vehicle in Fig. 2.5 is $\theta = \frac{\pi}{4}$ and the desired orientation is $\theta_d = \frac{\pi}{2}$ then the instantaneous

error in heading is,

$$\theta_e = \theta_d - \theta = \frac{\pi}{2} - \frac{\pi}{4} = \frac{\pi}{4}. \tag{3.7}$$

Substituting e for θ_e in Eqn. 3.6 and if constants were chosen such that $K_p > 0$, $K_I > 0$, and $K_d > 0$ then $u > 0$. A control $u > 0$ applied to the vehicle will result in a $w > 0$ or $\dot{\theta} > 0$ and a counterclockwise rotation as expected.

If multiple obstacles are present, obstacles will be grouped and collectively avoided according to the method detailed in the next section.

3.5 Multiple Obstacle Avoidance

If there are multiple obstacles present on the current straight line to the goal, then the obstacles are grouped together. It is necessary to group obstacles together in order for the robot to avoid local minima formed by two or more obstacles. Furthermore, it is important to consider the relative position of obstacles in order to travel efficiently towards the goal. Determining if an obstacle is on the straight line path to a goal can be performed with coordinate geometry. However, generating a complete list of obstacles that reside on the straight line path to the goal as well as obstacles that are within one robot diameter of conflicting obstacles, requires a series of conditional evaluations.

These evaluations are outlined in Algorithm 1. O_d is a matrix of obstacles currently on the path to the goal and obstacles that are touching obstacles on the path to the goal. O_n is a matrix of obstacles that are not part of O_d and therefore do not presently pose a conflict. This section concludes with an illustrative example where an agent must maneuver around

a series of three obstacles that form a local minima.

Algorithm 1: Determine heading of robot to avoid obstacles

Data: Current heading and location of Johnny Robot, locations of obstacles, location of goal

Result: Desired heading for Johnny Robot

Add the radius of the robot to the radius of each obstacle

Determine the distance d from the center of each obstacle to a straight line between the robot and the goal

if $d <$ *radius of obstacle & robot* **then**

    add obstacle to matrix O_d;

else

    add obstacle to matrix O_n;

end

if O_d *exists* **then**

    Assign the closest obstacle to "current" obstacle;

    Determine if any other obstacles are touching the closest obstacle;

    Determine if other obstacles are touching the "touching" obstacles;

    if *obstacles are touching closest obstacle* **then**

        Create new rows in matrix O_d to include touching obstacles;

        Determine averaged position of touching obstacles;

        Determine closest obstacle among touching obstacles and assign the closest obstacle to current obstacle;

        Calculate new d using averaged position of obstacles;

    end

    r = radius of current obstacle, x and y = distance from robot to current obstacle

    $\dot{x} = \frac{d}{|d|} * y + x * (r^2 - x^2 - y^2) \quad \dot{y} = -\frac{d}{|d|} * x + y * (r^2 - x^2 - y^2)$;

    Desired Heading = atan2$(\dot{y}, \dot{x})$;

else

    Desired Heading = Go to goal;

end

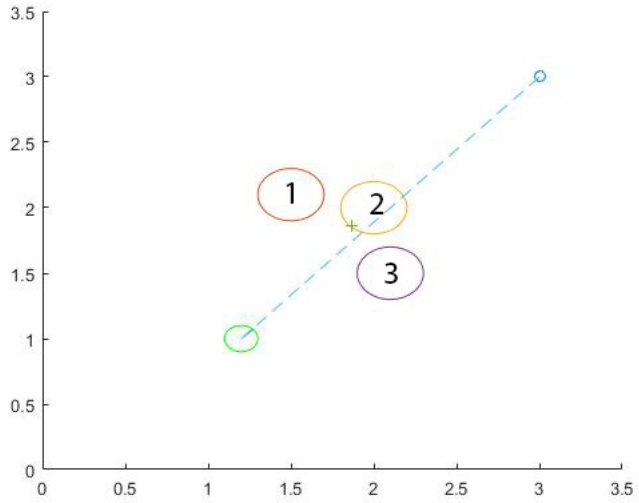


Figure 3.2: Multiple Obstacle Avoidance with Local Minima

Example 3.5.1. *This example references Fig. 3.2. The obstacles are represented by circles #1, 2, & 3 and their positions and size are listed in Table 3.1. The agent is initialized at* $(X, Y) = (1, 1.2)$ *and the goal is located at* $(X, Y) = (3, 3)$*. When executing lines 1 through 6 of Alg. 1, matrix* O_d *contains obstacle #2 and matrix* O_n *contains obstacles #1 and #3. Because* O_d *exists, an evaluation is made to determine if other obstacles are "touching" the current closest obstacle that resides on the straight line path to the goal. Accounting for the radius of the agent, Obstacles #1 and #3 are determined to pose a conflict when maneuvering around #2. Matrix* O_d *is increased to include obstacles #1 and #3 and the locations of the touching obstacles are averaged (lines 13 and 14). The averaged location is represented by + in Fig. 3.2. Obstacle #3 is found to be the closest obstacle and is assigned to be the "current" obstacle for purposes of calculating* r*,* x *and* y *in line 18. The new desired heading is listed in Table 3.2.*

Obstacle avoidance algorithms are tailored to the sensing and computational capabilities of the hardware platform. The limit cycle obstacle avoidance algorithm is optimized for a

Obstacle #	X	Y	Radius
1	1.5	2.1	0.2
2	2	2	0.2
3	2.1	1.5	0.2

Table 3.1: Position and Size of Obstacles

Averaged position	(1.8667,1.8667)
d	-0.0842
r	0.3
x	-0.9
y	-0.5
Desired Heading	-0.2935

Table 3.2: Results of Algorithm 1 from Example 3.5.1

system where the relative two dimensional position of the robot and obstacle are known and frequently updated. A limited number of calculations are required each time an update in obstacle and robot location is obtained. In this way a new desired heading can be determined and implemented before the next update is obtained. The limit cycle obstacle avoidance algorithm proved to work well when implemented on the Johnny Robots. The results of the implementation are presented in Section 5.2.

Chapter 4

SIMULATION RESULTS

This chapter presents the results of implementing the original cosine Kuramoto model found in [1], the limit cycle obstacle avoidance algorithm found in [2] and combination of the two through the use of simulated agents. The cosine Kuramoto model and limit cycle obstacle avoidance algorithms were first run independently in order understand their capabilities and limitations. The cosine Kuramoto algorithm was implemented in simulation using two wheel steering dynamics (2.10). The limit cycle obstacle avoidance algorithm found in [2] was implemented using unicycle dynamics. Implementing these algorithms independently eased the debugging process. After implementing the two algorithms independently, they were combined through the use of simulated agents.

The original implementation of the cosine Kuramoto algorithm was dependent on the orientation of the vehicles. However, in order for a vehicle to drive around an obstacle, it must alter its heading. These two requirements are in direct conflict. After implementing the original cosine Kuramoto model and the limit cycle obstacle avoidance algorithm separately in simulation, the initial idea of using a second set of simulated agents was developed. In this initial implementation, the physical representation of each robot was assigned to a simulated version of itself. This series of simulated agents ignored the presence of obstacles and simply distributed themselves according to the cosine Kuramoto dynamics. Each robot then attempted to follow the simulated version of itself while avoiding obstacles when necessary, see Fig. 4.1. The use of a second set of simulated agents enabled the convoy to evenly distribute and avoid obstacles. The specifics of this method are detailed in Section 4.4.

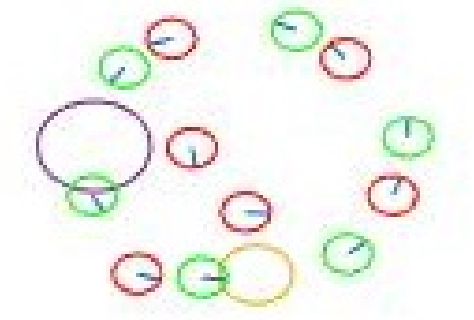


Figure 4.1: Simulated Agents (Green), Actual Agents (Red), Obstacles (Purple & Yellow)

4.1 Performance Metric

The Lyapunov function described in (B.1), and repeated here in (4.2), is a compact method of measuring the difference in spacing between agents.

$$x \triangleq D^T \phi \ mod \ 2\pi \tag{4.1}$$

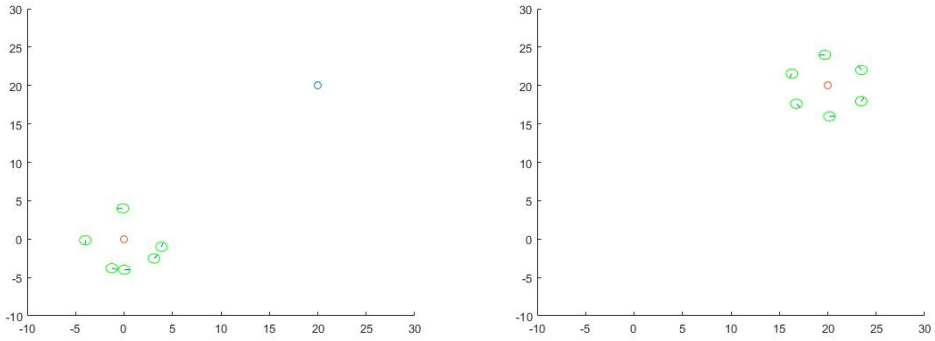
$$V = \frac{1}{2}\|x\|^2 \geq 0 \tag{4.2}$$

The more even the spacing between agents, the smaller the value of the function. This provides a measure of how close the system of agents are to achieving the goal of an even distribution.

4.2 Balanced Convoy Distribution

In order to gain familiarity with the work performed in [1] the algorithm was repeated in simulation. Six agents were randomly positioned around a central node on a fixed radius, see Fig. 4.2a. As in [1], the orbiting agents were two-wheel steering cars (2.8), programmed to execute the system dynamics of (2.10). The central node was initialized at the origin and executed a seeking behavior, Sec. 3.1, with the goal located at $(20, 20)$. The orbiting agents distributed themselves evenly around the central node while maintaining a fixed radius.

The rate at which the orbiting agents were able to obtain a balanced configuration is shown in Fig. 4.3. As expected, the Performance Metric or Lyapunov function consistently decayed. At 7 time units, there is a step decrease in the Performance Metric. This is due to the center of the convoy reaching the goal. The Performance Metric decreases to a value of approximately 3.29 as determined by (B.1).



