Viewpoint-Aware Task Planning and Model Predictive Control for Applications in Videography and Multi-Target Tracking

by

Aaron Castagna Ray

Sc.B., Brown University (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
Department of Electrical Engineering and Computer Science
August 27, 2021

Certified by .....
Daniela Rus
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Viewpoint-Aware Task Planning and Model Predictive Control for Applications in Videography and Multi-Target Tracking

by

Aaron Castagna Ray

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

We seek to combine high level planning with low level reactive control to solve a variety of viewpoint-constrained target following tasks. In the scenarios we consider, a team of tracking agents is desired to gain some sort of visual information about one or more target agents. A high level planning algorithm accounts for coarse, global decisions, such as "Which targets should each tracker be responsible for?", or "When should a tracker visit each target?" This level of planning is combinatorial in nature and requires coordination between the tracking agents. We combine this process with a lower-level reactive control accounts for stochastic target motion. By making this controller aware of a viewpoint cost function, the behavior of the tracking agents can be both more performant and easier to deploy on real robots.

Thesis Supervisor: Daniela Rus
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to start by thanking my advisor, Daniela Rus, who has been an inspiration and mentor throughout my time in the Distributed Robotics Lab. Her guidance has been indispensible in learning how to conduct high quality research, and I look forward to our future work together.

I would also like to thank Javier Alonso-Mora for a productive collaboration and helpful guidance on my first paper as a graduate student. His gracious hospitality made it possible for me to spend a month working in Delft as I formulated the project that became the basis of this thesis and enjoy what turned out to be some of the last pre-Covid weeks.

Alyssa Pierson and Hai Zhu also instrumental in helping me define my research direction and implement the ideas behind the thesis. I learned a lot from late night paper writing with Alyssa and frantic transatlantic troubleshooting with Hai.

My time in the Distributed Robotics Lab would not have been the same without a wide-ranging cast of labmates. Wilko answered endless MPC queries, introducing me to the wonderful world of automatic differentiation through Casadi; Cenk is always ready to answer all of my theory questions; Ramin has shown me the big world of little NCPs; Noam is there to toss around ideas about multi-agent MPC and help me with experiments; Tim has shown me the value of meticulous testing and quantitative evaluation; Alexander has taught me how to build large data pipelines for machine learning and patiently assisted with testing days at Devens; Igor gave me key insights into ellipsoid intersections for this thesis and gave me many insights into the nuanced world of academia; Yutong and I have had fruitful discussions about multi-target tracking from the computer-vision perspective; Veevee has kindled my interest in ocean robotics and I always enjoy his fresh stories from the Cape.

I would like to specially thank Lucas and Brandon, whose ever-present support, guidance, and friendship has made the lab a fun place to work. Some of my best mem-

ories at MIT have been casual-dinners-turned-spirited-debate with you that resulted in everyone coming away with new insights.

I would probably would not be at MIT without Izzy. He has always pushed me to perform at my best since we met in a freshman math class neither of us understood, and our wide-ranging discussions about robotics, general intelligence, and philosophy have expanded my horizons.

I would certainly not be where I am today without the unconditional love and support of my parents and grandparents. You have always stressed the importance of working hard but leading a balanced life, and you have shown me the way to both success and happiness.

Finally, I cannot thank Nina enough for her endless love and support. I know it is not easy to live with the chaotic life of a graduate student, but you still manage to carry me through the lows and help me ride the highs. You inspire me to live each day better than the last, and still be home for dinner.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots are increasingly expected to safely operate in dynamic multi-agent environments. For many tasks, such as videography, intrusion detection, and disaster relief, the operational environment includes multiple moving targets of interest which the robot must keep in view while avoiding collisions. These tasks are often solved more efficiently with cooperating *teams* of robots. To this end, we study how a coordinating team of "tracker" agents can effectively maintain desired viewpoints of other "target" agents in the environment.

We investigate how to combine high level planning with low level reactive control to solve a variety of viewpoint-constrained target following tasks. In the scenarios we consider, a team of tracking agents is desired to gain some sort of visual information about one or more target agents. A high level planning algorithm accounts for coarse, global decisions, such as "Which targets should each tracker be responsible for?", or "When should a tracker visit each target?" This level of planning is combinatorial in nature and requires explicit coordination between the tracking agents. A lower-level reactive control drives the tracking agents to the desired view- points and accounts for stochastic target motion. By making this controller aware of a viewpoint cost function, the behavior of the tracking agents can be more aggressive than a decoupled

pipeline where the low level path planning and control are oblivious to the desired viewpoint and visibility constraints. Our primary objective is to show that a variety of viewpoint-constrained target following problems can be solved by co-designing the high and low level control. Advances in this viewpoint-constrained following problem are of interest because they enable a range of new capabilities for robot teams. These capabilities include videography/cinematography, intrusion detection, and animal tracking. While some form of solution to each of these problems exists in the literature, our advancements in making the lower-level controller aware of the viewpoint constraint will lead to better performance and easier transfer to real robots. Much of the work on target following assumes holonomic or first-order systems. Most real robot platforms are more constrained in their motion, so an intermediate controller must make the system act like the idealized version. This prevents using the full dynamics of the robot to the maximum advantage. Making the lower-level control viewpoint aware also leverages recent advances in realtime MPC solvers (and the associated embedded computers they can run on) to make the system more reactive. If the only viewpoint-aware planning is done in a lower-rate path planning module, the robot cannot react to the changing viewpoint constraints as quickly. By incorporating these considerations into the trajectory generated by the low level control, we can adjust the tracker's behavior based on the target's motion more quickly.

One class of problems within this broad scope is following a ground-based moving agent (e.g. robot or human) with an aerial vehicle, subject to constraints such as "keep a certain feature of the agent in the field of view while avoiding environmental obstacles". This problem is challenging because it requires a real time adaptive solution for the local control with global objectives and constraints. Practical and robust solutions will enable new applications such as autonomous drone videography that go beyond today's recording capabilities. Current drone videography can follow an actor. In this thesis we describe a solution that supports finer-grain specifications, such

as "keep the actor's face in the field of view". The solution has to combine real-time local response with global planning to accommodate the presence of obstacles (e.g. avoid bridges).

More specifically in the first part the this thesis, we enable a team of videography drones to autonomously track and capture a sequence of desired shots of a moving target such as a ground robot or a person. Framing a scene for videography is a complicated process that depends on a range of aesthetic preferences of the videographer. However, many of the important framing primitives can be distilled into a small set of parameters that define the camera's desired viewpoint. We would like for a videographer to be able to specify a set of desired shots based on these parameters, and have the team of drones determine the best trajectories for capturing the video of the subject as it moves through the environment.

We present a two-stage viewpoint optimization pipeline that enables a team of drones to capture a series of shots that match desired aesthetic qualities. A high-level planner uses a visibility heuristic and an Integer Linear Program (ILP) optimization routine to choose when each drone should capture which shot. This assignment results in a reference trajectory that can be tracked by an online Model Predictive Control (MPC) algorithm with a cost function based on the specified viewpoint parameters. The controller can locally optimize the drones' trajectories to account for stochastic target motion. We demonstrate a videography scenario with a pair of drones assigned to capture several shots of a remote controlled racecar in the presence of obstacles.

In the second part of the thesis, we draw inspiration from the single-target videography setting and extend the solution to a multi-target viewpoint-aware information gathering task. There are many cases where a team of autonomous agents may need to gather information from a number of targets of interest, and the information gain is dependent of the viewpoint of the target. For example, the identification tag on an elephant seal may only be visible from one side, or a suspicious backpack lost in a

crowd may only be identifiable from its wearer's rear. A team of autonomous agents hoping to make observations from the desired viewpoints must both plan a high level path to decide which order to visit the targets of interest, and find low-level control inputs to drive the tracking agent to the region of interest.

We explore an alternate method for parameterizing obstacles that leads to more scalable operating environments and provide a demonstration in a pair of simulated multi-target tracking environments. This method parameterizes the free space as a collection of ellipsoids instead of each obstacle as en ellipsoid. The ellipsoidal decomposition results in better performance in the presence of many obstacles, and we show how it leads to a simpler implementation compared to other free-space decomposition methods for obstacle avoidance. The decomposition also inspires a novel method for evenly assigning regions for each tracking agent to patrol.

1.1 Contributions

The main contributions of this thesis are:

1. Presenting a novel high-level videography planner based on a visibility heuristic to globally optimize ordering of shots of a dynamic target

2. Implementing a viewpoint-aware receding horizon controller to locally optimize shot aesthetics along the reference trajectory generated by the assignment algorithm

3. Demonstrating of these videography algorithms in hardware experiments with a pair of drones and remote control racecar videography target

4. Developing an easy-to-implement method for enforcing collision avoidance constraints with an MPC in cluttered environments

5. Proposing a new algorithm for evenly dividing responsibility between agents in area monitoring tasks

The videography planner, viewpoint-aware controller, and hardware experiments appear in [1].

1.2 Organization of Thesis

In Chapter 2, we review existing work related to drone videography, multi-robot monitoring tasks, and optimization-based methods for collision-free vehicle motion. Chapter 3 describes the work on single-target drone videography, including problem definition (3.1), assignment of shots to drones (3.2) online viewpoint optimization (3.3), and experimental results on physical robots (3.4). Chapter 4 extends the basic ideas from Chapter 3 to the multi-target context, introducing task assignment and path planning in 4.2, the new environmental decomposition in 4.2.2, and online motion optimization in 4.3. In Chapter 5, we discuss directions for future work and lessons learned.

Chapter 2

Related Work

2.1 Drone Videography and Autonomous Shot Selection

A large body of literature exists related to defining and executing certain shot aesthetics for a videography drone. Existing work can be roughly grouped into two major areas of focus. The first of these areas descends from the computer graphics and animation community and addresses how a camera can move through a scene in an aesthetically-pleasing manner and how to provide a human interface to these algorithms and parameterizations that make it easy for the human operator to achieve desired effects [2, 3, 4].

For example, [4] introduces the Toric Space representation for parameterizing camera state for virtual camera positioning. A camera's state is usually specified by seven parameters (six for the rigid body camera state, and one for the field of view). However, connecting these parameters to desired viewpoint aesthetics, such as on-screen positioning, size, and vantage angle (the azimuth/elevation angles of the camera relative to the target) leads to complex relationships. The Toric Space reduces the camera representation to four parameters (three for a point on a "toric manifold" and one for

the field of view). The reduction of representational complexity is possible because the new space is implicitly tied to the state of target objects in the scene. The paper shows that the viewpoint parameters have simple relationships to the Toric Space and can be manipulated from the image space, a desirable property for interactive viewpoint definition. In contrast to [4], this thesis focuses more on the global planning of which shots to take. We don't use the Toric space parameterization, but it is not fundamentally incompatible with our method. We use a subset of the desired viewpoint constraints and the current target position to reduce the dimensionality of our camera state decision space, which has a more straightforward connection to our global planning process.

Other work related to the viewpoint definition side of the field has focused on algorithmic frameworks for interactively helping directors achieve desirable aesthetic qualities of their shots by learning from prior cinematographic input[5, 6]. A taxonomy of shot types of particular to drone videography is given in [7].

The second distinct area of focus for drone videography deals more directly with the robotics and control aspect. While the first area is still relevant on physical hardware, drones present a particular set of constraints on feasible camera trajectories and placement. Some work in this area has examined the feasibility of dynamic shots [8] and assigning viable sequences of shots [9][10]. Other work studies global trajectory generation that results in smooth, aesthetic trajectories (e.g. [11]), but these methods still require a low-level controller to track the reference trajectory and account for the drone's dynamics. Traditionally these low-level controllers have been task-oblivious controllers like PID or LQR controllers. An increasingly common improvement to the lower-level controllers is to formulate the problem of following the reference path as a constrained nonlinear optimization that includes viewpoint-related objectives in its cost function [12, 13, 14]. The resulting optimization is solved online in a receding horizon fashion and can locally improve the precomputed reference trajectory

in response to a dynamic environment. The drone videography domain has also been explored by the reinforcement learning community, e.g. in [15].

The shot parameterization and drone control system presented in [16] extends the Toric Space concept from [4] and bridges these two halves of the field by developing shot parameterizations particularly suited for cameras on drones and demonstrate an associated controller for the drone. Their work adds a concept of safety distances between the camera and targets, and addresses constraints on realizable camera states (e.g. from a motion-limited gimbal). They plan paths for a team of videography drones by finding shortest-paths on a precomputed visibility-aware roadmap of the environment and fitting smooth reference curves to guide the drones. The work addresses multi-drone deployments by having a primary drone filming the desired shot and other drones providing complementary viewpoints. When changes in the dynamic environment would cause undesirable mutual visibility between drones in the main camera frame, a local replanning of the viewpoint assignments is carried out to optimize the quality of captured shots. In contrast the prior work, we incorporate the viewpoint desiderata directly into the low-level controller, and focus on longer-term combinatorial planning for use in scenarios where we have a prior on target motion and would like the drones to automatically decide when to take shots from a set of desired angles.

We build upon previous work in [13] and focus on the problem of optimizing sequences of shots from multiple cameras and perspectives subject to constraints. In contrast to [13] and other recent work [15][17] that has focused on reactively finding good viewpoints in unstructured scenes, we assume that the operational environment is known and the subject's motion can be estimated ahead of time. In this context, a director may have a set of desired shots that should be captured during the scene. It is nontrivial to decide when each shot should be taken so that the shots are minimally obstructed and the ordering is feasible for the drones. A higher-level global planner

is used to augment the online local planner and ensure the desired viewpoints can all be captured.

2.2 Multi-Robot Target Monitoring

The target/tracker dichotomy in our problem setting is quite similar to the literature on pursuer-evader games. A useful survey of these problems is presented in [18]. Our formulation is most similar is the "cops and robbers" problem, where a number of cops try to corner one or more robbers. The discrete formulation of this problem first proposed by [19] and [20], and there has been much literature on work characterizing what kinds of graphs favor the pursuing or evading agent, and the computational complexity of finding optimal strategies. While [19] showed that three cops is always sufficient to capture a robber on a planar graph, in [21] the general case even finding the smallest number of cops necessary to capture a robber on a given graph is EXPTIME-Complete.

This problem has also been analyzed in continuous (time and/or space) domains for varying numbers of cops and robbers. These results are based on reachability analysis. This often involves solving Hamilton-Jacobi-Isaacs (HJI) equations to find optimal differential play. However, this method is computationally expensive in complex environments with many agents. Notable results include that three cops is sufficient to catch a robber in any planar polygon environment (not necessarily convex) under simplified assumptions on dynamics [22]. With more general dynamics constraints, sophisticated reachability analysis must be employed, e.g. [23]. Other approaches assume unconstrained single-integrator dynamics and show distributed capture guarantees.

For example, [24] introduces a decentralized algorithm that guarantees capture of a group of evader agents by a team of pursuer agents. The algorithm is based on

a Voronoi decomposition of the environment and the control policy ensures that the area in a subset of the evaders' Voronoi cells shrinks at each timestep. An appealing property of the resulting control law is its simplicity – it has no memory and depends instantaneously on the state of the target and tracking agents. A local coordination technique is employed to avoid symmetry traps that would result in target prioritization switching before a target is captured. The algorithm is demonstrated with a team of robots with unicycle dynamics. Gradient-based control laws such as these can be difficult to extend to more complicated vehicle dynamics or obstacle-rich environments. In contrast, our optimization-based algorithms for multi-target tracking support general dynamics models and a wider class of environment representation and complexity at the expenses of giving up a global provable capture guarantee.

In summary, these existing works either simplify dynamic and kinematic constraints in order to provide capture guarantees, or consider the full reachability analysis at great computational cost. In contrast to these approaches, we sacrifice provable capture in order to develop real-time algorithms that are guaranteed to provide *realizable trajectories* for the pursuit agents while still making intelligent (but possibly not optimal) higher-level planning decisions.

2.3 Collision-Free Motion

The implementation of our advancements in videography and target tracking rely on existing methods in optimization-based control. Although we do not make any novel contributions in this area, understanding the considerations that go into choosing a motion planning framework helps to motivate some of our specific design choices. The following categories of control approaches are not mutually exclusive, but they serve as useful guideposts for comparing different motion planning and feedback control paradigms.

Linear optimal control problems with quadratic costs are very well understood and characterized [25], including important results on the design and optimization of various robust controllers [26]. Unfortunately the drone platforms of interest have nonlinear dynamics, and more complex methods that deal directly with the nonlinearities (instead of local linearization and attempting to bound the suboptimality of the resulting controller) are a better fit. A more general theory of optimal control through the lens of dynamic programming allows control of more general nonlinear systems [27], but breaks down for higher-dimensional systems due to the curse of dimensionality.

Sampling-based motion planners like Rapidly-Exploring Random Trees [28] present an elegant solution to nonlinear motion planning through complex configuration spaces, but the large number of samples they require presents challenges in realtime applications often requiring a lower-level controller to track the trajectory generated by the slower RRT algorithm.

Yet another class of motion planning algorithms precomputes a library of feasible trajectories and stabilizing controllers that can be efficiently selected at runtime [29, 30]. These algorithms are very appealing for their low computational cost at runtime and strong safety guarantees, but they are less flexible than finding the trajectory online and the number of trajectories that must be stored grows quickly with the size of the state space and trajectory length.

We choose to employ a different class of algorithms known as Model Predictive Control (MPC). MPC is a receding horizon control framework, where the control objective is optimized over a finite horizon, the best control at the current time is executed, and the whole problem is resolved at the next timestep. For tasks like videography, where motion is merely a means to achieve the underlying objective (a good viewpoint of the target) as opposed an end in and of itself, MPC is particularly appealing as it enables much flexibility in modeling the objective function.

We can encode the viewpoint constraints as objectives of the optimization, and find dynamically feasible motion that enables the good viewing conditions.

As a local optimization method, MPC often needs to be paired with a higher-level planner (possibly a sampling-based kinematic-aware motion planner like RRT, but potentially an even simpler planner that is not even aware of the agent's dynamics).

MPC for constrained linear systems with quadratic costs is very well understood [31] and easy to solve in realtime. The MPC can be transcribed as a constrained linear program via the single shooting (free variables are control inputs), multiple shooting (free variables are states and control inputs), or collocation methods (free variables parameterize a polynomial describing the state). The three methods increase in problem size from single shooting to collocation, but also increase in sparsity. Once a transcription method is chosen, a range of mature algorithms and software are available to solve the linear program. Solving nonlinear MPC also involves choosing between single shooting, multiple shooting, and collocation transcriptions, but there are more options and tradeoffs for solving the resulting optimization.

If an MPC formulation is convex, then a variety of specialized, efficient solvers such as CVX [32] can be employed. In the general nonconvex case, there is a wider range of options. One popular approach is to consider the optimization as a constrained Nonlinear Program (NLP). The resulting NLP problem is usually solved with either a Sequential Quadratic Program (SQP) method (e.g. SNOPT [33]) where the nonlinear functions are locally treated as quadratic, or an Interior Point (IP) method (e.g. IPOPT [34]), where a feasible point of the decision space is iteratively updated to reduce the cost. The SQP method has the advantage that it can make better use of a warm start than the IP method [35]. There is an alternative method for solving the transcribed MPC problem known as the Augmented Lagrangian approach that involves formulating a sequence of unconstrained nonlinear minimization problems. This results in simpler computational steps in calculating the minimum, but a se-

quence of optimizations must be solved for changing values of the penalty parameters and Lagrange multipliers (e.g. [36]). See [37] for a more in depth but approachable discussion about transcription and solution methods.

There are many existing options for realtime MPC solvers. While we will not discuss them exhaustively, we will briefly mention some notable examples. Forces Pro [38] is a paid, proprietary option that is tailored for embedded deployment. Forces Pro uses the SQP solution method. Acado [39], and its successor Acados [40], are free and open source implementations of an SQP method, also meant for embedded deployment. IPOPT [34] is an interior point optimizer with an interface built into the automatic differentiation framework Casadi that can easily be used to specify MPC problems. GRAMPC [41] and OpEn [42] are recent examples of Augmented Lagrangian based MPC libraries.

We choose to use Casadi + IPOPT as our MPC solver, due mostly to ease of implementation and high quality of available documentation and examples. We employ a direct collocation transcription of the NLP to take advantage of IPOPT's good performance on large, sparse problems. However, nothing fundamentally ties the algorithms we present to this solver, and the same algorithms should easily transfer to other options such as Acados.

Chapter 3

Viewpoint-Aware Drone Videography

3.1 Problem Definition

When a predictable subject is operating in a known environment, we would like to be able to coordinate a set of desired shots such that each shot is captured with minimal occlusion. For example, repeated takes on a movie set or a well-trodden mountain bike path are scenarios where the subject's motion is fairly restricted and can be estimated ahead of time, even if the exact path is not known. Existing videography approaches focus on reactive planning to maintain good views of a target. We are interested in global planning that ensures a team of videography drones sequences the set of desired shots to ensure they are all fit in within the allotted time and with minimal obstruction.

