Abstract:

The Caltech Multi-Vehicle Wireless Testbed (MVWT) is a platform designed for experiments involving cooperative control. This report reviews the platform design and summarizes results from implementation of three different algorithms on the testbed. Two different types of vehicles were used. The first kind (Steelebot) has wheels and is a first order control vehicle. The second kind (Kelly) is a second order control vehicle that uses fans to propel itself. The algorithms implemented include (1) 'greedy' cooperative searching through a list of spatial targets (Steelebot), (2) boundary tracking using a method that is built upon the ability to sense a substance and to make small circular motions in both directions (Kelly), and (3) motion towards and away from objects using virtual potentials (Kelly).
1. Summary and Overview
2. Caltech MVWT
   2.1 Caltech MVWT Overview
      2.1.1 Master Control Architecture
      2.1.2 RoboFlag Architecture
   2.2 Vehicles
      2.2.1 Steelebots
      2.2.2 Kellys (a.k.a. MVWT I)
      2.2.3 Bats (a.k.a. MVWT II)
   2.3 Lab Positioning System
   2.4 Software
3. Greedy Search Algorithm
   3.1 General Description
   3.2 Algorithm
   3.3 Implementation
   3.4 Methods and Results
      3.4.1 One Vehicle, Three Waypoints
      3.4.2 Two Vehicles, Two Waypoints, Unlimited Communication
      3.4.3 Two Vehicles, Two Waypoints, Limited Communication
      3.4.4 Two Vehicles, Three Waypoints, Limited Communication
   3.5 Observations and Conclusions
4. Boundary Tracking
   4.1 General Description
   4.2 Proposed Method
      4.2.1 Single Vehicle Boundary Tracking
   4.3 Data
5. Swarming Mechanism
   5.1 General Description
   5.2 Motion Equation Transformation
   5.3 Experimentations and Analysis
      5.3.1 Achieving Target Point
      5.3.2 Point-to-Point
      5.3.3 Pole Avoidance
      5.3.4 Orbiting
      5.3.5 Follow the Leader
6. Conclusion

Acknowledgments
Appendix
References
1. SUMMARY and OVERVIEW

This report describes a joint project between UCLA and CalTech during summer 2004. The purpose of the project was to implement algorithms for distributed control of multiple autonomous vehicles on the CalTech Mult-Vehicle Wireless Testbed. Three UCLA students, C. Hsieh, B. Nguyen and D. Tung, spent summer 2004 working with CalTech to first further develop the testbed and then to test novel distributed algorithms for three different tasks: (1) cooperative searching through a target list, (2) cooperative boundary tracking, and (3) cooperative interactions using virtual potentials.

CalTech graduate students Zhipu Jin and Ling Shi supervised respectively the vehicle team and the command control team. Jin's algorithm for circle path tracking was incorporated as a sub-controller into the boundary tracking implementation. Duke Physics student Yao-Li Chuang formulated a method to port attractive and repulsive virtual forces for particles to the vehicular controls of one of the vehicles (Kelley) on the testbed. Postdoc Dan Marthaler worked closely with Hsieh on boundary tracking. The control algorithms were developed in Andrea Bertozzi’s research group (includes all UCLA participants).

2. MVWT

2.1 Caltech MVWT Overview

The Caltech Multi-Vehicle Wireless Testbed (MVWT) is a platform designed for experiments involving cooperative multi-vehicle control. The MVWT consists of vehicles that can communicate over a wireless network, an arena for the vehicles to run in, an overhead camera system called the Lab Positioning System (LPS), and an offboard computer network. Each vehicle in the MVWT has an onboard computer, onboard sensors, and an 802.11b wireless Ethernet card [4].

An overhead view of the Caltech MVWT arena is shown in Figure 1 from [4]. There is a smooth floor with dimensions of approximately 6.7 m x 7.3 m where the vehicles run. There is a post in the middle of the field that had been used for some experiments. Video cameras mounted on the ceiling take pictures of the field at 60 Hz and are connected to a computer that processes the vision data. The vehicles that run in the field have symbols on top of them that the vision system uses to identify each vehicle, its position, and its orientation.
Figure 1. A schematic of the Caltech MVWT arena [4]. It consists of a 6.7 m x 7.3 m field, center post (C), vision system (A), and vehicles (B) that run in the field.

In the past, two computer network architectures, the Master Control architecture and the RoboFlag architecture, have been used to run vehicles on the MVWT.

2.1.1 Master Control Architecture. The Master Control provides a user interface for sending high level commands to a particular type of robot called the Kellys [1] and can be used for recording data. A flowchart of the Master Control architecture is shown in Figure 2 from [4]. The Kellys receive signals from the Master Control, data from the vision system, and information from their onboard sensors, such as the gyro, in order to control their dynamics. The Master Control computer runs a QNX operating system, a real time operating system similar to Linux. This entire architecture was used for the algorithms in sections 4 and 5.
2.1.2 RoboFlag Architecture. The Caltech MVWT has a computer network set up to play RoboFlag [8]. RoboFlag is a robotic capture the flag game with two teams of players and each team controlling a group of robots. The field is divided into two zones, with one zone belonging to each team. Each team tries to capture the other team’s flag by moving their robot into the other team’s goal and bringing it back into their own goal without getting tagged. This game employs varying levels of autonomous play since participants are able to program their own game strategies.

The RoboFlag architecture is shown in Figure 3 [8]. Each team plays on networked computers with GUI interfaces. They use these computers to execute their autonomous strategies and to move their robots from point to point. The vehicle commands are sent from the team networks to an arbiter. The arbiter acts as a referee to filter the commands coming from the teams. The MVServer acts as a communication interface between the vehicles in the arena, the arbiter, and the vision system. It sends vehicle commands from the arbiter to the vehicles in the field. It also receives vehicle position data from the camera system and sends it to the arbiter, which sends it to the team networks. The MVServer runs on Linux, while the arbiter and computer network for each team run Windows XP.