(a) (b)

Figure 4.2: Initial (a) and Final (b) Configuration of Agents Executing the Cosine Kuramoto Dynamics

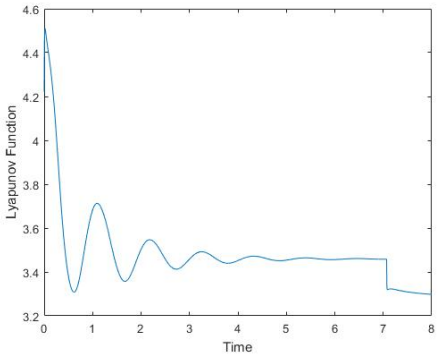
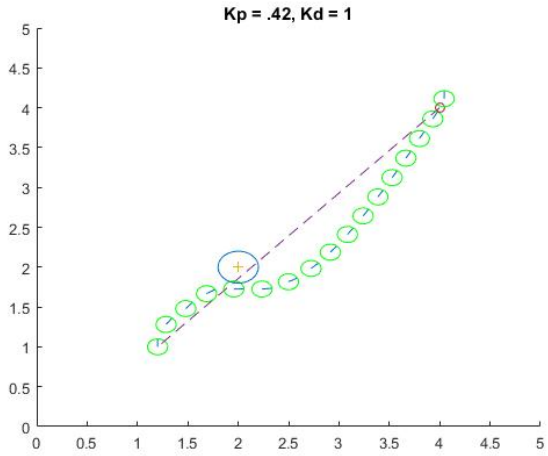


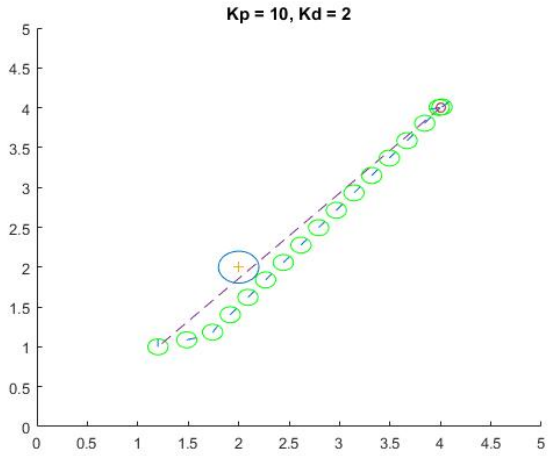
Figure 4.3: Performance Metric for the Cosine Kuramoto Dynamics

4.3 Obstacle Avoidance

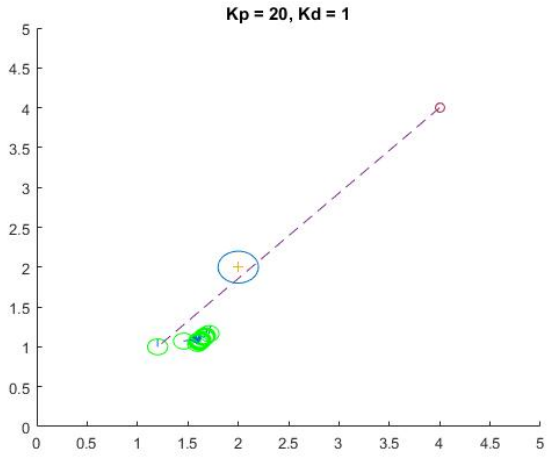
4.3.1 Single Obstacle Avoidance



(a)



(b)



(c)

Figure 4.4: The Resulting Trajectories from Various Gain Amplitudes for Single Obstacle Avoidance

Successive versions of [2] were implemented in simulation. In both versions, a node or agent executes a seeking behavior while avoiding various configurations of obstacles. In the simplest form, a desired heading is determined by Algorithm 1. The simulated agent was then oriented in accordance with the output of Algorithm 1. In a subsequent iteration, a PID controller was added to adjust the heading of the vehicle in accordance with the dynamics presented in (2.14). In the process of choosing gains for the PID controller it was found that gains which were too large would cause an over-rotation in heading and lead to instability (see Fig. 4.4c). Gains which were too small would result in the vehicle being unable to maneuver around obstacles that were in close proximity (see Fig. 4.4a).

4.3.2 Multiple Obstacle Avoidance with local minima

In Fig. 4.5, the + represents the averaged center of the obstacles and a dashed line − is included to display where the agent initially resided with respect to the averaged center. In both simulations presented in Fig. 4.5, the agent was initialized with a heading of $\frac{\pi}{2}$. The algorithm successfully avoided the minima created by the position of the obstacles and did not attempt to pass between them because it accounted for the radius of the agent.

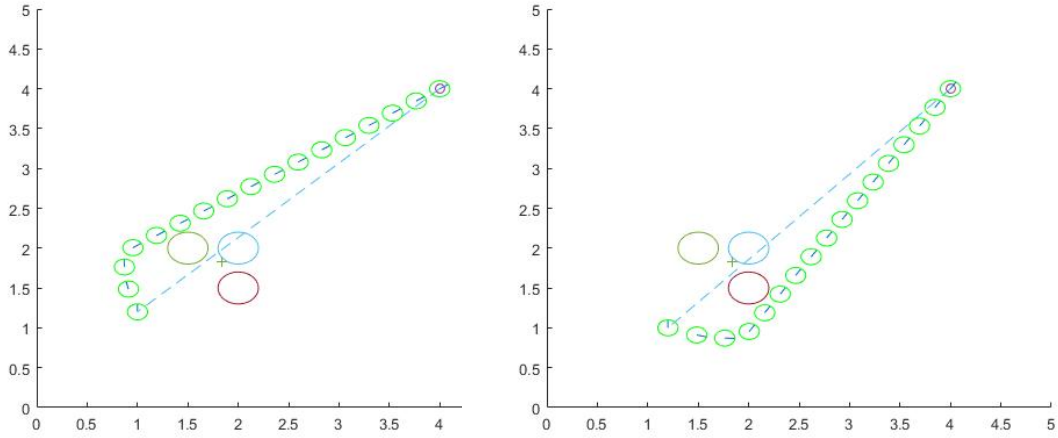


Figure 4.5: Three Obstacle Avoidance Trajectory with Minima

4.4 Convoy Distribution with Obstacle Avoidance

Prior to implementing multiple simulated agents (Section 5.3), a method for obstacle avoidance that maintained only one simulated agent per vehicle was implemented (see Fig. 4.6 and Fig. 4.7). A group of simulated agents executed the cosine Kuramoto dynamics of (2.6), while neglecting the presence of obstacles. Fig. 4.6 is a single snapshot in time and Fig. 4.7 is a series of snapshots.

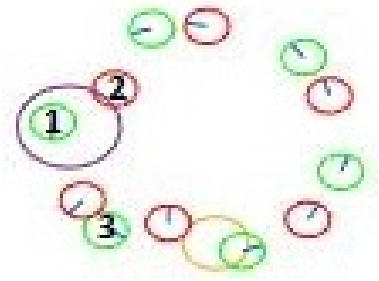


Figure 4.6: Simulated Agents (Green), Actual Agents (Red), Obstacles (Purple & Yellow)

The actual agents attempted to minimize the Euclidean distance between their position and the position of the simulated agents that they were assigned to. If a simulated agent was in conflict with an obstacle, such as agent (**1**) in Fig. 4.6, then the actual agent (**2**) would determine its heading by executing Algorithm 1, using the location of the next simulated agent in the convoy (**3**), as the location of a goal. Agent (**2**) continued to regulate its forward velocity based upon the distance from the original simulated agent (**1**). This prevented the actual agent from moving too far ahead of the original simulated agent. Once the original simulated agent was no longer in conflict with obstacles, then the actual agent would return to tracking the original agent. If both the original simulated agent and the next simulated agent in the convoy were in conflict with an obstacle, then the agent would track the next conflict free agent in the convoy for purposes of executing Algorithm 1 and obtaining a collision free heading.

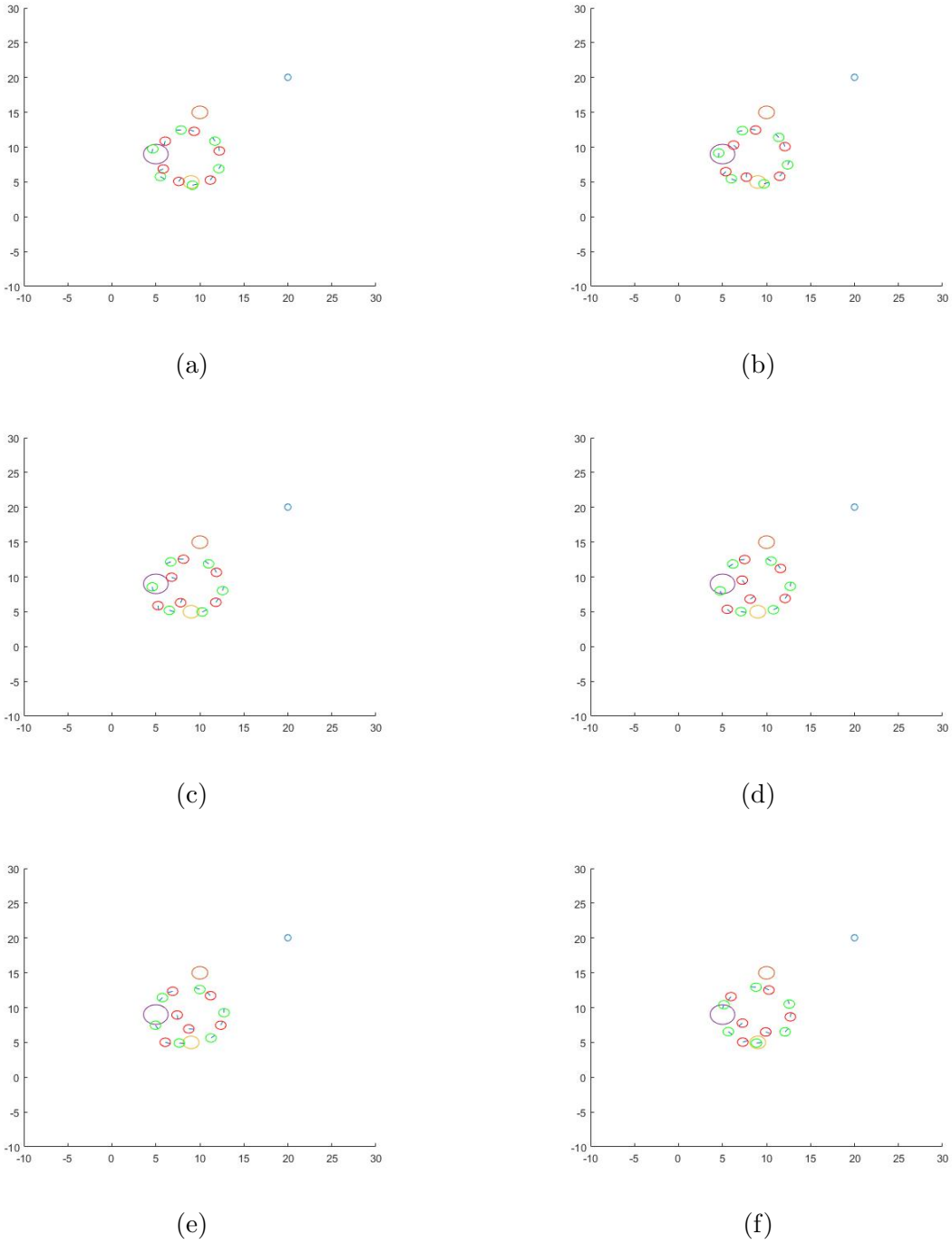


Figure 4.7: A Balanced Distribution with Multi-Obstacle Maneuvering Using Simulated Agents

