Formation Control Implementation Using Kobuki TurtleBots and Parrot Bebop Drone*

Nicolas Gallardo, Karthik Pai, Berat A. Erol, Patrick Benavidez and Mo Jamshidi
Department of Electrical and
Computer Engineering
The University of Texas at San Antonio
San Antonio, TX 78249
Email: hbq744@my.utsa.edu, dxq821@my.utsa.edu, berat.erol@utsa.edu, patrick.benavidez@utsa.edu, moj@wacong.org

***Abstract*—Formation control of a collection of vehicles is a topic that has generated a lot of interest in the research community. This interest primarily stems from the increased performance and robustness that is provided by a swarm of agents as compared to an individual member. Formation control can be achieved through many approaches. The approach used by this paper is based on a leader-follower premise. A network of agents can be controlled by assigning a leader for each agent in the formation. The group as a whole will be capable of following either a Virtual Leader (VL) or an agent within the group. The algorithm applied to a test-bed consisting of three Kobuki TurtleBot2 robots. Each Turtlebot2 is programmed to follow a pre-defined virtual point in the formation. The test space is monitored by a Parrot Bebop drone hovering overhead that identifies agents uniquely through image processing techniques. The agents can then move in the test space, based on the leader's position, while maintaining a formation.**

***Index Terms*—formation control, virtual leader, robot, drone, UGV, UAV, turtlebot, parrot bebop**

I. Introduction

Groups of autonomous vehicles provide additional performance and functionality beyond the capabilities of individuals. A group can solve problems and achieve objectives which are difficult for individual robots to achieve in a reasonable time frame. In a target identification scenario, for example, the probability of targets being missed in a highly cluttered environment is much greater for a single surveillance vehicle than it is for a formation [1]. Environmental exploration is another area in which a formation of agents will have increased performance as compared to a single agent. Simultaneous Localization and Mapping (SLAM) is an algorithm used in environmental exploration that uses a graph-based map method. In [2], an implementation of SLAM is presented that represents nodes as the current pose of a robot and graph edges as constraints between each pose. In this example, a map of the robot's environment according to its position and pose can be easily communicated to collaborating agents. The additional sources of data from the members of the formation will only help with accuracy of the generated maps. Incremental and personal map data from multiple agents can be merged to get a complete map of the whole area. New data only needs to be inserted into the graph by connecting them to the correct nodes [3]. In a formation case, agents require methods to isolate detected features of their peers from that of the environmental landmarks.

Distributed group motion and control was first identified by Craig Reynolds through his study on schooling fishes and flocking birds [4]. Since then, attempts have been made to model swarm behavior based on rules of attraction and repulsion between neighbors in the swarm [5]–[7]. Such rules of attraction and repulsion can be used to guide groups of autonomous vehicles together in a controlled formation.

We were inspired by the formation control approach outlined in [8] mainly because the method has light computational requirements and has not yet been implemented on hardware. This approach requires that we know the current position of the agents, such that the attractive and repulsive forces between them can be calculated real-time and an initial formation can be maintained. Information regarding the current position of the agents can be obtained by several different methods. Finding coordinates of the agents through GPS is an attractive solution because of its high accuracy [9], so we wanted our approach to be expandable in order to make use of GPS in future real-world applications. In our GPS-denied laboratory setup, we use an overhead camera with color identification and tracking as an efficient solution to obtaining the coordinate data of each Unmanned Ground Vehicle (UGV) in the system. Color tracking is implemented using a video feed from a drone hovering overhead. UGV coordinate data is used to calculate the forces applied to the UGV to maintain the desired formation. The forces are applied as to modify the UGV's speed and orientation. This entire process is outlined in the following sections.

The underlying software architecture is an important part of our work. We use Robot Operating System (ROS), a robust message passing infrastructure, to pass all of the navigation data mentioned above back and forth between each subprocess that may need this data. For example, the coordinates of each UGV are obtained by a C++ program running the color identification and tracking algorithm. In parallel, another C++ program calculates all the forces needed to maintain

* This work was supported by Grant number FA8750-15-2-0116 from Air Force Research Laboratory and OSD through a contract with North Carolina Agricultural and Technical State University.

the formation according to the coordinate data from the color tracking program. This situation, where one program depends on incoming data from another program, is a key strength of ROS's message passing architecture. It utilizes a publisher/subscriber relationship between parallel programs to share data. In the example above, once coordinate data is published as a topic in ROS, this topic is subscribed to by the program that needs this information. Not only is ROS a robust message passing platform, it is also open source, allowing collaborative development and sharing of software packages. This minimizes the need to rewrite a software package for common functions that have already been addressed.

