2019 IEEE Conference on Control Technology and Applications (CCTA)
Hong Kong, China, August 19-21, 2019

Implementation of an Open-Loop Turn Circle Intercept Controller

Addison Schwamb
Department of Electrical and Computer Engineering
Oklahoma Christian University
Edmond, Oklahoma
addison.schwamb@eagles.oc.edu

Lynnae Frisco
Department of Manufacturing Engineering
Central State University
Wilberforce, Ohio
lynnaesheree@gmail.com

Pavlos Androulakakis
Department of Electrical Engineering
University of Cincinnati
Cincinnati, Ohio
androups@mail.uc.edu

Samuel Susanto
Department of Electrical Engineering
Wright State University
Dayton, Ohio
susanto.2@wright.edu

Zachariah Fuchs
Department of Electrical Engineering
University of Cincinnati
Cincinnati, Ohio
zachariah.fuchs@uc.edu

***Abstract*—We examine the practical implementation of an evolutionary algorithm to solve a pursuit evasion problem. In this problem, an Attacker and a Target move about an infinite plane with constant speeds. The Target moves in a constant circle, whereas the Attacker is free to change its direction with a bounded turn rate. The goal of the Attacker is to capture the Target in minimum time. We used pursuit evasion robots to physically model this problem and implement an evolutionary algorithm to evaluate the controller performance in real-world systems.**

I. Introduction

Pursuit evasion scenarios have many applications, from search and surveillance to missile tracking and dogfighting scenarios. In pursuit evasion scenarios, the pursuer's goal state is tied to a mobile agent that evades capture. As a result, path planning is of vital importance, particularly because the optimal path must take both goal state and arrival time into account. Analytic optimal control methods have often been used to analyze pursuit evasion problems. Perhaps the simplest example is that of a single turn-constrained agent pursuing a stationary target. This is known as the Dubins Vehicle Problem, and was analytically solved by Lester E. Dubins in 1957 [1]. Further work has been done to solve more complex scenarios, in which the target evades the pursuer [2].

While analytic methods can find the exact optimal strategy for simple pursuit evasion games, more complex real-world scenarios cannot be solved analytically. Even when analytical solutions are technically possible, they become exceedingly complicated as greater numbers of pursuers and targets are added. In these cases, evolutionary algorithms can be used to search the space of admissible solutions. Evolutionary algorithms use the principles of natural selection to create populations of possible solutions. As they iterate through many generations, they are capable of searching large solution spaces in a relatively efficient manner. This efficiency makes them well-suited to solving pursuit evasion scenarios when an analytical solution is not practical. Evolutionary algorithms have often been used to create controllers for trajectory optimization. Kok and Gonzalez use an evolutionary algorithm to develop a path-planning controller for an unmanned aerial vehicle (UAV) [3]. Navigation of an environment strewn with obstacles is managed through an evolutionary algorithm in [4].

In this project, we implement an open-loop controller produced by an evolutionary algorithm to solve a turn circle intercept problem, which was previously developed in [5]. The turn circle intercept problem considers two agents, an Attacker and a Target, moving about a two-dimensional, obstacle-free, infinite plane. The Target has a constant turn rate resulting in a circular trajectory about the origin. The Attacker is free to choose its control but has a bounded turn rate, which results in a minimum turning radius. The Attacker strives to capture the Target in minimum time by moving into a position behind it on the Target's turn circle. To solve this problem, an evolutionary algorithm is used to produce the optimal Attacker control strategy. The resulting optimal controller was previously analyzed in simulation [5], and in this paper, we will show that the resulting controllers are also implementable and effective in physical systems.

Optimal control strategies are only the first step, however. There are many applications of path planning, and those applications need practical solutions that work in real physical systems. A control strategy that creates a path leading to the Target is not particularly useful if a physical Attacker cannot use it to catch a physical Target. Our goal, therefore, was to implement their algorithm with pursuit evasion robots, and thus determine whether it is practical for use in physical systems. The problem is defined in Section II. Section III

The work was supported in part by NSF grant EEC-1659813

978-1-7281-2767-5/19/$31.00 ©2019 IEEE

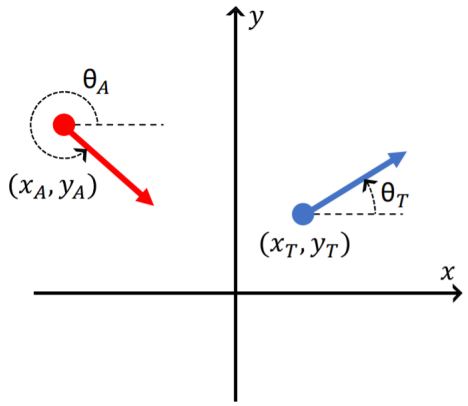


Fig. 1. Global Coordinates

gives a brief description of the evolutionary algorithm in [5] that was used to generate paths for the pursuit evasion robots. In Section IV, we discuss our methods of implementing the algorithm with the robots and the experiments conducted to test their performance. Results are presented in section V, and conclusions are drawn in Section VI.

