UNIVERSITY OF CALIFORNIA SAN DIEGO

Predictive Sampling-based Robot Motion Planning in Unmodeled Dynamic Environments

A Thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Engineering Science (Mechanical Engineering)

by

Javier Matias Ruiz

2019

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank professor Sonia Martínez for all the guidance and patience she has given me; for all her support during the ups and downs that I encountered during my time working on this project and thesis. I am truly grateful to have had her as my advisor.

I would also like to thank Troy Harden and Beth Boardman from LANL for their help and support in performing the research and experiments needed for this thesis (supported by LANL and approved for release under LA-UR-19-31029).

I thank my committee members, professor Jorge Cortés and professor Maurício de Oliveira, for their time in reviewing my thesis.

I also want to give additional thanks to Sonia and Jorge for giving me the opportunity as an undergrad to be at UC San Diego and to help out with the MURO lab's summer projects.

I want to thank my lab mates Parth, Aamodh, and Aaron, for their advice, for all the lunch breaks, and for their friendship, making time in the lab more enjoyable.

Lastly, I thank my parents and grandparents, my brother and sister, my aunts and uncles, and my girlfriend. I truly could not ask for more love and support from them, and appreciate all that they've done.

This thesis, in part, has been submitted for publication as it may appear in IEEE Robotics and Automation Letters, 2020, Ruiz, Javier; Boardman, Beth; Harden, Troy; Martínez, Sonia. The thesis author was the primary author of this paper.

ABSTRACT OF THE THESIS

Predictive Sampling-based Robot Motion Planning in Unmodeled Dynamic Environments

by

Javier Matias Ruiz

Master of Science in Engineering Science (Mechanical Engineering)

University of California San Diego, 2019

Professor Sonia Martínez, Chair

This thesis describes a predictive sampling-based algorithm for real-time robot motion planning to reach dynamic goals. The planner utilizes all available information about future obstacle and goal positions over a time window to select a path that approximately minimizes the time to reach this goal. Then, we integrate the proposed method with an online learning algorithm that predicts future goal and obstacle positions. Because future states are predicted by propagation, an incorrect model would result into a greater prediction error for large horizons. Thus, using a pool of candidate models, we utilize a Multiple Model Adaptive Estimation (MMAE) method with online parameter estimation to learn an appropriate model that keeps this error bounded. Several simulations show the efficacy of the proposed algorithms.

Chapter 1

Introduction

1.1 Motivation

The problem of path-planning for robots in dynamic environments has been one of great importance over the years. This is particularly relevant in collaborative scenarios where robots are used alongside humans, which requires the design of adaptive planners that can handle moving targets and obstacles. Of special interest is the case of manipulator-human collaboration, where a desired goal (human hand) is always close to a moving obstacle (human arm). The utilization of data-based models of uncertain systems for efficient planning is gaining significant attention. Motivated by this, we aim to develop a new data-driven predictive planning algorithm that can learn paths to moving targets with unmodeled dynamics when close to a dynamic obstacle.

1.2 Objectives

We approach predictive path planning for dynamic environments in two parts. First, we propose path planning algorithms for changing environments. We do this under the assumption that future states of the environment (goal and obstacle position) are known. Then we drop the assumption about the future states being known, and apply an adaptive method to predict these states when the model of the environment is not explicitly known.

1.2.1 Planning for Dynamic Environments

First, we assume that the motion of the goal is known and non-adversarial. Based on existing algorithms for planning with moving goals and obstacles, we propose a reactive planning algorithm for these type of environments called the Moving Goal Tree (MGT) algorithm. Then, based on dominance-based pursuit-evasion problems, we adapt the MGT to incorporate information about future positions of the goal and obstacle. We call this new algorithm the Predictive Moving Goal Tree (PMGT) algorithm.

The proposed PMGT algorithm adapts an initial RRT* tree into a sequence of trees that are rooted at future goal positions. Each tree approximates optimal paths to a future position in the presence of obstacles over a time window. The robot then chooses to move to a next configuration on the path that allows to reach the future goal position simultaneously or earlier than the goal. We establish how this results in a capture condition that converges to the robust optimal capture solution when the planning cost is given by the time to capture goal, and as the number of nodes and time window go to infinity.

1.2.2 Making Predictions for Unmodeled Systems

The second part of the thesis focuses on predicting the future states of the environment when its dynamics are unmodeled. We consider the case where the dynamics of the environment can be described by one of N possible parametrized models, with identity of this model and its parameters being unknown. Our approach combines simultaneous state and parameter estimation by way of a nonlinear Kalman Filter with a multiple model estimation method. This allows us to estimate the parameters of multiple models online alongside their weights, getting rid of the need to have an individual model for each parameter candidate.

1.3 Organization

This thesis is organized as follows. Chapter 2 discusses background for the methods introduced in the thesis. The RRT* and GT algorithms are outlined, and the concept of dominance in pursuit-evasion games is discussed. Finally, this chapter discusses the Kalman Filter algorithm, and the use of estimators in Multiple Model Adaptive Estimation. In Chapter 3, we introduce the Moving Goal Tree (MGT) and Predictive Moving Goal Tree (PMGT) algorithms. We give a Lemma that shows that under conditions commonly assumed in pursuit-evasion games, and assuming that future evader positions are known up to some horizon, that following our PMGT gives an approximate minimum time-to-capture. Chapter 4 discusses our method for predicting the state of the goal with unmodeled dynamics. Chapter 5 shows simulation results for our planning and prediction algorithms.

Chapter 2

Background

2.1 Sampling-based Planning

When performing path planning in continuous environments, sampling-based planning methods, such as the RRT* [1] and the Probabilistic Roadmap (PRM) [2], are widely used. These types of planners have been modified for use in dynamic obstacle environments. More recently, reactive planning algorithms such as the Goal Tree (GT) [3] and RT-RRT* [4], which are based on RRT*, restructure the tree by removing conflicting nodes when the obstacle configuration changes. Other algorithms use predictions of the obstacle motion in their algorithms, see e.g. [5] and [6], which bias their point sampling toward regions that are safe for the robot. However, these papers assume that the goal is stationary and the obstacle's dynamic model is known and independent from the goal.

Sampling-based planning algorithms are widely used for motion-planning problems dealing with robotics. These types of algorithms are popular for planning in continuous high-dimension spaces as they don't need to exhaustively search the entire space. Instead, they use a subset of randomly sampled points in the configuration to build a path from the desired start position to a goal. These types of planners are probabilistically complete, meaning that if a path between the desired points exists, the probability of finding one of these paths approaches 1 as the number of sampled points increases to infinity.

2.1.1 Asymptotically Optimal Rapidly-exploring Random Tree

The Asymptotically Optimal Rapidly-exploring Random Tree (RRT*) [1] algorithm is based on the RRT algorithm. RRT* builds a tree (typically rooted at the starting position) using each sampled point. Each node u in the tree has a position x associated with it, as well as a cost, denoted by $\text{Cost}(u)$. This cost is the current sum of all edge costs, $C_{\text{edge}}(\cdot)$, for each edge in the path from u to the root.

At each iteration, a point x_{rand} is randomly sampled. If there is some obstacle-free path between x_{rand} and an existing node on the tree with distance less than ε, a new node u_{new} is made at this position. Otherwise, a new point, x_{new}, is made by incrementing by ε along the path between the nearest node on the tree and x_{rand}, then u_{new} is made at this point. We find u_{parent} by searching a neighborhood around u_{new}, and choosing the node that minimizes the sum of the distance from u_{new} to u_{parent} plus the distance from u_{parent} to the root. More explicitly, u_{parent} is chosen according to

$$u_{\text{parent}} = \arg\min_{u \in \mathscr{U}} \Big(\text{Cost}(u) + C_{\text{edge}}(e_{\text{new,u}}) \Big), \tag{2.1}$$

where $\mathscr{U}$ is the set of nodes within the neighborhood of u_{new}. Then u_{new} is added to the tree as a child of u_{parent}.