The rest of the paper will be organized as follows: The architecture of the system will be highlighted in Section II, with the hardware setup in Section III, followed by a detailed explanation of the software processes in Section IV. Finally, experimental data will be discussed in Section V along with conclusions drawn from this experiment in Section VI.

II. Architecture of the System

The ROS Master is the Central Command Station, it can be executed on any piece of hardware that is running ROS. In our experiment we assigned the ROS Master role to a laptop. Each Agent and the Drone communicates with the ROS Master through Wireless Networks. The ROS Master gets real-time Video from the Drone, identifies and tracks each agent from the video feed using the HSV Algorithm, and calculates the forces between the agents based on the Algorithm explained in further sections. This Force value is then translated into linear and Angular data to the agents, so that they can move in formation. This structure is shown in Fig. 1. The orientation of each agent is also collected real-time to keep track of error in movement, so that further commands can be issued to maintain formation.

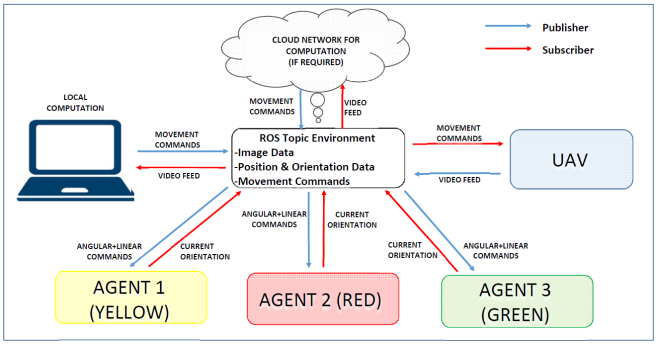

Fig. 1: System Architecture

III. Experimental Setup

A. The Test Space

The Test Space consists of three Kobuki TurtleBot Unmanned Ground Vehicles, each distinctly identified through an assigned color on the top of the TurtleBot, such that they are in visual range of the Parrot Bebop UAV.

(a) Parrot Bebop UAV

(b) Kobuki TurtleBot UGV

Fig. 2: UAV and UGVs used in the experimental test bed.

A Kobuki TurtleBot is a low-cost, open source differential drive robot. It consists of a mobile base, an RGB-D sensor and an Arduino processor making it a perfect entry-level mobile robot platform. The Kobuki was chosen because it is an open source UGV platform, making it perfect for research and development. The Kobuki SDK is based off ROS, which is the preferred development platform for ACE researchers because of its intuitive publisher/subscriber message passing structure that allows robust and simple communication within multiple facets of a robotic system.

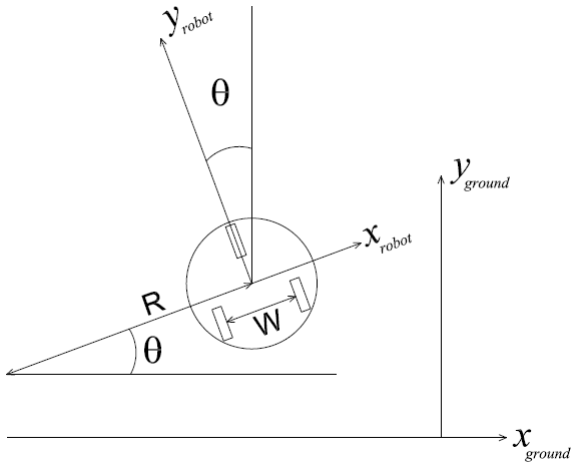

Fig. 3: Representation of the TurtleBot2 UGV in the Coordinate System

The Turtlebot uses a differential drive to steer, and Cook provides a clear approach in order to obtain and simulate the robots movement [10]. The following are the kinematic equations, as shown in the Figure 3, R is the instantaneous radius of curvature of the robot's trajectory, and W stands for the distance between the wheels.

