Tracking Multiple Vehicles Constrained to a Road Network Using One UAV with Sparse Visual Measurements

Jared Joseph Moore
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Engineering Commons

BYU ScholarsArchive Citation
Moore, Jared Joseph, "Tracking Multiple Vehicles Constrained to a Road Network Using One UAV with Sparse Visual Measurements" (2020). Theses and Dissertations. 8887. https://scholarsarchive.byu.edu/etd/8887

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.
Tracking Multiple Vehicles Constrained to a Road Network

Using One UAV with Sparse Visual Measurements

Jared Joseph Moore

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Cameron K. Peterson, Chair
Randal W. Beard
D.J. Lee

Department of Electrical and Computer Engineering
Brigham Young University

Copyright © 2020 Jared Joseph Moore
All Rights Reserved
ABSTRACT

Tracking Multiple Vehicles Constrained to a Road Network Using One UAV with Sparse Visual Measurements

Jared Joseph Moore
Department of Electrical and Computer Engineering, BYU
Master of Science

Many multiple target tracking algorithms operate in the local frame of the sensor and have difficulty with track reallocation when targets move in and out of the sensor field of view. This poses a problem when an unmanned aerial vehicle (UAV) is tracking multiple ground targets on a road network larger than its field of view. We propose a Rao-Blackwellized Particle Filter (RBPF) to maintain individual target tracks and to perform probabilistic data association when the targets are constrained to a road network. This is particularly useful when a target leaves then re-enters the UAV’s field of view. The RBPF is structured as a particle filter of particle filters. The top level filter handles data association and each of its particles maintains a bank of particle filters to handle target tracking. The tracking particle filters incorporate both positive and negative information when a measurement is received.

We implement two path planning controllers, exhaustive receding horizon control (ERHC) and a neural net trained with deep reinforcement learning (Deep-RL), and compare their ability to improve the certainty for multiple target location estimates. The controllers prioritize paths that reduce each target’s entropy. While the ERHC achieved optimal stead-state estimates the Deep-RL controller identified more efficient sweeping search patterns when there is limited information regarding target locations. The neural net achieves $O(1)$ computational complexity during decision making but must first be trained on a given map.

In addition, we provide a theorem that calculates the lower-bound for the average-entropy of the RBPF. Particle Filter entropy is used as a unit of measurement as it gives a way of accurately comparing the precision of complex multi-modal estimates. This gives a reliable way of establishing the resources needed to accomplish mission objectives as well as providing a reliable method of determining the effectiveness of different multi-agent path planners.

Finally we outline results both in simulation and hardware. In simulation we obtained the results for our different path planners over 2000 Monte Carlo runs and show how the different path planners compare and measure up to the lower-bound of average-entropy. The results from a hardware test provide evidence that the ideas presented in this thesis hold true in an end-to-end solution.

Keywords: Particle Filter, Multi-Target Tracking
# TABLE OF CONTENTS

Title Page ........................................................................................................ i

Abstract .......................................................................................................... ii

Table of Contents .......................................................................................... iii

List of Tables ................................................................................................ iv

List of Figures ................................................................................................ v

Nomenclature ................................................................................................ viii

Chapter 1 Introduction ..................................................................................... 1

Chapter 2 Related Work .................................................................................. 3
  2.1 Rao-Blackwell Particle Filter ................................................................. 3
    2.1.1 Target Tracking ............................................................................... 3
    2.1.2 Data Association ........................................................................... 8
  2.2 Visual MTT .............................................................................................. 10
    2.2.1 Visual Frontend ............................................................................. 11
    2.2.2 R-RANSAC .................................................................................. 14

Chapter 3 Single Agent Predictive Path Planning ......................................... 16
  3.1 Dijkstra’s Algorithm ............................................................................. 16
  3.2 Exhaustive Receding Horizon Control .................................................. 18
  3.3 Deep-RL ............................................................................................... 20

Chapter 4 Lower Bound for Average Entropy of RBPF ............................... 26
  4.1 Minimum Time to All Targets ............................................................... 28
  4.2 Single Mode Entropy ........................................................................... 30
  4.3 Estimated Number of Modes ................................................................. 38
  4.4 Theorem for Lower Bound of Average Entropy ................................... 42

Chapter 5 Results ............................................................................................ 46
  5.1 Simulation Results ................................................................................ 46
  5.2 Hardware ............................................................................................... 50

Chapter 6 Conclusions ................................................................................... 56
  6.1 Future Work .......................................................................................... 57

References ...................................................................................................... 59
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The general processing parameters are used to modify the feed from the camera to maximize the capabilities of all the Frontend modules that are activated. Frame stride determines how many frames are skipped at a time during processing while the resize scale determines the resolution of the image being processed.</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>The Feature Motion module uses the motion of the features to determine which features should be passed into R-RANSAC for track identification. The velocities referred to are pixel velocities across the image not the inertial frame velocity of the targets.</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>The color detector applies a mask out everything in the image but the desired hue range. The blob size further refines the mask by eliminating noise in the subsequent image.</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Parameters used in Visual MTT R-RANSAC to identify features for target tracking.</td>
<td>15</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 The single-target particle filter maintains a finite number of hypotheses, even after the vehicle has traveled some distance since being seen. Particles are plotted as transparent dots to indicate density. The blue diamond shows true target position, the star shows the UAV position, and the dashed circle delineates the UAV’s field-of-view $F$. 4