The biggest difference between RRT* and RRT is that after u_{new} is added to the tree, a rewiring is done around u_{new}. For each node u in $\mathscr{U}$, we re-assign u_{new} as the parent

u if doing so would decrease their cost. More explicitly, for all $u \in \mathscr{U}$, if

$$\mathrm{Cost}(u_{\mathrm{new}}) + C_{\mathrm{edge}}(e_{\mathrm{new,u}}) \leq \mathrm{Cost}(u),$$

then u_{new} is assigned as the new parent of u.

The RRT* planner is outlined in Algorithm 1. The inclusion of finding the optimal parent according to (2.1), in addition to the rewiring process, makes RRT* asymptotically optimal. This means that as the number of nodes in the tree increases to infinity, the cost of the path found by RRT* approaches the cost of the optimal path. RRT* can continue running after finding a path to continue improving the cost of this path.

Algorithm 1 RRT*

```
Require: goal position $x_G$, robot position $x_R$, space $X$, iteration count $K$
 1: Initialize tree $T$
 2: $u_{\mathrm{root}} \leftarrow$ AddNode( pos=$x_G$, parent=NULL, $T$)
 3: while NumberOfIterations $< K$ do
 4:     $x_{\mathrm{rand}} \leftarrow$ Sample($X$)
 5:     $x_{\mathrm{new}} \leftarrow$ Extend($x_{\mathrm{rand}}, T$)
 6:     $u_{\mathrm{parent}} \leftarrow$ FindParent($x_{\mathrm{new}}, T$)
 7:     if $u_{\mathrm{parent}}$ then
 8:         $u_{\mathrm{new}} \leftarrow$ AddNode($x_{\mathrm{new}}, u_{\mathrm{parent}}, T$)
 9:         Rewire($u_{\mathrm{new}}, T$)
10:     end if
11: end while
12: return $T$
```

2.1.2 The Goal Tree algorithm

The GT algorithm [3] is based on the RRT*, is rooted at the desired goal position, and was designed for planning in an environment with changing obstacles. When an obstacle changes, or if a new obstacle is introduced to the environment, the tree is trimmed of all edges and nodes that are in conflict with the new obstacle configuration. Rather than checking each edge in the tree for conflict, a set of nodes,

$$\mathscr{U}_{\text{conflict}} = \left\{ u \;\middle|\; \|x_O - x_u\| \leq r_O + r_{\max} \right\},$$

is created. Here, x_O is the center of an obstacle, r_O is the radius of that obstacle, and $r_{\max}$ is the maximum edge cost in the tree. Each of these nodes are then removed from the tree, along with all of their descendants. The GT algorithm is outlined in Algorithm 2. Figure 2.1 shows a typical tree and resulting path computed from RRT, RRT*, and GT.

Algorithm 2 Goal Tree

```
Require: goal position $x_G$, robot position $x_R$, space $X$, iteration count $K$
 1: Initialize tree $T$
 2: $u_{\text{root}} \leftarrow$ AddNode( pos=$x_G$, parent=NULL, $T$)
 3: while NumberOfIterations $< K$ do
 4:     $x_{\text{rand}} \leftarrow$ Sample($X$)
 5:     $x_{\text{new}} \leftarrow$ Extend($x_{\text{rand}}, T$)
 6:     $u_{\text{parent}} \leftarrow$ FindParent($x_{\text{new}}, T$)
 7:     if $u_{\text{parent}}$ then
 8:         $u_{\text{new}} \leftarrow$ AddNode($x_{\text{new}}, u_{\text{parent}}, T$)
 9:         Rewire($u_{\text{new}}, T$)
10:     end if
11:     if ObstaclesChanged( ) then
12:         TrimTree($T$)
13:     end if
14: end while
15: return $T$
```

2.2 Pursuit-Evasion Games

In devising an efficient solution, it makes sense that the planner is adapted to reach the goal in the smallest possible time accounting for all its future locations. This connects with the theme of pursuit-evasion games in the presence of dynamic obstacles. Dominance regions [7] are a main tool to identify which areas of the environment are reachable by

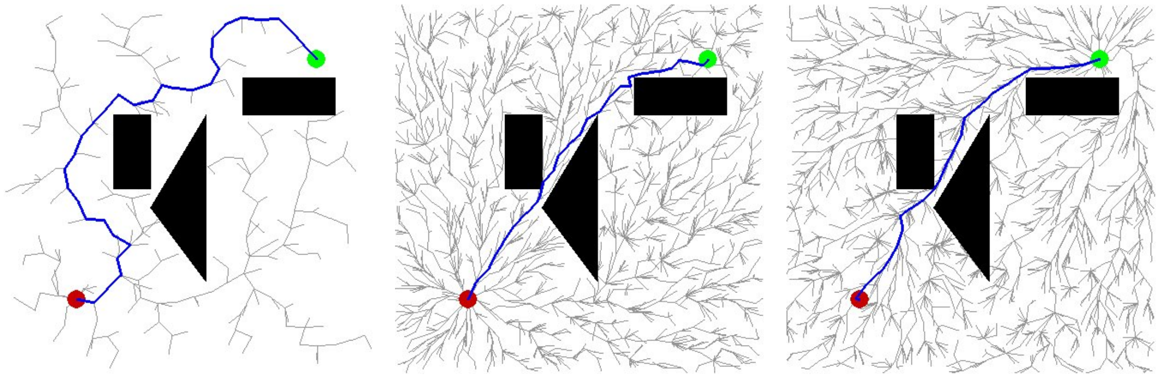

(a) RRT (b) RRT* (c) GT

Figure 2.1: Comparison of typical trees created by the RRT (a), RRT* (b), and GT (c) algorithms. The robot starting position and goal position are indicated by the red and green dots, respectively. The black shapes represent the obstacles in the environment. The RRT was stopped after a path was found; RRT* and GT were stopped after 500 nodes were added to the tree.

a player before any other, and can be used to define appropriate motion strategies. Thus, while pursuers try to catch the evaders as quickly as possible, evaders can aim to stay in their dominance region for as long as possible, see [8, 9]. While one can get inspiration from these techniques, this approach can be too conservative for collaborative scenarios (goals are not adversarial) and (unnecessarily) hard from a computational perspective.

In order to reach the moving goal in the smallest possible time, we explore the idea of dominance regions used in pursuit-evasion problems [7]. In relating our current problem to pursuit evasion, we say that our robot is analogous to the pursuer, and the moving goal is analogous to an evader, with the human being some dynamic obstacle in the environment.

We assume that the pursuer's maximum speed is larger than that of the evader's, and that both the pursuer and evader are subject to the same collision constraints with respect to the obstacle. We also assume that the evader and obstacle positions are known for future discrete time steps up to some horizon b. We note that we do not strictly treat this problem as a pursuit- evasion game, since our evader is not adversarial, and is not trying to maximize the time to capture.

By definition, the pursuer's dominance region consists of all points that the pursuer can reach before the evader does, given their current positions and their maximum velocities. Likewise, the evader's dominance region is defined by the points that the evader can reach before the pursuer. Let $m = \frac{v_p}{v_e}$ be the ratio of the maximum pursuer velocity to that of the evader, and let x_e and x_p be the positions of the evader and pursuer, respectively.

When there are no obstacles, the boundary between both players' dominance regions is given by the Apollonius circle. This circle is defined as the locus of points x, such that,

$$\frac{\|x_p - x\|}{\|x_e - x\|} = m, \tag{2.2}$$

where $\|\cdot\|$ is the Euclidean norm [10]. Given a fixed velocity of the evader, the pursuer's optimal strategy consists of moving straight to the point x_a on the Apollonius circle that intersects the trajectory of the evader. If $m > 1$, then the pursuer will catch the evader in finite time, the pursuer's motion results in a minimal capture time for this particular motion of the pursuer, and both the evader and pursuer reach this point simultaneously.

