A Reactive/Deliberative Planner Using Genetic Algorithms on Tactical Primitives

by

Stephen William Thrasher, Jr.

B.S. Engineering and Applied Science
California Institute of Technology, 2002

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author . . . .
Department of Aeronautics and Astronautics
May 29, 2006

Certified by . . . .
Christopher Dever
Senior Member, Technical Staff, C.S. Draper Laboratory
Thesis Supervisor

Certified by . . . .
John Deyst
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . .
Jaime Peraire
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

A Reactive/Deliberative Planner Using Genetic Algorithms on Tactical Primitives

by

Stephen William Thrasher, Jr.

Submitted to the Department of Aeronautics and Astronautics
on May 29, 2006, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Unmanned aerial systems are increasingly assisting and replacing humans on so-called dull, dirty, and dangerous missions. In the future such systems will require higher levels of autonomy to effectively use their agile maneuvering capabilities and high-performance weapons and sensors in rapidly evolving, limited-communication combat situations. Most existing vehicle planning methods perform poorly on such realistic scenarios because they do not consider both continuous nonlinear system dynamics and discrete actions and choices. This thesis proposes a flexible framework for forming dynamically realistic, hybrid system plans composed of parametrized tactical primitives using genetic algorithms, which implicitly accommodate hybrid dynamics through a nonlinear fitness function. The framework combines deliberative planning with specially chosen tactical primitives to react to fast changes in the environment, such as pop-up threats. Tactical primitives encapsulate continuous and discrete elements together, using discrete switchings to define the primitive type and both discrete and continuous parameters to capture stylistic variations. This thesis demonstrates the combined reactive/deliberative framework on a problem involving two-dimensional navigation through a field of threats while firing weapons and deploying countermeasures. It also explores the planner's performance with respect to computational resources, problem dimensionality, primitive design, and planner initialization. These explorations can guide further algorithm design and future autonomous tactics research.

Thesis Supervisor: Christopher Dever
Title: Senior Member, Technical Staff, C.S. Draper Laboratory

Thesis Supervisor: John Deyst
Title: Professor of Aeronautics and Astronautics

Acknowledgments

Blessed be the God and Father of our Lord Jesus Christ! According to his great mercy, he has caused us to be born again to a living hope through the resurrection of Jesus Christ from the dead, to an inheritance that is imperishable, undefiled, and unfading, kept in heaven for you, who by God's power are being guarded through faith for a salvation ready to be revealed in the last time. — St. Peter

This work is dedicated to the memory of my mother, Marilyn. I pray that I am always at the ready to care for the helpless, the poor, the sick, and the hungry.

Thank you, Chris Dever, for your patience, your ideas, and your close supervision. Thank you, John Deyst, for willingly advising me on this thesis. Thank you, Jeff Miller, for your time and insight into all things machine learning. Thank you, Brent Appleby, for late-evening conversations about life, work, and MIT DLF history.

Thanks to Drew Barker for gymnastics, friendship, and being an all-around quality guy; to Jon Beaton for the encouragement to keep going; to Tom Krenzke for being a good sport of a classmate, a good friend, a steadfast workout partner, and a cohort in shenanigans, like the flipbook in the lower right-hand corner of your master's thesis; to Pete Lommel for your discussions about autonomy and whatever else; to Brian Mihok for your big smile and honest friendship.

Thanks to Jeff Blackburne for chili and cornbread, and to Ali Hadiashar for making me things. You are both great roommates and brothers in Christ.

Thanks to Greg Glassman for helping me retain my sanity through short bursts of varied, intense physical effort.

Thank you, Stephen W. Thrasher, Sr., for being a loving and supportive dad. And to Jimmy and Thomas, who are excellent brothers.

Secondmost of all, thanks to Rachel Edsall, who is awesome. I look forward to learning more about you and the many ways in which you are awesome.

Finally, thank you readers. I recommend skimming through Chapters 1 through 3. The material in Chapters 4 and 5 should interest you the most. Flying dots are so much better than details and pseudocode.

Amen. Come Lord Jesus.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Independent Research and Development Project Number 20325, Tactics Generation for Autonomous Vehicles.

Contents

List of Figures

List of Tables

Chapter 1

Tactical Decision Making

People become bored and error-prone when they perform *dull*, repetitive actions. They must take costly precautions when working in *dirty* environments that are contaminated with chemical, biological, or nuclear waste. *Dangerous* jobs such as mining or combat put people's lives at risk. Autonomous systems offer a means of assisting or replacing humans on tasks that fall into these well known "three D's"—dull, dirty, and dangerous.

In some ways, the capabilities of autonomous vehicles (AV) far exceed those of manned vehicles. AVs accelerate and turn sharply without causing pilot blackouts. Engineers design smaller and smaller AVs because no human needs to fit inside. They remain aloft for days without requiring pilots to take sleep shifts or performance-enhancing drugs.

At the time of this writing, however, unmanned aerial vehicles (UAVs) used in combat require large supervisory teams and fly mostly high-altitude surveillance missions. Their high-performance flight envelopes go unexplored, and they always remain on a communication tether to a remote human operator. Modern UAVs do not perform terrain-following flight or win dogfights against fighter pilots. With improved autonomy, however, a single operator could control several UAVs at once, UAVs could evade new threats even after losing their communication link, and AVs could operate in the complex theaters of air-to-air combat, urban warfare, and underwater reconnaissance.

Thus far, AVs perform well in highly regulated environments on well defined tasks, but they perform poorly on many tasks that human pilots do well, such as formation flight, planning in novel situations, and executing coordinated, agile maneuvers in highly constrained environments. Algorithms for controlling AVs have not caught up with AV capabilities. Agile flight of these high-performance systems simultaneously requires the control of a nonlinear, continuous flight path, decisions about how to accomplish multiple objectives, and effective use of sensors and weapons. Optimally and reliably satisfying these requirements in real time is an open research effort of the controls engineering community.

This thesis contributes to that effort with a nonlinear, hybrid system method for maneuvering a vehicle and controlling its discrete actions. This method incorporates human input in several forms, closing the performance gap between humans and AVs.

Its building blocks can mimic human behaviors while allowing for stylistic variations, giving an AV some flexibility during planning. Engineers can also directly design these building blocks using trial-and-error and intuition. The optimization algorithm can start from scratch, but it also accepts human-inspired and computer-generated candidate solutions. The method is flexible and general enough to work on any system whose necessary behaviors can be condensed into a small library of building blocks and which can evaluate its cost function quickly. This approach does not guarantee optimality or reliability, but several means of increasing the likelihood of near-optimal solutions exist.

Planning in adversarial scenarios can be called "tactical decision making." The next two sections define this term intuitively and mathematically, and the following sections present the background behind incorporating human knowledge into planning algorithms and give a review of the vehicle planning literature.

1.1 Description of Tactics

Merriam-Webster's Collegiate Dictionary defines tactics as "the art or skill of employing available means to accomplish an end" [34]. Common applications that require tactics include the deployment of military forces and the execution of plays in a sports game. In such situations commanders and coaches take their knowledge of past approaches and innovatively use them to defeat their opponent. Every battle or play requires specificity and creativity to handle the complexity of the engagement and to overcome the opponent's tactics.

One way to approach AV planning is to separate it into layers. Figure 1.1 shows a possible breakdown of a military force into planning layers. Each layer abstracts its internal workings to present higher layers with a condensed set of capabilities and constraints and lower layers with a condensed set of requirements and commands. For example, the trajectory generation layer at the bottom of the figure could condense the full AV flight envelope into several packaged agile maneuvers [18]. The "Tactics, Maneuvering" layer is the topic of this thesis: given waypoints and abstracted low-level vehicle capabilities, how should the vehicle interact with its environment between waypoints?

The generation and execution of AV tactics requires specificity and creativity. An AV must choose its actions based on a complex context involving resource constraints, threats, limited knowledge, environmental features, and mission goals. Complexity and uncertainty prevent an AV tactics designer from figuring out an optimal response to every possible situation offline, and complexity and computation time also prevent an AV from solving a near-optimal control problem online from scratch. A combination of offline and online optimization can give an AV better behavior than using just one or the other, just as previous experience and knowledge combined with creativity can give a coach a winning strategy.

Many algorithms exist to control an AV in a specific situation with very good performance, where performance is defined as some measure of efficiency in time and resources or effectiveness in some objective such as disabling threats or remaining

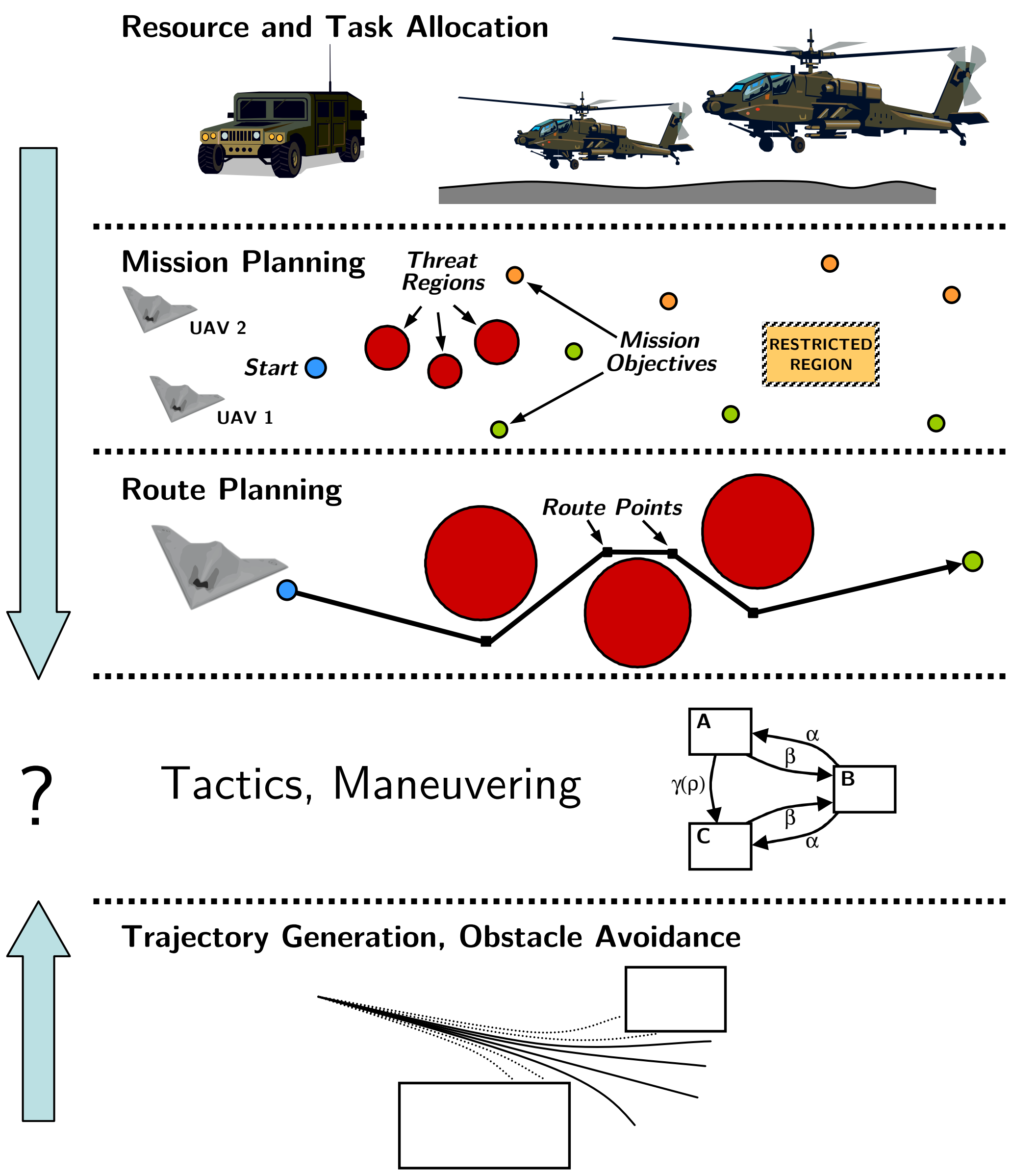


Figure 1.1: Representation of a military vehicle planning hierarchy.

undetected. Example situations from past research include terrain-following flight, evasive maneuvers for pop-up threats, and high-performance maneuvering around obstacles [16, 30, 38]. The existence of many tailored algorithms turns the tactics problem into the question of how to choose between each behavior depending on context.

The research in this thesis considers the tactics of a single UAV in an environment that demands different behaviors in different situations in order to achieve good performance. To leverage past experience and specialized algorithms while maintaining computational feasibility, the UAV restricts its behavior to selections from a finite but rich set of pre-designed modes. The method of restricting behavior to a small set of useful maneuvers has been shown to decrease computation time while retaining maneuverability on the path planning level [16, 18]. Based on its state and model of the environment, the UAV selects a mode and assigns values to its parameters. Example modes and associated parameters include a firing attack with variable altitude, distance to target, and weapon type; an evasive maneuver with turn direction, turn radius, and altitude change; and tracking a target with standoff distance, angle to target, and relative altitude.

When encountering an unexpected situation such as a pop-up threat, an AV has limited time to react. The threat could attack the AV in the time it takes to generate a plan from scratch. If the AV executes a pre-programmed reaction, it could increase its chances of survival. Pre-programmed reactive tactics leverage large offline computing resources to give good performance in a limited range of cases. On the other hand, while reactive tactics give good short-term behavior and take advantage of offline computing, a limited set of rules often cannot give good performance across a very wide range of possibilities. Online planning takes advantage of specific knowledge of the situation, but has to use limited online resources effectively.

Using reactive and planned tactics together has the potential to give better behavior than using just one or the other. High-performance control often involves a predictive element and a reactive element. Many modern high-performance control methods contain a feed-forward plan generator for predictive planning and a feedback controller for regulating errors [35]. The planner in this thesis incorporates both deliberative and reactive elements to leverage the specificity of online planning algorithms and the speed of rule-based reactions.

1.2 Tactics Problem Statement

This section builds on the intuitive definition of tactics with a mathematical representation to precisely define the problem faced in this thesis. Represented as an optimal control problem, a system's state x evolves according to its dynamics when controlled by u. Following is a conceptual simplification of a controlled hybrid system model, which is given in [10], but a fully rigorous definition is not within the scope of this work.

The goal is to choose an input u based on x to minimize a cost function,

$$\min_{u(t)} J(x(t), u(t)) \text{ s.t. } x(t) \in \mathbf{X},\ u(t) \in \mathbf{U}, \tag{1.1}$$

where $\mathbf{X}$ represents the set of possible ways a particular scenario can proceed given vehicle and environment dynamics and constraints, and $\mathbf{U}$ represent the sets of possible controls. The function J maps these possible values to a cost value: $J : \mathbf{X} \times \mathbf{U} \to \mathbb{R}$.

In this research, the goal is to choose u by choosing control modes from a set $\mathcal{M} = \{m_1(x(\cdot)), \dots, m_n(x(\cdot))\}$. Thus, the problem becomes determining a sequence and duration of modes

$$M(x(t), t) = \begin{cases} m_{i_1}(x(t), t) & t \in [t_0, t_1) \\ m_{i_2}(x(t), t) & t \in [t_1, t_2) \\ \quad\vdots & \\ m_{i_q}(x(t), t) & t \in [t_{q-1}, t_q] \end{cases} \tag{1.2}$$

to minimize $J(x(t), M(x(t), t))$. Each mode is a mapping from the system state and time to a control. Modes must be chosen so that $x(t) \in \mathbf{X}$ and $u(t) \in \mathbf{U}$ hold. In addition, the modes can be parametrized, and the choice of parameters must be made: choose $m_k(x(\cdot), t; \vec{p}_k)$ and associated parameters $\vec{p}_k$, with an admissible set $\vec{p}_k \in \mathbf{P}_k$. In this case, let $\mathcal{M}$ denote the set of modes and parameters.

In many cases, mode choices are made with only partial knowledge of the problem constraints or the cost function. For example, an environmental element might be detected halfway through a scenario. This is consistent with the above formulations if x, the system state, includes information about what is known and unknown about the scenario, and the control choices are made in a consistent causal manner.

1.3 Incorporating Expert Knowledge

Compared to most computer algorithms, humans are better at varied, intuitive, high-level reasoning. If an algorithm incorporates human knowledge in its execution, it can often perform better than without that knowledge. For example, Deeper Blue, the chess computer that defeated the reigning world chess champion, Gary Kasparov, used machine learning on example games from several grandmasters. As another example, researchers have found that human-computer collaboration on helicopter mission planning increases performance over planning with just the human or just the computer [17].

Expert knowledge that is useful for AV algorithms comes in several forms. A designer can specify bounds on a system's parameters to keep an algorithm from searching a region of the problem space that will yield no solution. Restrictions could also be applied to the form of search elements, allowing only elements that a designer knows are useful. An educated guess of where to start a search could decrease search time, especially in cases with large problem spaces of high dimension. Experts can provide example maneuvers and decisions. The skills and methods of a human pilot

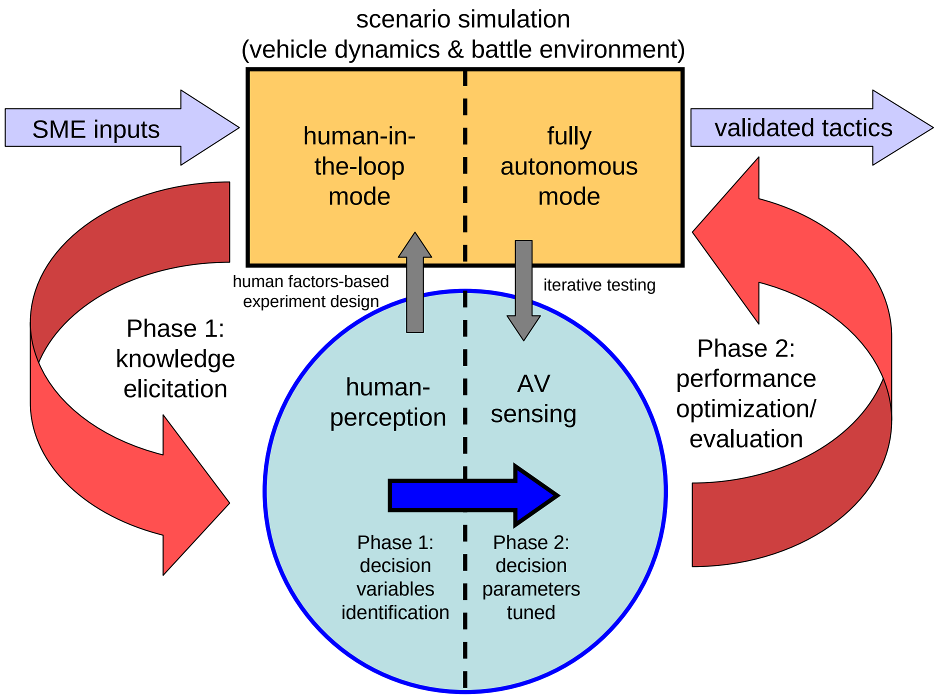

scenario simulation
(vehicle dynamics & battle environment)
SME inputs
human-in-the-loop mode
fully autonomous mode
validated tactics
human factors-based experiment design
iterative testing
Phase 1: knowledge elicitation
Phase 2: performance optimization/ evaluation
human-perception
AV sensing
Phase 1: decision variables identification
Phase 2: decision parameters tuned

Figure 1.2: Two phases of human-inspired autonomy.

can help define AV algorithms either through intuitive observation and design or by measurement and analysis [7].

The methods in this thesis can work in concert with methods for designing the structure of an AV's modes and tactics through studying the choices and performance of a human operator. Figure 1.2 shows a two-phase method of gathering useful information from subject matter experts (SME) in human-in-the-loop experiments for the design of tactics. For example, a pilot could explain what to do in a particular situation, fly and record the scenario in a simulator using the AV's dynamics, and critique the AV's resulting human-inspired behavior [7]. As an example of learning from pilot input, trajectory primitives can be generated directly from flight data [19].

When incorporating expert knowledge, it is difficult to extract exhaustive plans from pilots, who control vehicles by intuition built through training. Also, to construct behaviors that are meaningful in many situations, a pilot would have to spend many hours in a simulator, and even then the pilot would have explored only a finite sample of the problem space. Though expert knowledge aids in the construction of tactics, it must be carefully incorporated into a framework that handles situations that do not arise in the knowledge extraction process.

1.4 Literature Review

The problem of making decisions for continuous systems in the presence of static or dynamic costs is well-studied. Approaches mentioned here are differential games, Markov decision processes, different path planning algorithms, and reactive systems.

The field of differential games is concerned with the precise mathematical modeling of conflict and its outcomes. Classic example problems include air-to-air combat between vehicles of different capabilities and the Homicidal Chauffeur, who seeks to drive a car into a pedestrian [26]. Differential gaming is akin to optimal control in that it seeks to find mathematically optimal solutions. Existing methods for solving differential games are tailored to specific problems that involve linear, continuous dynamics and smooth costs, so they are not suited for problems involving nonlinear, mixed continuous-discrete dynamics and costs.

Markov decision processes (MDP) are discrete-time stochastic processes with discrete states, actions, and transition probabilities [41]. Transition decisions and transition probabilities between states depend on a history of only the current state. Though they are discrete, they are effective for modeling continuous processes that are discretizable to a sufficient fidelity. The problem of using a history of one state can be overcome by letting each state be an ordered tuple of previous states. This approach multiplies the number of states, however, and along with high-fidelity discretization makes graph generation and solution times infeasible. MDPs can be solved using dynamic programming or machine learning.

Path planning is a major focus of AV control research and an important part of AV tactics. The path planning problem involves generating a trajectory in the presence of environmental and vehicle constraints to optimize a cost function. Schouwenaars, et al., wrote a path planning problem in the form of a mixed-integer linear program (MILP) and were able to solve it using commercial optimization software [42]. Bellingham, et al, used MILP with approximate long-range cost estimates in receding horizon control [8]. Frazzoli applied rapidly-exploring random trees (RRT) to his maneuver library to navigate static and deterministic dynamic environments [18]. Pettit used GA to generate evasive maneuvers for unexpected threats [38].

Receding horizon control reduces computation of an optimal control problem by choosing inputs that are optimal for a reduced horizon plus an estimated cost for system behavior in the remaining time. For AV control, it is often necessary to plan using short horizons to avoid computation problems in a complex environment [8]. Also, when the environment is only known within a short horizon, it makes sense not to spend computation time planning in an unknown region. The disadvantage of receding horizon control is that behavior beyond the horizon must be captured in a cost-to-go estimate which might not model the future accurately enough, leading to sub-optimal behavior.

Reactive systems are systems whose actions depend only on their immediate state. Brooks's subsumption architecture is the classic example of reactive control, where robots learned complex motions, such as walking, from simple rules [11]. Arkin created the motor-schema concept, which calculates a weighted sum of competing actions, such as reactions to nearby obstacles or propulsion toward a goal [37]. GA or other machine learning algorithms tuned these motor-schema weights. Grefenstette and Schultz evolved if-then rules mapping sensor ranges to action sets to approach air-to-air combat, autonomous underwater vehicle navigation through minefields, and missile evasion [44, 45]. GA also evolved numerical parameters associated with sensors or actions, such as *turn left with radius 5*. One disadvantage is that reactive systems are deterministic, and deterministic programming might not perform well in previously unencountered situations. Many approaches seek to overcome this limitation, with varying scope and success.

Reactive and deliberative planning can work in concert to handle a diversity of situations that require rapid response. Arkin's Autonomous Robot Architecture (AuRA) uses a planner to choose an appropriate reactive behavior for each situation [4]. Several planners use a combination of activity planning and reactive control [39]. Some RoboCup teams use similar architectures to plan and act at a rapid pace in the complex environment of a robot soccer match [12].

One way of handling decisions in a complex environment is to choose actions from a discrete or parametrized set of maneuvers. This idea is pursued in computer graphics as "motion primitives" and in MDP research as "macro-actions" [2, 28]. In AV research, Frazzoli developed the maneuver automaton, an optimization framework which chooses maneuvers from a library to complete some task [18]. These maneuvers included turns, barrel rolls, and loops. Dever extended this research to include continuously parametrized maneuvers [15, 16]. For example, a loop could be parametrized by its size, or a dashing stop could be parametrized by its displacement. Adding parameters increases the richness of a maneuver library without greatly increasing its size, thus allowing for better performance with often only a small increase in computation.

In the modeling of tactics, Hickie interviewed helicopter pilots about particular scenarios, such as running fire attacks, wrote them in statechart form for ease of communication, and simulated the tactics in a force-on-force simulation [23]. Statecharts are frameworks for describing reactive systems [22]. They represent finite state automata (FSA) that are modified to include hierarchy, concurrency, and communication. Statecharts work well for linking specific actions and algorithms to specific contexts, and deliberative algorithms can fit in the statechart paradigm because of its generality. However, Hickie's work considered reactive tactics alone.

1.5 Contribution

The planner introduced in this thesis offers a novel approach to vehicle planning involving both continuous dynamics and discrete actions through complex, parametrized action primitives. This approach can capture probabilistic problem elements and incorporate initial solution guesses. This thesis also explores several aspects of the planner, such as effects of computation, dimensionality, primitive design, and initialization on performance, as well as comparing two types of optimization algorithms, giving guidance on how to use the planner in practice.

Taking these factors together, the main contribution of this thesis is that the proposed planning method handles a diversity of problems and a large variety of problem elements, such as nonlinearities, stochastic outcomes, hybrid dynamics, rapid environmental changes, and human-inspired candidate solutions, with the possibility of running in realtime.

1.6 Thesis Organization

Proceeding forward, Chapter 2 discusses design elements involved in the construction of tactical primitives, deliberative planning, and reactive planning. This includes methods of parametrizing primitives and encoding plans composed of primitives into a form appropriate for GA optimization. Chapter 3 presents the combined planner framework independent of any specific problem and includes pseudocode. Chapter 4 gives a statement of the threats problem, a two-dimensional UAV problem that exhibits complex behaviors due to its hybrid nature, and delivers results on three threats-problem variations, showing the planner's flexibility and generality. Chapter 5 explores the planner's performance with different computational resources, problem dimensionality, primitive libraries, optimization algorithms, and planner initialization. Chapter 6 summarizes the thesis results and points the way for future work. Appendix A gives a brief overview of GA to support the explanation of the deliberative planner.

Chapter 2

Algorithm Design Elements

The intent of this thesis is to develop a combined reactive/deliberative architecture for a single unmanned aerial vehicle (UAV) operating in a hostile environment. This chapter defines tactical primitives, walks through several design decisions, presents background on methods of creating rules for reactive tactics, and presents the reasoning behind choosing genetic algorithms (GAs) for the selection of tactical primitives.

A general tactics mathematical problem description was given in Section 1.2, but here the details and background of specific design elements are discussed as they relate to UAV tactical primitives and optimization. These elements are important for constructing the reactive/deliberative planning algorithm which is given in the following chapters.

2.1 Tactical Primitives

Vehicle path planning algorithms can be classified according to whether or not the low-level maneuvers consider the state of environmental features. For example, a Voronoi diagram for threats defines line segments that are equidistant to threats, keeping a vehicle as far away from threats as possible, and a visibility graph considers the edges of objects as nodes for path planning [41]. On the other hand, Frazzoli's motion primitives are considered in the vehicle frame of motion, and his use of rapidly-exploring random trees (RRTs) does not leverage features of the environment when carrying out motion planning aside from a cost-to-go evaluation [18].

The disadvantage of Voronoi diagrams and visibility graphs is that they are networks of discrete lines segments, and paths on line segments are not consistent with the dynamics of a UAV past a certain level of resolution. Frazzoli's maneuver library consists of motion primitives that are consistent with the dynamics of the vehicle so that any path formed by an appropriate sequence of primitives and trim conditions is a dynamically feasible path for the vehicle.

This thesis proposes the use of tactical primitives, which are system-feasible actions that are based on environmental features. The phrase "system-feasible" signifies both maneuvers that are dynamically feasible and other actions that a system can perform, such as targeting, arming a missile, and firing a weapon.

