ASYMMETRIC ROBOT MOTION DESIGN FOR PURSUIT-EVASION GAMES

A Thesis

by

AUSTIN LANCE LUNDGREN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2020

Major Subject: Computer Engineering

ABSTRACT

Symmetric turning control is the typical design choice for most machines. However, historical examples of asymmetric machine design, as well as examples of asymmetry in nature, suggest that asymmetric turning may be a potential advantage in adversarial applications. For instance, aircraft of World Wars I and II were plagued by asymmetric turning controls as a result of gyroscopic forces from the rotating engine. Pilots of the time actually believed this to be a feature, not a bug, suggesting that the asymmetric turning improved strategic evasion and pursuit during battle. As autonomous robots become increasingly critical in military operations, it is imperative that we endow them with strategic designs for better performance. We seek to understand if asymmetric turning is an advantageous design.

Using Karaman and Frazzoli's sample-based algorithm for pursuit-evasion games, software simulates robot motion planning in an asymmetric Dubins state space to observe how asymmetric turning influences agent success. We demonstrate mathematically that the Dubins interval path solutions are applicable to asymmetric Dubins vehicles, as both are utilized within the simulation. The *Open Motion Planning Library* (OMPL) is leveraged to implement the pursuit-evasion game algorithm. To simulate asymmetric action, agents are assigned varying degrees of asymmetric turning constraints, such that as one turn sharpens, the other broadens. Agents then compete in a pursuit-evasion game. Pursuit-evasion games are simulated across a range of asymmetric turning match-ups and agent starting positions.

Results show that pursuer success increases as its asymmetry increases. Evader success remains constant, regardless of asymmetric turning influence. Furthermore, the advantages of asymmetric turning can be further augmented when considered in conjunction with relative agent starting position. The results of this research inform more intelligent machine design strategies for vehicles in dynamic spaces.

ACKNOWLEDGMENTS

Special thank you to Professor Shell for his boundless guidance and encouragement, whose mentorship made this project a joy.

Thank you to my committee members, Professor Chaspari and Professor Chakravorty.

Thank you to Texas A&M University and the Department of Computer Science and Engineering.

And thank you to my family. Though they could not always relate to the problem, they always listened.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Dylan Shell (advisor) and Professor Theodora Chaspari of the Department of Computer Science and Engineering, and Professor Suman Chakravorty of the Department of Aerospace Engineering.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

No outside financial support was garnered for the duration of this project.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Asymmetric aircraft designs throughout history demonstrate that asymmetric machine design offers potential strategic advantages. Historical precedence suggests that motion asymmetries can be exploited for improved maneuvering through hostile space [1]. Similarly, some animals have evolved asymmetric traits to better adapt to predator-prey relationships [2] [3]. Interest in strategic autonomous path planning for adversarial scenarios has increased as robots and autonomous vehicles become more commonplace in military operations, trade, and transportation. Asymmetric turning design may offer a competitive edge hitherto unused.

Symmetry is the current standard. Quadcopter drones, for instance, are symmetrical along two axes. This design is intended for stability and easy directional shifts [4]. While aircraft of World Wars I and II featured asymmetrical turning controls as a result of rotary engine torque [1], modern engineering has aimed to overcome this, implementing design features such as counter-rotating propellers [5]. But should asymmetrical design be deserving of the push back? Accounts of the World War I aircraft, the *Sopwith Camel*, suggest otherwise. The *Sopwith Camel* had incredible turn agility to the left, and a slower turn to the right, as a result of gyroscopic forces from the rotating engine torque. While novice pilots considered this a weakness, experienced pilots considered this a feature [6]. This inspires our research question, *does asymmetric turning design affect performance?*

The scope of our study focuses on autonomous robot path planning in adversarial conditions. To demonstrate that asymmetric turning yields more strategic paths to a goal, or more advantageous evasions from an opponent, would suggest that asymmetric turning controls are a useful feature to consider in robot machine design. Military strategy and autonomous vehicle path planning can benefit from this research. Should asymmetry prove advantageous, military robots caught in adversarial airspace might leverage asymmetry to out-maneuver opponents if design decisions were made to do so. In traffic situations, autonomous cars might utilize asymmetry to avoid collision with surrounding traffic.

An agent demonstrates improved performance with asymmetric turning if it outperforms its opponent more often than with symmetrical controls. We measure this by observing the win-loss-draw frequencies for different pursuit-evasion game scenarios. Before entertaining asymmetry as an advantage, we seek to understand if it has a measurable effect on performance. To observe some change in an agent's win-loss-draw frequency when asymmetry is introduced proves that asymmetric controls change the competitive advantage for an agent in some way. Should the win frequency of an agent improve as its controls become increasingly asymmetric, then we can conclude that asymmetry is a benefit for that agent. If loss frequency increases as controls become increasingly asymmetric, then we can conclude that asymmetry is a handicap. Furthermore, knowledge is gleaned from changes in path characteristics. For example, observing how turns, evasions, and maneuvers are executed by an agent offers further insight into the influence of asymmetry.

Applying these measures of success, our research demonstrates that asymmetry has an effect on agent performance and that the effect is beneficial, or at worst neutral, to the outcome. We illustrate this by observing three game scenarios. The first models an evading craft crossing a channel with an intercepting craft approaching from the right. The second simulation mimics the first, except with the craft approaching from the left. The third scenario simulates a pursuing craft approaching the evader head on. These scenarios demonstrate the general trends of asymmetry influence, and also highlight how relative direction of the opponent impacts choice of asymmetric controls.

The thesis is outlined as follows: Chapter II provides commentary on background research, including precedents of asymmetry in nature and machines, the choice for sampling-based motion planning, descriptions of pursuit-evasion games, and an explanation of the Dubins vehicle model employed in this project. Chapter III contains a mathematical discussion on the compatibility of the optimal Dubins interval and Dubins asymmetric path solutions. Chapter IV presents the problem domain, explaining the scope of the project and its implications. Chapter V outlines the methodology, detailing the construction of the software algorithms used, and specifying experimental methods. Chapter VI presents the results. Chapter VII discusses results and further study.

II. LITERATURE REVIEW

To answer the guiding question, *does asymmetric turning design affect performance*, we survey literature over a range of topics, including asymmetric design, sampling-based motion planning, pursuit-evasion game theory, and Dubins vehicle motion constraints. In the sections below, we begin by providing analysis on established examples of asymmetry in machines and nature. Our experiment employs sampling-based motion planning, and so we explore the benefits of this planning strategy, and how it is most effective for simulating pursuit-evasion games. Next, we summarize pursuit-evasion game theory, and provide commentary on existing pursuit-evasion game research for autonomous robots. Lastly, we discuss the Dubins vehicle, which is the motion model we apply to the robots in simulation.

2.1 Asymmetric Designs in Nature and Machines

Asymmetric design has demonstrated performance advantages within both nature and machines. Morphs of the freshwater fish *Perissodus Microlepis* possess left-sloping mouths, offering predatory advantage when attacking the right side of their victims.[1] *Perissodus Microlepis* fish possessing increased mouth asymmetry have higher predation success due to the tactical advantages the trait offers [2]. Consider also the male Fiddler Crab, which develops one pincer to be significantly larger than the other. Asymmetric claws endow male Fiddler Crabs an advantage in male-male, inter-species mating competitions [3]. The asymmetry of male Fiddler crabs also extends to the morphological design of their legs [7]. Interestingly, this mobility asymmetry actually handicaps the crab in predator evasion scenarios [3], demonstrating the double-sided nature of asymmetric design.

As in nature, asymmetric machine designs have also yielded performance improvements. A renowned World War I Aircraft, the *Sopwith Camel* had a considerably more reactive left-turn than right-turn, caused by its spinning rotary engine. Because of this, pilots demonstrated a strong

[1]Some have right-sloping mouths, with a preference for attacking the left side of victims.

preference for left turns, even when the left turn angle would exceed the right turn angle. The arc length for a right turn was perceived to be too costly otherwise [8]. This ingenious quirk in design proved beneficial in battle, particularly against the German *Fokker Dr I* plane. To offset the asymmetry of the *Sopwith Camel*, Fokker pilots were actually encouraged to utilize right handed turns in evasive maneuvers [1]. Pilots' capitalization on the asymmetric operation of the aircraft highlights how motion strategies are shaped by asymmetric design. While not always as eccentric or utilized as with the Sopwith Camel, many aircraft of World Wars I and II tended to behave asymmetrically for similar reasons, favoring turns to one side over the other. The *Blohm and Voss BV 141* aircraft was considered unorthodox because of the asymmetrical body design, which was used as a means to counter this asymmetric drift caused by engine torque [9].

2.2 Sampling-Based Motion Planning

A robot's current *state* defines its current condition. For our discussion, the robot's state mainly refers to its position and orientation, (x, y, θ) within its operating space. The set of all possible robot states is the *state space*. The task of autonomous robot motion planning is to find a cost-minimal path from the start state to the goal state. Within the space of operation, the robot's environmental and kinematic constraints impose numerous complexities on this problem. The two most studied motion planning approaches are *Combinatorial Motion Planning* and *Sampling-Based Motion Planning* [10].

Combinatorial approaches are considered exact and complete, such that the algorithms of this class will definitively find a problem solution, or no solution exists [11] [10]. Combinatorial algorithms decompose the space into geometric cells. The shortest paths across each cell are determined and linked together to attain the optimal path across the space [10]. Though combinatorial algorithms can provide a complete and exact solution, they are not the best suited for a dynamic obstacle space. One shortcoming is the issue of mixed cells. If a cell straddles both an obstacle and free area of the space, it is considered mixed. If an algorithm accepts the mixed cell as valid, the returned solution may be unsound. If the cell is rejected as being in the way of an obstacle, the returned solution may be incomplete. This can be overcome by shrinking cell size and also adjusting cell geometry to more properly align with obstacles [11]. However, moving obstacles, such as a pursuing vehicle, dynamically interrupt what is considered free versus invalid motion space. This necessitates an iterative re-planning strategy that updates with every change to the obstacle space. Such algorithms are considered *resolution complete*, meaning that their completeness is dependent on the density of cells that subdivide the space.

Explicit instantiation of obstacles and space constraints can be avoided with sampling-based planners. Sampling-based planners explore the space by iteratively sampling available motions and constructing graphs between valid positions. Sampling-based planners make use of validity checking functions to avoid collisions with obstacles. The validity checking function removes from consideration samples which are invalid. The planning algorithm can therefore operate independently from knowledge of the space constraints [10].

The Rapidly-exploring Random Tree (RRT) algorithm is one such sampling-based motion planner. RRT constructs a graph data structure of random samples of agent configurations across the state space. The space of invalid states does not have to be explicitly represented. Given an initial state x_{init}, the random tree is generated as follows: a random state, x_{rand} is sampled from the space. An input u is determined that will connect x_{rand} to the nearest neighboring node in the existing tree, x_{near}. Next, u is evaluated to ensure that the new motion passes validity constraints of the space. Finally, if the motion is valid, the newly sampled state is added to the tree [12].

RRT is considered *rapidly exploring* because of its utilization of the nearest neighbors feature. It is biased towards un-visited regions of the state space because the Voronoi regions of a tree node (that is, a region of the state space such that all points in that region are closest to that tree node), are largest in unexplored space. These larger Voronoi regions are more probable to be sampled, and thus vertices with larger Voronoi regions are more likely to expand [12].

RRT* improves on the Rapidly-exploring Random Tree algorithm by ensuring an asymptotic convergence to the optimal solution. This function is achieved by re-evaluating *total* path cost of the tree for a newly added node. As in RRT, a state is randomly sampled in the space, and a nearest neighbor node to that newly sampled node is identified. In RRT*, the new edge is connected not necessarily to the absolute nearest neighbor, but to the node which results in the minimum accumulated cost from x_{start} to x_{new}. The neighborhood of vertices around x_{new} are also evaluated to determine if existing path costs can be improved by passing through x_{new}, and rewired for optimization accordingly [13]. This rewiring feature and consideration for accumulated path costs is what allows RRT* trees to converge to the optimal solution, unlike RRT.

The sampling-based nature of RRT* ensures the necessary flexibility for a dynamic pursuit-evasion environment. Valid paths for each agent are influenced by the other's motions. For instance, if the evader wants to strategically avoid capture, it must recognize states near the pursuer as invalid, but the validity of these states is constantly shifting with the movement of the pursuer. Sample-based planning better accommodates this propagation of uncertainty in state validity than combinatorial approaches.

2.3 Pursuit-Evasion Games

The mathematics and theory of Pursuit-Evasion problems have a wide array of real-world applications. 18th Century Mathematician Pierre Bouguer was concerned with determining a function for the *line of pursuit*, the curved path a pursuing navy vessel must take such that its velocity vector is always pointed directly toward the evading ship [14]. Other worthwhile scenarios include a plane crossing a channel while trying to avoid an orthogonally approaching missile, or a vessel guarding a base from an oncoming enemy. With the introduction of autonomous vehicles, the theory of pursuit-evasion games becomes particularly pertinent for military or traffic collision avoidance applications.

While details of the game setup and rules vary across different game classes, a pursuit-evasion game is a pursuing agent attempting to capture an evading agent. One form of this is the Man-Lion game. In the Man-Lion game, a pursuing lion agent tries to capture an evading man agent. Both agents are set in a closed arena, always knowledgeable of each other's position, and take turns to move. The lion wins the game if it successfully captures the man. Some versions of the game include multiple lions and obstacles [15]. Other pursuit-evasion games are sensing based. Rather than agents having constant knowledge of each other's location, they are only identified to each other when within a distance threshold [16].

For our research, the pursuit-evasion game is designed as follows: The evading agent (*evader*) executes paths towards a stationary goal point on one side of the space. The pursuing agent (*pursuer*) moves through the space attempting to intercept the evader. While not an intrinsic requirement of pursuit-evasion games, both agents plan their paths before execution, which ensures both agents are operating strategically. The evader wins the game if it reaches its goal point without being intercepted by the pursuer. The pursuer wins the game if it successfully catches the evader before the evader reaches the goal.

We employed a sampling-based algorithm for solving this class of pursuit-evasion games, as described in [17]. This algorithm is an extension of the RRT* path planning algorithm. All the same RRT* procedures are followed for general path planning with one additional condition: sampled nodes considered *near capture* are removed from the evader's tree. As the evader grows its tree, it removes all nodes that it imagines the pursuer may reach more quickly. This removal of nodes that are close to capture provides the necessary evasive action mechanism for the evader to play a strategic game.