The algorithm in section 3 is implemented using the MVServer, vision system, and hardware test bed components from the RoboFlag architecture shown in Figure 3. The arbiter and everything above it is replaced by a program that interfaces with the MVServer.
2.2 Vehicles

The MVWT currently has three vehicle types, the Steelebots, Kellys, and Bats. The Steelebot is driven by wheels and is capable of making sharp turns. It is modeled by first order dynamics. The Kelly accelerates by fan forces and moves in smoother paths (i.e. aircraft-like vehicle); this is an example of second order dynamics. The next generation of the Kelly is the Bat, which has hovercraft features.

2.2.1 Steelebots. The Steelebot, see Figure 4, was produced at the University of West England [12]. It was originally known as the Moorebot since it was located in the Moore building, and is now known as the Steelebot since it is located in the Steele building. It has a two wheel differential drive system with an independent actuator for each wheel. This makes it useful for experiments that require a vehicle with the ability to make sharp turns and to brake.

The Steelebot is based on two main systems. The first is a tiny Amporo PC that runs the Linux operating system. The second is a large motherboard which provides power for operation and communication among the various devices on the robot [11]. The Steelebot has an 802.11b wireless Ethernet card that it uses to communicate on the MVWT wireless network. It also has binary proximity sensors that are based on Infrared (IR) receivers and standard IR LED transmitters [11], which could be used to detect an object in its path.

There is a library made for the Steelebot that allows programs written in the C programming language to control the robot. It contains routines that allow an application programmer to do things such as set the speed of each motor and check the value of inputs [11].
2.2.2 Kellys. Another type of vehicles used at MVWT is the Kelly (figure 5), which is equipped with an onboard 700 MHz Dell lightweight laptop, local sensors, and the ability to communicate with others and central command via Local Wireless Network [1]. Each Kelly sits on three omni-directional casters and drives by two ducted fans. The Kelly is not capable of making sharp turns like the Steelbot. The Kelly’s smooth turning behavior is described by second order dynamics. More descriptions on the dynamics and controllers of the Kelly can be found in [1,4].

Figure 5. The Kelly fundamentally consists of 700 MHz laptop, two ducted fans, 3 casters, gyro, hat, and a body frame. Additional sensors can also be customized.
2.2.3 Bats. The Bat [10] is intended to continue the study of second order vehicle characteristic when the MVWT is shifted to a larger arena located on the rooftop of the Steele building at Caltech. This new arena does not have as smooth a surface as the current testbed, requiring a true hovercraft design. Future studies in large scale multi-vehicle required the Bat to be smaller then the Kelly.

2.2.3.1 Bat I. The initial Bat design is shown in figure 6a. It has a compact design, low weight, and is easy to build. The lift fan is off center, and the bulk of the weight is from a Zaurus handheld computer (www.sharpusa.com) in the rear of the Bat. The none-symmetrical construction causes the Bat to tip to one side, making contact with the arena surface. The current test bed arena is smooth and thus does not prevent the motion of the Bat I. However, with the expansion of the MVWT to the rooftop venue, where “Berber Max” foam matting will be used as the surface, ground contact with the mat would cause too much drag for the push fan to overcome. Therefore a new design was created.

2.2.3.2 Bat II.

Figure 6. a) Original Bat design [10], b) New Bat design

UCLA student Chung Hsieh contributed to the new Bat design. The changes retain the positive features of compactness, low weight and ease of build seen in the first Bat design. The new Bat incorporates various commercial and model hovercraft designs 1,2,3. It features the Zaurus mounted above a centralized lift fan (Figure 7). Mounting the Zaurus above the fan centralizes the weight and helps with the stability of the platform and the lift force distribution. The high placement of the Taurus proves stable during testing. The various battery packs are use as ballasts to further shift the center of gravity to the center of the base plate.

---

1 www.gohover.com
2 www.model-hovercraft.com
3 www.hoverdril.com
Figure 7. Side view of the Bat

The result is a Bat capable of floating over rough surfaces such as Berber Max mat and short hair carpeting.

2.2.3.3 Specifications.
Table 1 is the specifications of both Bat versions.

<table>
<thead>
<tr>
<th>Vehicle Parameters</th>
<th>Bat II</th>
<th>Bat I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>22(W)*20(L)*17.5(H) cm</td>
<td>22(W)*20(L)*7.5(H) cm</td>
</tr>
<tr>
<td>Mass</td>
<td>1.03 +/- 0.05 kg</td>
<td>0.75 +/- 0.05 kg</td>
</tr>
<tr>
<td>Moment of Inertia</td>
<td>0.0043 +/- 0.0015 kg m^2</td>
<td>0.0031 kg m^2</td>
</tr>
<tr>
<td>Max Lift Force</td>
<td>1.5 N</td>
<td>0.7 N</td>
</tr>
<tr>
<td>Linear Friction</td>
<td>TBD</td>
<td>&lt;0.001 kg/s (max thrust)</td>
</tr>
<tr>
<td>Rotational Friction</td>
<td>TBD</td>
<td>&lt;0.0003 kg/s (max thrust)</td>
</tr>
<tr>
<td>Max Speed</td>
<td>&gt; 2.5 m/s</td>
<td>&gt; 2.5 m/s</td>
</tr>
<tr>
<td>Computation unit</td>
<td>Sharp Zaurus SL5500</td>
<td>Sharp Zaurus SL5500</td>
</tr>
<tr>
<td>Cost per vehicle</td>
<td>&lt; $800</td>
<td>&lt; $800</td>
</tr>
<tr>
<td>Battery lifetime</td>
<td>TBD</td>
<td>35-40 minutes</td>
</tr>
</tbody>
</table>

Table 1. Comparison of the two Bat versions.