2.3 Estimation through Kalman Filtering

The previous references require full information of the dynamic models of moving objects. When dealing with unmodeled nonlinear human movement, algorithms that involve learning from expert advice may be particularly useful. Learning from experts [11] [12] [13] and multiple model estimation [14] [15] make predictions based on information from a pool of different models. The more accurate a model's prediction is, the more it is weighted in relation to the others in the pool. Therefore, these algorithms' performance is bounded by that of the best expert. The direct application of these algorithms to our setting poses several challenges: first, a long prediction horizon tends to amplify the propagation errors of the wrong dynamic model. Thus, the pool of experts needs to be typically large so that these

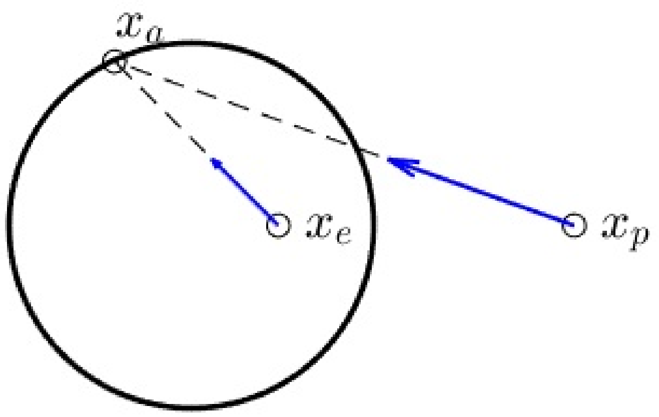


Figure 2.2: Visualization of Apollonius circle given pursuer and evader positions, x_p and x_e, and their maximum speeds v_p and v_e. When $m > 1$, equation (2.2) gives a circle that encompasses x_e. If the evader's velocity is known, the point x_a can be computed, which gives the point on the circle where the evader is expected to cross. This is the optimal point for the pursuer to travel to.

errors are kept small. If the general form of the dynamics is known up to some unknown parameters, then a pool can be created by discretizing the parameter space and making an individual model available for each parametric value [16]. This leads to a second problem as if the true dynamics is described by one of many parametric models, we can be lead to a prohibitively large number of candidate pool elements.

State estimation techniques are often used when dealing with a system that has process and measurement noise. A common type of estimation algorithm is the Kalman Filter (KF), along with some of its variants. The KF uses information about the process and measurement noise, the process model, and previous state measurements of the system to generate an estimate of the system state.

2.3.1 The Kalman Filter Algorithm

The KF algorithm provides an estimate of the state of a linear system. In particular we look at the discrete-time case. Our notation for estimates of state x is given by $\hat{x}^{(k_1|k_2)}$, which indicates the estimate of x at time step k_1 given information up to time step k_2. Consider the system

$$\begin{aligned} x^{(k+1)} &= F^{(k)}x^{(k)} + \eta^{(k)} \\ z^{(k)} &= H^{(k)}x^{(k)} + \nu^{(k)} \end{aligned} \tag{2.3}$$

where $\eta^{(k)}$ and $\nu^{(k)}$ are zero-mean Gaussian noise with covariances Q and R, respectively. We give the outline of the KF state estimation method.

The prediction of the state at the next time step is calculated using the system model,

$$\hat{x}^{(k|k-1)} = F^{(k)}\hat{x}^{(k-1|k-1)}, \tag{2.4}$$

and a prediction of the error covariance, P, is given by

$$P^{(k|k-1)} = F^{(k)}P^{(k-1|k-1)}F^{(k)^T} + Q. \tag{2.5}$$

Then the innovation and the Kalman gain are computed:

$$y^{(k)} = z^{(k)} - H^{(k)}\hat{x}^{(k|k-1)}, \tag{2.6}$$

$$K^{(k)} = P^{(k|k-1)}H^{(k)}\left(H^{(k)}P^{(k|k-1)}H^{(k)^T} + R\right)^{-1}. \tag{2.7}$$

The final estimate of the state and covariance are given by

$$\hat{x}^{(k|k)} = \hat{x}^{(k|k-1)} + K^{(k)}y^{(k)}, \tag{2.8}$$

$$P^{(k|k)} = \left(I - K^{(k)}H^{(k)}\right)P^{(k|k-1)}. \tag{2.9}$$

2.3.2 The Extended Kalman Filter Algorithm

The Extended Kalman Filter (EKF) is a nonlinear variation of the KF. We consider the following system,

$$\begin{aligned} x^{(k+1)} &= f^{(k)}\left(x^{(k)}\right)+\eta^{(k)}, \\ z^{(k)} &= h^{(k)}\left(x^{(k)}\right)+\nu^{(k)}. \end{aligned} \tag{2.10}$$

Estimates of state x are computed in the same manner as the KF, using (2.4)-(2.9), where $F^{(k)}$ and $H^{(k)}$ are the Jacobian of the process function and measurement function, respectively.

$$\begin{aligned} F^{(k)} &= \left.\frac{\partial f^{(k)}}{\partial x}\right|_{\hat{x}^{(k|k-1)}}, \\ H^{(k)} &= \left.\frac{\partial h^{(k)}}{\partial x}\right|_{\hat{x}_j^{(k|k-1)}}. \end{aligned} \tag{2.11}$$

Unlike in the case of the linear KF, using an EKF gives a suboptimal estimation of the state when the model is nonlinear. This is due to the linearization of the process and measurement models to get locally approximate estimates.

2.3.3 Multiple Model Adaptive Estimation

Multiple Model Adaptive Estimation (MMAE) is an approach to estimating states using a pool of different models [16]. This approach is useful when one of the models in our pool matches the true model of the system, without knowing which model is the correct one. MMAE methods typically compute a weighted mean of different state estimates using each model. The weight associated with each model is often synonymous with the probability that the model is correct, based on measurements we take of the state.

Let us say we have the following system

$$x^{(k+1)} = g^{(k)}\left(x^{(k)}\right)+\eta^{(k)}, \tag{2.12}$$

whose state we want to estimate, and noisy measurements

$$z^{(k)} = x^{(k)} + \nu^{(k)}, \tag{2.13}$$

where $\eta^{(k)}$ and $\nu^{(k)}$ are zero-mean Gaussian noises with covariances Q and R respectively. We have a pool of N estimators, each based on one of N possible models, $g_j^{(k)}$, $j \in \{1,\ldots,N\}$. We assume each model is subject to its own unique process noise, with differing covariances for $\eta_j^{(k)}$, but each has access to the same measurements from (2.13). At every iteration k, we get an estimate of the current state $\hat{x}_j^{(k|k)}$ from each model. Each estimate has a weight $w_j^{(k)} \in [0,1]$ associated with it, initialized to $\frac{1}{N}$. The estimate $\hat{x}^{(k)}$ is a weighted mean of all $\hat{x}_j^{(k|k)}$, given by

$$\hat{x}^{(k)} = \sum_{j=1}^{N} w_j^{(k)} \hat{x}^{(k|k)}. \tag{2.14}$$

In the case where each model has a KF or EKF applied to it, we use the same weight update method as in [14] and [15]:

$$w_j^{(k)} = \frac{\tilde{h}_j(z^{(k)}) w_j^{(k-1)}}{\sum \tilde{h}_i(z^{(k)}) w_i^{(k-1)}}. \tag{2.15}$$

Here, $\tilde{h}_j(z^{(k)})$ is a Gaussian density function based on the innovation, $y_j^{(k)}$, and its covariance $S_j^{(k)} = H^{(k)} P_j^{(k|k-1)} H^{(k)^T} + R$,

$$\tilde{h}_j(z^{(k)}) = \frac{\exp(-\frac{1}{2}(y_j^{(k)})^T (S_j^{(k)})^{-1} y_j^{(k)})}{\sqrt{(2\pi)^{n_z} |S_j^{(k)}|}},$$

where n_z is the dimension of our measurement vector. Following this update rule, our weights end up being the probability that the corresponding model is the correct one, and thus the sum of all $w_j^{(k)}$ adds up to 1 for all k.