Tactical primitives can be built from of body-frame primitives, but they may offer an advantage over body-frame primitives because they are tied to the environment. This distinction may make a difference when they are paired with a stochastic search algorithm. If a maneuver library contains only right and left turns in the body frame, an algorithm like GA or RRT will test many trajectories that steer an autonomous vehicle (AV) into an obstacle. If the maneuver library contains primitives like *follow wall #3*, *skirt threat #6 on the left*, or *go to the goal*, the search algorithm will concentrate its search on useful trajectories and avoid many infeasible or undesireable trajectories.

2.1.1 Choice of Primitive Set

Choosing what tactical primitives to include in a primitives library is a matter of engineering judgment and testing. No method exists for the automatic generation of maneuver primitives or the evaluation of their performance. Frazzoli defines the controllability of a maneuver set as the ability to reach any desired goal state in finite time [18]. Thus, the library should be rich enough to accomplish all desired behaviors, and it should contain maneuvers for navigating possible constrained situations and actions for accomplishing problem objectives. In addition, the library should be as small as possible to reduce the size of the problem search space. A trade-off exists between richness and compactness.

Grouping linked actions into single primitives makes the primitive library more compact. Hickie's work centered on packaging simple actions together into coherent maneuvers using statecharts [23]. A helicopter attack tactic consisted of coordinated positioning, altitude changes, firing, and egress. For instance, when firing on a target, a vehicle must first reach the target, so the approach maneuver and firing action can be paired into one primitive. When deploying countermeasures, the vehicle must keep the flare or chaff between itself and the threat, so the countermeasures action and egress maneuver can be linked into one primitive. Grouping linked actions into single tactical primitives preserves a library's small size while including tactics that encompass a wide variety of desired behaviors.

Within a particular tactic, there may be variations. One way to include these variations is to include a primitive for each variation, as Frazzoli does. In Frazzoli's framework, a right turn with one radius was a separate maneuver from a right turn with another radius, even though the two actions are similar. Another method for creating and classifying maneuvers is to introduce continuous parameters that define the variations, as Dever does [16].

2.1.2 Primitive Parameters

Dever's extension of Frazzoli's maneuvers supports the idea of using continuous variables to parametrize tactical primitives. The result of parametrization is a richer set of possible behaviors with the addition of few new dimensions or even no new dimensions. For example, Frazzoli created transition maneuvers between steady trim conditions [18]. Among others, a rotorcraft has trim states at hover and at forward flight. A possible parameter in a transition from hover to forward flight is the final velocity value. In Frazzoli's framework, a discrete set of final velocities would define a set of separate maneuvers, but with continuous parametrization, the discrete parameter is replaced with a continuous variable representing final forward velocity, creating a maneuver class with a single continuous parameter, expanding the set of achievable final velocities and adding hierarchical organization.

One disadvantage of continuous parametrization is that many optimization algorithms cannot handle both discrete and continuous variables. The tactical primitives problem is a hybrid problem, having a discrete set of possible actions, both discrete and continuous action parameters, discrete events, and continuous dynamics. In graph search and RRT algorithms, continuous parameters must be discretized by regular or random discretization. Nonlinear programming (NLP), simplex search, and gradient methods do not handle discontinuities well, and discrete variables do not fit into their frameworks. Algorithms that handle both discrete and continuous parameters include stochastic algorithms such as GA or simulated annealing and mixed continuous-discrete algorithms such as mixed-integer linear programming (MILP). Schouwenaars used MILP to generate optimal trajectories for constraints and objectives that could be written in linear terms [42]. Further discussion on optimization methods follows in Section 2.3.3.

2.2 Reactive Tactics

When encountering a new event such as a pop-up threat, a fast AV reaction time could greatly improve performance, preserve health, and sustain the ability to achieve mission objectives. Some online planning algorithms have small enough solution times to generate online trajectories from scratch, but it is unknown how their performance scales with problem complexity or how well they find the global optimum in the presence of many local optima [35]. With offline design of tactical primitives for evasive situations, algorithms for responding to pop-up threats reduce to a means of selecting an appropriate primitive and its parameters.

Choosing what tactic to execute in response to an unexpected event can be done in several ways. The most straightforward method is to reduce the number of possible actions to a small, pre-determined set that can be evaluated online. For example, when a threat is detected, a UAV can evade the threat by dodging behind one of the nearest obstacles, fire on the threat as a target of opportunity, or even disregard the threat and continue as planned. Each action will have variations, such as whether to go left or right, or how closely to approach the threat before firing the first shot. If there are few enough possible actions, each one can be evaluated online in the vehicle's environmental model according to some scoring function, and the best performing action can be chosen for execution. If many options must be considered, determining certain relationships beforehand will reduce the number of scenarios that must be evaluated online. For example, it could be determined that a threat with a particular effective weapons range r_{threat} typically should be fired on at distance $d(r_{\text{threat}})$ or within some small range of distances for maximum payoff. This information could reduce the number of online evaluations and therefore the amount of necessary online computation.

Another method is to use a classifier system to reduce the dimensionality of online optimization. In this method, the optimal primitive and parameter choices are computed offline for many points in the problem space, such as distance to threat, fuel level, distance to goal, etc. This sampling of the problem space can be done in a Monte Carlo fashion, at regular spacings, or in some adaptive way. Once the mapping of states to primitives and parameters is done, the state space can be partitioned with a classifier system, such as a support vector machine (SVM). An SVM takes labeled data and creates a partition matrix describing the boundaries between data of different labels [14]. When an SVM is trained to map states to the optimal action choice, an AV can query the SVM with its current state to discover what action to execute in the event of a pop-up threat.

A third possible method is the use of learning classifier systems (LCSs). These are algorithms for evolving rules that map states to actions. Each rule has the form `IF (...) THEN (...)`, where each `IF` statement involves whether the sensed state is in a particular set, and each `THEN` statement involves an action or set of possible actions.

Combinations of the above methods are possible. For example, near a partition boundary of an SVM, it might not be clear which tactical primitive to choose because Monte Carlo evaluations do not test every point in the state space. In this case, the AV can evaluate the two nearest actions in an online simulation before executing. With an LCS, if there are several rules that tie, the action associated with each rule could be evaluated and compared before execution.

2.3 Genetic Algorithms and Deliberative Tactics

GAs stochastically search a problem space for optimal solutions using a very general multi-point method that mimics natural evolution. For many problems that give analytical methods difficulty due to high dimensionality, nonlinearity, or multiple local optima, GAs find near-optimal solutions in little time. They are well-suited for planning using tactical primitives because they can handle hybrid problems implicitly through a fitness function. An overview of GAs and a description of design issues in using GAs with tactical primitives is given in Appendix A. The following are additional points to consider when applying GAs to planning using tactical primitives.

Because tactical primitives are sequential actions, it makes sense to encode them into the GA chromosome as a list. Between primitives, the number of parameters can vary, and some can be discrete while others are continuous, making them difficult to encode into a chromosome that can be used with an off-the-shelf GA software program. In addition, sequences will be of different lengths, whereas most off-the-shelf GA libraries use chromosomes of fixed length.

Two encoding methods can solve these problems. The first is to only encode a fixed number of variables into the chromosome. To do this, actions can have their number of parameters reduced by careful design, segments for actions involving few parameters can be padded, or fewer parameters than are necessary can be encoded for each action, and then, when evaluating an action, remaining variables can be optimized on a small scale.

The second approach is to use a *null* action. This way, the length of the chromosome can be fixed at a certain number of actions, and fewer actions can be encoded by padding the chromosome with *null* actions. The *null* action along with a method for fixing the number of parameters mentioned above allows the problem to be fit into a standard GA optimization program.

Because tactical primitives operate on environmental features, and GA selects which features to interact with on the fly, a means of choosing features must exist. One idea is to label each feature with a unique identification number. Labeling must be tailored to the application. For discrete items such as threats and obstacles, labeling is straightforward; each element receives a unique number. However, in some problems, labeling features is not straightforward. For example, for threat evasion in the presence of terrain, features could be waypoints at low-altitude positions. These waypoints must be chosen to yield trajectories that achieve terrain masking, and evasion maneuver primitives and waypoints together must be capable of forming trajectories that yield good fitness. All of these principles are a matter of careful design.

To leverage the power of GA mutation, nearby features should have identification numbers that are similar to each other so that when using small mutations, changes in trajectories and actions remain small. GA can yield good solutions without such labeling, but solution times could be longer and final fitness values lower. One way to accomplish this is to connect features in a graph and apply a special mutation operator that mutates a feature number by one step in the graph. Mutations would then yield minor path changes and the fitness landscape would be smoother.

2.3.1 Initial Solution Candidates

GAs have been found to achieve higher fitnesses faster when seeded with good initial solution guesses [1, 44]. For deliberative tactics, fast initial guesses can be constructed using tailored algorithms operating on full or reduced environmental models. For instance, for navigating among circular threats, one solution is to seek the shortest path that incurs no threat exposure [5]. This solution will not be optimal when nonlinear exposure, multiple tactical primitives, pop-up threats, varying velocity, and extra objective function elements are introduced, but it can give a good initial candidate from which a GA can evolve a better solution.

In addition to overall chromosome initialization, initialization can occur on a smaller scale. Generally good initial primitive parameter values can be chosen off-line. A single primitive can be optimized for a variety of situations, and the mean value can be used as the starting value whenever the primitive is initialized.

2.3.2 Modeling Uncertainty

Uncertainty exists in almost every real environment. It comes in several forms: estimation uncertainty, parameter uncertainty, hidden states, noise, etc. There are many methods for estimating or bounding uncertainty in control systems, such as Kalman filtering or robust control.

To handle hidden states and probabilistic events in the objective function, one can use Monte Carlo simulation [40], or if the number of outcomes is few enough, one can model every outcome and weight the objective function by each corresponding probability. For example, if a UAV fires on a threat, there is a certain probability of a miss. In a scenario where a single shot is taken, the outcome is a hit or a miss. Simulating the scenario twice, one with each outcome, and taking a sum of the objective function values of the two outcomes weighted by their respective probabilities will give the expected value of the fitness. This probability splitting can be continued each time a probabilistic event occurs. This method is preferable to Monte Carlo because it gives a directly calculated probability, though if there are many probabilistic events or many outcomes to each event, high dimensionality increases computation time to make direct calculation impractical.

This idea can be extended to include more than two outcomes per event. For example, if an objective function includes a time element, and the duration of an enemy engagement varies, the probability density function (PDF) can be used in the final time calculation. If two such events occur sequentially, then their sum is a random variable whose PDF is the convolution of the PDFs of the two individual variables.

2.3.3 Competing Optimization Methods

Hybrid optimization is an area of active research, and no optimization method clearly outperforms all others. Many optimization algorithms can be tailored to operate on hybrid problems, with varying success. Below is a description and comparison of optimization methods as they would apply to the deliberative tactical primitives problem.

Simulated annealing mimics a metallurgical process where a material's microstructure is altered by a controlled heating and cooling process [27]. In this algorithm, a starting point is chosen and its fitness is evaluated. A random nearby point is chosen according to a user-defined function, and its fitness is evaluated. The probability of accepting the new point is a function of the fitness improvement and of a global "temperature" value. As iterations progress, the temperature decreases. The result is that near the beginning of an optimization run, movement in the search space is almost random, and as the temperature decreases, the algorithm increasingly accepts higher fitness values and rejects lower fitness values. The idea behind simulated annealing is that more random jumps at the beginning will help avoid local optima, and later improvement-only search will mimic hillclimbing to find the global optimum.

GA and simulated annealing are similar in that they are both stochastic search algorithms that can operate in hybrid search spaces. Annealing algorithms consider a single point at a time, as opposed to GA, which operates on a population and therefore benefits from considering a much larger portion of the problem space at once. The temperature idea can be incorporated in GA if convergence to local optima occurs often for a particular problem. Overall, unless a problem benefits greatly from GA's crossover operator, simulated annealing is competitive with GA.

Graph search methods can be used on tactical primitives problems by forming a search tree that branches at choices of individual parameters [41]. For instance, the first choice could be which environmental feature to visit first, then which primitive to execute, then which primitive parameter to use, chosen from a discretized set. In order to reach a goal with good fitness and low computation time, an informed search method with heuristics should be used. Methods like A* and branch-and-bound return optimal solutions if they use a cost-to-go heuristic that is admissible, that is, it never overestimates the true cost-to-go. If the heuristic greatly underestimates the cost-to-go, however, the algorithm searches more nodes and requires more computation time. Construction of a good admissible heuristic might be very difficult for some problems. In addition, in problems with many maneuver types, environmental features, and continuous parameters, the search tree can be large, making computation times very large. These search algorithms have exponential time complexity when used with poor heuristics, whereas GA uses a fixed number of iterations. One similarity between search methods and GA is that both can return suboptimal solutions before optimization is completed. Overall, GA is much more general than tree search. It can handle continuous parameters directly, and it returns its best solution in less time. On the other hand, tree search algorithms are guaranteed optimal under certain conditions.

Frazzoli used RRTs to plan paths using maneuver primitives [18]. The original RRT algorithm forms a tree of possible trajectories by repeatedly choosing a random configuration and expanding the nearest node in the tree towards that configuration in a dynamically feasible manner. RRTs work well for maneuvering in problems where cost involves distance only. When costs are path-dependent, such as in the case of navigating threats whose exposures vary with distance, it is unclear how to expand the tree, as a tailored distance metric involving integral costs is necessary. Additionally, because RRTs build on existing tree nodes, if the first path to a particular configuration has low fitness, it can be difficult to improve the path to that configuration. Finally, RRTs would need to be tailored to handle the continuous and discrete variable mix of tactical primitives, whereas GAs commonly fit more naturally.

If the continuous part of a tactical primitives problem can be represented or approximated as having linear costs and constraints, MILP algorithms might be appropriate methods. MILP can be used for real-time, optimal, online planning for some hybrid problems [16, 42]. However, many problems cannot be well-approximated in linear form. Mixed-integer nonlinear programming (MINLP) involves the global optimization of nonlinear functions with nonlinear constraints and both discrete and continuous variables. For larger problems, however, computation time is infeasible for real-time applications.

Several methods that work only on continuous variables can be combined with methods for discrete variables to solve hybrid problems. For example, nonlinear simplex search, also known as the amoeba algorithm, places a simplex of points around a specified region and contracts the simplex around the optimal value until its size is within a specified tolerance [36]. The amoeba algorithm works even in the presence of some discontinuities, but it is very expensive to evaluate. To mitigate computational expense, the accuracy of the final solution can be sacrificed, as the simplex method converges rapidly in its first few iterations [29]. Another continuous optimization method is gradient search [9]. Gradient search can be faster than simplex, but it cannot optimize in the presence of discontinuities. For hybrid optimization, algorithms such as GA or tree search can choose values for discrete variables, and once the discrete variables are fixed, amoeba or gradient search can optimize over continuous variables.

Optimization algorithms other than GAs might be more suitable in certain cases, but in general, GAs work well for more problems with less tailoring. GAs provide no guarantee of an optimal solution, but no other nonlinear, hybrid optimization provides this guarantee, either. For online tactics generation, a good solution in short time is better than an optimal solution in longer time.

This chapter discussed several aspects of approaching AV tactics as a hybrid optimization problem. It considered two pieces: planning using GAs on tactical primitives and reacting to pop-up events. The next chapter will synthesize the elements discussed in this chapter into an algorithm for both deliberative and reactive planning.

Chapter 3

A Reactive/Deliberative Planner

The last chapter discussed several design factors for the formulation of a combined reactive/deliberative planner. This chapter synthesizes these factors, beginning with separate reactive and deliberative modules, putting them together into a combined algorithm, and outlining how an autonomous vehicle (AV) would use the algorithm with tactical primitives. The goal of this chapter is to present and explain pseudocode for the combined planner.

To set the stage for the planner, Figure 3.1 shows the architecture of an AV and its interaction with the environment. The vehicle senses the environment and updates its internal environmental model. The environmental model contains data relevant for planning and executing plans, such as data about other vehicles, terrain, weather, etc. The planner uses this model and knowledge about the vehicle's state to create a plan composed of a sequence of tactical primitives. The executor takes the primitive sequence and combines it with the environmental model to form actuator commands, which both modify the vehicle's state and affect the environment. Internal to the executor are the vehicle model and feedback controller to steer continuously varying states as well as controllers for the vehicle's discrete actions, such as targeting, weapons, and countermeasures subsystems.

The planner exists to solve the problem stated in Section 1.2, that is, to maximize a user-defined fitness function in an environment with unknown features. The deliberative planner uses available computation time to generate high-performance plans using the vehicle's knowledge of the environment, while the reactive planner quickly modifies the vehicle's plan when its environmental model changes in a way that affects the plan's fitness.

3.1 Deliberative Planning

The deliberative planner uses a genetic algorithm (GA) to choose a sequence of primitives and their corresponding parameters based on the AV's initial condition, environmental model, and fitness function. Once initialized with a population of candidate solutions, GA will attempt to continuously improve the plan with each generation. As a result, the deliberative planner can run the GA as a side process indefinitely,

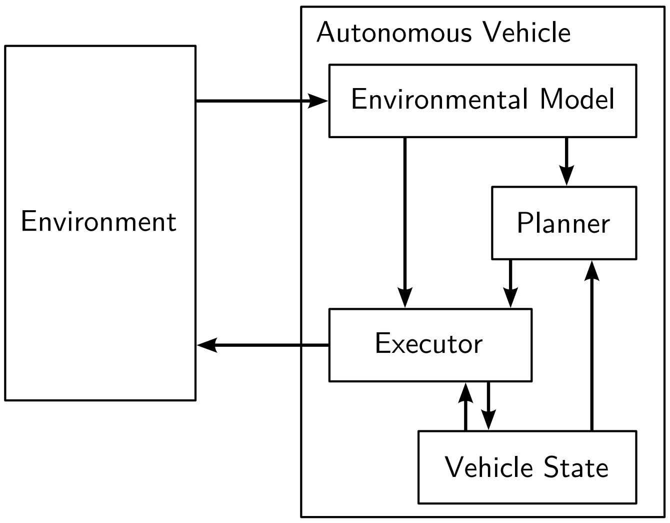


Figure 3.1: Vehicle architecture.

and the vehicle controller can request a plan from the deliberative planner at any time.

When the AV updates its environmental model using new sensor data or external communications, the existing plan becomes out of date and might not perform well on the new model. This would be the case for a pop-up threat or the detection of a new target of opportunity. The deliberative planner must restart and evolve a new plan with the new model. Often, the new data will change the model only slightly so that when using the previous plan as a seed, the GA can evolve a new plan with an appropriate fitness level within a small number of generations. In other cases, the model might change in a way that allows the deliberative planner enough time to create a new plan that performs well. When there is not enough time to replan using GA, a faster method is necessary, and the reactive planner given below seeks to fill this gap.

When viewed as a side process, the deliberative planner can be represented by two functions: startDeliberativePlanner, which requires the environmental model, an initial state, a previous plan, and a previous GA population of plans, and returns nothing; and getDeliberativePlan, which takes no arguments and returns the best plan found since the start of the GA run and the latest population of plans. After the GA is started with startDeliberativePlanner, the GA runs until it is restarted or stopped.

Details about problem representation and GA chromosomes for tactical primitives are given in Sections 2.3 and A.2.1. GA is not necessarily the only method that will work with the deliberative planner; several other methods are discussed in Sections 2.3.3 and 5.3.

```
input  : environmental model model, AV state state, previous plan plan
output: modified plan plan
1 candidateList ← generateCandidateList(model,state,plan);
2 bestfitness ← evaluateFitness(plan);
3 foreach candidate in candidateList do
4     fitness ← evaluateFitness(candidate);
5     if fitness > bestfitness then
6         plan ← candidate ;
7         bestfitness ← fitness ;
8     end
9 end
```

Figure 3.2: Pseudocode for generateReactivePlan.

3.2 Reactive Planning

A sudden, unforeseen event might cause a drop in the fitness of the AV's plan or create the opportunity for an increase in fitness, but the deliberative planner's GA might not find a new plan fast enough to avoid or exploit the situation. The reactive planner's purpose is to mitigate against this poor performance by modifying the plan faster than the deliberative planner. It works by evaluating a very small set of plan modifications and choosing the modified plan with the best fitness according to the user-defined fitness function.

The pseudocode in Figure 3.2 illustrates the operation of the generateReactivePlan function. Line 1 includes a subfunction for generating a list of modified plans. This subfunction is user-designed and tailored to the specific problem. The generateReactivePlan function then calculates the fitness of each plan in the list and returns the candidate plan with the highest fitness.

Because there are few options to evaluate, the reactive planner is much faster than the deliberative planner, but the reactive planner offers no advantage unless its candidate list contains modified plan elements that increase the deliberative plan's fitness. The generateCandidateList should modify plans in ways that are appropriate for the situation. For example, the candidates for reacting to a pop-up threat would be different from those for discovering a target of opportunity. In addition, offline computation and logic can be used to generate fast rules that narrow the choice of primitive parameters according to the environmental model and vehicle state. Section 2.2 discusses two possible rule generation methods, support vector machines and learning classifier systems, which can narrow the candidate list to contain plan modifications that are likely to increase fitness.

```
input  : environmental model model, AV state state, previous plan plan,
         GA population of plans pop =NULL, planning period $T$
output: new plan plan, new population pop
1  plan ← initializePlan(model,state);
2  event ← NULL;
3  while event is not a terminationevent do
4      nextstate ← predictNextState(model,state,plan,$T$);
5      startDeliberativePlanner(model,nextstate,pop,plan);
6      event ← waitForEvent($T$);
7      if event is a reactiveevent then
8          state ← getCurrentState();
9          model ← getCurrentWorldModel();
10         plan ← generateReactivePlan(model,state,plan);
11     else if event is a planrequest then
12         (plan,pop) ← getDeliberativePlan();
13     end
14 end
```

Figure 3.3: Pseudocode for the combined planner.

3.3 Combined Planner

The deliberative module generates tailored plans according to the AV's environmental model, and the reactive module quickly modifies plans using user-designed candidates. The combined reactive/deliberative planner uses both modules together for better performance in a nonlinear, varying environment.

Figure 3.3 gives the pseudocode for the combined planner. First, initializePlan generates a plan using the deliberative module or some other means. Then, the deliberative planner runs iteratively. The deliberative planner needs an initial vehicle state, but because the GA takes time to run, the vehicle's state will change between the time that the GA is started and the time that the combined planner requests a plan. The initial state given to the GA must be a future state. The function predictNextState takes the AV's current plan and calculates where the AV will be after a certain period, and when that period has ended, the planner calls getDeliberativePlan and retrieves it.

The function waitForEvent illustrates how the AV executes the plan until an event occurs. This event could be an environmental model update, a plan request from the executor, or a statement that the goal has been reached. The waitForEvent function returns a planRequest event when the vehicle reaches nextState after a planning period of T seconds.

The duration of the planning period T is a design choice and a trade-off between time to improve the GA result and how quickly new plans are implemented. If the period is long, the GA has more time to improve the plan proceding from nextState, but an unexpected event is more likely to interrupt the planner before the plan can be retrieved and implemented. If the period is short, the planner is less likely to be interrupted, but the GA must be restarted frequently with slightly different starting states, possibly preventing the GA from converging to a good plan.

When the event is a reactiveEvent, the planner calls generateReactivePlan and goes back to the top of the *while* loop to restart the deliberative planner. When the reactive event comes from something that will affect the AV within a short time horizon, the reactive modification might improve the plan fitness while the GA does not have enough time to do so. When the event is an environmental model update that does not immediately affect the AV, the reactive event essentially restarts the deliberative planner with the new environmental model.

If the environment contains many unknown features, and the AV discovers them in rapid succession, the vehicle will never request a deliberative plan, and the plan will only be modified by generateReactivePlan. Because of the limited nature of generateReactivePlan's candidate list and the complexity of the environment, the planner has the potential to perform very poorly. This is a shortcoming of the architecture. Shortening the planning period in the presence of many reactive events or limiting the vehicle's planning horizon to a small area around the vehicle and disabling reactive events might improve performance. With a smaller horizon, the GA might run faster and return plans at a rate that will preserve the vehicle's fitness.

In the next chapter, this combined reactive/deliberative planner is applied to a simulated vehicle navigating regions with threats and obstacles. This simulated vehicle has the option of firing on threats or deploying countermeasures to prevent the threats from causing harm, making the problem hybrid in nature. Several properties of the planner are discussed at the end of the next chapter and in the chapter following, including performance of the GA algorithm in several scenarios, computational properties of the algorithm, using nonlinear programming mixed with GA or as an alternative to GA for optimization, the use of alternative primitives libraries on the same problem, and comparison of the planner on different vehicles.

Chapter 4

Threats Problem

The threats problem is a kinematic, single-vehicle simulation in two dimensions for testing and validating the algorithm presented in the previous chapter. It contains a nonlinear, hybrid fitness function, unexpected events for testing the reactive portion of the planner, path planning for a vehicle with continuous dynamics, and discrete events. This chapter presents a detailed description of the simulation and several simulation results for three different unmanned aerial vehicle (UAV) scenarios: crossing a field of threats, attacking targets while avoiding a no-fly zone, and navigating an urban canyon. The first scenario generally tests the algorithm using threats alone. The second adds an obstacle and a different fitness function. The third stresses the algorithm by not giving the UAV any clear paths and forcing it to interact with the threats. Together, they demonstrate the ability to plan in complex, nonlinear environment with constraints and varying costs.

4.1 Problem Description

Figure 4.1 shows an overview of the threats problem. A single UAV must fly to a goal position as quickly as possible while minimizing exposure to threats. The UAV can fire a weapon to attempt to disable threats, or it can deploy countermeasures to decrease its exposure to nearby threats. To incorporate the planner from the previous chapter, a genetic algorithm (GA) selects tactical primitives to control both the UAV's maneuvers and the discrete actions. The following pages contain descriptions of the simulation, the tactical primitives, the use of GA for planning using primitives, and the design and implementation of reactive maneuvers.

4.1.1 Vehicle Model

The UAV moves in two dimensions and has a simple kinematic model. The UAV's motion consists of a sequence of circular arcs and line segments. The UAV's velocity can vary between a specified minimum and maximum, and a maximum acceleration value bounds both acceleration and deceleration. A minimum turning radius parameter constrains the turning radius. This model ignores aerodynamic effects and actuator

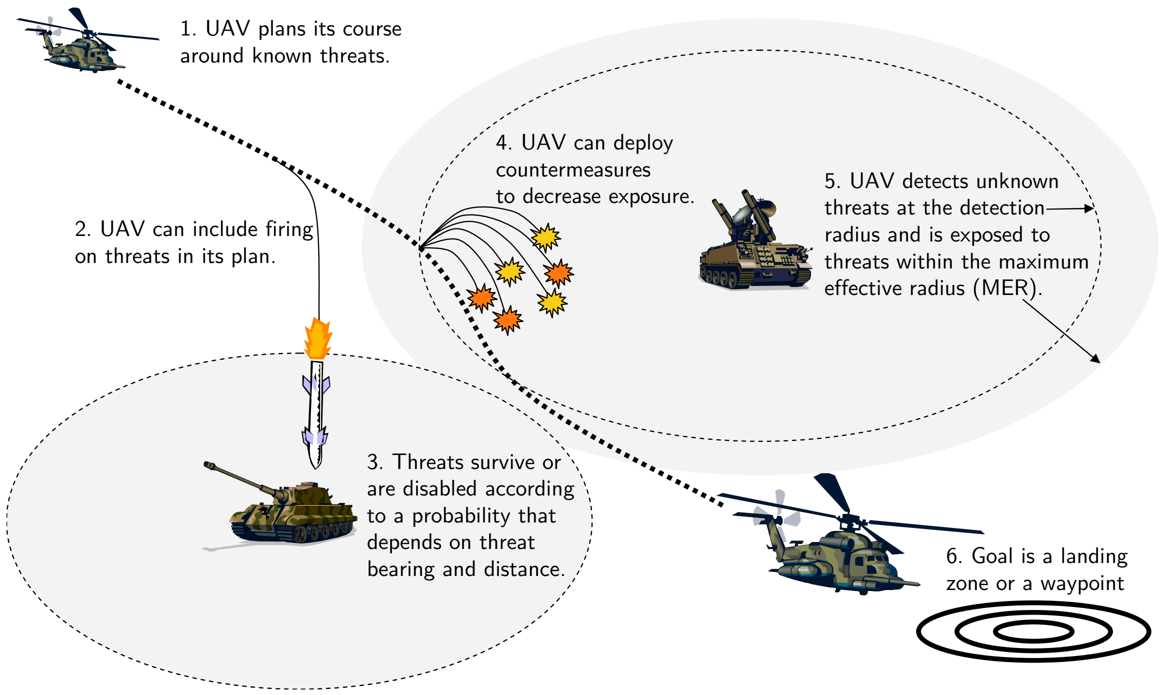

1. UAV plans its course around known threats.
2. UAV can include firing on threats in its plan.
3. Threats survive or are disabled according to a probability that depends on threat bearing and distance.
4. UAV can deploy countermeasures to decrease exposure.
5. UAV detects unknown threats at the detection radius and is exposed to threats within the maximum effective radius (MER).
6. Goal is a landing zone or a waypoint

Figure 4.1: In the threats problem, the UAV navigates to reach the goal quickly while minimizing exposure to threats. It also can fire on threats and deploy countermeasures.

and sensor noise. Table 4.1 defines the constants used in this chapter's simulations.