The dynamic planning necessary for evasive maneuvers, given the uncertainty of the pursuer's motions, suits sampling-based motion planning to pursuit-evasion games particularly well. Exact algorithms based on Dynamic Programming principles have also been proposed for solving pursuit-evasion games, but suffer from extremely limited tractability to only low-dimensional configuration spaces [18]. The sampling-based algorithm utilized in [17] and this research provides a probabilistically complete solution, and is also more scalable.

2.4 Dubins Vehicles

Standard vehicles, such as planes, cars, and boats, are not designed to translate immediately perpendicular to their forward trajectory vector. This means that, if your car is facing North, and you want to reposition it ten feet to the East, you have to steer the car to face East before you can start driving East. The kinematic constraints of vehicles require them to turn through an arc to reach a position on a perpendicular axis. A Dubins vehicle models these constraints. A standard Dubins vehicle meets the following conditions [19]:

- The vehicle moves in the (x, y) plane.
- The vehicle can only move forward.
- The vehicle has constant speed.
- The vehicle has a minimum turning radius r_{min}.

Dubins vehicles are named after Lester Dubins, who solved the minimum path problem for a vehicle moving between two (x, y, θ) points [20]. The shortest path for a Dubins vehicle is called a *Dubins curve*. Dubins proved that there are only six possible path shapes, each of which are a combination of three segment types. A Dubins path consists of *Left* (L), *Right* (R), and *Straight* (S) path segments. A left or right segment indicates a steering control in that direction, and a straight path is a translation that maintains the same velocity vector orientation angle. The shortest path between any two (x, y, θ) points for a Dubins vehicle will be one of the following segment combinations: RSL (Figure 2.1), RSR (Figure 2.2), LRL (Figure 2.3), RLR, LSL, LSR, or a subset thereof.

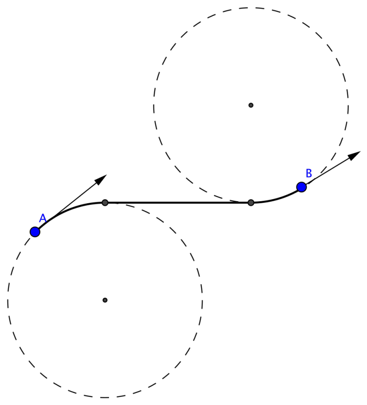


Figure 2.1: Right-Straight-Left (RSL) Dubins path. Reprinted with permission from [21].

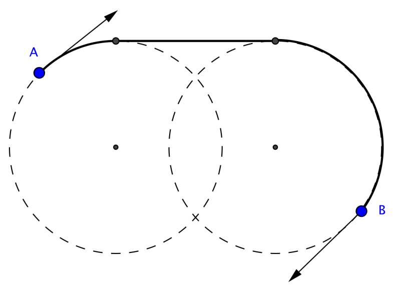


Figure 2.2: Right-Straight-Right (RSR) Dubins path. Reprinted with permission from [22].

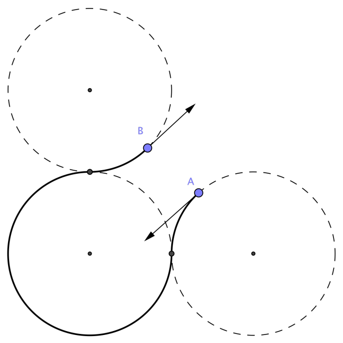


Figure 2.3: Left-Right-Left (LRL) Dubins path. Reprinted with permission from [23].

An asymmetric Dubins vehicle follows the same constraints, but now with differing r_{min} for left and right turns. In this case, the shortest path will still be one of the six standard cases, however the minimal path solution under asymmetric conditions may be a different solution than under symmetric conditions for the same start-endpoint pair, as shown in [24].

Applying Dubins vehicles within a pursuit-evasion game presents additional complexities surrounding angle of approach. In the standard Dubins path planning, the goal point has a specific orientation of rotation that must be met. To avoid capture within pursuit-evasion games, agents must often whiz past goal points and return at a more advantageous moment later in the game. Given the dynamic nature of the path planning constraints, it becomes impractical to apply strict goal arrival orientations. Rather, agents need to be allowed to approach the goal point from any direction.

The interval Dubins case describes a Dubins vehicle which is allowed an interval of arrival orientations at the goal point. It is necessary to introduce this path planning case when an agent approaches the goal point to expand degrees of motion freedom necessary for success. Solutions for the shortest paths with a range of allowable goal arrival angles is presented in [25]. We demonstrate the compatibility of the asymmetric and interval cases in the following chapter.

III. COMPATIBILITY OF DUBINS INTERVAL AND ASYMMETRIC PATH SOLUTIONS

Combining the Dubins interval and Dubins asymmetric cases has, to our knowledge, not been previously studied. Here, we show mathematically that our algorithm's incorporation of solutions from [24] and [25] is sound. This chapter proves the following proposition:

Proposition 1: *The optimal path between two points with defined departure angle θ_0 at starting position and terminal angle $\theta_f \in [0, 2\pi]$ at final position is of the form LS, RS, LR, RL, LRL, RLR, or a subset thereof, for a Dubins vehicle with differing left/right turn radii ρ and ϱ.*

The optimal path problem for an asymmetric Dubins vehicle with an arrival angle interval of $360°$ is characterized as follows: we want to minimize the total length of the vehicle path from initial position (x_0, y_0, θ_0) to final position (x_f, y_f, θ_f), where x and y are 2D Cartesian coordinates, and θ is the angle of orientation. The arrival orientation $\theta_f \in [0, 2\pi]$. Let $u(t)$ be the vehicle control input at time t. The differential system of equations for Dubins vehicle motion is

$$\dot{x} = \cos\theta,\ \dot{y} = \sin\theta,\ \dot{\theta} = u(t) \tag{3.1}$$

For the asymmetric case, $u(t) \in [-1/\rho, 1/\varrho]$, where ρ is minimum turn radius to the right and ϱ is minimum turn radius to the left.

Let $\Lambda = (\lambda_x(t), \lambda_y(t), \lambda_\theta(t))$ be the adjoint variables associated with $p(t) = (x(t), y(t), \theta(t))$. Then, the Hamiltonian equation for this system is:

$$H(\Lambda, p, u) = e + \lambda_x \cos(\theta) + \lambda_y \sin(\theta) + \lambda_\theta u \tag{3.2}$$

and the adjoint variable differential equations are:

$$\begin{aligned}\dot{\lambda_x} &= 0, \\ \dot{\lambda_y} &= 0, \\ \dot{\lambda_\theta} &= \lambda_x \sin(\theta) - \lambda_y \cos(\theta).\end{aligned} \tag{3.3}$$

As demonstrated in [26], we can transform the Hamiltonian equation as follows for easier interpretation. Let $\lambda_x = \psi \cos(\phi)$, $\lambda_y = \psi \sin(\phi)$, where $\psi = \sqrt{\lambda_x^2 + \lambda_y^2}$ and $\tan(\phi) = \lambda_y / \lambda_x$. We can then rewrite the Hamiltonian equation and adjoint variable equations as:

$$H(\Lambda(t), p(t), u(t)) = e + \psi \cos(\theta - \phi) + \lambda_\theta u \tag{3.4}$$

$$\dot{\lambda_\theta} = \psi \sin(\theta - \phi) \tag{3.5}$$

By Pontryagin's Minimum Principle, when executing the optimal action trajectory $u^*(t)$,

$$H(\Lambda(t), p(t), u^*(t)) \equiv 0, \tag{3.6}$$

and

$$e \in \{0, 1\} \tag{3.7}$$

for all $t > 0$. Additionally,

$$u^*(t) = \arg\min_u \{H(\Lambda(t), p(t), u(t))\}. \tag{3.8}$$

Equation 3.8 states that the Hamiltonian is minimized at $u(t) = u^*(t)$. In order to satisfy equations 3.6 and 3.7 for an optimal control along any optimal path,

$$\psi \cos(\theta - \phi) \leq 0 \text{ and } \lambda_\theta u \leq 0. \tag{3.9}$$

The following three lemmas are presented in [25] and [26] assuming symmetrical turning. It is shown here that these lemmas are also true when turning is asymmetrical.

Lemma 1: $\lambda_\theta = 0$ *at all path inflection points and at all points along straight (S) path segments.*
Proof: First we will prove that $\lambda_\theta = 0$ for all points along a straight path segment. $\frac{\partial H}{\partial u} = \lambda_\theta$. Pontryagin's minimum principle states that the adjoint vector Λ must be nonzero. In the case that $\frac{\partial H}{\partial u} \equiv 0$, we recognize that it is therefore impossible for $\psi = 0$, because otherwise $e = 0$ by virtue of equation 3.4, and $\Lambda = 0$.

Eliminating the possibility that $\psi = 0$, let's consider the case that $\psi \neq 0$. Consider equation 3.5. If $\dot{\lambda_\theta} = 0$, then it must be that $\theta = \psi$ or $\theta = \psi + \pi$, and the path is a line segment with ϕ as the direction. Because λ_θ is continuous, this argument extends to saying that $\lambda_\theta = 0$ at inflection points between straight segments and curved segments.

For inflection points between two curved segments, we recognize the $\dot{\theta}$ will change sign. Therefore, λ_θ must also change sign by equation 3.9 and 3.1.

Lemma 2: *For any optimal path,* $\lambda_\theta - \lambda_x y + \lambda_y x = k$ *where* k *is a constant. Furthermore, all line segments and inflection points for an optimal path lie on the same straight line.*
Proof: Plugging in the definitions from equations 3.1 and 3.3, we see that $\dot{\lambda_\theta} - \lambda_x \dot{y} + \lambda_y \dot{x} = (\lambda_x \sin(\theta) - \lambda_y \cos(\theta)) - \lambda_x(\sin(\theta)) - \lambda_y(\cos(\theta)) = 0$. Integrating this expression with respect to time, we have $\lambda_\theta - \lambda_x y + \lambda_y x = k$.

If $\lambda_\theta = 0$, then $\lambda_y x - \lambda_x y = k$. Rearranging this, we more clearly see it is the equation for a line, $y = \frac{\lambda_y}{\lambda_x} x - k$.

Lemma 3: *Any curved optimal path between two inflection points where* $\lambda_\theta = 0$ *turns through* $> \pi$ *radians.*
Proof: Let t_1 and t_2 represent the times corresponding to the two inflection points. Consider if path arc angle were $\leq \pi$. $\lambda_\theta = 0$ at all inflection by lemma 1, so $\lambda_\theta(t_1) = \lambda_\theta(t_2) = 0$. By equation

3.4 and 3.6,

$$\cos(\theta(t_1) - \phi) = \cos(\theta(t_2) - \phi) = -e/\psi \tag{3.10}$$

Along any curved path, $\text{sign}(\dot{\theta})$ is constant. Therefore, $\text{sign}(\lambda_\theta)$ is also constant by virtue of equation 3.9 and 3.1. So, λ_θ achieves a minimum or maximum somewhere along the arc. Let this extremum be noted as t_3, and equation 3.5 gives us

$$\dot{\lambda_\theta}(t_3) = \psi \sin(\theta(t_3) - \phi) = 0. \tag{3.11}$$

Equation 3.11 tells us that either $\sin(\theta(t_3) - \phi) = 0$, or that $\psi = 0$. Equation 3.10 becomes undefined if $\psi = 0$, so it must be that $\sin(\theta(t_3) - \phi) = 0$. Furthermore, $\sin(\theta(t_3) - \phi) = 0$ when $\cos(\theta(t_3) - \phi) = 1$ or -1.

We know it is the case that $\cos(\theta(t_3) - \phi) = -1$ by considering the following: equation 3.10 indicates that $\cos(\theta - \phi)$ has the same sign at t_1 and t_2. Because we have assumed that the arc angle will be $\leq \pi$, then $\cos(\theta - \phi)$ will also have the same sign at t_3. Both e and ψ are positive by definition, so therefore the $\text{sign}(\cos(\theta - \phi)) = -1$ all along the arc, and $\cos(\theta(t_3) - \phi) = -1$.

Substituting -1 for $\cos(\theta(t_3) - \phi)$ into equation 3.4, we have

$$e - \psi + \lambda_\theta(t_3)u(t_3) = 0. \tag{3.12}$$

Remembering that both e and ψ must be ≥ 0 by definition, then it must also be true that

$$e - \psi + \lambda_\theta u \leq e - \psi. \tag{3.13}$$

Then,

$$0 \leq e - \psi \implies \frac{e}{\psi} \geq 1. \tag{3.14}$$

Equation 3.10 implies that $e/\psi = 1$, and so $e = \psi$.

Now, from equation 3.12, it must be that:

$$\lambda_\theta(t_3)u(t_3) = 0. \tag{3.15}$$

Because $|u^*(t_3)| > 0$, then $\lambda_\theta(t_3) = 0$. Using equation 3.5,

$$\psi \sin(\theta - \phi) = k \tag{3.16}$$

where k is a constant, and

$$\theta = \arcsin(\frac{k}{\psi}) + \phi, \tag{3.17}$$

which is also a constant. Because θ is constant, this path must be line segment, and not an arc. Therefore, our assumption that the curved path arc angle can be $\leq \pi$ does not hold, and the arc angle of a curved segment must be $> \pi$.

Following the presentation of these three lemmas, let us return our consideration to proposition 1. For the interval $[0, 2\pi]$, $\lambda_\theta(T) = 0$, where T is the time at which the vehicle arrives at the destination point, as shown in [25]. Let us first consider $\lambda_\theta(0) \neq 0$. By lemma 1 and lemma 2, the first segment in the path is a Left or Right curve (L or R), and the second segment in the path is a straight line (S). If $\lambda_\theta(0) = 0$, then an optimal path may be a single straight line segment from lemma 2, or a series of curved segments each with arc angle greater than π for each curve from lemma 3.

Therefore, the pursuit-evasion algorithm evaluates LS, RS, LR, and RL paths in addition to all original Dubins path shapes, as discussed further in Chapter 5 and shown in algorithm 4. As shown in [25], the symmetric Dubins case further bounds the optimal path shapes to only curve-straight (CS) and curve-curve (CC) segment combinations. The proof of this relies on the condition that three curve segments of length πR (where R is the turn radius) cannot be an optimal path, as shown in [25] and [27]. For an asymmetric Dubins vehicle, [24] applies an upper bound on the third curve length of an optimal path greater than πR. We leave it as future work to show that the optimal path may be further bounded to CC or CS shapes for an asymmetric Dubins vehicle.