3.1.1 Preliminaries

We assume that n_d videography drones operate in an environment with known set of obstacles $\mathcal{O}$. While the MPC formulation we use requires obstacles to be represented as ellipsoids, our main contribution of the higher-level planner supports any obstacle set representation that allows for computing ray intersections efficiently. We also

Table 3.1. Symbols and Conventions Table

$\mathbf{w}$	Lowercase bold letters denote vectors
M	Capital letters denote matrices
$\mathcal{S}$	Calligraphic letters denote sets
$\|\mathbf{w}\|_Q^2$	Weighted vector norm, equivalent to $\mathbf{w}^T Q \mathbf{w}$
n_d	Number of videography drones
$\mathcal{O}$	Set of obstacles in the environment
$\mathbb{Y}$	Probability distribution over target's motion
θ^i	Elevation angle of desired drone viewpoint relative to target, for shot specification i
ψ^i	Azimuth angle of desired drone viewpoint relative to target, for shot specification i
ρ^i	Desired distance between drone and target, for shot specification i
$C_x^i,\ C_y^i$	Desired x and y coordinates of the target in the camera's image coordinate system
$t_0^i,\ t_f^i$	Initial and final times during which shot i can be taken
τ^i	Duration of shot i

assume access to a probability distribution over the videography subject's future trajectory, denoted $\mathbb{Y}$. We sample from this distribution to predict good times for shots to be taken.

3.1.2 Defining Desired Shots

Much of the defining aesthetic of a camera framing is determined by:

1. **Orientation** of the camera relative to the film subject, denoted by azimuth and elevation, (ψ, θ),

2. **Subject's Distance** to the camera, ρ,

3. **Subject's Position** in the frame, (C_x, C_y).

as presented in [13]. We also note that it is usually desirable for the videography subject not to be occluded in the image.

In addition to the spatial viewpoint parameters, we consider a time window within which a shot should be taken, $[t_0, t_f]$, and a desired duration τ^i.

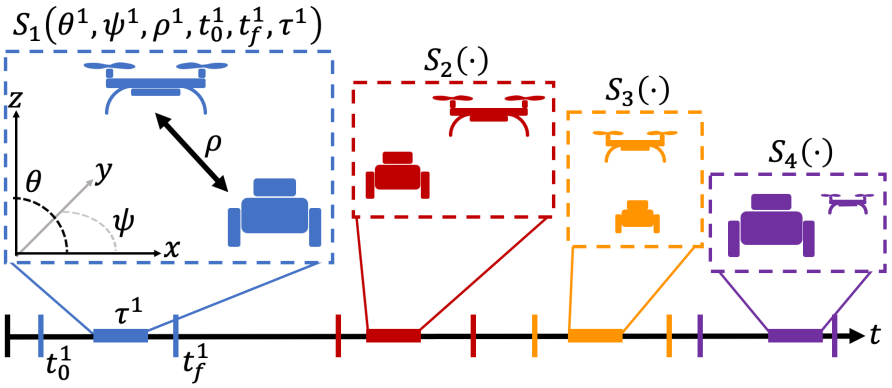


Figure 3-1. Illustration of the shot schematic. Within each shot, we optimize based on viewpoint costs. We also optimize the shot assignment for the sequence of shots between drones.

Definition 1 (Shot Specification)**.** *A shot specification* $S_i = (\theta^i, \psi^i, \rho^i, C_x^i, C_y^i, t_0^i, t_f^i, \tau^i)$ *is set of viewpoint parameters and timing constraints define each shot. We denote the set of desired shot specifications* $\mathcal{S}$*.*

Figure 3-1 illustrates these parameters. The viewpoint properties depend on the drone state $\mathbf{x}_b$, target state $\mathbf{x}_t$, and obstacles in the environment $\mathcal{O}$. We briefly define a cost function associated with each parameter, and refer to [13] for derivation. Let $\mathbf{r}_{ct}$ and $\mathbf{r}_{ct}^c$ denote the vector from the camera origin to the target expressed in the world and camera reference frames respectively. Let $\mathbf{r}_d^c$ denote the desired vector from the camera origin to target as defined by image position parameters C_x and C_y. Let $\boldsymbol{\alpha}_d$ represent the unit vector corresponding to the desired angles (θ, ψ) for the drone relative to the target. With these definitions, we can define cost functions that penalize a drone's state as it deviates from the desired viewpoint.

Let c_{image} denote the cost for the target's position in the image:

$$c_{image} = \left\| \frac{\mathbf{r}_d^c}{\|\mathbf{r}_d^c\|} - \hat{\mathbf{r}}_{ct}^c \right\|_{Q_i}.$$

Note that while c_{image} is minimized when the target is centered at the desired location of the image, this cost function is slightly different than the actual displacement between the actual and desired location of the target in the image plane. Calculating

the projection of the target into the image plane requires rescaling homogeneous coordinates onto the image plane, and the the resulting $1/z$ operation results in unstable performance when numerically optimizing. Instead, we use the difference in direction vectors between the camera and the target in calculating the viewpoint cost, which is quite close to the desired cost when the target appears close to the desired location.

The cost c_{scale} penalizes distance deviation from the target,

$$c_{scale} = \left\| \|\mathbf{r}_{ct}\| - \rho \right\|_{Q_s},$$

and c_{angle} is a cost that depends on the deviation of the drone's angles relative to the target:

$$c_{angle} = \left\| -\frac{\mathbf{r}_{ct}}{\|\mathbf{r}_{ct}\|} - \frac{\alpha_d}{\|\alpha_d\|} \right\|_{Q_a}.$$

Finally, we consider a cost that penalizes drone positions that have an obstructed view of the target. We consider ellipsoidal obstacles defined as

$$O(M, \mathbf{r}_o) = \{\mathbf{v} \mid Mv + \mathbf{r}_o, \|\mathbf{v}\| \leq 1\}$$

for some positive definite shape matrix M. The cost of occlusion from an ellipsoid is most straightforward when we consider a coordinate system centered at the ellipsoid, and a transformation of the space such that the ellipsoid is a unit sphere. Thus we define

$$\begin{aligned}
\mathbf{r}_c^o &= -\mathbf{r}_{co}^o = M^{-1}(\mathbf{r}_c - \mathbf{r}_o), \\
\mathbf{r}_t^o &= M^{-1}(\mathbf{r}_t - \mathbf{r}_o), \\
\mathbf{r}_{ct}^o &= \mathbf{r}_t^o - \mathbf{r}_c^o, \\
r_{proj} &= \frac{\mathbf{r}_{ct}^{oT}\mathbf{r}_{co}^o}{\|\mathbf{r}_{co}^o\|}.
\end{aligned}$$

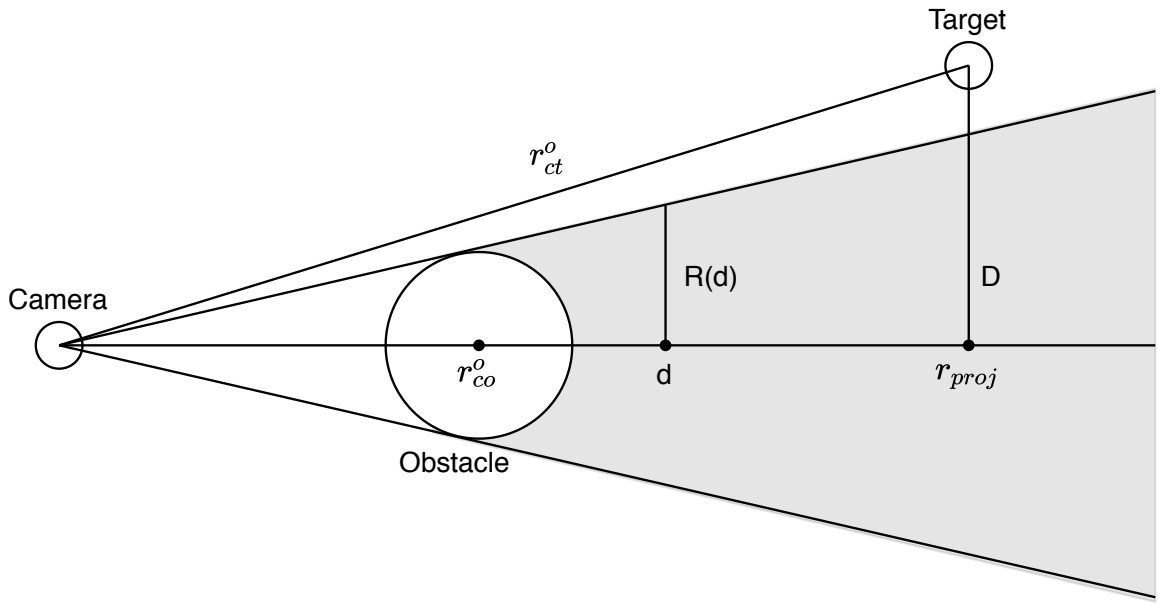


Figure 3-2. The gray region represents the area that is included in the camera image by an obstacle. It is the frustrum of a cone, with a medial axis along r^o_{co} and radius $R(d)$ where d is the distance along the medial axis from the camera. A target is visible from the camera if its distance perpendicular to the medial axis is larger than the radius at that point.

As shown in Figure 3-2, the target is visible from the camera when the target is outside of the occlusion cone. Let $R(d)$ be the radius of the cone at a distance d from the camera along the occlusion cone's medial axis. The target's perpendicular distance to the medial axis must be larger that $R(r_{proj})$. We denote this perpendicular distance D. The "occlusion distance" d_v, which represents how far a target is inside the occlusion cone, can be represented as

$$d_v = R - D = \frac{\mathbf{r}_{ct}^T r_{co}}{\sqrt{\|\mathbf{r}_{co}\|^4 - \|\mathbf{r}_{co}\|^2}} - \sqrt{\|\mathbf{r}_{ct}\|^2 - \frac{(\mathbf{r}_{ct}^T \mathbf{r}_{co})^2}{\|\mathbf{r}_{co}\|^2}}$$

$$c_{occlusion} = \begin{cases} \max(0, d_v)^2, & \mathbf{r}_{ct}^T \mathbf{r}_{co} \geq \|\mathbf{r}_{co}\|^2 - 1 \\ 0, & \text{otherwise} \end{cases}.$$

The derivation of these relationships is presented in detail in Appendix A.

3.1.3 Defining Desired Sequences

These costs c aid in quantifying the quality of a single shot. Our viewpoint objective is to minimize these costs over a sequence of shots.

Definition 2 (Shot Sequence)**.** *A shot sequence $A_i = (S_j^{t_a}, S_k^{t_b}, \ldots)$ is the set of all shots assigned to drone i. Each assigned shot $S_j^{t_a}$ corresponds to a shot specification S_j and a time when the drone should be capturing that shot, t_a. A_i^j denotes the j-th shot assigned to drone i. The set of all assigned shots across all drones and times is $\mathcal{A} = \cup_i A_i$.*

For each shot, we define a vector of cost functions $\mathbf{J}_i^j$ that penalizes drone i's state from deviating from the parameters defined by shot S_j. $\mathbf{J}_i^j$ contains penalties for deviating from the desired location of the target in the image plane, its size in the image plane, and the camera position relative to the target. $\mathbf{J}_i^j$ also penalizes drone states that have an obstructed view of the target. Intuitively,

$$\mathbf{J}_i^j = [c^i_{\text{image}}, c^i_{\text{scale}}, c^i_{\text{angle}}, c^i_{\text{occlusion}}]^T. \tag{3.1}$$

Given the trajectory of the videography target, $\mathbf{J}_i^j$ is a function of drone i's state and time. For a given set of shots $\mathcal{S}$, our goal is to then find the trajectories for all drones such that $\mathbf{J}$ is minimized over all shots.

Problem 1. *For a desired set of shots $\mathcal{S}$ and a target trajectory Y, we define the total videography cost L as a function of the trajectories of the n_d drones:*

$$L(\mathbf{x}_1, \ldots, \mathbf{x}_{n_d}) = \sum_{j=0}^{|S|} \min_{t_0^j \le t \le t_f^j} \min_i \int_t^{t+\tau^j} \left\| \mathbf{J}_i^j(\mathbf{x}_i(t), t) \right\|_{Q_x} dt. \tag{3.2}$$

In general, Y is stochastic, and by extension $\mathbf{J}_i^j$ is as well. We seek a control policy for each drone that will minimize the expected videography cost L.

We use a two stage approach to find good policies for minimizing L. First, a high-level planning algorithm uses a simple heuristic to predict good times for capturing each shot. A discrete optimization algorithm assigns each drone to a sequence of shots that minimizes the heuristic cost. Next, the optimized assignment and expected target trajectory are used to generate a reference trajectory for a videographic Model Predictive Control (MPC) algorithm that locally optimizes the aesthetic parameters online.

3.2 Shot Assignment

The high-level shot planning is carried out by a centralized algorithm that finds a sequence of shots for each drone to capture, as well as a reference trajectory to follow. The algorithm requires the desired set of shots $\mathcal{S}$, a distribution over target trajectories $\mathbb{Y}$, and a set of obstacles in the environment $\mathcal{O}$. It relies on a heuristic to choose a low-cost reference trajectory that guarantees visibility of the target. The heuristic provides an estimate of the best times to start capturing each shot. We sample the lowest-cost times for each shot and assign each drone a sequence of shots and times based on an ILP minimization.

3.2.1 Assignment Heuristic

We seek a simple heuristic for each shot i that maps time to a reference position, while ensuring that the target is visible. We refer to the position given by this heuristic as $\hat{\mathbf{x}}^i(t)$, and $H^i(t)$ as the associated cost. By focusing on a single cost and position at each time, the problem of optimizing shot cost over all possible trajectories simplifies to choosing a sequence of shots and times for each drone.

The target is visible from the drone if and only if a ray cast from the target to the drone hits no obstacles before reaching the drone. This observation inspires a

heuristic for the reference shot position — cast a ray from the target's position $\mathbf{x}_t$ in the desired shot direction, stopping at the desired shot distance or the intersection with an obstacle, whichever is first. Let $\mathbf{x}^*(t) = \mathbf{x}_t(t) + \rho^i \boldsymbol{\alpha}$ and $\hat{\mathbf{x}}(t) = \mathbf{x}_t(t) + d\boldsymbol{\alpha}$ where

$$\boldsymbol{\alpha} = [\cos\theta\cos(\psi + \psi_t), \cos\theta\sin(\psi + \psi_t), \sin\theta]^T,$$

and d is the smaller of ρ_i and the closest obstacle intersection along $\boldsymbol{\alpha}$. Intuitively, $\mathbf{x}^*$ is the drone position that would have zero viewpoint cost, and $\hat{\mathbf{x}}$ is as close as we can get to that position along the desired direction from the target, before hitting an obstacle. We similarly define $\tilde{\mathbf{x}}(t)$ as a ray cast from $\mathbb{E}(\mathbf{x}(t))$ in the expected direction of $\boldsymbol{\alpha}$. The path $\tilde{\mathbf{x}}$ will be used as the reference path to transition between shots. As the reference position is constructed such that it satisfies the desired relative angles ψ and θ, c_{scale} is the only nonzero viewpoint cost. The reference trajectory is also guaranteed to have visibility of the target. For shot i, we consider a heuristic videography cost H^i_{vid} that is a function of only time:

$$H^i_{vid}(t) = \|\mathbf{x}^*(t) - \hat{\mathbf{x}}(t)\|.$$

The position heuristic $\hat{\mathbf{x}}$ does not enforce continuity of the trajectory, so we add an additional discontinuity cost H_{dis} for each shot:

$$H^i_{dis} = q_{dis} \left\| \frac{d\hat{\mathbf{x}}}{dt} \right\|^2,$$

where q_{dis} is a tuning parameter that adjusts the importance of ensuring the reference trajectory can be perfectly tracked.

Together, H_{vid} and H_{dis} sum to the heuristic for instantaneous cost, H_{fine}. Each shot S_i lasts for a duration τ^i. We define a shot cost H^i_{shot} that estimates the cost incurred by capturing shot S_i starting at time t by taking a forward-looking rolling

average of length τ along H_{fine}. We calculate the expectation over the target's trajectory distribution by sampling.

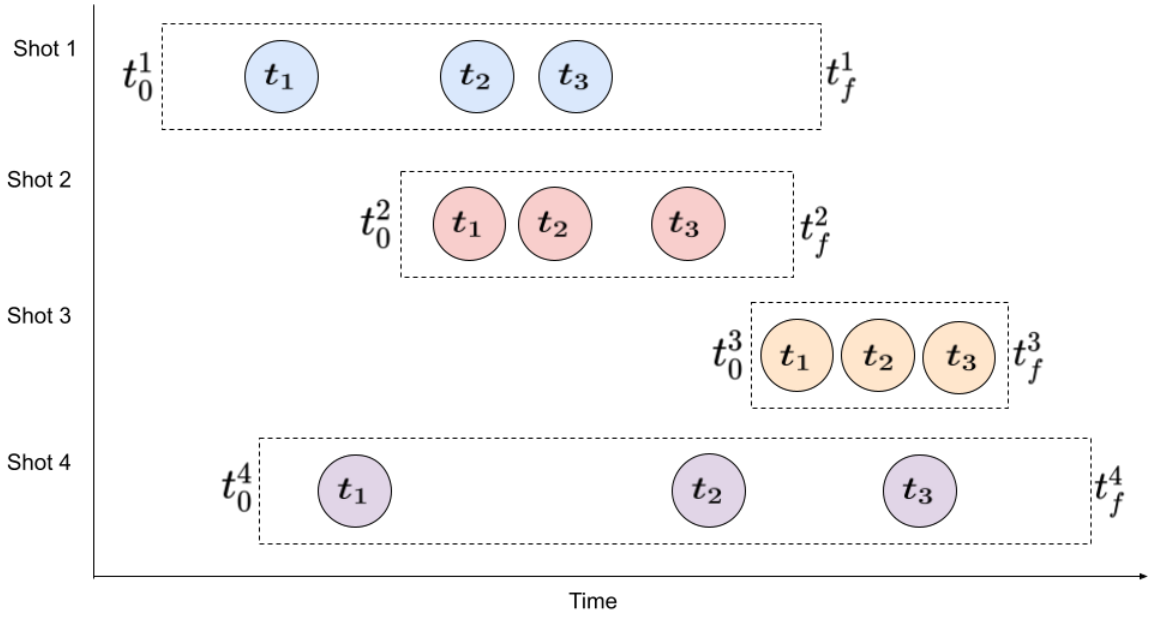
We can now use this heuristic H_{shot} to find good times to capture each shot by finding an assignment of shots to drones such that the sum of H_{shot} over all of the chosen shot times is minimized and each drone can be expected to feasibly capture all of its assigned shots. The size of the optimization problem depends on how finely we sample H_{shot}. We reduce the number of times considered for starting each shot by sampling p random time indices $\mathbf{t}^i_{sampled}$ for each shot, biasing the samples toward lower-cost times. The sampling process greatly reduces the size of the problem.

3.2.2 Choosing the Best Shots

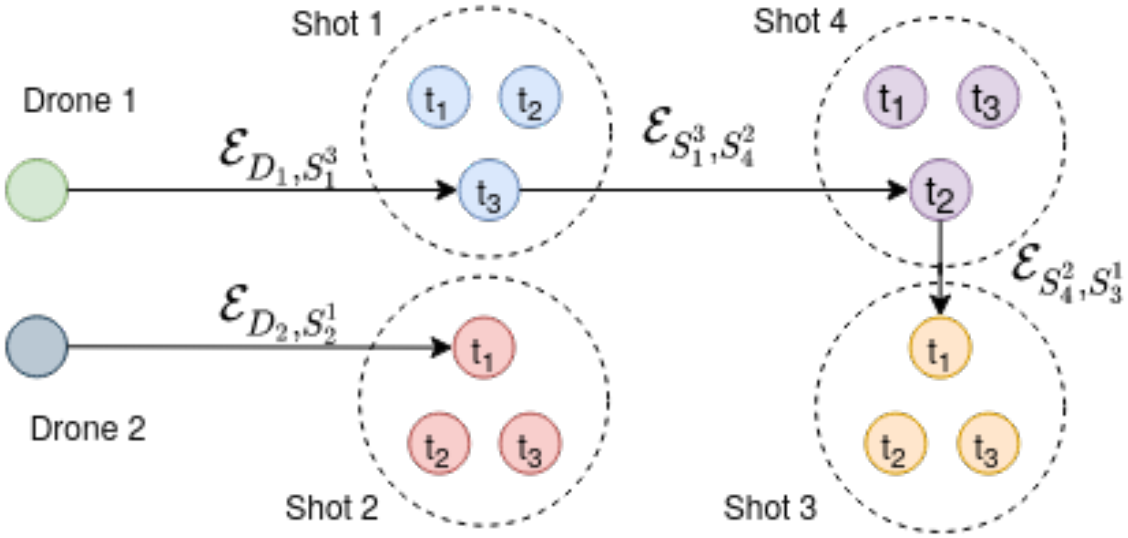
An assignment of drones to shots can now be computed by finding an assignment that minimizes the total heuristic cost incurred. Recall from Definition 2 that $S^{t_a}_j$ denotes the shot j that has been assigned to begin at time t_a, A_i denotes the sequence of shots that have been assigned to drone i, and $\mathcal{A} = \cup_i A_i$ is the set of all assigned shots and times. We must search for an assignment $\mathcal{A}$ that minimizes the total sum of shot costs, while ensuring that each drone has enough time to transition between the shots it has been assigned.