The exclusion of significant dynamic effects and sources of noise from the vehicle model is a drawback, but the threats problem remains an effective test for the planner. Though a high-fidelity simulation would validate the planner's performance to a greater extent, Frazzoli and Dever have already demonstrated that kinematic planning with pre-selected, dynamically feasible maneuver primitives yields trajectories that function well in both high-fidelity simulation and actual experiment [16, 18]. The methods of this thesis are general enough to accommodate a different parameter set than the one in Table 4.1 or even a library of primitives that uses dynamics of higher fidelity. Also, though the UAV's minimum turning radius is not a function of its instantaneous velocity, the variable velocity nonetheless adds extra dimensions to the problem that test the deliberative planner's ability to handle more dimensions. One advantage of the kinematic nature of the simulation is that it allows for very fast computation of entire plans and their fitnesses, which is necessary for the many evaluations that GA requires.

4.1.2 Threat Model

In this thesis, threats are stationary entities that decrease the UAV's fitness when the UAV passes within a certain distance. Each threat begins a simulation as either known or unknown, and each one has a position, a maximum effective radius (MER), and, if unknown, a detection radius. When the UAV enters the circle defined by

Vehicle		**Fitness**	
Minimum velocity (m/s)	0.3	Time weighting	1
Maximum velocity (m/s)	3	Exposure weighting	10
Velocity threshold (m/s)	1.5	Target bonus weighting	100
Acceleration limit (m/s^2)	1	Weapon cost weighting	2
Minimum turning radius (m)	1	CM cost weighting	2
Countermeasure (CM) range (m)	5	Velocity penalty weighting	0.33
CM max exposure reduction factor	0.8	No-fly zone penalty weighting	10
Firing maximum range (m)	9	Point spacing (s)	0.3
Firing probability peak	0.95	**Deliberative Planner**	
Firing time (s)	0.5	Primitives in chromosome	6
Threats		Population size	80
Exposure function peak	1	Planning period (s)	2.5
Evasion circle size constant	1.9		

Table 4.1: Simulation parameters used in Chapter 4.

an unknown threat's detection radius, the threat becomes known. The simulation assigns each threat a random integer label.

In simulation, detection of a threat occurs when the UAV's trajectory intersects a threat's detection circle. For a particular segment of the trajectory, intersections of a circular arc or a line segment with each threat's detection circle are calculated analytically [47, 48].

A threat decreases the UAV's fitness by exposure. Exposure is defined as the integral of a risk function along the UAV's trajectory within the threat radius. This expression simulates the probability of a threat detecting and firing on the UAV, because the more time the UAV spends within the threat region, the more probable it is that a threat would detect and attack it. The exposure function used throughout the simulations in this thesis is given by

$$e(t) = \int_{\mathcal{I}} e_{\max}\big(1 - \big(\tfrac{r(\tau)}{r_{\mathrm{MER,t}}}\big)^2\big)d\tau \tag{4.1}$$

where $e(t)$ is the total exposure for a single threat, $e_{\max}$ is a constant defining the peak value of the differential exposure function, $r(t)$ is the UAV's distance to the threat as a function of time, and $r_{\mathrm{MER,t}}$ is the threat's MER. The symbol $\mathcal{I}$ denotes the time interval up to time t where $r(t) < r_{\mathrm{MER,t}}$, that is, where the UAV is within the threat's MER, and where $t < t_d$, that is, for points in time before the threat is disabled, if the threat is disabled. The integrand of Equation 4.1 can be thought of as the incremental threat exposure. The UAV's total exposure is a sum of the exposures of each threat. Figure 4.2 illustrates the exposure calculation.

In simulation, as the UAV flies a trajectory, it records its position at regularly spaced points. To calculate the exposure to a threat, the UAV calculates the incremental exposure at each trajectory point and integrates it using Simpson's Rule [3].

This simulation unrealistically models unknown threats whose detection radii are

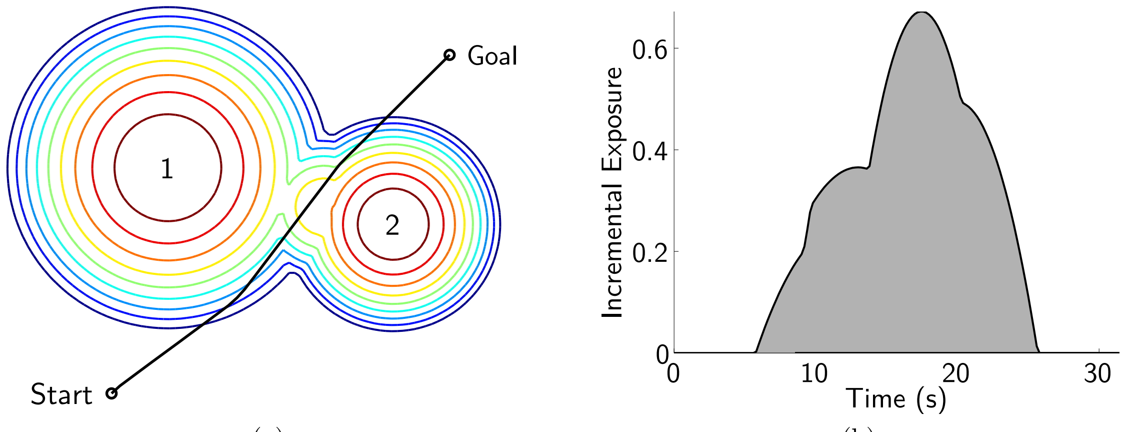


Figure 4.2: Exposure calculation for a trajectory through two adjacent threat regions. The black line represents the vehicle trajectory. As the UAV traverses the threat regions (a), it accumulates exposure. The total exposure is the integral of the incremental exposure, and is therefore equal to the area in grey (b).

smaller than their MERs. In reality, if a threat has a longer detection and strike range than a UAV, then the threat might detect the UAV before the UAV detects the threat. If the threat attacks the UAV, the UAV will then detect the threat. However, in this simulation, the UAV can fly through a threat region without ever detecting the threat, which only leads to high exposure. The UAV does not detect this exposure during flight, but it nevertheless enters the final fitness score.

4.1.3 Firing on a Threat

The UAV can fire a weapon to attempt to disable a threat, thus eliminating future exposure to the threat. In this simulation, there is no partial damage; either the threat is disabled completely, or the UAV misses completely. In addition, there is a fixed cost for each use of a weapon, and there are a limited number of uses. When the UAV fires on a threat, the probability of disabling the threat depends on the UAV's distance to the threat and the bearing, meaning the relative angle between the UAV's heading and the direction from the UAV to the threat. This probability is given by

$$p_{\text{hit}}(d,\theta) = \begin{cases} p_{\max}\left(1 - \frac{d}{r_{\text{MER,UAV}}}\right)\cos\theta & d < r_{\text{MER,UAV}},\ |\theta| < \pi/2 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where p_{hit} is the probability that a particular shot disables a threat, d is the distance to the threat, $r_{\text{MER,UAV}}$ is the UAV's MER, and θ is the bearing to the threat. This model simply represents a forward-firing weapon whose accuracy depends on distance. It is not derived from a real system. Figure 4.3 shows the probability function graphically.

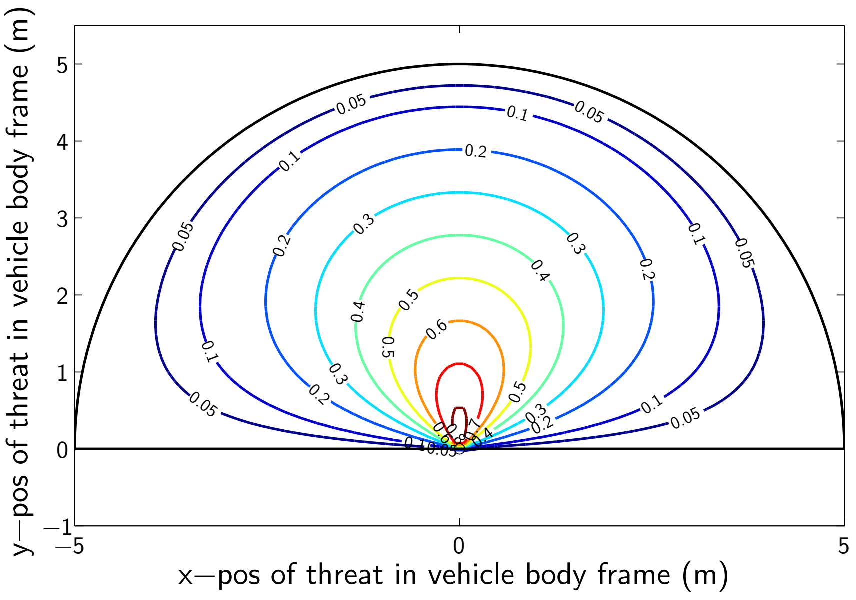


Figure 4.3: Probability of disabling a threat. Distance and bearing determine the probability function, but here contours are shown in Cartesian coordinates. The positive y axis corresponds to the UAV's forward direction.

4.1.4 Countermeasures

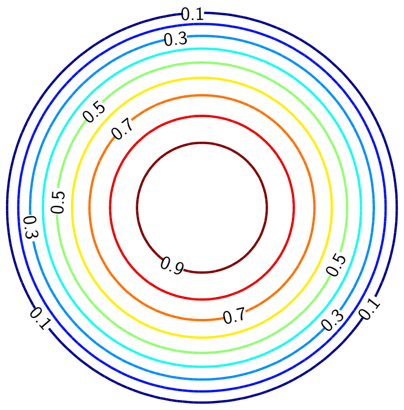
A countermeasure is "a military system or device intended to thwart a sensing mechanism" [34]. In this simulation, the UAV deploys countermeasures that remain in one place and are effective until the simulation ends. Each countermeasure has a fixed cost, and there are a limited number of countermeasures. Countermeasures reduce the exposure functions of nearby threats by a multiplicative factor that depends on the threat distance to the countermeasure. This factor is given by

$$f_{\mathrm{cm}}(r_{\mathrm{cm}}, t) = \begin{cases} 1 - f_{\mathrm{max,cm}}(1 - \frac{r_{\mathrm{cm}}}{r_{\mathrm{max,cm}}}^2) & r_{\mathrm{cm}} < r_{\mathrm{max,cm}}, t > t_{\mathrm{cm}} \\ 1 & \text{otherwise} \end{cases} \tag{4.3}$$

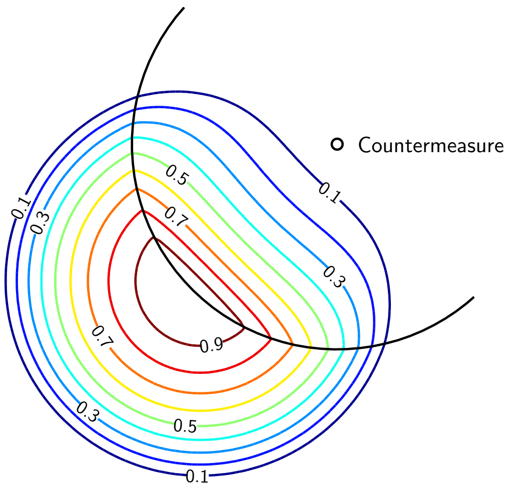
where $f_{\mathrm{max,cm}}$ is the maximum exposure reduction, r_{cm} is the distance to the countermeasure, $r_{\mathrm{max,cm}}$ is the maximum range at which the countermeasure has an effect, and t_{cm} is the time at which the UAV deploys the countermeasure. Figure 4.4 shows the effect of a countermeasure on threat exposure.

4.1.5 No-Fly-Zone

In two of the scenarios at the end of this chapter, polygons represent no-fly zones that the UAV must avoid. Section A.2.4 presents several options for enforcing constraints within the GA framework. Problem space mapping or chromosome repair to avoid violating constraints would be very difficult because limiting trajectories to certain sequences of primitives is a large hybrid and nonlinear problem in itself. Chromosome



(a)



(b)

Figure 4.4: Effect of a countermeasure on threat exposure. Subfigure (a) shows a threat's incremental exposure function, and subfigure (b) shows the same threat with a countermeasure added.

deletion would require modifications of the GA algorithm.

The most straightforward method of including no-fly zone constraints is to heavily penalize trajectories that violate them. The advantage of penalizing trajectories instead of modifying them is that the penalty can capture "how badly" the trajectory violates the constraint, and the GA can use these fitness gradations to move solutions outside of constraint regions during search. Accordingly, the penalty for a trajectory entering a no-fly zone polygon is equal to a penalty constant plus the number of points inside the polygon, given by

$$\mathcal{P}_{\text{nfz}} = \sum_{i=1}^{N_{\text{nfz}}} (\mathcal{P}_{\text{nfz,min}} + n_i) \tag{4.4}$$

where N_{nfz} is the number of no-fly zones, i iterates over each no-fly zone, $\mathcal{P}_{\text{nfz,min}}$ is the minimum penalty for entering a no-fly zone, and n_i is the number of trajectory points inside no-fly zone i. In the simulation, these trajectory points are the same as in Section 4.1.2, which explains how the UAV uses them to calculate threat exposure.

4.1.6 Fitness Function

The fitness function must reflect the vehicle's objective to reach a goal position in the least amount of time while incurring the least exposure from threats. To achieve all desired objectives using an off-the-shelf GA, they must be combined into a single scalar fitness value. The method chosen to handle multi-objective optimization in this simulation is a weighted sum, given by

$$\text{fitness} = \sum_i w_i \mathcal{J}_i \tag{4.5}$$

where w_i represents the weight for the i-th objective, $\mathcal{J}_i$.

In addition to time and exposure, the fitness function includes several other components: a bonus for disabling specific threats that are labeled as targets, costs for using weapons and countermeasures, and a penalty on high velocity to simulate greater fuel consumption. The bonus is a fixed value times the number of targets disabled. Weapons and countermeasures cost a fixed amount for each usage. A penalty for exceeding a specified velocity threshold forces the UAV to make a trade-off between conserving fuel and moving quickly to decrease total time and exposure. This velocity penalty is given by

$$\mathcal{P}_v = \int_{t_i}^{t_f} \max\{v(t) - v_{\text{threshold}}, 0\} dt \tag{4.6}$$

where t_i and t_f are the initial and final time, $v(t)$ is the velocity magnitude, and $v_{\text{threshold}}$ is the threshold above which excess velocity is penalized. Because the penalty is integral, short bursts of high velocity above the threshold incur little penalty. In the simulation, the trajectory vector is differenced to yield a vector of velocities, and the integral is implemented as a simple sum of points in the vector.

Altogether, the fitness function is given by

$$\text{fitness} = -w_t t_f - w_e e + w_d n_d - w_{\text{nfz}} \mathcal{P}_{\text{nfz}} - w_w n_w - w_c n_c - w_v \mathcal{P}_v \tag{4.7}$$

where the components represent the fitnesses of time, exposure, targets disabled, no-fly zone penalty, cost of weapons, cost of countermeasures, and velocity penalty, respectively. The variables n_d, n_w, and n_c represent the number of targets disabled, the number of weapons used, and the number of countermeasures used.

4.2 Tactical Building Blocks

Tactical primitives for the threats problem involve basic maneuvering, firing weapons, and deploying countermeasures. Each one has a collection of parameters that completely specify the behavior of the UAV during execution of the primitive. Each primitive is interruptible, meaning that complete execution of a primitive and a series of interrupted executions of the same primitive will have the same behavior. This is important because the deliberative planner interrupts primitives when updating the active plan. Primitives for the threats problem were designed intuitively, but they could have been extracted from observation in human-in-the-loop experiments. See Figure 4.5 for a graphical depiction of each primitive in the threats problem.

4.2.1 Maneuvers

Maneuvers/trajectories constitute the continuous part of the vehicle's behavior, though by using a library of motions with continuous and discrete parameters, maneuvering becomes a hybrid problem. This approach is similar to Frazzoli, who hybridized his problem to turn a purely continuous dynamical problem into a discrete kinematic problem using motion primitives so that he could use rapidly-exploring random trees (RRT) for path planning [18]. Here, each maneuver consists of circular arc segments and line segments, and each maneuver can be interrupted when the UAV detects a new threat or the vehicle's planner interrupts after a certain time period has passed in order to begin execution of a new plan.

During each maneuver, the UAV moves with constant acceleration to a specified velocity parameter. Because acceleration is bounded by a constant, the UAV might not actually achieve the specified velocity by the end of the maneuver. Each primitive has a single velocity parameter for the entire maneuver.

Go to Point

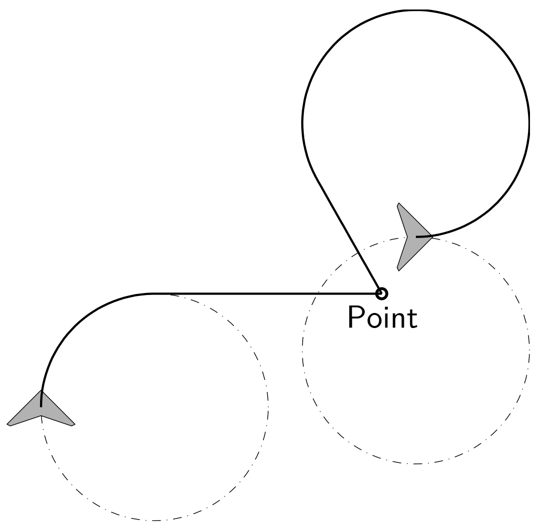
The *gotopoint* primitive is a building block for other primitives and is not executed alone except at the end of every primitive sequence in order to command the UAV to go to the goal position. The *gotopoint* primitive defines its trajectory using four parameters: an (x, y) point, a turning radius, and a velocity value. First the UAV turns toward the point using the specified turning radius, accelerating toward the velocity value. If the UAV's turning radius is large enough to encircle the point, the UAV turns away from the point and circles around until it faces the point. After the turn is completed, the UAV goes forward until it reaches the point, continuing to accelerate to the specified velocity value.

Go to Threat

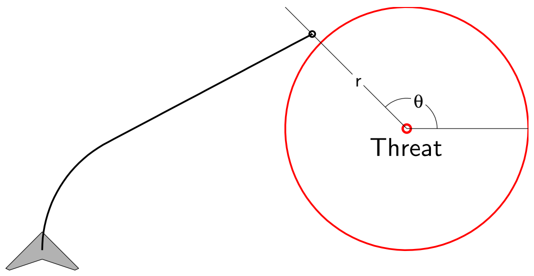
Navigating with respect to the threats themselves has the potential to create trajectories that tightly skirt threat regions, reducing total travel time while still achieving low exposure. The *gotothreat* primitive defines a trajectory using five parameters: an integer label denoting a particular threat, an angle with respect to a threat, a distance to the threat, an initial turning radius, and a velocity parameter. The angle and the distance define a point with respect to the threat, and the *gotothreat* primitive calls the *gotopoint* primitive, passing it the point, turning radius, and velocity parameters.

Threat Evasion

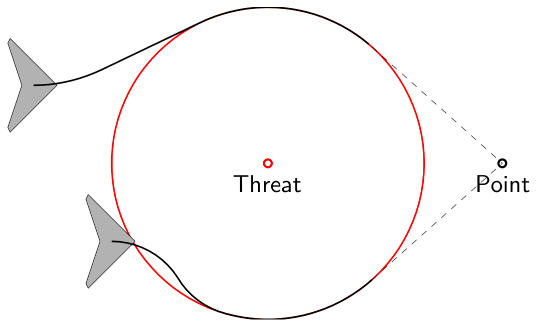
The final maneuver primitive is the *evade* primitive, which steers a vehicle away from a particular threat, around the threat, and towards a designated point. The *evade* primitive is used only for reactive planning and does not enter the deliberative planner's GA unless the deliberative planner interrupts it in order to begin execution of a new plan. The parameters given are a label denoting the threat, the point, a turning radius, and a desired final velocity. The UAV determines its turn direction



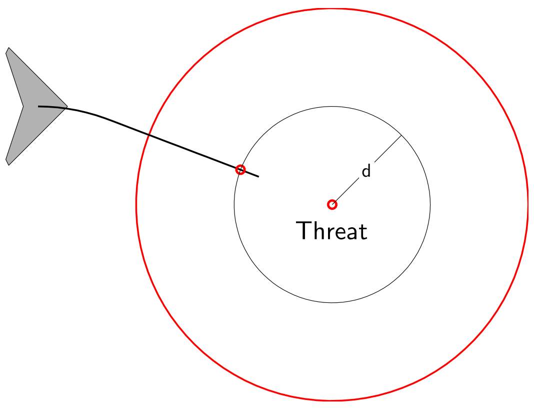
(a) *gotopoint*



(b) *gotothreat*

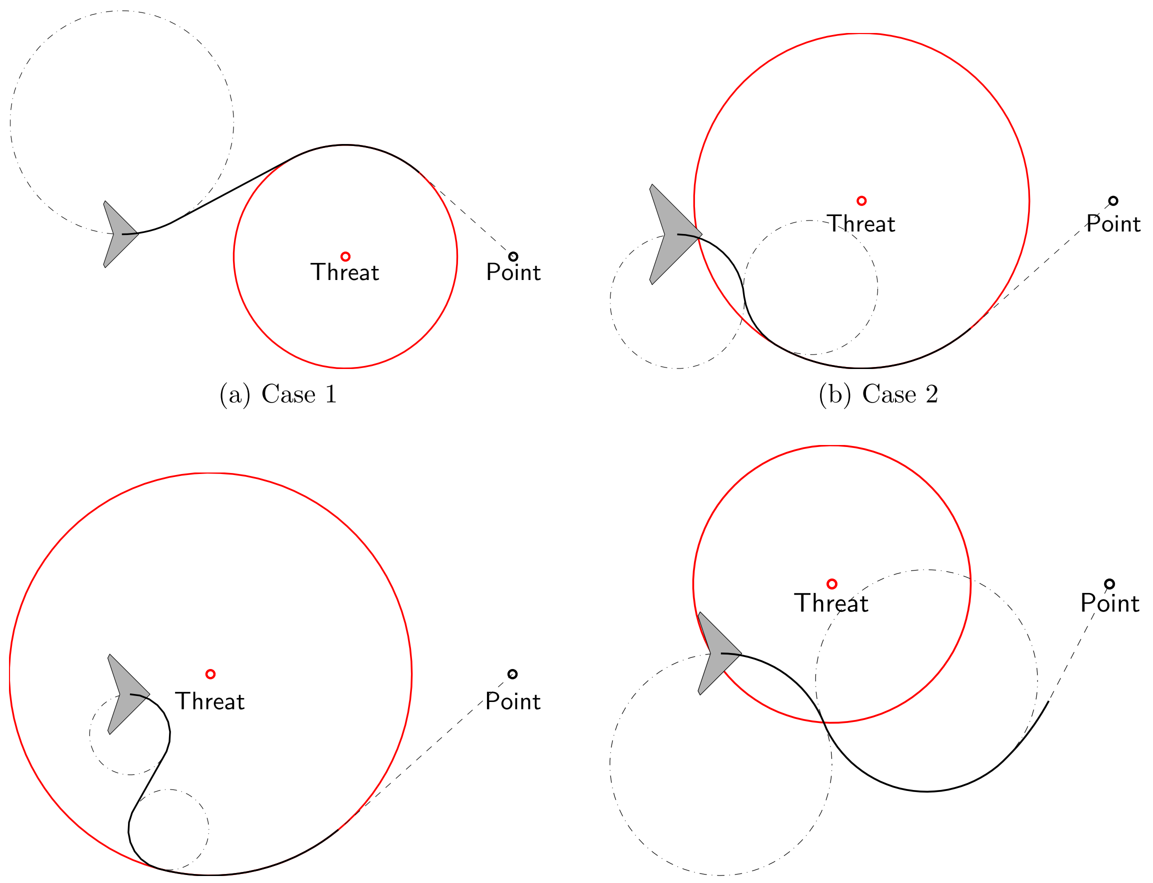


(c) *evade*



(d) *fireonthreat*

Figure 4.5: Tactical primitives for the threats problem. (a) In *gotopoint*, if the closer turning direction encircles the point, the UAV turns in the opposite direction. (b) In *gotothreat*, the UAV flies to a point defined by r, θ, and a threat label using *gotopoint*. (c) In the *evade* primitive, the UAV circles the threat and faces the point. (d) In *fireonthreat*, the UAV fires on the threat when it is within the distance defined by the parameter d, then drifts straight for a small time. The *deploycm* primitive involves the same maneuver as the *gotothreat* primitive and deploys a countermeasure at the endpoint.



(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Figure 4.6: Different cases of the *evade* maneuver.

based on the sign of the turning radius—a positive radius signifies a turn purely away from the threat, and a negative radius signifies a turn that steers initially toward and then past and away from the threat.

UAV behavior during the *evade* maneuver is determined by the relative position and direction of the threat at the beginning of the maneuver and by the size of the threat region. See Figure 4.6 for a graphical representation of the following cases.

Case 1 If the UAV begins the maneuver outside the threat MER, and a turn at the specified turning radius and direction would not make the trajectory intersect the circle defined by the threat's MER, then the UAV skirts the threat circle by traveling along the line segment defined by the tangent line between the turning circle and the threat circle. In the latter case, the UAV then orbits the threat until it faces the point.

Case 2 If the UAV evades a threat whose MER is greater than or equal to some specified constant (the circlesizefactor constant) times the magnitude of the turning radius, and if the turn will intersect the threat circle, then the UAV turns just far enough so that a second turn in the opposite direction at the same radius will put the UAV in position to orbit the threat along the threat circle. The UAV then orbits the threat until it faces the point. The circlesizefactor constant defines a boundary between small and large threats, dividing behavior into cases and preventing the UAV from trying to orbit a threat whose MER is less than the vehicle's turning radius.

Case 3 If the UAV evades a threat whose MER is greater or equal to circlesizefactor times the turning radius, and if the UAV is far enough inside the threat region so that a turn will not intersect the threat circle, the UAV turns away from the threat and travels radially outward until a second turn with the given turn radius will put the UAV in orbit along the threat circle. If the turning direction is such that the threat position is encircled, then the UAV cannot face along an outward radial. This case is tested before the beginning of the maneuver, and when it occurs, the turning direction is changed, and the maneuver is restarted. If the point is inside the threat circle, the maneuver exits after the straight segment. If there is a second turn, the UAV turns in the closer direction of the point and then orbits the threat until facing the point.

Case 4 If the UAV evades a threat whose MER is less than circlesizefactor times the turning radius, and the turn will intersect the threat circle, the UAV turns until it intersects the threat circle on its way out of the threat region. If the turning radius is too small to intersect the threat, the turning radius is enlarged until the turn will intersect the threat region. The UAV then turns towards the point and, if necessary, orbits the threat until it faces the point.