IV. PROBLEM STATEMENT

Pilots of the *Sopwith Camel* aircraft reported that its asymmetric turning feature was a genuine advantage when intelligently utilized [1]. We seek to better understand this phenomenon by studying agent path planning strategies within a pursuit-evasion game. The competitive nature of pursuit-evasion games allows us the opportunity to examine how different asymmetric turning strategies fare against each other, and if any are more or less advantageous.

Asymmetric turning constraints are modeled by a *turn budget*. The *turn budget* value represents the total units of turn radius an agent is allowed across its right and left directions. If a smaller turn radius is given to the left, then a larger turn radius is required on the right, such that the total of the right and left turn radii equals the turn budget. In a game scenario, both agents are given the same turn budget. This ensures that they both maintain a comparable range of motion. This model mimics the effects of asymmetric aircraft. In the case of the Sopwith Camel, the effects of the engine torque sharpened left turns, but at the cost of broader right turns [1]. This cost trade off between a sharp turn and broader turn makes the asymmetric case particularly interesting. At a given juncture, the agent must consider if an agile evasive maneuver with its sharper turn is more effective, even if it takes it away from the goal. On the other hand, is a broader turn in the correct direction the better option, even if the agility cost risks capture? We will explore how having both a broader and sharper turn can be useful.

Vehicles are simulated in a 2-dimensional plane. This is sufficient for modeling left-right turn actions for cars, boats, and aircraft. Aircraft have additional axes of motion which could expand the problem space further, such as rolling or pull up maneuvers. Left-right turning in 2D simplifies the model space to observe more directly the effects of asymmetry as the binary choice between a sharper or broader turn.

Pursuit-evasion provides a dynamic interaction space to observe asymmetric strategies. A pursuing agent pressures the evader to choose its turns wisely, as a choice of one turn versus the other could place the evader suddenly at risk, or sweep it away from danger. Evaluating which decisions lead to better outcomes allows us to understand how asymmetric turning can be utilized by an agent, and reveal the strengths and weaknesses of the scheme. We focus specifically on adversarial, pursuit-evasion games for this reason. However, we can glean knowledge about non-adversarial path planning conditions as well. Our results observe the general path planning characteristics of different asymmetry settings, the patterns of which are translatable to peaceful navigation conditions.

To gauge the effectiveness of an agent's strategy, we record the agent's wins, losses, and draws over a series of games under the same conditions. These game outcomes are then analyzed to determine which agent demonstrated a statistically significant advantage for a given asymmetric condition. The pursuit-evasion game variant used in this research employs a specific goal point for the evader so that we can measure both the evader and pursuer's success.

Space parameters include agent velocity, starting position, and planning time. To focus specifically on asymmetric turning effects, velocity and planning time are held constant throughout all experiments. Starting positions are varied intentionally in some experiments to explore how the relative positions between agents influences which strategies are more effective.

We implement the sampling-based pursuit-evasion algorithm posed in [17]. Agents are modeled as Dubins vehicles with asymmetric turning radii. Using the methods explained in [24], we develop an asymmetric Dubins vehicle state space within which the pursuit-evasion game can operate. Furthermore, we implement special Dubins interval cases outlined in [25] to allow agents to reach the goal position from any orientation. All games are simulated in software.

V. EXPERIMENTAL METHODS

We designed a computer program to simulate sample-based pursuit-evasion games with variable turn symmetry conditions. In addition to providing statistical data for simulated games, such as agent win rates, the program also outputs animations and illustrations of the game paths taken by each agent. The software was designed in C++ and run on a Linux machine with Ubuntu 18.04 operating system.

A single pursuit-evasion game is run as follows: input parameters are provided, such as start position, velocity, and planning time. The pursuer takes the first turn, planning a path to intercept the evader. During the pursuer planning phase, the pursuer also extends a tree of paths that it *imagines* the evader will take. These paths are not necessarily the paths the evader will plan or execute, but approximations that the pursuer predicts will happen. Once the planning time expires, the pursuer selects the most optimal path motion to execute.

With both agents still in their starting positions, the evader plans its move. The evader generates a tree of paths that will move it closer to its goal point. At the same time, the evader generates a tree of paths it *imagines* the pursuer will execute. Based off of this believed pursuer tree, the evader will eliminate nodes from its path that it considers too close to capture, as described in [17]. Once planning time expires, the evader selects the most optimal path motion to enact.

After both agents have planned, they move to their next positions simultaneously. Goal checking functions evaluate if the pursuer successfully intercepted the evader, or if the evader successfully reached the goal position. If neither agent achieved its goal, their positions are established as their new start states to provide the planner, and the process repeats.

The following sections detail the code architecture used for the pursuit-evasion algorithm and simulation software. Following that, we present an outline of experimental procedures, detailing simulation parameters and game scenarios used.

5.1 GamePlan Module

The **GamePlan** module handles input parameters and calls to the planning algorithm. The module has flexibility for a wide range of game scenario parameters. The input parameters for a single pursuit-evasion game are:

- **Plan Time**: amount of time an agent has to plan its next motion.
- **Velocity**: The speed at which an agent moves. The evader and pursuer velocities are independently defined. Each executed motion is considered a single unit of time, so this value is equivalent to the distance an agent moves each turn.
- **Turn Budget**: The total amount of turn radius allotted to each agent. This budget is divided between the right and left turn radii based on the asymmetry split parameter.
- **Asymmetry Split**: Defines how much of the turn radius budget is given to the right or left turn radius. This is measured as percentage given to left turn radius. Consider a turn budget of 10 units. An asymmetry split value of 0.6 indicates that the left turn radius will be 6 units and the right turn radius will be 4 units. Asymmetry split is defined independently for the evader and pursuer.
- **Goal Bias**: Percentage of motion samples for which the goal point is sampled.
- **Start Position**: Agent start position, provided as a coordinate (x, y, θ), where theta is the angle of orientation.
- **Goal**: The stationary goal point that the evader is attempting to reach, provided as a coordinate (x, y).

This module also executes the possible experiment routines, which include:

- **run_single_trial**: runs a single pursuit-evasion game for a given set of input parameters. Outputs visualizations of the agent paths.

- **calculate_asymmetric_influence**: All input parameters are defined by the user except *asymmetry split*. The program iteratively simulates pursuit-evasion games with a range of different asymmetry match ups. Each asymmetry match up is repeated a number of trials, and win rates for each match up are recorded for analysis. Outputs win rate statistics and visualizations of agent paths.

Users input the scenario parameters and select the experiment routine they wish to run. After instantiating the game state space and planner objects, the **GamePlan** module will simulate the game scenarios requested, and output experiment data.

5.2 Overview of the Open Motion Planning Library

The *Open Motion Planning Library* (OMPL) was leveraged as the foundational code base for the pursuit-evasion planner [28]. OMPL is an open source code base, distributed by Rice University, that contains code for standard robotics motion planners, including RRT*. The library's planners are originally designed to account for only a single agent. Therefore, the first step was to expand on existing code structures to allow separate object tracking for two robots.

The OMPL Application Programming Interface (API) is illustrated in Figure 5.1. The **GamePlan** module is represented by the *user code* box. Because each agent has individual turn-symmetries, this required that each agent have their own defined **StateSpace** and **MotionValidator** classes. These classes, in particular the **StateSpace** class, define the available motions an agent can implement. The **StateSpace** class describes all available states that the planner can sample, the distances between states, and agent properties at each state. The **MotionValidator** checks that paths between states do not break kinematic or obstacle constraints. With these individual state spaces, a **SpaceInformation** class was instantiated for each agent to manage their individual controls. The **SpaceInformation** class is the central hub object processing foundational motion planner routines, such as the parameters of the state space and validity of agent motion. The class includes essential functions such as setting the state space type (such as SE2, SE3, or Dubins) and allocating states.

The **ProblemDefinition** class manages start and goal states, defines what the planner solution looks like, and monitors if a planner solution has been found. For a pursuit-evasion game,

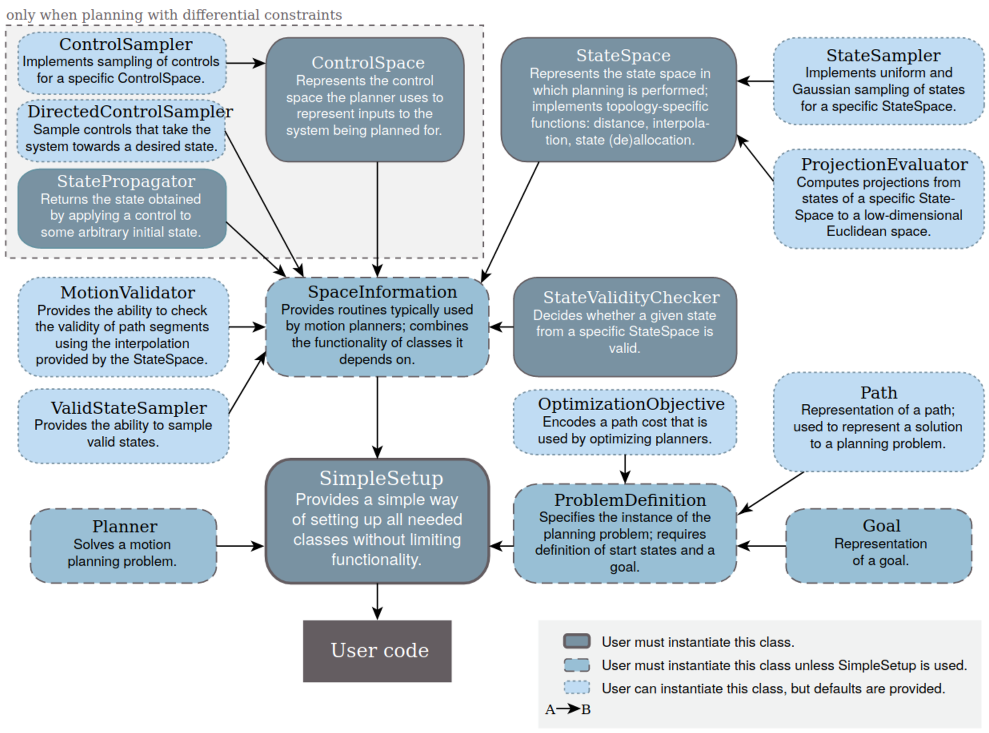

only when planning with differential constraints
ControlSampler
Implements sampling of controls for a specific ControlSpace.
DirectedControlSampler
Sample controls that take the system towards a desired state.
StatePropagator
Returns the state obtained by applying a control to some arbitrary initial state.
ControlSpace
Represents the control space the planner uses to represent inputs to the system being planned for.
StateSpace
Represents the state space in which planning is performed; implements topology-specific functions: distance, interpolation, state (de)allocation.
StateSampler
Implements uniform and Gaussian sampling of states for a specific StateSpace.
ProjectionEvaluator
Computes projections from states of a specific StateSpace to a low-dimensional Euclidean space.
MotionValidator
Provides the ability to check the validity of path segments using the interpolation provided by the StateSpace.
SpaceInformation
Provides routines typically used by motion planners; combines the functionality of classes it depends on.
StateValidityChecker
Decides whether a given state from a specific StateSpace is valid.
ValidStateSampler
Provides the ability to sample valid states.
OptimizationObjective
Encodes a path cost that is used by optimizing planners.
Path
Representation of a path; used to represent a solution to a planning problem.
SimpleSetup
Provides a simple way of setting up all needed classes without limiting functionality.
Planner
Solves a motion planning problem.
ProblemDefinition
Specifies the instance of the planning problem; requires definition of start states and a goal.
Goal
Representation of a goal.
User code
User must instantiate this class.
User must instantiate this class unless SimpleSetup is used.
User can instantiate this class, but defaults are provided.
A→B

Figure 5.1: Class ownership diagram for the Open Motion Planning Library. Each major class is described. A → B means that object B owns object A. Reprinted from [29].

the addition of a second agent constitutes additional definitions of game termination conditions. Therefore, the **ProblemDefinition** class was expanded to manage separate evader and pursuer win conditions. Functions were created to define a pursuer start state, evader start state, pursuer goal state and evader goal state. Separating out the definition of these states then allowed the **Planner** class to feasibly monitor each agent's starting and win condition independently.

5.3 Planner

The **Planner** class is where the pursuit-evasion algorithm is implemented. The **Planner** class runs the pursuit-evasion game and returns the solution. With RRT* as the basis for planning, we extended the RRT* planner available in OMPL to include the additional pursuit-evasion functions described in [17]. The planner algorithm is described in Algorithm 1. This follows closely with the pseudocode outlined in [17], except that variable names have been adjusted to follow nomenclature in this discussion, such as the use of the *plan_time* parameter.

Algorithm 1: Planner Algorithm

```
$V_e \leftarrow (z_{e,start}); E_e \leftarrow 0;$
$V_e \leftarrow (z_{p,start}); E_p \leftarrow 0;$
while time < plan_time do
    $G_e \leftarrow (V_e, E_e); G_p \leftarrow (V_p, E_p);$
    $z_{e,rand} \leftarrow$ Sample$_e()$;
    $(V_e, E_e, z_{e,new}) \leftarrow$ Extend$_e(G_e, z_{e,rand})$;
    if $z_{e,new} \neq NULL$ then
        $Z_{p,near} \leftarrow$ NearCapture$_e(G_p, z_{e,new}, V_p)$;
        for all $z_{p,near} \in Z_{p,near}$ do
            if time$(z_{p,start}, z_{p,near}) \leq$ time$(z_{e,new}, z_{e,start})$) then
                Remove$(G_e, z_{e,new})$;
    $z_{p,rand} \leftarrow$ Sample$_p()$;
    $(V_p, E_p, z_{p,new}) \leftarrow$ Extend$_p(G_p, z_{p,rand})$;
    if $z_{p,new} \neq NULL$ then
        $Z_{e,near} \leftarrow$ NearCapture$_p(G_e, z_{p,new}, V_e)$;
        for all $z_{e,near} \in Z_{e,near}$ do
            if time$(z_{p,start}, z_{p,new}) \leq$ time$(z_{e,near}, z_{e,start})$) then
                Remove$(G_e, z_{e,near})$;
return $G_e, G_p$
```

After trees are generated using Algorithm 1, we determine which path will achieve the goal motion with the least cost. This is done using Algorithm 2. The **bestGoalMotion** function takes as input the agent's goal position and the tree G, then outputs the leaf node that most closely achieves the goal. If any of the nodes in the tree are in the goal position, these are saved in the *goalMotionVect* vector. These motions are then compared to determine the least cost path option between them. If there are no nodes that achieve the goal, then *approxGoalMotion* is returned, representing the node that is the closest distance to the goal. The program traces the end node back to the starting position to generate the agent's optimal path for that turn. The original OMPL code included a version of this algorithm for RRT* path planning. It was generalized to work for both the evader and pursuer trees.