The distance between shots is not deterministic, as it depends on the target's stochastic trajectory. Moreover, even if the drone cannot fully complete the transition between shots in the allotted time, its actual position at the beginning of the next shot may be as good as the desired reference position $\hat{\mathbf{x}}(t)$, especially if it can get close to $\hat{\mathbf{x}}$. Instead of hard transition feasibility constraints, we introduce a cost function that penalizes transitions that may be dynamically infeasible. For a pair of shot assignments $S^{t_a}_j, S^{t_b}_k$, the transition cost H_{trans} is defined as

$$H_{trans}(S^{t_a}_j, S^{t_b}_k) = \mathbb{E}(\left\| [\max(0, d_{trans} - d_{max}), d_{trans}]^T \right\|_{Q_t}), \tag{3.3}$$



(a) An example of shots four shots, S_1, S_2, S_3, S_4. Three times with low heuristic cost H_{shot} have been sampled for each shot, as discussed in Section 3.2.1. In reality, many more times are sampled. Our experiments use 20 samples per shot.



(b) An example of a solution to the PDAG-Minimum-Paths problem for $m = 2$, where the edges not on a solution path have been omitted. Note that each color set is on exactly one path, and there are two paths. This diagram also exemplifies how a minimization in the form of (3.4) maps to an instance of PDAG-Minimum-Paths. The edge labels $\mathcal{E}$ demonstrate the semantics of the construction of $\mathcal{G}_{shot}$. Note that the three times chosen for Drone 1's assignment are in ascending order in (a).

Figure 3-3. An illustrative example of how (a) a set of desired shots are considered at discrete times and (b) turned into a graphical optimization problem.

where d_{max} is the maximum distance the drone can travel between time t_a and t_b and d_{trans} is the distance required to transition between the end of shot $S_j^{t_a}$ and the beginning of shot $S_k^{t_b}$. The weighting Q_t is set such that H_{trans} incurs a very large penalty when the transition distance between shots is expected to be too long (i.e. $d_{trans} > d_{max}$), and a smaller penalty based on d_{trans} which encourages assignments that result in the drones traveling shorter distances.

In addition to this cost term, we require that if one shot precedes another in the assignment ordering, the first shot must end before the second begins. We define slightly different notation for the heuristic cost H_{shot} to refer to the cost of an assignment rather than the cost of a shot. For assignment $A_i^j = S_k^{t_a}$, we notate the assignment cost as $H(A_i^j) = H^k(t_a)$. We want to solve the following minimization:

$$\mathcal{A}^* = \underset{\mathcal{A}}{\operatorname{argmin}} \sum_{i=1}^{n_d} \left(\sum_{j=1}^{|A_i|} H_{shot}(A_i^j) + \sum_{j=2}^{|A_i|} H_{trans}(A_i^{j-1}, A_i^j) \right) \tag{3.4a}$$

$$\text{subject to} \quad \forall s \in S, \quad s \in \mathcal{A}, \tag{3.4b}$$

$$|S| = |\mathcal{A}|, \tag{3.4c}$$

$$\forall S_j^t \in A, \quad t \in \mathbf{t}_{sampled}^j, \tag{3.4d}$$

$$\forall (S_j^{t_a}, S_k^{t_b}) \in A_i, \quad t_b \geq t_a + \tau^j. \tag{3.4e}$$

Constraints (3.4b) and (3.4c) ensure that each shot appears exactly once in the ordering. Constraints (3.4d) and (3.4e) force the chosen times to be from $\mathbf{t}_{sampled}$ and satisfy the temporal consistency previously discussed.

3.2.3 Constructing a Graph Formulation

To find a minimizing solution to (3.4), we draw inspiration from minimum-cost flow assignment methods [43]. While this assignment problem cannot actually be solved

with minimum-cost flow solvers, encoding it in a graph yields a straightforward interpretation as a constrained minimum-cost path problem in a Partitioned Directed Acyclic Graph (PDAG), which can be solved with an ILP.

Definition 3 (PDAG)**.** *A Partitioned Directed Acyclic Graph (PDAG) is a Directed Acyclic Graph $\mathcal{G}$, such that each vertex is assigned one of k labels.*

For purpose of illustration, we will refer to these labels of the PDAG by k different colors. We now define Problem 2 on finding a set of minimum-cost paths through the PDAG.

Problem 2 (PDAG-Minimum-Paths)**.** *Consider a weighted PDAG, $\mathcal{G}$. Given a maximum number of paths m, PDAG-Minimum-Paths(m, $\mathcal{G}$) is the problem of finding the lowest-weight set of paths $\mathcal{P}$ such that $|\mathcal{P}| \leq m$ and exactly one vertex of each color is on some path or determining that there is no such satisfying $\mathcal{P}$.*

Lemma 1. *PDAG-Min-Paths is NP-Hard*

Proof. Consider a PDAG $\mathcal{G}$ with all edge weights set to zero. PDAG-Min-Paths becomes the problem of finding any set of at most m paths such that at least one vertex of each color is on a path. We call this simpler version PDAG-Sat-Paths. If PDAG-Sat-Paths is NP-Hard, then PDAG-Min-Paths is as well. We show PDAG-Sat-Paths is NP-Hard by reducing from HAMPATH, the Hamiltonian path finding problem.

Consider a graph $\mathcal{G}_1 = (V_1, \mathcal{E})$. We construct a PDAG $\mathcal{G}_2 = (V_2, \mathcal{E}_2)$ that has $|V|^2$ nodes, labeled $|V|_2^{i,j}, 1 \leq i, j \leq |V|$.

We define the edges in $\mathcal{G}_2$ as

$$(V^{i,j}, V^{k,j+1}) \in \mathcal{E}_2 \iff (V_i, V_k) \in \mathcal{E}_1.$$

All vertices in each row share a color, for a total of $|V|$ colors. Any path in $\mathcal{G}_2$ can

be converted to a path in $\mathcal{G}_1$ with the mapping $V_2^{i,j} \rightarrow V_1^i$. A solution to PDAG-Sat-Paths, with the number of paths restricted to 1, results in a path that visits every vertex exactly once. This shows that an algorithm for solving PDAG-Sat-Paths can also solve the Hamiltonian path problem, so PDAG-Sat-Paths, and by extension PDAG-Min-Paths, is NP-Hard. □

We can solve PDAG-Minimum-Paths with an Integer Linear Program (ILP). Consider a weighted PDAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. To solve PDAG-Minimum-Paths$(n_d, \mathcal{G})$, we first construct a modified graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ which is the same as $\mathcal{G}$, but it contains an extra n_d vertices. Each of the extra vertices is connected to all of the original vertices, and they each have a unique color. These additional vertices are denoted $\hat{\mathcal{V}}$.[1]

We construct an ILP that contains a variable x_i for each edge $e_i \in \mathcal{E}'$. The cost of each variable C_x is the same as the cost of its corresponding edge. Let $\hat{\mathcal{X}}$ denote the set of variables corresponding to edges connected to $\hat{\mathcal{V}}$. Let $\mathcal{X}_{*,i}$ denote the set of variables corresponding to edges incident on V_i. Let $\mathcal{X}_{i,*}$ denote the set of variables corresponding to edges directing away from V_i. Let $\mathcal{X}^k$ denote the set of variables corresponding to edges incident on color k. The ILP solving PDAG-Minimum-Paths$(D, \mathcal{G})$ can be defined as:

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmin}} \sum_{x \in \mathcal{X}} x C_x \tag{3.5a}$$

$$\text{s.t.} \quad \forall x \in \mathcal{X}, x \in \{0, 1\}, \tag{3.5b}$$

$$\forall k, \ \sum\nolimits_{i=1}^{|\mathcal{X}^k|} \sum\nolimits_{x \in \mathcal{X}^k_{*,i}} x = 1, \tag{3.5c}$$

$$\forall i, \ \sum\nolimits_{x \in \mathcal{X}_{i,*}} x \leq \sum\nolimits_{x \in \mathcal{X}_{*,i}} x \tag{3.5d}$$

$$\forall i, \ \sum\nolimits_{x \in \hat{\mathcal{X}}_{i,*}} x \leq 1. \tag{3.5e}$$

The edges corresponding to variables with a value of one in $\mathcal{X}^* \backslash \hat{\mathcal{X}}$ are considered to

[1]Note that in the case of the drone assignment problem, the "drone" vertices that were added can be considered as $\hat{\mathcal{V}}$.

be the set of edges in the solution to PDAG-Minimum-Paths. If there is no satisfying assignment for $\mathcal{X}^*$, then there is no solution to PDAG-Minimum-Paths.

Proposition 1. *The Integer Linear Program in* (3.5) *generates a solution PDAG-Minimum-Paths.*

Proof. Constraint (3.5c) ensures that the total number of edges incident on a color set is equal to one. This satisfies that all colors are part of some path. Constraints (3.5d) and (3.5e) restrict the solution set to contain at most D paths, by limiting the number of selected edges attached to $\hat{\mathcal{V}}$ to D and requiring that all other edges start at a node with an incident edge. Thus, every solution of (3.5) encodes a valid set of paths in PDAG-Minimum-Paths. By the construction of (3.5), every PDAG-Minimum-Paths problem can be represented by the ILP. Thus, (3.5) solves PDAG-Minimum-Paths. □

Corollary 1. *PDAG-Minimum-Paths is NP-Complete*

Proof. Lemma 1 demonstrated that PDAG-Minimum-Paths is NP-Hard. As shown in Proposition 1, PDAG-Minimum-Paths can be solved with an ILP, and is therefore in NP. Thus, PDAG-Minimum-Paths is NP-Complete. □

3.2.4 Drone Assignment as PDAG-Minimum-Paths

Figure 3-3 demonstrates how a solution to a PDAG-Minimum-Paths problem with $m = 2$ encodes the shot assignment problem. By design, a path must visit each of the graph partitions, which means that it passes through every color label of the graph. We now discuss how to construct a PDAG that encodes the constraints of (3.4).

Note that (3.4e) implies that the assignment of shots must satisfy a strict partial ordering that ensures the sequence is monotonically increasing in time and has no shot overlaps.

We use this partial order to generate a DAG $\mathcal{G}_{shot}$, where vertices represent each S_i^t, for $t \in \mathbf{t}_{sampled}^i$. This selection of vertices enforces (3.4d). We choose the color label of each vertex $S_i^t \in \mathcal{G}_{shot}$ to be its shot index i. We also add a vertex in $\mathcal{G}_{shot}$ for each of n_d drones that can be assigned a shot. These drone vertices are connected to all existing vertices in $\mathcal{G}_{shot}$ and each have a unique color. The directed edge between $S_j^{t_a}$ and $S_k^{t_b}$ (if it exists) is denoted $\mathcal{E}_{S_j^{t_a} S_k^{t_b}}$ with cost given by

$$H(\mathcal{E}_{S_j^{t_a} S_k^{t_b}}) = H_{shot}(S_k^{t_b}) + H_{trans}(S_j^{t_a}, S_k^{t_b}). \tag{3.6}$$

The cost for edges connected to the drone nodes is denoted

$$H(\mathcal{E}_{D_i, S_k^{t_b}}) = H_{shot}(S_k^{t_b}). \tag{3.7}$$

We now present Proposition 2, which connects $\mathcal{G}_{shot}$ to (3.4).

Proposition 2.

Solutions to PDAG-Minimum-Paths(n_d, G_{shot}) minimize (3.4).

Proof. We will show that the constraints and costs in (3.4) are equivalent to the constraints and costs of solutions to PDAG-Minimum-Paths(n_d, G_{shot}) (PMP). Let each shot $s \in \mathcal{S}$ correspond to a color in G_{shot}. If we consider the set of assigned shots $\mathcal{A}$ as corresponding to the set of vertices in G_{shot} that are on paths selected by PMP, then enforcing each shot $s \in S$ to appear in the assignment A as in (3.4b) is equivalent to enforcing that every color appears on the solution to PMP. Enforcing $|\mathcal{S}| = |\mathcal{A}|$ as in (3.4c) is then equivalent to restricting each color to appear at most once on the PMP solution. The vertices of G_{shot} are chosen by construction to respect constraint (3.4d). The connectivity of G_{shot} was defined by the partial ordering implied by (3.4e), so every solution to PMP respect this ordering constraint. Each path in the solution of PMP corresponds to the assignment ordering A_i for a drone. The cost of the path

consists of the shot cost connected to each vertex and the transition cost connected to each edge, as in (3.6) and (3.7), the same expressions used to express the costs of assignments A_i in (3.4a). As the constraints and costs are equivalent, the solution to PDAG-Minimum-Paths(n_d, G_{shot}) also minimizes (3.4). □

While we now have a method for solving the assignment satisfaction and cost minimization from (3.4), there remains the question of whether this method is the most efficient possible. We answer this in the affirmative by showing that the decision form of (3.4) (i.e. finding a satisfying assignment, regardless of cost) is NP-Hard. As a result, we do not expect to find a polynomial-time solution to the assignment problem, and the ILP formulation of solving the assignment problem is indeed necessary for solving it.

A seemingly straightforward way of proving the complexity of (3.4) would be to provide a reduction from PMP. However, there is a subtlety that prevents this reduction from being as simple as it might appear. While it is easy to frame any assignment problem in the form of (3.4) as a PMP problem by constructing a DAG based on the temporal ordering of the intervals, the converse is not true. The temporal constraints of any set of intervals can be represented as a DAG, but not every DAG can be represented by a set of intervals satisfying the same partial ordering. Figure 3-4 demonstrates a simple counterexample. Instead, we provide a reduction directly from 3-SAT to the assignment problem to prove its NP-Completeness.

Lemma 2. *The decision form of* (3.4) *is NP-Complete.*

Proof. We provide a reduction from 3-SAT to (3.4). In the videography context, the intervals are partitioned by "shots." Here, as with the PMP problem, we denote each partition as a separate "color" for the interval. The reduction assigns each 3-SAT clause a distinct color. For each clause, a separate interval is generated corresponding to each variable in the clause. Each variable and its negation also receive intervals

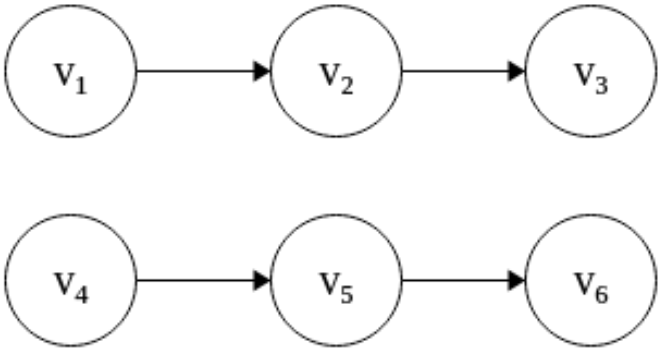


Figure 3-4. Example of a DAG where the implied partial ordering is not consistent with any sequence of intervals. In order for v_4 to be incomparable to v_1, v_2, and v_3, it must overlap with all three of them. However, intervals corresponding to v_5 and v_6 begin after v_4, and therefore cannot overlap with v_1 and v_2 and should have connections to those vertices.

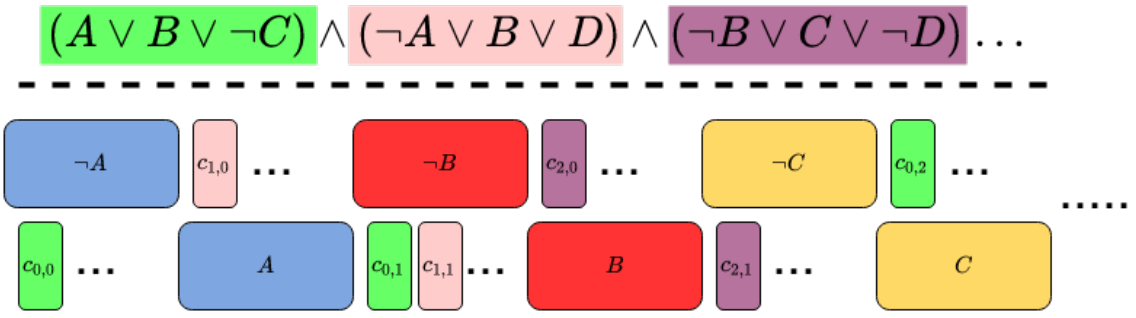


Figure 3-5. Example of how a 3-SAT instance can be represented in the same form of (3.4). Each SAT clause is associated with a different color (i.e. shot index). The constraint that every color must be chosen ensures that every clause must contain a variable that evaluates to True, and that each variable must be assigned a value. The requirement against temporal overlap of the chosen intervals ensures the clauses that evaluate to True are consistent with their constituent variables.

that share a color. A specific placement of these variable intervals and clause intervals encodes the 3-SAT instance. Figure 3-5 provides a visual example of the following development of the problem encoding.

Consider a 3-SAT instance containing m variables $v_0, \ldots, v_{m-1}$ and n clauses $c_0, \ldots, c_{n-1}$. For each variable v_i, we construct intervals $I_{v_i} = [2 \cdot i, 2 \cdot i + 1)$ and $I_{\neg v_i} = [2 \cdot i + 1, 2 \cdot (i + 1))$. Note that all of these intervals are disjoint. Consider each clause $c_i = (x_j \vee x_k \vee x_l)$, where x_j is v_j or $\neg v_j$. If $x_j = v_j$, then we construct an associated interval $I_{c_{i,1}}$ that is a subset of I_{v_j} and disjoint from all other intervals that occur during I_{v_j}. Likewise, if $x_j = \neg v_j$, then $I_{c_{i,1}}$ is constructed as a subset of $I_{\neg v_j}$. The same process is repeated for x_k and x_l, resulting in three intervals $I_{c_{i,1}}, I_{c_{i,2}}, I_{c_{i,3}}$ associated with clause c_i. These three intervals are considered to have the same color. The same process repeats for each clause.

The resulting assignment problem involves choosing $3n + m$ non-overlapping intervals such that each color is chosen once. The solution to this assignment problem solves the original 3-SAT instance. For each v_i, we let $v_i = True$ if I_{v_i} was chosen and $v_i = False$ if $I_{\neg v_i}$ was chosen. For each clause c_i, at least one clause interval $I_{c_{i,j}}$ must have be chosen because of the clause color constraint. By construction, choosing this clause interval is mutually exclusive with choosing a variable interval (i.e. $c_{i,j}$) that does not make the clause true. Thus, the variable intervals that are chosen are forced to be consistent with the clause intervals that are chosen. Thus, if there exists a satisfying assignment for the interval assignment problem, then the 3-SAT instance is satisfiable. If there is no satisfying assignment for the interval assignment problem, then there is no way to consistently assign the SAT variables such that every clause is satisfied, so the original 3-SAT problem is unsatisfiable. Thus, a solution algorithm for (3.4) would solve 3-SAT, and the problem is NP-Hard. As we have already shown that it is solvable with an ILP, it is also NP-Complete □

The full high-level shot assignment pipeline is summarized in Algorithm 1. For

each shot, the heuristic cost H_{shot} is computed (line 3), and low-cost times t_{sampled} are sampled (line 4). Next, the transition cost (3.3) for each potential transition is computed (line 6). Finally, the shot assignments are computed by formulating the PDAG-Minimum-Paths Integer Linear program (3.5).

Algorithm 1 Videography Assignment

1: **procedure** AssignShots($\mathcal{S}, \mathbb{Y}, \mathcal{O}$)
2: **for** $i = 1 : |\mathcal{S}|$ **do**
3: $H^i_{\text{shot}} \leftarrow \text{CalculateShotHeuristic}(\mathbb{Y}, \mathcal{O}, S_i)$
4: $\mathbf{t}^i_{\text{sampled}} \leftarrow \text{ImportanceSample}(H^i_{\text{shot}})$
5: **for each** $(S^{t_a}_i, S^{t_b}_j) \in \text{PotentialShotTransitions}$ **do**
6: $H_{\text{trans}}(S^{t_a}_i, S^{t_b}_j) \leftarrow \mathbb{E}(\text{TransitCost}(S^{t_a}_i, S^{t_b}_k, \mathbb{Y}))$
7: $\mathcal{A}^* \leftarrow \text{IntegerProgram}(t_{\text{sample}}, H_{\text{sample}}, H_{\text{trans}}, \mathcal{S})$
8: **return** $\mathcal{A}^*$

3.3 Online Viewpoint Optimization

The assignment $\mathcal{A}^*$ from (3.4) is used to guide the online MPC. For each shot assignment $S^{t_a}_j \in \mathcal{A}^*_i$, drone i will attempt to minimize the viewpoint cost for shot j in the range $[t_a, t_a + \tau^j]$. In the intermediate time between successive assignments $S^{t_a}_j$ and $S^{t_b}_k$, the MPC follows a reference path that leads it from the expected end of one shot to the expected beginning of another, $\tilde{\mathbf{x}}(t_a + \tau^j)$ to $\tilde{\mathbf{x}}(t_b)$. We precompute a Probabilistic Roadmap (PRM)[44] of free space in the environment, and use it to compute the reference path between $\tilde{\mathbf{x}}(t_a + \tau^j)$ and $\tilde{\mathbf{x}}(t_b)$. During each shot period, the MPC locally optimizes the viewpoint costs. The MPC solves for drone trajectories and gimbal controls in real-time separately for each drone. The rest of the MPC formulation is materially the same as the method presented in [13], but we summarize it here for completeness. For speed considerations and analysis of the nonlinear videography MPC performance, we refer to [13].