Chapter 3

Predictive Planning

In this chapter, we present the Moving Goal Tree algorithm, which builds on the GT algorithm of [3] and accounts for a dynamically moving goal. This is the basis for our main predictive planning algorithm, the PMGT, presented in second part of this chapter.

3.1 Sampling-Based Planning for Moving Goal

The GT algorithm is an RRT*-based algorithm where the tree is rooted at the goal rather than the starting position. A path is found when the tree expands to include the starting position, and, as more nodes are added to the tree, the path is refined and becomes asymptotically optimal.

In the Moving Goal Tree (MGT) algorithm, a node is added at the new goal position. This node is then set as the new root of the tree, becoming the parent of the previous goal node. We call the new goal node $u_{\mathrm{G,new}}$ and the previous goal node $u_{\mathrm{G,prev}}$, while the new edge between them is $e_{\mathrm{new,prev}}$, and the cost of this edge is denoted by $C_{\mathrm{edge}}(e_{\mathrm{new,prev}})$. The cost-to-come of each node in the tree is increased by $C_{\mathrm{edge}}(e_{\mathrm{new,prev}})$. If we denote the cost-to-come of each node, u, prior to adding the new goal as $\mathrm{Cost}_{\mathrm{prev}}(u)$, then the cost of each

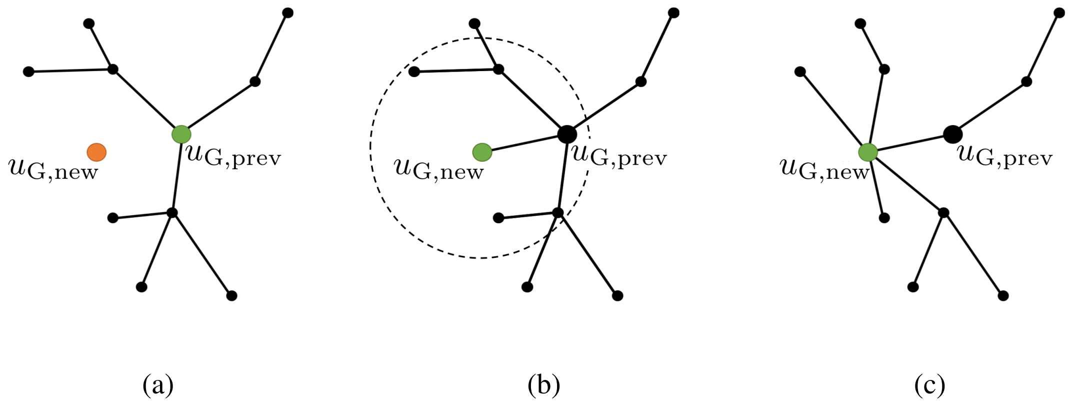


Figure 3.1: Evolution of a tree when a new goal is added. Figure 3.1a shows the tree with root represented by $u_{\text{G,prev}}$. In Figure 3.1b, the new goal node, $u_{\text{G,new}}$, is added, and is made the new root. A rewire is done in a neighborhood around $u_{\text{G,new}}$, indicated by the dashed circle. Figure 3.1c shows the tree after rewiring is done near the new goal.

node after updating the goal is

$$\text{Cost}_{\text{new}}(u) = \text{Cost}_{\text{prev}}(u) + C_{\text{edge}}(e_{\text{new,prev}}). \tag{3.1}$$

With this new cost function, a rewire is performed in a neighborhood of $u_{\text{G,new}}$, such that for all nodes u in this neighborhood, if $C_{\text{edge}}(e_{\text{new,u}}) < \text{Cost}_{\text{new}}(u)$, then the tree is rewired and $u_{\text{G,new}}$ becomes the parent of u. Figure 3.1 shows the process for a 2-D example, for a distance cost function. This restructuring is similar to that of RT-RRT*, though we root our MGT at the goal rather than the robot position. For reasons explained in the next section, we assume our goal to move more slowly than the robot. Therefore, by rooting the our algorithm at the goal, the motion of the root will induce less changes than if we had it rooted at the robot position.

Algorithm 3 shows the main part of the proposed method. Lines 8-18 capture the behavior of the GT algorithm. We use u_* to denote nodes in the tree, and x_* to denote positions. Extend( ) finds the closest node position to the sampled point x_{rand} and increments

Algorithm 3 Moving Goal Tree (MGT)

```
Require: goal position $x_G$, robot position $x_R$, space $X$
 1: Initialize tree $T$
 2: $v_{\text{root}} \leftarrow$ AddNode( pos=$x_G$, parent=NULL, $T$)
 3: while $x_R \neq x_G$ do
 4:     if GoalChanged( ) then
 5:         $x_G \leftarrow$ NewGoal( )
 6:         $u_{\text{root}} \leftarrow$ ChangeRoot($x_G, u_{\text{root}}, T$)
 7:     else
 8:         $x_{\text{rand}} \leftarrow$ Sample($X$)
 9:         $x_{\text{new}} \leftarrow$ Extend($x_{\text{rand}}, T$)
10:         $u_{\text{parent}} \leftarrow$ FindParent($x_{\text{new}}, T$)
11:         if $u_{\text{parent}}$ then
12:             $u_{\text{new}} \leftarrow$ AddNode($x_{\text{new}}, u_{\text{parent}}, T$)
13:             Rewire($u_{\text{new}}, T$)
14:         end if
15:     end if
16:     if ObstaclesChanged( ) then
17:         TrimTree($T$)
18:     end if
19:     if PathExists(T) then
20:         $x_R \leftarrow$ MoveRobot($T$)
21:     end if
22: end while
23: return $T$
```

Algorithm 4 ChangeRoot

```
Require: new root pos $x_G$, prev root node $u_{\text{G,prev}}$, tree $T$
 1: $u_{\text{G,new}} \leftarrow$ AddNode($x_G$, NULL , $T$)
 2: $u_{\text{G,prev}}$.parent $\leftarrow u_{\text{G,new}}$
 3: for all nodes $u$ in $T \setminus \{v_{\text{G,new}}\}$ do
 4:     $u.\text{Cost} \leftarrow u.\text{Cost} + C_{\text{edge}}(e_{\text{new,prev}})$
 5: end for
 6: Rewire($u_{\text{G,new}}, T$)
 7: return $u_{\text{G,new}}$
```

that position toward x_{rand} to get x_{new}. FindParent( ) searches in a neighborhood of x_{new} for the parent that would minimize the cost-to-go of the node at x_{new}. Rewire( ) iterates through nodes in a neighborhood of u_{new} and changes these nodes' parent to u_{new} if it results in a lower cost-to-go.

In the proposed MGT algorithm, we add the ChangeRoot method. In the cases where the goal moves, we move the root to the new goal, update the costs of the nodes in the tree according to (3.1), and rewire around the new goal, outlined in Algorithm 4.

Here, we expand on the MGT algorithm by incorporating knowledge about the future positions of the goal and obstacles in the environment. For the time being, we assume that the goal and obstacle trajectories are known.

3.2 Predictive Moving Goal Tree

We achieve a similar effect to the Apollonius pursuit strategy in discrete time with our sampling-based planner. We assume that we know the future evader positions $x_e^{(k+i)}$, where $i = 0$ gives the current position at time step k and $i \geq 1$ indicates the goal positions i time steps in the future up to horizon b. We define Δt to be a constant time difference between all time steps k. Then, a set of points reachable from $x_p^{(k)}$ with distance less than $iv_p\Delta t$ is computed. The point enclosed by the set with the lowest future time index to include an evader position $x_{e,k+i}$ is the optimal position to intercept. Figure 3.2 illustrates the idea. This results in an asymptotically optimal capture condition as we formalize next.