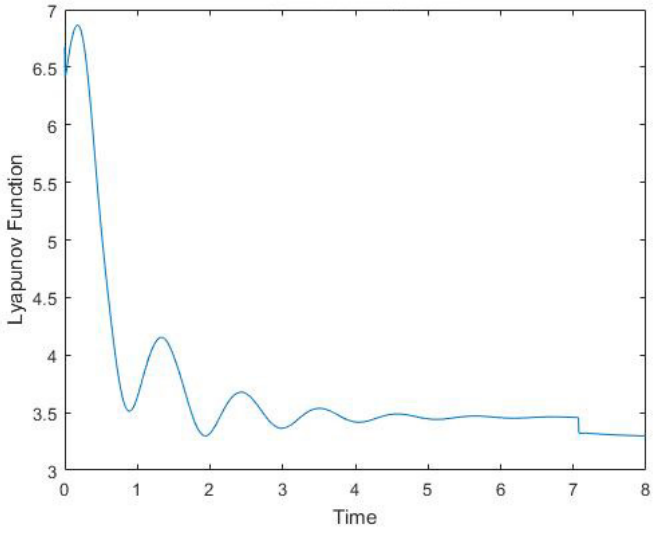


Figure 4.8: Performance Metric of Simulated Agents

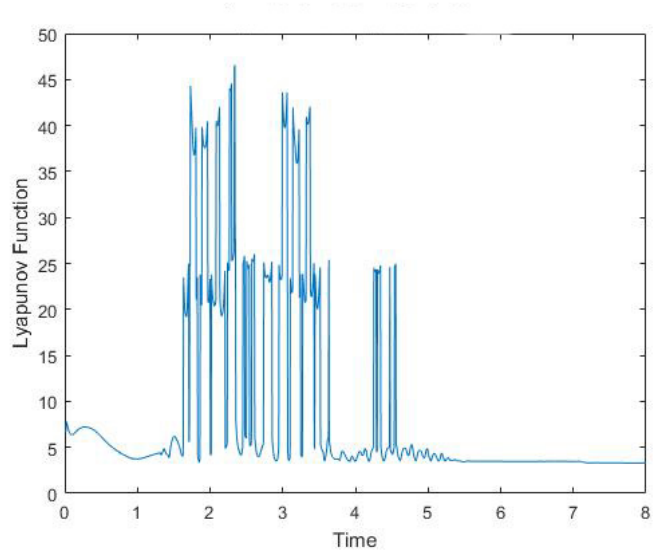


Figure 4.9: Performance Metric of Agents Avoiding Obstacles

As expected, the Performance Metric or Lyapunov function for the simulated agents consistently decayed, see Fig. 4.8. At 7 time units, there is a step decrease in the Performance Metric. This is due to the center of the convoy reaching the goal. The Performance Metric decreases to a value of approximately 3.29 as determined by (B.1). The Performance Metric of the actual agents also decays, but exhibits large spikes whenever an obstacle is encountered.

This Chapter presented the results of implementing the cosine Kuramoto model and the limit cycle obstacle avoidance algorithm in simulation. Furthermore, the Performance Metric presented in Section 4.1 proved to be a useful measure of the current distribution of the convoy. The combination of the cosine Kuramoto model and the limit cycle obstacle avoidance was achieved by having each agent track a single simulated agent. After implementing this method on the hardware testbed, the method was improved by generating multiple simulated agents per robot. This method will be presented in Section 5.3.

Chapter 5

HARDWARE IMPLEMENTATION AND RESULTS

In order for an agent to avoid a local obstacle, it must alter its heading. While maneuvering around the obstacle, the agent will deviate from the radius of the convoy and assume a heading that can be greatly different from one that is perpendicular to the radius of the convoy. In-stability arises if this change in heading is too large or assumed for too long a time period. The issue is exacerbated if there are multiple vehicles attempting to avoid multiple obstacles at the same time.

In order to overcome the limitation present in distributing agents based solely upon their current heading as in [1], a series of simulated agents was developed. The simulation results of a convoy utilizing a single set of simulated agents was presented in Section 4.4. Implementing this method on a hardware test bed illustrated the need for a *series* of simulated agents. The additional agents allow the robot to have a goal that is a predefined distance in front of the vehicle. The specifics of this method are detailed in Section 5.3.

This chapter presents the results of implementing the original cosine Kuramoto model found in [1], the limit cycle obstacle avoidance algorithm found in [2] and combination of the two through the use of simulated agents. Once again, the cosine Kuramoto model and limit cycle obstacle avoidance algorithm were first run independently in order understand their capabilities and limitations.

The platform that the cosine Kuramoto algorithm was implemented on consisted of unicycle robots instead of a two wheel steering car. Additionally, gains were tuned empirically and not found through optimization. It was possible for this algorithm to quickly achieve a balanced configuration and was able to overcome mild disturbances. However, the gains were uniquely tuned for this implementation. A modification to the algorithm or a change

in platform would require different gains.

The results of implementing the original obstacle avoidance algorithm found in [2] is also documented. This method proved to work well with unicycle robots. Implementing this algorithm separately from the cosine Kuramoto model eased the debugging process.

Finally, the two algorithms were combined through the use of simulated agents. This version was tested first without obstacles. It demonstrated resilience to disturbances. It was possible to completely displace multiple agents from their position on the radius of the convoy and have them successfully return to their position and a balanced configuration. The simulated agent version was then successfully tested in the presence of obstacles. The unicycle robots quickly achieved a balanced configuration while evenly distributing around a moving central node and avoiding static and dynamic obstacles.

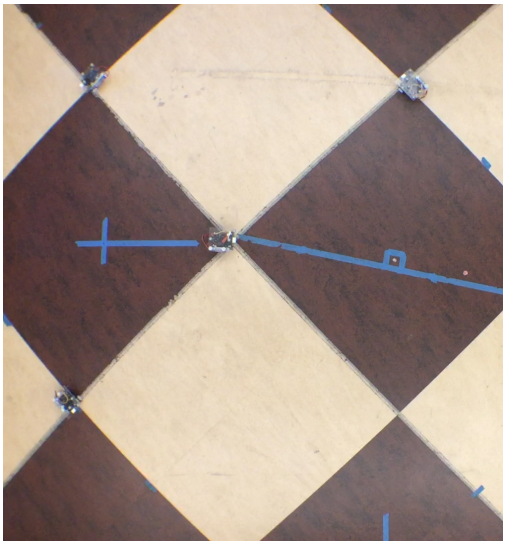
5.1 Balanced Convoy Distribution

This section presents the results of implementing a cosine Kuramoto model on a group of unicycles (2.16), to obtain an even distribution of agents as described in [1]. As detailed in Section 2.2, each agent attempts to maintain a heading that is perpendicular to a desired radius from a central node. Additionally each agent attempts to adjust its position on the radius based upon the heading of neighboring agents.

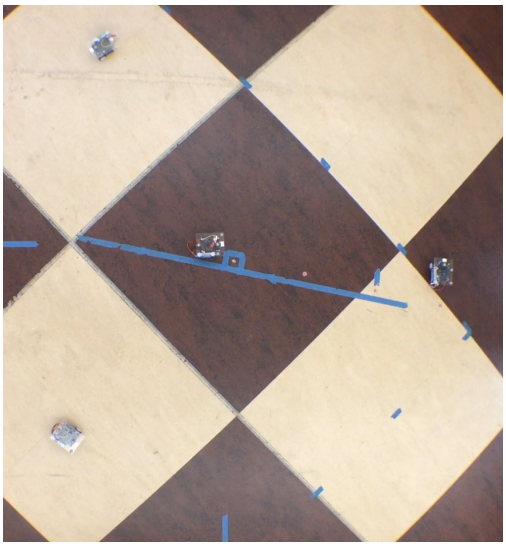
In order to implement this algorithm four gains are required; proportional and derivative steering gain (K_p and K_d), a cosine Kuramoto coupling gain (K), and a base angular velocity gain (w). When determining appropriate gains for the system, first K_p and K_d were adjusted to keep the orbiting agents on the desired radius, while w was set to keep agents orbiting the central node and still allow any single vehicle to increase its rate of orbit. Lastly, K was increased to distribute the agents.

Fig. 5.1 displays the results of running the Cosine Kuramoto model on three Johnny Robots, orbiting a central node, also a Johnny Robot. The central node traveled half of a meter and then stopped. The orbiting agents maintained a fixed radius of one half meter, while orbiting the central robot and distributing themselves. The performance metric in

Fig. 5.1 reaches the lowest value of 6.56 once the central node has come to rest at its final position. The value of 6.56 is as expected for three agents ($N = 3$).