3.3.1 Drone and Camera Model

We assume the drone's motion model follows some nonlinear differential equation $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, with $\mathbf{x}(0) = \mathbf{x}_0$ and $\mathbf{x}^k \in \mathcal{X} \subset \mathbb{R}^{n_x}$ denotes the state of the drone and $\mathbf{u}^k \in \mathcal{U} \subset \mathbb{R}^{n_u}$ the control inputs at time step k. $\mathcal{X}$ and $\mathcal{U}$ are the admissible state space and control space, respectively. While $\mathbf{x}$ has been used in previous sections to denote only position, here it represents the full state. $\mathbf{x}_0$ is the initial state of the drone. Our experiments use the Parrot Bebop 2 quadrotor with its integrated front camera and electronic gimbal. It accepts desired roll and pitch angles, yaw rate, vertical velocity, and gimbal angles as input.

3.3.2 MPC Formulation

For each drone in the team, we formulate a receding horizon constrained optimization problem with N time steps and planning horizon $N\Delta t$, where Δt is the sampling time. The optimization seeks to minimize the weighted squared norm of a cost vector $\left\|\hat{\mathbf{J}}\right\|_Q^2$. Recall that $\mathbf{J}_i^j$ denotes the instantaneous viewpoint cost incurred by drone i capturing shot j. As desired viewpoint and drone are unambiguous in the MPC formulation, we drop the sub- and superscripts in this section. We augment $\mathbf{J}$ to include the control inputs, distance from reference state $\mathbf{x}_{ref}$, and collision avoidance slack variables $\mathbf{S}$:

$$\hat{\mathbf{J}}^k = [\mathbf{J}^T, \mathbf{u}^T, \|\mathbf{x}_{ref} - \mathbf{x}\|, \mathbf{S}^T]^T|_{t=k}.$$

A matrix of weightings Q defines the relative importance of each cost function. When the drone is assigned to take a shot, the weighting matrix does not penalize reference position error. When the drone transitions between shots, the relative viewpoint and distance are not penalized, although the image position cost c_{image} is kept. This keeps the drones pointing toward the target while repositioning for the next shot.

We use a third order collocation method that implicitly represents the drone's trajectory as a third order spline and the control inputs as piecewise linear functions of time as described in [37]. The dynamics constraints are applied at the collocation points $t_{c,n}$, and the drone state. The full MPC optimization can be written as:

$$\min_{\mathbf{x}^{1:N},\mathbf{u}^{1:N}} \quad \sum_{k=1}^{N-1} \left\| \hat{\mathbf{J}}^k \right\|_Q^2 + \left\| \hat{\mathbf{J}}^N \right\|_{Q_f}^2 \tag{3.8a}$$

$$\text{s.t.} \quad \mathbf{x}^1 = \mathbf{x}(0), \tag{3.8b}$$

$$\dot{\mathbf{x}}(t_{c,n}) = f(\mathbf{x}(t_{c,n}), \mathbf{u}(t_{c,n})), \tag{3.8c}$$

$$\mathbf{x}^k \in \mathcal{X}, \tag{3.8d}$$

$$\mathbf{u} \in \mathcal{U}, \tag{3.8e}$$

$$\forall k \in \{1, \ldots, N\}. \tag{3.8f}$$

At each time step, the drone solves the formulated nonlinear constrained optimization problem online to generate a local trajectory and executes the first time step controls in a receding horizon fashion. The optimization is performed using CasADi [45] for automatic differentiation and IPOPT [34] for optimization. Note that the optimization is not necessarily solved to completion — the current iterate after 40 ms of computation is used to select the next control output.

In order to estimate the costs at each timestep in the horizon, the drone must have an estimate for the future trajectories of dynamic obstacles and the videography target. Our prediction model assumes constant linear and angular velocity in the target's body frame, based on current estimated velocity. For our experiments the velocity estimate is provided by a motion capture system.

We implement the MPC for multiple drones in an asynchronous manner: the MPC is solved independently for each drone, and the expected trajectory each drone generates is used to compute collision avoidance by future iterations of the other

drones' MPC as described in detail in [46].

3.4 Experimental Results

We validated the tracking system with a pair of Bebop 2 drones tracking a human-driven RC racecar[2]. In order to generate the predictive target distribution necessary to calculate expected shot costs, the racecar was driven around two obstacles in a loosely defined pattern for ten warmup laps. Figure 3-10 shows the paths from these ten runs and their mean. These warmup runs were used to model a multivariate Gaussian distribution over the car's trajectory. Nine desired shots were specified for the drone, varying in length from four to six seconds and all constrained to occur within the ninety second duration of the car's lap. The image position parameters C_x and C_y were set to center the target in the frame in all shots. The settings for the other parameters are shown in Figure 3-9. The high-level shot planner assigned each drone a sequence of shots. The heuristic cost $H_{shot}(t)$ for each shot is shown in Figure 3-8. The planner is able to find an assignment of shots to the drones such that the expected viewpoint cost for each shot is quite low. We note that the planner assigns overlapping shots to the two drones resulting in a sequence of shots that would have been infeasible with a single drone.

The resulting assignments were tracked over a series of ten trials in which the racecar was manually driven along the same route as the warmup laps. Figure 3-10 shows the trajectories for the drones during one of the runs, in addition to the car's nominal path and two obstacles. The figure also demonstrates that the planning algorithm's penalty for transition distance between shots H_{trans} leads to the assigned shots being spatially segmented between the drones, reducing the total distance each has to travel.

Figure 3-9 illustrates the relative viewpoint achieved by each drone relative to the

[2] `https://mit-racecar.github.io/`

Table 3.2. The set of aesthetic parameters for the nine shots captured by the pair of drones. The gray cells correspond to the shots assigned to Drone 1.

Elevation Angle (Deg)	Azimuth Angle (Deg)	Distance (m)	Duration (s)
20	45	1.4	5
15	135	1.5	4
30	225	1.7	5
30	270	1.5	5
35	45	1.6	6
20	90	2.9	6
50	180	1	5
20	315	1	6
70	0	1	5

desired parameters during each shot. During each assigned shot window (denoted in green), the viewpoint parameters (in blue) are close to their desired values (in orange). The high-level planning heuristic and reference trajectory places the drones near the desired viewpoints, so in many cases the desired viewpoint is reached before the shot even starts. The drones are able to achieve lower-deviation trajectories during some shots compared to others. The higher-deviation shots occur when the desired drone position is obstructed (because of a static obstacle or other drone), or if the trajectory necessary to follow the desired position is dynamically infeasible (most commonly when the car is turning quickly). Even in cases where there is significant deviation from the desired viewpoint, such as the distance in Drone 1's third shot (at about T = 25s in (3-9(e))), we note that the other two parameters have low deviation and the drone maintains an unoccluded view of the target as shown in the accompanying video. Instances where the observed viewpoint deviations are greater than expected from the heuristic costs in Figure 3-8 are mostly due to the drones acting more conservatively when close to obstacles than expected by the planner.

The performance bottleneck for the high-level assignment algorithm is solving the ILP. The size of the ILP depends on the number of shots and the number of sampled times per shot. Our scenario contained nine shots, with twenty times sampled for each

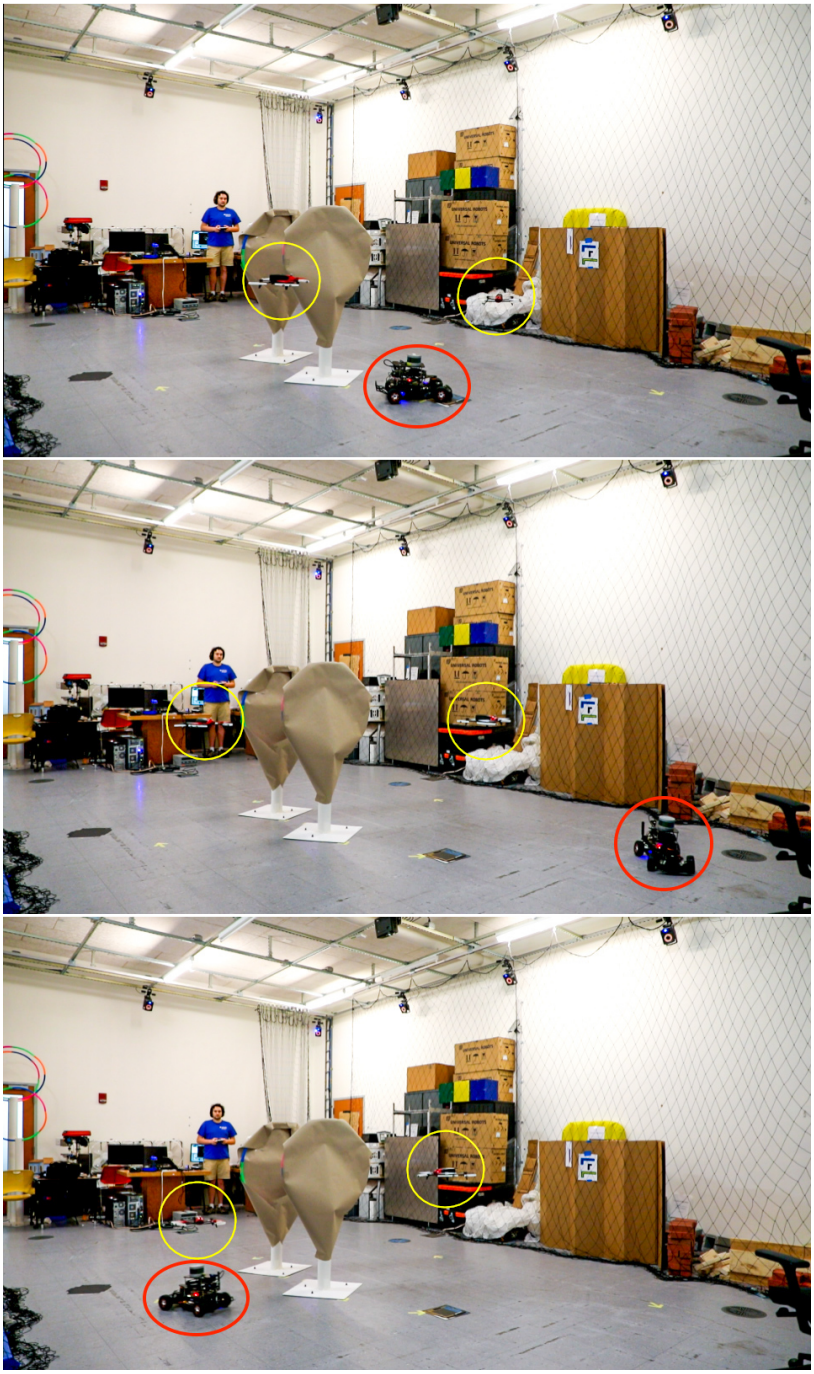


Figure 3-6. Positions of the drones at three different times as they track the car.

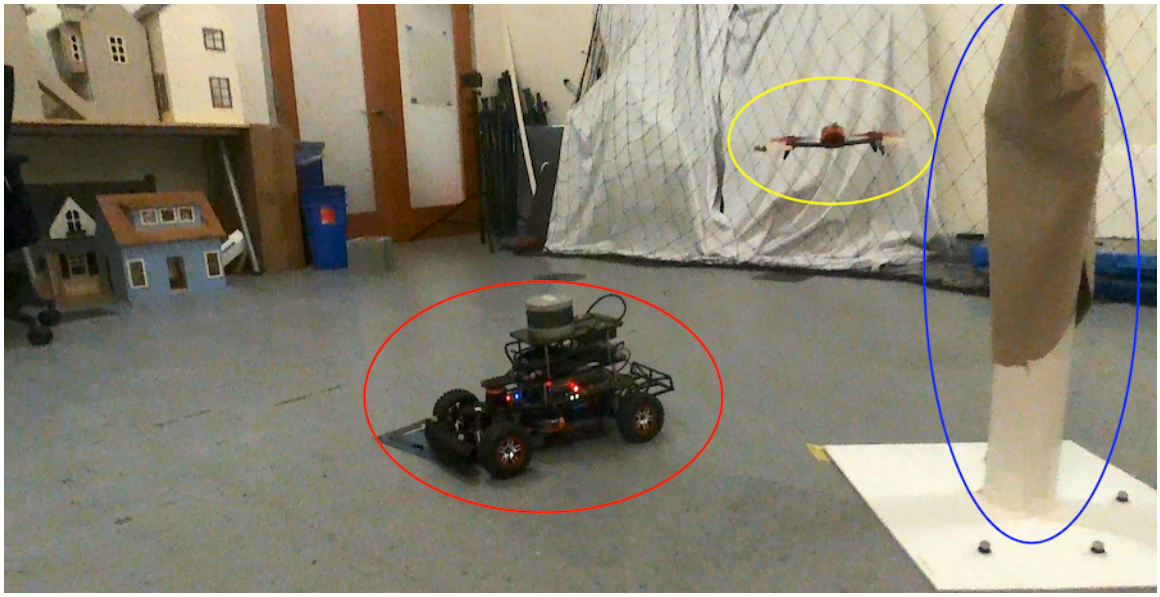
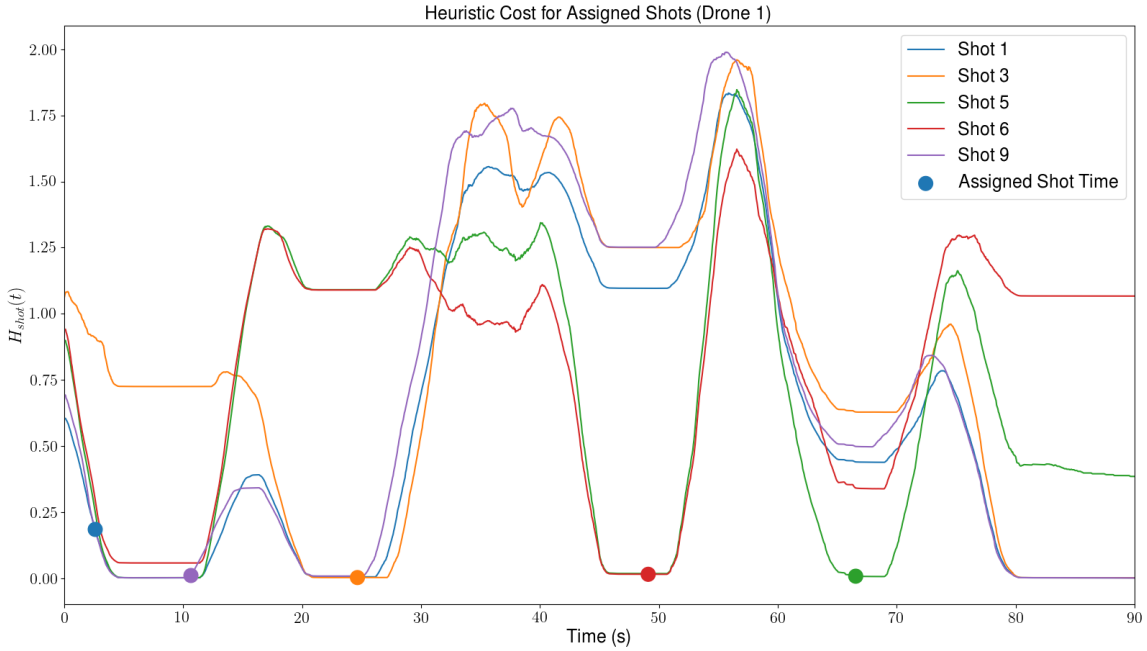
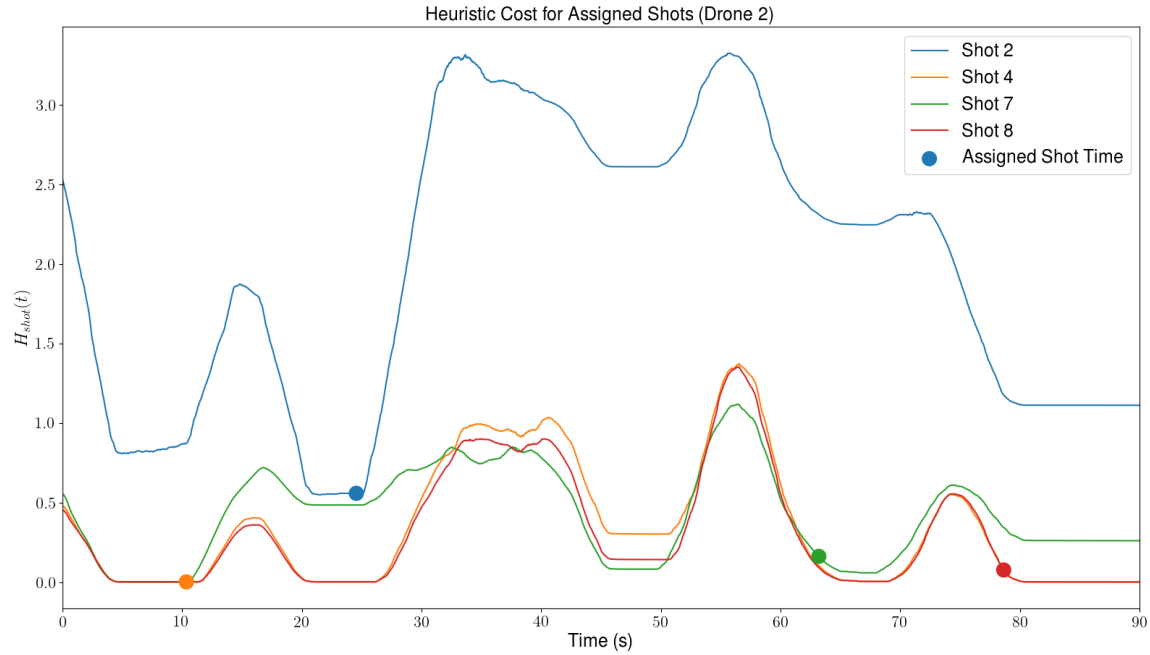


Figure 3-7. Viewpoint from a videography drone looking at a target (red). Another drone (yellow) optimizes the capture of a different shot while avoiding occlusions from obstacles (blue).

shot. The resulting ILP solved in under thirty seconds on a laptop with a 2.6 GHz Intel Core i7-9750H 6-Core processor and 16 GB RAM. Different balances between optimality and solution speed can be achieved by tuning the number of sampled times. We note that while the ILP does not run in real time, the thirty second solution time of this assignment could have been updated at least once online during the ninety second scenario to optimize the assignment and timing of remaining shots based on actual target motion so far. Thus, while the assignment algorithm is not fast enough to be considered online, it is not fully offline either – longer series of shots can be updated partway through a scenario.

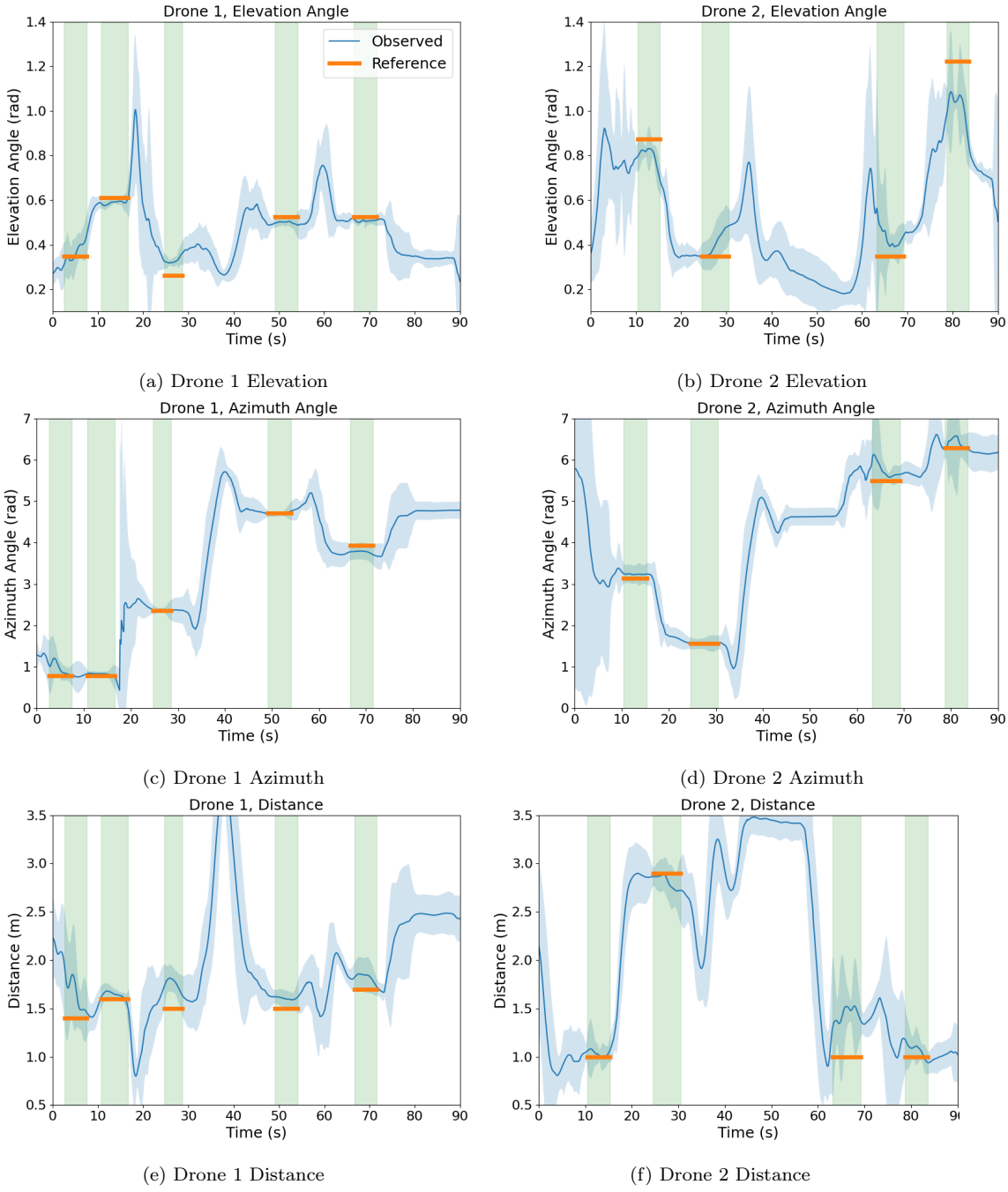


(a) Heuristic cost $H_{shot}(t)$ for each of the five shots ultimately assigned to Drone 1.