4.2.2 Discrete Actions

In the threats problem, the UAV can both fire on threats and deploy countermeasures. Executing these actions from an arbitrary position is ineffective at increasing vehicle fitness; the vehicle must be near threats in both cases, and in the case of firing, the vehicle must be in a position whose probability of disabling the threat is high. Combining these two actions with preceding maneuvers can increase their effectiveness, and the following two tactical primitives combine these actions with maneuvers to increase their effectiveness.

Fire weapon

In the *fireonthreat* primitive, the UAV approaches a threat and fires a weapon. This primitive takes four parameters: a label identifying the target threat, a distance at which to take the shot, a final velocity value, and a flag that determines whether to execute an *evade* maneuver in the case of a miss. In the case where the UAV starts a firing primitive with no weapons, the primitive exits without any change in the state of the vehicle. If the UAV still has weapons remaining, the UAV executes a *gotopoint* maneuver toward the threat position using the minimum turning radius. When the UAV reaches the specified distance to the threat, it fires on the threat. If the UAV's starting position is within that distance, then the UAV immediately fires. To simulate the UAV taking time to fire a weapon, the UAV drifts forward for a specified time, remaining at the same velocity as when it fired.

The outcome of firing on a threat is probabilistic. When the UAV plans using GA, the *fireonthreat* primitive must model the possibilities of both success and failure. When the UAV fires on a threat during planning, the trajectory branches into two offspring trajectories: one where the threat is disabled, and one where the UAV misses the threat. The first branch is assigned the probability p_{hit} and the second $(1 - p_{\text{hit}})$, where p_{hit} is determined by Equation 4.2, and each of these probabilities is multiplied by the probability of the parent trajectory. This branching with probabilities models the cost of an uncertain event. Each branch continues until the UAV reaches the goal position, and the final fitness is weighted by the probability associated with the branch. In the second branch, if the evasion flag is set, the UAV executes an *evade* maneuver.

During execution of a plan, if the UAV successfully disables a threat, the primitive ends. If the UAV misses, this triggers the UAV to go into reactive planning mode.

Deploy countermeasures

Countermeasures are only effective if they are placed near a region of non-zero exposure (see Figure 4.4. Thus, deploying a countermeasure is linked with a *gotothreat* maneuver. The *deploycm* primitive takes all the parameters of the *gotothreat* primitive. It executes *gotothreat* and deploys a countermeasure at the end of the maneuver. If there are no countermeasures left, *deploycm* is equivalent to *gotothreat*.

4.3 Deliberative Planning

In this thesis, planning is performed using an off-the-shelf GA [25]. In order to plan using GA, a chromosome must be designed to represent a sequence of primitives with their associated parameters, a sequence which defines an entire trajectory. Sections 2.3 and A.2.1 discuss some details about chromosome design, such as padding with *null* values.

The remaining difficulty in representing primitives with a chromosome lies in differences between parameter ranges and types, whether they are discrete or continuous. For example, if a *gotothreat* primitive's fourth parameter is the final distance to the threat, and the *fireonthreat* primitive's fourth parameter is the true/false evasion flag, then when one replaces the other, the fourth position in the chromosome must either be able to represent both continuous and discrete parameters, or it must change the nature of the chromosome depending on the primitive. Because the threat problem uses an off-the-shelf GA, changing the nature of the chromosome is difficult.

Instead, each element in the chromosome is a random number on the interval $(0, 1]$, and the element is interpreted differently based on the primitive. Each primitive is represented by a list of six floating-point numbers. Table 4.2 gives the meaning of each number based on the primitive. The first number represents which primitive to

1: type		**2: threat label**	**3**	**4**	**5**	**6**
null	$(0, 0.25]$	–	–	–	–	–
gotothreat	$(0.25, 0.75]$	evenly spaced intervals map to threats	final distance to threat $(0, 7]$	initial turn radius $(r_{\min}, r_{\min} + 6]$	final angle from threat $(-\pi, \pi]$	desired final velocity $(v_{\min}, v_{\max}]$
deploycm	$(0.75, 0.875]$	*same*				
fireonthreat	$(0.875, 1]$	*same*	firing distance $(0, \text{maxrange}]$	evadeflag $(0, 0.5]$: false, $(0.5, 1]$: true	–	*same*

Table 4.2: The chromosome represents each primitive with six numbers. The first two numbers represent primitive type and threat. The primitive type is determined by which range the first number occupies. For example, a first number of 0.1 would mean a *null* primitive, and 0.875 would mean *deploycm*. The threat is determined by dividing the interval $(0, 1]$ evenly by the number of known, active threats and seeing which range the second number occupies (see Figure 4.7. The primitive type determines the meaning of the last four numbers in this primitive's section of the chromosome.

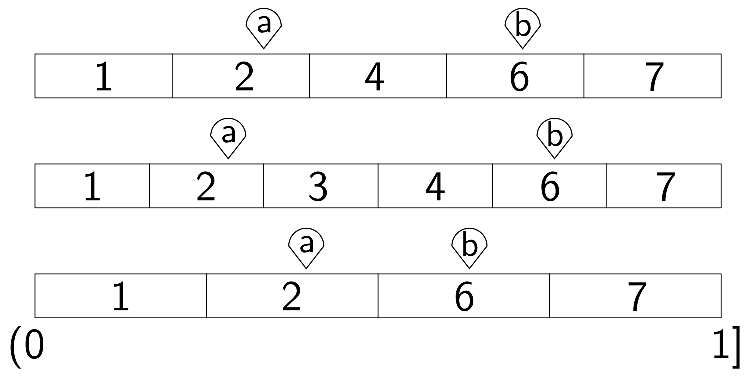


Figure 4.7: Fixing the threat label parameter. Markers a and b represent two floating-point numbers on the interval $(0, 1]$ which represent threats 2 and 6, respectively. The UAV discovers threat 3 between the top line and the second, and then the UAV disables threats 3 and 4 between the second and the third. In each case, the numbers a and b must be scaled to stay in the same position relative to the threats they represent.

execute. Each primitive type has an assigned range, and which range contains the number determines which primitive is executed. For example, if the first element is less than 0.25, then the primitive is a *null* action. The size of each primitives range is equal to the probability of selecting that primitive with a random uniform mutation.

The second number denotes which threat the primitive interacts with. The number's range is divided into equal-sized partitions, with as many partitions as there are known, non-disabled threats. Each partition represents a particular threat, and that threat is represented when the number falls into its partition. As threats are added or disabled, the partitions shift, and each number that represents a threat must be shifted to its appropriate partition in order to preserve the meaning of subsequent primitives in the plan (see Figure 4.7).

The remaining numbers in each primitive have meanings that depend on what kind of primitive it is. Continuous parameters map linearly from $(0, 1]$ to a specified range, and discrete parameters map intervals to discrete values. Putting similar-meaning parameters in the same position for different primitives is important, because switching primitives changes the meanings of the parameters, and having similar meanings can help GA search the problem space. For example, the parameters for *gotothreat* and *deploycm* have the same meaning, and the same parameter is used to denote desired final velocity in all primitives.

One or more primitives in a row define a trajectory from the vehicle's position to a specified goal position. As the UAV completes a non-*null* primitive, the planner trims it off from the chromosome and adds *null* primitives to the end of the plan to preserve the length of the overall chromosome. If the plan is composed entirely of *null*'s, the vehicle executes the *gotopoint* primitive to go to the goal position.

4.4 Reactive Planning

During the execution of plans, new threat detections or missed shots trigger the reactive planning mode. In this mode, the plan is modified in several ways, the modified plans are tested, and the plan with the best fitness is chosen as the new plan. The following list gives the reactive plan modifications for the threats problem. Certain modifications require the plan to contain at least one primitive; if the original plan is empty, the modification is not performed.

- No plan modification.
- Trim off the first primitive, if the plan is at least one primitive long.
- Add evasion to the near side of the threat to the beginning of the plan.
- Replace first primitive with evasion to the near side.
- Add evasion to the far side.
- Replace first with evasion to the far side.
- Add firing at various distances with evasion on miss.
- Replace first with firing at various distances with evasion on miss.
- Add firing at various distances without evasion on miss.
- Replace first with firing at various distances without evasion on miss.
- Add going straight for various distances and deploying countermeasures.
- Replace first with going straight for various distances and deploying countermeasures.

The reactive planner creates a list of these plans, evaluates each one on the environmental model, and chooses the plan with the highest fitness. Figure 4.8 shows a plot of the plans in this list with the highest-fitness plan in bold. The number of plans in this list is small compared to the number of plans evaluated during a period of deliberative planning, taking much less time to compute. Whereas the deliberative planner completes on the order of one thousand evaluations, this list has about twenty plans total. Plans that involve firing or countermeasures are removed from the list when weapons or countermeasures are exhausted, respectively.

4.5 MATLAB and the Genetic Algorithm Optimization Toolbox

The threats problem is written in the MATLAB® programming language, and GA optimization is performed using the Genetic Algorithm Optimization Toolbox (GAOT) [25, 32]. GAOT is used to optimize an array of floating-point numbers on the interval $(0, 1]$, which represents a chromosome of a fixed number of primitives. The entire chromosome is composed of six numbers for each primitive and its parameters, times

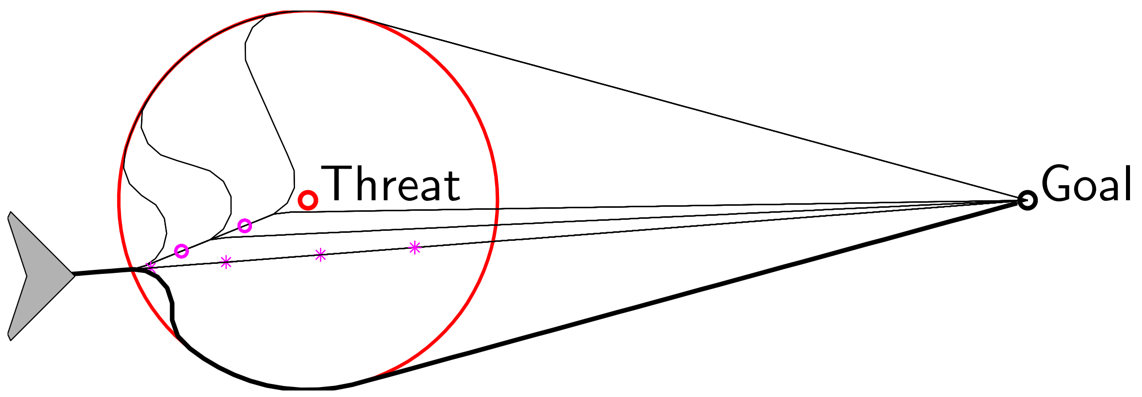


Figure 4.8: Candidate reactive plans. The plan with the highest fitness is shown in bold. Magenta circles denote positions from which the UAV fires, and magenta stars denote countermeasures.

population size	80
termination test	terminate after 30 generations
selection method	geometric with $r = 0.08$ (see Section A.1.4)
crossover methods	arith, heuristic, simple
mutation methods	multiNonUnif, nonUnif, unif

Table 4.3: GAOT parameters for threat problem simulations.

the number of primitives, which is typically six, plus one number at the end to represent the final turning radius to go to the goal position. The choice for the number of primitives was determined by taking the typical number of non-*null* primitives used in the simulations of the scenarios in this thesis and adding several more. For longer distances or more threats, more primitives could prove useful, but keeping the number of primitives low and using a receding horizon approach would preserve a lower problem dimensionality, which might aid the GA in finding solutions.

The runtime parameters for GAOT that were used in simulations for the threats problem are given in Table 4.3. Most of these parameters are the default GAOT parameters. Various mutation operators might favor certain regions of the interval, leading to undesireable bias in discrete parameter selection. For mutation of an unstructured list of discrete numbers, non-uniform random replacement mutations have no phenotypic proximity to leverage. However, because the chromosome contains both discrete and continuous parameters, and GAOT does not change mutation type for different parameters, a balance between effectiveness on discrete and continuous parameters must be found. To this end, boundary mutation for floating-point numbers was disabled because it would cause the GA to favor the first and last primitives and threats in their respective lists.

GAOT returns the best solution found during the course of the GA run, the final population, a trace of the best population, and a matrix of the best and mean fitnesses for each generation. The best solution is the plan that the vehicle executes. The final population of a run is used to seed following runs of the GA; see Figure 3.3. The last two outputs are useful for analyzing GA behaviors such as convergence and rate of improvement.

Though the simulation does not run like an actual real-time system, where the GA optimizes a new plan while the vehicle executes the current plan, the GA planner and kinematic vehicle simulator mimic real-time execution by running the GA when an event occurs and scaling the number of GA generations by the ratio of available time to the length of the planning period. For example, if the GA usually takes 2.5 seconds to run 100 generations, and a primitive is exited 1.25 seconds after the last event, then the GA runs for only 50 generations.

4.6 Simulations

The rest of Chapter 4 contains simulations of the threats problem for various scenarios and presents data about GA convergence, performance against a simple baseline algorithm, consistency in finding solutions, and best and worst case runs for each scenario. First, an example run of the combined planner on simple threat arrangments is given to familiarize the reader with the simulation and plotting conventions.

4.6.1 Example Scenario

Figure 4.9 shows a scenario with several threats, a start position, and a goal position. Each threat is labeled with an integer, and the circle defined by each threat's MER is shown. All of the threats in this scenario are known, which is denoted by plotting the threat circles in red. The UAV is initially at the start position facing right with a velocity of 1 meter per second. The deliberative planner initializes the trajectory plan by running the GA with twice as many generations as normal. This plan is shown in magenta in the top subfigure of Figure 4.9. The planner then predicts where the vehicle will be after 2.5 seconds and runs the GA to find a trajectory from that position. After 2.5 seconds, the planner returns the GA plan, which is the second magenta line in the second subfigure. The planner continues these predict-plan-update iterations, shown as black circles along the trajectory in the bottom subfigure, but the GA does not improve the plan beyond the second iteration because the second plan was a very good one. The UAV follows the line of the second plan to the goal.

To continue the example, Figure 4.10 shows the same scenario with two threats added. These threats are initially unknown to the UAV, a fact denoted by plotting the threats in cyan. For this scenario's unknown figures, the detection radius equals the MER, but if it did not, the detection radius would be dotted cyan, and the MER would be solid cyan.

The vehicle planner generated the initial plan, shown in magenta in the top subfigure, without knowledge of the two cyan threats. When the vehicle crosses threat 7's MER, the threat becomes known, and the UAV generates a reactive plan. The highest fitness option is to fire on the threat, so it executes the option and is successful in disabling the threat. Disabled threats are shown in yellow. The reason that firing

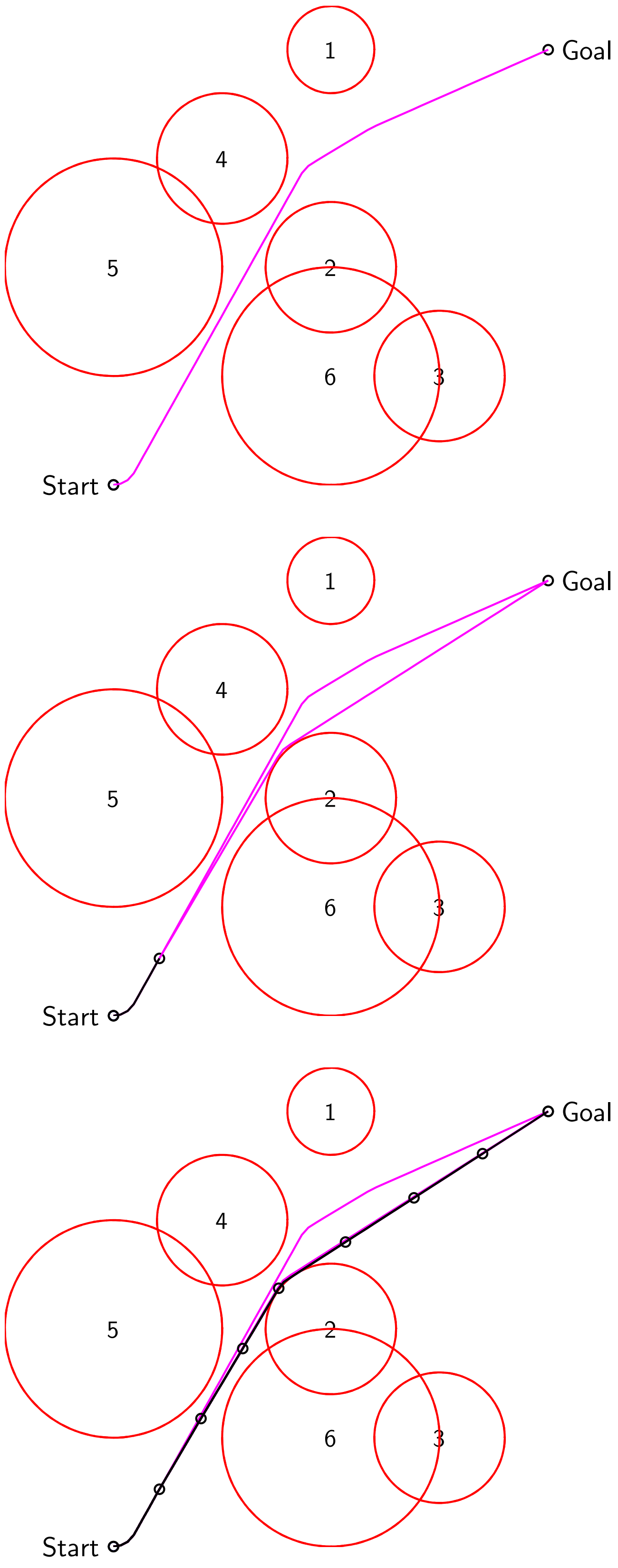


Figure 4.9: A field of threats and the combined planner.

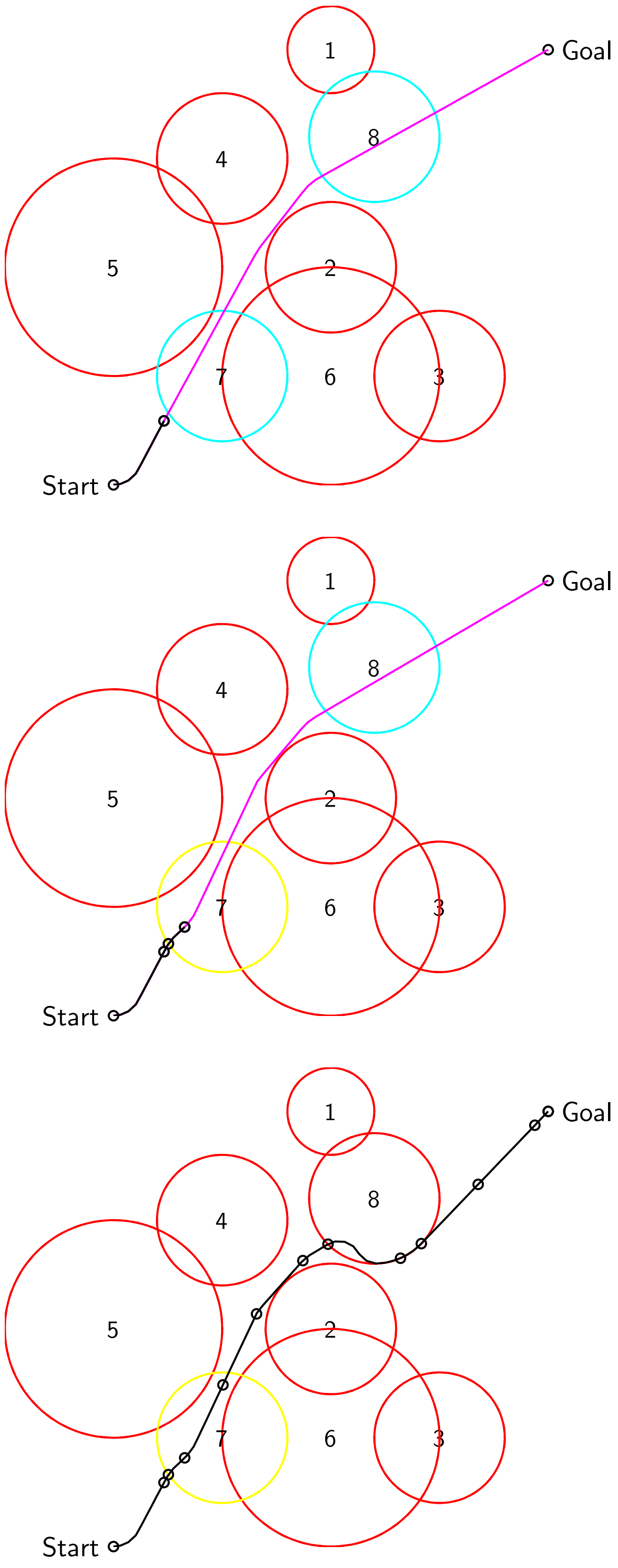


Figure 4.10: A field with unknown threats and the combined planner.

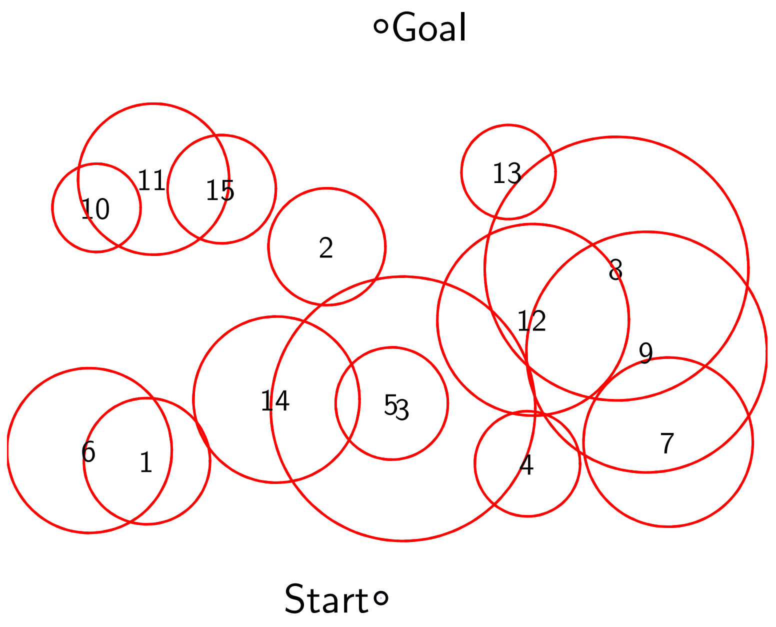


Figure 4.11: First scenario: crossing a field of threats.

on the threat might have higher fitness than circling the threat on the side is that threats 5 and 6 border threat 1 on either side so that trajectories that skirt threat 1's circle would accumulate exposure from these two threats. The UAV continues with its predict-plan-update iterations until it detects threat 8, changing the threat circle color to red. The reactive option with the highest fitness is to evade to the near side, which it executes. The UAV completes the scenario by going to the goal position.

4.6.2 Crossing a Field of Threats

The first scenario involves crossing a field of randomly placed threats of random radii as shown in Figure 4.11. This scenario is meant to generally test the planner with enough threats to create many local optima. The vehicle starts facing the goal position, which lies 30 meters away. Fifteen threats occupy a rectangle that is 40 meters wide and 16 meters long, centered between the start and goal positions, and their radii vary between 2 and 7 meters.

Figure 4.12 shows the results of 100 runs with different random numbers seeding the GA. Greyscale lines show the trajectories of the UAV, where a black line corresponds to a trajectory with the highest fitness out of the 100 runs, and a 10% grey line corresponds to a trajectory with the lowest fitness. Magenta asterisks denote countermeasures, magenta circles denote positions from which the UAV hit a threat, and cyan circles denote positions from which the UAV missed a threat.

For these 100 runs, the deliberative planner generated paths that largely went between threats, never double back, go through areas of low exposure accumulation, and often use weapons and countermeasures where they would be most effective. Because GA is a stochastic search algorithm, and the initial population was random,

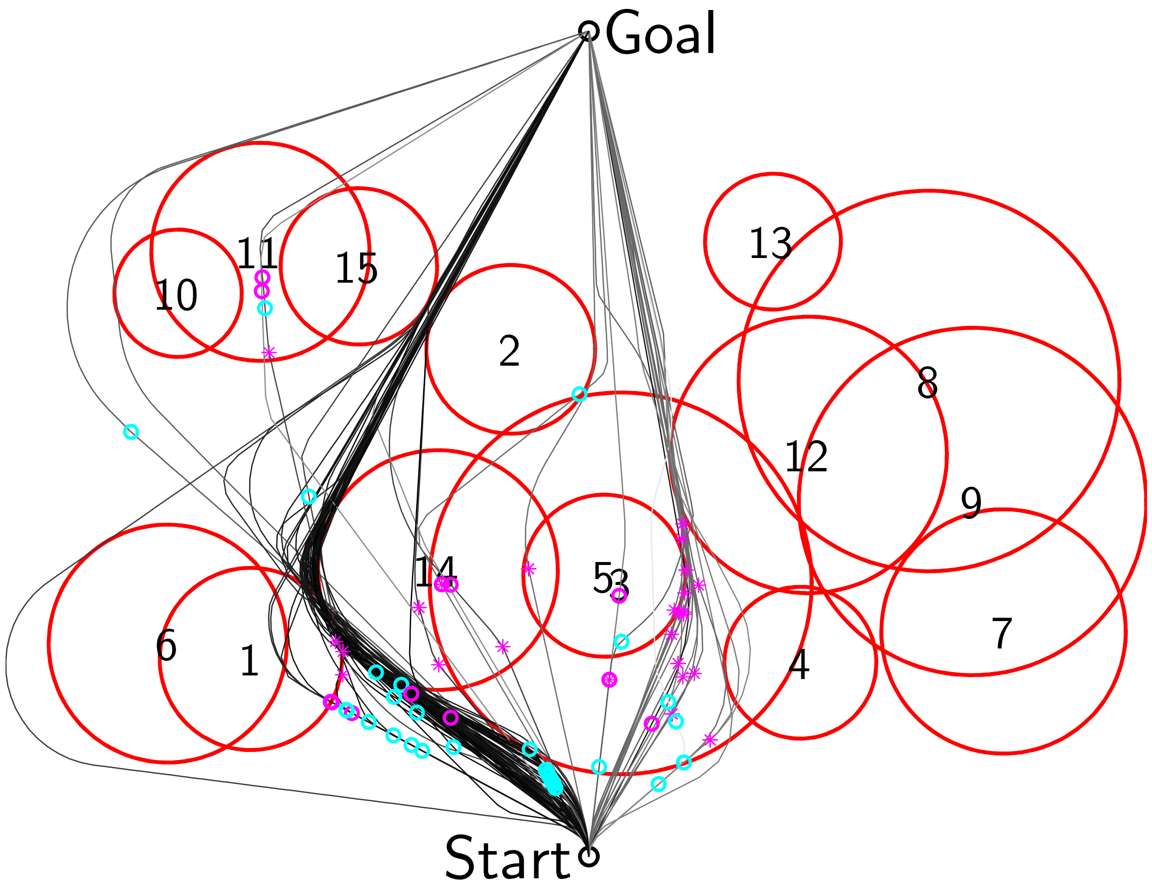


Figure 4.12: Trajectories for field-crossing scenario with threats known.