2.2 Visual MTT Frontend passes features to R-RANSAC which identifies different tracks, object location and identification in the camera frame, and outputs the track data. 11

3.1 Using the naïve value function in Equation (3.1), the UAV exclusively tracks a single target because of the closely packed particles of that target. Ideally the UAV should weight the target with highest entropy more heavily. Particles are plotted as transparent dots to indicate density. The diamonds shows true target positions, the star shows the UAV position, and the dashed circle delineates the UAV’s field-of-view $F$. One target is represented in blue while the other is green. 17

3.2 Using edge values weighted by target entropy, the UAV prioritizes the blue particles over the green since the entropy of blue is larger than that of green. 19

3.3 Each epoch the reward over random scenarios is recorded. The average reward is kept to track training progress. The dataset of rewards is then trained over the shuffled dataset using the value network as a reference. Once the policy network has been trained the value network is trained on the same dataset using the updated policy network as a reference. 21

3.4 The top layer of the neural net is equal to the size of the feature space which in the case of our map is 574. Each subsequent layer is reduced by 40% until the bottom layer which is equal to either the action space in the case of the policy network or a single neuron to estimate the value of the state for the value network. 23

3.5 This figure depicts the neural net’s reward function over the training time. Within the first 80 epochs the neural net has reached an average reward of 80 units on random simulations. At this point it is outperforming a random path planner but the results are not competitive. Since the learning has hit a plateau, it takes 500 more epochs to bridge the gap to the point where it is performing at the level shown in Figure 5.3. 24

4.1 In the case of five equally-spaced targets the shortest round-trip distance is a regular polygon. These four polygons represent all possible polygon sizes. Taking the side length $S_{avg}$ gives us the average distance between targets on the minimum round-trip path shown in Equation (4.5). 28

4.2 This graph shows the error between the positional variance estimate given in Equation (4.12) and the positional variance computed in simulation when running without a positive sighting for $t = 55s$. For each value of $\sigma_v^2$ the error in $\sigma_p^2$ is calculated over 100 Monte Carlo runs. 33

4.3 The predicted variance growth calculated using Equation (4.12) and the average variance from 100 Monte Carlo simulations are plotted for a $t = [0,55]$s time window. As expected from the results in Figure 4.2 the two plots line up perfectly. 34
4.4 This graph plots the relationship between the size of the map and the average minimum variance for different measurement covariance values. Increasing $R$ shows less trust in the sensors and results in less effective positive updates. Increasing the map size (X-axis) allows for more time between positive updates and also results in a larger minimum variance $\sigma^2_p$. .......................................................... 36

4.5 Comparison of simulated entropy generated by 100 Monte Carlo runs with the estimate of the entropy from Equation (4.22). This graph shows that for a single mode the estimate matches the simulation’s average. .................................................. 37

4.6 Mode Splitting Analysis for a 3x3 map. A 3x3 map is a grid comprising of nine nodes in a 3x3 layout with each edge equidistant. .......................................................... 38

4.7 Probability of mode existence $P_D$ for map 4.6a over time $t = [0,20]$ s. The probability of a mode that contains $D$ percent of the particle filter is represented by the different lines plotted. As time progresses the probabilities of larger modes decrease as they evolve into smaller modes with increasing likelihood. .................................................. 41

5.1 Simulation tracking two targets on a 3x3 city block map. Here the UAV is drawn toward the blue target because its estimate has higher entropy. ...................... 47

5.2 Entropy while simulating the tracking of two targets on a 3x3 city block map. In region A, the UAV is searching for targets. In region B, the UAV has located the first target and is looking for the second. In region C, the UAV is trying to minimize entropy across both targets. .......................................................... 48

5.3 The ideal scenario and the lower bound are included for comparison. In the ideal scenario the UAV is always aware of the actual positions of the targets and flies directly between the two. The Lower Bound describes the average lower bound for entropy. We can see that the neural net path planner dropped in entropy faster than the exhaustive path planner. This is because the neural net learned efficient search patterns for uniformly distributed particles while the exhaustive path planner only made efficient decisions after it knew the general locations of the targets. ........................................ 49

5.4 Snapshot of the scenario implemented in hardware at 185s. The red asterisk represents the quadcopter. The green and blue diamonds are the current locations of each of the two vehicles. As can be seen, the green particle cloud is closely aligned with the actual position of its target. And while there are multiple modes for the blue target representing different estimates, the blue target is closely aligned with one of its modes. The black star and circle represents the estimate and error margin of a target sighting, which when funneled through our data association algorithm, performs a positive update on the green target. .......................................................... 50

5.5 The quadcopter runs a TX2 on board as well as an oCam. The body of the quadcopter is a DJI Matrice 100 and uses ROScopter as an onboard flight controller and an F4 board to house ROSflight. .......................................................... 51

5.6 The onboard computer is a TX2 running an autopilot based on the coordinates received from the motion capture cameras. The cameras locate the silver dots on the vehicle in order to give an estimate of the vehicle’s position. These coordinates are then converted into NED coordinates. The car drives Dubins paths along the predefined roadmap using an electronic speed controller (ESC) to control the motor. .............................. 52
5.7 The path of one of the vehicles is illustrated here. The Dubins path shown attempts to minimize the car’s deviation from the road network given the turn radius of the vehicle.