II. Problem Description

Consider two agents on an infinite plane: an Attacker pursuing and a Target evading. This problem was originally presented in [5], and a description of the system is included in this section for completeness. The goal of the Attacker is to capture the Target in minimum time. This occurs when the Attacker moves into a position behind the Target on the Target's turn circle. The Target and the Attacker both have constant speed. The Attacker has a variable but bounded turn rate, and the Target has a constant turn rate ω_T.

A. System Model

The states of the Attacker and the Target are defined by their positions, (x_A, y_A) and (x_T, y_T), and their heading angles, θ_A and θ_T. An illustration of the two agents with their labeled states is shown in Figure 1. The complete state of the system is defined by the individual agent states, as well as a time state, denoted by τ:

$$\mathbf{x} := (x_A, y_A, \theta_A, x_T, y_T, \theta_T, \tau) \tag{1}$$

The system dynamics $\dot{\mathbf{x}} := \mathbf{f}(\mathbf{x}, u_A, u_T)$ are defined by a system of seven ordinary differential equations:

$$\dot{x}_A := v_A \cos(\theta_A) \tag{2}$$
$$\dot{y}_A := v_A \sin(\theta_A) \tag{3}$$
$$\dot{\theta}_A := \frac{v_A}{\rho_A} u_A \tag{4}$$
$$\dot{x}_T := v_T \cos(\theta_T) \tag{5}$$
$$\dot{y}_T := v_T \sin(\theta_T) \tag{6}$$
$$\dot{\theta}_T := \frac{v_T}{\rho_T} u_T \tag{7}$$
$$\dot{\tau} := 1 \tag{8}$$

The constants $v_A > 0$ and $v_T > 0$ are the Attacker and Target's respective speeds. The constants $\rho_A > 0$ and $\rho_T > 0$ are the Attacker and Target's respective minimum turn radii. The agents control their headings through $u_A \in [-1, 1]$ and $u_T \in [-1, 1]$. The state at initial time t_0 is defined as $\mathbf{x}(t_0) = \mathbf{x}_0 := (x_{A0}, y_{A0}, \theta_{A0}, x_{T0}, y_{T0}, \theta_{T0}, \tau_0)$.

For this problem, the Target is constrained to a constant turn rate of $u_T = 1$. Using this control strategy and initial condition, the Target trajectory can be calculated by integrating the dynamics (5)-(7) with respect to time:

$$x_T(t; \mathbf{x}_0) = x_{T0} - \rho_T \sin\theta_{T0} + \rho_T \sin\left(\theta_{T0} + \frac{v_T}{\rho_T} t\right) \tag{9}$$

$$y_T(t; \mathbf{x}_0) = y_{T0} + \rho_T \cos\theta_{T0} - \rho_T \cos\left(\theta_{T0} + \frac{v_T}{\rho_T} t\right) \tag{10}$$

$$\theta_T(t; \mathbf{x}_0) = \theta_{T0} + \frac{v_T}{\rho_T} t \tag{11}$$

This results in a circular trajectory with a a radius of ρ_T and center located

$$(x_c, y_c) := (x_{T0} - \rho_T \sin\theta_{T0}, y_{T0} + \rho_T \cos\theta_{T0}). \tag{12}$$

Unlike the Target, we allow the Attacker to choose its control $u_A \in [-1, 1]$ throughout the course of the pursuit. However, analysis of problems with similar dynamics [6], [7], [8] have determined that the optimal strategy for the Attacker typically employs a "bang-zero-bang" strategy, in which the pursuer will either make a hard right turn $(u_A = -1)$, a hard left turn $(u_A = 1)$, or go straight $(u_A = 0)$. Using those results as justification, we restrict the Attacker's control variable (u_A) such that it can only take on the discrete values of $u_A \in \{-1, 0, 1\}$.

In order to compute the Attacker's trajectory, we parameterize the controller with a sequence of individual time segments in which the control is held constant as shown in the following equation.

$$u_A(t; \mathbf{C}_A) = \begin{cases} u_1 & 0 \le t < t_1 \\ u_2 & t_1 \le t < t_2 \\ u_3 & t_2 \le t < t_3 \\ \vdots & \\ u_N & t_{N-1} \le t < t_N \\ 0 & t_N \le t \end{cases} \tag{13}$$

The parameter set $\mathbf{C}_A := \{\{u_1, u_2, ..., u_N\}, \{t_1, t_2, ..., t_N\}\}$ contains the control, $u_i \in \{-1, 0, 1\}$, and switch time, $t_i \ge 0$ for each segment. Note that this control structure assumes that $0 \le t_1 \le t_2 \le ... \le t_N$