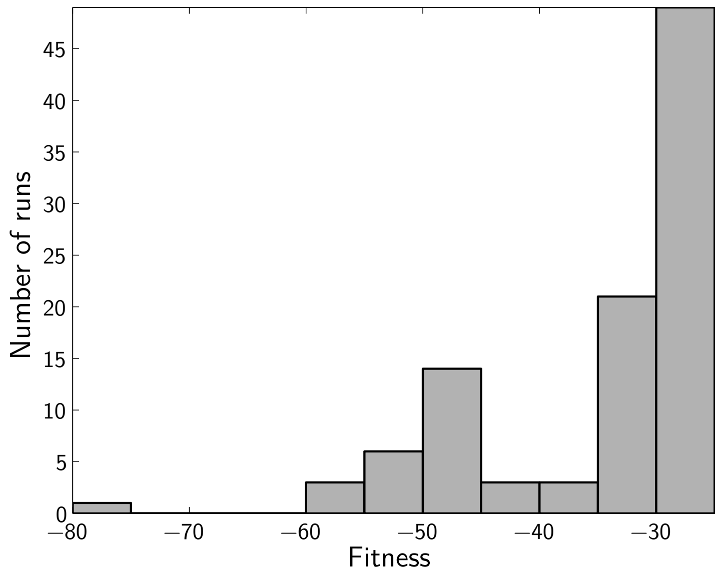


Figure 4.13: Fitness histogram for field-crossing scenario with threats known.

there is no guarantee that the GA would find a solution with these characteristics.

Most trajectories cluster around two paths: skirting threat 14 on the left and skirting threat 5 on the right. Each of these clusters has variations due to the random nature of the GA search, the lack of time for convergence to a uniform local optimum, and the use of different primitives in constructing the trajectories. Given more time to converge, the GA algorithm would likely choose the path around 14, but because of the stochastic nature of GAs, it could take many generations to do so.

Though not as good as the path around threat 14, the path around the right of threat 5 is a local optimum, since making slight alterations to the primitive parameters would carry the trajectory into the regions of threats 5 and 12. In these cases, the GA could not find a better solution, and the GA population likely converged to the local optimum at the right of threat 5, preventing exploration of other regions of the problem space.

The GA runs for only 60 generations to generate the initial plan, and this initial plan determines what path the vehicle takes for the first 2.5 seconds. Because the deliberative planner attempts to continuously improve the plan, it is possible that the UAV's plan could begin in one local minimum and then change its plan to a better local minimum during flight. For example, one path in Figure 4.12 initially plans to go around threat 6 to the left, but it then changes its plan to go between threats 1 and 14. However, no path changed from skirting threat 5 on the right to the better path between 1 and 14 because after 2.5 seconds, the UAV was well inside the region of threat 3, and turning to skirt threat 14 would have been worse than continuing to skirt threat 5. Thus, in many cases the initial plan generation largely determines the overall path that the UAV takes.

Figure 4.13 shows a histogram of the fitness of the 100 trajectories. There are two main clusters in the histogram, one on the right between -40 and -25 and one in the middle between -60 and -40. In general, the cluster on the right corresponds to the paths between threats 1 and 14, and the cluster in the middle corresponds to the paths to the right of threat 5.

Two plans from Figure 4.12 appear in Figure 4.14. The path in thick black achieved the highest fitness out of the 100 runs. The path in thin black achieved the lowest fitness. The dotted line shows the shortest path that does not enter any threat regions. This third path is a manually generated solution similar to the circular threat navigation algorithm of Asseo [5]. Asseo's algorithm has no ability to handle discrete events. The three paths have fitnesses of -25.4, -76.5, and -36.1, respectively. The best-performing run involved disabling threat 14 from outside its threat region and then taking the shortest path intersecting no threats. The worst-performing run missed several shots, incurring the cost of three weapons without lessening exposure. Most plans outperformed the dotted plan, though if threats 1 and 14 had a gap between them, the algorithm in [5] would have outperformed most of the paths generated by the planner.

Figure 4.15 shows the fitness over time for the deliberative planner's GA population throughout the entire run for the runs with the highest and lowest fitnesses. The time axis begins at -5 seconds to show the fitness during the initial plan generation.

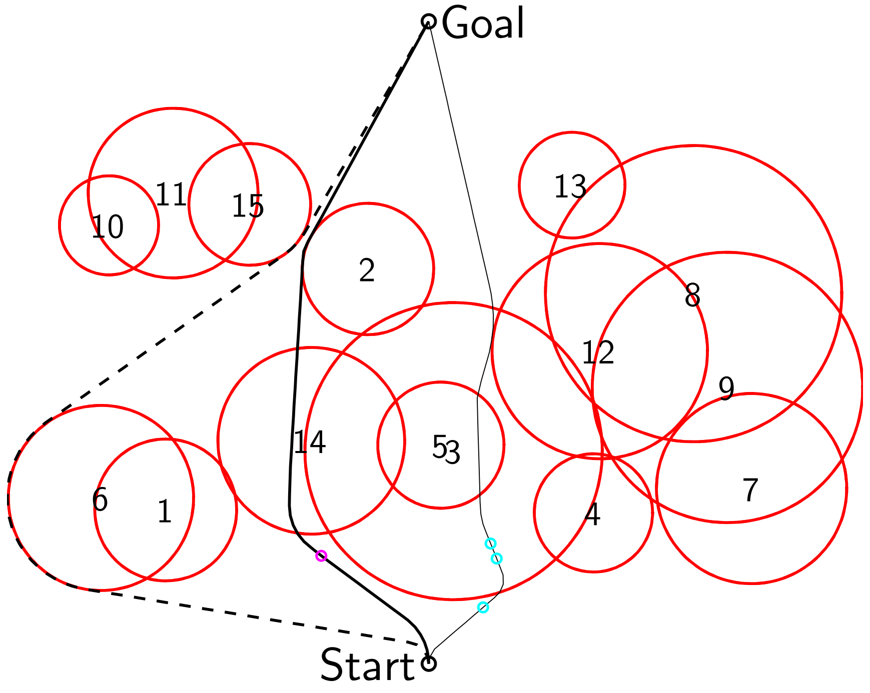
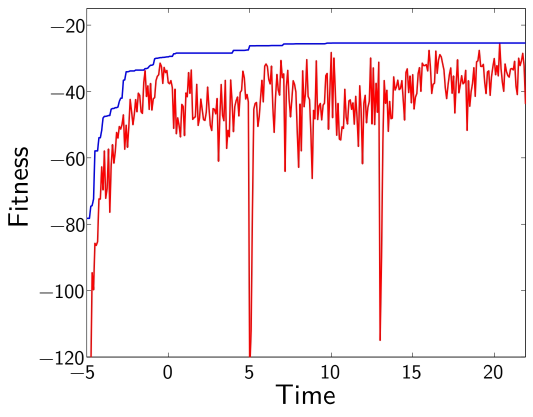


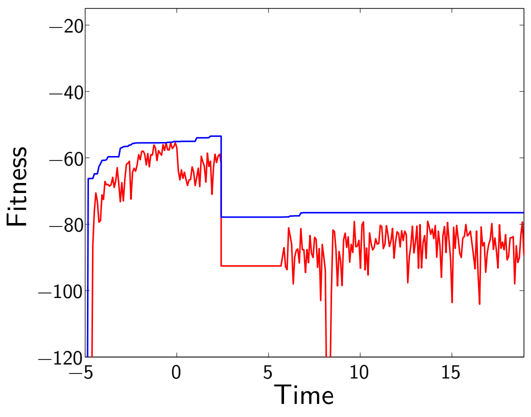
Figure 4.14: Selected trajectories for field-crossing scenario with threats known.

The UAV begins its flight at time zero. The blue line denotes the best expected fitness up to that point in time, and the red line denotes the average fitness of each generation. Because missing a shot triggers a reactive event, and because the simulation does not run GA when a reactive event would interrupt planning due to its simulated real-time implementation, the plot for the lowest-fitness run contains gaps in the period before and during the three missed shots.

The highest-fitness run rapidly increased its fitness during its initial 5 second planning stage and then gradually increased its fitness over the course of several planning iterations, each of which lasted 2.5 seconds. During a run, dips in the GA population's average fitness occur with certain changes in the environment or the plan. For example, these occur where the UAV disables a threat or completes a primitive. When the UAV disables a threat, the mapping of the primitive parameter denoting threat labels changes according to Figure 4.7, but in this implementation, the planner only repairs the UAV's plan, not every plan in the GA population. The threat labels change, and primitives shift to other threats in ways that might decrease fitness. Similarly, when the UAV completes a primitive, the planner removes it from the plan, but no change is made to the GA population, causing a possible change in the population's average fitness. This phenomenon might occur because the plans in the GA population contain primitives that the vehicle has already completed that would cause the vehicle to backtrack unnecessarily. Also, because the vehicle plans from its position after one iteration, members of the GA population that were created for the last iteration might cause backtracking. Dips in average fitness for the highest-fitness run occurred at $t \approx 5$ seconds, when the UAV disabled threat 14, and at $t \approx 13$



(a)



(b)

Figure 4.15: Fitness over time for (a) the run with highest fitness and (b) the run with lowest fitness. The blue line denotes the best expected fitness up to that point in time, and the red line denotes the average fitness of each generation. Gaps in deliberative planning appear as flat lines.

seconds, when the UAV reached threat 2 and exited a primitive.

The lowest-fitness run fired a weapon and missed three times. When planning, expected fitness is a weighted average between the fitness of the successful and unsuccessful outcomes. When the UAV missed, the outcome became known, and the fitness decreased accordingly.

Unknown Threats

Figure 4.16 shows results for the same scenario except with three threats made unknown and assigned random detection radii, shown with cyan dotted lines. In many runs, the UAV takes the right-hand path because it does not know about threat 12, which leads to lower fitness and a greater spread in the histogram of Figure 4.17 toward lower fitness. When reacting to threat 12, the UAV either deployed countermeasures or fired a weapon. Evasion would have resulted in higher exposure due to neighboring threats.

The planner uses primitives creatively, such as by calling for a shot on threat 6 just to maneuver the vehicle around threat 14 and between threats 2 and 15. The UAV often takes these shots from distances that make disabling the threat improbable. This shows that the UAV's goal is not to disable the threat, but simply to maneuver using the *fireonthreat* primitive.

Though it might appear as though a bug caused the figure-eight near threat 10, this is not the case. The planner first decided to fire on threat 10 in order to have the UAV make a left turn and pass between threats 2 and 15. While running its GA, the planner found a better plan that went around the outside of threats 10 and 11.

Figure 4.18 shows fitness over time for the relatively low-fitness run shown in the subplot on the right. The fitness increased at $t \approx 5$ seconds when the UAV disabled

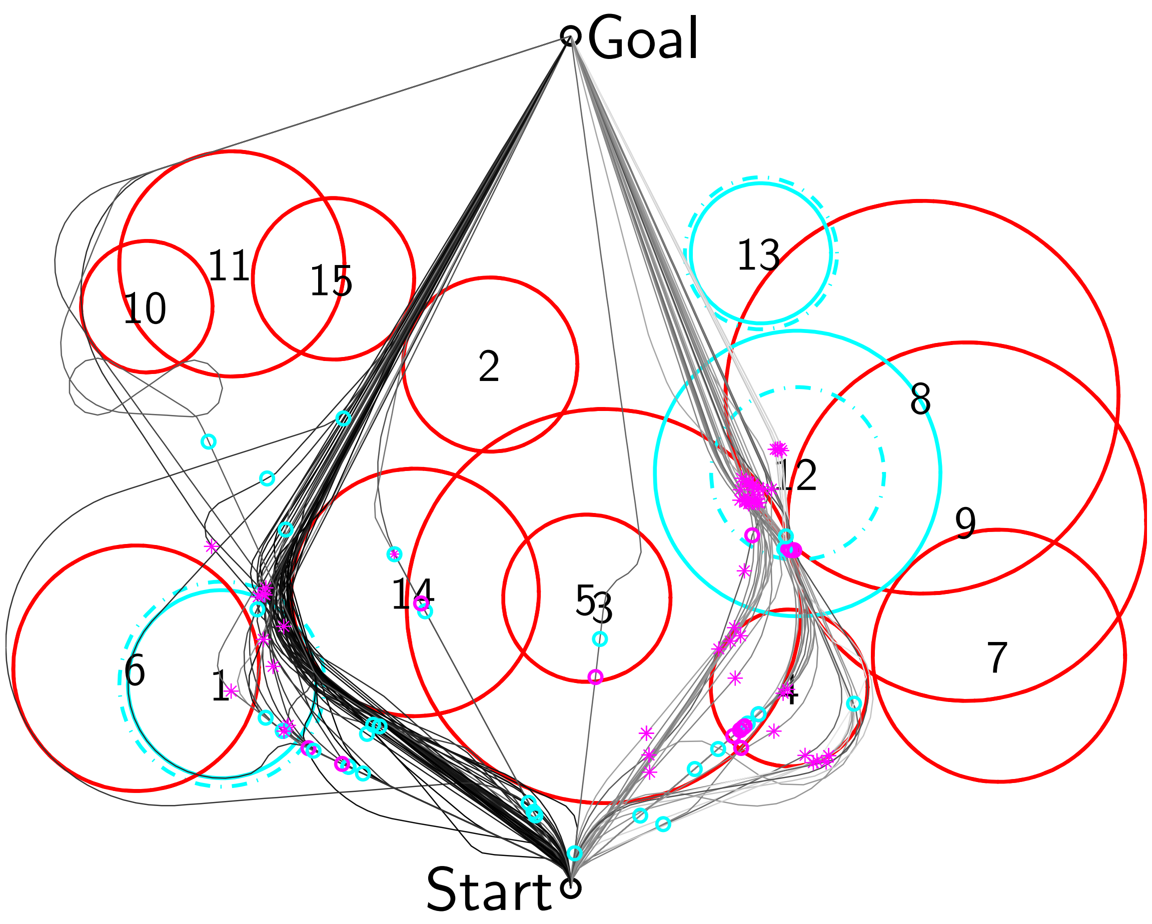


Figure 4.16: Trajectories for field-crossing scenario with three unknown threats.

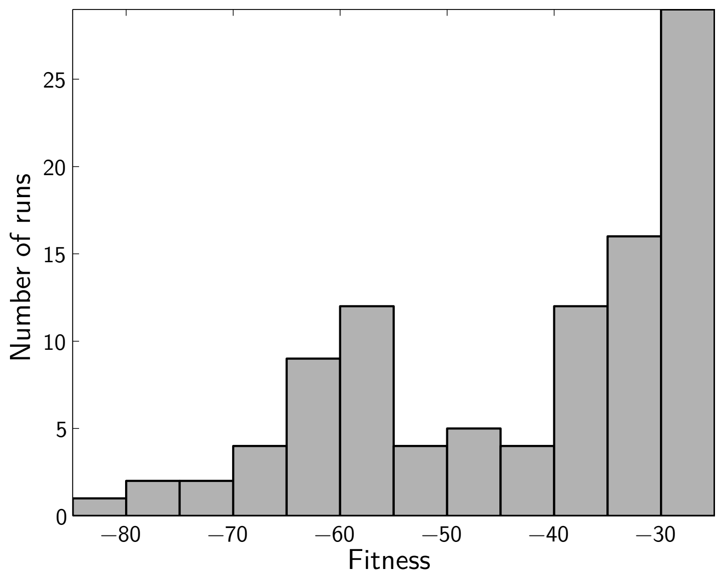


Figure 4.17: Fitness histogram for field-crossing scenario with some threats unknown.

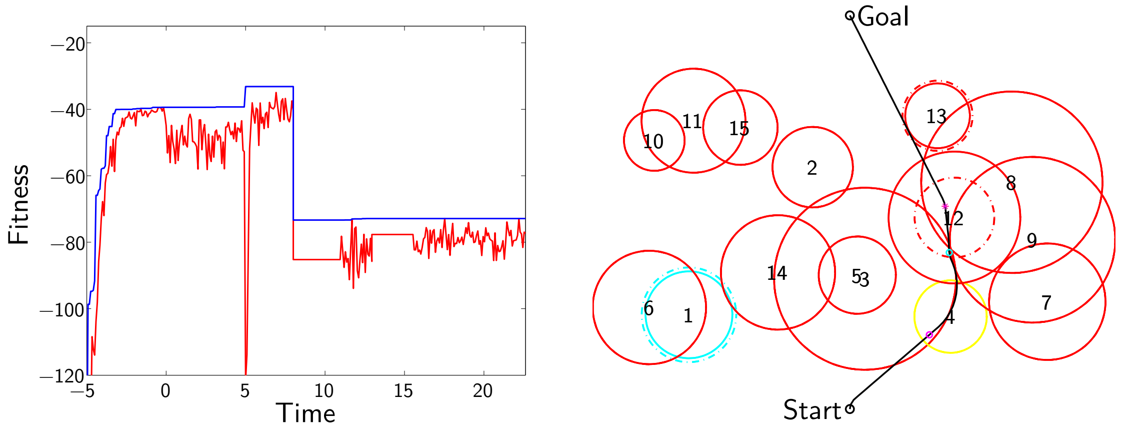


Figure 4.18: (a) Fitness over time for the field-crossing run shown in (b). In (a), the blue line denotes the best expected fitness up to that point in time, and the red line denotes the average fitness of each generation. Gaps in deliberative planning appear as flat lines.

threat 4, and it dropped at $t \approx 8$ seconds when the UAV detected threat 12. Gaps began around $t = 5$ seconds and $t = 15$ seconds due to reactions to threats 12 and 13, respectively. The interaction with threat 13 produced no change in fitness because the UAV's trajectory intersected the threat's detection radius but did not enter the threat region.

4.6.3 Attack Mission

The second threats scenario for Chapter 4 appears in Figure 4.19. In this scenario, threat 7 is a target, and the UAV receives a fitness bonus of 100 for disabling it. The shaded pentagon is a no-fly zone, and the UAV receives a penalty for entering the pentagon according to Section 4.1.5. The UAV is supplied with six weapons instead of three.

The proximity of the no-fly zone to the target decreases the probability that the planner would discover the optimal plan, which involves firing on threat 7 at a close distance while avoiding the no-fly zone. To help the planner find that optimum, the initial population included a single plan (out of 80 total) containing a *fireonthreat* primitive with *null* primitives on either side, then three more *null* primitives. The GA could then use this plan to explore the region of the optimal plan. Section 5.6 presents more results related to seeding the GA.

Figure 4.20 shows the results of 100 runs with different random seeds to seed the deliberative planner's GA. Though many runs did fire on threat 7, many did not and instead opted to circle the field of threats entirely or pass through the local optimum to the right of threat 5. The reason for this behavior is the quality of the initial seed plan. The initial seed plan took the UAV through the no-fly zone and achieved a low fitness. The GA then assigned that chromosome a probability of reproducing that was proportional to its fitness rank. In some cases, the seed plan would have been

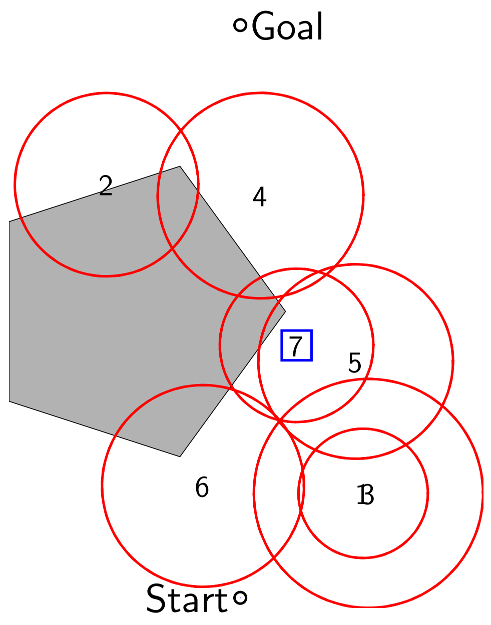


Figure 4.19: Setup for one-target attack scenario. The UAV receives a bonus for disabling threat 7, which is a target, shown here in a blue square. The UAV is heavily penalized for entering the grey no-fly zone. *Note:* threats 1 and 3 overlap so that their labels look like 13.

discarded in the first several generations. Sometimes the GA mutated the seed to find a plan involving firing on threat 7 that did not enter the no-fly zone, sometimes it settled the population into the local optimum of flying around all threats, and in a few cases, the UAV entered just the corner of the no-fly zone, and the penalty of entering the no-fly zone offset the fitness gains from disabling targets.

One reason that the planner had difficulty finding plans for firing on the target is that the *fireonthreat* primitive involves turning toward its target threat at the tightest radius. In the seed plan, after firing on threat 7, the UAV would turn sharply to the left and cut across the no-fly zone to go to the goal. If the *fireonthreat* primitive included a variable turning radius parameter, the UAV might have found optimal plans for a higher percentage of the 100 runs.

In at least two cases, the UAV's trajectory intersected a corner of the no-fly zone, but the equally spaced trajectory points used to calculate the no-fly-zone fitness penalty did not actually enter the polygon, and the UAV received no penalty. This could be repaired by a closer trajectory point spacing or a different method of calculating the fitness penalty.

Two-Target Attack Scenario

Another case of the attack scenario involves unknown threats and two targets, threats 1 and 2, shown in Figure 4.21. Both targets lie very close to corners of the polygon, requiring tight maneuvering to achieve both an effective shot and a feasible trajectory.

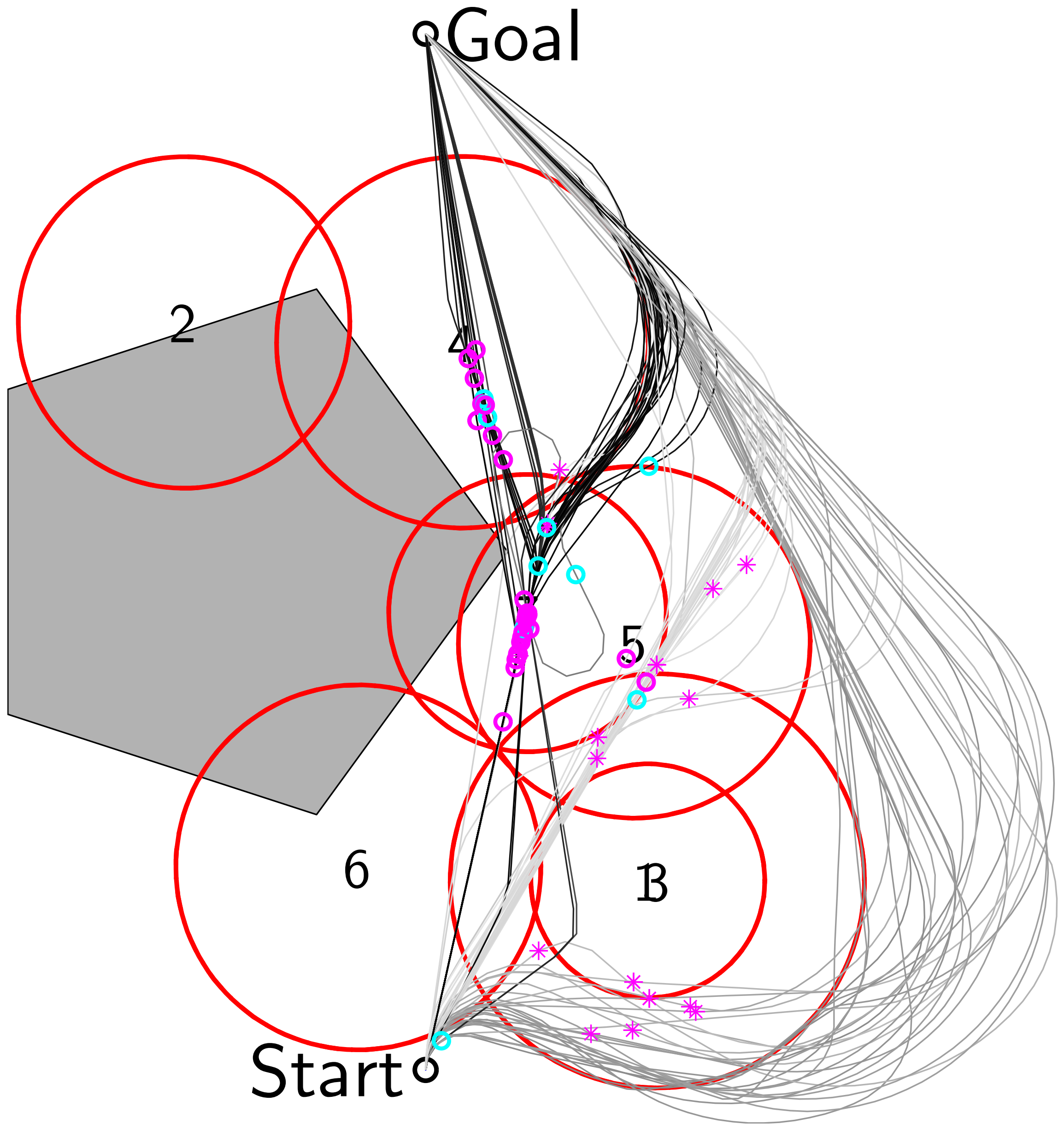


Figure 4.20: Trajectories for one-target attack scenario.

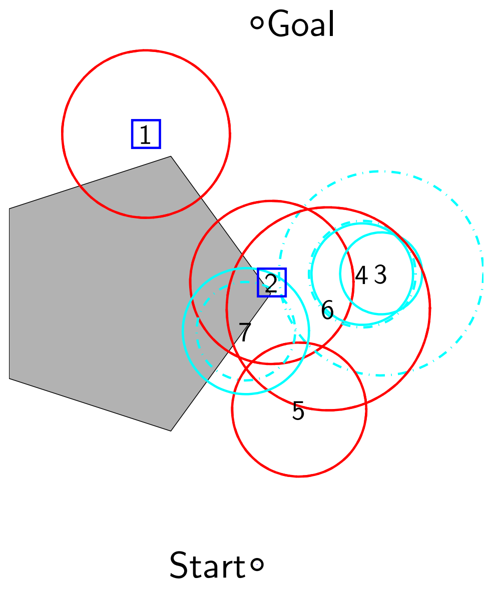


Figure 4.21: Setup for two-target attack scenario. The UAV receives bonuses for disabling threats 1 and 2, which are targets, shown here in blue squares.

Figure 4.22 presents the results of 100 runs.

Two interesting features mark this simulation. First, many but not all of the trajectories enter the no-fly zone on one or two corners. Though trajectories exist for firing on the threats successfully without entering the polygon, not all runs found one. In addition, the target bonus begins to offset the penalty so that the UAV prefers to clip the polygon to get a good shot. Second, the UAV often turns around on missed shots for a second approach. The loops near threat 1 in the upper left make this apparent. The deliberative planner found the target again through GA and gained another opportunity for the target bonus.

4.6.4 Urban Canyon

The final scenario of Chapter 4 is the urban canyon scenario, shown in Figure 4.23. This scenario stresses the algorithm because it gives the UAV no easy path and demonstrates the planner's flexibility. The UAV cannot go around the field of threats but must find ways to deal with the threats it encounters. Because there is no open path to the goal, Asseo's method would not even find a path within the canyon [5]. The two walls are implemented as no-fly zones, and the UAV is prevented from entering the canyon walls through the fitness penalty described in Section 4.1.5.