Let us define $i^* \in \{1,\dots,b\}$ to be the index of the future evader position $x_e^{(k+i^*)}$ we want the pursuer to move toward at time k. We choose i^* in order to minimize the time to

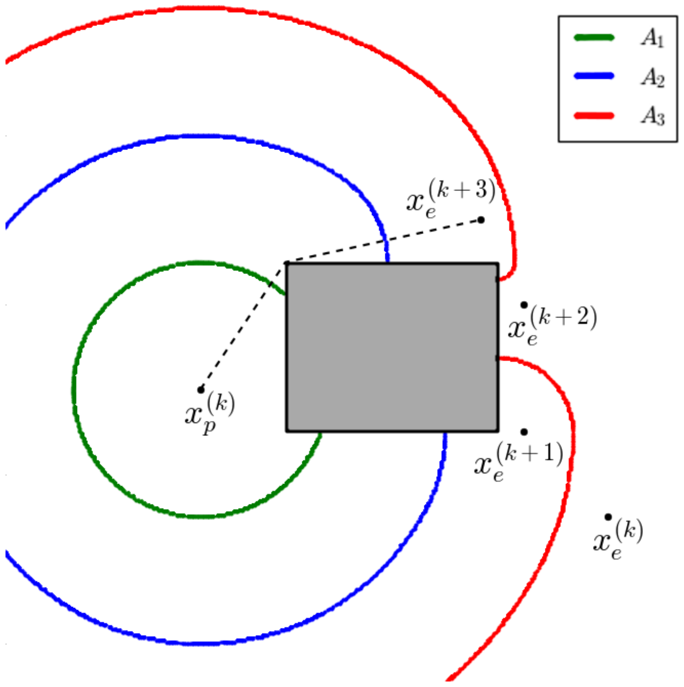


Figure 3.2: Visualization of pursuit given future evader positions. Each set A_i denotes all points reachable from $x_p^{(k)}$ in i time steps. Since $x_e^{(k+3)}$ is the earliest evader position (almost simultaneously) reachable by both the pursuer and evader (at $i = 3$), the optimal strategy is to take the shortest path there.

capture the evader:

$$i^* = \begin{cases} \min\limits_{i\Delta t \geq \tau_i} i & \text{if } \exists i, \text{ s.t.}: \ i\Delta t \geq \tau_i \\ \arg\min\limits_{i} (\tau_i - i\Delta t) & \text{otherwise} \end{cases} \tag{3.2}$$

where τ_i is the optimal time it takes for the pursuer to reach $x_e^{(k+i)}$ given its maximum speed, and $i\Delta t$ is the time for the evader to reach the same point. The first condition of (3.2) chooses $x_e^{(k+i^*)}$ to be the earliest position that the pursuer can reach before or at the same time as the evader. If no such point exists in our horizon, then the second condition chooses $x_e^{(k+i^*)}$ that minimizes the time between when the evader reaches the point and when the pursuer can.

Lemma 1 (Goal capture condition)**.** *Assume that the number of nodes in each of the sampling-based trees of the MGT algorithm is very large so that the path from the robot to a future evader position is practically optimal. Given future evader positions $x_e^{(k+i)}$, $i \in \{1, \dots, b\}$, we use* (3.2) *to choose $x_e^{(k+i^*)}$ for the pursuer to move to along one of the optimal paths. Assuming the pursuer's speed, v_p, is constant, and that $v_p > v_e$, then there exists some time step such that an i^* can be computed from* (3.2) *that satisfies*

$$\tau_{i^*} \leq i^* \Delta t. \tag{3.3}$$

Proof. Let us consider evader positions at some time steps $k+i$ and $k+i+1$. From the pursuer position, $x_p^{(k)}$, the lengths of the optimal paths to evader positions $x_e^{(k+i)}$ and $x_e^{(k+i+1)}$ are equal to $v_p\tau_i$ and $v_p\tau_{i+1}$, respectively. Since the evader does not necessarily move optimally, the length of the optimal path from $x_e^{(k+i)}$ and $x_e^{(k+i+1)}$ is upper bounded by $v_e\Delta t$.

By the triangular inequality, and using the fact that $v_p > v_e$, we get

$$v_p \tau_{i+1} \le v_p \tau_i + v_e \Delta t < v_p \tau_i + v_p \Delta t.$$

Dividing by v_p and rearranging terms gives us

$$\tau_{i+1} - (i+1)\Delta t < \tau_i - i\Delta t. \tag{3.4}$$

We note that the inequality (3.4) shows a sequence of time differences decreasing with each successive prediction.So, as new predictions are made for $x_e^{(k+i)}$ at each new time step k, the time differences will decrease until an i^* exists where (3.3) is satisfied. □

Remark 1. From (3.4), we see that $(\tau_i - i\Delta t)$ is decreasing in i. This means that the expression in the second condition of (3.2) can be approximated by b (provided the number of nodes in each tree is very large, it will reduce to b almost always). We rewrite (3.2) as the following:

$$i^* = \begin{cases} \min\limits_{i\Delta t \ge \tau_i} i & \text{if } \exists i, \text{ s.t.}: \ i\Delta t \ge \tau_i \\ b, & \text{otherwise} \end{cases} \tag{3.5}$$

Remark 2. From Lemma 1, we have that our algorithm results in a capture strategy when the cost function is related to time to reach the goal. Effectively, our algorithm is solving a robust optimization algorithm over a sliding window. If the time window was sufficiently large so that the robot can meet the goal point at one of the its future locations, we deduce that the algorithm will approximate the minimum time path to reach that goal location.

To implement the Apollonius strategy in the presence of obstacles in a bounded environment, we create copies of an initial tree and shift the root to the predicted goal positions. The obstacle and goal positions are updated every step k, with Δt being the length of each step. At every k, the robot receives the current goal position, $x_G^{(k)}$, the subset of the environment contained in the obstacle, $O^{(k)}$, and the position of the center of the obstacle, $x_O^{(k)}$. The robot is able to compute future information, $x_G^{(k+i)}$, $O^{(k+i)}$, and $x_O^{(k)}$, up to time $k+b$. At every time step, we create copies of the current tree T_0, shift the goal to $x_G^{(k+i)}$, and trim nodes that conflict with obstacles $O^{(k+i)}$. These new trees are denoted as T_i for $i \in \{1,\dots,b\}$. If the predicted goal positions are correct, then we can avoid copying T_0 and shifting the goal, by simply copying T_{i+1} from time step k into T_i at time step $k+1$. After doing this, observe that tree refinement is implemented.

Given a maximum robot velocity, the time τ_i needed to reach the goal position at time step $k+i$ along a path on T_i can be computed. We use (3.5) to choose i^* which will give the robot the best Tree T_* to determine its path. Algorithm 5 outlines this method for choosing a path to a future goal. The entire algorithm is outlined in Algorithm 6.

Algorithm 5 BestTree

```
Require: robot position $x_R$, Tree set $T$
1: $\tau_i \leftarrow$ TimeToReach($x_R, T_i$), $T_i \in T$
2: if $\exists i \in \{1,\dots,b\}$, s.t. : $i\Delta t \geq \tau_i$ then
3:     $i^* = \min_{\tau_i \leq i\Delta t} i$
4: else
5:     $i^* = b$
6: end if
7: return $T_{i_*}$
```

To prevent a conflict when the robot is near an obstacle, we place a circular buffer around each obstacle, with a radius of $d_O + \Delta t v_O$, where d_O and v_O are the diameter and maximum velocity of the obstacle, respectively. This prevents us from sampling in space where the obstacle might be in the next time step. The buffer and algorithm are illustrated in Figure 3.3. If the robot is inside this buffer, we choose the closest parent that exists outside of the buffer.