Substituting the Attacker's control into the Attacker dynamics (2)-(4) and integrating with respect to time yields the Attacker's trajectory:

$$x_A(t; \mathbf{x}_0, \mathbf{C_A}) = \tag{14}$$
$$\begin{cases} x_{Ai} - \rho_T \sin\theta_{Ai} + \rho_A \sin(\theta_{Ai} + \frac{v_i}{\rho_A}(t - t_i)) & u_{i+1} = 1 \\ x_{Ai} + v_i \cos(\theta_{Ai})(t - t_i) & u_{i+1} = 0 \\ x_{Ai} + \rho_A \sin\theta_{Ai} - \rho_A \sin(\theta_{Ai} - \frac{v_i}{\rho_A}(t - t_i)) & u_{i+1} = -1 \end{cases}$$

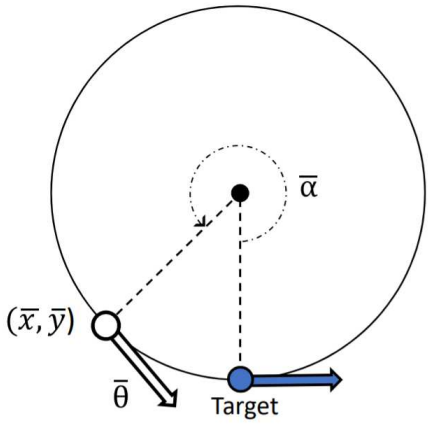


Fig. 2. Desired Separation

$$y_A(t; \mathbf{x}_0, \mathbf{C_A}) = \tag{15}$$
$$\begin{cases} y_{Ai} + \rho_T \cos\theta_{Ai} - \rho_A \cos(\theta_{Ai} + \frac{v_i}{\rho_A}(t - t_i)) & u_{i+1} = 1 \\ y_{Ai} + v_i \sin(\theta_{Ai})(t - t_i) & u_{i+1} = 0 \\ y_{Ai} - \rho_A \cos\theta_{Ai} + \rho_A \cos(\theta_{Ai} - \frac{v_i}{\rho_A}(t - t_i)) & u_{i+1} = -1 \end{cases}$$

$$\theta_A(t; \mathbf{x}_0, \mathbf{C_A}) = \begin{cases} \theta_{Ai} + \frac{v_i}{\rho_A}(t - t_i) & u_{i+1} = 1 \\ \theta_{Ai} & u_{i+0} = 0 \\ \theta_{Ai} - \frac{v_i}{\rho_A}(t - t_i) & u_{i+1} = -1 \end{cases} \tag{16}$$

where the index i satisfies $i = \arg\max_i t_i < t$. The intermediate state components are computed recursively as $x_{Ai} = x_A(t_i; \mathbf{x}_0, \mathbf{C}_A)$, $y_{Ai} = y_A(t_i; \mathbf{x}_0, \mathbf{C}_A)$, and $\theta_{Ai} = \theta_A(t_0; \mathbf{x}_0, \mathbf{C}_A)$. The initial conditions are defined as $x_A(t_0; \mathbf{x}_0, \mathbf{C}_A) = x_{A0}$, $y_A(t_0; \mathbf{x}_0, \mathbf{C}_A) = y_{A0}$, and $\theta_A(t_0; \mathbf{x}_0, \mathbf{C}_A) = \theta_{A0}$.

B. Utility

Given an initial condition $\mathbf{x}_0$, control parameter set $\mathbf{C}_A$, and terminal time t_f, the final system state $\mathbf{x}_f$ is computed using the trajectories defined in (9)-(11) and (14)-(16):

$$\begin{aligned} \mathbf{x}_f(\mathbf{C}_A, t_f) &:= (x_{Tf}, y_{Tf}, \theta_{Tf}, x_{Af}, y_{Af}, \theta_{Af}, t_f) \\ &= (x_T(t_f; \mathbf{x}_0), y_T(t_f; \mathbf{x}_0), \theta_T(t_f; \mathbf{x}_0), ... \\ &\quad x_A(t_f; \mathbf{x}_0, \mathbf{C}_A), y_A(t_f; \mathbf{x}_0, \mathbf{C}_A), \theta_A(t_f; \mathbf{x}_0, \mathbf{C}_A), t_f). \end{aligned}$$

In this problem, terminal time is assumed to be the maximum time of the final control segment: $t_f = t_N$.

The Attacker's goal is to move into a position behind the Target which is both on the Target's turn circle and at a desired separation distance. The separation distance between the Attacker and the Target is defined in terms of $\bar{\alpha}$ as shown in Figure 2. Using these constraints, the desired terminal (x,y)-coordinates of the Attacker can be expressed as

$$\bar{x} = (x_{Tf} - x_c)\cos(\bar{\alpha}) - (y_{Tf} - y_c)\sin(\bar{\alpha}) + x_c$$

$$\bar{y} = (y_{Tf} - y_c)\cos(\bar{\alpha}) + (x_{Tf} - x_c)\sin(\bar{\alpha}) + y_c$$

in which x_c and y_c are the (x,y)-coordinates of the center of the Target's turn circle as defined in (12).