(a)



(b)

Figure 5.1: Cosine Kuramoto Model Initial (a) and Balanced (b) Configurations

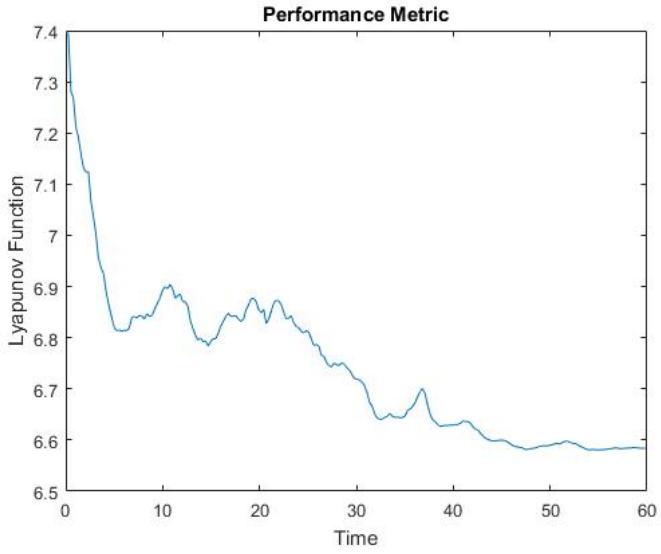


Figure 5.2: Performance Metric of Cosine Kuramoto Model

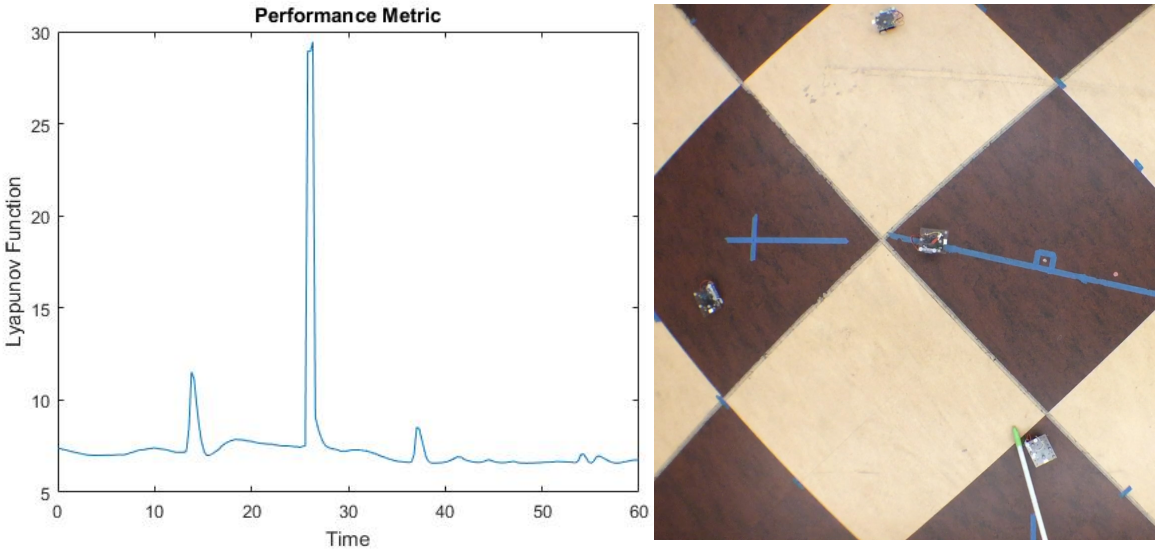


Figure 5.3: Temporary Disturbance to Agent Heading

Fig. 5.3 displays the results of executing the same objectives as Fig. 5.1, except during this run, one of the orbiting agents was randomly moved, altering its heading approximately ninety degrees. This temporary alteration in heading caused a spike in the performance metric. Each time, the Johnny Robot was able to make a full recovery from the disturbance and continue to move toward a balanced distribution. However, during other runs, if a vehicle was held in place for an extended period of time, or if the disturbance was large, then the entire convoy would ultimately fail to evenly distribute and maintain the desired radius.

Several challenges arose when implementing this algorithm in its original form. First, due to disturbances, an agent must modify its heading in order to maintain its radial distance from the central node. These changes in heading are then used by other agents in the convoy to adjust their distributions. The desire to adhere to the prescribed radius and evenly distribute are coupled and can be at odds. Second, any large deviation in heading, or slowing in relative orbit by a single agent is a disturbance that can and often is transmitted to all other agents in the convoy.

5.2 Obstacle Avoidance

This section presents the results of implementing the limit cycle obstacle avoidance algorithim detailed in [2]. The Johnny Robots were used as both a platform to demonstrate the limit cycle method as well as obstacles to be avoided. In each of the scenarios detailed below a single Johnny Robot is attempting to drive from its current location, roughly at the origin, towards a goal location of $(X, Y) = (1, 1)$. The other vehicles are positioned in various locations in order to force the moving robot to drive around them.

Implementation of the limit cycle obstacle avoidance algorithm required empirical tuning of gains. Furthermore, obstacles that were currently on the straight-line path to the goal, as well as obstacles that were within one diameter of the moving robot were accounted for when determining a path to the goal. The grouping of obstacles and the determination of a path is detailed in Section 3.5. Algorithm 2 obtained the location of the obstacles, called Algorithm 1 in order to determine an appropriate heading, and applied PID control in order

to maneuver around obstacles.

Algorithm 2: Implementation of Limit-cycle obstacle avoidance

Data: Current heading and location of Johnny Robot, locations of obstacles, location of goal

Result: Angular and linear velocity command for Johnny Robot

1 Obtain current location and heading of Johnny Robot from overhead camera system

2 Obtain current location of obstacles from overhead camera system

3 Determine desired heading of Johnny Robot using **Algorithm** 1 using data from lines 1 and 2 and location of goal.

4 Calculate error between current heading and desired heading

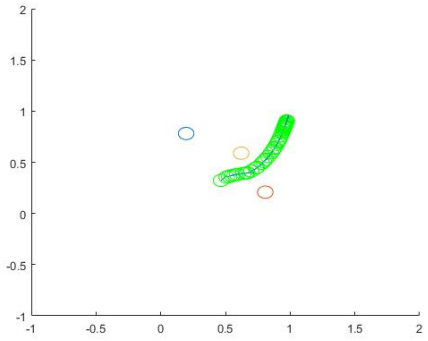
5 Calculate error between current location and location of goal

6 PID heading error to determine angular velocity

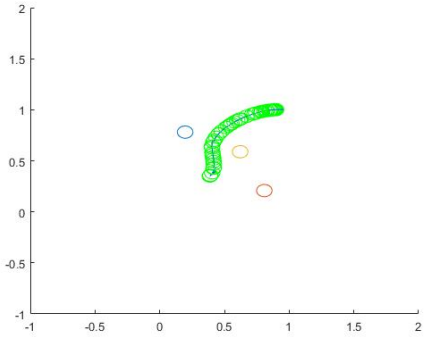
7 PID distance error to determine linear velocity

1. Single Obstacle Avoidance

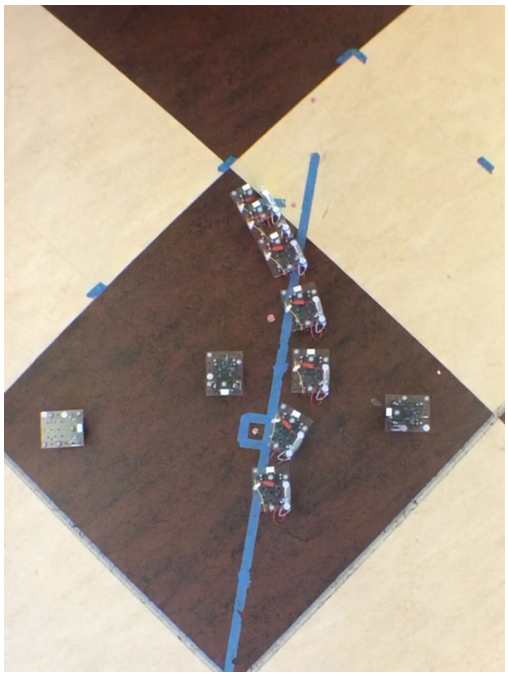
Fig. 5.4 documents the results of executing the limit cycle obstacle avoidance using the Johnny Robots. Fig. 5.4a is a plot containing the location of the obstacles and the vehicle each time a position reading was recorded. Fig. 5.4c is a composition of images pulled from video taken during the same run. These results agree with the results obtained in simulation.



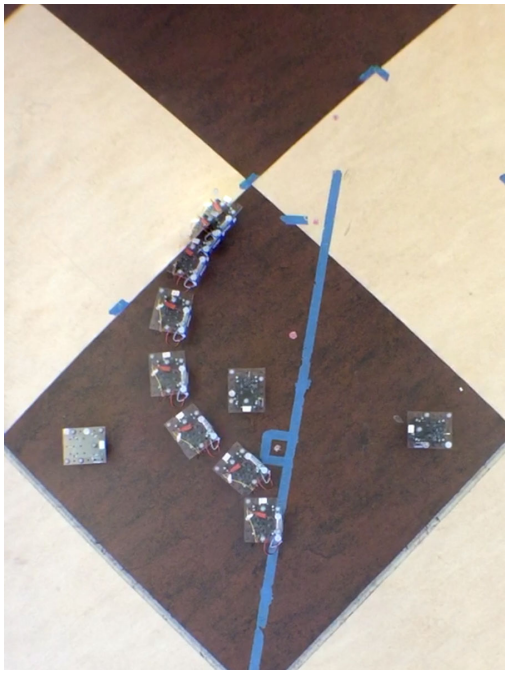
(a) Right Side



(b) Left Side



(c) Right Side



(d) Left Side

Figure 5.4: Single Obstacle Avoidance Trajectory w/ Multiple Obstacles Present

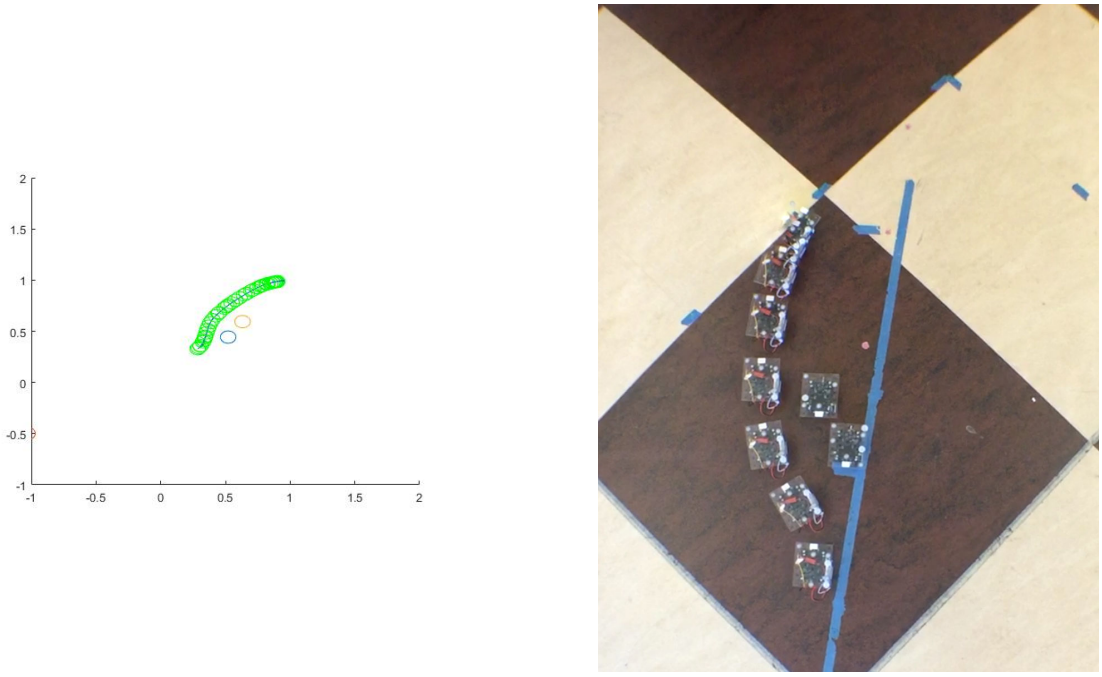


Figure 5.5: Two Obstacle Avoidance Trajectory

2. Multiple Obstacle Avoidance with local minima

As can be seen in Fig. 5.6 the limit cycle obstacle avoidance method worked well in practice, even in complex obstacle configurations that would pose problems for other algorithms. Furthermore, as demonstrated by Fig. 5.7, even if a tight passage exists between obstacles a collision free path is found and executed.