Algorithm 6 Predictive Moving Goal Tree

```
Require: goal positions $x_G^{(k)}$, obstacle positions $O^{(k)}$, robot position $x_R$, horizon $b$
 1: Initialize tree $T_0$
 2: $v_{\text{root},0} \leftarrow$ AddNode( pos=$x_{G,0}$, parent=NULL, $T_0$)
 3: while $x_R \neq x_G^{(k)}$ do
 4:     if EnvironmentChanged( ) [$\Delta t$ time has passed] then
 5:         for $i \in \{0, 1, ..., b\}$ do
 6:             if GoalPredictionCorrect( ) then
 7:                 $T_i \leftarrow$ Copy($T_{i+1}$)
 8:             else
 9:                 $T_i \leftarrow$ Copy($T_0$)
10:                 $u_{\text{root,i}} \leftarrow$ ChangeRoot($x_G^{(k+i)}, u_{\text{root,i}}, T_i$)
11:                 TrimTree($T, O^{(k+i)}$)
12:             end if
13:         end for
14:     else
15:         for $i \in \{0, 1, ..., b\}$ do
16:             $x_{\text{rand}} \leftarrow$ Sample( )
17:             $x_{\text{new}} \leftarrow$ Extend($x_{rand}, T_i$)
18:             $u_{\text{parent}} \leftarrow$ FindParent($x_{new}, T_i$)
19:             if $u_{\text{parent}}$ then
20:                 $u_{\text{new}} \leftarrow$ AddNode($x_{\text{new}}, u_{\text{parent}}, T$)
21:                 Rewire($u_{\text{new}}, T_i$)
22:             end if
23:         end for
24:     end if
25:     for $i \in \{0, 1, ..., b\}$ do
26:         if Conflict($x_R, O^{(k)}, T_i$) then
27:             $T_i \leftarrow$ ResolveConflict($x_R, O^{(k)}, T_i$)
28:         end if
29:     end for
30:     $T_* \leftarrow$ BestTree($x_R, T$)
31:     $x_R \leftarrow$ MoveRobot($T_*$)
32: end while
```

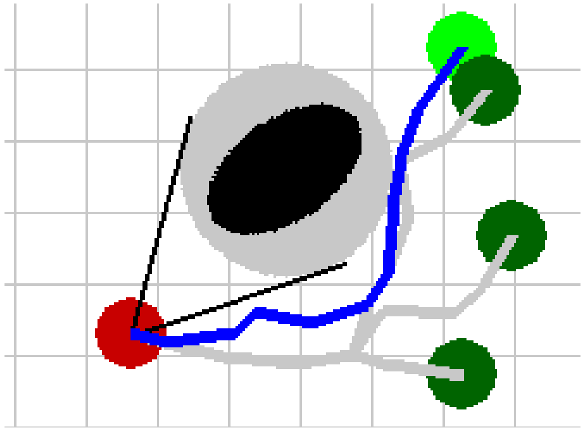


Figure 3.3: Planning with a moving obstacle. Light green target represents the current goal position, darker green targets represent future goal positions. The black ellipse represents the current obstacle, and the gray circle is a buffer representing the possible space the obstacle can be in at the next step.

Chapter 4

Predicting Goal Positions

In this chapter, we describe how we predict future goal and obstacle positions when their motion model is unknown. We combine the MMAE approach with an online EKF-based parameter estimation method.

4.1 Accounting for Obstacles in with MMAE

Because this method computes a weighted average of the estimations, the prediction space must be convex. This means we can not use this method directly to predict a goal position in the presence of obstacles. As mentioned before, we are considering this prediction method in the context of manipulator-human collaborative tasks (i.e. receiving something from/giving something to a human), where our goal will be near the dynamic obstacle. Therefore, we use the MMAE method to predict the dynamic obstacle positions $\hat{x}_O^{(k+i|k)}$. We then apply a translation to this position to get our predicted goal position $\hat{x}_G^{(k+i|k)}$ such that it is not in conflict with our obstacle.

4.2 Estimating Parameters in MMAE Models

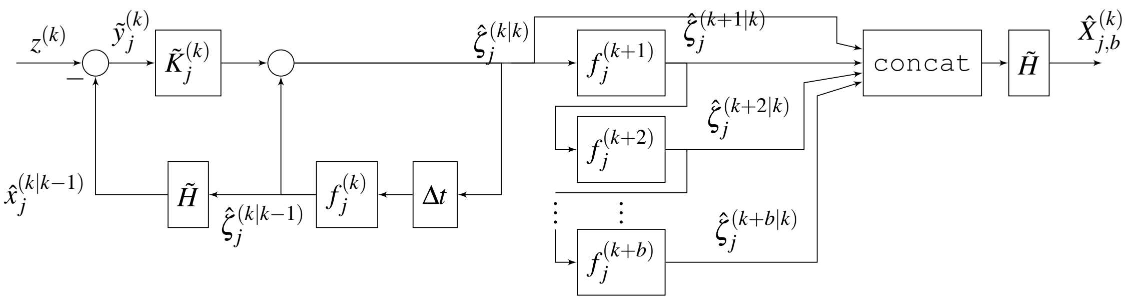


Figure 4.1: Diagram of estimator j. The EKF estimates the augmented state, then propagates it through the model to predict $\hat{\zeta}_j^{(k+i|k)}$. The `concat` block denotes concatenating $\hat{\zeta}_j^{(k+i|k)}$ horizontally into a matrix $\left[\hat{\zeta}_j^{(k|k)} \ \ \hat{\zeta}_j^{(k+1|k)} \ \dots \ \hat{\zeta}_j^{(k+b|k)}\right]$.

Assume that a parametric family of mappings is available to describe a system evolution. Then, one can aim to find the value of the best parameter that provides the best state predictions over time [17]. Assuming that the best parameter is stationary, this can be translated via the estimation of an augmented state that includes the original state and parameter vector. We apply this idea inside the MMAE filter to reduce the model pool size N to a set of structurally-different elements of a pool of experts.

First, we parameterize each model with a vector $\theta_j^{(k)}$:

$$x_j^{(k+1)} = g_j^{(k)}(x_j^{(k)}, \theta_j^{(k)}) + \eta_j^{(k)}$$

$$\theta_j^{(k+1)} = \theta_j^{(k)} + r_j^{(k)},$$

and measurement model where $\eta_j^{(k)}$ and $r_j^{(k)}$, are zero-mean Gaussian noise vectors with covariances Q_j and q_j respectively. To estimate the parameters of the model along with the state, we augment the state and parameters,

$$\zeta_j^{(k)} = \begin{bmatrix} x_j^{(k)} \\ \theta_j^{(k)} \end{bmatrix},$$

so that our augmented model looks like

$$\zeta_j^{(k+1)} = \begin{bmatrix} g_j^{(k)}(x_j^{(k)}, \theta_j^{(k)}) + \eta_j^{(k)} \\ \theta_j^{(k)} + r_j^{(k)} \end{bmatrix} = f_j^{(k)}(\zeta_j^{(k)}) + \begin{bmatrix} \eta_j^{(k)} \\ r_j^{(k)} \end{bmatrix}.$$

Our augmented noise vector is Gaussian, zero-mean with covariance

$$\tilde{Q}_j = \begin{bmatrix} Q_j & 0 \\ 0 & q_j \end{bmatrix}. \tag{4.1}$$

Then, we apply an EKF as usual.

We compute our prediction of the state and error covariance,

$$\hat{\zeta}_j^{(k|k-1)} = f_j^{(k)}(\hat{\zeta}_j^{(k-1|k-1)}) \tag{4.2}$$

$$\tilde{P}_j^{(k|k-1)} = F_j^{(k)} \tilde{P}_j^{(k-1|k-1)} (F_j^{(k)})^T + \tilde{Q}_j,$$

where $F_j^{(k)} = \frac{\partial f}{\partial \zeta}|_{\hat{\zeta}_j^{(k|k-1)}}$. Then we find the innovation

$$\tilde{y}_j^{(k)} = z_j^{(k)} - \tilde{H}\hat{\zeta}_j^{(k|k-1)}, \tag{4.3}$$

where $\tilde{H} = [I \quad 0]$, since we are not directly measuring the parameters. The Kalman gain is given by

$$\tilde{K}_j^{(k)} = \tilde{P}_j^{(k|k-1)} \tilde{H}^T (\tilde{H} \tilde{P}_j^{(k|k-1)} \tilde{H}^T + R_j)^{-1}.$$

A final estimate of the augmented state is given by

$$\hat{\zeta}_j^{(k|k)} = \hat{\zeta}_j^{(k|k-1)} + \tilde{K}_j^{(k)} \tilde{y}_j^{(k)}. \tag{4.4}$$