Figure 4.24 shows the results from 100 runs through the canyon. The best-performing runs involve flying between threats 2 and 7, disabling threat 9 or threat 10 or flying between them, and skirting threat 11. Unlike the attack scenario, trajectories remain outside of the no-fly zones. This scenario contains no tight constraints like the attack scenario, where targets lay near the corners of the no-fly zone and forced several trajectories inside. The UAV fired more times than in the scenario of

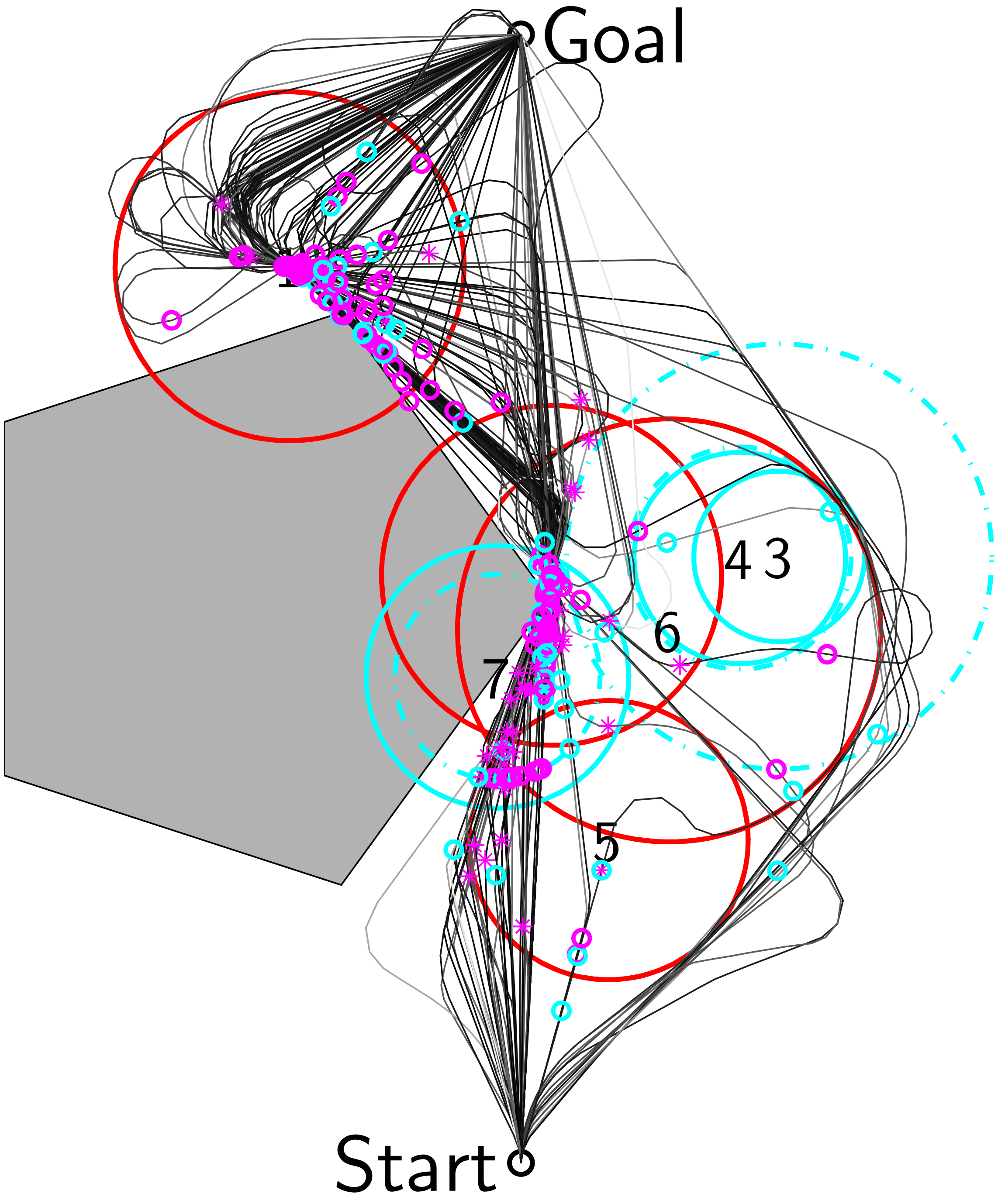


Figure 4.22: Trajectories for two-target attack scenario.

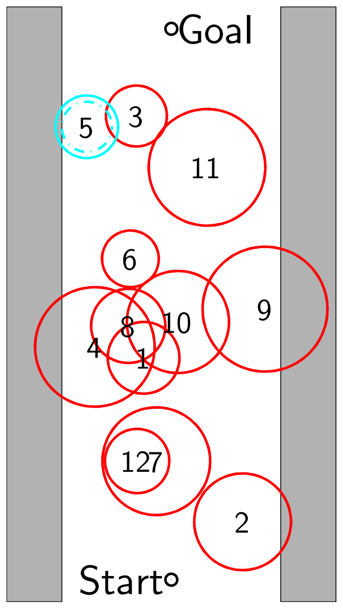


Figure 4.23: Setup for urban canyon scenario.

Figure 4.12 (78 vs. 47), it had a higher percentage of successful shots (56% vs. 28%), and it used more countermeasures (67 vs. 33). In the field-crossing scenario, the UAV could take shots from a distance then go around the threats if they missed. The UAV needed shots and countermeasures to be effective in the urban canyon scenario because there was no path around the threats. Though shots cost the same, the UAV had six weapons in this scenario and only three in the field-crossing scenario.

4.7 Discussion

The previous section presented results from the application of the combined reactive/deliberative planner to three different scenarios of the threats problem. In each case, the planner improved a population of random plans into plans that performed well, using the available tactical primitives to their fullest advantage and generating many plans that rapidly reached the goal without incurring large exposure. The fact that the method worked on three different scenarios shows the versatility of the planner, a fact that the next chapter will continue to explore.

It is remarkable that the planner performs well with both continuous dynamics and discrete events without any tailoring beyond the design of tactical primitives. One notable example is the thick black trajectory of Figure 4.14, where the UAV disables a threat and takes advantage of the new shortest exposure-free path to the goal. In many cases the deliberative planner places countermeasures near positions along the trajectory where the exposure is greatest.

Though the GA generated many plans with high fitness, many others achieved much lower fitness. The discussion below and in Chapter 5 will present suggestions for overcoming GA's inconsistency and lack of guaranteed solution quality. Also,

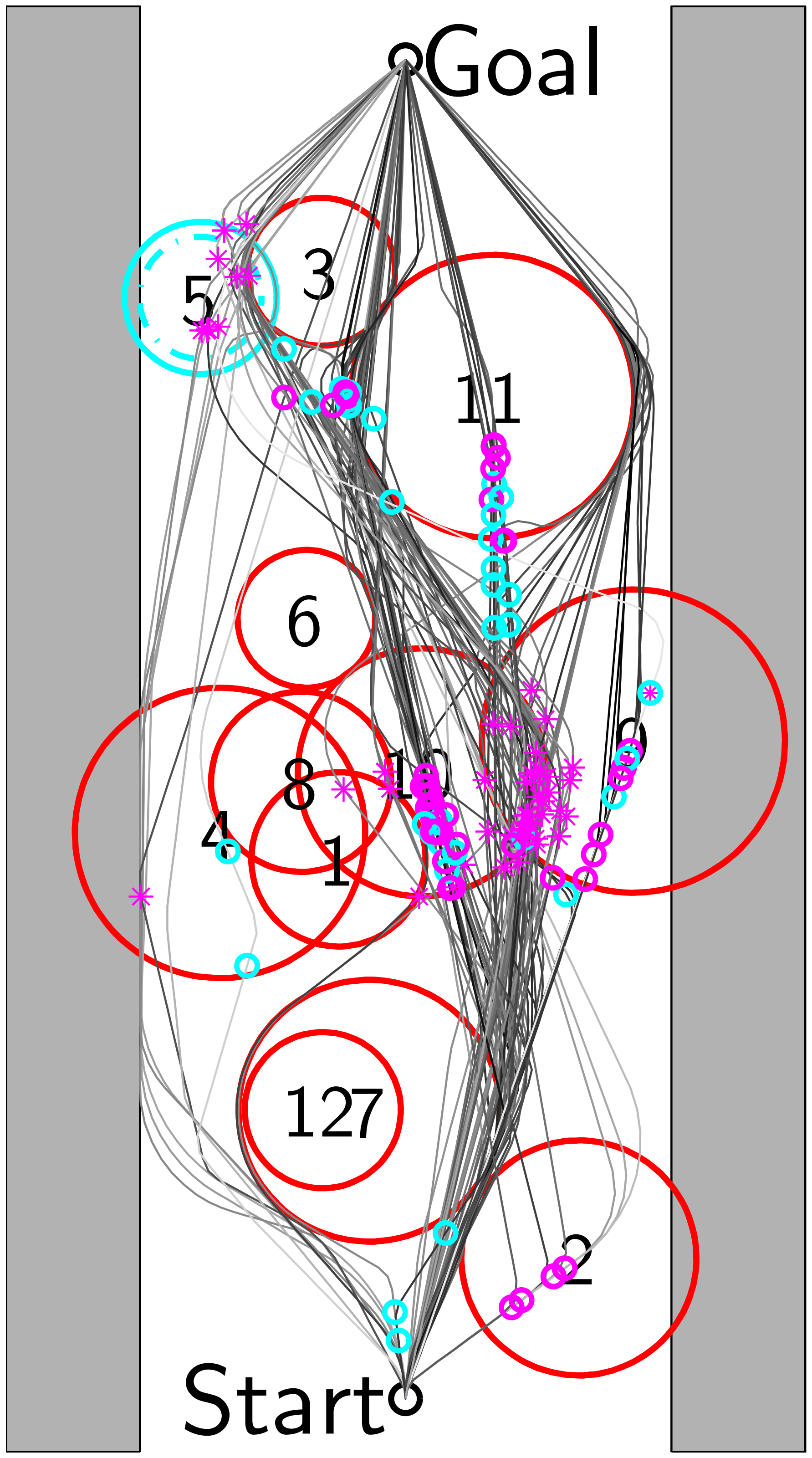


Figure 4.24: Trajectories for urban canyon scenario.

alternative methods that guarantee optimal solutions, if they exist, would probably have runtimes that scale exponentially with a particular dimension of the problem space, due to numerical calculations of gradients, etc. On the other hand, GA's runtime scales only with the calculation time of the fitness function, the population size, and the number of generations, with these last two elements chosen by a designer.

The field-crossing scenario of Figure 4.11 had a *null* primitive and three types of primitives that were linked to threats. With fifteen threats, there were forty-six different combinations of those two discrete parameters in one primitive. With six primitives, $46^6 \approx 10$ billion plans with different discrete parameters exist for one GA run. This count does not include continuous parameters or iterations in the deliberative planner. Even with this many possible solutions, the planner was able to find plans that make intuitive sense.

With many more than 10 billion possible plans, the planner could benefit from more directed search, such as an improved initial population or improved mutation operators. The attack scenario demonstrated the effectiveness of seeding the population with an initial plan, but the GA often rejected the initial plan because it was too constrained. By including several versions of the seed plan with variations in primitive parameters, the GA would be able to explore that region of the problem space near the seed plan and find a useful modification. Asseo's fast circular threats navigation algorithm could be modified to generate multiple useful initial plans during each iteration of the deliberative planner [5]. For example, seeding the GA with several plans that thread multiple paths from start to goal would give the GA access to several local optima in the search space and cause the GA to converge to a higher fitness faster.

These scenarios use the supplied tactical primitives effectively, showing that the primitives encapsulated relevant behavior and were generally conducive to GA search. In some cases the GA chose a primitive that made less sense, such as using *fireonthreat* with an ineffective shot for maneuvering instead of *gotothreat*. In this case the GA population likely converged to a local optimum. Because the UAV expends a weapon and incurs a weapon's cost, it could be advantageous to include a primitive that mimics the motion during *fireonthreat* without actually firing on the threat, though this would increase the size of the problem space and make the GA search more difficult.

In the attack scenario, the no-fly zone's proximity to the targets constrained the physical space, and the *fireonthreat*'s turning radius constrained allowable motion during firing. These constraints caused difficulties in finding parameters for *fireonthreat* that allowed the vehicle to accomplish its mission. This factor should be considered during primitive design. For example, a different maneuver could be paired with firing a weapon, or more parameters to create variations could be added.

Mutation operators that are tailored to the threat layout could help the GA search the problem space more effectively. Small changes in the primitive parameter denoting which threat to approach can create large changes in the UAV's trajectory because threats that have neighboring integer labels might be separated in space, generating chromosomes with low fitness that could immediately be discarded. To make small mutations more effective, nearby threats could be labeled with adjacent integers, or a graph could connect nearby threats, and a custom mutation operator could move primitives between neighboring threats.

Initial GA solutions appear to determine the performance of an entire run to a large extent. If the GA gets trapped at a local optimum, it can take many generations to find a better local optimum. The GA population converges, and productive mutations become rare. Because the number of local optima is high, local optima are often separated in the problem space, and very few local optima produce “good” plans, modifications to the baseline GA algorithm might be necessary to avoid convergence. Section A.1.6 gives several methods of avoiding convergence, which are replacement, similarity penalty, and annealing.

In conclusion, this chapter has presented the threats problem setup and a basic application of the combined reactive/deliberative planner. The next chapter will discuss specific details about the planner, such as the effect of computation power, application to different vehicles, the application of nonlinear programming, and comparing different primitive libraries. These give a clear picture of several design elements' effects on the planner's performance.

Chapter 5

Algorithm Performance

The previous chapter defined the threats problem and established the planner's basic performance. This chapter explores further aspects of the planner including the relationship of plan fitness to computation time, dimensionality of the environmental model, comparison with a different optimization algorithm, comparison between different primitive libraries, and exploring the generality of the algorithm using different vehicle models. This knowledge can help guide algorithm implementation, further algorithm design, and future planning research.

5.1 Computation and Performance

With genetic algorithms (GA), more computation generally yields better performance. The GA attempts to continuously improve the solution, and running for more generations allows the GA to search more of the solution space [20]. This is a consequence of the stochastic nature of GAs—given infinite time, the probability of exploring the entire problem space is one. However, the progress of GA usually slows due to convergence of the population to some local optimum. Running GA for different numbers of generations shows how often the algorithm converges to a near-optimal solution, how often it becomes trapped, whether the GA continues to improve with more computation, and therefore how well it works on a particular problem.

To study computation and performance, the GA searched for plans for the field-crossing problem in Figure 4.11 with varying numbers of generations per deliberative planning iteration. From Chapter 3, a deliberative planning iteration lasts for a fixed amount of time, on the order of seconds, and involves an entire GA optimization run to update the plan, whereas a GA generation is internal to the GA algorithm and lasts on the order of tenths of seconds. The scenario ran for 100 runs at each number of generations, with each run's random number generator seeded by its run number, from 1 to 100. The legend of Figure 5.1 shows eight different numbers of generations, for a total of 800 runs. These numbers follow a logarithmic scale because the GA slows its improvement over time, and they range from 5 to 300 generations to capture GA behavior from "barely improved over random initialization" to "most solutions lie in the region of the global optimum."

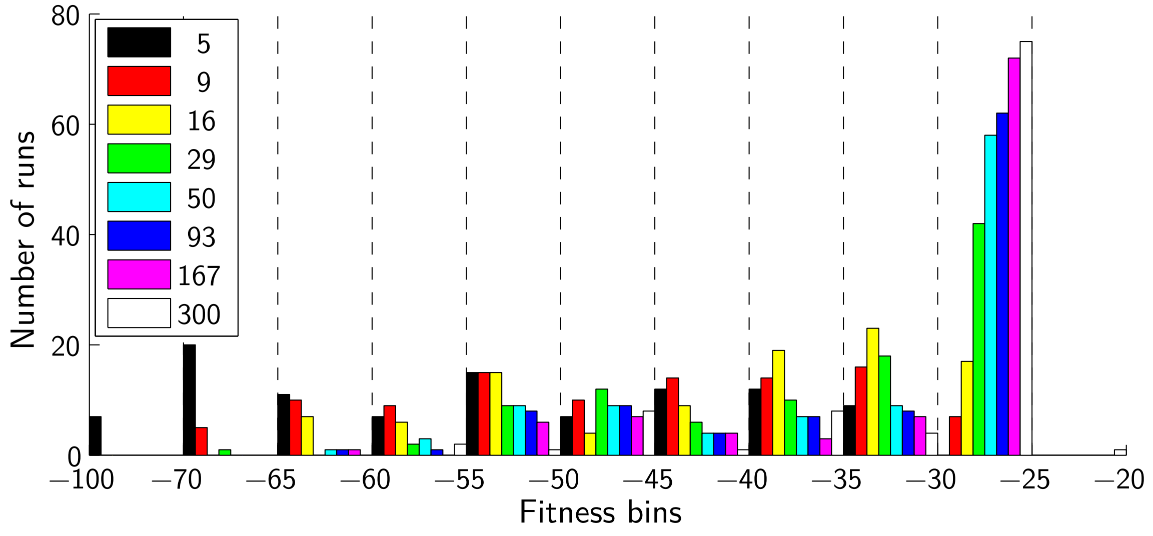


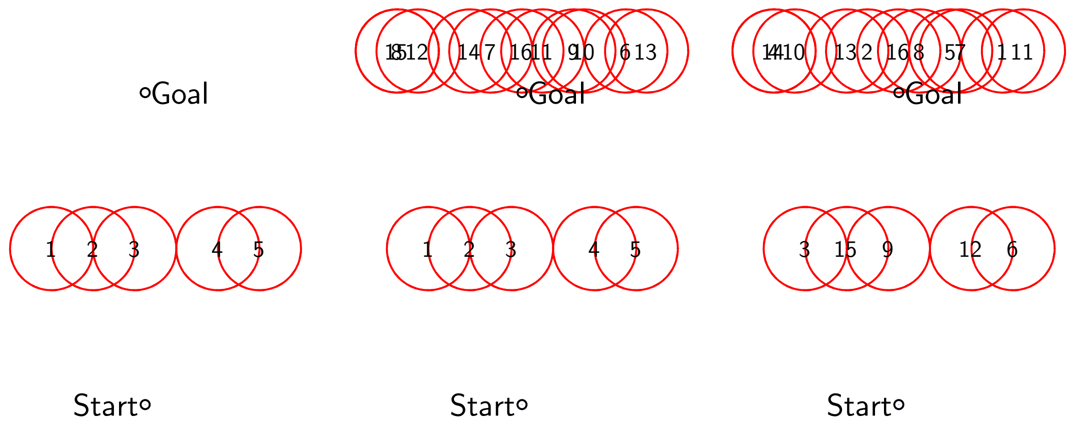
Figure 5.1: Fitness histograms for different amounts of computation. Each color shows a histogram of 100 runs of the threat problem with a different number of GA generations per deliberative planning iteration. The histogram bins are shown by the dotted lines. The left-most bin captures all runs with fitness lower than -70.

Figure 5.1 shows eight histograms with similar bins side-by-side. Each histogram bin spans five fitness points, and the dotted lines show the bin edges. The most notable feature is the nature of the -30 to -25 bin. It contains no 5-generation runs and 75 of the 300-generation runs, and in between, the number of runs with fitness above -30 steadily increases with more generations. The number of runs in this bin are, from 5 generations to 300 generations in order, 0, 7, 17, 42, 58, 62, 72, and 75. The increase between 5, 9, and 16 generations is small at gains of 7 and 10 runs, the increase between 16, 29, and 50 generations is larger at 25 and 16 runs, and the gains between 50, 93, 167, and 300 generations tapers off at 4, 10, and 3 runs. The fitness improvement rate is greatest around 29 generations and slows even with respect to the logarithmic arrangment of the generation numbers after 50 generations. In order to use GA with tactical primitives on a particular system, a designer should find this point of diminishing returns and ensure that the system has the appropriate computation resources.

5.2 Effects of Dimensionality

The threats problem has many dimensions, including primitive type, threat label, and three or four continuous parameters per primitive for each primitive in a plan. GA is able to find solutions even as problem dimensionality increases, but finding solutions of desireable fitness in a larger space takes more generations on average. This section presents a study to demonstrate this effect.

Figure 5.2 contains three threat arrangements, (a) one with five threats and (b,c) two with sixteen threats. The optimal plan is the same for all three arrangments



(a) 5 threats. (b) 16 threats ordered. (c) 16 threats jumbled.

Figure 5.2: Dimensionality study setup.

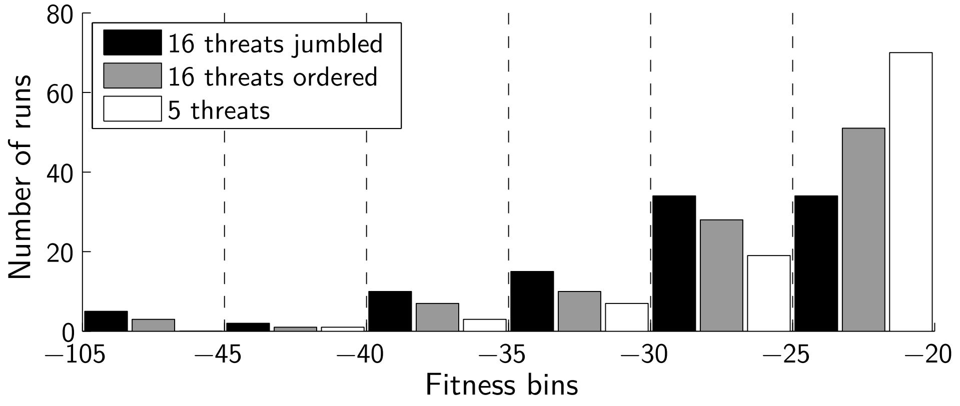


Figure 5.3: Histograms of fitness for the two setups. Dotted lines show bin edges. The left-most bin captures fitnesses between -105 and -45.

because the five foreground threats are the same: fly through the gap between threats 3 and 4 in (a) and (b), which is equivalent to between threats 9 and 12 in (c). In (a) and (b), adjacent threats have consecutive labels so that small mutations of the threat label parameter produce small changes in the resulting trajectory. In (a), the five threats fill the $(0, 1]$ interval of the threat label parameter, but in (b) and (c), they only occupy 5/16 of that interval. Behind the goal in (b) and (c) sit eleven threats to distract the GA. Additionally in (c), the eleven threats have labels between the labels of the five threats, separating the five threats in the chromosome.

To compare the GA performance between the three arrangments, 100 initial GA plans ran with a population size of 16 for 30 generations, compared to the full run with deliberative planner iterations, a population size of 80, and an initial run of 60 generations for the scenarios of Chapter 4. With so few threats in a simple arrangment, the GA would likely find good solutions for both arrangments if allowed to run just a few more generations.

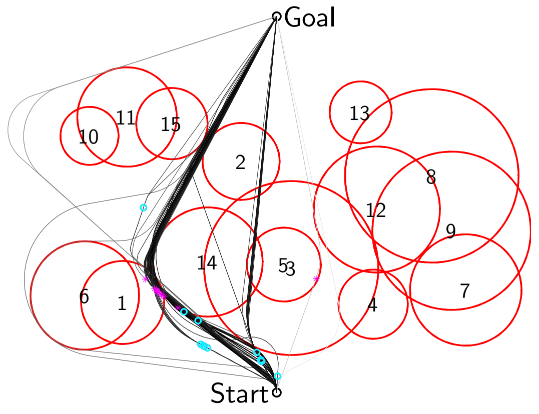
Figure 5.3 shows a fitness histogram for each run of 100 plan generations. Case (a) outperforms (b), and (b) outperforms (c). This result shows that GA performance depends on both the dimensionality of the problem space and the proximity of adjacent threats in the chromosome. Thus, reducing dimensionality and adding structure to bring phenotypic proximity into the chromosome improves GA performance. This result suggests that a preprocessor for sorting labels would be an effective algorithm. Many methods exist for reducing dimensionality, such as system approximation through some sort of model reduction, receding horizon control, and intelligent search. Model reduction takes many forms, but a simple method for this problem would be to encapsulate overlapping threats into one convex shape. Receding horizon control would simply ignore elements outside a certain range. An element of intelligent search could ignore much of the space and find notable gaps and isolated threats. All of these methods can lessen the cost of high dimensionality by focusing GA on certain parts of the problem space.

5.3 GA and Nonlinear Programming

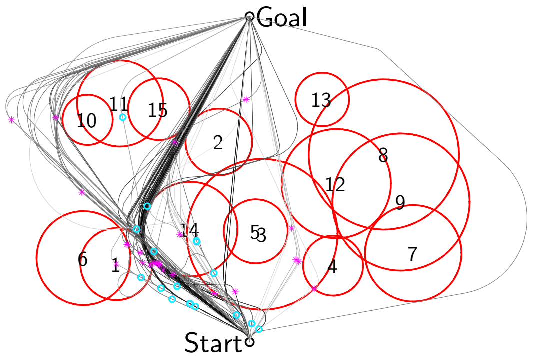
The threats problem is nonlinear and hybrid, and no current solution method is the clear choice for solving it. In an attempt to create a new algorithm for comparison with GA on the threats problem, GA was combined with nonlinear programming (NLP). MATLAB's NLP function `fmincon` works by solving a quadratic program at each iteration and then performing a line search for the optimum [9, 33]. NLP works on continuous parameters, though not well in the presence of discontinuities because the gradient becomes undefined, and it does not guarantee an optimal solution because it might become trapped in local optima. In this combined hybrid method, GA chooses and freezes the primitive type and the threat label parameter for each primitive, GA chooses initial continuous parameters, and then NLP optimizes the continuous parameters using the GA-chosen initial solution guesses.

The test scenario was the same as the one in Figure 4.11. As in the section above, only initial plans were considered, as opposed to full runs with multiple deliberative planner iterations. Comparing GA alone and GA-NLP at two different computation times allows for the study of short-term and long-term optimization behavior. In all cases, the plan consisted of only two primitives. In the short-term case, GA alone ran with a population size of 80 for 100 generations with computation lasting for 28 seconds on average, and GA-NLP ran with a population size of 8 for 2 generations for an average computation time of 34 seconds. Long-term runs used the same population size, but the number of generations increased to 700 and 10. In addition, `fmincon` by default exits when tolerances in the function and the solution shrink to 10^{-6}, but this number was increased to 0.01, sacrificing accuracy to decrease computation time. Figure 5.4 shows the trajectories from both runs.

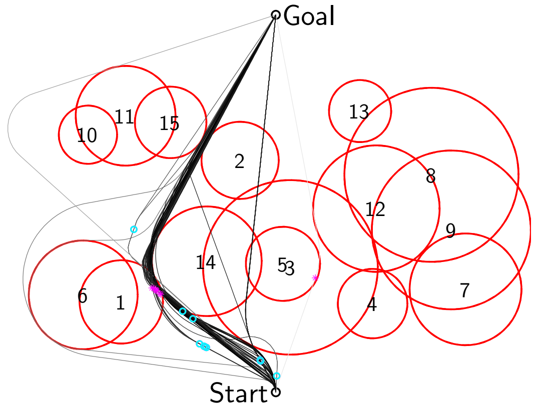
Table 5.1 shows statistics for the four 100-run simulations. For the short runs, GA alone outperformed GA with NLP. GA-NLP had only two generations to choose its discrete parameters, and even for individual subproblems involving a specific, frozen set of discrete parameters, NLP might have converged to a relatively low-fitness local



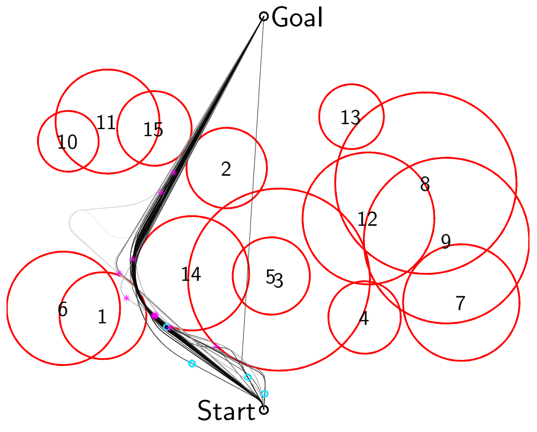
(a) GA-alone short run.



(b) GA-NLP short run.



(c) GA-alone long run.



(d) GA-NLP long run.

Figure 5.4: Results of GA alone (a,c) and GA mixed with nonlinear programming (b,d). Short and long refer to computation time. Greyscaling of trajectories to denote fitness is independent between the two figures.

	Mean computation time (sec)	Min fitness	Mean fitness	Max fitness
GA-alone short	28	-53.5	-28.8	-25.4
GA-NLP short	34	-57.5	-38.3	-25.3
GA-alone long	71	-49.3	-28.1	-25.3
GA-NLP long	203	-37.0	-26.7	-25.2

Table 5.1: GA-NLP comparison runtimes and fitnesses.

optimum. For the long runs, GA alone improved very little over the short run despite running for seven times longer. GA-NLP greatly improved its minimum and its average while running for just 10 generations with a population of 8.

Though the long GA runs contain seven times the number of generations as the short runs, their computation does not last seven times longer. The GA population likely converges to local optima whose fitness values are fast to compute because they correspond to short trajectories. Without a means of maintaining diversity in the population, progress with GA alone stagnates. The long GA-NLP run performs better than any other field-crossing scenario run, with the highest minimum, maximum, and mean fitnesses. An optimized gradient search method might deliver higher fitness than GA in realtime. However, the dimensionality of the problem matters more with NLP than it does with GA in terms of computation time, since GA's computation time depends only on the fitness function and the number of generations, while NLP's numerical gradient calculation does not scale as well. On problems with few continuous dimensions, GA-NLP could be a competitive solution method.