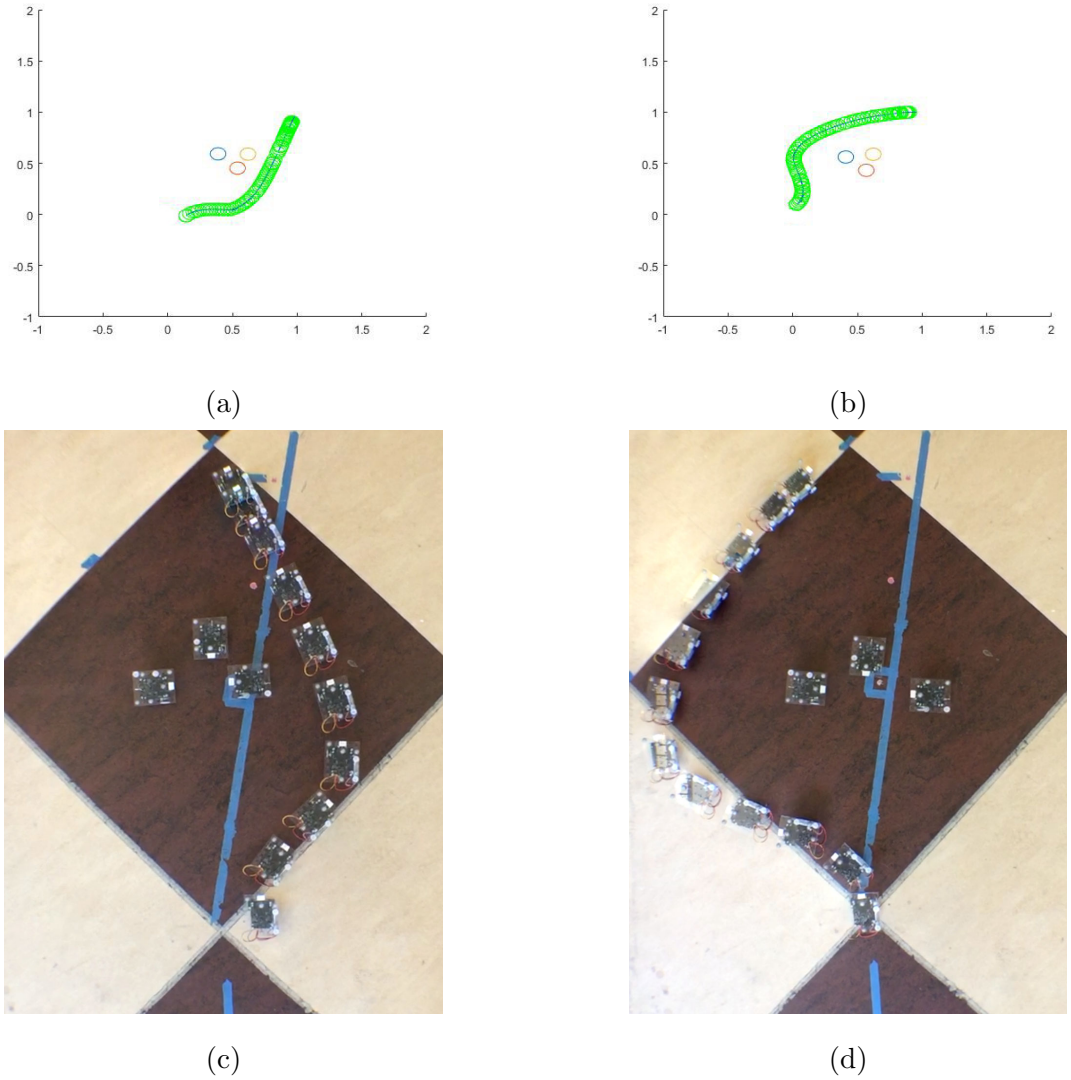


Figure 5.6: Three Obstacle Avoidance Trajectory w/ Minima

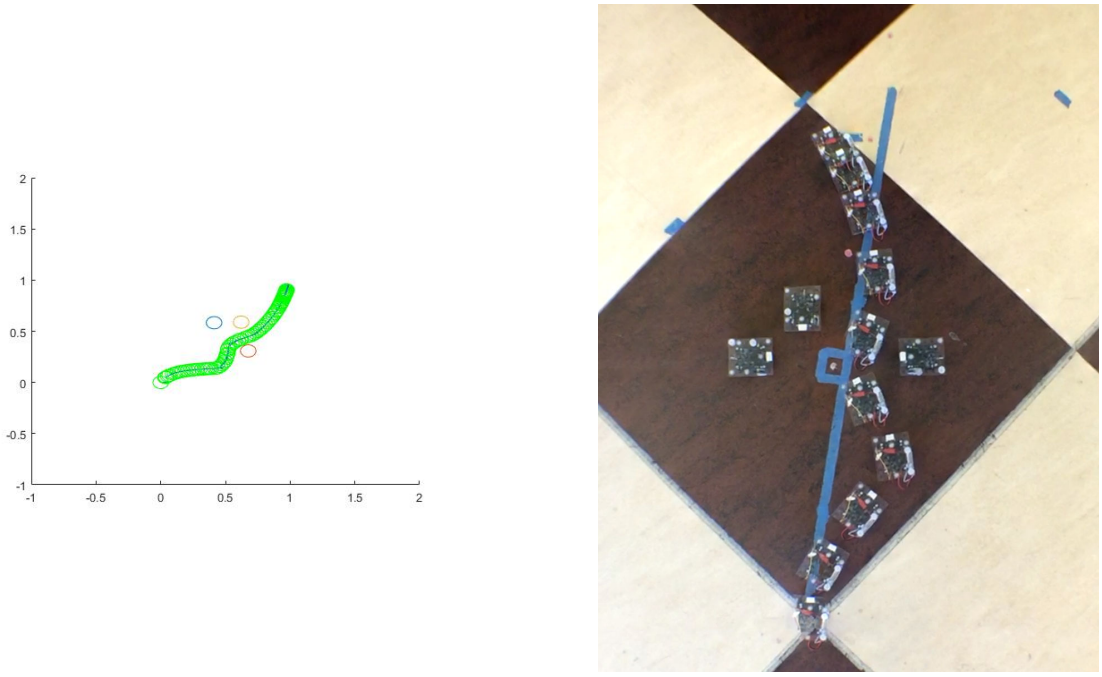


Figure 5.7: Three Obstacle Avoidance Trajectory with Passage

5.3 Simulated Agents

In order for an agent to avoid a local obstacle, it must alter its heading. While maneuvering around the obstacle, the agent will deviate from the radius of the convoy and assume a heading that can be quite different from one that is perpendicular to the radius of the convoy. In-stability arises if this change in heading is too large or maintained for too long. The issue is exacerbated if there are multiple vehicles attempting to avoid multiple obstacles at the same time.

In order to overcome the limitation present in distributing agents solely based on their current heading as in [1], a series of simulated agents was developed, see Fig. 5.8. In this configuration, the agent attempts to minimize the Euclidean distance between its current position and the current location of the simulated "Velocity" agent. Additionally, a "Heading 1" simulated agent is maintained a short distance from the Velocity agent.

In practice, the actual agent may temporarily maneuver on top of the Velocity agent

since the goal is to minimize the Euclidean distance. Separating the Velocity and Heading 1 agent provides a consistent target for the agent to orient towards. This prevents the actual agent from overshooting the target heading and then spinning 180^0.

The obstacle avoidance algorithm must have a conflict free goal location. If an obstacle appears on the simulated path of the agent, and Heading 1 is in conflict with the obstacle, then "Heading 2" will serve as a goal location, until Heading 1 is no longer in conflict, see Fig. 5.9. The spacing between Heading 1 and Heading 2 is dependent on the size of the anticipated obstacles and the radius of the convoy. The spacing must be large enough so that Heading 1 and Heading 2 cannot be in conflict with an obstacle simultaneously. However, if the spacing is too large, the agent may drive across the convoy when oriented towards Heading 2.

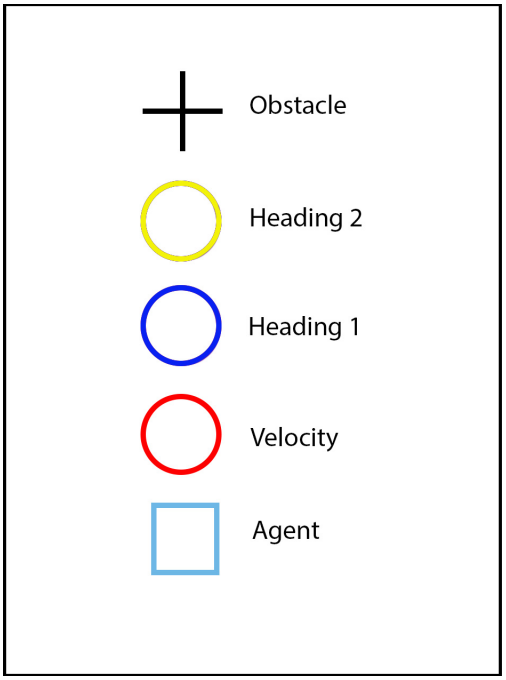


Figure 5.8: Graphical Representation of Simulated Agents

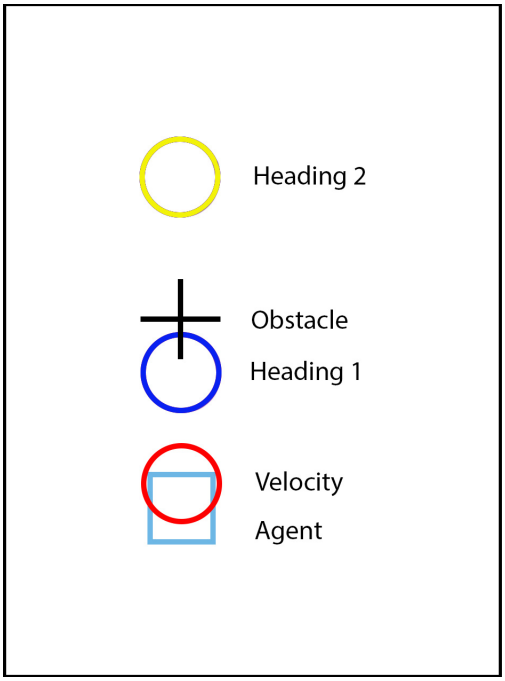


Figure 5.9: Heading 1 In Conflict with Obstacle

5.4 Balanced Convoy and Obstacle Avoidance

This section presents the results of combining limit cycle obstacle avoidance with a cosine Kuramoto model to evenly distribute a convoy through the use of simulated agents. A series of simulated agents, as detailed in Sec. 5.3 were initialized and stepped forward in time by Algorithm 3 and Algorithm 4. The simulated agents executed the cosine Kuramoto dynamics, while the actual agents or Johnny robots executed limit cycle obstacle avoidance or seeking behavior depending on the presence of obstacles.