2.2.4 Comparisons of vehicles at MVWT.

<table>
<thead>
<tr>
<th></th>
<th>Steelebot</th>
<th>Kellys</th>
<th>Bats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>37(W)*27(L)*18(H) cm</td>
<td>22(W)*20(L)*17.5(H) cm</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>5.05 +/- 0.05 kg</td>
<td>1.03 +/- 0.05 kg</td>
<td></td>
</tr>
<tr>
<td>Microprocessors</td>
<td>25Mhz 386</td>
<td>700Mhz Pentium III</td>
<td>204Mhz StrongARM</td>
</tr>
<tr>
<td>Communication</td>
<td>802.11b</td>
<td>802.11b, USB, Serial</td>
<td>802.11b, Serial</td>
</tr>
<tr>
<td>Payload Capacities</td>
<td>&gt; 2kg</td>
<td>100 grams</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Comparison of all vehicles
2.3 Lab Positioning System

The Lab Positioning System is comprised of a vision system and accompanying computer hardware. The vision system consists of a computer (Pentium III 933 MHz with an Intel motherboard), four Pulnix TM-6710 cameras (1/2" monochrome CCD cameras, 648 x 484 pixels of resolution, mounted to the ceiling and looking straight down), and four Matrox Genesis vision processing boards in the computer’s PCI slots. In short, the Lab Positioning System provides positions of vehicles to whichever devices require them. Detailed information on hardware and software of vision system can be found in [http://www cds. Caltech.edu/~mvwt/meltzer.html].

2.4 Software

The MVWT Application Programmer Interface (API), shown in Figure 8, was derived from the RoboFlag architecture and developed for users to control robots on the MVWT testbed [8]. The MVServer is an interface from RoboFlag between offboard computers, the vision system, and robots in the field that is also useful for developing non-RoboFlag applications (see 2.1.2). The MVWT API was developed to allow users to develop application programs for the MVWT that interface with the MVServer. It contains functions from the RoboFlag arbiter code that handle communication with MVServer. It also contains a point to point control function that has been tested to work smoothly and accurately on the Steelebots. Programmers would write custom code that interfaces with the API functions, forming the Client. The MVWT API has been tested to work on computers running Windows XP.
Figure 8. The MVWT API architecture [8].

The Master Control architecture and the laptops on the Kellys run QNX and RHexLib. RHexLib is a collection of software libraries that is used with computers running QNX. It is designed to help programmers develop applications to control robots.

3. GREEDY SEARCH ALGORITHM

3.1 General Description

The underwater mine countermeasure (MCM) problem involves locating, visiting, and identifying mines from a list of possible target sites. Because this task is very hazardous to the diver-dolphin teams that currently accomplish it, the use of autonomous vehicles to solve the MCM problem is an objective of the US Navy.

The Fractional Bandwidth approach to the MCM problem is a decentralized algorithm for a set of multi-agent vehicles to go to a set of possible target locations. Each vehicle is capable of accomplishing the entire mission, which makes this approach robust against vehicle loss and communication failure. Inter-vehicle communication prevents them from going to the same target sites, making communication a performance accelerator [7].

The Fractional Bandwidth approach is implemented on the Caltech MVWT testbed using a greedy search algorithm [7]. In this algorithm, each vehicle keeps track of targets as “available”, “selected”, or “done”, with all targets being initially “available”. Each vehicle also keeps track of its “next target”, which its current target destination. The vehicles have a set communication
range, which is the maximum vehicle to vehicle distance allowed for communication between vehicles.

The following is a description of one iteration of the greedy search algorithm loop. This loop is continued until all targets are marked as “done”. Each vehicle selects the nearest “available” target as its “next target”. If there are no “available” targets, a vehicle selects the nearest “selected” target as its “next target”. The vehicles then communicate their position information and “next target” selection with other vehicles within communication range. If they share the same “next target” selection, the vehicle that is farther away from the target marks this target as “selected” so that it would be able to select another “available” target. If they do not share the same “next target” selection, in the case that the received “next target” selection is “available”, it is upgraded to “selected”. Vehicles within communication range share their “done” target lists and accordingly upgrade “available” and “selected” targets to “done”. Each vehicle travels to its “next target” selection waypoint. If a vehicle is within the search radius of the target, the vehicle marks the target as “done”.

### 3.2 Algorithm

The Greedy algorithm was implemented using the following structure:

**Initialization:** Each vehicle has the target locations and corresponding target flag of “available”, “selected”, and “done”. All targets are initially “available”. Each vehicle also has a “next target” variable for the target it is traveling to.

**Program Loop:**

1. **Select closest target:** Each vehicle selects the nearest “available” target. If no target is “available”, the vehicle selects the nearest “selected” target. The selected target index is stored as the “next target.” If no “available” and “selected” targets are left, the mission is successful and ends.

2. **Communicate “next target” selection:** Each vehicle finds the next target and position of every other vehicle within communication range.
   
   a) **Arbitrate “next target” selection:** If two vehicles share the same “next target” selection, the vehicle that is further away marks this target as “selected” so that it would be able to select another target that is “available” at step 2 of the next iteration.
   
   b) **Communicate “next target” selection:** If two vehicles do not share the same “next target” selection and the current status of the received “next target” selection is “available”, it is upgraded to “selected”.

3. **Communicate “done” target list:** Each vehicle within communication range share their target lists for all targets they have as “done”. If the receiving vehicle’s target status is “available” or “selected” it is upgraded to “done”.

4. **Travel to the “next target” selection waypoint:** This is done for each vehicle using a point to point controller.

5. **Mark located target as “done”**: For each vehicle, if the vehicle is within the search radius of the target, it is marked off as “done.”

6. Return to step 2 until all targets are “done”.

3.3 Implementation

The Steelebots were selected for the greedy algorithm because their first order dynamics made them the easiest vehicle on which to implement a point to point controller. The algorithm was implemented using the MVWT API, seen in Figure 8. The inter-vehicle communication was simulated in the Client by storing and communicating the target information for each vehicle in the field. The Client was also used to limit the communication range between vehicles. The MVWT API provided the point to point controller for the Steelebots.

The experiments were carried out using either one or two Steelebots.

3.4 Methods and Results

To conclude that the algorithm was successfully implemented, the following three experiments were run:

1. One vehicle, multiple waypoints.
2. Multiple vehicles, multiple waypoints, unlimited communication.
3. Multiple vehicles, multiple waypoints, limited communication.

Experiments were conducted to verify that the algorithm was producing expected results. Each experiment was first run on a numerical simulation, and then its results were compared with those from the testbed.

The experiments in Figures 8, 9, 10, and 11 all have the following characteristics. The coordinate system represents the field of the testbed with the x and y axes indexed in meters. The circles represent the starting points of the vehicles, and the x’s represent the vehicle targets. The dashed lines represent the idealized data logged from the simulation, while the other lines represent the position data logged from the vehicles in the live runs.