The performance of a given Attacker control strategy is based on a weighted sum of three functions. The first function is an error function that computes the distance between the Attacker's terminal position (x_{Af}, y_{Af}) and the desired position $(\bar{x}, \bar{y})$:

$$h_1(\mathbf{x}_f; \bar{\alpha}) := \sqrt{(\bar{x} - x_{Af})^2 + (\bar{y} - y_{Af})^2}.$$

The second function is an error function that computes the difference between Attacker's terminal heading θ_{Af} and the desired heading $\bar{\theta}$:

$$h_2(\mathbf{x}_f) := \sqrt{(\cos\theta_{Af} - \cos(\theta_{Tf} + \bar{\alpha}))^2 + (\sin\theta_{Af} - \sin(\theta_{Tf} + \bar{\alpha}))^2}$$

The third function is a cost function that is equal to the total amount of time elapsed.

$$h_3(\mathbf{x}_f; \mathbf{C}_A) = t_N.$$

The overall utility function is defined as a weighted sum of the three utilities:

$$U(\mathbf{C}_A) = w_1 h_1 + w_2 h_2 + w_3 h_3, \tag{17}$$

where w_1, w_2, and w_3 are positive weight coefficients. Adjusting the relative magnitudes of the weights prioritizes satisfying the terminal constraints or minimizing the total time to capture.

C. Problem Definition

Optimization of the overall utility function (17) is stated as the minimization over the Attacker's parameter set

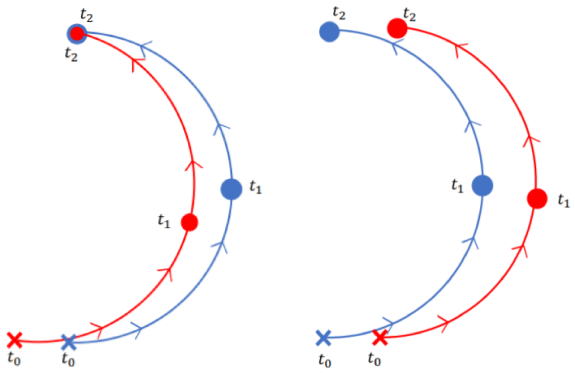
$$\min_{\mathbf{C}_A} U(\mathbf{C}_A) \tag{18}$$

This equation is used to find the optimal parameter set $\mathbf{C}_A^*$, which will yield the optimal Attacker strategy $u_A^*(t) = u_A(t; \mathbf{C}_A^*)$. Because the optimization problem defined in (18) contains both integer $\{u_1, u_2, ..., u_N\}$ and continuous $\{t_1, t_2, ..., t_N\}$ values, optimization is not possible using gradient methods and we instead use an evolutionary algorithm to find near optimal open-loop solutions.

III. Evolutionary Algorithm

The evolutionary algorithm (EA) implemented in [5] begins by creating an initial generation $\mathbf{G}_0$ of N parameterized controllers $\mathbf{G}_0 = \{\mathbf{C}_{A1}, \mathbf{C}_{A2}, \ldots, \mathbf{C}_{AN}\}$. Each of these controllers is randomly initialized with control and time values. The EA then evaluates each of the controllers in the current generation from a given initial condition $\mathbf{x}_0$ and assigns them their corresponding utility U as shown in Equation 17. The next generation is then created with a population distribution as follows. The controllers with utility in the top 5% of the population are passed to the next generation with no change. This process is usually referred to as *elitism* and is done to ensure that the best controllers persist from one generation to the next. The remaining 95% of the next generation is filled with the children that result from crossing and mutating of the previous generation.

Crossing is an integral part of most EAs and is performed in order to mix the genetic information of the current population in such a way that the offspring will be able to achieve