(b) Heuristic cost $H_{shot}(t)$ for each of the four shots ultimately assigned to Drone 2.

Figure 3-8. Heuristic cost over time for each desired shot. The solid circles denote the time that the high level planner determined was the lowest expected cost.



(a) Drone 1 Elevation

(b) Drone 2 Elevation

(c) Drone 1 Azimuth

(d) Drone 2 Azimuth

(e) Drone 1 Distance

(f) Drone 2 Distance

Figure 3-9. Performance of the two drones. Active shots are shaded in green, and orange bars represent the reference. The solid blue line represents the mean value over the 10 trials, and the transparent blue denotes one standard deviation of the data from the 10 trials. The drones converge to the reference values while in a shot.

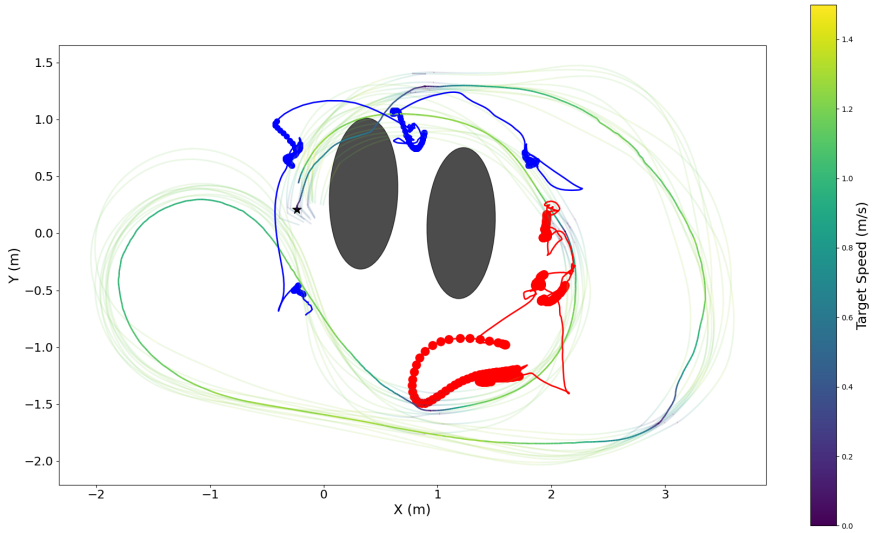


Figure 3-10. The predicted mean of the racecar trajectory (opaque blue/yellow curve) and its training runs (transparent blue/yellow curves). Car speed shown on color axis. The drone trajectories for one run are shown in red and blue. Circular markers denote times when the drones were taking a shot.

Chapter 4

Multi-Target Tracking

In the previous chapter, we considered a single mobile agent as the target of the tracking robots. Now, we consider the multi-target, multi-tracker problem, where a team of N trackers seeks to observe M targets from a specific relative viewpoint. Associated with each target $m \in M$ is a desired viewing direction η_m, a viewpoint tolerance θ_m, and a minimum distance r. In order for a tracking agent to successfully observe target m, it must enter a cone relative to m with medial axis η_m, opening angle θ_m, and height r. The targets and trackers may have arbitrary but known dynamics. The targets are driven by random, bounded control inputs and do not react to the motion of the trackers. The operational environment may have a set of obstacles, $\mathcal{O}$. The obstacle set $\mathcal{O}$ consists of convex polytopes. While each individual polytope representation must be convex, the sum of subsets of $\mathcal{O}$ need not be convex.

The team of tracking agents must be controlled such that all targets are successfully observed, and no tracker collides with an obstacle. We seek to minimize the time required for task completion.

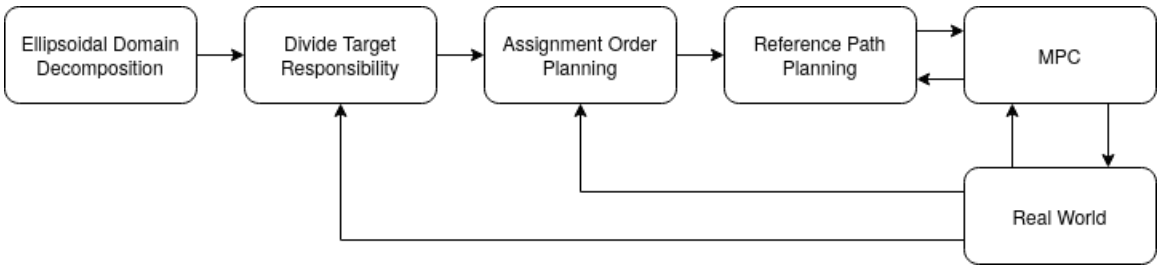


Figure 4-1. System Overview

4.1 Solution Overview

Our proposed system for achieving the desired viewpoint of each target is summarized in Figure 4-1. Before execution, an approximate ellipsoidal decomposition of free space in the domain is constructed offline. At runtime, each tracking agent is assigned a subset of the targets it is responsible for planning to observe. This subset updates according to changes in target and tracker distribution. Each tracking agent plans an order for visiting each of its assigned targets. The assignment order is used to generate a collision-free reference path for the tracker, which is followed by a receding horizon Model Predictive Controller (MPC).

4.2 Assignment and Path Planning

The proposed algorithm assigns each tracking agent a sequence of targets it is responsible for gathering information about. Each tracking agent plans a collision-free path that nominally guides it from its current position to the necessary viewpoint of its next target. The target assignment algorithm is decentralized, and both the assignment order and reference path are updated online.

4.2.1 Task Assignment

In order to plan paths for viewing all targets, we use a decentralized planning procedure where each tracking agent is responsible for assigning itself a visitation order to a

subset of the targets. The algorithm runs quickly and is updated online to incorporate information about current target positions.

Each tracking agent is responsible for viewing each target that is closer to it than any other agent. In other words, each tracker is responsible for the targets in its cell in the Voronoi tessellation generated by the tracking team. Following this division of the targets, each tracking agent independently plans a visitation order.

The visitation order is computed by finding a Hamiltonian path among the targets, with minimal length based on the *current* positions of each target and constrained to start at the current tracker position. The desired Hamiltonian path is found by constructing a modified graph where the shortest Hamiltonian cycle can be interpreted as the shortest Hamiltonian path in the original graph. We construct a complete graph consisting of a vertex for each target location, with edge weights corresponding to the distance between targets. An additional vertex is added to represent the tracker's current position. This additional vertex is connected to the other vertices with outgoing edges only, and edge weight corresponding to distance. Finally, all target vertices are connected to a "sink" vertex by directed edges pointed into the sink and with zero cost. The tracker vertex and sink vertex are connected by an edge of zero cost.

Finding the shortest Hamiltonian cycle in this graph is an instance of the asymmetric Traveling Salesman Problem (TSP). We turn the problem into an instance of symmetric TSP by adding ghost vertices as described in [47] so that high-performance solvers such as Concorde [48] can be employed. The resulting assignment order is denoted $\mathcal{T}$. While this assignment process ignores the future motion of the targets, we have found that frequent replanning with a fast TSP solver leads to better performance than updating less frequently from complicated assignment formulations that model the temporal information. The process of setting up and solving the TSP instance is further illustrated in Figure 4-2

While frequent updating of the assignment order is important for reacting to stochastic target motion, changing the assignment order every time a shorter Hamiltonian path is found may lead to undesirable behavior. Switching to a shorter Hamiltonian path may increase the distance to the first assigned target. As the target locations are time-varying, the assigned path length may increase over time, and a new, low-distance assignment may again change the first assignment. We avoid this behavior by only updating assignments when the proposed assignment has both a shorter total distance and a shorter distance to the first target.

4.2.2 Ellipsoidal Decomposition of Free Space

After a target visitation order has been determined, we require a reference path to guide the tracking agent from its current position to the target while avoiding obstacles. The low-level details of the final motion are be computed by a Model Predictive Control algorithm, which relies on the reference path to guide it around obstacles. Motivated by the MPC's collision avoidance constraints that keep the agent within obstacle-free ellipsoids, the reference path should be fully contained within a small number of ellipsoids to minimize the number of times the MPC constraints must be changed. It it also desirable to have a short reference path, and paths that can be contained within the minimal number of ellipses are usually short.

We begin by pre-computing an approximate decomposition of the environment's obstacle-free space as a union of ellipsoids. We generate a lattice $\mathcal{L}$ of points with spacing d and begin with no ellipsoids, i.e. ellipsoids set $\mathcal{E} = \emptyset$. Then, for each point in $l \in \mathcal{L}$, if it is not contained by any ellipsoid in $\mathcal{E}$, we add a new ellipsoid that has maximal volume while containing l and not intersecting with any obstacles. While finding the global maximum volume for an ellipsoid subject to these constraints is computationally difficult, methods such as [49] have demonstrated good locally

optimal results. We use the IRIS library[1] developed in [49] to compute the ellipsoids. The resulting ellipsoid set contains most of the free space in the environment, and the approximation quality can be adjusted by changing the lattice spacing d. IRIS does not actually constrain l to be inside the ellipsoid. Instead it uses l as a seed point of the optimization, but we have found that in practice the resulting ellipsoid set is still a good approximation of the obstacle-free volume. While the algorithmic development in this thesis does not consider agents with extent (i.e. the trackers are all point masses), the ellipsoids generated in the decomposition process can be shrunk to account for agent size.

Algorithm 2 Ellipsoid Decomposition

```
1:  procedure EllipsoidDecomposition(𝒪)
2:      ℰ = ∅
3:      ℒ ← LatticePoints(d)
4:      ℒ ← ℒ \ ℒ ∩ 𝒪
5:      while |ℒ| > 0 do
6:          l ← l ∈ ℒ
7:          ℒ ← ℒ \ l
8:          ℰ ← InflateEllipsoid(l, 𝒪) ∪ ℰ
9:          for p ∈ ℒ do
10:             if p ∈ ℰ then
11:                 ℒ ← ℒ \ p
12:     return ℰ
```

In order to use the ellipsoid set $\mathcal{E}$ for collision-free planning, we must be able to query which subset of ellipsoids contain a path connecting desired start and end points. To facilitate this, we construct a graph $\mathcal{G} = (V, \mathscr{E})$ that represents the connectivity of the ellipsoids. Each ellipsoid $\mathcal{E}_j \in \mathcal{E}$ is represented by a vertex $v_j \in V$, and two vertices share an edge if they intersect.

We utilize the method discussed in [50] to determine ellipsoid intersection. We

[1] `https://github.com/rdeits/iris-distro`

define ellipsoids $\mathcal{M}(M, \mathbf{m}$ by shape matrix M and center $\mathbf{m}$, i.e.

$$\mathcal{M}(M, \mathbf{m}) = \{\mathbf{x} \mid (\mathbf{x} - \mathbf{m})' M (\mathbf{x} - \mathbf{m}) \leq 1\}.$$

Proposition 3. *Consider two ellipsoids $\mathcal{A}(A, \mathbf{a})$ and $\mathcal{B}(B, \mathbf{b})$. Let*

$$K(\lambda, \mathcal{A}, \mathcal{B}) = 1 - (\mathbf{b} - \mathbf{a})^T \left(\frac{1}{1-\lambda} A^{-1} + \frac{1}{\lambda} B^{-1} \right)^{-1} (\mathbf{b} - \mathbf{a}).$$

$\mathcal{A}$ and $\mathcal{B}$ intersect if K is positive for all values of $\lambda \in (0, 1)$. Furthermore, K is a convex function of λ.

Proof. We refer to the presentation of Corollary 1 and Proposition 3 in [50]. □

Corollary 2. *As K is convex in λ, determining whether $\mathcal{A}$ and $\mathcal{B}$ intersect can be efficiently determined by comparing the minimum of $K(\lambda, \mathcal{A}, \mathcal{B})$ to zero*[2].

The edge set $\mathscr{E}$ of the connectivity graph is defined by

$$\mathscr{E} = \left\{ (i, j) \mid \min_{\lambda} K(\lambda, \mathcal{E}_i, \mathcal{E}_j) > 0 \right\}$$

for $\mathcal{E}_i, \mathcal{E}_j \in \mathcal{E}$.

4.2.3 Path Planning

In order to view each target, the tracking agents each need to navigate from their current position $\mathbf{x}^{tracker}$ to some nominal goal position $\mathbf{x}^{goal}$. To find a collision-free path from $\mathbf{x}^{tracker}$ to $\mathbf{x}^{goal}$, we augment $\mathcal{G}$ with an additional vertex for the start and goal positions. Each of the two additional vertices shares an edge with vertices corresponding to the ellipsoids containing that point. The augmented vertex set can

[2]The negativity condition can alternatively be checked by utilizing Sturm's theorem to count the roots of K on $(0, 1)$, although we do not employ this method. See [50] for details.

be written as

$$V' = V \cup \{v_{tracker}, v_{goal}\},$$

and the augmented edge set as

$$\begin{aligned}\mathscr{E}' = \mathscr{E} \cup \left\{(v_{tracker}, j) \mid \left\|(\mathbf{x}^{tracker} - \mathbf{e}_j)^T E_j (\mathbf{x}^{tracker} - \mathbf{e}_j)\right\| \leq 1\right\} \\ \cup \left\{(v_{goal}, j) \mid \left\|(\mathbf{x}^{goal} - \mathbf{e}_j)^T E_j (\mathbf{x}^{goal} - \mathbf{e}_j)\right\| \leq 1\right\},\end{aligned}$$

for $\mathcal{E}_j(E_j, \mathbf{e}_j) \in \mathcal{E}$.

The shortest path from v_{start} to v_{goal} in the augmented graph $\mathcal{G}' = (V', \mathscr{E}')$, which we denote $\mathcal{E}^*$, corresponds to the minimum number ellipsoids from $\mathcal{E}$ that must be passed through to move from $\mathbf{x}^{tracker}$ to $\mathbf{x}^{goal}$. When there are multiple paths through $\mathcal{G}'$ of equal length, ties are broken arbitrarily. While the minimum number of ellipsoids does not necessarily correspond to the shortest distance, in practice choosing this set of ellipsoids usually leads to reasonable paths and is desirable for minimizing the number of times the Model Predictive Control algorithm must switch its active constraints.

Lemma 3. *If there exists an ellipsoid $\mathcal{E}_j \in \mathcal{E}$ that contains both $\mathbf{x}^{tracker}$ and $\mathbf{x}^{goal}$, then there will be exactly one ellipsoid in $\mathcal{E}^*$ and it will contain both $\mathbf{x}^{tracker}$ and $\mathbf{x}^{goal}$.*

Proof. Let $\mathcal{E}_j$ contain both $\mathbf{x}^{tracker}$ and $\mathbf{x}^{goal}$. Then the path $[v_{start}, v_j, v_{goal}]$ exists in $\mathcal{G}'$ and $\mathcal{E}^*$ would contain one ellipsoid. As $\mathcal{E}^*$ is defined by the shortest path in $\mathcal{G}'$, it must contain one ellipsoid. Note that it may not contain $\mathcal{E}_j$ if multiple ellipsoids contain $\mathbf{x}^{tracker}$ and $\mathbf{x}^{goal}$, as ties for shortest path are broken arbitrarily. □

A reference path is generated from $\mathcal{E}^*$ by choosing a waypoint in the intersection of each pair of ellipsoids in the path. For two ellipsoids $\mathcal{A}(A, \mathbf{a})$ and $\mathcal{B}(B, \mathbf{b})$, let

$$E_\lambda = \lambda A + (1 - \lambda) B,$$

and

$$\mathbf{m}_\lambda = E_\lambda^{-1}(\lambda A\mathbf{a} + (1-\lambda)B\mathbf{b})$$

for some $\lambda \in [0,1]$. As proven in [50], $\mathbf{m}_\lambda$ is in both ellipses simultaneously for some value of λ on $[0,1]$ if and only if the two ellipsoids intersect. For $\lambda = 0$ and $\lambda = 1$, $\mathbf{m}_\lambda$ is equal to the origin of ellipse $\mathcal{B}$ and $\mathcal{A}$ respectively.

Proposition 4. *Consider two ellipsoids $\mathcal{A}(A,\mathbf{a})$ and $\mathcal{B}(B,\mathbf{b})$. $\|\mathbf{m}_\lambda - \mathbf{b}\|_B$ is monotonically increasing for $\lambda \in [0,1]$ and $\|\mathbf{m}_\lambda - \mathbf{a}\|_A$ is monotonically decreasing for $\lambda \in [0,1]$.*

Proof. Without loss of generality, let $\mathbf{b} = \mathbf{0}$ (because $\mathbf{m}_\lambda - \mathbf{b}$ is translation invariant). Consider a change of coordinates such that $\mathcal{B}$ is a unit (hyper)sphere. We denote the representation of the ellipsoids in the transformed space as $\hat{\mathcal{B}}(I,\mathbf{0})$ and $\hat{\mathcal{A}}(\hat{A},\hat{\mathbf{a}})$. As A is symmetric, it can be diagonalized as $A = PDP^{-1}$. We apply a second change of coordinates such that $\mathcal{A}$ is axis-aligned. This results in ellipsoids $\tilde{\mathcal{A}}(D,\tilde{\mathbf{a}})$ and $\tilde{\mathcal{B}}(I,\mathbf{0})$. Note that we can interpret this second transformation as rotating the coordinate system such that A is an axis-aligned ellipsoid, so $\mathcal{B}$ stays a unit sphere. As the coordinate transformations are linear, if $\|\mathbf{m}_\lambda\|$ is monotonically increasing in this new space, then $\|\mathbf{m}_\lambda - \mathbf{b}\|_B$ is also monotonically increasing in the original space. In the new coordinate system, we have

$$E_\lambda = \lambda D + (1-\lambda)I = \lambda \mathrm{diag}(d_1,\ldots,d_n) + (1-\lambda)I$$

and

$$\tilde{\mathbf{m}}_\lambda = \lambda E_\lambda^{-1} D\mathbf{a}.$$

As E_λ is diagonal, we can easily compute its inverse as

$$E_\lambda^{-1} = \mathrm{diag}\left(\frac{1}{1+\lambda(d_i-1)}\right).$$

We then have that

$$\begin{aligned}\left\|\tilde{\mathbf{m}}_\lambda\right\|^2 &= \lambda^2 \tilde{\mathbf{a}}^T D E_\lambda^{-1} E_\lambda^{-1} D \tilde{\mathbf{a}} \\ &= \sum_i \tilde{a}_i^2 d_i^2 \left(\frac{\lambda}{1+\lambda(d_i-1)}\right)^2.\end{aligned}$$

$\frac{\lambda}{1+\lambda(d_i-1)}$ is monotonically increasing for $\lambda \in [0,1]$ as long as $d_i > 0$, which holds as D is positive definite, so $\left\|\tilde{\mathbf{m}}_\lambda\right\|^2$ is monotonically increasing as well. As $\left\|\tilde{\mathbf{m}}_\lambda\right\|^2$ is related to $\left\|\mathbf{m}_\lambda\right\|_B^2$ through linear transformations, $\left\|\mathbf{m}_\lambda\right\|_B$ must be monotonically increasing, and therefore $\left\|\mathbf{m}_\lambda - \mathbf{b}\right\|_B$. By the symmetry of the proof, swapping $\mathcal{A}$ and $\mathcal{B}$ would show a monotonically increasing $\left\|\mathbf{m}_\lambda - \mathbf{a}\right\|_A$. However, swapping $\mathcal{A}$ and $\mathcal{B}$ is equivalent to switching λ for $1-\lambda$ (i.e. time-reversal), so for the original assignments of $\mathcal{A}$ and $\mathcal{B}$ we have that $\left\|\mathbf{m}_\lambda - \mathbf{a}\right\|_A$ is monotonically decreasing. □

We can use the resulting montonicity of $\mathbf{m}_\lambda$ to conduct a binary search over λ to find a value such that $\mathbf{m}_\lambda$ is in both $\mathcal{A}$ and $\mathcal{B}$.

Thus, a binary search over λ can find a point $\mathbf{m}_\lambda$ that is contained within both ellipsoids. A series of waypoints between $\mathbf{x}^{tracker}$ and $\mathbf{x}^{goal}$ is constructed by finding such a point in all sequential pairs of ellipsoids in $\mathcal{E}^*$. We denote this series of waypoints P^*, and let P_i^* correspond to the i^{th} waypoint along the path.

The waypoints implicitly define a piecewise linear path from $\mathbf{x}^{tracker}$ to $\mathbf{x}^{goal}$. A controller that follows this path perfectly would be guaranteed to avoid collision with obstacles. As nontrivial system dynamics may make it impossible to follow the reference path exactly, our controller only uses the intersection waypoints P^* as a guide, and runs an online nonlinear optimization routine that constrains trajectories to stay within $\mathcal{E}^*$.