$$v_l = \dot{\theta}(R - \frac{W}{2}), v_r = \dot{\theta}(R + \frac{W}{2}) \quad (1)$$

where:

v_l: velocity of the left wheel

v_r: velocity of the right wheel

$\dot{\theta}$: angular rate

Next, the Angular Rate of the robot is calculated:

$$v_r - v_l = \dot{\theta}(R + \frac{W}{2} - R + \frac{W}{2}) = \dot{\theta}W$$

$$\dot{\theta} = \frac{v_r - v_l}{W} \tag{2}$$

Next the instantaneous radius of curvature is calculated using (1):

$$R = \frac{v_l}{\dot{\theta}} + \frac{W}{2}$$

Using (2) we have:

$$R = \frac{v_l}{\frac{v_r - v_l}{W}} + \frac{W}{2} = \frac{2Wv_l + W(v_r - v_l)}{2(v_r - v_l)}$$

$$R = \frac{W}{2} * \frac{v_r + v_l}{v_r - v_l} \tag{3}$$

Velocity along the robot's longitudinal axis is calulated using (2) and (3):

$$v_y = \dot{\theta}R = \frac{v_r - v_l}{W} * \frac{W}{2} * \frac{v_r + v_l}{v_r - v_l} = \frac{v_r + v_l}{2} \tag{4}$$

Representing the robot's velocity on earth coordinates we have:

$$\dot{x} = -v_y cos(90 - \theta) = -v_y(cos(90)cos(\theta) + sin(90)sin(\theta))$$

$$\dot{y} = v_y cos(\theta)$$

Using equation (4) we have:

$$\dot{x} = -\frac{v_r + v_l}{2} sin(\theta) \tag{5}$$

$$\dot{y} = \frac{v_r + v_l}{2} cos(\theta) \tag{6}$$

For this reason the control variables will be:

$$\dot{v_r} = u_1$$

$$\dot{v_l} = u_2$$

The colors for the UGV identification are distinct colors with a white background with very different HSV windows, so that mis-identification is avoided.

The Parrot Bebop Drone is a lightweight yet robust quad copter with a 1080p "fish-eye" camera and 3-axes image stabilization. This allows the drone to be able to pan its camera in-flight to be able to access the location of the agents at all times. For this test setup in lab, we fixed the drone in a static "hover position" on top of a long pole, in order to maximize the battery life. The ROS-Bebop driver was used in order to obtain data from the Bebop and make it available in ROS.

B. Data from the UGV TurtleBots and the Bebop Drone

a) Bebop Drone: The Bebop drone primarily sends in the video feed from the test space to the ROS master. The HSV algorithm is run on this input video to detect objects within the specified HSV values for red, yellow and green. For the identified objects in the video feed (the mobile agents) the HSV algorithm calculates the coordinates of the geometric center of each object in reference to the Camera Frame as shown in Fig. 4. From the coordinates obtained by the HSV algorithm, we can calculate a Virtual Leader position for the formation. The Virtual Leader is essentially the geometric center of the formation, calculated from the agent's coordinates.

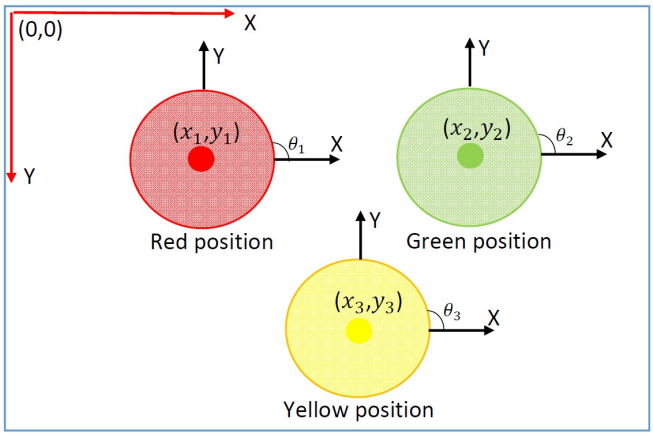

Fig. 4: The Test Space from the UAV's perspective

b) UGV TurtleBots: The UGVs publish their current heading and state to the ROS Master. This data is essential as their current heading is with reference to their own body frame. So, as the formation control algorithm outputs commands to the UGVs with reference to the world frame, the body frame data from the UGVs is required in order to obtain the right output data after the necessary translation and rotation. The ROS topics of interest published by the UGVs are current orientation, current linear velocity, and current angular velocity data as feedback.