Unlike the evader, which is path planning towards a static goal, the pursuer's goal is dynamic. Therefore, it is to the pursuer's advantage to sometimes select a path towards a projected goal point. If the pursuer always plans towards the evader's current position, it is handicapped by being one

Algorithm 2: bestGoalPath(G, goal_position)

```
$z_{goal} \leftarrow goal\_position$; $goalMotionVect \leftarrow \emptyset$;
$bestGoalMotion \leftarrow \emptyset$; $bestCost \leftarrow \emptyset$;
$approxGoalMotion \leftarrow \emptyset$; $distanceFromGoal \leftarrow \infty$;
$approxDist \leftarrow 0$;
for all $V \in G$ do
    if $V = z_{goal}$ then
        $goalMotionVect \leftarrow V$
    if !bestGoalMotion and goalMotionVect $\neq \emptyset$ then
        $bestGoalMotion = goalMotionVect.$front();
        $bestCost = bestGoalMotion.$cost();
    else
        for $motion \in goalMotionVect$ do
            if motion.cost() < bestCost then
                $bestGoalMotion = motion$;
                $bestCost = bestGoalMotion.$cost();
    if $goalMotionVect = \emptyset$ and $distanceFromGoal < approxDist$ then
        $approxDist = distanceFromGoal$;
        $approxGoalMotion = V$;
if bestGoalMotion then
    return $bestGoalMotion$;
else
    return $approxGoalMotion$;
```

step behind the evader. To address this, an α parameter is applied to the pursuer's choice of goal motion. This operation is illustrated in Algorithm 3. The value for α is generated from a uniform distribution. Fifty percent of the time, the pursuer will choose to move towards the evader's current position. The rest of the time, the pursuer will move towards the evader's predicted motion.

Algorithm 3: Pursuer Alpha Parameter

```
$\alpha =$ generateRandomVal();
if $\alpha < 0.5$ then
    $GoalMotion_{pursuer} =$ bestGoalPath$(G_p, GoalMotion_{evader})$;
else
    $GoalMotion_{pursuer} =$ bestGoalPath$(G_p, z_{e,start})$;
```

Both the pursuer and evader agents run identical copies of the planner object. The tree returned for the opposing agent is what the planning agent uses as its imagined opponent path. In this way, both agents plan under the assumption of an optimally strategic opponent.

5.4 Dubins Asymmetric State Space

OMPL includes a standard Dubins State Space that determines motions according to the standard Dubins vehicle described in [20]. For asymmetric vehicles, this state space required reworking to allow variable turn radius to the left and right side.

The state space space inherits the foundational attributes and functions of an SE2StateSpace so that basic concepts like (x, y θ) coordinates and space boundaries are maintained. The Dubins state space overrides the **distance**() function to return Dubins distances. When the state space is queried for the distance between two states, the DubinsStateSpace evaluates the Dubins path options (LSL, RSR, LSR, RSL, RLR, LRL), and returns the shortest path distance of the six possibilities. Additionally, the nature of pursuit-evasion games necessitates that the goal point be approached from any angle. Overly limiting the goal orientation severely handicaps an agent's ability to path plan towards the goal in the dynamic environment. Therefore, if the end point of a given path is equal to the goal, we return the shortest path to the Dubins interval cases [25], with the arrival orientation interval set to 360 degrees. The Dubins distance operations are illustrated in Algorithm 4. If the final path state z_{end} is equivalent to the goal state, then we test the interval Dubins path options in addition to the standard options to determine the shortest path.

5.5 Path Visualization

Paths were visualized using Python's matplotlib library [30]. The pursuit-evasion game algorithm outputs ascii text files containing each sampled motion in a path, as well as interpolated points between the sampled motions to visualize Dubins curves. The NumPy library was used to sort, analyze, and manage data for plotting [31]. We analyzed both animations and static visualizations of the complete paths taken by the pursuer and evader agents for select games.

Algorithm 4: DubinsDistance

```
$tmp \leftarrow \emptyset; path \leftarrow \emptyset$;
$pathSet \leftarrow [RSR, LSL, LSR, RSL, RLR, LRL]$;
$len, minlength \leftarrow \infty$ ;
for $path \in pathSet$ do
    $tmp \leftarrow path$;
    $len \leftarrow tmp$.length();
    if $len < minLength$ then
        $minLength = len$;
        $path = tmp$;
if $z_{end} = goal$ then
    $pathSet \leftarrow [LR, RL, RS, LS]$;
    for $path \in pathSet$ do
        $tmp \leftarrow path$;
        $len \leftarrow tmp$.length();
        if $len < minLength$ then
            $minLength = len$;
            $path = tmp$;
```

5.6 Experimental Procedure

We executed the pursuit-evasion game algorithm across a range of agent asymmetry settings and examined how performance was affected. The main metric of performance improvement used was *game win rate*, which is the proportion of games won by a particular agent for a given parameter input set. Win rate results were modeled as Bernoulli distribution samples for statistical analysis.

In our first experiment, we examined the scope of asymmetry permutations. In this experiment, the simulation was run with a discrete series of *asymmetry_split* value pairs, 0.1 to 0.9. For a given simulation, each agent was assigned an *asymmetry_split* value, and the simulation run 250 times. Every combination of *asymmetry_split* value pair was tested in the set {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}. Win rates for evader and pursuer were recorded for each permutation, and plotted in a 3D bar graph to examine trends. Visualizations of select games were also generated from this experiment to examine qualitative characteristics about executed paths. With the exception of *asymmetry_split*, all other game parameters were held constant during these trials. The following game parameters were used:

- **Plan Time**: 0.01 seconds
- **Velocity, Evader**: 40 distance units
- **Velocity, Pursuer**: 40 distance units
- **Turn Budget**: 30 distance units
- **Goal bias**: 10%
- **Space Dimensions**: 200 distance units $\times$ 200 distance units

Next, we studied how agent start orientation influences the asymmetry strategy. In the first trial, the evader was positioned to the left of the pursuer at the start of the game. In the second trial, the evader was positioned to the right of the pursuer at the start of the game. In the third trial, the two agents faced each other directly. Figure 5.2 illustrates these game setups. Games were run 1000 times per asymmetry permutation. We compared results from both trials to examine if the most effective asymmetry strategy shifted based on relative agent position.

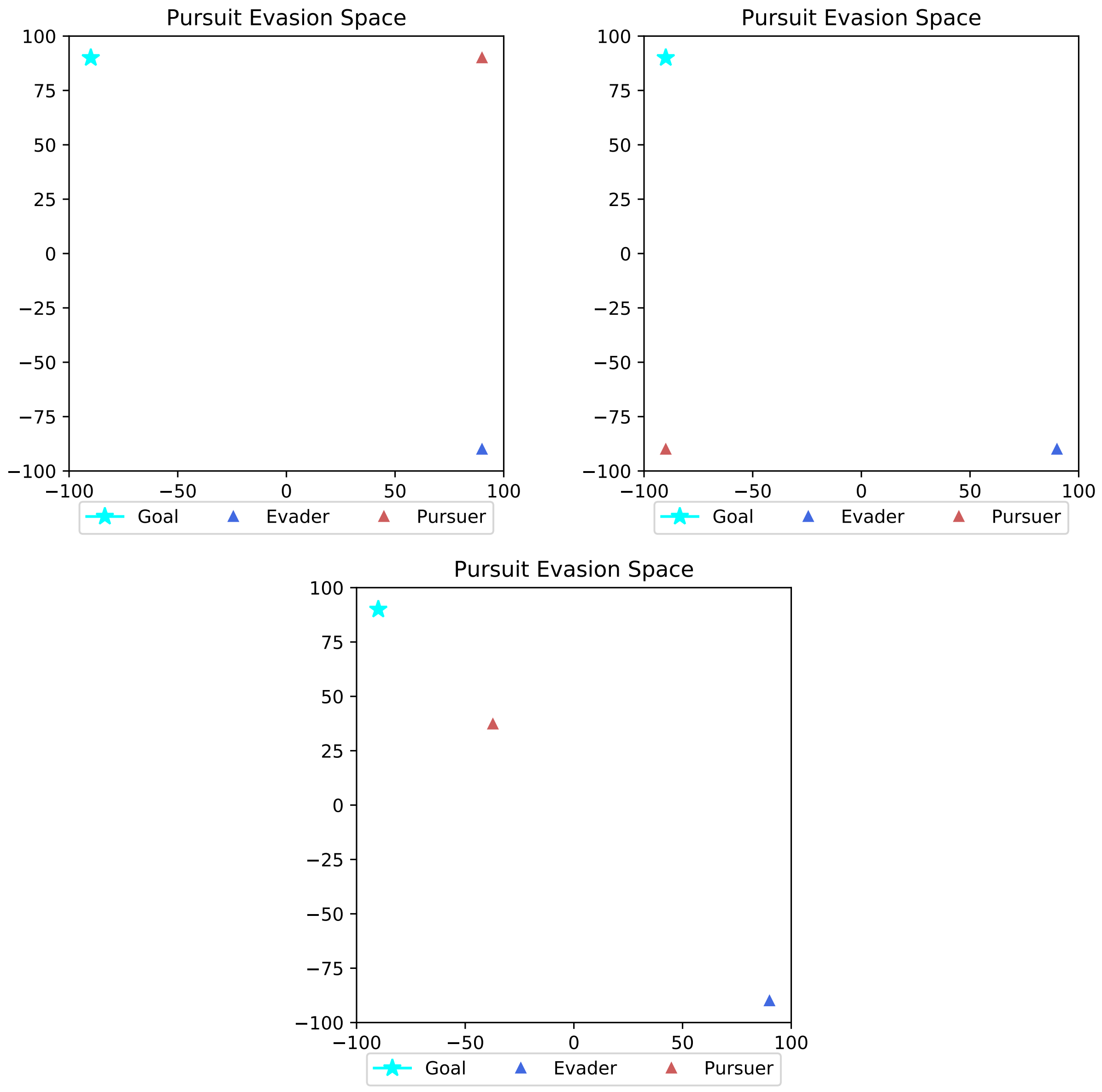


Figure 5.2: Experiment starting positions.

VI. RESULTS

In this chapter, we present data and experimental results that show the following:

1. The introduction of asymmetric turning has an affect on pursuit-evasion game outcome.
2. The pursuer gains the most advantage when given asymmetric turning, whereas the evader has minimal performance improvement.
3. Variability in scenario starting positions affect which asymmetric turning conditions are most advantageous to the agents.

6.1 The Influence of Asymmetry

We begin by considering the game illustrated in Figure 6.1. Pursuit-evasion games with symmetrical turning for both agents are illustrated in Figure 6.2. All figures shown were generated using our pursuit-evasion game algorithm. This game setup models an evading aircraft crossing a channel, with a pursuing aircraft attempting to intercept the evader perpendicularly. We use this particular game scenario as our benchmark scenario for analysis. This is a valuable game setup for comparison for a few reasons. Firstly, it is modeled after a real life pursuit-evasion scenario: that of a plane avoiding an intercepting craft. For instance, a plane crossing the English channel from a base in Paris to a base in London, while avoiding an oncoming missile from the German border, would have a similar geometric setup. Secondly, the pursuer and evader begin on either side of one another. Later demonstrations will illustrate how different starting sides influence trends. Lastly, this particular scenario has the goal and start positions sufficiently spread out so as to easily visualize path characteristics.

In all figures, blue represents the evader, red represents the pursuer, and the star is the evader's goal point. Triangles indicate the agent's start and end positions. Numbered positions indicate the sampled motions over time. For instance, both agents begin at their respective position *0*. After each has the opportunity to plan their next move, they synchronously move to their respective position *1*, and so on.