Remark 1. *The first ellipsoid on the path to the target, $\mathcal{E}_1^*$, contains both $\mathbf{x}^{tracker}$ and P_1^*. As a result, $\mathcal{E}_1^*$ can always be chosen as a convex constraint on $\mathbf{x}^{tracker}$ that allows P_1^* to be reached, assuming the agent is controllable enough to stay inside of*

$\mathcal{E}_1^*$.

4.3 Collision-Free Viewpoint Optimization

Once the assignment order and reference path have been generated, a receding horizon Model Predictive Control algorithm controls each tracking agent to achieve the desired viewpoint and avoid collision.

4.3.1 Observation Cost Model

We first define a viewpoint cost function that encodes the desire to view the target from a specific angle and distance. Consider a target at position $\mathbf{x}^{target}$ with a desired viewing direction η_t. In order to get the desired observation of the target, the tracking agent must enter into a cone (or triangle in 2D) with opening angle θ around η_t, and within distance r. Let $\tilde{\mathbf{x}}$ denote the vector from $\mathbf{x}^{target}$ to $\mathbf{x}^{tracker}$. The viewpoint cost function l is composed of a term that penalizes the deviation of $\tilde{\mathbf{x}}$ from the direction of η_t and the length of $\tilde{\mathbf{x}}$ from the desired viewing distance. The viewpoint cost function is defined as

$$l(x) = \frac{\tilde{\mathbf{x}}^T}{\|\tilde{\mathbf{x}}\|}\eta_t + (\|\tilde{\mathbf{x}}\|^2 - r^2)^2. \tag{4.1}$$

The position and direction of interest of the target are predicted over the horizon of the MPC planning. As a result, $l(\mathbf{x})$ is implicitly a function of time from the time-dependence of $\tilde{\mathbf{x}}$ and η. We make this explicit by notating $l_i(\mathbf{x})$ as the cost function relative to the target at time i. Our implementation predicts the target's future motion assuming no control inputs are applied to the target over the planning horizon, although more sophisticated prediction methods would work as well. When a tracker agent's current position relative to its current target enters the desired viewing cone, we consider the target to have been visited and exclude it from future planning.

4.3.2 Optimizing for Sequential Goals

As the desired visitation order for several targets is known when executing the MPC, we can include an additional cost term that incentivizes progress toward the next target succeeding the current one. These two targets are denoted the "primary" goal and "secondary" goal, and have viewpoint cost functions denoted $l^1(\mathbf{x})$ and $l^2(\mathbf{x})$. Simply adding l^2 as a terminal cost to encourage progress toward the secondary goal is undesirable, as careful tuning of the cost functions would be required to ensure the secondary cost does not prevent progress toward the primary goal. Instead, we define a *target switching index* h_t, which determines when the MPC switches from the primary cost function to the secondary cost function. Initially, h_t is set to $N+1$, one larger than the MPC horizon. Each time a solution is returned from the MPC, the predicted trajectory for the tracking agent is compared to the desired viewing region of the target. h_t is set to the first index where the tracking trajectory enters the desired viewing region. Intuitively, after the tracking agent is expected to achieve the desired viewpoint, it should spend the rest of the horizon optimizing for the secondary goal.

As the tracker gets closer to the primary goal, more of its trajectory is incentivized to move toward the secondary goal. Because of stochastic target motion, there are cases where the MPC's predicted trajectory will no longer achieve the desired viewpoint. When the expected tracker trajectory at some iteration does not attain the viewpoint for any timestep in the planning horizon, h_t is slightly increased for the next MPC iteration. The intuition is that as more of the planning horizon has a cost function prioritizing the secondary goal, the optimal trajectory may be "pulled" out of the primary viewing region. Increasing h_t serves as negative feedback to increase the trajectory's cost related to the primary goal on the next MPC iteration.

4.3.3 Avoiding Collisions

The tracking agents can avoid collisions with the environment by staying within the union of obstacle-free ellipsoids $\mathcal{E}$. However, the union of these ellipsoids is nonconvex and imposing the constraint directly onto the MPC increases solution times and leads to poor local optima. Previous work, such as [51], [52] formulate the constraints as a mixed-integer convex nonlinear program, where exactly one ellipsoid is chosen to be the active constraint at each timestep. However, requiring the solver to support integer constraint removes the possibility of using popular solvers for MPC such as IPOPT. Analogous to the target switching index for deciding which cost function to use, we employ an *ellipsoid switching index* h_e, that controls which collision avoidance ellipsoid constrains each stage of the MPC. The ellipsoid used as a constraint at each timestep follows directly from the sequence of reference waypoints P^*.

Recall that $\mathcal{T}$ denotes the ordered list of targets assigned to an agent. Let T_1 denote the first target in $\mathcal{T}$. If there are at least two targets in $\mathcal{T}$, let T_2 denote the second target. Otherwise, let $T_2 = T_1$. Furthermore, let $\mathcal{E}^+$ denote the sequence of ellipsoids required to move from T_1 to T_2. If there is an ellipsoid that contains both the tracking agent and target T_1, then P^* is empty and the tracker's primary goal is to obtain the desired view of the target. Otherwise, P^* contains at least one waypoint, and the primary goal is waypoint P_1^*. If the primary goal is a waypoint, the secondary goal is waypoint P_2^* unless P_1^* shares an ellipsoid with T_1, in which case the secondary goal is T_1. If the primary goal is target T_1, then the secondary goal is either T_2 or a waypoint on the way from T_1 to T_2. Figure 4-6 summarizes the rules for choosing primary and secondary goals.

The MPC is constrained by two ellipsoids, $\mathcal{M}_1(M_1, \mathbf{m}_1)$ and $\mathcal{M}_2(M_2, \mathbf{m}_2)$. The first corresponds to $\mathcal{E}_1^*$ which contains both the tracking agent and the primary goal, whether the primary goal is target T_1 (by Lemma 3) or waypoint P_1^* (as discussed in Remark 1). The second constraint ellipsoid $\mathcal{M}_2$ is set to $\mathcal{E}_2^*$ if $|\mathcal{E}| > 1$. In this

case, $\mathcal{E}_2^*$ contains P_1^* and P_2^*, so the MPC can plan a path up to P_2^*. If $\mathcal{E}_1^*$ is the only ellipsoid on the path to the current target, the first ellipsoid on the path from T_1 to T_2 is chosen, $\mathcal{E}_1^+$.

The first ellipsoid constraint is active until step h_e of the MPC horizon. The second ellipsoid constraint is active from step h_e onward. This is denoted as a pair of quadratic constraints for each agent,

$$\left\|(\mathbf{x}_k - \mathbf{m}_i)^T M_i (\mathbf{x}_k - \mathbf{m}_i)\right\|^2 \leq d_k^i,\ i \in \{0, 1\},$$

defined for each step k of the MPC horizon.

Setting d_k^i to 1 constrains $\mathbf{x}$ to be within ellipsoid $\mathcal{M}_i$ at time k. Setting d_k^i to ∞ turns off the constraint at time k. We set d_k^1 to ∞ if $k \geq h_e$, and 1 otherwise. Similarly, d_k^2 is equal to ∞ if $k < h_e$, and 1 otherwise. As a result, in the interval $[0, h_e)$, only the $\mathcal{M}_1$ ellipsoid constraint must be satisfied, and vice-versa for $[h_e, N-1]$. Within each of these intervals, the feasible positions for $\mathbf{x}$ are convex, which leads to much more reliable MPC performance.

The primary waypoint P_1^* is considered a stationary primary goal with a desired viewing distance of $r = \epsilon$ and a cone viewing angle of $\theta = 2\pi$. As a result, the target switching index h_t considers the waypoint as visited if the trajectory expected by the MPC comes within ϵ of the waypoint. The distance ϵ can be chosen to trade off between tracker speed and robustness. Larger values of ϵ allow for more aggressive corner cutting, but may lead to the MPC solver getting stuck on nonconvex corners at ellipsoid intersection points, especially in more difficult environments.

4.3.4 Optimization Formulation

Only the tracker's position, $\mathbf{x}$, has been necessary to define the viewpoint cost and collision avoidance constraints. However, controlling the agent may require planning

trajectories in a larger state space, which we denote $\mathbf{w}$. We take $\mathbf{x}$ to implicitly be a subset of the state elements in $\mathbf{w}$. The vector $\mathbf{u}$ represents the agent's control inputs. The dynamics are assumed to be represented by some smooth function f, such that

$$\dot{\mathbf{w}} = f(\mathbf{w}, \mathbf{u}).$$

The full MPC formulation can be written as a constrained nonlinear optimization:

$$\begin{aligned}
\underset{\mathbf{w}_{1:N}, \mathbf{u}_{1:N}}{\operatorname{argmin}} \quad & \sum_{k=1}^{N} l_k^1(\mathbf{x}_k)\mathbb{1}(k < h_t) + l_k^2(\mathbf{x}_k)\mathbb{1}(k \geq h_e) + q(\mathbf{u}_k) && \text{(4.2a)} \\
\text{s.t.} \quad & \mathbf{w}_1 = \mathbf{w}(0), && \text{(4.2b)} \\
& \dot{\mathbf{w}}(t_{c,n}) = f(\mathbf{w}(t_{c,n}), \mathbf{u}(t_{c,n})), && \text{(4.2c)} \\
& \left\| (\mathbf{x}_k - \mathbf{m}_i)^T M_i (\mathbf{x}_k - \mathbf{m}_i) \right\|^2 \leq d_k^i && \text{(4.2d)} \\
& \mathbf{w} \in \mathcal{W}, && \text{(4.2e)} \\
& \mathbf{u} \in \mathcal{U}, && \text{(4.2f)} \\
& \forall k \in \{1, \ldots, N\}. && \text{(4.2g)}
\end{aligned}$$

The dynamics of the tracking system are enforced with a third order collocation method (4.2c), as discussed in [37]. $t_{c,n}$ denotes the time corresponding to the n^{th} collocation point, and the resulting constraint be represented directly in terms of $\mathbf{x}_{1:N}$ and $\mathbf{u}_{1:N}$. A control effort regularization function q is added to the cost function to encourage smoother control inputs. We let $q(\mathbf{u})$ be proportional to $\|\mathbf{u}\|^2$. The first control input $\mathbf{u}_1$ is applied to the system, and then the optimization is run again at each successive timestep. We show in Section 4.4 that (4.2) can be solved in real time.

Algorithm 3 summarizes the steps in the algorithm multi-target tracking algorithm. In lines 2-3, the ellipsoidal decomposition of the domain is constructed as

discussed in Section 4.2.2. Lines 6-7 assign a sequence of targets to each tracking agent as discussed in Section 4.2.1. The series of ellipsoids between the tracker's current position and the target and the reference waypoints developed in Section 4.2.3 are generated on lines 8-9. The MPC optimization from Section 4.3.4 is solved on line 10, and the updates to h_t and h_e discussed in Sections 4.3.2 and 4.3.3 occur on line 11.

Algorithm 3 Multi-Target Tracking

1: **procedure** RUNTRACKING($\mathcal{O}$)
2: $\mathcal{E} \leftarrow$EllipsoidDecomposition($\mathcal{O}$)
3: $\mathcal{G} \leftarrow$BuildConnectivityGraph($\mathcal{E}$)
4: **while** Targets are Available **do**
5: **for** $i = 1 : N$ **do**
6: $\mathbf{r}_i \leftarrow$ TargetsInVoronoiCell($\mathcal{X}^{trackers}$)
7: $\mathcal{T} \leftarrow$ TaskAssignment($\mathbf{r}_i,\ \mathbf{x}_i$)
8: $\mathcal{E}^* \leftarrow$ EllipsoidPath($\mathbf{x}_i, T_1$)
9: $P^* \leftarrow$ GenerateWaypoints($\mathcal{E}^*$)
10: $\mathbf{w}_i^{pred}, \mathbf{u}_i \leftarrow$ SolveMPC($\mathbf{x}_i, \mathcal{T}, \mathbf{h}, \mathcal{E}^*, \mathcal{P}^*$)
11: $\mathbf{h} \leftarrow$ UpdateSwitchingIndices($\mathbf{h}, \mathbf{w}_i^{pred}$)

4.4 Evaluation

We evaluate the proposed algorithm in a 2D simulation. Teams of tracking agents and targets both move subject to double integrator unicycle dynamics. The state consists of position, heading angle, linear velocity, and angular velocity:

$$\mathbf{w} = \begin{bmatrix} x & y & \theta & v & \omega \end{bmatrix}^T,$$

and the control input consists of longitudinal and angular acceleration:

$$\mathbf{u} = \begin{bmatrix} a & \alpha \end{bmatrix}$$

The dynamics evolve according to:

$$\dot{\mathbf{w}} = f(\mathbf{w}, \mathbf{u}) = \begin{bmatrix} v\cos(\theta) & v\sin(\theta) & \omega & a & \alpha \end{bmatrix}^T.$$

The viewpoint-optimizing cost function (4.1) for each agent is optimized subject to the agent dynamics by implementing (4.2) in Casadi [45] and solving with IPOPT [53]. The MPC considers a five second horizon, comprised of 40 equally-spaced steps. Each MPC solution takes about 50ms. The Traveling Salesman Problem instances that are solved by the higher-level planner take about 30ms for up to 20 targets (per tracker) and up to 200ms for 50 targets. All experiments were conducted on an Intel Core i7-9750H CPU @ 2.60GHz.

We present two different simulation scenarios. The is first environment inspired by an urban setting and comprises large square obstacles in a grid, shown in Figure 4-8. The second environment simulates a forest, where circular obstacles are randomly generated, shown in Figure 4-7. In both cases, the target agents move according to bounded random control inputs and are allowed to enter obstacles. The tracking agents are constrained to not collide with any obstacles as described in Section 4.3.3.

We simulate the urban and wooded environment each with 20 random starting configurations for 20 targets and one, two, five, and seven tracking agents. Figures 4-9 and 4-10 illustrate the scaling of total simulation time until the observation of all targets as a function of the number of tracking agents.

We also analyzed the performance gained by the MPC being able to account for predicted future motion of the targets. In each domain, 100 trials with and without dynamics target prediction were run. Each trial consisted of a single tracking agent and 20 targets with random starting positions. For the urban domain, the average time to capture of all targets was 90.4 seconds when prediction was enabled, compared to 95.8 seconds when prediction was disabled. In the forest domain, the average time

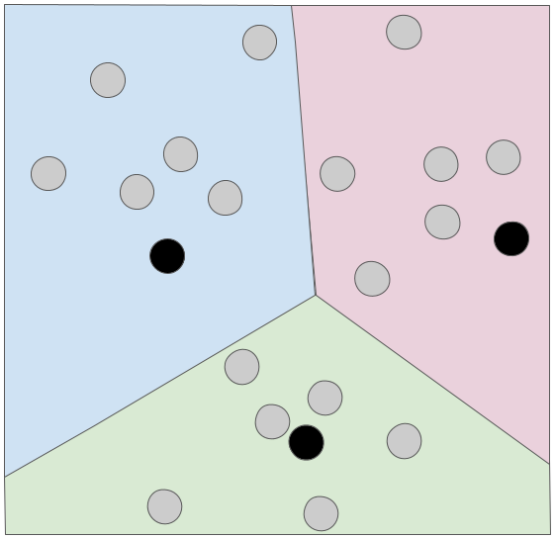
was 156.0 seconds with prediction enabled versus 172.25 seconds with it disabled.

4.4.1 Discussion

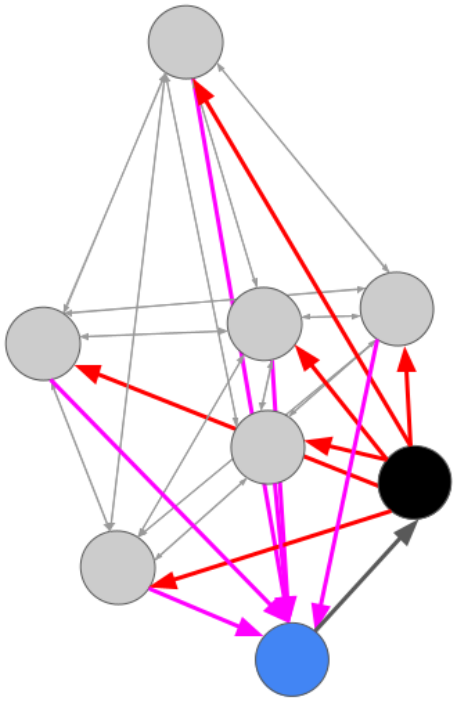
While the algorithm's simulation results show promise, there are currently a few known limitations. First, the assignment submodule does not explicitly handle targets that are fully obstructed by an obstacle. When a tracking agent is assigned to visit a target that is fully obstructed, it will optimize its position within the ellipsoid closest to the desired target, even if there are other targets that still need to be visited that are not currently obstructed. While this behavior works well when targets are obscured only briefly, specially handling targets that cannot currently be visited may improve performance in some situations.

The second potential limitation is that the reference path generated by the planning submodule may be longer than necessary depending on the underlying ellipsoidal decomposition of the free space. The algorithm's heuristic for choosing the minimal number of ellipsoids to travel between two points often works well, especially when there is ample free space. However, sometimes the least number of ellipsoids between two points may involve a path that is substantially longer than the shortest possible collision-free path (this tends to happen when very long, skinny ellipsoids intersect).

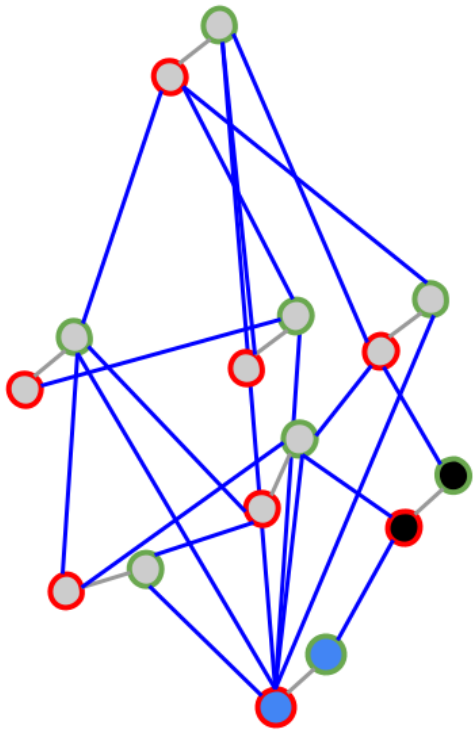
Figure 4-2. Visual summary of how target locations are assigned to trackers (a), the visitation order is solved as Asymmetric TSP (b), the ATSP instance is turned into a Symmetric TSP instance (c), and the solution order is extracted (d-e).



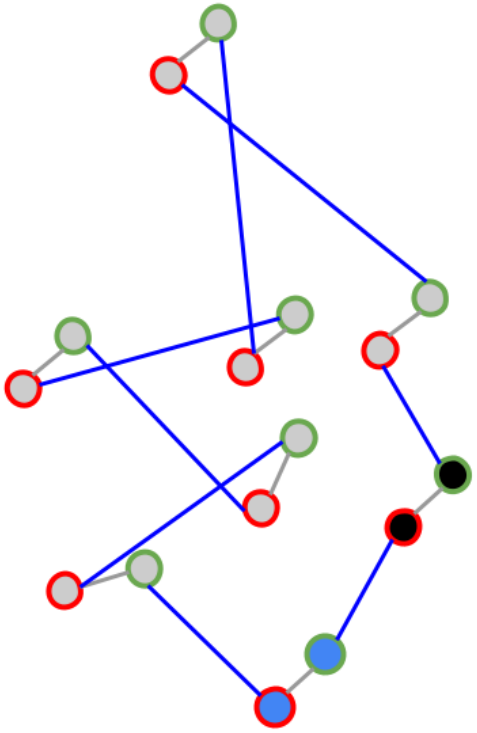
(a) Responsibility for the targets (gray circles) is divided among the tracking agents (black circles) based on which tracker is closest to each target (colored regions).



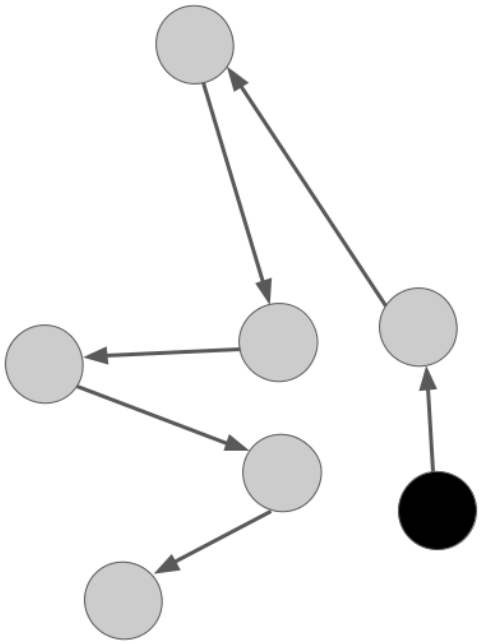
(b) For the targets assigned to each tracker, we consider the complete graph connecting the targets, where edge is defined by distance. A node corresponding to the tracking agent connects to all targets with only outgoing edges (red edges). All target nodes are connected to a "sink" node (blue circle) with outgoing edges (magenta edges). A single directed edge connects the sink node to the tracker node (black edge). A Hamiltonian cycle in this graph is a Hamiltonian path among the target and trackers, with the tracker at one end. The directed edges makes this an Asymmetric TSP Problem.