C. Color Identification and Tracking

The developed color tracking ROS software package is a ROS compatible version of the open source C++ color tracking software in [11]. This program uses OpenCV to detect the HSV values of objects in an image frame. Each UGV was equipped with a color tag (see Fig. 6) and when viewed by the UAV's camera from above the color tracking software can extract both the color and the (x, y) coordinate of the center point of the tag as per the algorithm shown in Fig. 5. The coordinate data is made available for all other ROS nodes to use. The formation control ROS package is our experimental implementation of the controller presented in [8]. Written in C++, it utilizes incoming UGV coordinate values to calculate a force magnitude that translates into the desired speed and direction of the robot.

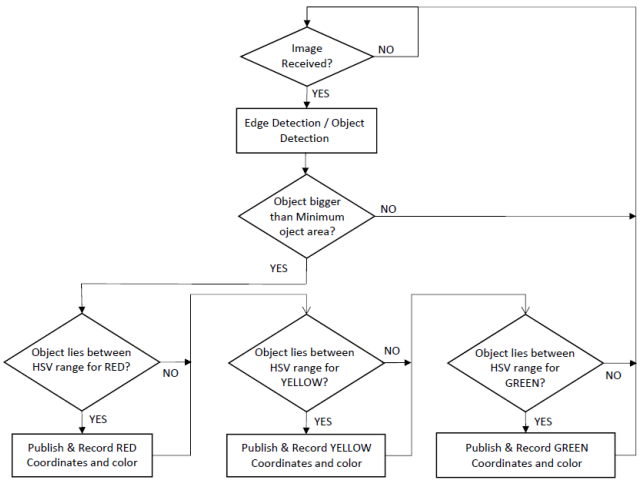

Fig. 5: Color Detection and Tracking

D. Formation Control Algorithm

In this process, potential functions between the agents are used to implement group cohesion and separation. With such a methodology, navigation of the swarm is a process of harnessing the various potentials between the agents. A lot of research in Swarm behavior has been done proposing various formation control and coordination techniques [12]. However, as the algorithms become more complex, computational requirements are increased and the communication demands from each agent become very prohibitive. The implemented algorithm is based off [8] and [13]. The methodology uses energy potential between agents called 'Agent Forces' and between an 'artificial' virtual leader called 'Leader Forces'. This Virtual leader is used in order to advance the group through its environment. The Virtual Leader is not one of the agents or a physical vehicle, but an imaginary point calculated from the agents' positions. It is used as a guide for movement of the group. The goal of using potential energy functions between agents is to repel or attract vehicles from and to each other as the Virtual leader is moved through the environment.The dependence on the Virtual leader motion enables us to plan only the motion of the virtual leader, thereby reducing the task planning requirements.

a) Virtual Leader Forces: Once the Agents are in their initial positions, an initial Virtual leader can be calculated by finding the center of mass for the whole group.

$$x_{cm} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$y_{cm} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

where x_{cm} and y_{cm} are the coordinates of the center of mass, or the geometric center of the formation.

The initial distance between each agent determines the position of the Virtual Leader and the distance between the agents and the Virtual leader d_0^{VL} is the equilibrium distance between each agent and the Virtual leader. The Virtual leader forces F_{VL} can be defined as:

$$F_x^{VL} = K_{VL}[d_x^{VL} - d_{x0}^{VL}]$$
$$F_y^{VL} = K_{VL}[d_y^{VL} - d_{y0}^{VL}]$$
$$d_x^{VL} = x_{VL} - x_i$$
$$d_y^{VL} = y_{VL} - y_i$$

where K_{VL} is the Gain Constant, which can be adjusted as per the required elasticity of the forces between the agents and the virtual leader.

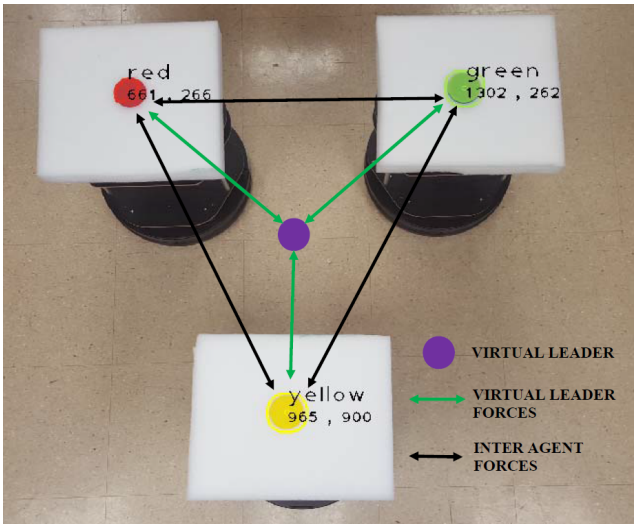

Fig. 6: The Formation forces

b) Inter-Agent Forces: Inter-Agent forces are required so that the agents maintain their equilibrium distance d_0^{ij} from each other as they follow the Virtual Leader, and the formation is maintained. The inter-agent forces F_{ij} can be defined as:

$$F_x^{ij} = K_{ij}[d_x^{ij} - d_{x0}^{ij}]$$
$$F_y^{ij} = K_{ij}[d_y^{ij} - d_{y0}^{ij}]$$
$$d_x^{ij} = x_{ij} - x_i$$
$$d_y^{ij} = y_{ij} - y_i$$

Fig. 6 shows a pictorial representation of the position of the Virtual leader and the various forces acting on the agents.

IV. Output Data to the UGVs

After the algorithm computes the various forces acting on the agents, a single "Sum of all Forces" is sent as output to the UGVs. As the Total force output to the UGV is a vector, both the magnitude and direction needs to be translated for the UGV. Therefore, from the algorithm described in the earlier section, the x-component of the Force and Y-Component of the Force are calculated separately using the X and Y distances separately. Once we know the values of F_x and F_y, we can translate them into commands that can be understood by the UGV, namely "Linear Command" and "Angular command".

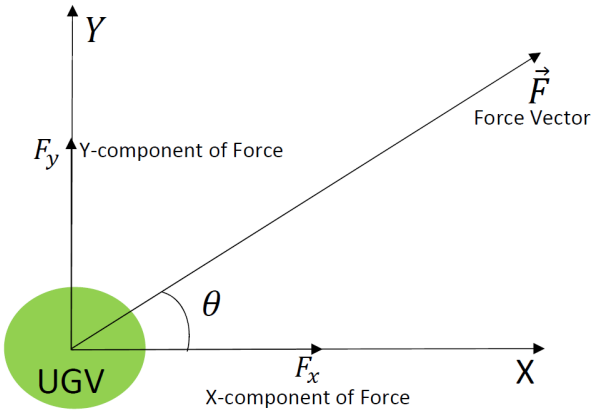

Fig. 7: Force Vector for the UGV

a) Inter-Agent Forces: The Formula for the direction of the forces is:

$$\theta = tan^{-1}\frac{F_y}{F_x}$$

The Magnitude of the force in the direction θ is:

$$|\vec{F}| = \frac{|F_x|}{cos\theta} \quad or \quad |\vec{F}| = \frac{|F_y|}{sin\theta}$$

It is to be noted that the reference frames for the UAV and UGVs are different as can be seen from Fig. 4. That is one aspect of the translation that needs to be carried out. Also, the real-time orientation of the UGV is important to understand which quadrant the angle of the UGV is in, so that the necessary translation can be done as per Fig. 8. Special cases such as when the θ is very close to 0 $\deg$ or 30 deg are taken care of separately, in order to avoid unnecessary rotation of the UGVs.

Also, if the final target θ is smaller than 180 $\deg$, the UGV rotates anti-clockwise towards its target, and if greater than 10 $\deg$, turns clockwise towards the target. This is essentially for optimization of the movement of the UGVs.

Thus, the UGV knows where it is currently located and pointed, and the "Sum of all Forces" from the algorithm is converted to a linear force magnitude and the corrected angle of the force and they are given to the UGV as its linear and angular commands respectively.

V. Experimental Results

The UGV's successfully maintain formation for simple trajectories. However, two major factors come into play in the implementation of the algorithm. Disturbance-free color tracking in varied lighting conditions and the update rate of the commands sent to the UGV. We are working on fine tuning the parameters to minimize the probability of undesirable movements of the UGV when for example, tracking of one UGV is suddenly lost during movement, or when the UGVs attain their formation but the feedback is lost due to update rate factors. Due to such factors, when we attempt to apply a more complex trajectory, the formation sometimes