5.4 Primitive Types

Section 2.1 claimed that tactical primitives that are linked to environmental features might outperform primitives that are defined in the body frame. This scenario takes the setup in Figure 4.11 and tests this idea. The primitive library used in Chapter 4 can be said to operate in the "threats frame," since each primitive is linked to a particular known threat. In contrast, the body-frame primitives in this scenario include the *turn* and *straight* maneuvers. The *turn* maneuver is parametrized by radius, on the interval $(1, 11]$, and length, defined by an angle on the interval $(-\pi, \pi]$ radians, where positive and negative angle correspond to left and right turns, respectively. Angles less than 0.1 radians are lengthened to 0.1 radians to prevent *turn* from looking too much like *null*. The *straight* maneuver is parametrized by a length on the interval $(0, 10]$. Firing and countermeasure deployment have no associated maneuvers, though firing does include the same drift described in Section 4.2.2. Velocity parameters also remain the same as for threats-frame primitives, as described in Section 4.2.1. When the deliberative planner returns all *null* primitives, the vehicle executes *gotopoint* to go to the goal.

Figure 5.5 shows a comparison of the 100 runs for both primitive libraries, and Figure 5.6 shows fitness histograms. The threats-frame library results in tighter trajectories with higher fitness. The body-frame runs spread out over a large area that often intersects threat regions before going to the goal. In many cases, the best plan found involves going straight to the goal and deploying countermeasures along the way, as evidenced by the straight line of magenta asterisks between the start and the goal.

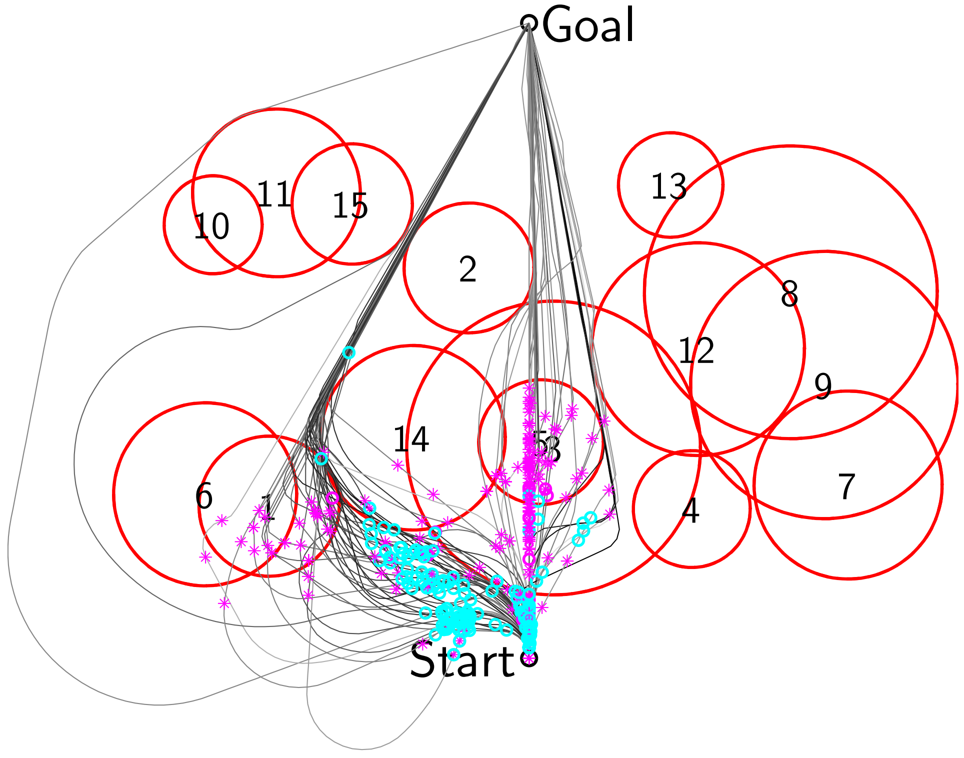
In the body frame, the vehicle fires more, but it disables threats proportionately less often. The UAV accomplishes 13 hits out of 283 shots with body-frame primitives (5%) and 13 out of 47 shots with threats-frame primitives (30%). The reason the UAV takes so many shots in the body frame is likely related to the GA's difficulty in finding good trajectories. If the GA finds an improved plan in its last few generations, and that plan happens to contain a firing action, then the GA might not have time to find a mutated version of that plan that removes the firing action and therefore the cost of firing. The accuracy is in part low because there is no maneuver paired with firing to move the UAV into a high-probability position (see Figure 4.3). As the GA tries to steer the vehicle between threats, it is unlikely that the vehicle will ever point directly at a threat at close range. The two objectives compete, and the UAV rarely positions itself effectively for a shot.

Mutation is a key factor in using GA to search for solutions. With body-frame maneuvers, a mutation in an early maneuver changes the trajectory drastically. With the threats frame, a small mutation in an early primitive does not drastically change the entire plan. For example, if a plan calls for skirting threat 14 then threat 15, changing the primitive parameters for skirting threat 14 will not greatly alter the UAV's skirting of threat 15, except for approach direction and speed. On the other hand, threat-linked primitives make small mutations between threats difficult because neighboring threats often have non-neighboring labels. Creating a custom mutation parameter that links neighboring threats in a graph could prevent this phenomenon, as could linking only the firing action to threats and using a waypoint-based library for maneuvering.

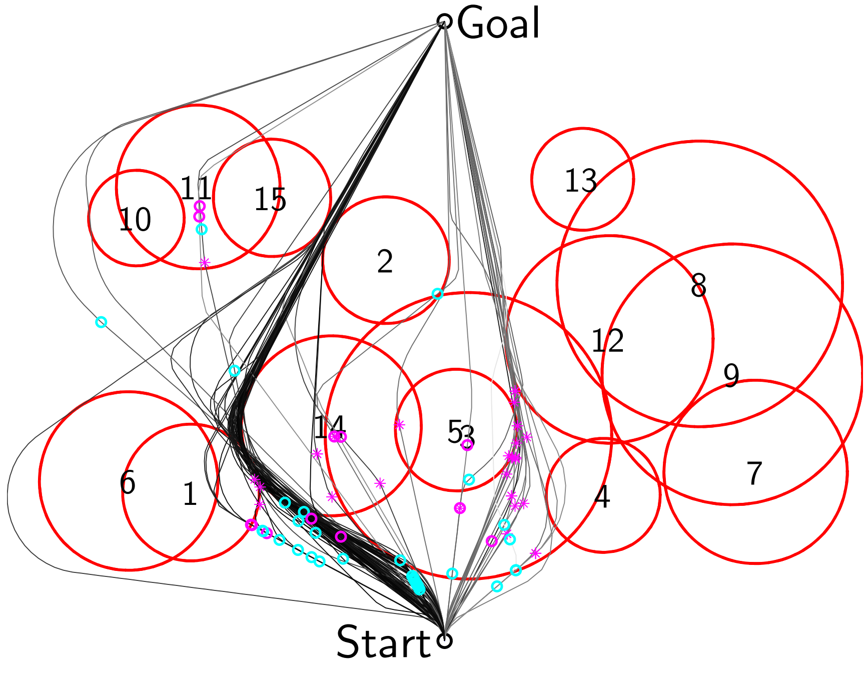
5.5 Vehicle Characteristics

The flexibility of the combined reactive/deliberative planner stems from the flexibility of GA, which works to optimize whatever fitness function it is given. The purpose of this section is to demonstrate that the algorithm can easily accept different vehicle parameters with no structural changes. The data in this section consist of 100 full runs of the combined planner on four vehicles with different turning radii, velocity thresholds, and weapon probability functions. The four vehicles have the parameters shown in Table 5.2. Vehicles 1, 2, 3, and 4 can be characterized by the titles "slow, bad shot," "slow, good shot," "fast, bad shot," and "fast, good shot," respectively.

Three trajectory characteristics are notable. First, the vehicles that have greater weapons range and accuracy fire weapons more often than the vehicles with poorer weapons properties. Second, the vehicles that are slower with better turning radius make use of the turning radius to steer more precisely between threats. Third, though feasible trajectories between threats 1 and 14 exist for the fast vehicles, they do not



(a) Body frame.



(b) Threats frame.

Figure 5.5: Comparison of trajectories between using (a) a body-frame primitives library and (b) a threats-frame primitives library. The figure in (b) is the same as Figure 4.12. Greyscaling of trajectories to denote fitness is independent between the two figures.

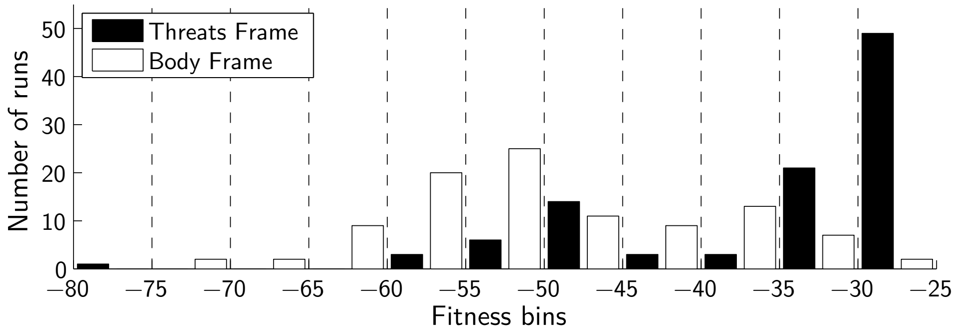


Figure 5.6: Histogram comparing fitnesses between the body-frame run of Figure 5.5a and the threats-frame run of Figure 5.5b.

Property	1	2	3	4
Minimum turning radius (m)	1	1	5	5
Maximum hit probability	0.7	0.95	0.7	0.95
Maximum weapons range (m)	4	10	4	10
Velocity threshold (m/s)	1.5	1.5	2.5	2.5

Table 5.2: Four different vehicle parameter sets. Vehicles 1, 2, 3, and 4 can be characterized by the titles "slow, bad shot," "slow, good shot," "fast, bad shot," and "fast, good shot," respectively.

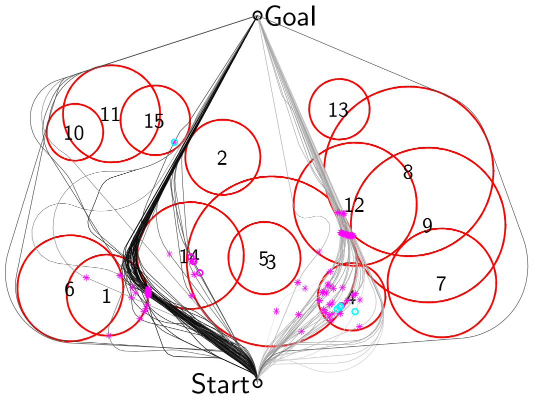
find them as often as the slow vehicles because the fast vehicles travel farther before replanning.

5.6 Initialization with Candidate Solutions

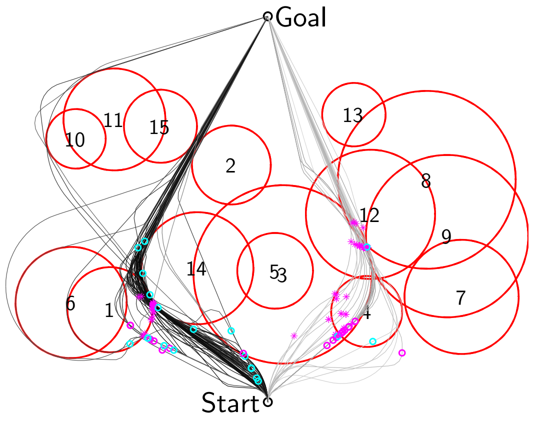
Schultz and Grefenstette evolved `IF-THEN` rules for directly mapping sensor values to actuator values in an evasive maneuvers problem, where a UAV attempted to evade a missile in a two-dimensional environment [44]. At first they initialized their evolutionary algorithm with empty plans. They then found that starting with a human-generated ruleset led to faster fitness increases and higher fitness values. Thus, they showed that seeding an evolutionary algorithm with human-inspired candidate solutions can lead to better performance.

In the attack scenario of Chapter 4, the initial GA population of 80 plans included a single seed plan and 79 random plans, but the GA often tossed out the seed plan because it passed through the no-fly zone, which resulted in low fitness. One seed plan did not give the GA sufficient opportunity to explore that highly constrained region of the problem space. Several seed plans with random variations could provide enough genetic material for the GA to find a good plan of the same form as the seed plan.

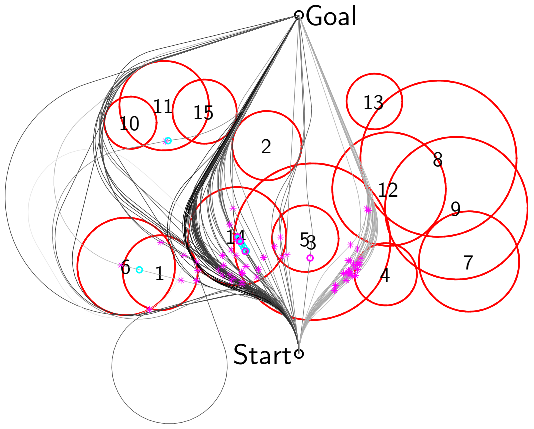
Figure 5.8 shows both the single-target, single-seed attack scenario from Chapter 4 and the same scenario with ten seeds. Like the single-seed case, the ten seed plans



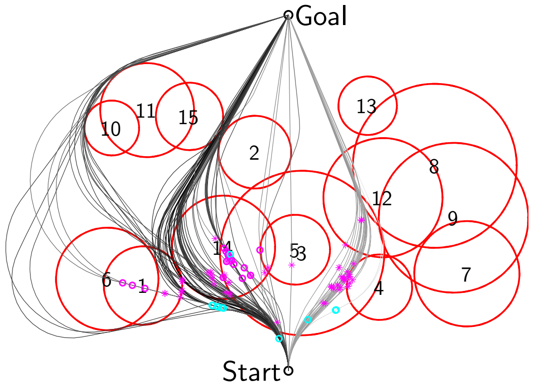
(a) Slow, bad shot.



(b) Slow, good shot.



(c) Fast, bad shot.



(d) Fast, good shot.

Figure 5.7: Trajectories of vehicles with different parameter sets. *Note:* the figures do not have quite the same scale, though the scenarios are the same.

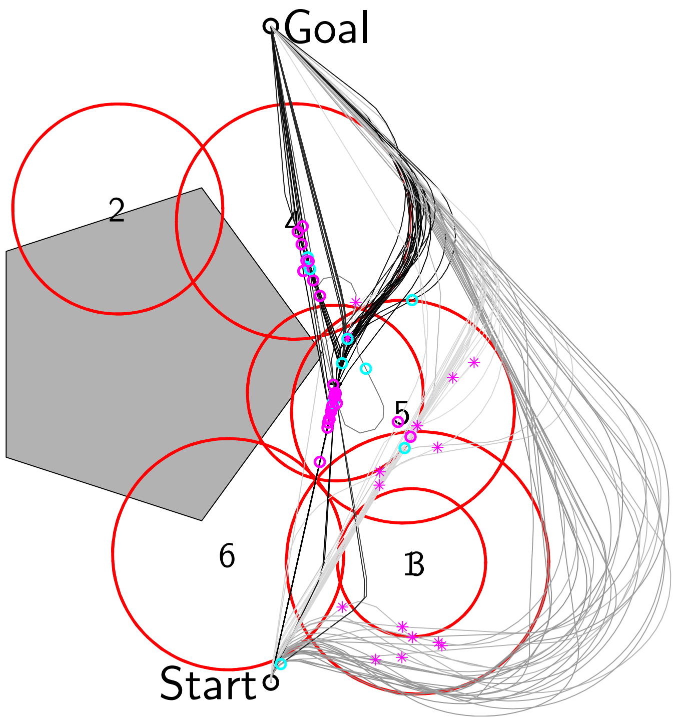
have a *null* primitive, a firing primitive, and four more *null* primitives. The firing primitive is initialized for threat 7 with a random firing distance, random evade flag, and velocity parameter at the velocity threshold.

With ten varied seeds, the planner found plans involving a shot on threat 7 much more often than before. Initial shots on threat 7 were often much closer to the threat, and fewer trajectories entered the no-fly zone, though two trajectories still clipped the no-fly zone during a second pass at threat 7. Only five out of 100 trajectories did not shoot at threat 7 at all. For those 5 trajectories, all of the ten varied seeds must have resulted in poorly performing plans which the GA discarded. Thus, even multiple seeds can lack the genetic material to guarantee that a form of their plan will be chosen unless an initial plan is known to have high fitness.

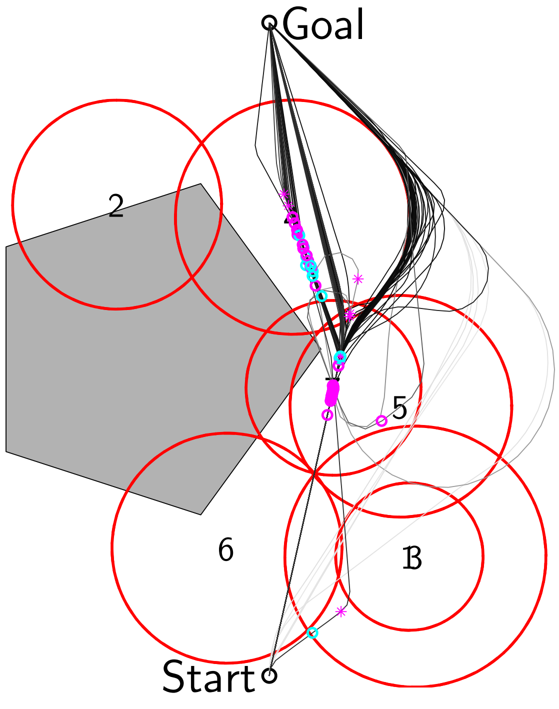
Good candidate solutions can improve performance if they supply the GA with enough genetic material to search the local problem space for a high-fitness solution. Though Schultz and Grefenstette seeded their algorithm with a human-generated plan, the seed plans in the attack scenario were generated with a simple rule involving the *fireonthreat* primitive and the target labels. Thus, pre-processor algorithm is also a valid means of generating candidate solutions. Humans and algorithms both should generate good candidate solutions, as bad candidate solutions could decrease performance by focusing the GA's search in one low-fitness area and causing undesireable convergence.

5.7 Lessons Learned

These simulations give many useful insights into the details of GA and the reactive/deliberative planner. This knowledge can help an engineer make critical design choices. When deciding the number of GA generations, a designer should test improvement and find the "sweet spot" that balances final fitness and the cost of computation. Low dimensionality and the adjacency of physical problem elements in the chromosome does lead to better GA performance, so that model reduction of some form and careful chromosome representation can be worth the effort. An optimized gradient search algorthm could give the best combination of speed and performance, though results with GA would still be comparable. Each tactical problem needs appropriately designed primitives that group linked actions and mutate in a sensible way. A designer can trust the generality of the planner when applying similar primitives to different vehicles. Finally, initializing the GA with a variety of sensible solutions leads to better performance, so that using variations of human-generated solutions or creating a pre-processor algorithm to find approximate solutions are both worthwhile. The next chapter summarizes the findings on the combined planner and describes future research directions.



(a) Single seed.



(b) Ten seeds.

Figure 5.8: Attack scenario with (a) a single seed for firing on threat 7 at a distance of 3 meters and (b) ten seeds for firing on threat 7 at various distances. Subfigure (a) is the same as Figure 4.20. *Note:* threats 1 and 3 overlap so that their labels look like 1̶3. Greyscaling of trajectories to denote fitness is independent between the two figures.

Chapter 6

Conclusion

Unmanned aerial vehicles (UAV) and other autonomous vehicles (AV) have and will continue to assist and replace people in repetitive or dangerous missions, and they must be able to generate high-performance plans in rapidly changing environments with little-to-no human input. Planning in the presence of hybrid dynamics, that is, including both continuous and discrete system dynamics, requires specialized optimization algorithms to search among problem space discontinuities and avoid local optima.

This thesis presents a framework for hybrid-system planning for systems whose relevant behaviors can be encapsulated into high-performance, packaged, parametrized actions called tactical primitives. Tactical primitives contain continuous behaviors such as flexible maneuvering, discrete switching between these continuous maneuvers, and discrete actions such as firing a weapon or deploying countermeasures. In this framework, planning involves choosing a sequence of primitives and their associated parameters. The choice and design of tactical primitives affects optimization performance, as shown in Section 5.4, and the design of tactical primitives offers a means of capturing complex human-inspired input for a UAV's use.

Through processes that reflect concepts from biological evolution, a genetic algorithm (GA) attempts to improve a fitness function depending on the primitives' discrete and continuous parameters. GAs can find good primitive sequences in relatively few fitness function evaluations. GA outperforms the mixed GA/nonlinear-programming (NLP) algorithm of Section 5.3 in terms of fitness gain for short computation time. Given its speed, GA is a promising algorithm for real-time planning in cases where the fitness function itself requires very little computation time. GA with NLP outperforms GA alone for long computation times, and with an optimized gradient search, it could be a viable solution method. However, GA tailored to avoid convergence to local minima could outperform GA with NLP. Both variations in GA and GA with NLP need more investigation as they relate to planning with tactical primitives.

The fitness function can capture complex problem elements such as probabilistic outcomes and constraints. Trajectories can branch on uncertain events, and probability distributions can weight the fitness of multiple outcomes to produce an expected fitness value. However, many branches result in a large increase in computation time. Fitness penalties or chromosome manipulation effectively enforce problem constraints, though fitness penalties must be applied carefully to avoid having rewards offset penalties in tightly constrained situations.

GA offers few solution guarantees, but several methods can increase the probability of finding a good solution, such as problem abstraction, receding horizon control, and initializing the GA with good plans. Section 5.2 showed a decrease in GA performance even for extra options along a single discrete-parameter dimension, giving evidence that problem abstraction and receding horizon control could result in higher fitness by reducing problem dimensionality. Section 5.6 successfully demonstrated that the latter method is effective. With multiple seeds, most plans successfully find a high-fitness solution. These methods show promising ways to overcome local optima and high dimensionality, which work together to make the globally optimal solution very difficult to find.

Some rapid environmental changes, such as the presence of a new threat, require rapid vehicle response to ensure survival. With deliberative planning alone, GA might evaluate the fitness function several thousand times before it finds a suitable response. The reactive planner modifies the vehicle's plan with one of several candidate modifications. These modifications involve inserting or changing primitives that are designed offline to capture behaviors that can boost fitness quickly. With few enough candidates, an AV can quickly test each modification and execute the best one. Because reactive plan modifications are designed offline for certain scenarios, reactive planning might not produce good plans when faced with scenarios with frequent rapid change.

The combined reactive/deliberative planner produces plans for some nonlinear, hybrid, probabilistic, changing environments. It is flexible and general enough to work on any system whose necessary behaviors can be captured in a small primitives library and whose fitness function requires little computation. The ideas of tactical primitives, GA optimization, and rule-based reactions have much room for improvement and can extend to other hybrid problems. The next sections discuss future directions for related research.

6.1 Immediate Extensions

The following paragraphs present clear extensions to the combined planner. Some ideas have already been discussed in previous chapters, such as tailored mutation of primitives in the GA, better plan representation in the chromosome through lists, pre-selection of reactive plans using rules, methods of avoiding GA population convergence, and optimized gradient search. Their details are not reproduced here.

Trajectories branch on probabilistic events during planning, but plans remain the same "downstream" of an event. In certain cases, it makes sense to condition future primitives on the outcomes of events. For example, if shooting and disabling a threat opened an exploitable corridor, but missing that shot meant that the best path involved turning around and circling a group of threats, then the plan itself should branch on the shot. Fitting this approach into the GA framework would require either a creative use of the standard fixed-length array or a tailored chromosome representation like a tree structure.

Section 5.2 showed that GA performance decreases with increasing numbers of environmental features and suggested that receding-horizon planning could mitigate this effect. In receding-horizon planning, the planner considers only a neighborhood around the vehicle with full planner fidelity, and outside that neighborhood, i.e., beyond that horizon, the planner approximates the trajectory cost. The design of the horizon and the cost-to-go function determines to a large part whether the planner successfully finds high-fitness trajectories. The horizon could be different for different features such as threats versus no-fly zones or targets. Cost-to-go determination could be a fast trajectory generation algorithm such as Asseo's circular-threat navigation algorithm [5].

Another method of reducing dimensionality is the abstraction of environmental features, which is a form of model reduction. For example, the convex hull of a group of threats could replace the individual threats. In addition to high number of environmental features, Section 5.2 showed that increasing disjointedness of those features in the problem space decreases performance. A sorting algorithm for arranging environmental feature labels could restore that performance. Development of these pre-processor algorithms could lead to an increase in planner performance.

Because GA searches stochastically, there is no guarantee that it will find a "good-enough" solution, though Section 5.6 showed that seeding the GA with good candidate solutions effectively improves the possibility. Along other lines, the "safe trajectory" idea would provide the planner with time to optimize its plan and provide a safe loiter area for the vehicle to return to should it need more time to plan [43]. The planner could identify safe loiter zones through GA or through a tailored algorithm. The loiter could itself be a primitive that enters the GA like other primitives, or it could be a candidate reactive plan modification. Safe loiter zones would provide the planner with the time it needs to find good solutions.

Guaranteeing good performance under different types of uncertainty is an integral part of trajectory planning. The fitness function in the threats problem modeled the uncertain outcome of firing a weapon on a threat, and Frazzoli and Dever's maneuver primitives each have bounded trajectory errors for bounded disturbances [16, 18]. Replanning helps lessen the effect of errors on system performance. In a real system, many other states and sources of error exist, such as vehicle health, partial threat damage, and sensor error. Generating all possible outcomes and weighting them by a probability distribution can quickly become computationally infeasible. Studying which sources of error can be ignored and how to incorporate greater dimensions of error without incurring too much computation can make the planner feasible for more realistic scenarios.

Tactical primitives so far have been limited to simple kinematic trajectories combined with simple discrete actions. Firing a missile in reality requires communication, targeting, arming the missile, firing, and target damage assessment. In theory, tactical primitives can capture even complicated human behaviors [7, 23]. However, online optimization with complex primitives poses the unanswered challenge of rapidly calculating fitness on a high-fidelity model with limited computational resources while preserving robustness. Extending the methods of this thesis to apply them to a realistic problem would expose the method's weaknesses and bring it closer to actual implementation.

Along these lines, the urban warfare and surveillance scenario poses many challenges that would test the tactical primitive framework. In this environment covered with various buildings and possible threat types and locations, complex events evolve rapidly, giving the vehicle little time to react, requiring a strong library of reactive tactics and a focused GA search of the problem space. Tactical primitives offer the potential to perform agile maneuvers while simultaneously executing discrete actions, such as taking surveillance photographs and using specific sensors.

Another system that would stress environmental modeling is the autonomous underwater vehicle (AUV). Though AUV scenarios often evolve at a slower tempo than UAV scenarios, the complex acoustic environment would make fitness function evaluation expensive, and limited sensing would make many errors and probabilities enter the model.

6.2 Future Work

Many extensions to the idea of planning using tactical primitives would require extensive research beyond what has been done. This section describes such extensions whose nature diverges from the focus of this thesis.

Graph search is a well-studied problem, and many variations exist to accommodate different time and space constraints. In cases where a heuristic estimate of the cost from a graph node to the goal node exists and meets certain criteria, algorithms such as A*, Dijkstra's algorithm, and branch-and-bound can exploit the heuristic and graph structure in their search for a solution [41]. GA, on the other hand, does not use any problem information except the fitness function. Much like the mixed GA/NLP algorithm of Section 5.3, graph search could choose discrete parameters while some version of NLP optimizes continuous parameters. Heuristic design for this arrangment would offer difficulties, as the cost of a particular sequence of discrete parameters would depend highly on the optimized continuous parameters so that a heuristic is ineffective when building a trajectory node by node. A simple greedy depth-first search that continues to explore the graph after finding a solution could still offer an advantage over GA by directing its search.