(c) To turn the Asymmetric TSP into a standard TSP, add a "ghost" node (green border) for every original node (red border), that has the same connectivity as the original node. Only the edges that connect opposite-colored nodes are kept. Each ghost node is connected to its corresponding original node by a zero-cost edge (gray edges). Only a subset of edges in the graph are shown for easy of reading.



(d) Example solution to the TSP problem on the modified graph.



(e) Corresponding Hamiltonian Path solution on the original set of tracker and targets.

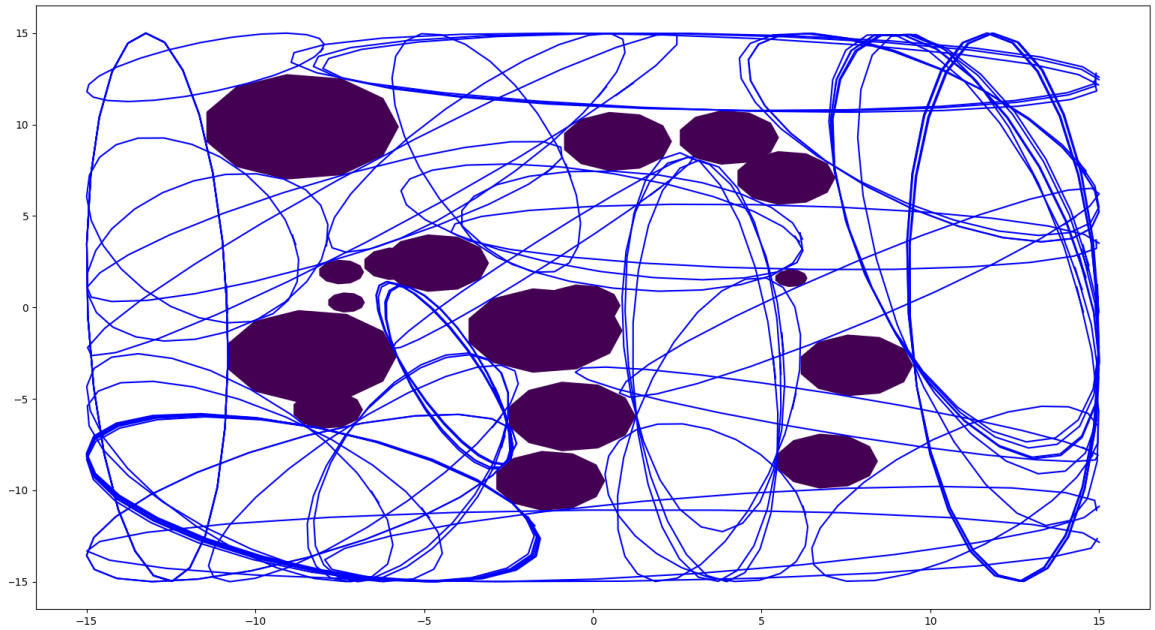


Figure 4-3. Example ellipsoid decomposition of an environment, generated by Algorithm 2

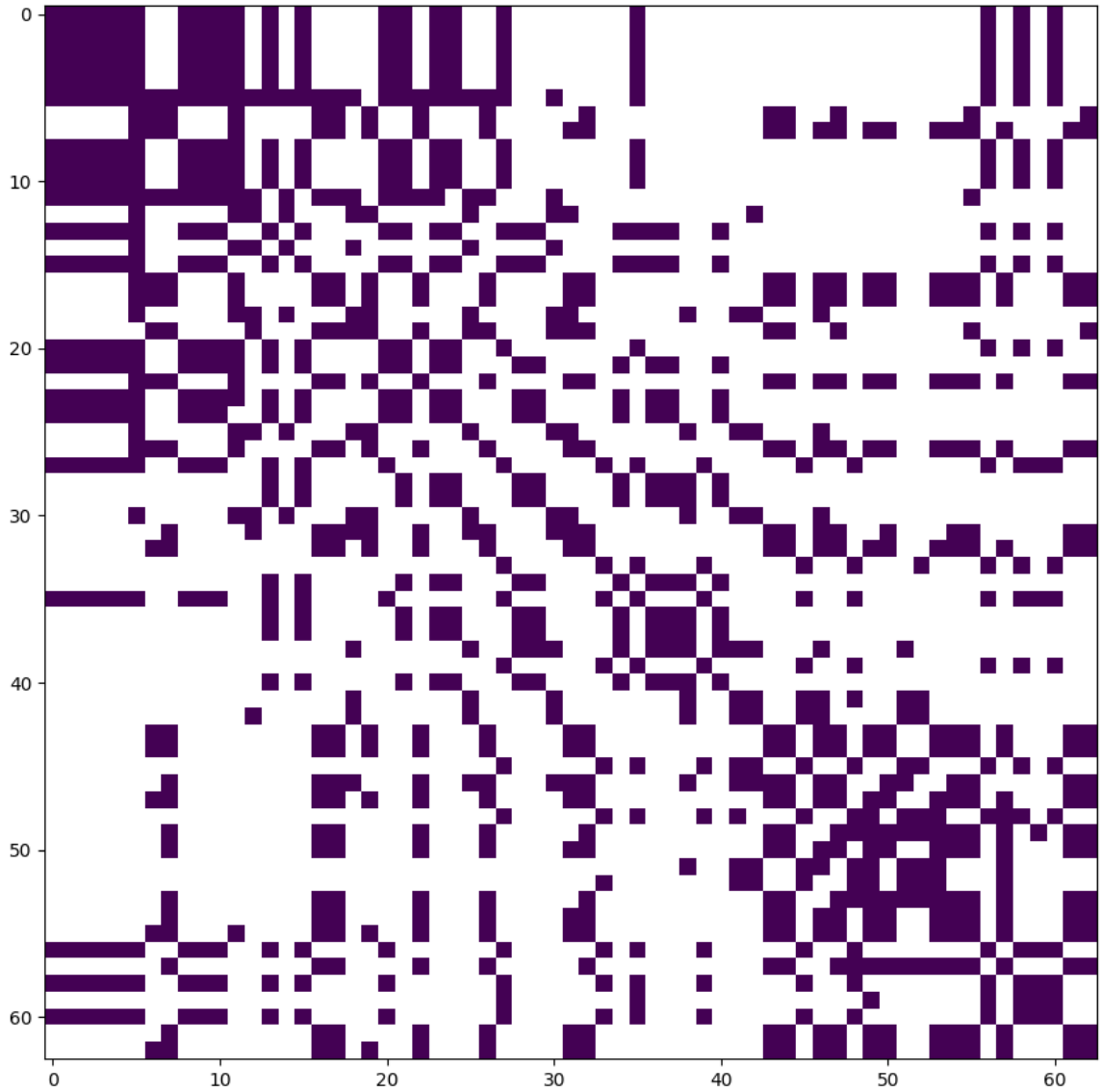


Figure 4-4. Graph adjacency matrix associated with the decomposition in Figure 4-3

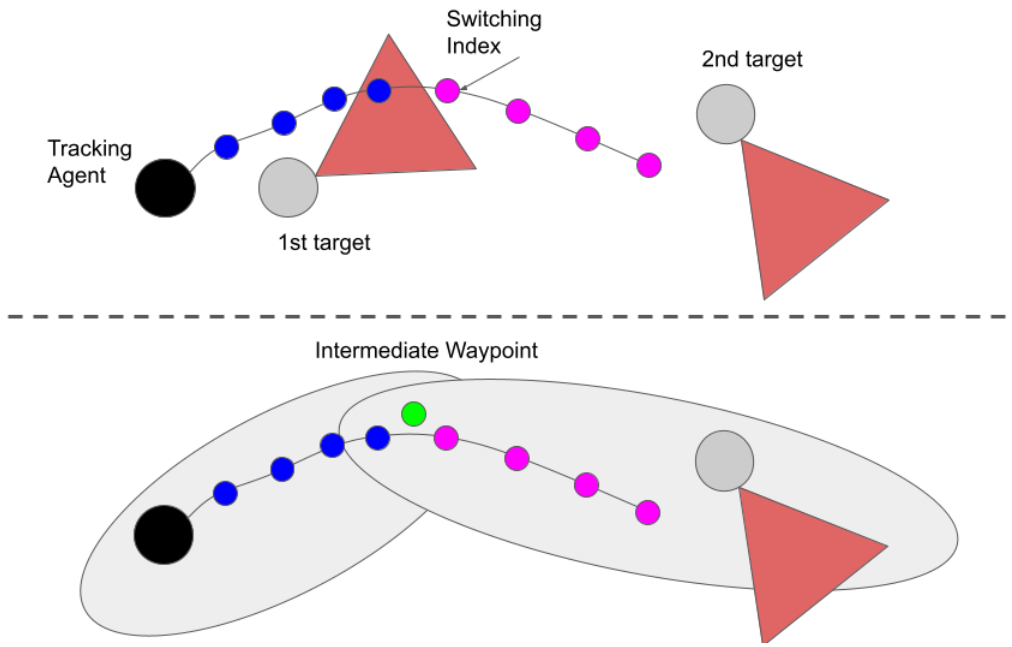


Figure 4-5. Example of the target switching index h_t. *Top*: the first half of the trajectory (in blue) optimizes cost $l^1(x)$ for a good viewpoint of the first target. The second half of the trajectory (in magenta) optimizes cost $l^2(x)$ for a good viewpoint of the second target. The switching index h_t is updated between MPC iterations depending on which timesteps of the previous solution intersected with the target's viewing cone. *Bottom*: The same principle is used when navigating via an intermediate waypoint (shown in green) instead of target locations, as described in the next section. The sparse waypoints are used to calculate the initial cost l^1 until the switching index h_t, and then the rest of the horizon optimizes for progress to the following waypoint or target position. An analogous switching process is used to change which ellipsoid constraints are active during the trajectory

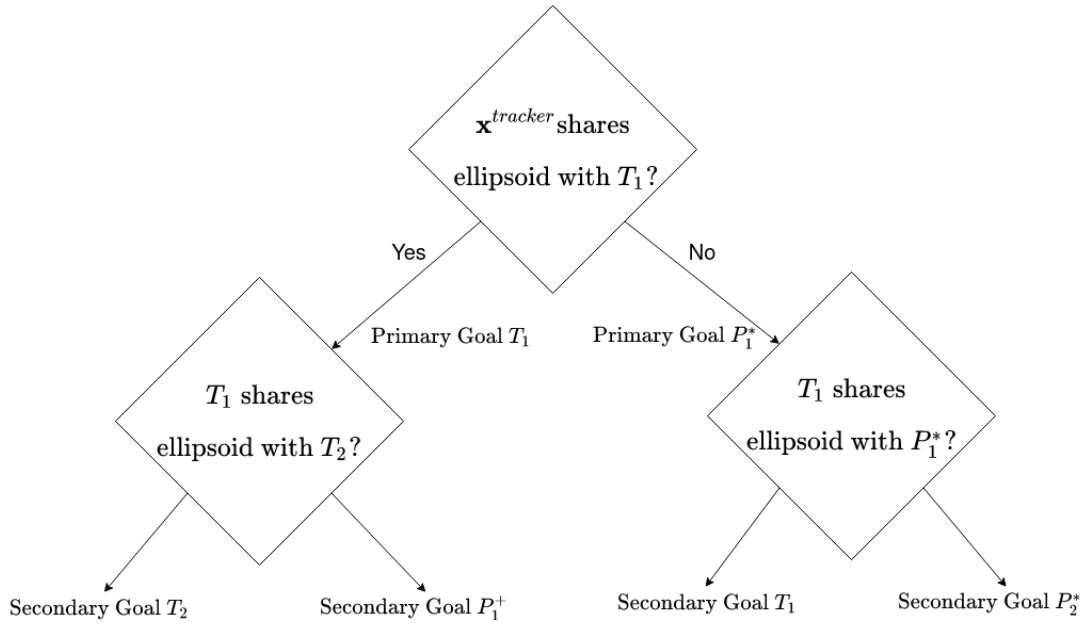


Figure 4-6. Rules for choosing the tracking agent's primary and secondary goals for the MPC planning horizon.

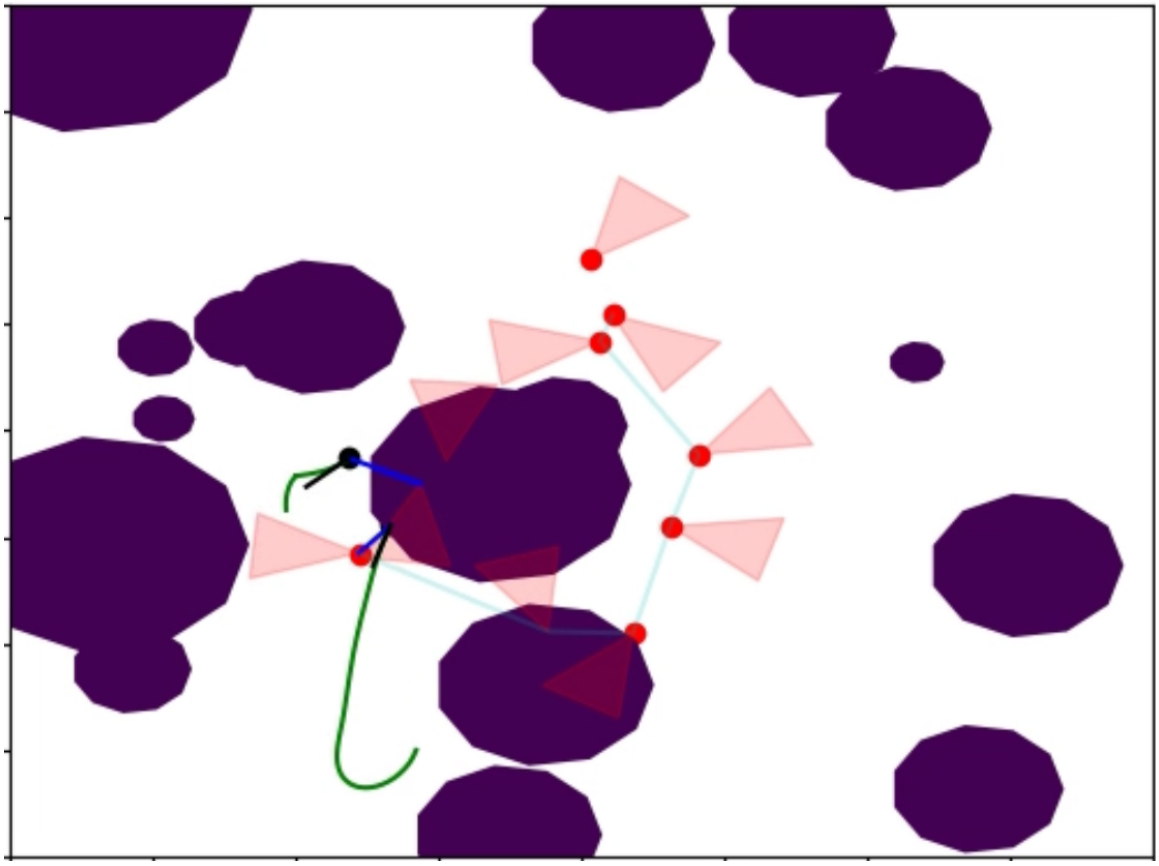
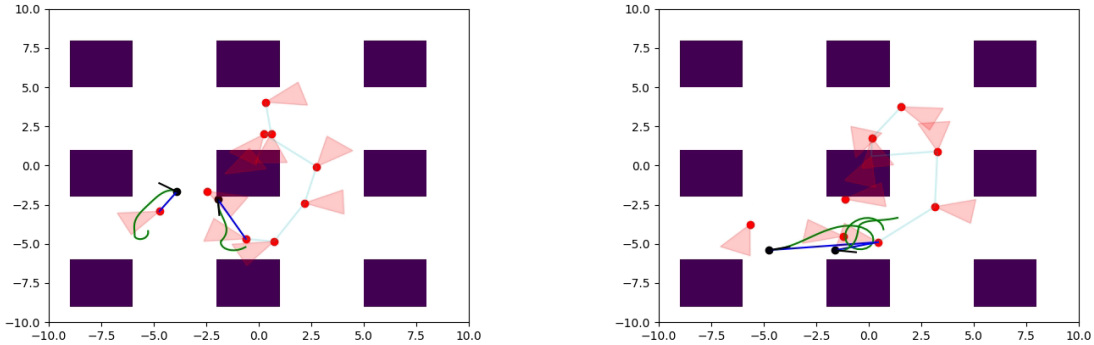
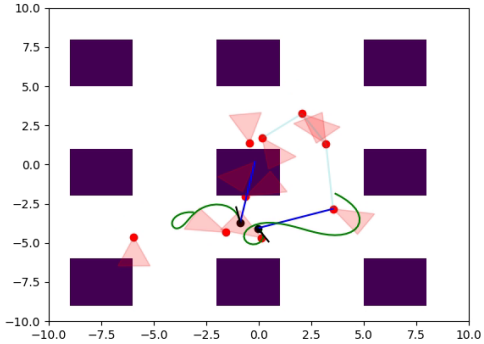


Figure 4-7. Forest-Inspired Domain. The purple blocks represent obstacles. The red dots represent targets of interest, and the red triangles are relative viewpoint from which they must be viewed by the trackers. The green lines are the tracking agent's MPC plan at the current iteration. The light blue line is the sequence of agents assigned to each tracker. The dark blue line is the currently-assigned target.

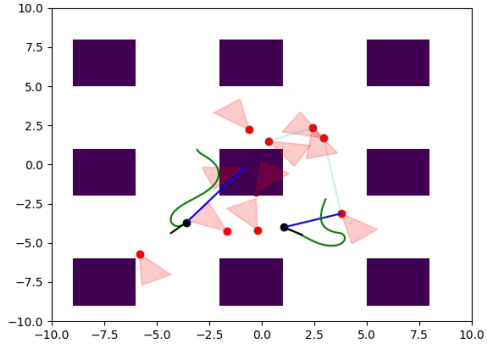


(a) Step 39

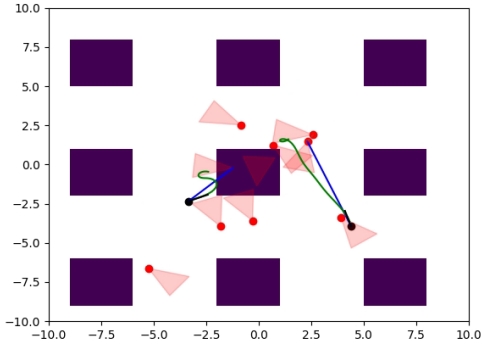
(b) Step 67



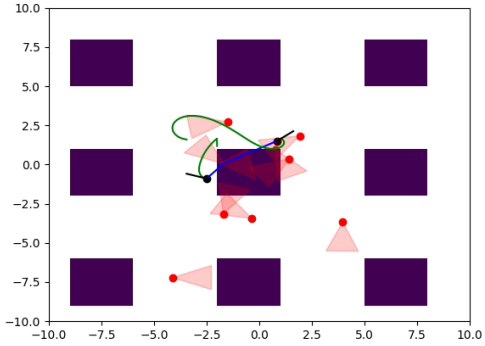
(c) Step 82



(d) Step 101



(e) Step 119



(f) Step 144

Figure 4-8. Evolution of a tracking simulation over time. Two tracking agents are pictured as black dots. The purple blocks represent obstacles. The red dots represent targets of interest, and the red triangles are relative viewpoint from which they must be viewed by the trackers. The green lines are the tracking agents' MPC plan at the current iteration. The light blue line is the sequence of agents assigned to each tracker. The dark blue line is the currently-assigned target.

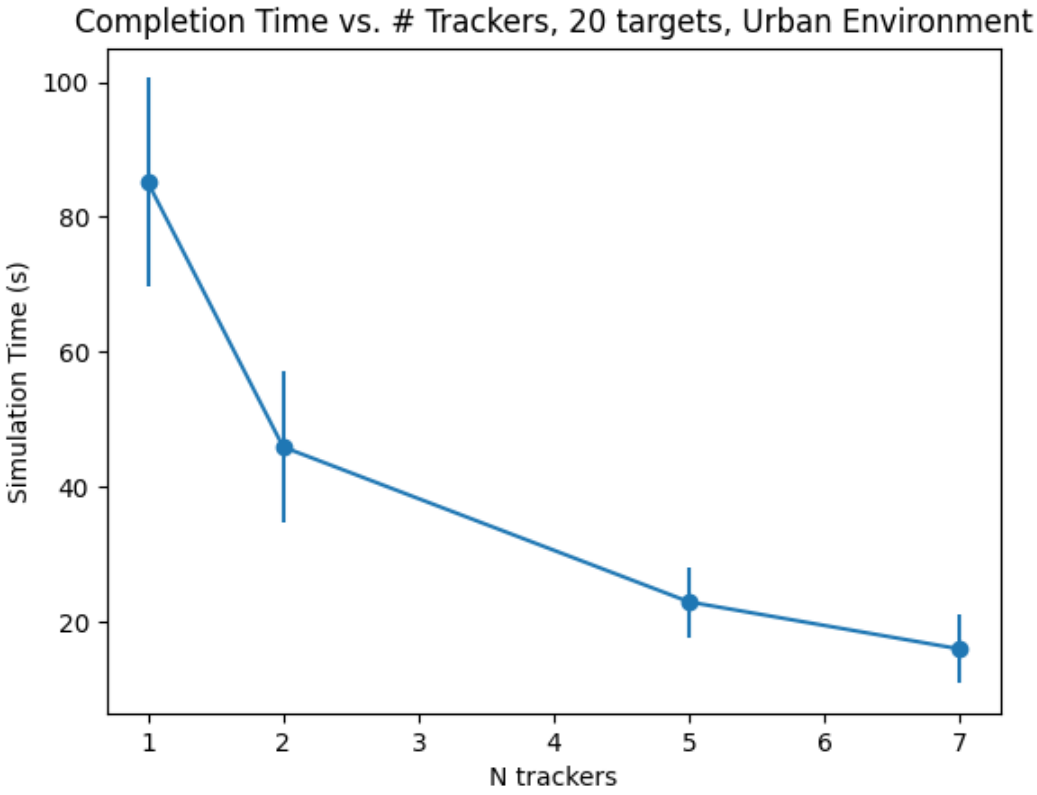


Figure 4-9. Scaling of task completion time in the urban domain for an increasing number of agents.