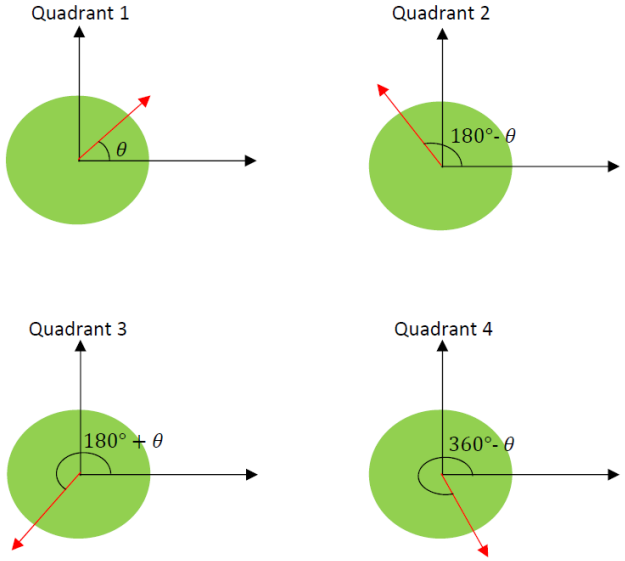

Fig. 8: Angle Translation

exhibits undesirable behavior. Work is ongoing in this front to have more predictable behavior under various circumstances. The following experimental plots show the behavior of the formation control algorithm. In Fig. 9 the Virtual Leader is varied about the image frame, and the Inter-agent (Fig. 10) and Agent-Leader (Fig. 11) Equilibrium Error respond accordingly. The sudden changes seen in Fig. 11 correspond to the variation in the Virtual Leader's position. The plots show that the formation control algorithm is capable of driving the error back close to zero.