Algorithm 3: Cosine-Kuramoto based distribution of a convoy with Limit-cycle obstacle avoidance through the use of simulated agents

Data: Current heading and location of Johnny Robots, locations of obstacles, location of goal

Result: Angular and linear velocity command for Johnny Robots

1 Obtain current location and heading of Johnny Robots from overhead camera system
2 Obtain current location of obstacles from overhead camera system
3 **if** *First iteration* **then**
4 | Initialize simulated agents $\triangle$ ahead of current locations of Johnny Robots;
5 **end**
6 Step simulation forward using **Algorithm** 4
7 **if** *SimHeading conflict with Obstacle* **then**
8 | SimHeading = SimHeading2;
9 **end**
10 Determine desired heading of Johnny Robot using **Algorithm** 1 using data from lines 1 and 2 and location of goal.
11 Calculate error between current heading of Johnny Robots and desired heading
12 Calculate error between current location of Johnny Robots and simulated positions
13 PID heading error to determine angular velocities
14 PID distance error to determine linear velocities

Algorithm 4: simulation step.m

Data: time step (dt), central node goal, radius of convoy, kuramoto gain (K), simulated positions for velocity

Result: updated sim. pos. for velocity and heading

1 $D = [IncidenceMatrix]_{size\ convoy}$

2 $\dot{\theta} = K * D * cos(D^T * \theta)$

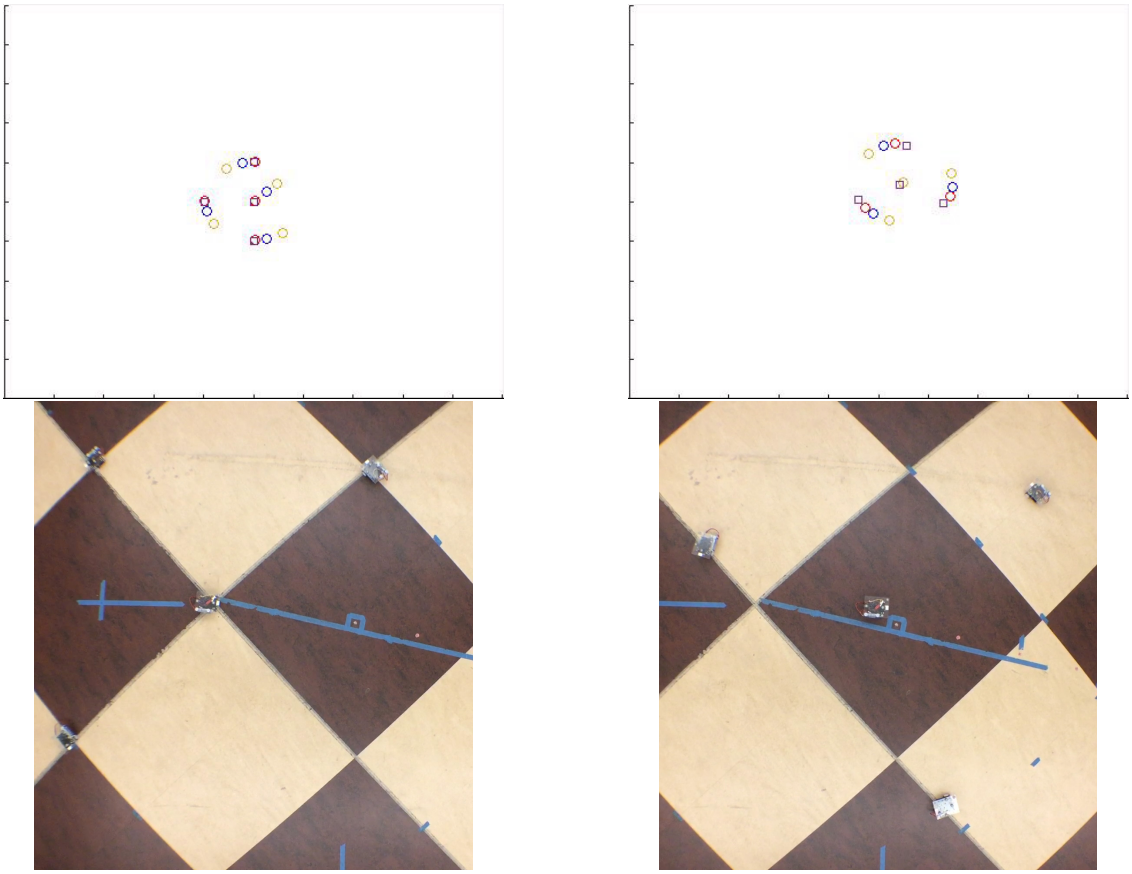
3 $\theta = \theta + dt * \dot{\theta}$

4 Move center node towards goal $(\triangle_x, \triangle_y)$

5 Move circling agents $(\triangle_x, \triangle_y) + radius * (cos\theta, sin\theta)$

1. Balanced Distribution with Simulated Agents

First, a test of the simulated agents was performed without obstacles in order to compare the performance to the cosine Kuramoto distribution of Sec. 5.1. As demonstrated by the performance metric shown in Fig. 5.11, the agents were able to evenly distribute around a moving central node in approximately sixty seconds. This is the same amount of time that it took the agents to distribute in Sec. 5.1. Fig. 5.10 is a photo of the robots in their initial and final configurations and plots the locations of the robots and simulated agents at those times.



(a) Initial configuration

(b) Balanced configuration

Figure 5.10: Initial and Balanced Configurations with Simulated Agents

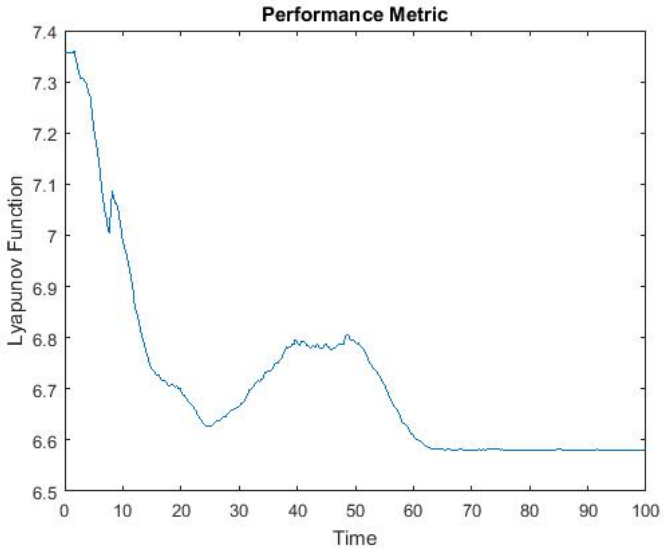
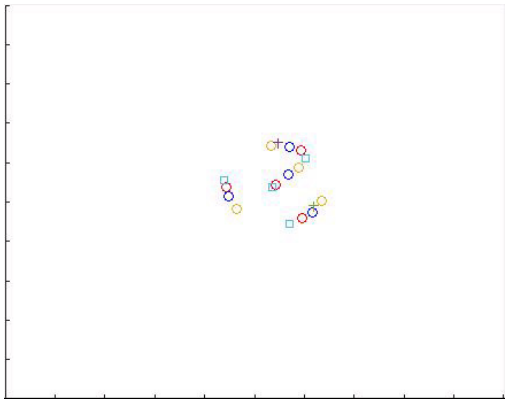


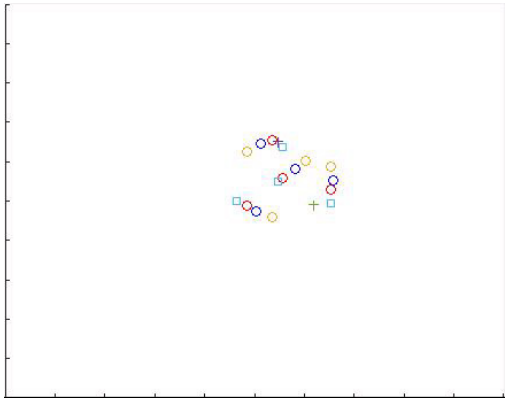
Figure 5.11: Performance Metric of Agents Evenly Distributing without Obstacles

2. Balanced Distribution with Simulated Agents and Obstacles

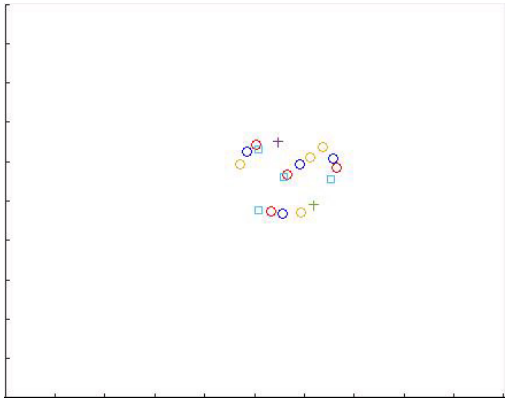
The run shown in Fig. 5.13 and Fig. 5.12 is a successful implementation of a cosine Kuramoto based distribution while avoiding multiple obstacles simultaneously. Fig. 5.12a shows the simulated agents and the Johnny Robots just as the Heading 1 (see Sec. 5.3) simulated agents come into conflict with the obstacles. This triggered the execution of a limit cycle behavior and a re-assignment of the goal location to Heading 2. Fig. 5.12b shows the locations of simulated agents and robots as they are part way around the obstacles. By this time, all simulated agents were no longer in conflict with obstacles, and the goal location has returned to Heading 1. Fig. 5.12c shows the robots after they have returned to the radius of the convoy and tracking appropriately. As anticipated, the performance metric spikes when robots are maneuvering around obstacles, but returns to the expected value of 6.56 for three orbiting agents.