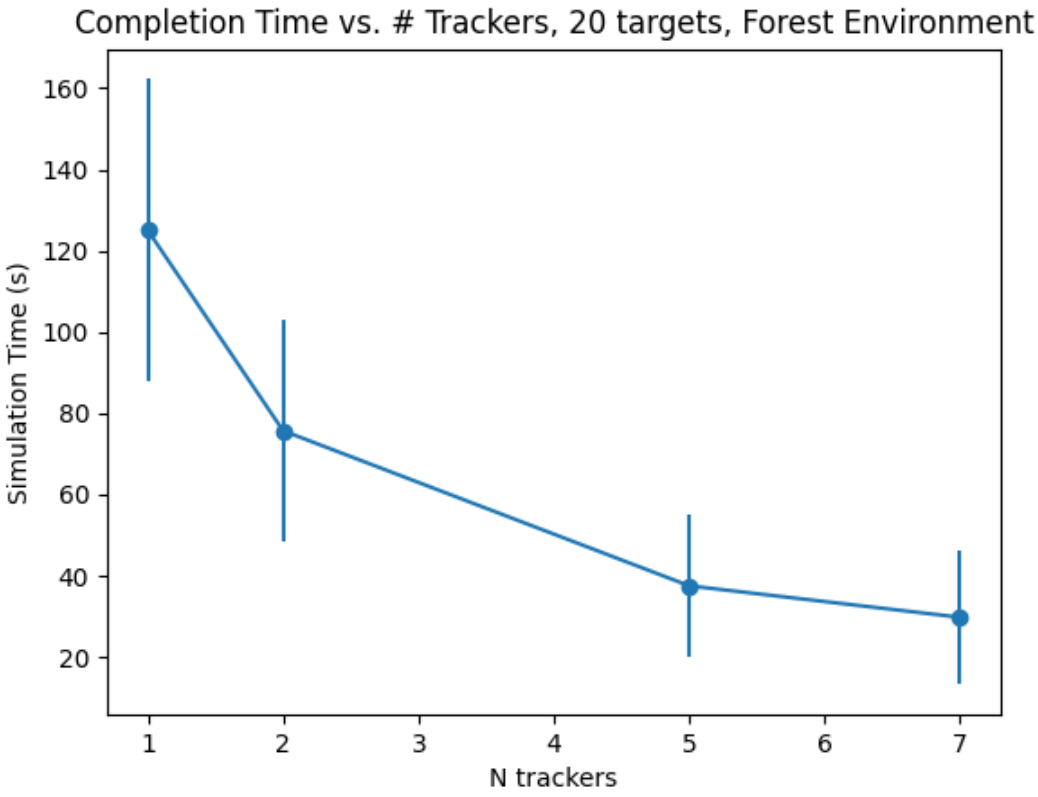


Figure 4-10. Scaling of task completion time in the forest domain for an increasing number of agents.

Chapter 5

Conclusion

5.1 Conclusion

In this thesis, we demonstrated the potential for co-designing high- and low-level viewpoint-aware tracking algorithms to solve important tasks related to videography and multi-target tracking. We first demonstrated a practical algorithm for assigning a team of drones to capture a series of desired shots and locally optimizing the drone trajectories to ensure each shot is captured as well as possible. Our experiments have shown that the planning and control pipeline works on physical systems in the presence of obstacles and uncertainty over the videography target's trajectory.

In the second part of the thesis, we presented a planning and control algorithm for the multi-tracker, multi-target optimization problem. Our framework for combining ellipsoidal collision avoidance constraints with a viewpoint-based cost function enables teams of robots to efficiently partition the environment among themselves and track targets of interest while avoiding obstacles. Our method for optimizing vehicle motion within ellipsoidal regions of free space is straightforward to implement and is particularly well-suited for testing MPC algorithms applied in new problem domains.

5.2 Future Work

The insights from our work will enhance the capabilities of drone videography platforms and improve the efficacy of multi-target mobile information gathering systems. We look to extend these capabilities in future work in two main directions.

One important extension to our existing work is to consider more structured predictions of target motion. The current videography work makes few assumptions on the structure of prediction distributions, but it is only able to (approximately) optimize expected costs over the entire trajectory. We look forward to considering more restricted classes of multimodal distributions and working to understand how an assignment algorithm can reason about temporal correlation in the trajectory prediction to achieve better online adaptability. For example, the current trajectory prediction formulation is parameterized solely by time. If we instead consider what regions of the environment a subject of interest is attracted to, and the semantic relationships governing how and when a target would move between regions, the assignment algorithm can reason spatially about *where* to position the drones such that they can react efficiently rather than reasoning when a target will be at specific locations. Along similar lines, analyzing a minimum-regret formulation of this problem would lead to interesting insight for tasks in cluttered environments, where some possible target paths have no feasible viewpoints for a following drone, but we do not want to sacrifice the quality of task assignment along other trajectories just to try optimizing the difficult one.

A second interesting extension is to consider more tightly coupled coordination between tracking agents to account for a wider range of behavior of target agents. The target agents considered here have been ambivalent to the presence and motion of the tracking agents and generally less physically capable. When the targets of interest are adversarial or have greater ability than individual tracking agents, the tracking agents must develop specific strategies to maintain visibility. We expect

our free-space aware environment partitioning and online optimization of viewpoint costs to be useful in this pursuit, but further additions to account for strategy of the cooperating agents must be made to the online optimization.

5.3 Lessons Learned

The work presented in this thesis has resulted in several key takeaways. It has emphasized the importance of carefully selecting what can be computed offline ahead of time versus what should be computed online at runtime, and understanding the assumptions necessary to make this division sensible. The videography shot selection offloads a costly dynamic assignment to an initial offline phase, enabling a global optimization of drone shot assignment. The computational burden is lessened with the knowledge that the final objective will be locally optimized at runtime, so the expensive global assignment does not need to guarantee that the reference trajectory is feasible. On the other hand, the offline optimization relies on the fact that the target's motion can be well-predicted over long horizons, which is reasonable under only a narrow range of tasks.

The work has also demonstrated the importance of obstacle parameterization for optimization-based control. The videography MPC implementation explicitly represents each obstacle as an ellipsoid, which causes issues in both defining obstacles (every obstacle must be designed as a union of ellipsoids) and scalability (MPC NLP size scales with environment complexity). By switching to a ellipsoidal parameterization of free space in the multi-target tracking work, MPC solution time is basically independent of obstacle complexity, at the expense of slightly more computation required to actually decompose the free space. In most cases, moving this computational burden outside of the higher-rate MPC problem is a large computational improvement.

This work has also exposed the complicated landscape of technical implementation challenges of an MPC algorithm. Developing an MPC implementation requires choosing NLP transcription method (single shooting, multiple shooting, collocation, etc.), NLP solution method (sequential quadratic program, interior point, etc.), and specific solver implementation. The range of free versus proprietary licenses, tradeoffs between general flexibility and tuning for low-memory applications, and differing online community support for the solvers makes implementing the MPC a tricky task. Our choice of Casadi/IPOPT as the solver was born of a desire to rapid development, free license, and clear documentation, but has come at the price of performance compared to more specialized solvers such as Force Pro (expensive, proprietary software) and Acados (still immature and sparse documentation).

5.3.1 Funding

This research was supported in part by the Office of Naval Research (ONR), Toyota Research Institute (TRI), Singapore University of Technology and Design Project Astralis, and the MIT Jacobs Presidential Fellowship. This thesis solely reflects the opinions an conclusions of its author, and not any other funding entity.

Appendix A

Deriving Occlusion Cost

We consider ellipsoidal obstacles defined as

$$O(M, \mathbf{r}_o) = \{\mathbf{v} \mid Mv + \mathbf{r}_o, \|\mathbf{v}\| \leq 1\}$$

for some positive definite shape matrix M. The cost of occlusion from an ellipsoid is most straightforward when we consider a coordinate system centered at the ellipsoid, and a transformation of the space such that the ellipsoid is a unit sphere. Thus we define

$$\begin{aligned}
\mathbf{r}_c^o &= -\mathbf{r}_{co}^o = M^{-1}(\mathbf{r}_c - \mathbf{r}_o), \\
\mathbf{r}_t^o &= M^{-1}(\mathbf{r}_t - \mathbf{r}_o), \\
\mathbf{r}_{ct}^o &= \mathbf{r}_t^o - \mathbf{r}_c^o, \\
r_{proj} &= \frac{\mathbf{r}_{ct}^{o^T} \mathbf{r}_{co}^o}{\|\mathbf{r}_{co}^o\|}.
\end{aligned}$$

As shown in Figure 3-2, the target is visible from the camera when the target is outside of the occlusion cone. Let $R(d)$ be the radius of the cone at a distance d from the camera along the occlusion cone's medial axis. The target's perpendicular distance to the medial axis must be larger that $R(r_{proj})$. We denote this perpendicular distance

D. The "occlusion distance" d_v, represents how far a target is inside the occlusion cone.

First we derive the radius of the cone as a function of distance from the camera along the cone's medial axis. We restrict our attention to a 2D slice of the cone (e.g. Figure 3-2) and denote the cone's slope as a, distance between the camera and obstacle as d, and t as the distance from the obstacle's origin, along the medial axis, where the radius of the cone is equal to the radius of the cone. At x coordinate $(d-t)$, the y coordinate must be $\pm\sqrt{1-t^2}$, as the obstacle radius is 1 in our transformed frame. Considering the y coordinate at the point where the cone intersects with the obstacle, we have that

$$a(d-t) = \sqrt{1-t^2}.$$

In addition, the cone only intersects the sphere at a single point, so it must be tangent at the intersection.

$$-\frac{d}{dt}a(d-t) = \frac{d}{dt}\sqrt{1-t^2} \tag{A.1}$$

$$a = \frac{t}{\sqrt{1-t^2}} \tag{A.2}$$

Note that in the first step we take the negative derivative of $a(d-t)$ because t is defined in the opposite direction as the other coordinates (increases to the left). Plugging a into the previous expression yields

$$(d-t)\frac{t}{\sqrt{1-t^2}} = \sqrt{1-t^2}$$

$$t(d-t) = 1-t^2$$

$$t = \frac{1}{d}$$

Solving for a in terms of d yields

$$a = \frac{\frac{1}{d}}{\sqrt{1 - \frac{1}{d}^2}} = \frac{1}{\sqrt{d^2 - 1}}$$

r_{proj} is the projection of $\mathbf{r}_{ct}$ onto the obstruction cone axis. The perpendicular distance from the cone's axis to the target is thus

$$\begin{aligned} D &= \left\| \mathbf{r}_{ct} - r_{proj} \frac{\mathbf{r}_{co}}{\|\mathbf{r_{co}}\|} \right\| \\ &= \sqrt{\|r_{ct}\|^2 - \frac{(\mathbf{r}_{ct}^T \mathbf{r}_{co})^2}{\|\mathbf{r}_{co}^2\|}} \end{aligned}$$

The radius of the cone at r_{proj} can be denoted

$$\begin{aligned} R &= \frac{1}{\sqrt{\|\mathbf{r}_{co}\|^2 - 1}} \cdot r_{proj} \\ &= \frac{\mathbf{r}_{co}^T \mathbf{r}_{co}}{\sqrt{\|\mathbf{r}_{co}\|^4 - \|\mathbf{r}_{co}\|^2}} \end{aligned}$$

The putative occlusion cost d_v is the amount by which the occlusion cone radius exceeds the distance of the target from the cone's centerline:

$$d_v = R - D = \frac{\mathbf{r}_{ct}^T \mathbf{r}_{co}}{\sqrt{\|\mathbf{r}_{co}\|^4 - \|\mathbf{r}_{co}\|^2}} - \sqrt{\|\mathbf{r}_{ct}\|^2 - \frac{(\mathbf{r}_{ct}^T \mathbf{r}_{co})^2}{\|\mathbf{r}_{co}\|^2}}$$

There are two small adjustments we make to d_v for use as a cost. First, we do not let it go below some threshold (usually zero or a negative number), as once the target is unoccluded there is little to gain by increasing separation from the occlusion cone. Second, we must ensure this only applies when the target is on the other side of the obstacle from the camera. Based on the intersection between the occlusion cone and

obstacle sphere calculated above, there is possible occlusion when

$$\mathbf{r}_{ct}^T \frac{\mathbf{r}_{co}}{\|\mathbf{r}_{co}\|} \geq \|\mathbf{r}_{co}\| - \frac{1}{\|\mathbf{r}_{co}\|}$$
$$\mathbf{r}_{ct}^T \mathbf{r}_{co} \geq \|\mathbf{r}_{co}\|^2 - 1$$

This leads to the full expression for the occlusion cost $c_{occlusion}$

$$c_{occlusion} = \begin{cases} \max(0, d_v)^2, & \mathbf{r}_{ct}^T \mathbf{r}_{co} \geq \|\mathbf{r}_{co}\|^2 - 1 \\ 0, & \text{otherwise} \end{cases} .$$

Bibliography

[1] A. Ray, A. Pierson, H. Zhu, J. Alonso-Mora, and D. Rus, "Multi-robot task assignment for aerial tracking with viewpoint constraints," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[2] M. Christie, P. Olivier, and J.-M. Normand, "Camera control in computer graphics," *Computer Graphics Forum*, vol. 27, no. 8, pp. 2197–2218, 2008. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01181.x

[3] M. Gleicher and A. Witkin, "Through-the-lens camera control," *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, p. 331–340, July 1992. [Online]. Available: https://doi.org/10.1145/142920.134088

[4] C. Lino and M. Christie, "Intuitive and efficient camera control with the toric space," *ACM Transactions on Graphics*, vol. 34, no. 4, pp. 82:1–82:12, July 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2809654.2766965

[5] C. Lino, M. Christie, R. Ranon, and W. Bares, "The Director's Lens: An Intelligent Assistant for Virtual Cinematography," in *ACM Multimedia*, Scottsdale, United States, Nov. 2011. [Online]. Available: https://hal.inria.fr/hal-00646398

[6] H. Jiang, B. Wang, X. Wang, M. Christie, and B. Chen, "Example-driven virtual cinematography by learning camera behaviors," *ACM Trans. Graph.*, vol. 39, no. 4, July 2020. [Online]. Available: https://doi.org/10.1145/3386569.3392427

[7] I. Mademlis, N. Nikolaidis, A. Tefas, I. Pitas, T. Wagner, and A. Messina, "Autonomous UAV Cinematography: A Tutorial and a Formalized Shot-Type Taxonomy," *ACM Computing Surveys*, vol. 52, no. 5, pp. 1–33, Sept. 2019. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3362097.3347713

[8] I. Karakostas, I. Mademlis, N. Nikolaidis, and I. Pitas, "Shot Type Feasibility in Autonomous UAV Cinematography," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 1937–1941, iSSN: 1520-6149.

[9] I. Mademlis, V. Mygdalis, N. Nikolaidis, M. Montagnuolo, F. Negro, A. Messina, and I. Pitas, "High-Level Multiple-UAV Cinematography Tools for Covering Outdoor Events," *IEEE Transactions on Broadcasting*, vol. 65, no. 3, pp. 627–635, Sept. 2019.

[10] S. M. Drucker and D. Zeltzer, "Intelligent camera control in a virtual environment," in *In Proceedings of Graphics Interface '94*, 1994, pp. 190–199.

[11] C. Gebhardt, S. Stevšić, and O. Hilliges, "Optimizing for aesthetically pleasing qadrotor camera motion," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–11, July 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3197517.3201390

[12] B. Sabetghadam, A. Alcántara, J. Capitán, R. Cunha, A. Ollero, and A. Pascoal, "Optimal Trajectory Planning for Autonomous Drone Cinematography," in *2019 European Conference on Mobile Robots (ECMR)*, Sept. 2019, pp. 1–7.

[13] T. Nageli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges, "Real-Time Motion Planning for Aerial Videography With Dynamic Obstacle Avoidance and Viewpoint Optimization," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1696–1703, July 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7847361/

[14] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges, "Real-time planning for automated multi-view drone cinematography," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–10, July 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3072959.3073712

[15] R. Bonatti, W. Wang, C. Ho, A. Ahuja, M. Gschwindt, E. Camci, E. Kayacan, S. Choudhury, and S. Scherer, "Autonomous aerial cinematography in unstructured environments with learned artistic decision-making," *Journal of Field Robotics*, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21931

[16] Q. Galvane, C. Lino, M. Christie, J. Fleureau, F. Servant, F.-l. Tariolle, and P. Guillotel, "Directing Cinematographic Drones," *ACM Transactions on Graphics*, vol. 37, no. 3, pp. 1–18, July 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3243123.3181975

[17] A. Bucker, R. Bonatti, and S. Scherer, "Do you see what i see? coordinating multiple aerial cameras for robot cinematography," 11 2020, preprint on webpage at https://arxiv.org/abs/2011.05437.

[18] T. Chung, G. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics," *Auton. Robots*, vol. 31, 11 2011.

[19] M. Aigner and M. Fromme, "A game of cops and robbers," *Discrete Applied Mathematics*, vol. 8, no. 1, pp. 1–12, 1984. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0166218X84900738

[20] R. Nowakowski and P. Winkler, "Vertex-to-vertex pursuit in a graph," *Discrete Mathematics*, vol. 43, no. 2, pp. 235–239, 1983. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0012365X83901607

[21] W. B. Kinnersley, "Cops and robbers is exptime-complete," *Journal of Combinatorial Theory, Series B*, vol. 111, pp. 201–220, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0095895614001282

[22] D. Bhadauria, K. Klein, V. Isler, and S. Suri, "Capturing an evader in polygonal environments with obstacles: The full visibility case," *The International Journal of Robotics Research*, vol. 31, no. 10, pp. 1176–1189, 2012. [Online]. Available: https://doi.org/10.1177/0278364912452894

[23] I. Mitchell, A. Bayen, and C. Tomlin, "A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[24] A. Pierson, Z. Wang, and M. Schwager, "Intercepting rogue robots: An algorithm for capturing multiple evaders with multiple pursuers," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 530–537, 2017.

[25] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-interscience New York, 1972, vol. 1.

[26] I. S. Khalil, J. Doyle, and K. Glover, *Robust and optimal control*. prentice hall, new jersey, 1996.

[27] D. P. Bertsekas *et al.*, *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont, 2000.

[28] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[29] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[30] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[31] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[32] M. Grant and S. Boyd, "Cvx: Matlab software for disciplined convex programming, version 2.1," 2014.

[33] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[34] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006. [Online]. Available: https://doi.org/10.1007/s10107-004-0559-y

[35] A. Forsgren, "On warm starts for interior methods," in *IFIP Conference on System Modeling and Optimization*. Springer, 2005, pp. 51–66.

[36] T. C. Lin and J. S. Arora, "Differential dynamic programming technique for constrained optimal control," *Computational Mechanics*, vol. 9, no. 1, pp. 27–40, Jan 1991. [Online]. Available: https://doi.org/10.1007/BF00369913

[37] R. Tedrake, "Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832)," chapter 10. [Online]. Available: http://underactuated.mit.edu/

[38] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *International Journal of Control*, vol. 93, no. 1, pp. 13–29, 2020.

[39] B. Houska, H. J. Ferreau, and M. Diehl, "Acado toolkit—an open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[40] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "Acados: A modular open-source framework for fast embedded optimal control," *arXiv preprint arXiv:1910.13753*, 2019.

[41] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (grampc)," *Optimization and Engineering*, vol. 20, no. 3, pp. 769–809, 2019.

[42] P. Sopasakis, E. Fresk, and P. Patrinos, "OpEn: Code generation for embedded nonconvex optimization," in *IFAC World Congress*, Berlin, 2020.

[43] J. B. O. Ravindra K. Ahuja, Thomas L. Magnanti, *Network Flows: Theory, Algorithms, and Applications*. The address: Prentice Hall, Inc, 1993.

[44] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[45] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[46] H. Zhu and J. Alonso-mora, "Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, apr 2019.

[47] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems," *Operations Research Letters*, vol. 2, no. 4, pp. 161–163, 1983. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167637783900482

[48] W. Cook, "Concorde TSP solver." [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/index.html

[49] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Workshop on the Algorithmic Fundamentals of Robotics*, vol. 107, 2014.

[50] I. Gilitschenski and U. D. Hanebeck, "A direct method for checking overlap of two hyperellipsoids," in *2014 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, 2014, pp. 1–6.

[51] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 42–49.

[52] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, "Aggressive quadrotor flight through cluttered environments using mixed integer programming," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1469–1475.

[53] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, 03 2006.