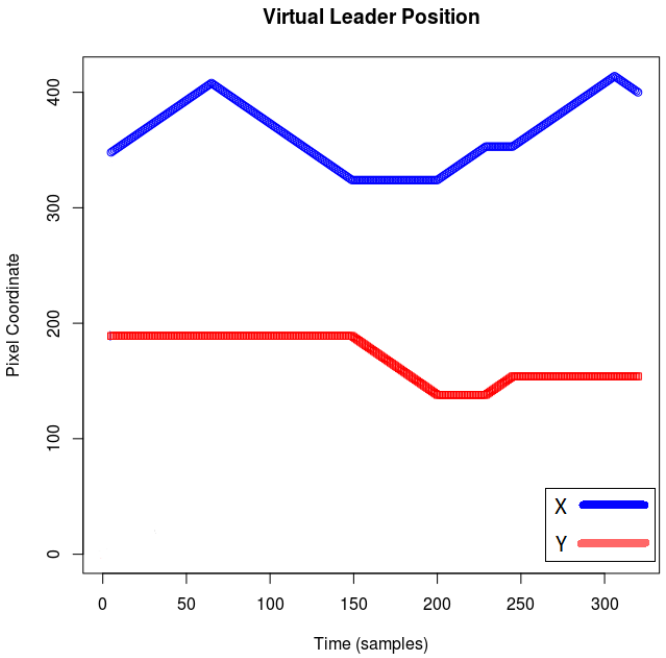

Fig. 9: Virtual Leader Position vs. Time

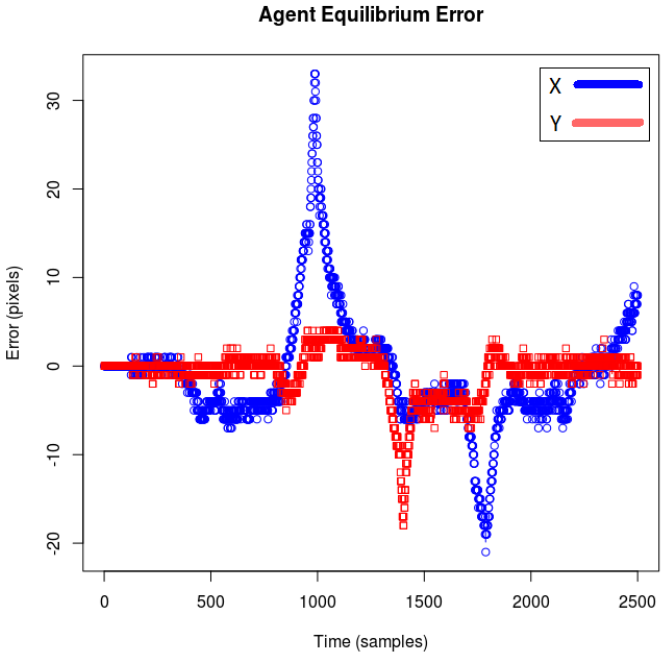

Fig. 10: Inter-agent Equilibrium Error vs. Time

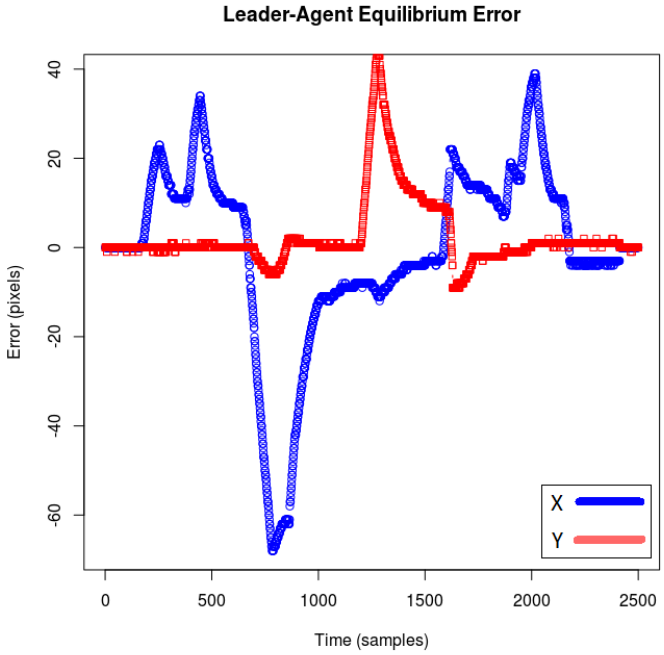

Fig. 11: Leader-Agent Equilibrium Error vs. Time

VI. Conclusion

Formation control can be accomplished using a variety of methods, with each one having pros and cons compared to the other. In our case, the controller we decided to implement was both straightforward theoretically and programming wise and was initially advertised as computationally lightweight. From the results obtained in Section V, it is reasonable to say that the proposed controller is in fact implementable and can be used to control a formation of N number of agents in the system. Future planned work involves moving the decision making process to the cloud so that a large number of agents can successfully hold formation. Better methods of detection and tracking of the agents are currently being researched as well.

References

[1] J. A. Sauter, R. S. Mathews, A. Yinger, J. S. Robinson, J. Moody, and S. Riddle, "Distributed pheromone-based swarming control of unmanned air and ground vehicles for rsta," in *SPIE defense and security symposium*. International Society for Optics and Photonics, 2008, pp. 69 620C–69 620C.

[2] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.

[3] M. Pfingsthorn, B. Slamet, and A. Visser, "A scalable hybrid multi-robot slam method for highly detailed maps," in *RoboCup 2007: Robot Soccer World Cup XI*. Springer, 2008, pp. 457–464.

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

[5] A. Okubo, "Dynamical aspects of animal grouping: swarms, schools, flocks, and herds," *Advances in biophysics*, vol. 22, pp. 1–94, 1986.

[6] K. Warburton and J. Lazarus, "Tendency-distance models of social cohesion in animal groups," *Journal of Theoretical Biology*, vol. 150, no. 4, pp. 473–488, 1991.

[7] G. Flierl, D. Grünbaum, S. Levins, and D. Olson, "From individuals to aggregations: the interplay between behavior and physics," *Journal of Theoretical Biology*, vol. 196, no. 4, pp. 397–454, 1999.

[8] G. H. Elkaim and R. J. Kelbley, "A lightweight formation control methodology for a swarm of non-holonomic vehicles," in *Aerospace Conference, 2006 IEEE*. IEEE, 2006, pp. 8–pp.

[9] A. R. Conway, "Autonomous control of an unstable model helicopter using carrier phase gps only," Ph.D. dissertation, stanford university, 1995.

[10] G. Cook, *Mobile robots: navigation, control and remote sensing*. John Wiley & Sons, 2011.

[11] A. Kaifi, H. Althobaiti, and Z. Rentiya, "akaifi/multiobjecttrackingbasedoncolor." [Online]. Available: https://github.com/akaifi/MultiObjectTrackingBasedOnColor

[12] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, "Distributed, physics-based control of swarms of vehicles," *Autonomous Robots*, vol. 17, no. 2-3, pp. 137–162, 2004.

[13] G. H. Elkaim and R. J. Kelbley, "Extension of a lightweight formation control methodology to groups of autonomous vehicles," in *Proc. ISAIRAS*, 2005, pp. 5–9.