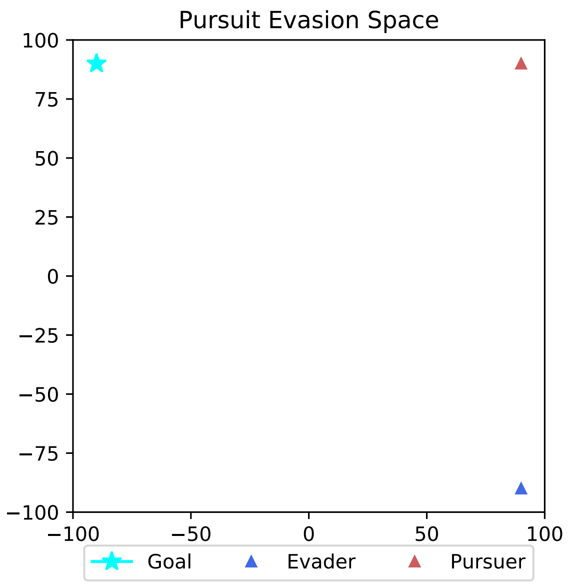
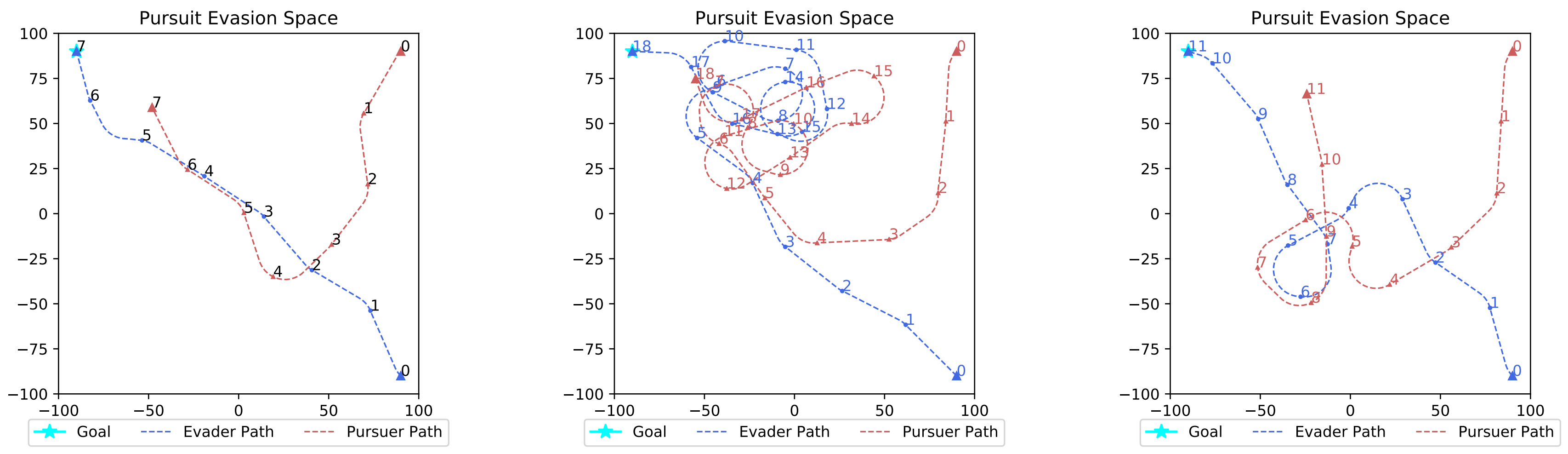
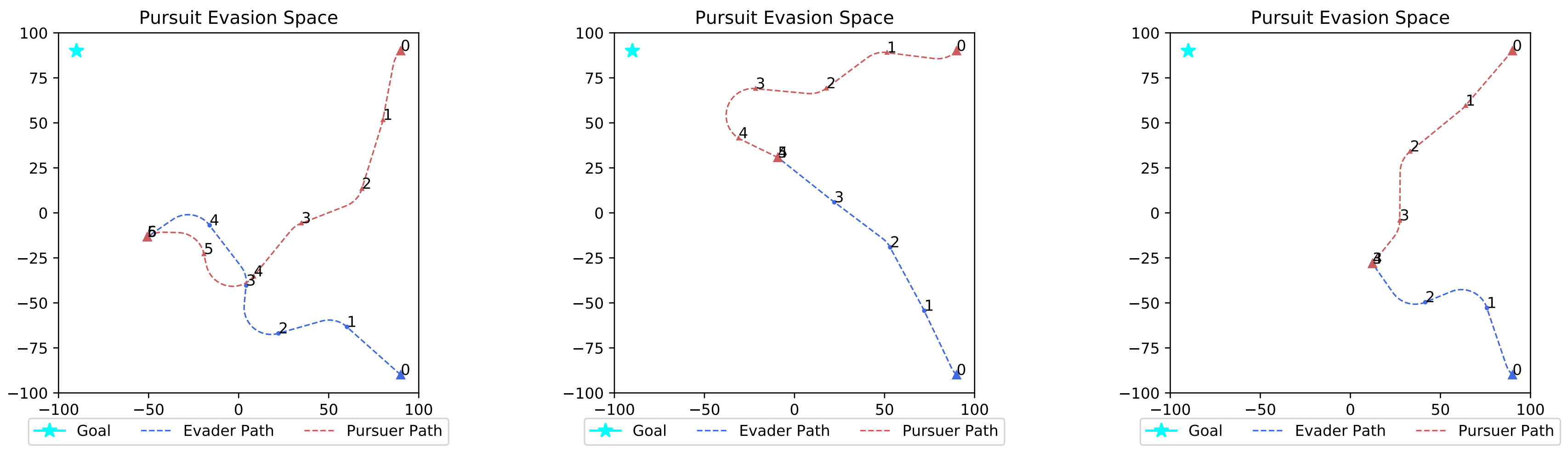


Figure 6.1: Game setup with evader on left side of the pursuer.



(a) Symmetric pursuit-evasion game with evader win.



(b) Symmetric pursuit-evasion game with pursuer win.

Figure 6.2: Pursuit-evasion games with symmetric agents.

Next, we introduce asymmetric turning into the game. Figures 6.3 and 6.4 illustrate how win rates for each agent differ across the complete range of turn symmetries. Evader and pursuer turn symmetry values, notated along the x and y axes, are written as the percent change from symmetrical turning. The transformation from right turn radius to turn symmetry value is

$$TurnSymmetry = (r_{right} - r_{sym})/r_{sym} \times 100, \tag{6.1}$$

where r_{right} is the right turn radius and r_{sym} is the radius value at which the turn budget is split symmetrically between right and left turns. Similarly, the transformation from left turn radius to turn symmetry value is

$$TurnSymmetry = (r_{sym} - r_{left})/r_{sym} \times 100, \tag{6.2}$$

where r_{left} is the left turn radius. Both equation 6.1 and 6.2 are equivalent. A clear interpretation is this: positive turn symmetry values indicate more turn radius provided to the right side (broader right turn, sharper left turn), and negative turn symmetry values indicate more turn radius provided to the left side (broader left turn, sharper right turn). Agents have equivalent left and right turn radii when turn symmetry value is 0.0.

Upon observing Figures 6.3 and 6.4, notice that the surfaces are not flat. If the surfaces demonstrated negligible change along the z-axis, that would suggest that asymmetric turning has negligible influence on agent performance. The convex shape of Figure 6.3 and concave shape of Figure 6.4 demonstrate that asymmetry does in fact influence performance. This establishes our first claim: the introduction of asymmetric turning has an effect on pursuit-evasion game outcome.

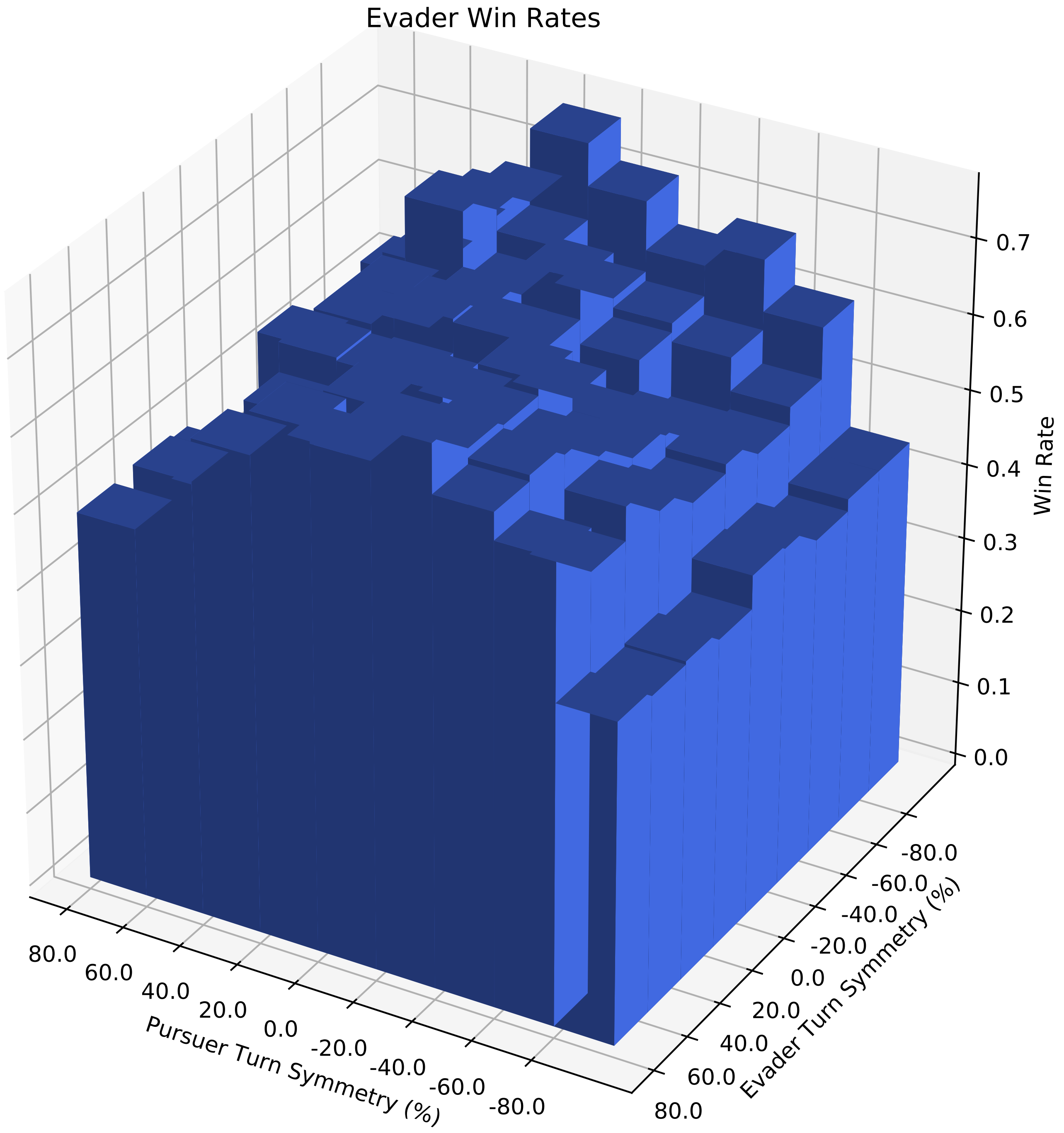


Figure 6.3: Evader win rates across all turn permutations. Left turns become sharper (right turns become broader) as turn symmetry values become more positive. Right turns become sharper (left turns become broader) as symmetry values become more negative. Turn radii are symmetrical at 0.0%.

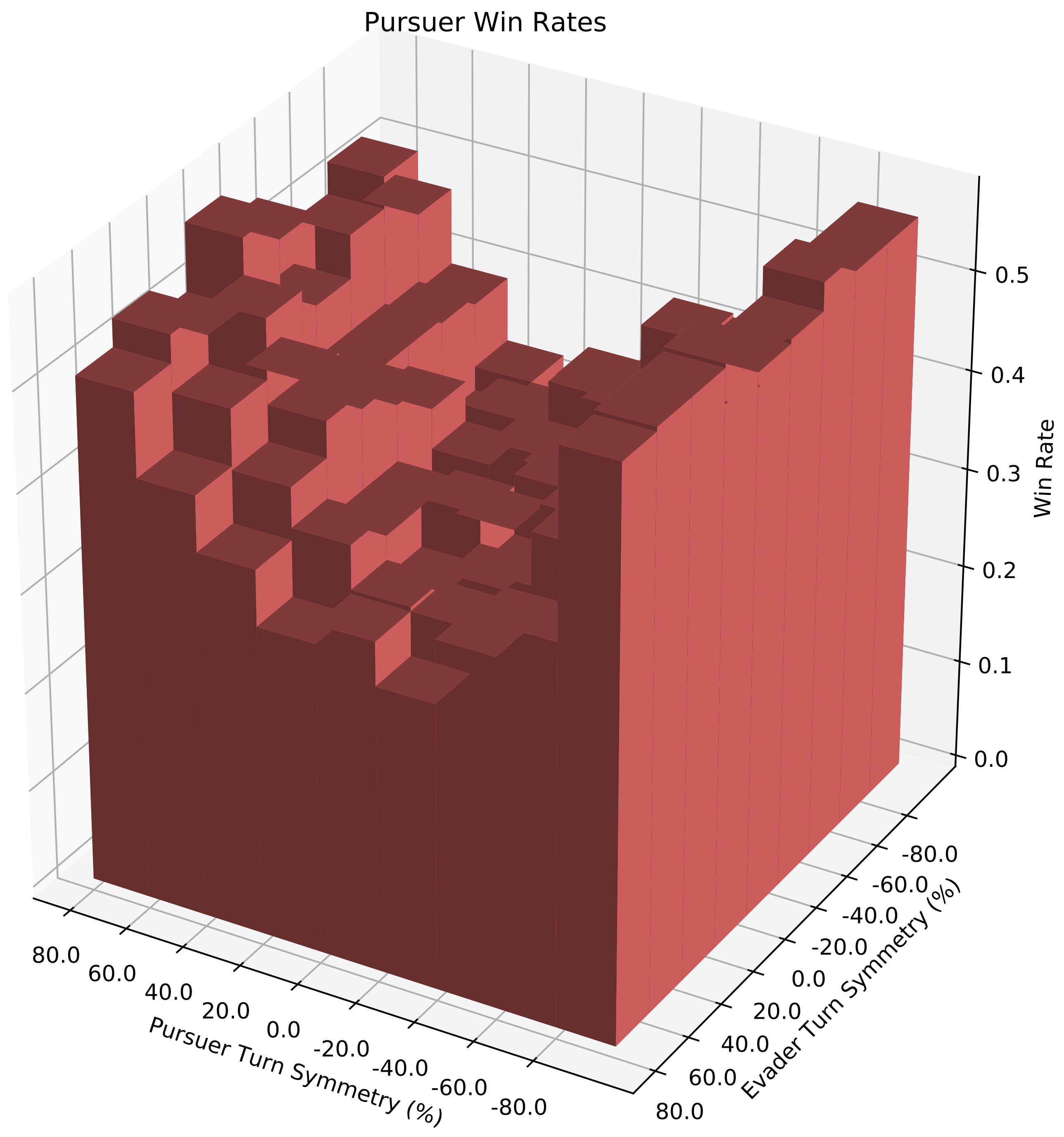
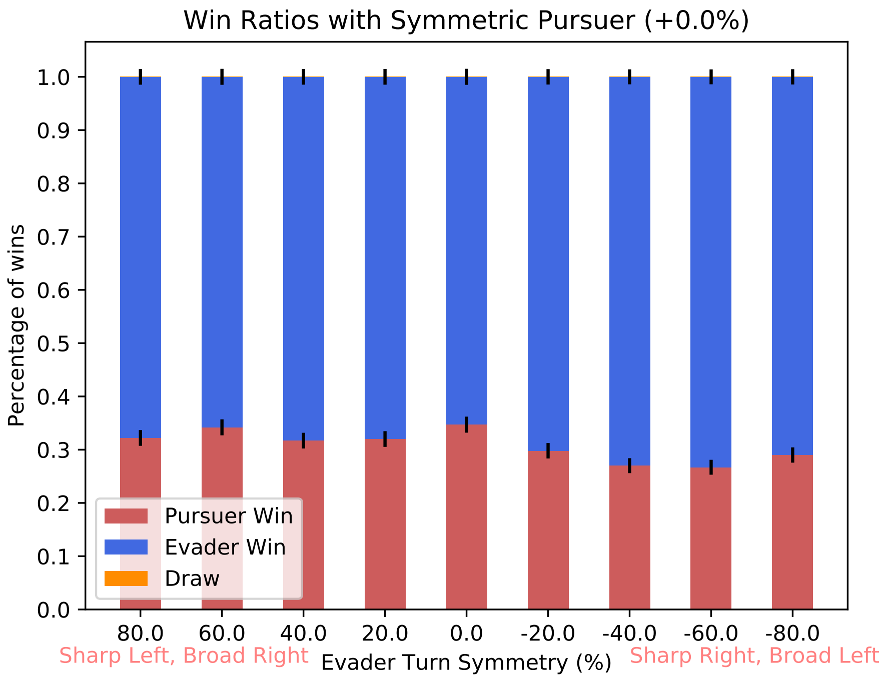