If our measurement model is (2.13), then to apply MMAE to this estimator, our weight update for the augmented case is the same as (2.15). To make predictions about future states up to step $k+b$, we take (4.4) and propagate it through each expert's model b times. We create a matrix from each of these predictions, defined as

$$\hat{Z}_{j,b}^{(k)} = \begin{bmatrix} \hat{\zeta}_j^{(k|k)} & \hat{\zeta}_j^{(k+1|k)} & \dots & \hat{\zeta}_j^{(k+b|k)} \end{bmatrix}.$$

We define the following

$$\hat{X}_{j,b}^{(k)} = H \hat{Z}_{j,b}^{(k)} = \begin{bmatrix} \hat{x}_j^{(k|k)} & \hat{x}_j^{(k+1|k)} & \dots & \hat{x}_j^{(k+b|k)} \end{bmatrix},$$

$$s_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad s_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \dots, s_b = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \quad s_i \in \mathbb{R}^{b+1}$$

such that each expert prediction at time step k is given by

$$\hat{x}_j^{(k+i|k)} = \hat{X}_{j,b}^{(k)} s_i. \tag{4.5}$$

Figure 4.1 shows a block diagram of our augmented EKF and model propagation. By substituting (4.5) into (2.14), we make our weighted predictions $p^{(k+i|k)}$ based on current parameter estimates.

Again, we use this framework to estimate the motion model of our obstacle, and predict $\hat{x}_O^{(k+i|k)}$ directly. We compute the expected positon of the goal by applying some translation to the obstacle, in the same manner described at the end of IV.A. Adapting the MMAE to include parameter estimation allows us to have multiple models of different structures without having to have different parameter realizations for each of them.

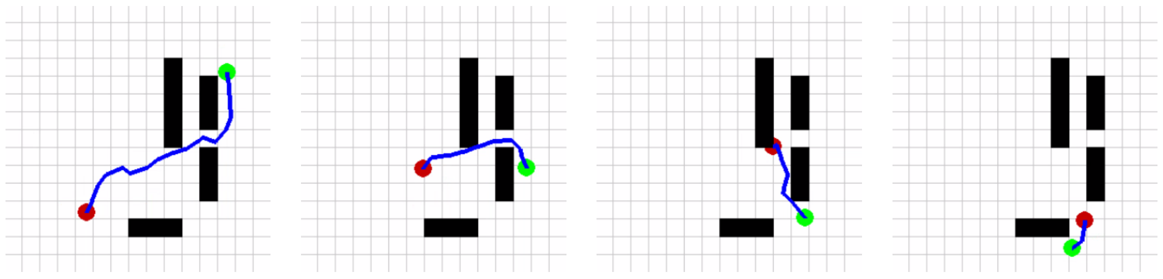
Chapter 5

Simulation Results

5.1 Moving Goal Tree

5.1.1 2D Non-Predictive Moving Goal Tree

The MGT is tested in the simple case of a point navigating through a 2D environment to catch a moving goal, as shown in Figure 5.1. The point is able to catch up the moving goal and find shorter paths around the obstacles as more nodes populate the tree.

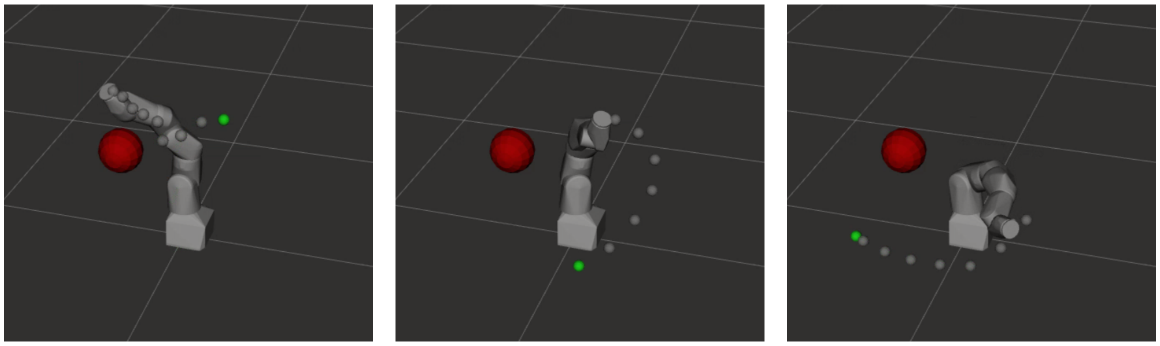


(a) (b) (c) (d)

Figure 5.1: Visual of robot (red) planning in real time to catch moving goal (green) in 2D space.

5.1.2 MGT with Robotic Manipulator

We also test the MGT algorithm in a simulation for a 7-DOF Motoman manipulator. The obstacle is static, and the manipulator moves around it to catch up to the moving goal configuration, shown in Figure 5.2.



(a) (b) (c)

Figure 5.2: Simulation of Motoman SIA5 robot moving along planned trajecotry using the proposed MGT. The robot navigates around the sphere to catch up to the green goal as it moves underneath the obstacle.

5.2 Expert Prediction

Figure 5.3 shows a simulation of a noisy nonlinear 1 - dimensional system. A pool of 3 models is used:

$$
\begin{aligned}
g_1^{(k)}(x^{(k)}) &= x^{(k)} + 0.1\cos(0.1k) \\
g_2^{(k)}(x^{(k)}) &= (x^{(k)})^{-2} - x^{(k)} \\
g_3^{(k)}(x^{(k)}) &= 0.9x^{(k)},
\end{aligned}
$$

where $g_1^{(k)}$ is the nominal model of the system. . As explained in Section 4.2, the algorithm learns how to assign the highest weight to the correct structural model. It can be seen how weighted predictions closely match the EKF predictions.

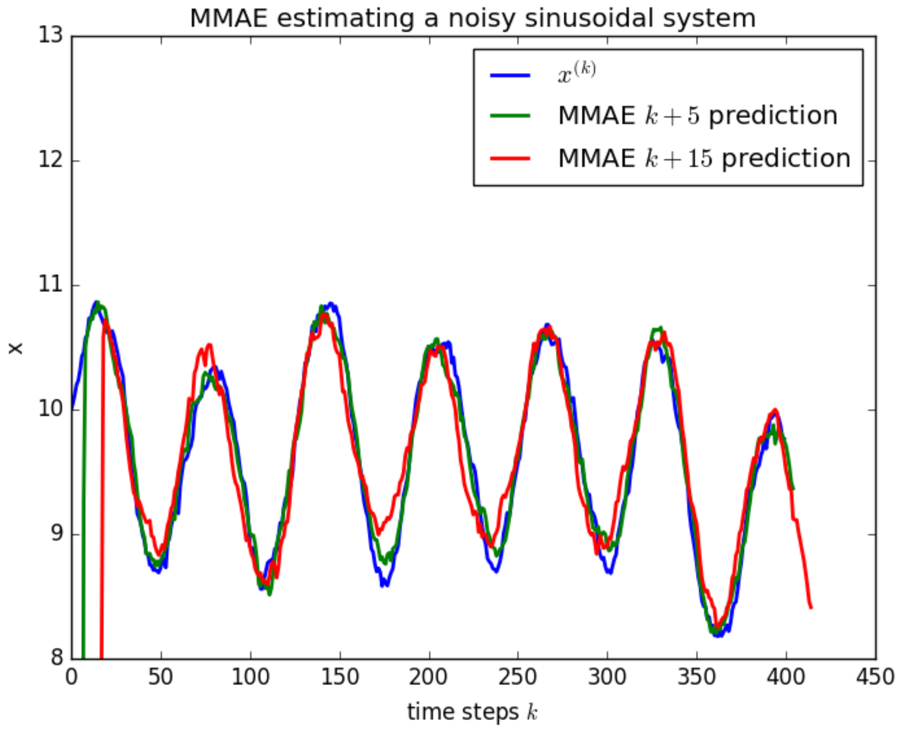


Figure 5.3: Simulation of MMAE, with one of the models matching the model of motion. The $k+5$ and $k+15$ prediction denotes the prediction of $x^{(k)}$ from 5 and 15 time steps ago.