3.4.1 One Vehicle, Three Waypoints. Figure 9: The vehicle consistently went to the next closest available target in both the simulation and in the testbed implementation and ended its mission when all three targets were found. Notice that in the live run, the vehicle goes through the first and second targets, and not the third target. This is because the live run used a threshold distance of 0.2 m between the target and the vehicle to consider a target “done”, while the simulation used only 0.02 m for this threshold.
3.4.2 Two Vehicles, Two Waypoints, Unlimited Communication Range. Figure 10(a): With unlimited communication range, the vehicles went to their nearest available target. Once they got to their targets, they communicated their target lists and stopped moving since all the targets were found.

Figure 10(b): Even though both vehicles are initially closer to the same target, they successfully arbitrated their “next target” selection with unlimited communication range. The one closer to this target went to it while the other vehicle immediately goes to the other target.

In both live runs, the vehicles do not cover the target locations. This is because the live runs used a threshold distance of 0.2 m between the target and the vehicle to consider a target “done”, while the simulation used only 0.02 m for this threshold.
3.4.3 Two Vehicles, Two Waypoints, Limited Communication Range. Figure 11(a): With communication limited to one meter, the each vehicle went to their nearest available target, and then tried to get to the other target. They had to get within communication range in order for them to share target lists, account for all the targets, and complete the mission.

Figure 11(b): Communication was limited to one meter. Both robots started off traveling to their nearest target, and they arbitrated their “next target” once they got within communication range. When this happened, the vehicle that was further away traveled to the other target.

Notice that in both live runs, the vehicles do not cover the target locations. This is because the live runs used a threshold distance of 0.2 m between the target and the vehicle to consider a target “done”, while the simulation used only 0.02 m for this threshold.

3.4.4 Two Vehicles, Three Waypoints, Limited Communication Range. Figure 12: With communication limited to one meter, the vehicles start off 0.75 m from each other. They went to their closest targets, while communicating their “next target” selection and target lists. As a result of their communication, they then go to the target at (0.5, -2.5) and finish when one of the robots reaches this target. In the live run, the final target location is not covered by the vehicle. This is because the live run used a threshold distance of 0.2 m between the target and the vehicle to consider a target “done”, while the simulation used only 0.02 m for this threshold.
3.5 Observations and Conclusions

The live run data demonstrates the algorithm performing well in a physical environment. This validates the greedy search algorithm’s structure and effectiveness.

The fact that the simulation data and the live run data match up very closely can be attributed to the accurate performance of the point to point controller from the API that is used on vehicles with first order dynamics (Steelebots).

The live runs used a threshold distance of 0.2 m between the target and the vehicle to consider a target “done”, while the simulation used only 0.02 m for this threshold. This accounts for the small disparities between the vehicle paths and the target locations. Future work on this problem could involve larger numbers of vehicles, second order dynamics, and anti-collision features between vehicles.

4. BOUNDARY TRACKING

4.1 General Description

The purpose is to be able to find and track the boundary of two differing materials, states or characteristics, using autonomous vehicles equipped with sensors that can measure the material. Examples include finding and tracking oil spills or marine organisms. In the original algorithm proposed by Kemp, Bertozzi and Marthaler [9], the vehicle turns clockwise when inside the region of interest and turns counterclockwise when outside the region. Following this pattern, the vehicle will continuously cross the boundary as it tracks the boundary. This type of tracking is suited for vehicles with local sensors capable of binary detection of region/objects of interest.
The algorithm is described as:

\[
\frac{d\theta}{dt} = \begin{cases} 
+\omega & \text{Outside}, \\
-\omega & \text{Inside},
\end{cases}
\]

\(V_{\text{ref}} = \text{Constant.}\) (Eq. 1)

Here the vehicle speed \((V_{\text{ref}})\) is constant, \(\theta\) is the heading and \(\omega\) is the angular velocity. The algorithm was shown to be stable and convergent [9]. However, depending on the contour of the boundary, total coverage may not be possible (Figure 14).

To ensure convergence and maximum coverage the radius of curvature formed by the UV \((R_{\text{ref}})\) must be less than the minimum radius of curvature of the contour to be tracked \((R_{\text{bound}})\) [9].

### 4.2 Implementation Method

We implemented the algorithm using virtual sensors and real position and control. A virtual boundary formed by a circle centered at \((3, 3.5)\) with radius \(R_{\text{bound}}\) of 1.7 meters is created on the Caltech MVWT testbed. When the position of the vehicle is within \(R_{\text{bound}}\) the vehicle is inside the region and will follow the rules dictated by (Eq 1).

The Kellys are used in the experiment, because they are second order dynamic vehicles therefore much closer in behavior to the underwater vehicles the algorithm was originally designed for. The Kelly also has a mature interface and a circle path following controller (App. A) on which to build this algorithm.
The controller is modified in two ways to suit our needs: 1) to allow dynamic change in the direction of rotation (DoR) of the vehicle. The original controller only rotates in counterclockwise direction, 2) to accept dynamic change to the center of rotation (CoR) of the circle that the vehicle follows.

The testing is done in two phases, first phase demonstrates that the vehicle is capable of searching out the boundary by tracking circles and following the algorithm described in [9]. The second phase is to incorporate additional vehicles, and provide an algorithm to space the vehicles along the boundary.

4.2.1 Single Vehicle Boundary Tracking. The algorithm is modified slightly to deal with the constraint of the test bed as well as the second order dynamics of the vehicles. In keeping the idea of (Eq. 1). The vehicle changes its direction of rotation as dictated by the algorithm but also shifts the CoR to ensure the vehicle continues to track the boundary. The vehicle is aware of its CoR and the coordinate of the boundary crossing point. A new CoR is projected along the vector Z (Eq. 2b) using the current CoR and the crossing point (Figure 15)

Figure 15. a) The original circle path. b) c) d) Crossing made, with new center of rotation projected each time and vehicle moving along the boundary.