Figure 6.4: Pursuer win rates across all turn permutations. Left turns become sharper (right turns become broader) as turn symmetry values become more positive. Right turns become sharper (left turns become broader) as symmetry values become more negative. Turn radii are symmetrical at 0.0

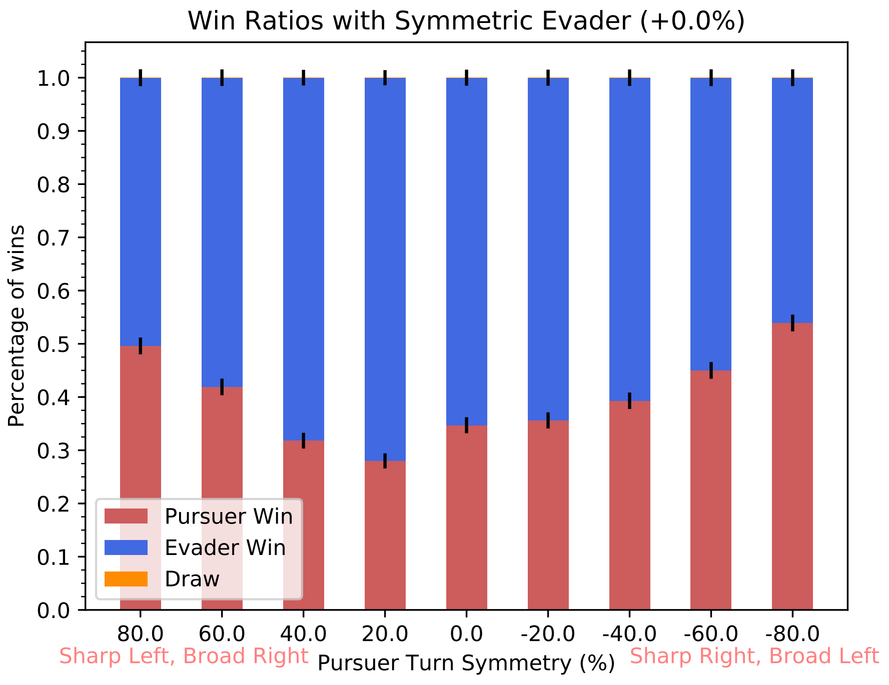
Next, we notice that as we move across the pursuer turn symmetry axis of Figure 6.4, pursuer win rates are higher on the outer edges of the surface and lower in the middle, indicating that as pursuer asymmetry increases, pursuer win rates increase. Moving across the evader turn symmetry axis, we do not see the same trend. Rather, for a given pursuer turn symmetry, the evader's win rates are relatively unchanged, regardless of evader turn symmetry.

Figure 6.5a illustrates the center cross section of Figures 6.3 and 6.4, in which a range of evader turn symmetry conditions is illustrated for a symmetric pursuer. Evader win rates range from $0.653 \pm .015$ to $0.733 \pm .014$, for a total spread of $0.08 \pm .02$.

Figure 6.5b is the perpendicular cross section to Figure 6.5a, in which a range of pursuer turn symmetry settings is presented for a symmetric evader. This illustrates that the pursuer's advantage increases as it become more asymmetrical on either side. Pursuer win rates range 0.280 ± 0.014 to 0.539 ± 0.016, for a total spread of $0.259 \pm .021$. The difference in pursuer win rates across Figure 6.5b is larger than the difference in evader win rates across Figure 6.5a. This evidence supports the second claim: the pursuer gains more advantage with asymmetry than the evader.



(a) Agent win rates for asymmetric evader against symmetric pursuer.



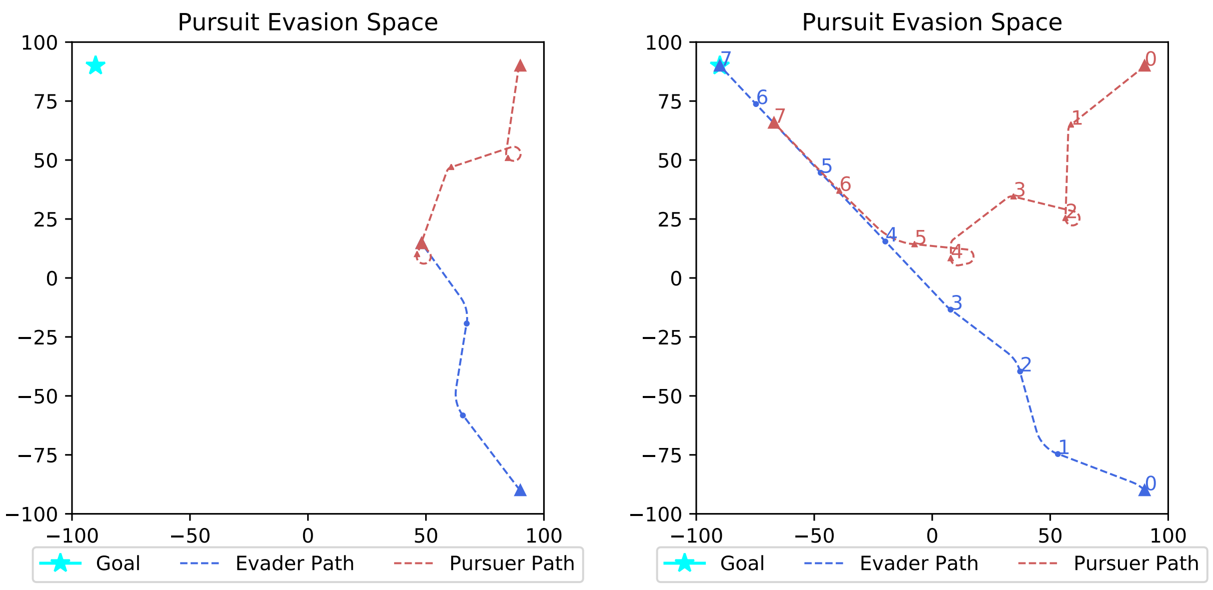
(b) Agent win rates for asymmetric pursuer against symmetric evader.

Figure 6.5: Asymmetric agent win rates against symmetric opponents.

Examining how paths adapt as asymmetry is introduced further illuminates these trends. Figure 6.6a illustrates games in which the pursuer turn symmetry is 80%, which indicates a sharp left turn and broad right turn ability. The pursuer shows exceptional favoritism for left turns over right turns, occasionally going so far as to choose $+180°$ loops to the left to avoid the cost of a right turn. This mirrors behavior reported by Sopwith Camel pilots [8].

We see similar behavior in Figures 6.6b, but this time with the pursuer receiving a sharp right turn. In this scenario, notice the pursuer becomes more effective once the evader moves to the pursuer's right hand side. The pursuer maintains costlier left hand turns early in the game, but becomes more aggressive with right hand turns once the evader transitions to the other half of the playing field.

Examples with evader asymmetry are shown in Figure 6.7a. The evader demonstrates similar turn behavior as the pursuer, favoring the sharper turns. A possible reason asymmetry improves the pursuer so much more than the evader is that the pursuer's goal is dynamic, whereas the evader's goal is static. One characteristic of asymmetry is increased direction change agility. The pursuer values this agility much more than the evader because the pursuer is more often needing to change direction. The evader's trajectory, on the other hand, trends in the same direction the entire game.


(a) Pursuit-evasion game with pursuer symmetry at 80%.

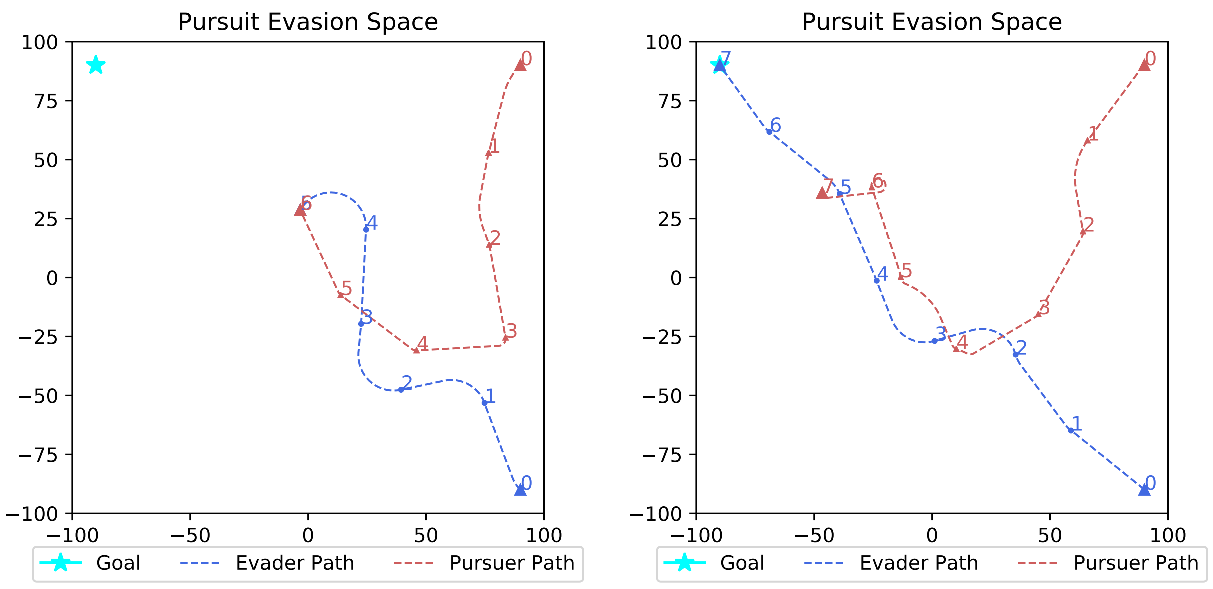
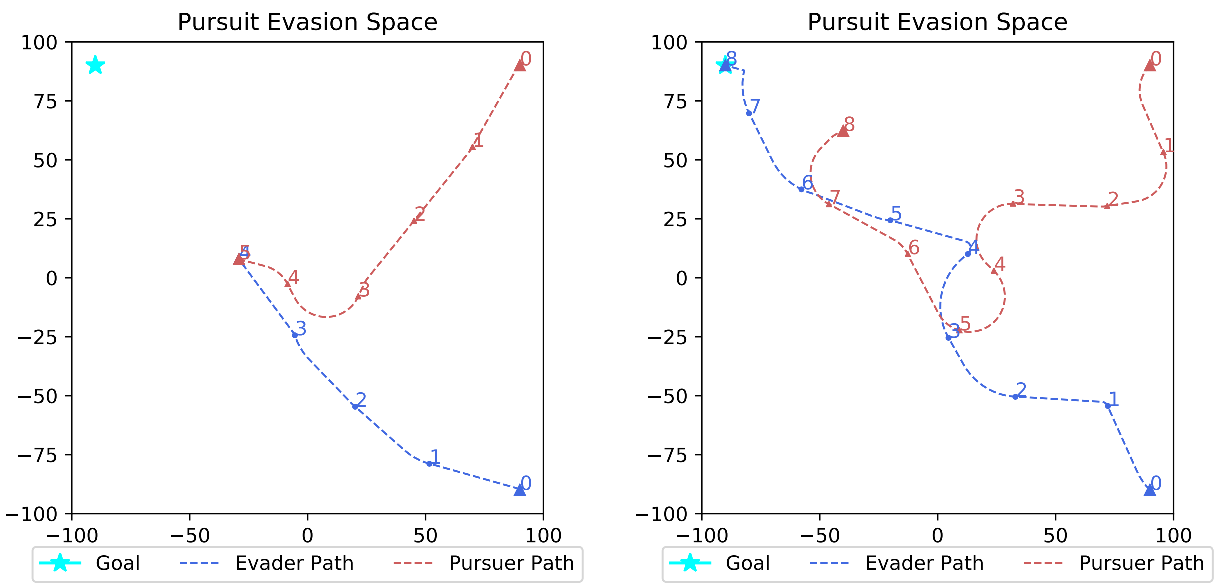
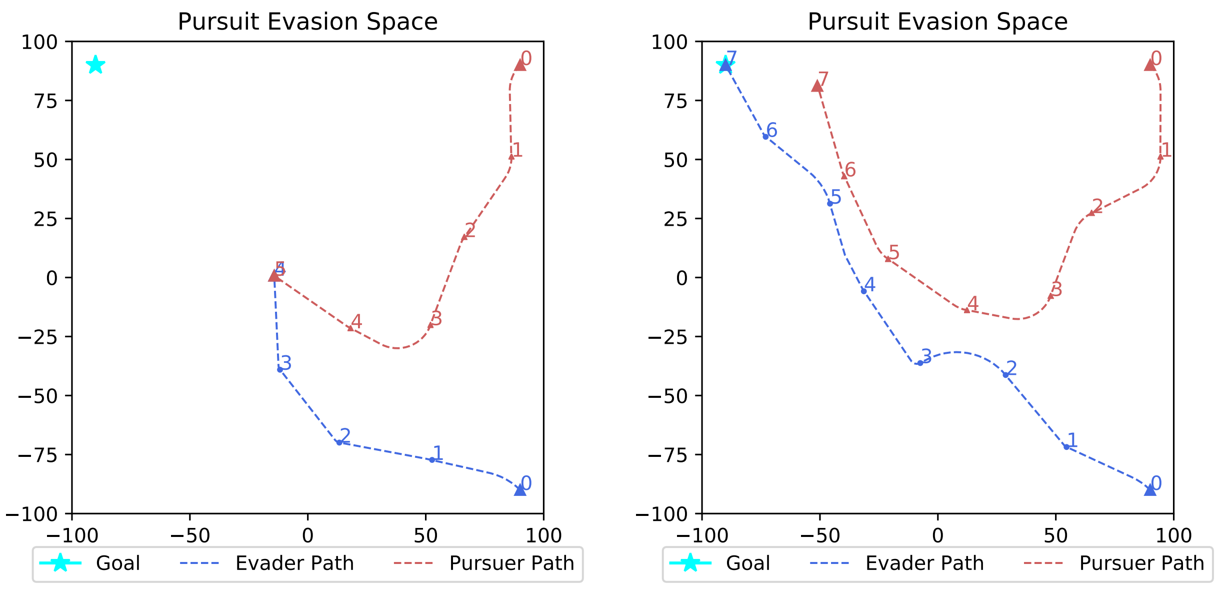

(b) Pursuit-evasion game with pursuer symmetry at -80%.