(a) Lead Pursuit (b) Lag Pursuit

Fig. 3. Air Combat Tactics

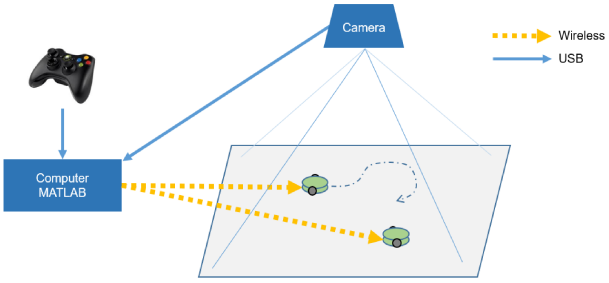


Fig. 4. Testbed System Diagram

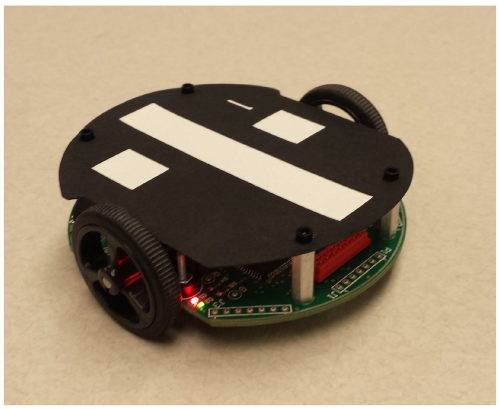


Fig. 5. Mobile Robot

higher utility. The crossing processes begins by randomly selecting two parents from the previous generation. Each of the controllers in the previous generation has an equal probability of being selected as a parent. Once two parents are selected, the child's control values are randomly selected from the parents as described in the following equation.

$$p(u_{ci}) = \begin{cases} .5 & u_{Ai} \\ .5 & u_{Bi} \end{cases}.$$

The child's time values are randomly chosen from a range of times defined by the parent's times as shown in the following equation.

$$p(\Delta t_{ci}) = \begin{cases} \frac{1}{\Delta t_{max} - \Delta t_{min}} & \Delta t_{ci} \in [\Delta t_{min}, \Delta t_{max}] \\ 0 & \text{otherwise} \end{cases},$$

The upper and lower bounds (Δt_{max} and Δt_{min}) are defined as a function of the parent's segment times

$$t_{min} = \max(0, .5(\Delta t_{Ai} + \Delta t_{Bi}) - \gamma_1 |\Delta t_{Ai} - \Delta t_{Bi}|)$$
$$t_{max} = \min(\overline{\Delta t}, .5(\Delta t_{Ai} + \Delta t_{Bi}) + \gamma_1 |\Delta t_{Ai} - \Delta t_{Bi}|).$$

where γ is a parameter that can stretch or shrink the selection region.

After crossing, mutation is performed. The goal of mutation is to add new genetic information into the population. This helps the EA explore new areas of the solution space. Each of the control and time segments in each of the children has a μ chance of mutating. When a mutation occurs to a particular value, a new value is randomly selected to take its place. If a control segment is mutated, a new control is chosen randomly from $[-1, 0, 1]$. If a time segment is mutated, a new time length is chosen randomly from a range of possible times defined $\Delta t_{mi} \in [t_{ci} - \sigma, t_{ci} + \sigma]$ where σ is a mutation severity parameter. For more details on the crossing and mutation methods and the reasoning behind their design, see Section III in [5].

IV. Methods of Implementation

A. Air Combat Tactics

The objective of this study is to implement the solutions found by the evolutionary algorithm. The authors of [5] test the evolutionary algorithm from three different initial conditions. These initial conditions each lend themselves to the implementation of a different canonical pursuit tactic: lead pursuit, lag pursuit, and an intercept trajectory. Lead and lag pursuit are general air combat tactics that are used to adjust an agent's position relative to another agent without changing speed [9]. In lead pursuit, the Attacker cuts inside the Target's turn circle and catches up to it by traveling a shorter distance. In lag pursuit, the Attacker falls behind the Target by traveling a path outside of the Target's turn circle. Figures 3a and 3b show illustrations of these pursuit techniques wherein the blue trajectory represents the Target and the red trajectory belongs to the Attacker. For the intercept trajectory, the Attacker is placed farther away from the Target and must time its entrance into the Target's turn circle. In this paper, similar initial conditions were selected in order to compare the real-world controller performance to the EA generated trajectories. The performance of each controller is evaluated by measuring the final separation distance between the desired position of intercept and the actual intercept position using and overhead camera system.

B. Hardware Description

The testbed system used to implement the evolved controllers consists of three main components: a central computer, an overhead camera system, and mobile robots. A basic system diagram that illustrates the interaction between the components is shown in Figure 4. The central computer uses

the overhead camera to estimate the position and heading of the multiple mobile robots. The robots are then controlled by a wireless link to the central computer. A more detailed description of the system design and functionality can be found in [10]. A picture of the custom robots used in this experiment is shown in Figure 5. Each robot utilizes a unique identification/localization pattern. The robots are wirelessly sent wheel speed commands from a central computer. Although the robots possess a differential drive train, we can emulate the robotic dynamics described in (2)-(8) using the following wheel speed relationships

$$v_R = 85.117v - u(2.857\rho) \tag{19}$$

$$v_L = 85.117v + u(2.857\rho) \tag{20}$$

where v_R and v_L are the right and left wheel speeds respectively. These conversions assume that the speed is in m/s and the turn rate is in rad/s. As the experiment is running, a video from the camera with overlaid calculated robot trajectories is displayed in real time. Figure 6 show an example screen capture of the camera feed watching over the testbed. In this scenario, Robot 2 is chasing a stationary Robot 1. The yellow trajectory shows the path predicted by the algorithm that Robot 2 will take to capture Robot 1. The actual path that Robot 2 takes can vary based on real world imperfections in the system.