Single vehicle boundary searching result

Figure 16 is a plot of actual data recorded from the Kelly on the MVWT testbed. The vehicle starts at the triangle in figure 16 and starts in a counterclockwise (+ω) direction. Upon crossing the boundary, the vehicle switches direction of rotation to clockwise (-ω) and tracks the new circle with CoR described by Eq. 3a and 3b. The jagged edges of the vehicle path are caused by the change in DoR and the shift of CoR. The edges are formed as the vehicle turns to follow the new CoR and DoR but continues to drift in the original path it was following, this is an effect of the second order dynamics of the Kelly. Once the vehicle orientates itself, it follows the circular path until crossing the boundary again.
4.2.2 Multi-Vehicle Boundary Tracking. After implementing the single vehicle boundary tracking, other vehicles were added, requiring local rules to space themselves apart. There are two main methods to change vehicle speed. 1) Keep constant speed and have the vehicle travel different size circles by varying $R_{\text{ref}}$. 2) Keep constant radius and vary the speed of the vehicle ($V_{\text{ref}}$). It is more practical to vary the vehicle speed because varying the radius of the circle is difficult for the vehicle to track quickly.

The speed of the vehicle varies based on the distance between the vehicle’s CoR. The vehicles communicate with other vehicles every 50ms, broadcasting the vehicle’s own CoR to other vehicles. Each vehicle has a designated leader and follower. The distance to the leader and follower is what is used to determine the speed. The vehicle tries to achieve equal distance between its CoR with tolerance of $2R_{\text{ref}}$ (Eq. 4).

\[
\begin{align*}
\text{d}_1 &= \text{Distance from current CoR to leader CoR} \\
\text{d}_2 &= \text{Distance from current CoR to follower CoR} \\
D &= \text{d}_1 - \text{d}_2 \\
V_{\text{ref}} &= \begin{cases} 
0.10\text{ms} & D < -2R_{\text{ref}} \\
0.15\text{ms} & -2R_{\text{ref}} \leq D \leq 2R_{\text{ref}} \\
0.20\text{ms} & D > 2R_{\text{ref}} 
\end{cases} 
\end{align*}
\] (Eq. 4)

The distance between CoRs is chosen over actual vehicle position as an indicator for vehicle separation.

Multiple vehicles boundary searching results
Figure 17. Three Kellys are various time steps as they explore the boundary, and space themselves out.

Figure 17 shows the vehicles at 1 second, 15 seconds, 45 seconds and 90 seconds respectively. As figure 17 progresses the vehicles start to separate. In figure 17b, the last vehicle has not started moving because its CoR is still within $R_{\text{ref}}$ of the vehicle in front, that is a safeguard programmed in to help prevent collision. Once the square vehicle’s CoR changes then the last vehicle will start moving. By figure 17d, the vehicles have spaced their CoR apart. An initial start sequence was required because the safeguard mention above prevents the system from starting. Since the front vehicle’s leader is the tail vehicle and initially they are at the same location, they are all within $R_{\text{ref}}$ of each other thus the vehicles will not move. To resolve this problem the front vehicle was given an initial ghost leader that crosses the boundary twice.
5. VIRTUAL ATTRACTIVE REPULSIVE POTENTIAL (VARP)

5.1 General Description

Swarming behavior is often observed in bird flocks, fish schools, and other organisms; coordinated large scale group formation and motion is achieved through decentralized local interactions within the group. A number of models exist in the literature and a subset of these is built around virtual interacting potentials between organisms. Couzin's model [5] is based on first order motion of organisms. A related model due to Levine et al [3] involves second order motion of particles. Both models exhibit collective motion including mills and flocks. We wish to take these ideas and design an algorithm for the Kellys on the MVWT. The Levine model is particularly appropriate for adoption to the second order Kelly vehicle. The governing equations for Levine's model are

\[ m_i \ddot{x}_i = \alpha \dot{x}_i - \beta \dot{v}_i - \vec{V} U, \]  

\[ \ddot{v}_i = \dot{v}_i. \]  

Where the total potential \( U \) is a summation of pair-wise potentials,

\[ U = \sum_{j \neq i} C_a \exp\left(-\frac{|x_i - x_j|}{l_a}\right) - \sum_{j \neq i} C_r \exp\left(-\frac{|x_i - x_j|}{l_r}\right), \]  

The terms \( \alpha \) and \( \beta \) define the magnitude of the self-propulsion and self-deceleration forces, respectively; \( C_a \) and \( C_r \) are the strength of the attractive and repulsive potentials, respectively; \( l_a \) and \( l_r \) are the attractive and repulsive characteristic lengths, respectively; and \( m_i, x_i, v_i \) are the mass, position, and velocity of the i-th particle, respectively.

Our goal is to design a control for the Kellys based on Levine model (Eq. 5, 6).

5.2 Motion Equations Transformation

Motion of a Kelly is modeled by the following set of equations [1]:

\[ \frac{du}{dt} = m \frac{d^2 x}{dt^2} = -\mu u + (F_R + F_L) \cos \theta \]  

\[ \frac{dv}{dt} = m \frac{d^2 y}{dt^2} = -\mu v + (F_R + F_L) \sin \theta \]  

\[ J \frac{d\Omega}{dt} = J \frac{d^2 \theta}{dt^2} = -\psi \Omega + (F_R - F_L) r_f, \]  

where
- \( m \) is the mass of the vehicle, \( m = 5.05 \) kg;
- \( x, y \) are the position coordinates of the vehicle;
- \( u, v \) are the component of the linear velocity of the vehicle (\( u = dx/dt, v = dy/dt \));
- \( \theta \) is the orientation of the vehicle, relative to the x-y coordinates (figure 17);
• $\Omega$ is the angular velocity relative to the x-y coordinates ($\Omega = d\theta/dt$);
• $\mu, \psi$ are linear and angular friction coefficients;
• $F_R, F_L$ are the output forces of the right and left fans;
• $r_f$ is the half-distance between left and right fans.

Now, we derive transformation that takes Levine’s model to the Kelly’s motion (Eq. 7).