5.3 Predictive MGT with Unknown Goal Positions

We simulate our combined algorithm in a 2-D environment with a moving goal and obstacle, shown in Figure 5.4. The goal is a fixed distance relative to the obstacle. A pool of 3 parameterized models is used to predict the obstacle position, $x_O^{(k)} = [x_x^{(k)} \;\; x_y^{(k)} \;\; x_\phi^{(k)}]^T$, where $x_x^{(k)}$ and $x_y^{(k)}$ are the 2-D coordinates, and $x_\phi^{(k)}$ is the obstacle's angle in the plane.

The parameter vector is two dimensional, given by $\theta = [\theta_a \;\; \theta_b]^T$. The models are given by:

$$g_1^{(k)}(x_O^{(k)}, \theta) = \begin{bmatrix} x_x^{(k)} + \sin(\theta_a x_\phi^{(k)}) \\ x_y^{(k)} + \cos(\theta_b x_\phi^{(k)}) \\ x_\phi^{(k)} + \frac{\pi}{18} \\ \theta_a \\ \theta_b \end{bmatrix},$$

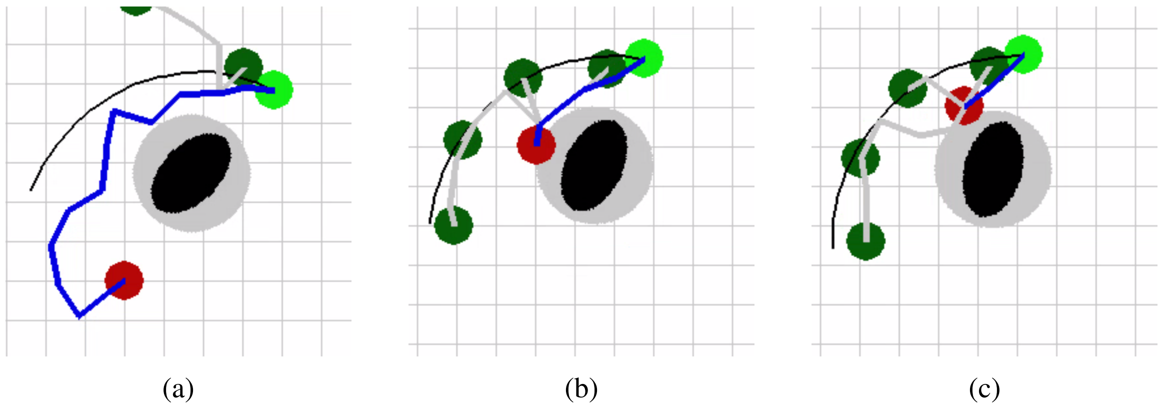
$$g_2^{(k)}(x_O^{(k)}, \theta) = \begin{bmatrix} \theta_a x_x^{(k)} \\ \theta_b x_y^{(k)} \\ x_\phi^{(k)} + \frac{\pi}{18} \\ \theta_a \\ \theta_b \end{bmatrix},$$

$$g_3^{(k)}(x_O^{(k)}, \theta) = \begin{bmatrix} x_x^{(k)} + \theta_a \\ x_y^{(k)} + \theta_b \\ x_\phi^{(k)} + \frac{\pi}{18} \\ \theta_a \\ \theta_b \end{bmatrix},$$

where $g_3^{(k)}$ is the nominal model with $\theta_a = \theta_b = -0.1$. The goal position is estimated from $x_O^{(k)}$ by

$$x_G^{(k)} = \texttt{rot}(x_\phi^{(k)}) \begin{bmatrix} 3 \\ 0 \end{bmatrix} + \begin{bmatrix} x_x^{(k)} \\ x_y^{(k)} \end{bmatrix}$$

where `rot` gives the standard rotation matrix, and the length of the obstacle is 2.5. The EKF learns the model of the obstacle and predicts future positions, with MMAE to appropriately weight the predictions of each expert at each time step. We see the predicted goal positions become more accurate over time.



(a) (b) (c)

Figure 5.4: Simulation of predictive MGT with MMAE algorithm to predict future goal and obstacle positions. The current goal is denoted by the lighter green color and future predicted goal positions denoted by darker green colors.

Chapter 6

Conclusion

We have proposed an online sampling-based planning algorithm to be employed in dynamic environments with a goal that moves in conjunction with an obstacle. The algorithm uses information about goal and obstacle configurations at future time steps to improve the time needed for the robot to catch the goal. We then propose a method to generate these predictions when there are various uncertainties about the motion model for the goal and obstacle. This method combines multi-model estimation with online parameter estimation for each model. For future work, we will investigate the efficient modification fo the PMGT to approximate a worst-case strategy in a pursuit-evasion approach, by dropping our assumption that the evader is non-adversarial, and making use of robust estimation methods.

This thesis, in part, has been submitted for publication as it may appear in IEEE Robotics and Automation Letters, 2020, Ruiz, Javier; Boardman, Beth; Harden, Troy; Martínez, Sonia. The thesis author was the primary author of this paper.

Bibliography

[1] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[2] 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.

[3] B. Boardman, T. Harden, and S. Martínez, “Improved performance of asymptotically optimal rapidly-exploring random trees,” *ASME Journal on Dynamic Systems, Measurement, and Control*, vol. 141, no. 1, 2018.

[4] K. Naderi, J. Rajamäki, and P. Hämäläinen, “RT-RRT*: A real-time path planning algorithm based on RRT*,” in *Proc. of the 8th ACM SIGGRAPH Conference on Motion in Games*. ACM, 2015, p. 113 –118.

[5] F. Damerow and J. Eggart, “Balancing risk against utility: Behavior planning using predictive risk maps,” in *IEEE Intelligent Vehicles Symposium*, 2015.

[6] S. Agarwal, A. K. Gaurav, M. K. Nirala, and S. Sinha, “Potential and sampling based rrt star for real-time dynamic motion planning accounting for momentum in cost function,” in *International Conference on Neural Information Processing*, 2018.

[7] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. Dover, 1999.

[8] Z. Zhou, W. Zhang, J. Ding, H. Huang, D. M. Stipanović, and C. J. Tomlin, “Cooperative pursuit with voronoi partitions,” *Automatica*, vol. 72, pp. 64 – 72, 2016.

[9] D. W. Oyler, P. T. Kabamba, and A. R. Girard, “Pursuit–evasion games in the presence of obstacles,” *Automatica*, vol. 65, pp. 1 – 11, 2016.

[10] C. Giovannangeli, M. Heymann, and E. Rivlin, "Pursuit-evasion games in presence of obstacles in unknown environments: towards an optimal pursuit strategy," in *Cutting Edge Robotics 2010*, V. Kordic, Ed. Rijeka: IntechOpen, 2010, ch. 4. [Online]. Available: https://doi.org/10.5772/10317

[11] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Information and Computation*, vol. 108, no. 2, p. 212–261, Feb. 1994.

[12] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: A meta algorithm and its applications," *Theory of Computing*, vol. 8, p. 121–164, 2012.

[13] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[14] C. Barrios, H. Himberg, Y. Motai, and A. Sadek, "Multiple model framework of adaptive extended kalman filtering for predicting vehicle location," in *IEEE Int. Conf. on Intelligent Transportation Systems*, 2006.

[15] V. Hassani, A. P. Aguiar, M. Athans, and A. M. Pascoal, "Multiple model adaptive estimation and model identification using a minimum energy criterion," in *American Control Conference*, 2009.

[16] A. P. Aguiar, "Multiple-model adaptive estimators: Open problems and future directions," in *European Control Conference*, 2007.

[17] L. Ljung, "Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems," *IEEE Transactions on Automatic Control*, vol. 24, no. 1, p. 36–50, February 1979.