C. Testing Predefined Initial Conditions

To begin testing, we utilized the test conditions previously evaluated via simulation in [5]:
Lag pursuit:

$$x_0 = (1.575, 0.653, \pi/4, 0, 0, 0, 0) \tag{21}$$

Lead pursuit:

$$x_0 = (-1.575, 0.653, -\pi/4, 0, 0, 0, 0) \tag{22}$$

Intercept trajectory:

$$x_0 = (-8.909, 0, 0, 0, 0, 0, 0) \tag{23}$$

To test these configurations, robots were placed in the given initial x and y positions with the given initial headings.

D. General Testing

After the robots were positioned in their specific configurations, tests were performed in which the testbed camera determined the robots' initial positions. This initial position is then fed into the evolutionary algorithm which runs for 1000 generations to estimate the optimal path of capture and returns the control matrix needed to implement the given path. The control matrix created by the algorithm is an n x 2 matrix. In our case, the control matrices were all 7-segmented, or 7 x 2. Figure 7 shows an example of one of these control matrices. Column 1 of the matrix is a vector filled with control elements $u \in [-1, 0, 1]$. These values tell the robot how to maneuver. When $u = 1$ the robot will turn left, when $u = 0$ the robot will go straight, and when $u = -1$ the robot will turn right. Column 2 is a time vector that regulates the amount of time to implement each corresponding control. The central computer converts these control outputs into wheel speeds and sends the wireless commands to the robots for the specified amount of time.

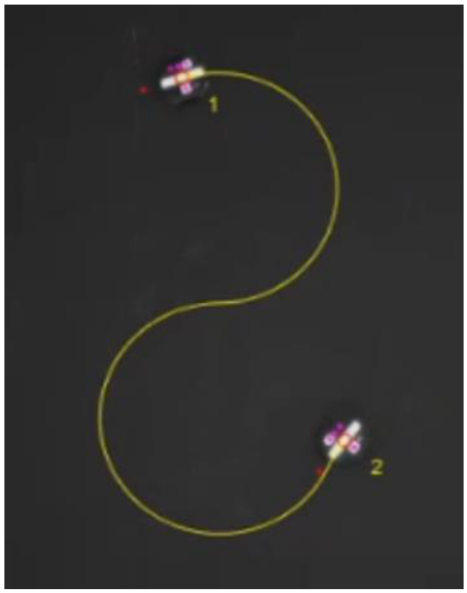


Fig. 6. Testbed Camera Image

V. Results and Analysis

A. Predefined Initial Conditions

We ran each configuration examined in [5] five times, and measured the distance between the robots' predicted ending positions and the robots' actual ending positions. These results are detailed in Table I.

In general, the expected and actual positions were quite similar with some minor errors from real world imperfections. However, a significant amount error was caused by collisions between the robots. The robots would occasionally collide over the course of the experiment, and the collisions would cause the robots to either be knocked off course, or to be locked to each other, unable to move at all. These collisions resulted in a few ending positions that were significantly different from the predicted ones which negatively influenced the average. Real world problems such as this are often overlooked when implementing things in simulation. By testing the evolved solutions with a real world system, we were able to show that while the solution makes sense in simulation, it is not always able to be implemented in the real world. This result can then be used by the authors of [5] to improve their algorithm by taking the robots physical size and collisions into account.

B. General Testing

To test the algorithm's performance in more varied situations, we ran two experiments: one with randomly generated initial positions, and one with initial positions well suited to lead, lag, and intercept pursuit. Guidelines were set to determine the differences between a capture, non-capture, or failed run. To be counted as a capture, the Attacker must end

TABLE I
RESULTS FROM TESTING PREDEFINED INITIAL CONDITIONS

		X Expected (in)	Average X Measured (in)	Y Expected (in)	Average Y Measured (in)	Average Error Distance (in)
Lag	A	-15.64	-16.09	27.37	28.87	2.77
	T	-17.60	-16.40	19.55	22.40	3.65
Lead	A	7.82	4.47	33.24	31.34	3.97
	T	1.96	3.46	35.19	33.94	2.27
Intercept	A	-17.60	-16.15	19.55	10.55	7.74
	T	-15.64	-18.89	11.73	11.98	5.43

TABLE II
RESULTS FROM GENERAL TESTS

	Capture Rate	Average Error Distance (in)	Standard Deviation (in)
Random Positions	60%	7.2	5.12
Lag Trajectories	90%	2.76	.814
Lead Trajectories	90%	3.52	2.25
Intercept Trajectories	30%	7.87	4.05

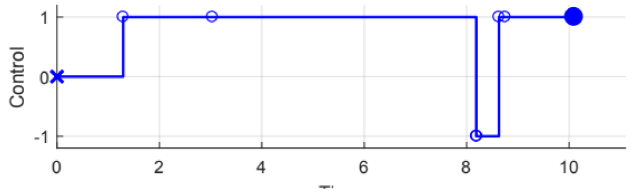


Fig. 7. Example Control Summary and Control Matrix

behind the Target within the predicted capture distance. If the robots were separated by more than the predicted capture distance of $\bar{\alpha}$ (set to $\pi/8$ for our experiments), the run was considered a non-capture. When a collision occurred, it was counted as a failed run.