Plugging Eq. 6 into Eq. 5, we have

$$m \ddot{w}_i = m \dddot{z}_i = \alpha \xi_i - \beta \dot{w}_i$$

$$-\nabla \varepsilon_i \sum_{j(j \neq i)} \left[ -C_\alpha e^{-\frac{||\varepsilon_i - \varepsilon_j||}{l_a}} + C_r e^{-\frac{||\varepsilon_i - \varepsilon_j||}{l_r}} \right],$$

where

• $\varepsilon_i, \dot{\varepsilon}_i$ represent the position and velocity of the i-th swarmer;
• $\alpha$ is the magnitude of the self propulsion force;
• $\beta$ is the friction coefficient;
• $C_\alpha, C_r$ define the strength of the attractive and repulsive potentials, which decay exponentially according to the characteristic lengths $l_a$ and $l_r$;

$\xi_i$ is the unit vector parallel to the direction of i-th vehicle’s velocity, which defines an alternative coordinate system other than the x-y coordinates. These two different coordinate sets are shown in (fig. 18). x-y coordinates are fixed on the ground, while $\zeta$-$\xi$ coordinates are fixed on the vehicle. Note that the $\zeta$-$\xi$ system is obtained if the x-y system is rotated by an angle $\theta$.

---

Figure 18. Two Cartesian coordinate systems describing the position and velocity of a vehicle. x-y coordinates are fixed on the ground and independent of the vehicle status and $\zeta$-$\xi$ coordinates are attached to the vehicle.
The $\xi-\zeta$ system is obtained if the x-y system is rotated by an angle $\theta$, and they have the following relationship:

$$\hat{x} = \cos \theta_i \xi_i - \sin \theta_i \zeta_i; \quad \text{Eq. 9a}$$

$$\hat{y} = \sin \theta_i \xi_i + \cos \theta_i \zeta_i. \quad \text{Eq. 9b}$$

Equation 8 can also be expressed by writing the x-y components explicitly:

$$m \frac{d^2 \hat{v}_i}{dt^2} = m \frac{d^2 \hat{z}_i}{dt^2} = \alpha \zeta_i - \beta \hat{v}_i$$

$$- \sum_{j (j \neq i)} \left( C_a \frac{\| \xi_j - \xi_i \|}{l_a} - C_r \frac{\| \xi_j - \xi_i \|}{l_r} \right) \left( (x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i \right) \hat{\xi}$$

$$\frac{(x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i}{\| \xi_i - \xi_j \|} \hat{\zeta} \quad \text{Eq. 10}$$

Substituting (Eq. 9ab) into (Eq. 10),

$$m \frac{d^2 \hat{v}_i}{dt^2} = \alpha \zeta_i - \beta \hat{v}_i$$

$$- \sum_{j (j \neq i)} \left( C_a \frac{\| \xi_j - \xi_i \|}{l_a} - C_r \frac{\| \xi_j - \xi_i \|}{l_r} \right) \left( (x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i \right) \hat{\xi}$$

$$\frac{(x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i}{\| \xi_i - \xi_j \|} \hat{\zeta} \quad \text{Eq. 11}$$

Notice that the first three terms above are parallel to motion equations of Kellys (Eq. 7abc), and can be expressed as following:

$$m \frac{d^2 \hat{u}_i}{dt^2} = \alpha \cos \theta_i - \beta \hat{u}_i$$

$$- \sum_{j (j \neq i)} \left( C_a \frac{\| \xi_j - \xi_i \|}{l_a} - C_r \frac{\| \xi_j - \xi_i \|}{l_r} \right) \left( (x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i \right) \hat{\xi}$$

$$\frac{(x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i}{\| \xi_i - \xi_j \|} \hat{\zeta} \quad \text{Eq. 12a}$$

$$m \frac{d^2 \hat{v}_i}{dt^2} = \alpha \sin \theta_i - \beta \hat{v}_i$$

$$- \sum_{j (j \neq i)} \left( C_a \frac{\| \xi_j - \xi_i \|}{l_a} - C_r \frac{\| \xi_j - \xi_i \|}{l_r} \right) \left( (x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i \right) \hat{\xi}$$

$$\frac{(x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i}{\| \xi_i - \xi_j \|} \hat{\zeta} \quad \text{Eq. 12b}$$

$$m \frac{d^2 \hat{w}_i}{dt^2} = \alpha \cos \theta_i - \beta \hat{w}_i$$

$$- \sum_{j (j \neq i)} \left( C_a \frac{\| \xi_j - \xi_i \|}{l_a} - C_r \frac{\| \xi_j - \xi_i \|}{l_r} \right) \left( (x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i \right) \hat{\xi}$$

$$\frac{(x_i - x_j) \cos \theta_i + (y_i - y_j) \sin \theta_i}{\| \xi_i - \xi_j \|} \hat{\zeta} \quad \text{Eq. 12c}$$

Notice that Eq. 12c governs the rotation of the Kelly since the force is perpendicular to the velocity along the t-direction, which can be written as:

$$\int \frac{d^2 \hat{v}_i}{dt^2} = \int \frac{d^2 \hat{w}_i}{dt^2} = \int \frac{d^2 \hat{z}_i}{dt^2} = - \sum_{j (j \neq i)} \left( C_a \frac{\| \xi_j - \xi_i \|}{l_a} - C_r \frac{\| \xi_j - \xi_i \|}{l_r} \right) \left( (x_i - x_j) \sin \theta_i + (y_i - y_j) \cos \theta_i \right) \hat{\xi}$$

$$\frac{(x_i - x_j) \sin \theta_i + (y_i - y_j) \cos \theta_i}{\| \xi_i - \xi_j \|} \hat{\zeta} \quad \text{Eq. 13}$$
There is a length of $\|x\|$ multiplied on the right hand side since the rotation changes the angle $\theta$, an angle relative to the origin of the x-y coordinates. As a result, its the torque $\vec{N} = \vec{x} \times \vec{F}$ has magnitude of $N = \|x\| F_{\perp}$, where $F_{\perp}$ is the force component that is perpendicular to x direction.