(a) Pre-obstacles



(b) Mid-obstacles



(c) Post-obstacles

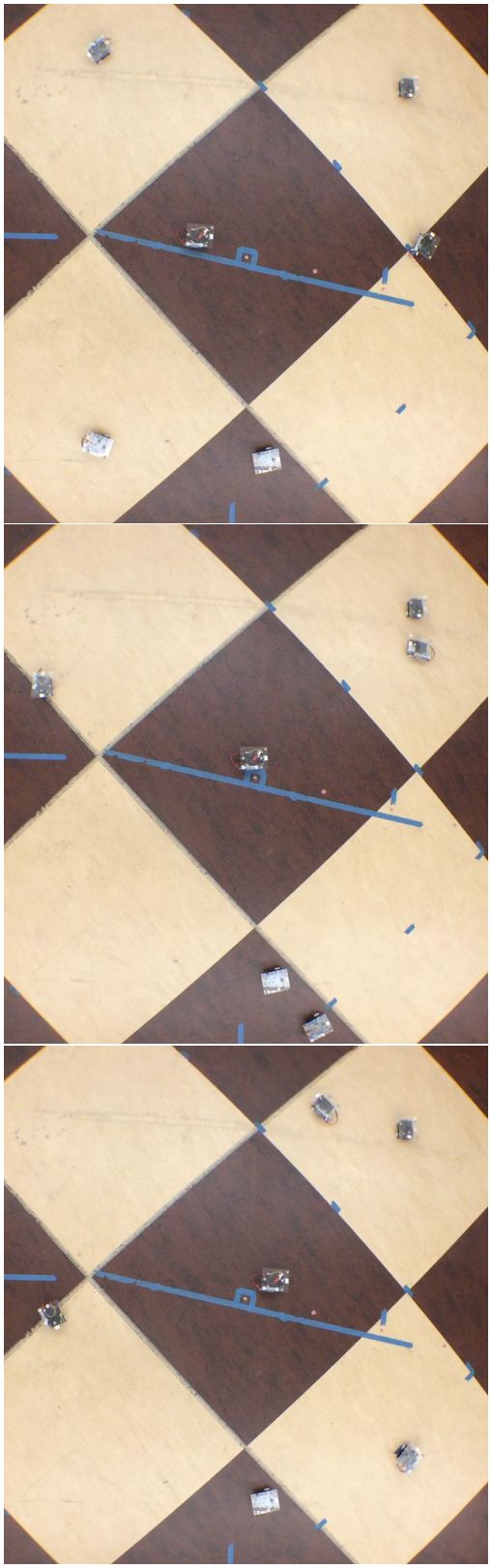


Figure 5.12: Multi-Obstacle Maneuvering Using Simulated Agents

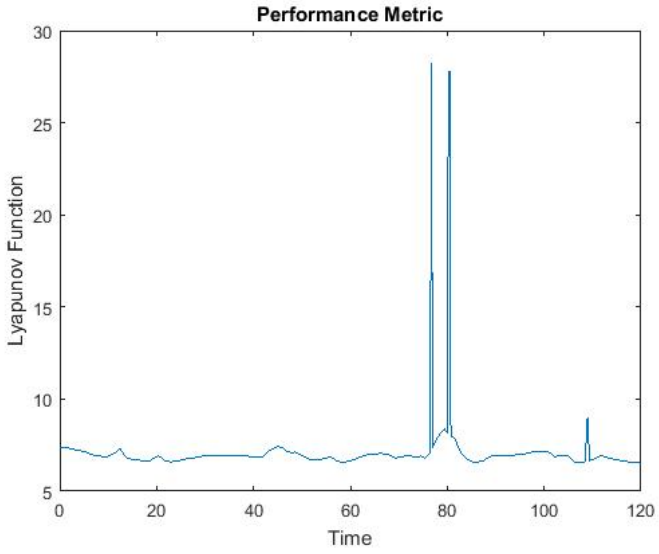


Figure 5.13: Performance Metric of Agents Evenly Distributing and Avoiding Obstacles

Chapter 6

CONCLUSION

In the cosine Kuramoto model presented in [1], the gains that kept agents on the radius of the convoy were tuned in conjunction with the gains that evenly distribute agents. The authors in [1] presented an optimization process to solve for appropriate gains in order to overcome this challenge. The optimization is difficult to implement and requires a new solution when modifications to the convoy parameters or vehicle dynamics are made. Furthermore, implementation of the cosine Kuramoto model as presented in [1] does not allow for sustained modification in heading or position of the vehicles.

The use of simulated agents allows for the combination of two or more sophisticated behaviors and also provides several other useful benefits. First, the tuning of gains to maintain a heading and velocity that are unique to the platform, can be performed independent of higher level behaviors, such as evenly distributing around a radius. Second, it was found that the use of simulated agents provided a robust and dynamically stable hardware implementation. There were many sources for disturbances when operating the Johnny Robots in the lab. As examples, the location information of a robot from the overhead camera system would be temporarily lost, or the wheels of the robot would slip due to surface debris leading to unintended heading changes. Using simulated agents prevented the robots from becoming unstable individually and/or collectively.

Although it was not implemented in this work, the use of simulated agents allows for vehicles with different dynamic properties to collectively execute higher level behaviors. The movement of simulated agents is dictated by higher level algorithms. The gains to follow the simulated agents are selected according to the vehicle type. Additionally, the behavior of the simulated agents can be changed from the cosine Kuramoto model to a different type

of behavior without re-tuning the PID velocity and heading gains.

6.1 Future Work

In this implementation, the rate at which the agents reached their intended distribution and orbited the convoy was determined by controller gains. These gains were tuned based upon hardware testing. The gains were increased until it was found that the Johnny Robots could no longer keep up with the simulated agents. In future work, it would be beneficial to characterize the maximum velocity and maximum angular rate of rotation that the Johnny Robots are capable of, and the interplay of the two. Additionally, it would be beneficial to characterize how the specific commands sent to the left and right wheels of the Johnny robot affect the velocity and angular rate of rotation. Using these characterizations and bounds, it would be possible to analytically limit the step size of the simulated agents to prevent them from outrunning the Johnny robots.

Furthermore, in this implementation, the processing of sensor data, and the determination of subsequent control signals is performed by a central computer. The graph theoretic methods used by the algorithms allow for the efficient scaling to a large number of agents. In future work, these computations could be performed by the processors on-board the vehicles. Pertinent sensor data would be sent to the respective vehicles. Then each vehicle would update the position of their respective simulated agent using the vehicle's on-board processor. Both the algorithm that distributes the agents and the algorithm that avoids obstacles have low computational overhead and could be easily handled by a micro-controller.

BIBLIOGRAPHY

[1] Z. Xu, M. Egerstedt, G. Droge, and K. Schilling, "Balanced deployment of multiple robots using a modified Kuramoto model," in *American Control Conference*, June 2013, pp. 6138–6144.

[2] D. Kim and J. Kim, "A real-time limit-cycle navigation method for fast mobile robots and its application to robot soccer," *Robotics and Autonomous Systems*, vol. 42, no. 1, pp. 17 – 30, 2003.

[3] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Princeton, NJ: Princeton University Press, 2010.

[4] E. W. Weisstein. (2014) Cycloid. Mathworld - a Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/Cycloid.html

[5] M. Egerstedt. (2014) Control of mobile robots. Coursera and Georgia Institute of Technology. [Online]. Available: https://www.coursera.org/learn/mobile-robot

[6] R. C. Arkin, *Behavior-Based Robotics*, 1st ed. Cambridge, MA: MIT Press, 1998.

[7] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, Dec 1998.

[8] G. V. Mohammed Dahleh, Munther A. Dahleh. (2011) Lectures on dynamic systems and control. Massachusetts Institute of Technology: MIT OpenCourseWare. [Online]. Available: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-241j-dynamic-systems-and-control-spring-2011/readings/

[9] X. Ding, A. Rahmani, and M. Egerstedt, "Multi-UAV convoy protection: An optimal approach to path planning and coordination," *IEEE Transactions on Robotics*, vol. 26, no. 2, pp. 256–268, April 2010.

[10] D. J. Klein, P. Lee, K. A. Morgansen, and T. Javidi, "Integration of communication and control using discrete time Kuramoto models for multivehicle coordination over broadcast networks," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 4, pp. 695–705, May 2008.

[11] S. M. LaValle, *Planning Algorithms*. New York, NY: Cambridge University Press, 2006.

[12] J. O. A. Mattuck, H. Miller and J. Lewis. (2011) 18.03sc differential equations. Massachusetts Institute of Technology: MIT OpenCourseWare. [Online]. Available: http://ocw.mit.edu/courses/mathematics/18-03sc-differential-equations-fall-2011/index.htm#

[13] P. Panyakeow, R. Dai, and M. Mesbahi, "Optimal trajectory for network establishment of remote UAVs," in *American Control Conference*, June 2013, pp. 4623–4628.

[14] H. K. Khalil, *Nonlinear Systems*. Upper Saddle River, NJ: Prentice Hall, 1996.

[15] Y. Kuramoto, "Self-entrainment of a population of coupled non-linear oscillators," in *International Symposium on Mathematical Problems in Theoretical Physics*. Heidelberg, Germany: Springer, 1975, pp. 420–422.

[16] K. Oh, M. Park, and H. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[17] E. Schoof, A. Chapman, and M. Mesbahi, "Bearing-compass formation control: A human-swarm interaction perspective," in *American Control Conference*, June 2014, pp. 3881–3886.

[18] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, Apr 1991, pp. 1398–1404.

[19] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14–23, Mar 1986.

[20] N. E. Leonard and E. Fiorelli, "Virtual leaders, artificial potentials and coordinated control of groups," in *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 3, Dec 2001, pp. 2968–2973.

[21] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, Sept 1989.

[22] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, Apr 1996, pp. 3375–3382.

[23] A. V. Savkin and H. Teimoori, "Decentralized formation flocking and stabilization for networks of unicycles," in *Proceedings of the 48th IEEE Conference on Decision and Control*, Dec 2009, pp. 984–989.

[24] M. Mesbahi and F. Y. Hadaegh, "Formation flying control of multiple spacecraft via graphs, matrix inequalities, and switching," in *Proceedings of the 1999 IEEE International Conference on Control Applications*, vol. 2, Aug 1999, pp. 1211–1216.

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

[26] A. Chapman, "Network topology design for UAV swarming with wind gusts," in *Semi-Autonomous Netwnorks: Effective Control of Networked Systems through Protocols, Design, and Modeling*. Springer, 2015, pp. 81–106.