In the first experiment, Target and Attacker robots were placed in random positions on the testbed. Testing the robots in random positions allowed the algorithm to create paths that have not been analyzed in [5]. We ran the algorithm and the robots 25 times, placing them in different positions each time. In this experiment, the Attacker caught the Target 60% of the time.

In the second experiment, we placed the robots in positions well suited to lead, lag, and intercept trajectories, to see if the robots worked better when they could employ typical air maneuvering tactics. Each configuration was run 10 times. Figure 8 shows examples of the predicted paths and the paths the robots actually traveled for each trajectory tested. In this experiment, the lead and lag configuration had a 90% capture rate, while the intercept trajectories only had a 30% capture rate, clearly indicating that the robots did indeed perform better when using lead and lag strategies.

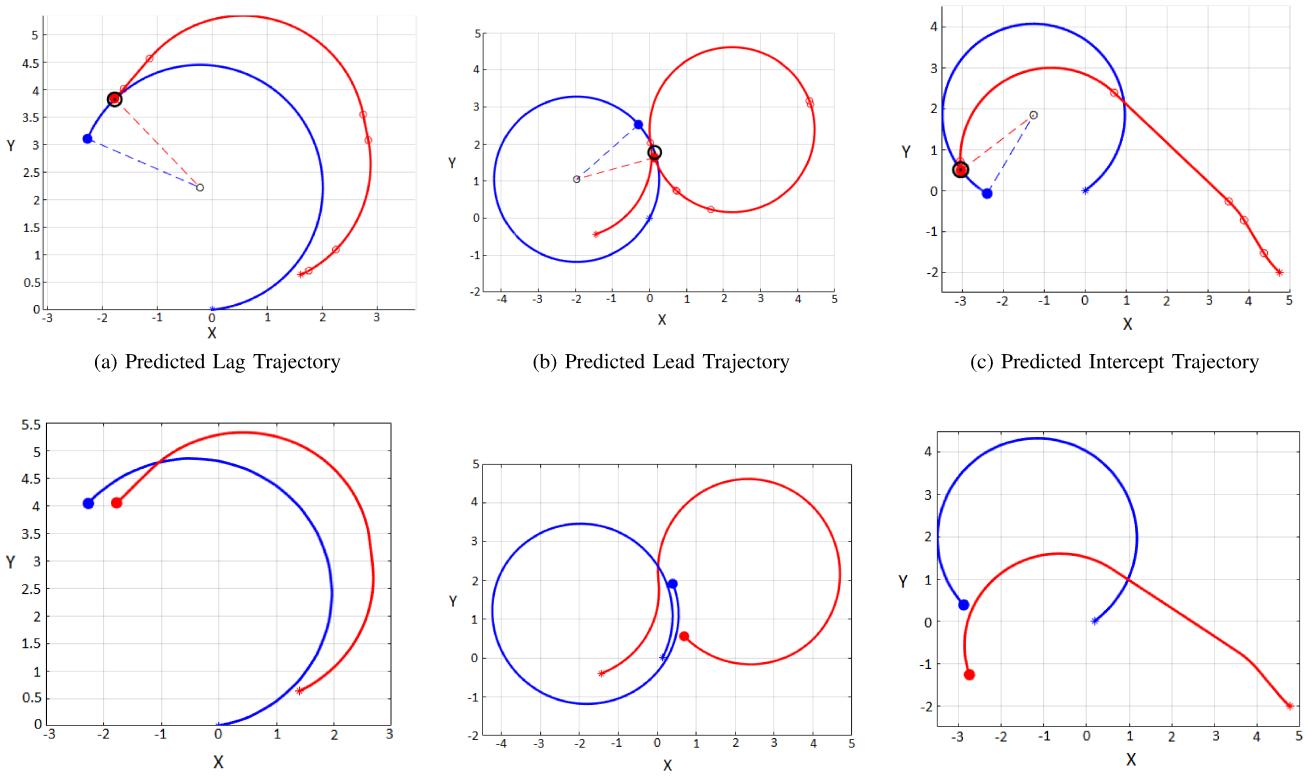
For both experiments, we measured how far off of the expected ending position the Attacker was, taking hand measurements with a ruler. In the first experiment, we marked the expected final positions of both robots on the testbed, and measured the x and y distances from the center of each robot to its expected position marking. In the second experiment, we measured the x and y distances from the center of the Target to the center of the Attacker, and compared these distances to the expected difference in position generated by the evolutionary algorithm. Using these measurements, we found the standard deviation and the percent error. These results are detailed in Table II.