One may relate the parameters of Kellys to those of swarming equations by comparing the set of Kellys’ equations (Eq. 7abc) to (Eq. 12abc), respectively.

\[
\mu = \beta
\]

\[
(F_{R} + F_{L}) = \alpha - \sum_{j(j \neq i)} \left( \left( \frac{C_{a}}{l_{a}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{a}}} - \frac{C_{r}}{l_{r}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{r}}} \right) \frac{(x_{i} - x_{j}) \cos \theta_{i} + (y_{i} - y_{j}) \sin \theta_{i}}{\|z_{i} - z_{j}\|} \right)
\]

\[
(F_{R} - F_{L}) r_{f} = -r_{f} \sum_{j(j \neq i)} \left( \left( \frac{C_{a}}{l_{a}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{a}}} - \frac{C_{r}}{l_{r}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{r}}} \right) \frac{-(x_{i} - x_{j}) \sin \theta_{i} + (y_{i} - y_{j}) \cos \theta_{i}}{\|z_{i} - z_{j}\|} \right)
\]

The differential force of the fans can be drawn from the comparison:

\[
(F_{R} - F_{L}) = -r_{f} \sum_{j(j \neq i)} \left( \left( \frac{C_{a}}{l_{a}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{a}}} - \frac{C_{r}}{l_{r}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{r}}} \right) \frac{(x_{i} - x_{j}) \cos \theta_{i} - \sin \theta_{i} + (y_{i} - y_{j}) (\sin \theta_{i} + \cos \theta_{i})}{\|z_{i} - z_{j}\|} \right)
\]

Consequently, the output force of the right and left fans are:

\[
F_{R} = \frac{\alpha}{2} - \sum_{j(j \neq i)} \left( \left( \frac{C_{a}}{l_{a}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{a}}} - \frac{C_{r}}{l_{r}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{r}}} \right) \frac{(x_{i} - x_{j}) \cos \theta_{i} - \sin \theta_{i} + (y_{i} - y_{j}) (\sin \theta_{i} + \cos \theta_{i})}{\|z_{i} - z_{j}\|} \right)
\]

\[
F_{L} = \frac{\alpha}{2} - \sum_{j(j \neq i)} \left( \left( \frac{C_{a}}{l_{a}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{a}}} - \frac{C_{r}}{l_{r}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{r}}} \right) \frac{(x_{i} - x_{j}) \cos \theta_{i} + \sin \theta_{i} + (y_{i} - y_{j}) (\sin \theta_{i} - \cos \theta_{i})}{\|z_{i} - z_{j}\|} \right)
\]

Gathering the identical terms within the above expressions, one may simply define:

\[
F_{1,i} = - \sum_{j(j \neq i)} \left( \left( \frac{C_{a}}{l_{a}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{a}}} - \frac{C_{r}}{l_{r}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{r}}} \right) \frac{(x_{i} - x_{j})}{\|z_{i} - z_{j}\|} \right)
\]

\[
F_{2,i} = - \sum_{j(j \neq i)} \left( \left( \frac{C_{a}}{l_{a}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{a}}} - \frac{C_{r}}{l_{r}} e^{-\frac{\|z_{i}-z_{j}\|}{l_{r}}} \right) \frac{(y_{i} - y_{j})}{\|z_{i} - z_{j}\|} \right)
\]

and
\[
F_R = \frac{\alpha}{2} + (\cos \theta_i - \sin \theta_i) F_{1,i} + (\sin \theta_i + \cos \theta_i) F_{2,i} \\
F_L = \frac{\alpha}{2} + (\cos \theta_i + \sin \theta_i) F_{1,i} + (\sin \theta_i - \cos \theta_i) F_{2,i}
\]

5.3 Experimentations and Analysis

We construct a controller that implements (Eq. 18) on any vehicle modeled by (Eq. 7) by calculating the amount of fan forces for each vehicle depending on their positions. We designed some test experiments to illustrate the feasibility of the control method. These simple test problems include (a) achieving target points including initialization from arbitrary location and orientation, (b) point-to-point motion using virtual attractive potentials, (c) object avoidance by virtual repulsive potentials, (d) multiple objects avoidance, and (e) follow the leader.

5.3.2 Achieving Target Point. In this experiment, one vehicle is placed at the starting point (5, 1), and the target point takes on only an attractive force. Under such conditions, the single vehicle strives to move toward the target point. In this test, the vehicle just needs to know its own position and the location of the target. As observed in the matlab simulation, the vehicle goes straight (fig. 19a) to the target point under ideal conditions (i.e. the fan forces mapping is the exact. In a real experiment the two fans often have a different response. In figure 19b, we include the fan force offset to the modeling equations and observe the shifting and self-correcting behavior of the vehicle in the process of getting to the target location.

We carry out the experiment described by Fig. 19 on the MVWT. The Kelly is also placed at some point in the vicinity of (1, 5) with various angles from facing the virtual target point at (5, 5). First, at zero degree from facing the target point (fig. 20), the Kelly tends to curve to the right due to the fans’ offset. Nonetheless, the self-correcting behavior of the algorithm allows the vehicle to find its path toward the target point. Similarly, when the Kelly is at 45, 90, and 180
degrees from facing the target point (fig. 20bcd), Kelly also smoothly turns and corrects itself to achieve the target point. Experimental data shows Kelly can hit the target point with tolerance of 20 cm. In the case of 180 degrees from facing the target point, observation shows that the turning radius is about one meter, which is an important parameter for designing multiple vehicle experiments.

Figure 20. Experimental data projecting Kelly’s trajectory of achieving the target point at (5, 5) from the vicinity of (1, 5) at various angles from facing the target point. a) zero degree, b) 45 degrees, c) 90 degrees, d) 180 degrees.

5.3.2 Point to Point. In this test, we direct the Kelly to autonomously runs from one target to another after the target is achieved. We place four virtual targets forming a square (fig. 20). The Kelly is commanded to run to the first target at (1.5, 1.5), and then to second target at (1.5, 5) after it is located within the tolerable vicinity of the first target, and then to the third target, then to the fourth, and keep looping in counterclockwise direction (fig. 20). There is some discrepancy in the path of the simulated vehicle and the path of the Kelly due to the offset between fan forces, frictional model of the Kelly, as well as other instrumental errors. Nonetheless, Kelly successfully achieves all targets.
5.3.3 Object Avoidance by Virtual Repulsive Potential. We can imagine a surface with hills and wells as the potential surface, where wells represent the attractive points and hills represent the repulsive points. An attractive potential pulls the vehicle to it, while the virtual repulsive potential acts as a hill that pushes the vehicle always from it. We place a virtual repulsive point at the center pole of the test floor and an attractive point is placed at (5, 5) (fig. 22). We can see the repulsive point pushes the vehicle out to prevent passing through it (fig. 23). This allows the vehicle to proceed to a waypoint while avoiding an obstacle.
5.3.4 Multiple Obstacle Avoidance. In the case of multiple obstacles, we need to consider the collective potential to determine the path of the vehicle. Work is in progress to determine the most probable path that the Vehicle would take.