Figure 6.6: Pursuit-evasion game with sharply asymmetric pursuer.



(a) Pursuit-evasion game with evader symmetry at 80%.



(b) Pursuit-evasion game with evader symmetry at -80%.

Figure 6.7: Pursuit-evasion game with sharply asymmetric evader.

6.2 Agent Relative Positioning Influences Asymmetric Effectiveness

This section will establish our third claim, that variability in scenario starting positions affect which asymmetric turning conditions are most advantageous to the agents. Consider the game scenario shown in Figure 6.8. The pursuer begins in the bottom left side of the space, rather than the top right. This places the pursuer the same relative distance and geometry to the evader and the evader's goal, but now the evader is on the right hand side of the pursuer, rather than the left as in Figure 6.1.

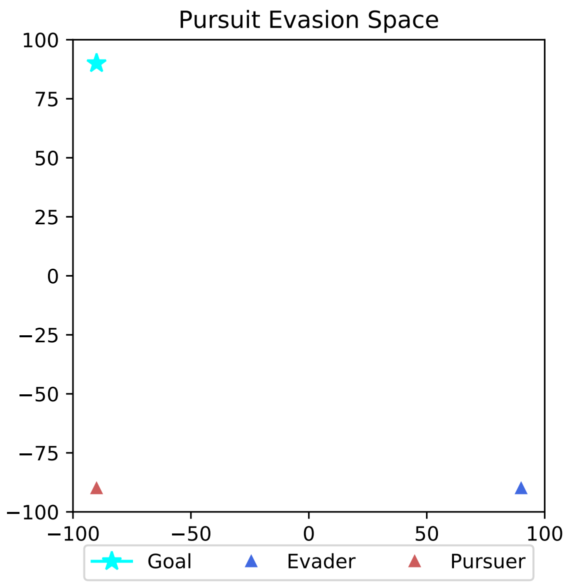
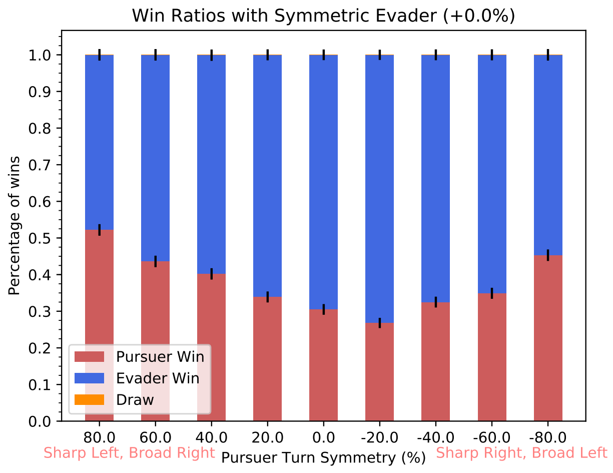
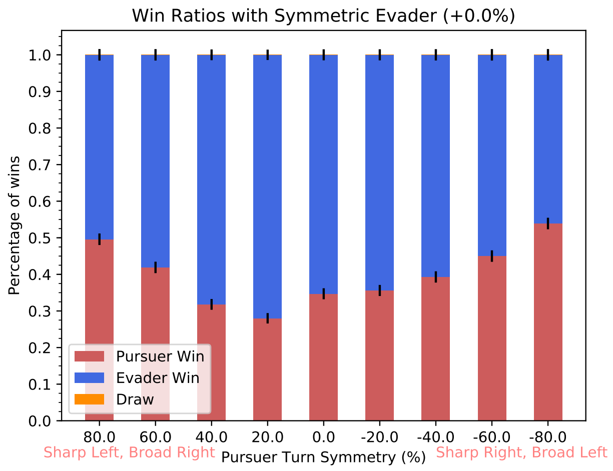


Figure 6.8: Game setup with evader on right side of the pursuer.

Figure 6.9 shows the agent win rates for an asymmetric pursuer against a symmetric evader. Figure 6.9a are win rates for games in which the evader starts on the right hand side of the pursuer (game setup as illustrated in Figure 6.8). Figure 6.9b are win rates for games in which the evader is on the left hand side of the pursuer (game setup as illustrated in Figure 6.1). Notice that the trends of these two graphs mirror each other. When the evader is on the right hand side of the pursuer, the pursuer win rates for games with sharp left/broad right turns are larger than pursuer win rates for games with sharp right/broad left turns. The inverse is true when the evader is on the left hand side of the pursuer: pursuer win rates for games with sharp right/broad left turns are larger than pursuer win rates for games with sharp left/broad right turns. We conclude from this that the pursuer demonstrates a statistically significant improvement in performance when it is given a broader turn in the direction of the evader's relative start position.



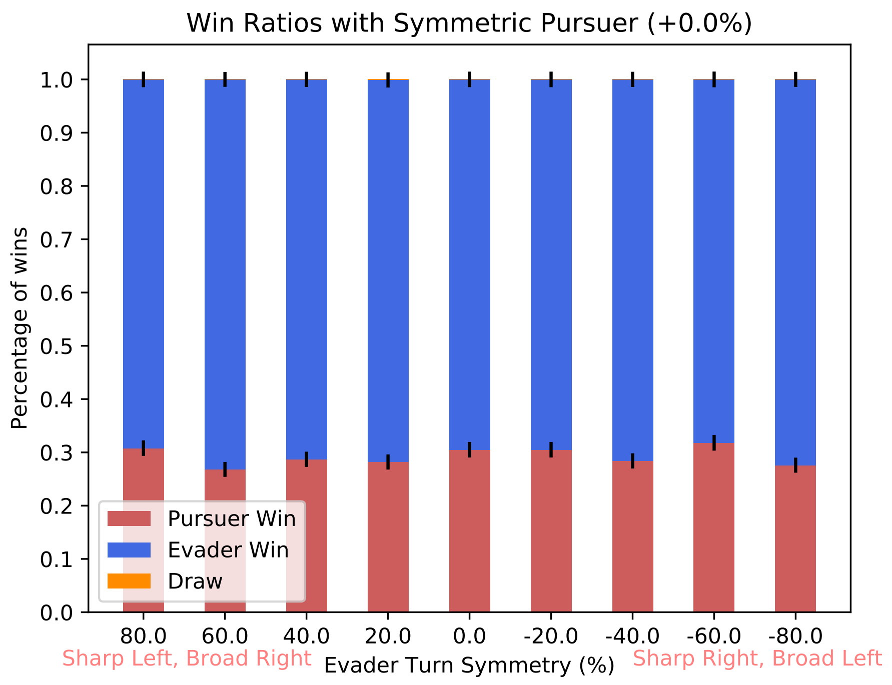
(a) Evader starting on right side of pursuer.



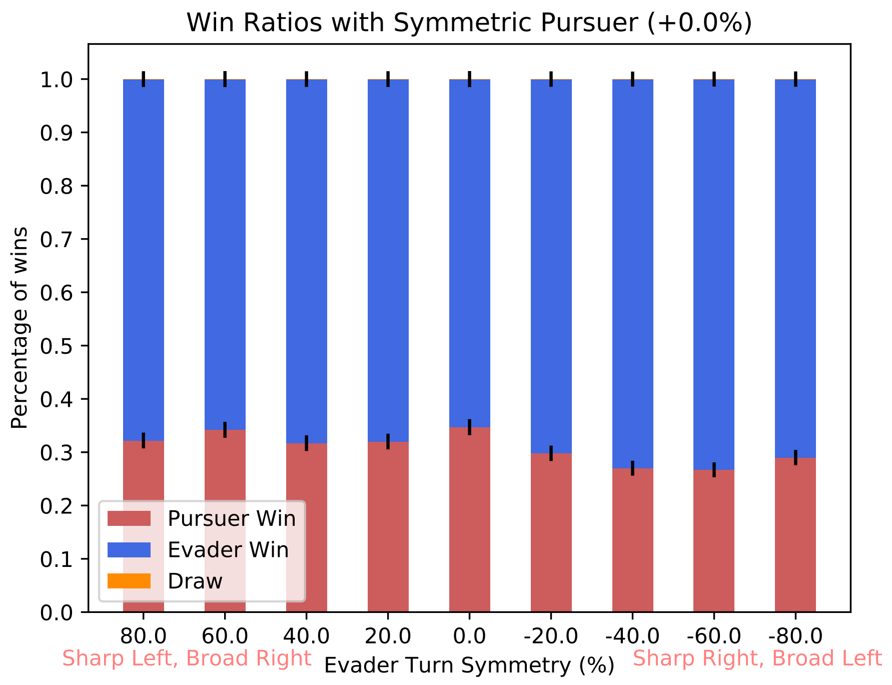
(b) Evader starting on left side of pursuer.

Figure 6.9: Agent win rates for asymmetric pursuer against symmetric evader, left versus right starting position.

Figure 6.10 shows the evader's comparative trends against a symmetric pursuer. Though more subtle than Figure 6.9, a similar phenomenon persists. Consider Figure 6.10b. Observing only the win rates with positive evader turn symmetry values (the left half of the chart, including the center), the values range from $0.653 \pm .015$ to $0.680 \pm .015$, and are therefore equivalent within uncertainty. The games with negative evader turn symmetry (the right half of the chart) begin to show a statistically significant improvement in the evader's performance. The evader demonstrates a statistically significant advantage with a sharp right/broad left turn symmetry when positioned to the left of the pursuer. Figure 6.10a demonstrates the inverse. Evader win rates range from $0.682 \pm .015$ to $0.732 \pm .014$, for a total spread of $.05 \pm .021$. Now, however, the evader has a statistically significant advantage with positive turn symmetry (sharp left, broad right) when starting on the right side of the pursuer. We can conclude, therefore, that the evader demonstrates a slight improvement in performance when given a sharper turn in the direction of the pursuer's relative start position.



(a) Evader starting on right side of pursuer.



(b) Evader starting on left side of pursuer.

Figure 6.10: Agent win rates for asymmetric evader against symmetric pursuer, left versus right starting position.

These trends level out when the evader and pursuer are positioned directly in front of each other. Figure 6.11 illustrates a game setup in which the pursuer is directly between the evader and the goal. The pursuer is the same distance away from the evader as in the previous game examples. Figure 6.12 shows the pursuer and evader win rates. Neither the left nor right half of the graphs shown in Figure 6.12 yield more of an agent advantage compared to the other.

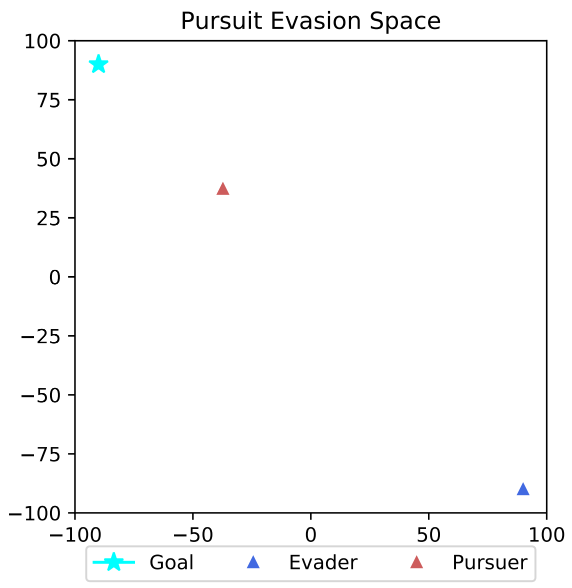
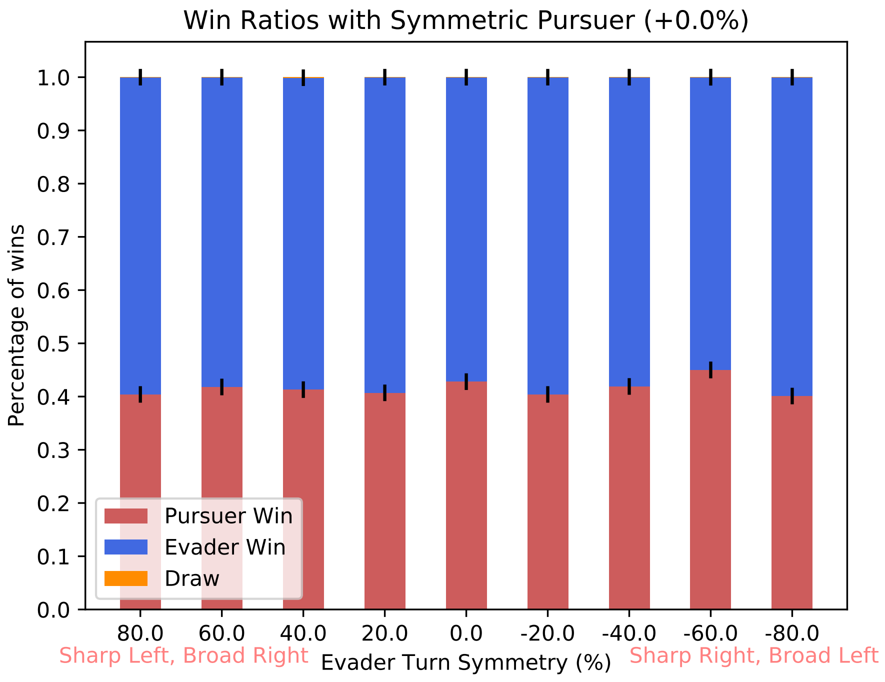
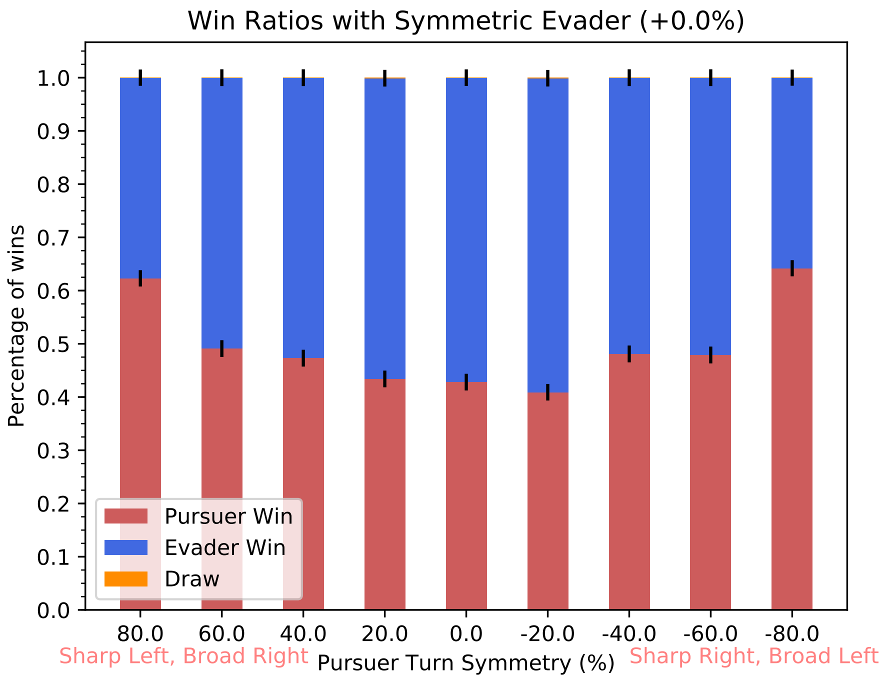


Figure 6.11: Game setup with pursuer and evader directly in front of each other.