No general method exists for the automatic generation of a primitives library, a fact first noted by Frazzoli [18]. Using a linear systems analogy, primitives should be orthogonal and span the problem space, but no such concepts have been developed for maneuver primitives. Some progress has been made in classifying and constructing primitives based on discussion with human subject matter experts and observation and deconstruction of their approaches to different situations [7, 23].

In the future, UAVs might work alongside pilots as scouts or wingmen. Because GA is stochastic, its proposed solutions are unpredictable, but pilots and the military in general value predictable behavior and the chain of command. In order to achieve predictable behavior with tactical primitives, plans could be generated entirely from rules, or they must have certain properties. Reactive tactics currently works on the low level of sensor to actuator mappings and has had few successes beyond control's lowest levels [44]. Integrating tactical primitives with a rule-based or other straightforward algorithm would offer the possibility of high-performance behavior and predictability, which may be desireable over optimality in some situations.

Finally, incorporating other vehicles both cooperatively and competitively offers a major challenge. The coordinated control of multiple vehicles is an active area of controls research [13]. Tactical primitives offer an excellent means of reducing the action spaces of each vehicle for high-performance coordinated control in rapidly evolving environments. Though GA would not be able to handle rapid evolution, it has been applied to multi-vehicle path planning [46]. The many aspects of coordinated control, such as concurrent actions and probabilistic dependencies, dynamic networks, multi-vehicle surveillance, and guaranteeing safe trajectories, potentially all benefit from the tactical primitives concept.

Appendix A

Genetic Algorithms

A genetic algorithm (GA) is a stochastic technique for searching a problem space for an optimal solution. Holland originated the idea of GAs in the 1960s, and since then they have been widely used because of their flexibility and straightforward nature [24]. GAs are members of a broad family of evolutionary computation methods, which involve iterative, random, population-based searches. The term *genetic* refers to similarities between GAs and the concepts of mutation and recombination from evolutionary biology. Many references discuss the underlying theory of GAs and the details of applying GAs to various problem domains [6, 20, 24]. This appendix offers a summary of those references and details about tailoring GA to planning with tactical primitives.

A.1 Overview

GAs consider several solution candidates in each algorithm iteration. As a group, these candidates are called a *population*, and the population of a specific iteration is called a *generation*. At each iteration, every individual is assessed by a fitness function, which is user-designed according to the problem. The GA evolves the population into the next generation based on that fitness. The GA keeps track of the overall best-performing individual, and when a termination condition is met, that individual is returned.

Natural selection is the evolutionary idea behind GAs. If an individual is more fit for its environment than the peers in its generation, information from its genes is more likely to survive and be passed on to the next generation. Inversely, if an individual has worse fitness than its peers, its genetic material has less chance of survival.

Evolutionary concepts found in most GAs include chromosomes, recombination, mutation, fitness, and selection. Briefly, selection chooses the best solution candidates of a particular iteration, and recombination and mutation evolve those candidates from one iteration to the next, preserving good solution characteristics. Pseudocode for a typical GA is found in Figure A.1.

```
population $P_0$ = initialize(pop-size);
best-individual $B$, best-fitness $f_B$;
generation number $g$;
while NOT termination-condition($P_0$,$f_B$ =NULL,$g = 1$) do
    $P_g$ = recombine-and-mutate($P_{g-1}$);
    $f_g$ = calc-fitness($P_g$);
    [$B'$,$f_{B'}$] = get-best-individual($P_g$,$f_g$);
    if $f_{B'} > f_B$ then
        $B = B'$;
        $f_B = f_{B'}$;
    end
    $P_g \leftarrow$ select($P_g$,$f_g$);
    $g = g + 1$;
end
```

Figure A.1: Pseudocode for a typical genetic algorithm.

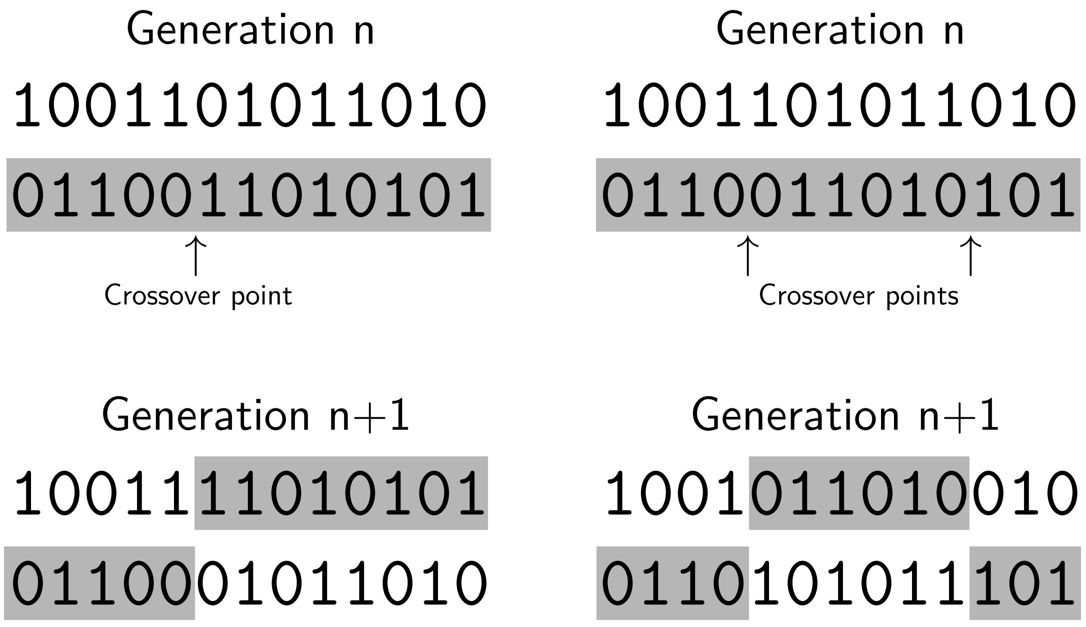
A.1.1 Chromosomes and Problem Encoding

In order for a parameter optimization problem to fit into the GA framework, its parameters must be encoded into a *chromosome*. Many options exist for how to encode a problem into a chromosome. The encoding chosen for a particular problem affects how well the GA performs, i.e., how fast the GA improves the fitness of the best individual and what final fitness is achieved.

For a typical GA, the chromosome is a bitstring or vector of floating-point numbers. In the case of floating-point numbers, each element in the vector is a parameter. In the case of a bitstring, the first n_1 bits could represent the first parameter, the second n_2 the second, etc., allowing the resolution of each parameter to be set independently. A simple continuous parameter encoding could be a linear mapping of its bit range to the desired parameter range, $[p_{\min}, p_{\max}]$. A simple discrete parameter encoding could be a one-to-one mapping between bitstring values and the parameter domain. Other possible encodings include nonlinear mappings or piecewise-linear mappings onto disjoint sets.

For high-level GAs, the chromosome can be a more complex data structure. Examples include trees, arrays, robotic lifeforms, and rules for reactive behaviors [21, 31]. Typical GA functions that are designed for bitstrings or other numerical representations might not work on these data structures, but however the problem is encoded, the mutation, recombination, selection, and fitness functions of a GA must be tailored to carry out their respective purposes.

A chromosome represents the form of an element in the problem space, e.g., three real numbers in the set $[0, 1)$, or a tree up to five levels deep with zero to three branches at each node. An individual solution candidate has the form of the chromosome and specific values. The basic encoding in bitstrings or floating-point numbers is called the *genotype*, and the genotype decoded into a set of parameters used in the calculation



(a) Single-point crossover. (b) Multi-point crossover.

Figure A.2: Typical bitstring recombination methods.

of an individual's fitness is called the *phenotype*.

A.1.2 Recombination

Recombination or *crossover* is the first part of the process of creating a new generation from a previous generation. It involves combining two or more genotypes of one generation (the parents) to form the new genotypes of the next generation (the children). The hope is that children will receive a combination of the good qualities of each parent. Besides combining the good properties of multiple parents into one child, recombination also provides a means of making large jumps in the search space.

A typical recombination is a single-point or multiple-point crossover. In the crossover, genetic material from two parent genotypes is divided into pieces, and material is exchanged to form two children. Bitstring recombination is illustrated in Figure A.2.

Recombination for non-bitstring representations involves swapping material in some other way. For an array, selecting a single row and column and swapping subarrays defined by that row and column constitutes a valid recombination. For a list, elements could be clustered before swapping. Some evolutionary computation problems with high-level representations have no obvious or meaningful way to exchange material, and they ignore recombination altogether.

A.1.3 Mutation

Mutation seeks to push individuals into new areas of the problem space by making random changes to individuals. Without mutation, a GA's search is limited to the genetic material it begins with. With mutation, any point in the search space is theoretically reachable. Compared to recombination, mutation typically makes small changes. This helps to explore problem space regions that have large variations of fitness in a small area.

Many options exist for how to perform mutation. For bitstring representations, a mutation operator could flip each bit with a certain probability, or it could flip a random bit with a certain probability. For floating-point representations, a parameter could be set to a uniform random number, replaced with its maximum or minimum value, or changed by a small amount, all with some specified probability. Mutation methods are chosen for a particular problem according to their appropriateness and effectiveness.

A.1.4 Fitness Evaluation and Selection

A GA will gradually find the fittest individual in a search space, where fitness is a user-defined function from problem variables to a scalar value. In the case of function optimization, the fitness is simply the value of the function. However, many problems involve optimization of multiple competing quantities. In an autonomous vehicle mission, time, fuel spent, health, risk, and targets destroyed must be optimized simultaneously. Design of the fitness function is heavily problem-dependent. Sometimes a simple sum will suffice, and sometimes physics or economics must be considered.

After calculating fitness, individuals must be selected for reproduction. A fittest fraction could be chosen. In this scenario, the population could quickly lose genetic diversity by not preserving at least some poor performers. The probability of selection could be weighted by the fitness. This is called roulette wheel selection, and it was one of the first selection methods of GAs. This way, fitter individuals are more likely to go on, but some poor-performing individuals still pass through and add diversity, which can help avoid the problem of convergence to a local optimum. Instead of fitness, the probability of selection could be a function of fitness rank. This could be done with a geometric distribution, where the first individual is chosen with probability p, the second with $p \cdot r$, the third with $p \cdot r^2$, and so on, with p and r chosen to normalize the distribution. The disadvantage of selecting by probability is that dominant genotypes can be selected multiple times, diminishing the genetic diversity of following generations. All of the above are useful selection methods.

A.1.5 Performance Metrics

A GA's performance is measured using fitness over time. The fitness of the best individual found so far versus generation number reflects what the GA would return at a particular point. Other performance measures include the fitness of the best individual in each generation, which is not necessarily the best found so far because of random selection, recombination, and mutation, and the mean fitness versus generation. These give greater insight into the behavior of the GA, such as whether the mutation rate is too high or the population too similar.

Another important metric is the rate of fitness increase versus fitness value. The higher this rate, the better the material the GA has to work with. If a GA improves slowly at a relatively low fitness, the population has likely converged on a local minimum.

A.1.6 Variations

Algorithm modifications often increase GA performance. One example is a local simplex search with every iteration or at regular iteration intervals. GAs are very good at escaping regions of local optima and finding the region of the global optimum, but they are not very good at finding the exact optimum. Without explicit hill climbing, GAs rely on stochastic jumps to find the solution, and the probability of selecting a single element in a continuous space by random search, even with the selection pressure of GAs, is negligible. If a simplex or hill-climbing search is paired with a GA, it can converge much faster and much more accurately.

If a GA designer chooses a high mutation rate, this might offset natural selection by always pushing solutions to random positions. If the mutation rate is too low, the population may converge to some local optimum because of the lack of new random genetic material to try points outside the local optimum region. Some ways of dealing with this tradeoff are simulated annealing, replacement, and similarity fitness penalty [6]. In simulated annealing, the mutation rate starts out at a high value and then decreases with each iteration. This allows for early exploration and later convergence. In replacement, randomly generated genotypes replace individuals that are the most similar to the rest of the population. Similarly, fitness penalties for similarity to the rest of the population promote diversity.

A.2 Primitive Selection and GA Design Issues

GAs are flexible search algorithms that are able to take high-dimensional, hybrid, nonlinear, discontinuous, multiobjective optimization problems and return near-optimal solutions. Because of their rapid, broad stochastic search, they are well-suited for the selection of tactical primitives and their associated parameters, which is a hybrid, nonlinear problem with a high-dimensional, complex fitness landscape. GA performance on these problems depends on many design factors, including problem representation, initialization, fitness, and constraints. GAs are very sensitive to each of these factors, and careful choices must be made to ensure that the GA will improve system performance.

A.2.1 Problem Representation

When treating tactical decision making as a parameter optimization problem, careful selection of the chromosome representation is required. A problem can be encoded into a chromosome in many different ways, and only some of them will yield effective GA performance.

One principle for selecting problem representation is the limitation of system behavior to a small set of rich behaviors that are known to contain "good solutions." One example in the literature is the generation of low-altitude, evasive UAV maneuvers in the presence of a pop-up threat [38]. If the GA parameter set includes a change in pitch at frequent time intervals, then random mutations in the GA will lead to many solution candidates that involve pitching into terrain. However, if one parameter is simply the change between several pre-defined levels of altitude, no solution will lead to terrain collisions. Thus, the problem space includes valuable altitude variations but avoids many solution candidates that are obviously worthless.

Another principle is the adjacency of phenotypes in the solution space [6]. One reason to cluster phenotypes is that GAs are good at searching for solution regions but not specific points. This approach is also important for effective recombination. If a problem encoding puts parameters that are phenotypically similar close together, then meaningful clusters can be exchanged between individuals. For example, in the above evasion problem, an initial quick turn combined with a drop in altitude might be a consistent feature of a good solution. If those two parameters are clustered in the chromosome, they are more likely to be preserved as a group.

When parameters are chosen, their set of possible values must be chosen. They can be discrete, continuous, or mixed, and their range and scaling should be chosen to reflect problem-specific preferences in selection. The fidelity of the parameters is important; too many possible values creates a larger search space, and too few reduces problem fidelity. In terms of mapping, a bitstring of three bits could represent the numbers zero through seven or some function of that array, such as 2^0 through 2^7. Even though this range stretches from 1 to 128, the probability of selecting values of 16 or below is $\frac{4}{7}$. Because all problems have their own natural scaling, the design of parameter domains is an important consideration.

Another important design factor is the form of the chromosome. As mentioned above, the simplest GA encoding is the binary string, but many other encodings exist. In the problem of selecting sequential tactical primitives, it may be necessary to vary the number of primitives in the chromosome, especially if the primitives are of different lengths in time, and a fixed time horizon is employed. Some options for variable length encoding include using a fixed-length list of actions and padding with a *null* action, with a fitness penalty for too many *null* actions; using a fixed length and weighting the fitness calculation by a time envelope, thus diminishing the effect of actions beyond a certain time; and using a true variable-length representation such as a linked list.

The methods of padding a chromosome and applying a time envelope can be fit into a standard GA optimization library, but they have extra genetic material that is discounted and is therefore allowed to drift without selection pressure. For example, if a chromosome involved one parameter that determined the maneuver type, and one parameter that represented an optional maneuver parameter, assigning the maneuver to *null* would leave the parameter without purpose, and it would be allowed to drift until reactivation by the selection of another maneuver type. The advantage of the third method is that no genetic material is ignored, but a GA algorithm would require a tailored approach.

A.2.2 Population Initialization

Most GA program libraries come with a function that initializes a population to a set of random chromosomes [25]. This gives the GA a large diversity of genetic material to explore. However, this approach is not the only option. It has been shown that GAs are more effective if they are seeded with good candidate solutions. In [44], the authors showed that good initial candidates increase both the final fitness of the solution and the rate of fitness improvement at a particular fitness. Though simply incorporating known good genotypes into a population of random candidates will work, the good genotypes would dominate a roulette or geometric selection technique. Another proposed method is to initialize the population with several mutated versions of the good candidate, where the mutation rate is inversely proportional to the candidate's confidence [1].

A.2.3 Fitness Function Design

Selection and scaling of a GA fitness function can be a large factor in determining GA performance. For example, in roulette-wheel selection, the probability of selection is weighted by fitness. If a high fitness value is orders of magnitude above a low fitness, the diversity of low-fitness candidates will be lost because low-fitness candidates have negligible probability of being selected. On the other hand, if fitness varies over a small positive range, such as $[10, 11]$, high-fitness candidates will have little selection advantages over low-fitness candidates with roulette selection.

Several methods overcome these selection problems. One method is fitness scaling or remapping, which involves subtracting an estimated minimum fitness and dividing by an estimated mean fitness. Estimating the mean and minimum fitness can be done by taking a Monte Carlo histogram of the fitness function, or by looking at the statistics of several preceding generations during a GA run, in which case the scaling is adaptive and is called *fitness windowing* [6]. One method that avoids the need to estimate fitness function bounds is fitness ranking, as described in Section A.1.4.

If a problem has multiple objectives, it is important to appropriately combine them into one fitness value so that no objective unexpectedly dominates or is ignored. Table A.1 shows a list of possible objective combinations.

Often a problem will involve a probabilistic element, such as the probability of hitting a target. This element can incorporated into the fitness. One way of modeling probabilistic events is to use Monte Carlo sampling and take the average fitness value to represent the fitness. This approach requires more evaluations and therefore more time. Another method is to enumerate every outcome and weight each by probability.

Weighted average	$(\sum_{i=1}^{m} w_i x_i)/(\sum_{i=1}^{m} w_i)$
Maxi-min	$\min(\vec{x})$
Exponential inverse	$1/(x_1^{\gamma_1} x_2^{\gamma_2}), \quad \sum \gamma_i = 1$
Pareto optimal	$\{\vec{x} : (\nexists \vec{x}' \in (\text{feasible } \vec{x}) : \vec{x}' > \vec{x})\}$

Table A.1: Options for combining multiple objectives into one fitness [20]. The symbols x and w correspond to objective values and weights, respectively.

For example, if an AV fires on a threat, then the simulation could continue with two branches, one for success and one for a miss, and the resulting fitness would be the fitness of each branch times its associated probability.

A.2.4 Problem Space Constraints

If a problem space is non-convex, the GA must handle infeasible genotypes that appear. These infeasibilities can arise from environmental constraints, such as obstacles in a motion planning problem, or they can be imposed by a designer who knows that certain feasible regions have no possibility of producing a useful solution. Feasible candidates can be enforced in several ways [38].

First, it is preferable to have a problem space mapping that avoids such regions, but this mapping and the prevention of infeasible candidates can be difficult.

Second, the GA can test for feasibility and delete infeasible candidates, replacing them with some other genotype, such as a random genotype. This is straightforward, but it may take more time than is desired, especially when the problem space is mostly infeasible, such as in a problem whose goal is to find a feasible solution.

Third, infeasible candidates can be repaired. This can be done by repositioning them to the nearest feasible region of the problem space, which has the problem of increasing the density of candidates at the edges of infeasible regions. Also, a specialized repair function might be difficult to construct for certain problems.

Finally, a fitness penalty can be imposed on infeasible candidates. This avoids candidate clustering and keeps the diversity of genetic material from infeasible regions. The fitness penalty is most useful when it captures the distance to feasibility, not just the fact of infeasibility or the number of constraints violated. If infeasible solutions are allowed, however, there is no guarantee that the GA will return a feasible solution. This can be handled by repairing genotypes before they are compared to the best individual.

A.2.5 Termination Condition

The termination condition for a GA is very flexible. One method is to have the GA return the best individual found at a regular interval, and to keep the GA running indefinitely. Though this method works for offline optimization, online optimization can be run either for a certain number of generations or for a fixed time. The best individual can then be returned.

Advantages	Disadvantages
• Straightforward and robust.	• Objective function design difficult.
• Handles high-dimensional, hybrid, nonlinear problems with many local optima.	• Chromosome representation difficult.
• Handles mixed parameter type and resolution.	• Stochastic; solution not guaranteed optimal.
• Easily parallelized for distributed computation.	• Must be fast enough to run many times.
• Operates on a group.	• For complex problems, must approximate to get good run times.
• Allows example solutions in initial population.	• Does not leverage problem specifics unless specifically designed.

Table A.2: Advantages and disadvantages of genetic algorithms.

The default termination option for many GA programs is to terminate after a fixed number of generations. However, certain population and fitness metrics can signal that a GA is unlikely to find a better solution, shortening an optimization run. If a theoretical optimum can be calculated, one termination metric is the proximity of the fitness to this optimum. Another metric is the level of population convergence. When a population converges, it has either found the region of the global optimum, or it has found a local optimum that it is unlikely to escape. Measures of population similarity include number of similar bits and weighted r.m.s. on the difference of parameters.

A.3 Conclusion

GA works well on nonlinear, hybrid problems on which other methods easily get trapped in local optima. It manages hybrid characteristics through the fitness function, which maps both continuous and discrete inputs to a single fitness value. GA rapidly explores large problem spaces through crossover and random mutations to avoid local optima. In addition, a designer has a host of options when implementing GA on a particular problem. These options often grant sufficient flexibility to find a version of GA that works very well. Careful design of the problem representation and the fitness function are the two most important elements. Table A.2 contains a list of advantages and disadvantages of the GA algorithm.

Bibliography

[1] Mohammad-Reza Akbarzadeh-Totonchi and Mohammad Jamshidi. Incorporating A-Priori Expert Knowledge in Genetic Algorithms. *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*, pages 300–305, 1997.

[2] Ramesh Amit and Maja J. Matarić. Learning Movement Sequences from Demonstration. In *Proceedings of the International Conference on Development and Learning (ICDL-2002)*, volume 8, pages 302–306, June 2002.

[3] Tom M. Apostol. *Calculus Volume II.* Wiley, second edition, 1969.

[4] Ronald C. Arkin. Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[5] Sabi J. Asseo. In-Flight Replanning of Penetration Routes to Avoid Threat Zones of Circular Shapes. In *Proceedings of the IEEE 1998 National Aerospace and Electronics Conference*, pages 383–391, 1998.

[6] David Beasley, David R. Bull, and Ralph R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58–69, 1993.

[7] Jonathan S. Beaton. Human Inspiration of Autonomous Vehicle Tactics. Master's thesis, MIT, 2006.

[8] John S. Bellingham, Arthur G. Richards, and Jonathan P. How. Receding Horizon Control of Autonomous Aerial Vehicles. In *Proceedings of the American Control Conference*, volume 5, pages 3741–3746, 2002.

[9] Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, second edition, 1999.

[10] Michael S. Branicky, Vivek S. Borkar, and Sanjoy K. Mitter. A Unified Framework for Hybrid Control: Model and Optimal Control Theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[11] Rodney Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[12] Brett Browning, James Bruce, Michael Bowling, and Manuela Veloso. STP: Skills, Tactics and Plays for Multi-Robot Control in Adversarial Environments. In *Proceedings of the Institution of Mechanical Engineers I, Journal of Systems and Control Engineering*, volume 219, pages 33–52, 2005.

[13] Sergiy Butenko, Robert Murphey, and Panos M. Pardalos, editors. *Recent Developments in Cooperative Control and Optimization.* Springer, 2004.

[14] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press, 2000.

[15] Chris Dever, Bernard Mettler, Eric Feron, Jovan Popović, and Marc McConley. Nonlinear Trajectory Generation for Autonomous Vehicles Via Parametrized Maneuver Classes. *Journal of Guidance, Control, and Dynamics*, 29(2):289–302, 2006.

[16] Christopher W. Dever. *Parametrized Maneuvers for Autonomous Vehicles.* Ph.D. thesis, MIT, 2004.

[17] Laura Major Forest, Susannah Hoch, Alexander C. Kahn, and Marshall Shapiro. Effective Design of Planning Systems to Support Human-Machine Collaborative Decision Making. In *2006 Proceedings for the Human Factors and Ergonomics Conference*, 2006.

[18] Emilio Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning.* Ph.D. thesis, MIT, 1994.

[19] Vladislav Gavrilets, Emilio Frazzoli, Bernard Mettler, Michael Piedmonte, and Eric Feron. Aggressive Maneuvering of Small Autonomous Helicopters: A Human-Centered Approach. *International Journal of Robotics Research*, 20(10):795–807, 2001.

[20] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, 1989.

[21] John J. Grefenstette, Connie L. Ramsey, and Alan C. Schultz. Learning Sequential Decision Rules Using Simulation Models and Competition. *Machine Learning*, 5(4):355–381, 1990.

[22] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[23] Mark Hickie. Behavioral Representation of Military Tactics for Single-Vehicle Autonomous Rotorcraft via Statecharts. Master's thesis, MIT, 2005.

[24] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[25] Christopher R. Houck, Jeffery A. Joines, and Michael G. Kay. A Genetic Algorithm for Function Optimization: A Matlab Implementation. Technical report, North Carolina State University, 1995.

[26] Rufus Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization.* Dover Publications, 1965.

[27] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[28] Richard E. Korf. Macro-Operators: A Weak Method for Learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[29] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. *SIAM Journal of Optimization*, 9(1):112–147, 1998.

[30] Tiffany R. Lapp. Guidance and Control Using Model Predictive Control for Low Altitude Real-Time Terrain Following Flight. Master's thesis, MIT, 2004.

[31] Hod Lipson and Jordan Pollack. Automatic Design and Manufacture of Robotic Lifeforms. *Nature*, 406:974–978, 2000.

[32] The MathWorks, Natick, MA. *Getting Started with MATLAB® Version 7*, 2006.

[33] The MathWorks, Natick, MA. *Optimization Toolbox User's Guide*, 2006.

[34] *Merriam-Webster's Collegiate Dictionary.* Merriam-Webster, eleventh edition, 2003.

[35] Mark B. Milam, Kudah Mushambi, and Richard M. Murray. A New Computational Approach to Real-Time Trajectory Generation for Constrained Mechanical Systems. In *2000 IEEE Conference on Decision and Control*, 2000.

[36] John A. Nelder and Roger Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1965.

[37] Michael Pearce, Ronald C. Arkin, and Ashwin Ram. The Learning of Reactive Control Parameters Through Genetic Algorithms. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 130–137, July 1992.

[38] Ryan L. Pettit. Low-Altitude Threat Evasive Trajectory Generation for Autonomous Aerial Vehicles. Master's thesis, MIT, 2004.

[39] Faisal Qureshi, Demetri Terzopoulos, and Ross Gillett. The Cognitive Controller: A Hybrid, Deliberative/Reactive Control Architecture for Autonomous Robots. *Lecture Notes in Computer Science*, 3029:1102–1111, 2004.

[40] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods.* Springer, second edition, 2005.

[41] Stuart Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach.* Pearson, second edition, 2003.

[42] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed Integer Programming for Multi-Vehicle Path Planning. In *Proceedings of the European Control Conference*, pages 2603–2608, 2001.

[43] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Decentralized Cooperative Trajectory Planning of Multiple Aircraft with Hard Safety Guarantees. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2004.

[44] Alan C. Schultz and John J. Grefenstette. Improving Tactical Plans with Genetic Algorithms. *1990 Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 328–334, 1990.

[45] Alan C. Schultz and John J. Grefenstette. Using a Genetic Algorithm to Learn Behaviors for Autonomous Vehicles. *AIAA Guidance, Navigation and Control Conference*, pages 328–334, 1992.

[46] Takanori Shibata and Toshio Fukuda. Coordinative Behavior by Genetic Algorithm and Fuzzy in Evolutionary Multi-Agent System. *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, 1:760–765, 1993.

[47] Eric W. Weisstein. Circle-Circle Intersection. *From MathWorld—A Wolfram Web Resource*, May 2006.
http://mathworld.wolfram.com/Circle-CircleIntersection.html.

[48] Eric W. Weisstein. Circle-Line Intersection. *From MathWorld—A Wolfram Web Resource*, May 2006.
http://mathworld.wolfram.com/Circle-LineIntersection.html.