As Table II shows, the percent error of the robots' positions is higher when the Attacker has a lower capture percentage. To test this correlation, we found the average error distance and standard deviation for captures and non-captures in the random and intercept trajectory tests. As the table shows, non-captures are correlated to greater differences between the expected and measured position and a higher percent errors. This is expected because the Attacker must be inside the separation angle $\bar{\alpha}$ in order to successfully capture the Target. If the Attacker is outside $\bar{\alpha}$, it is likely further from the predicted ending point.

We believe that the higher capture rate in the lead and lag pursuit tests is correlated to the simplicity of the paths the robots took. In both strategies, the Attacker has a relatively straightforward control matrix: in some cases, it effectively makes a single turn. In intercept trajectories, the Attacker's control matrix is more complicated, involving more turns and changes in direction. This added complication introduces more potential for errors to build up from hardware inaccuracies. In addition, intercept trajectories often involve the Attacker traveling a greater distance to reach the Target than in lead and lag pursuit. If the algorithm is slightly off for a small distance, it may not matter. Small errors compounded by a large distance and multiple turns, however, could result in a non-capture. This can be seen in Figure 8. While the lead and lag trajectories ended close to the predicted endpoints, the intercept trajectory followed a path similar to the predicted one, and yet ended fairly far away from the predicted endpoint.

VI. CONCLUSION

The implementation of the control strategies produced by the evolutionary algorithm was able to effectively control the Attacker to capture the Target in most cases. However, collisions and the build up of small errors over longer distances led to some significant sources of error. Therefore, we conclude that the algorithm is practical for use in physical systems, but could be improved by giving the agents a nonzero size and adding the corresponding physical restric-



(a) Predicted Lag Trajectory (b) Predicted Lead Trajectory (c) Predicted Intercept Trajectory

(d) Actual Lag Trajectory (e) Actual Lead Trajectory (f) Actual Intercept Trajectory

Fig. 8. Examples of predicted paths and paths the robots implemented.

tions. Based on the experimental results, it can be concluded that the algorithm works best when the relative positions of the robots are well-suited to a lead or lag pursuit strategy. The algorithm performance is worse when the Attacker must time its interception of the Target's path from a far away distance. In this case the capture rate is not as high because of the complexity of the paths coupled with the low precision of the hardware.

One limitation of the evolutionary algorithm is that it considers an infinite plane when predicting a path for capture. Our testbed is only 9'4" x 8'11", and these boundaries sometimes prevented the path predicted from being completed. In the future, work can be done to set parameters to prevent the algorithm from generating paths that cannot be completed in the testbed or in any environment in which the algorithm is being tested. This can be extended to the general case by adding support for obstacles.

REFERENCES

[1] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," American Journal of Mathematics, vol. 79, no. 3, pp. 497-516, Jul. 1957.

[2] D. W. Casbeer, E. Garcia, Z. E. Fuchs, and M. Pachter, "Cooperative target defense differential game with a constrained-maneuverable defender," in 2015 54th IEEE Conference on Decision and Control (CDC), Dec. 2015, pp. 1713-1718.

[3] N. K. J. Kok, L. F. Gonzalez, "Fpga implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning," Transactions on Evolutionary Computation, vol. 17, pp. 272-281, April 2013.

[4] C. Hocaoglu and A. C. Sanderson, "Planning multiple paths with evolutionary speciation," IEEE Transactions of Evolutionary Computation, vol. 5, no. 3, pp. 169-191, June 2001.

[5] P. Androulakakis, Z. E. Fuchs, and J. Shroyer, "Evolutionary design of an open-loop turn circle intercept controller," in IEEE Congress on Evolutionary Computation, July 2018, pp.

[6] P. Androulakakis and Z. E. Fuchs, "Evolutionary design of engagement strategies for turn-constrained agents," in IEEE Congress of Evolutionary Computation, May 2017, pp. 2354-2363.

[7] Z. E. Fuchs, D. W. Casbeer, and E. Garcia, "Singular analysis of a multi-agent, turn-constrained, defensive game," in American Control Conference (ACC), 2016, July 2016.

[8] A. Merz, "The homicidal chauffeur," AIAA Journal, vol. 12, no. 3, pp. 259-260, 1974.

[9] Multi-Command Handbook 11 F-16. United States Air Force, 1996.

[10] E. Nees, "Design and Demonstration of a Physical, Multi-Agent, Autonomous Controller Testbed", Master's Thesis, Wright State University, 2017.