[27] M. Mesbahi, "On state-dependent dynamic graphs and their controllability properties," *IEEE Transactions on Automatic Control*, vol. 50, no. 3, pp. 387–392, 2005.

[28] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[29] E. Schoof, A. Chapman, and M. Mesbahi, "Efficient leader selection for translation and scale of a bearing-compass formation," in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation*, May 2015, pp. 1816–1821.

[30] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[31] B. I. Triplett, D. J. Klein, and K. A. Morgansen, "Discrete time Kuramoto models with delay," in *Networked Embedded Sensing and Control*. Springer, 2006, pp. 9–23.

Appendix A

LYAPUNOV FUNCTIONS AND STABILITY THEORY

A Lyapunov function can be used to characterize the stability of an ordinary differential equation (ODE). A Lyapunov function for the dynamics of a particular system can be difficult to find, but for a physical system can often be constructed using conservation laws. Before providing the definition of a Lyapunov function it is necessary to define positive and negative definite functions, which roughly serve as a means to describe if a function V, is always increasing or decreasing around an equilibrium point.

Definition [8] - Let V be a continuous map from R^n to R. $V(x)$ is locally positive definite function around $x = 0$ if

1. $V(0) = 0$,

2. $V(x) > 0,\ 0 < \|x\| < r$ for some r, where r is a radius.

Furthermore, the function is locally positive semidefinite if $V(x) \geq 0$, locally negative definite if $-V(x) > 0$ for $x \neq 0$ or locally negative semidefinite if $-V(x) \geq 0$. With this background, we can now define the notion of a Lyapunov function.

Definition *Lyapunov function* [8] - If V is a local positive definite function and $\dot{V}$ is locally negative semidefinite with respect to the dynamics, then V is a Lyapunov function of the system.

The derivative of V with respect to time as defined in [8]:

$$\dot{V}(x(t)) = \frac{\partial V(x)}{\partial x}\dot{x} = \frac{\partial V(x)}{\partial x}f(x) \tag{A.1}$$

where $\frac{\partial V(x)}{\partial x}$ is a row vector - the gradient vector or Jacobian of V with respect to x - containing the component wise partial derivatives $\frac{\partial V}{\partial x_i}$.

If one is able to find a candidate Lyapunov function for a system, it is then possible to determine that an equilibrium point is stable using Theorem A.0.1.

Theorem A.0.1. *[8] - If there exists a Lyapunov function for $\dot{x}(t) = f(x(t))$, then $x = 0$ is a stable equilibrium point in the sense of Lyapunov. If in addition $\dot{V}(x) < 0$, $0 < \|x\| < r$ for some r, i.e., if $\dot{V}$ is LND (Locally Negative Definite), the point $x = 0$ is an asymptotically stable equilibrium point.*

The existence of a Lyapunov funtion that satisfies Theorem A.0.1 can determine that an equilibrium point is stable or asymptotically stable. However, if the candidate Lyapunov function does not satisfy the conditions of Theorem A.0.1, this does not mean that $x = 0$ is not a stable or asymptotically stable equilibrium point.

In order to develop some intuition about a Lyapunov function, an example will be provided. Often times, a Lyapunov function characterizes the energy of a system. For a mechanical system, it often is a combination of the kinetic and potential energy.

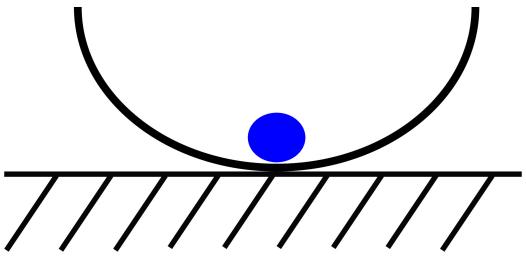


Figure A.1: A marble in the basin of a bowl

As a simple example, consider a marble and a bowl placed on the ground. And for this example, the marble can be placed at any location on the inside of the bowl. The energy of the marble can be described in terms of its height above the basin of the bowl, and its current velocity. This equation or Lyapunov function, would have its smallest value when the

marble is stationary and positioned at the basin of the bowl. This position is an equilibrium point. Furthermore, independent of where the marble is initially placed in the bowl, if no additional energy is being added to the system and friction is accounted for, then the system will always be loosing energy. Therefore, the Lyapunov function will always be decreasing. This intuitively leads to the idea that the basin is an asymptotic equilibrium point.

A Lyapunov function will be used in this work to prove that a group of nodes executing the cosine Kuramoto dynamics will converge to an even distribution.

Appendix B

COSINE KURAMOTO - PROOF OF CONVERGENCE

A Lyapunov function can be used to prove that the cosine Kuramoto model will converge to a balanced distribution presented in Theorem 2.2.1. Furthermore, the Lyapunov function plotted as a function of time can serve as a performance metric.

As in [1] define $x \triangleq D^T\phi \bmod 2\pi$ and its derivative:

$$\begin{aligned}\dot{x} &= D^T\dot{\phi} \\ &= D^T K D\cos(x) \\ &= K D^T D\cos(x)\end{aligned}$$

In order to prove 2.2.1 the "energy" of the system can be characterized by the Lyapunov function:

$$V = \frac{1}{2}\|x\|^2 \geq 0 \tag{B.1}$$

This is a valid Lyapunov function according to the Definition provided in Appendix A. Furthermore, if it can be shown that $\dot{V}$ is LND and a balanced configuration is asymptotically stable.

$$\begin{aligned}\dot{V} &= x^T\dot{x} \\ &= K x^T D^T D\cos(x)\end{aligned}$$

And using $L = D^T D$ from 2.4

$$\begin{aligned}&= K x^T L\cos(x) \\ &= -K x^T D\cos(x) - K x^T D^T\cos(x)\end{aligned}$$

where

$$x^T D\cos(x) = \sum_{i=1}^{N}(x_{i+1} - x_i)\cos x_i,\ x_{N+1} \triangleq x_1 \tag{B.2}$$

It can be shown that $x^T Dcos(x) \geq 0$ and $x^T D^T cos(x) \geq 0$, and therefore $\dot{V} \leq 0$. For a proof of $x^T Dcos(x) \geq 0$ and $x^T D^T cos(x) \geq 0$ see [1].

Theorem B.0.2. *LaSalle's Invariance Principle [14] - Let $\Omega \in D$ be a compact set that is positively invariant with respect to $\dot{x} = f(x)$. Let $V : D \to R$ be a continuously differentiable function such that $\dot{V}(x) \leq 0$ in Ω. Let E be the set of all points in Ω where $\dot{V}(x) = 0$. Let M be the largest invariant set in E. Then every solution starting in Ω approaches M as $t \to \inf$.*

Using Theorem B.0.2, define the set $\Omega_c = \{x \in [0, 2\pi)^N | \sum_{i=1}^{N} x_i = 2\pi, V(x) \leq c, c \in \Re^+\}$. Ω_c is a compact positively invariant set because $\dot{V} \leq 0$ and V is quadratic. Define

$$V_0 = \{x \in \Omega_c | \dot{V} = 0\} \tag{B.3}$$

As in [1] $\dot{V} = 0 \Leftrightarrow x_{i+1} = x_i,\ \forall i\ \in\ \{1, 2, ..., N\}$. Consider the condition $\sum_{i=1}^{N} x_i = 2\pi$, we get

$$V_0 = \{x \in \Omega_c | x_i = \frac{2\pi}{N}, \forall i \in \{1, 2, ..., N\}\} \tag{B.4}$$

Let $M = V_0$, which is (in terms of ϕ):

$$M = \{\phi | \phi_{i+1} - \phi_i = \frac{2\pi}{N}\ mod\ 2\pi,\ \forall i \in \{1, 2, ..., N\}\} \tag{B.5}$$

then every trajectory of ϕ will approach M as $t \to \inf$. This completes the proof.

Appendix C

LIMIT CYCLE

A general non-linear two-dimensional system is of the form:

$$\begin{aligned} x_1' &= f(x_1, x_2) \\ x_2' &= g(x_1, x_2) \end{aligned} \tag{C.1}$$

Can be shown to have a limit cycle using the Poincare-Bendix Theorem.

Poincare-Bendixon Theorem [12] *Suppose R is the finite region of the plane lying between two simple closed curves D_1 and D_2, and $\boldsymbol{F}$ is the velocity vector field for the system Eqn. C.1. If*

1. *at each point of D_1 and D_2, the field $\boldsymbol{F}$ points toward the interior of R, and*

2. *R contains no critical points,*

then the system Eqn. C.1 has a closed trajectory lying inside R.

The following 2nd-order non-linear dynamics are proposed in [2]:

$$\begin{aligned} \dot{x_1} &= x_2 + x_1(1 - x_1^2 - x_2^2) \\ \dot{x_2} &= -x_1 - x_2(1 - x_1^2 - x_2^2) \end{aligned} \tag{C.2}$$

with a Lyapunov function of:

$$V(x) = x_2^2 + x_2^2 \tag{C.3}$$

$$\begin{aligned} \dot{V}(x) &= 2x_1\dot{x_1} + 2x_2\dot{x_2} \\ &= 2x_1x_2 + 2x_1^2(1 - x_1^2 - x_2^2) - 2x_1x_2 + 2x_2^2(1 - x_1^2 - x_2^2) \\ &= 2V(x)(1 - V(x)) \end{aligned}$$

$\dot{V} > 0$ for $V(x) < 1$ and $\dot{V} < 0$ for $V(x) < 1$. Therefore on the level surface of $V(x) = c_1$ with $0 < c_1 < 1$, all the trajectories will be moving outward, while on the level surface of $V(x) = c_2$ with $c_2 > 1$, all the trajectories will be moving inwards. This shows that the annual region

$$M = \{x \in \Re^2 | c_1 \leq V(x) \leq c_2\} \tag{C.4}$$

is positively invariant in the sense that $x(0) \in M$ implies that $x(t) \in M$, $\forall t \geq 0$. It is also closed, bounded and free of equilibrium points since the origin $(x_1, x_2) = (0, 0)$ is the unique equilibrium point. Thus from the Poincare-Bendixson theorem, there is a periodic orbit in M. Since the above argument is valid for any $c_1 < 1$ and any $c_2 > 1$, c_1 and c_2 can be close to 1 so that the set M shrinks toward the unit circle. This shows the unit circle is a periodic orbit and a limit cycle [2].

VITA

William Howerton obtained his B.S. in Mechanical Engineering from Northwestern University in June 2007. From 2007 to 2011 he was employed as a Mechanical Engineer at Woodward MPC in Skokie, IL. At MPC, he primarily designed electric motors and actuators for aerospace applications. He developed an interest in control theory due to a desire to work on the applications that MPC's products were being used in. He welcomes your comments to `whower@uw.edu`.

- My page.

  `willhowerton.com`