(a) Agent win rates for asymmetric evader against symmetric pursuer.

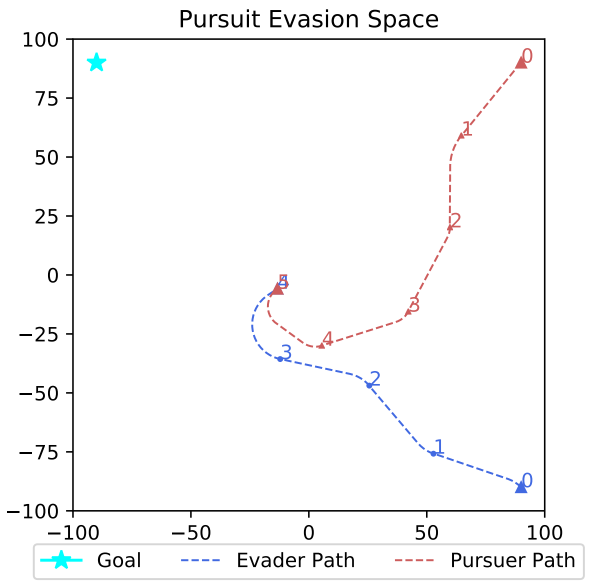


(b) Agent win rates for asymmetric pursuer against symmetric evader.

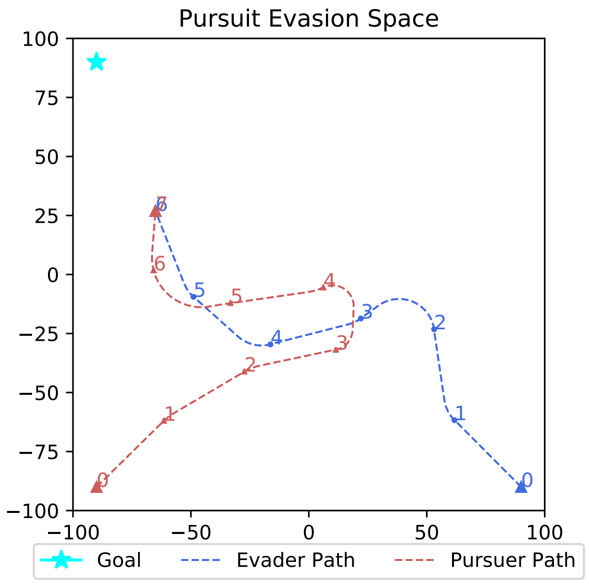
Figure 6.12: Agent win rates for pursuer directly in front of goal.

This is a particularly interesting trend. The evader is able to most effectively utilize its sharp turn side to outmaneuver the pursuer from its starting position. Consider that if the evader begins on the right side of the pursuer, the pursuer's trajectory will trend more to the right. The evader's sharp turns to the left allow it to zip around the pursuer's right turn arcs most effectively, and vice versa for when the evader begins on the left side. This phenomena is similar to *Fokker Dr I* pilots utilizing right turns in evasive maneuvers to escape the slower right turns of the Sopwith Camel [1].

Now consider the pursuer's perspective. One could imagine that it would be more advantageous to have the sharper turn towards the evader's start position. With the sharper turn towards the evader, the pursuer could more quickly angle itself in the evader's direction for a straight path to interception. Rather, the opposite is true. A broader turn in the direction of the evader's start position is more favorable. To explain this, consider Figure 6.13, which illustrates a pursuit-evasion game from both presented starting positions. It is to the pursuer's advantage to head to the middle of the arena and cut the evader off there. If the pursuer begins heading directly to the evader's starting position from the get-go, the evader will already be well across the field by the time the pursuer gets near evader's position *0*. By utilizing its broader turn at the start, the pursuer gently slopes towards the evader, and maintains its trajectory towards where the evader is going to be, rather than where it is. Once the evader crosses mid-field, the pursuer's sharper turn now comes into play. The pursuer can quickly shift directions using the sharper turn direction to capture the evader, as demonstrated in turns 3 through 6 of Figure 6.13b and turns 3 through 5 of 6.13a. This insight demonstrates how both the broad and sharp turning direction can be effectively utilized, and thus illustrating the value of the asymmetric strategy. The broad turn direction is helpful for the pursuer to gradually approach its target from a distance, and the sharp turn direction enables the pursuer to swiftly strike from a close range.



(a) Pursuer with -40% turn (sharp right, broad left), against symmetric evader.



(b) Pursuer with 40% turn (sharp left, broad right), against symmetric evader.

Figure 6.13: Pursuer advantage against different evader start positions.

VII. CONCLUSION

Asymmetry has a non-negligible influence on agent performance. The degree of influence, however, depends on whether we consider the pursuer or evader, as well as spatial configurations. The pursuer most effectively capitalizes on asymmetric turning because its trajectory is more dynamic compared to the evader's. The pursuer's goal point is constantly changing over the course of the game, which understandably handicaps the pursuer. Pursuer win rates when both agents were symmetric tended to be only around 30 to 40%. Asymmetry offered the pursuer a considerably more strategic set of motions, such that pursuer win rates improved to upwards of 50% when asymmetric turning was at an extreme. We can learn from this that asymmetric turning is a valuable compensation for destination uncertainty. Both the broad and sharp turn are utilized to the pursuer's advantage at different moments in the game. It is this diversity of motion options which makes asymmetric turning valuable when the destination is uncertain. The diversity of an agent's motion options is increased as asymmetry increases, giving the agent more tools to creatively plan towards its path objective. Asymmetric turning should be considered for agents with dynamic environments because it allows more diverse maneuverability.

When path planning towards a static goal, this diversity of motion is less useful, as demonstrated by the evader's consistent win rates, regardless of turn symmetry. Agents that are moving relatively predictable, routine paths are not aided by asymmetry. A vehicle path planning to a destination via highway, for instance, does not have much to gain from asymmetric turning, because its turning capabilities are already established to handle its task, and there is little chance the vehicle will face disruption from its current path. However, it is also valuable insight to realize that asymmetric turning did not ever handicap the evader. It's performance was consistent, regardless of the turn symmetry. Perhaps, then, it is useful to consider asymmetric turning as a feature for systems more generally, regardless of the dynamism of their task. In the event that the agent's goal may abruptly change, asymmetry can suddenly become a useful tool. Consider a surveillance drone that routinely scouts the same area, but then on occasion must quickly change its destination target to a hiding location to avoid being spotted. While the asymmetric turning feature may be underutilized most of the time, there are moments where it suddenly becomes pivotal.

There are reasonable limitations too. As asymmetry increased, agents were choosing to loop to their sharper turn side to avoid the cost of their broader turn, sometimes for relatively mundane actions. A car on an open road may be able to technically path plan across the countryside with an extremely sharp left turn and broad right turn, but it will often be making 270° loops to the left in order to ultimately change direction to the right. From this insight, we come to understand that asymmetric turning is limited by the constraints of the space and vehicle. Autonomous cars driving on the highway are in a highly standardized path planning environment. The road is carefully engineered with the expectation of a particular vehicle turning radius and speed. Asymmetry may be an antidote for planning towards an unpredictable destination, but asymmetry itself creates unpredictable behavior. Asymmetric turning is most effectively utilized in open environments, or environments where the rules of motion are not strict, such that an agent has the freedom to move creatively. Open airspace or waters are practical examples.

Future work should explore the scope of uncertainty that asymmetry can overcome. This research discovered that asymmetric turning is an effective tool to compensate for uncertainty towards a changing goal destination. Could it also compensate for other motion uncertainty? Experimentation with different vehicles could also yield interesting insights into the practical principles of implementing asymmetric turning. Studies with more specific space and vehicle constraints should be done to guide explicit directions on implementing asymmetric turning into vehicles. Furthermore, our study focused on asymmetric turning, but consideration for other asymmetric controls can further our understanding of the value of asymmetry. Experimentation with other types of pursuit-evasion games is also useful. Games which mimic aircraft dogfights, for instance, in which agents play as both a pursuer and evader, could reveal further wisdom as to the role of asymmetry in moment to moment game strategy.

Asymmetry is a valuable path planning tool with an observable effect on performance. In pursuit-evasion games, asymmetry is an asset to the pursuer, which can leverage asymmetric turning to diversify its path planning strategy, and more effectively intercept the evader. The evader is neither helped nor handicapped by asymmetric turning because of its comparatively static trajectory. The effects of asymmetry are also influenced by relative agent position within the space. Pilots of the Sopwith Camel were not wrong to value the aircraft's asymmetric turning features. Though the advantages are not immediately obvious, nor quite as expected, they are real and powerful.

REFERENCES

[1] J. Guttman, *Sopwith Camel*. Oxford, UK: Osprey Publishing, October 2012.

[2] H. J. Lee, V. Heim, and A. Meyer, “Genetic and environmental effects on the morphological asymmetry in the scale eating cichlid fish perissodus micolepis,” *Ecology and Evolution*, vol. 5 (19), pp. 4277–4286, August 2015.

[3] J. M. Jodao and R. F. Oliveira, “Sex differences in predator evasion in the fiddler crab,” *Journal of Crustacean Biology*, vol. 21 (4), pp. 948–953, April 2001.

[4] R. Allain, *How Do Drones Fly?* Wired, May 2017. `https://www.wired.com/2017/05/the-physics-of-drones/` (accessed March 09, 2020).

[5] Skybrary, *Counter-Rotating Propellers*. Flight Safety Foundation, July 2017. `https://www.skybrary.aero/index.php/Counter-Rotating_Propellers` (accessed March 09, 2020).

[6] R. Jackson, *Britain’s Greatest Aircraft*. Barnsley, UK: Pen and Sword, September 2007.

[7] S. Takeda and M. Murai, “Asymmetry in male fiddler crabs is related to the basic pattern of claw waving display,” *The Biological Bulletin*, vol. 184, pp. 203–208, 1993.

[8] P. Garrison, *Calculated Sopwith Camel*. Flying Magazine, April 2014. `https://www.flyingmag.com/pilots-places/pilots-adventures-more/calculated-sopwith-camel/` (accessed March 3, 2020).

[9] *Blohm and Voss Bv 141 Tactical Reconnaissance Aircraft*. Military Factory, June 2019. `https://www.militaryfactory.com/aircraft/detail.asp?aircraft_id=781` (accessed March 03, 2020).

[10] S. M. Lavalle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[11] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*. New York, NY, USA: Pearson, 2015.

[12] S. M. Lavalle, "Rapidly exploring random trees: A new tool for path planning," *Iowa State Univ. Tech Rep*, 1998.

[13] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *arXiv:1005.0416v1*, 2010.

[14] P. J. Nahin, *Chases and Escapes: The Mathematics of Pursuit and Evasion*. Princeton, NJ, USA: Princeton University Press, July 2012.

[15] N. Noori and V. Isler, "The lion and man game on polyhedral surfaces with obstacles," *Theoretical Computer Science*, vol. 739, pp. 39–58, May 2018.

[16] S. Boparkikar, F. Bullo, and J. Hespanha, "On discrete-time pursuit-evasion games with sensing limitations," *IEEE Transactions on Robotics*, vol. 24 (6), December 2008.

[17] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for a class of pursuit-evasion games," *Algorithmic Foundations of Robotics IX*, 2011.

[18] V. Isler, D. Sun, and S. Sastry, "Roadmap based pursuit evasion and collision avoidance," *Robotics: Science and Systems*, 2005.

[19] A. T. Becker and S. Shahroki, *Shortest Paths for Dubins Cars*. Wolfram, December 2017. `demonstrations.wolfram.com/ShortestPathForTheDubinsCar` (accessed March 05, 2020).

[20] L. 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, pp. 497–516, 1957.

[21] S. Alba, *Dubins1*. Wikimedia, February 2016. `https://commons.wikimedia.org/wiki/File:Dubins1.svg` (accessed May 02, 2020).

[22] S. Alba, *Dubins2*. Wikimedia, February 2016. `https://commons.wikimedia.org/wiki/File:Dubins2.svg` (accessed May 02, 2020).

[23] S. Alba, *Dubins3*. Wikimedia, February 2016. `https://commons.wikimedia.org/wiki/File:Dubins3.svg` (accessed May 02, 2020).

[24] E. Bakolas and P. Tsiotras, "The asymmetric sinistral/dextral markov-dubins problem," Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference, December 2009.

[25] S. Manyam, S. Rathinam, D. Casbeer, and E. Garcia, "Shortest paths of bounded curvature for the dubins interval problem," *arXiv:1507.06980v2*, August 2015.

[26] J.-D. Boissonat, A. Cerezo, and J. Leblond, "Shortest paths of bounded curvature in the plane," Proceedings of the 1992 IEEE International Conference on Robotics and Automation, May 1992.

[27] Z.-N. Bui, J.-D. Boissonat, P. Soueres, and J.-P. Laumond, "Shortest path synthesis for dubins non-holonomic robot," Proceedings of the 1994 IEEE International Conference on Robotics and Automation, May 1994.

[28] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics and Automation Magazine*, vol. 19, pp. 72–82, 2012.

[29] Kavraki Lab, Rice University, *Open Motion Planning Library: A Primer*, July 2019. `http://ompl.kavrakilab.org/OMPL_Primer.pdf` (accessed May 02, 2020).

[30] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science and Engineering*, vol. 9, pp. 90–95, 2007.

[31] T. E. Oliphant, *A Guide to NumPy*. USA: Trelgol Publishing, 2006.