Figure 23. Simulated Vehicle going from (0.5, 3) to (5, 3) while avoiding several obstacles on the map.

5.3.5 Follow the Leader. Let the first vehicle follow the moving attractive potential points as in section 5.3.2 and let there be a second vehicle. If the second vehicle experiences the attractive
and repulsive of the first vehicle, then the vehicles behave as leader and follower. Work is in progress to determine the geometry of the follower’s path and to determine factor that control the distance between leader and follower.

Figure 24. Simulation showing the geometry that the follower takes.

Acknowledgments:
This group research is supported by: NSF VIGRE grant DMS-9983726, NSF grant ACI-0321917, ONR grant N000140410054, ARO grant DAAD19-02-0055
Appendix A:

Circle Tracking Algorithm.

The simplified equations of motion (assuming perfect sensing and actuation, no delays, no disturbances, and linear friction) for the MVWT vehicles are listed in Equations (A1) – (A3). These are derived by observation from the simple schematic of the vehicle.

\[
\begin{align*}
    m\ddot{x} &= -\mu \dot{x} + (F_R + F_L) \cos \theta \\
    m\ddot{y} &= -\mu \dot{y} + (F_R + F_L) \sin \theta \\
    J\ddot{\varphi} &= -\varphi \dot{\varphi} + (F_R + F_L) r_f
\end{align*}
\]  

(A1)\hspace{1cm} (A2)\hspace{1cm} (A3)

However, the linearized dynamics are not controllable around any such equilibria. To achieve controllability, we consider the error dynamics around a constant velocity. We take \( \hat{x}_c, y_c, \hat{\theta}_c, \hat{\varphi}_c \) as the state variables, linearize the dynamics at the equilibrium point \([C,0,0,0,0]\), and get the linearized error dynamics as:

\[
\begin{bmatrix}
    \dot{\hat{x}}_c \\
    \dot{\hat{y}}_c \\
    \dot{\hat{\theta}}_c \\
    \dot{\hat{\varphi}}_c
\end{bmatrix}
= 

\begin{bmatrix}
    -\frac{\mu}{m} & 0 & 0 & 0 & 0 \\
    0 & 0 & 1 & 0 & 0 \\
    0 & 0 & -\frac{\mu}{m} & \frac{C\mu}{m} & 0 \\
    0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    \hat{x}_c \\
    \hat{y}_c \\
    \hat{\theta}_c \\
    \hat{\varphi}_c
\end{bmatrix}
+ 

\begin{bmatrix}
    \frac{1}{m} & 0 \\
    0 & 0 \\
    0 & 0 \\
    0 & \frac{r_f}{J}
\end{bmatrix}
\begin{bmatrix}
    u_1 \\
    u_2
\end{bmatrix}
\]  

(A4)

where \( u_1 \) and \( u_2 \) are propulsion and torque in the error dynamics.

MVWT testbed is only about \( 6.7 m \times 7.3 m \), so it's natural to follow a circle trajectory with radius \( r \) and constant speed \( r \cdot \dot{\xi} \) where \( \dot{\xi} \) is the constant angular speed. The algorithm is pretty simple. At each particular time, we measure the current states, calculate the reference states in cylindrical coordinates, get the states error vector \( e \). Here we assume that the current states and reference states have same angle with respect to the circle center. Then we use local classical controller or LQR controller to generate the control law. In circle trajectory, the equilibrium point is \( [r \cdot \xi, 0, 0, t \cdot \xi + \theta_0, \dot{\xi}] \).
Classical Controller

According to the linearized error dynamics, the dynamics between $\dot{x}_e$ and $u_1$ is totally decoupled from other state variables and inputs. So we can view this system as two parts:

- Speed dynamics. The speed $\dot{x}_e$ is totally controlled by $u_1$. This is a first-order subsystem.
- Departure dynamics. The departure $y_e$ are determined by $\theta_e$, and $\theta_e$ is driven by $u_2$. This is a two-layers second-order subsystem.

The speed dynamics is a first-order transfer function that is totally decoupled from other variables, so we just use a PI controller. For the departure dynamics, we need an inner controller for $\theta_e$ and an outer controller for $y_e$. Fig. 1 shows how this strategy works. If the inner layer responses much quicker than the outer layer, then we can design the controllers separately.

![Diagram]

(Figure A1)

LQR Controller

We do have a LQR controller based on the error dynamics. We select the diagonal matrices $Q$ and $R$ for any particular circle trajectory and get the gain matrix $G$ by using MATLAB function $LQR(A, B, Q, R)$, and the controller law is

$$F = -G \cdot e$$

$$F_{\text{nom}} = \begin{bmatrix} F_{L,\text{nom}} \\ F_{R,\text{nom}} \end{bmatrix} = \begin{bmatrix} 0.5 \sqrt{(mr\dot{z}_r^2)^2 + (\mu r \dot{z}_r^2 - \psi \dot{z}_r / r_f)} \\ F_{L,\text{nom}} + \psi \dot{z}_r / r_f \end{bmatrix}$$

$$F = F_e - F_{\text{nom}}$$

Please note that the nominal fan forces are constant.
References:
[1] Z. Jin, Dynamics and Controllers of Vehicle Kelly, April 7, 2004, California Institute of Technology, Pasadena, CA, USA


[8] J. Gibbs, D. Tung, Caltech MVWT API, California Institute of Technology, Pasadena, CA, USA


[12] Intelligent Autonomous Systems Laboratory (IAS), University of West England. www.ias.uwe.ac.uk

32