5.8 The base station, a laptop, runs the path planner and the RBPF for tracking the targets. The path planner uses the RBPF to decide the best course and sends waypoints to the autopilot while Visual-MTT processes the camera feed and relays target sightings to the particle filter. The autopilot is a waypoint follower. The RC cars run Dubins paths and both the RC cars and the quadcopter use the Motion Capture data to determine inertial frame coordinates.

5.9 The entropy for the two targets as well as the steady state entropy for the hardware test is shown. In region A, the UAV is searching for targets. In region B, the UAV has located the first target and is looking for the second. In region C, the UAV is trying to minimize entropy across both targets. The steady state entropy of around 3.207 which is just slightly higher than the lower bound of 3.11 units of entropy. This is expected as on a small map the ERHC planner will work comparatively to the ideal planner.
## NOMENCLATURE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A_F)</td>
<td>UAV’s field of view</td>
</tr>
<tr>
<td>(A_R)</td>
<td>2D area spanned by the road network</td>
</tr>
<tr>
<td>(A_{x,y}^q)</td>
<td>(x,y) coordinates of the (q)th agent</td>
</tr>
<tr>
<td>(B_L)</td>
<td>lower bound for entropy</td>
</tr>
<tr>
<td>(c_k)</td>
<td>data association of measurement (k) to target, (c_k = 1) means the target 1 was seen</td>
</tr>
<tr>
<td>(c)</td>
<td>index into nodes</td>
</tr>
<tr>
<td>(C)</td>
<td>number of nodes on the map</td>
</tr>
<tr>
<td>(d)</td>
<td>maximum number of edges leaving a node</td>
</tr>
<tr>
<td>(d_g)</td>
<td>line connecting two vertices on polygon</td>
</tr>
<tr>
<td>(D)</td>
<td>degree of a mode</td>
</tr>
<tr>
<td>(e)</td>
<td>an edge in the graph</td>
</tr>
<tr>
<td>(e_{m,n}^n)</td>
<td>edge the (n)th particle of target (m) is on</td>
</tr>
<tr>
<td>(e^n)</td>
<td>edge particle (n) is on</td>
</tr>
<tr>
<td>(E[k])</td>
<td>expected entropy at time (k)</td>
</tr>
<tr>
<td>(f)</td>
<td>number of edges on map</td>
</tr>
<tr>
<td>(F)</td>
<td>state transition matrix</td>
</tr>
<tr>
<td>(F_{\text{net}})</td>
<td>feature space for neural net</td>
</tr>
<tr>
<td>(</td>
<td>{\cdot}</td>
</tr>
<tr>
<td>(G)</td>
<td>number of polygons to estimate distance</td>
</tr>
<tr>
<td>(H)</td>
<td>number of history particles</td>
</tr>
<tr>
<td>(H_k)</td>
<td>observation model at timestep (k)</td>
</tr>
<tr>
<td>(I)</td>
<td>Identity Matrix</td>
</tr>
<tr>
<td>(j)</td>
<td>index into bins</td>
</tr>
<tr>
<td>(J)</td>
<td>number of bins in map</td>
</tr>
<tr>
<td>(k)</td>
<td>time step</td>
</tr>
<tr>
<td>(L)</td>
<td>number of lookahead steps</td>
</tr>
<tr>
<td>(L_{\text{map}})</td>
<td>length of map</td>
</tr>
<tr>
<td>(m)</td>
<td>index into (M) targets</td>
</tr>
<tr>
<td>(M)</td>
<td>Number of targets</td>
</tr>
<tr>
<td>(n)</td>
<td>index into (N) particles</td>
</tr>
<tr>
<td>(N)</td>
<td>Total number of particles per target</td>
</tr>
<tr>
<td>(N_{\text{eff}})</td>
<td>number of effective particles</td>
</tr>
<tr>
<td>(p_0)</td>
<td>initial particle weight</td>
</tr>
<tr>
<td>(P_D[k])</td>
<td>probability of mode of degree (D) existing at timestep (k)</td>
</tr>
<tr>
<td>(P_{\text{FA}})</td>
<td>probability of false positive</td>
</tr>
<tr>
<td>(P_{k_0})</td>
<td>kalman filter initial covariance</td>
</tr>
<tr>
<td>(P_k)</td>
<td>kalman filter covariance at timestep (k)</td>
</tr>
<tr>
<td>(P_k[b^h])</td>
<td>probability of a target existing in bin (b^h)</td>
</tr>
<tr>
<td>(P_{\text{null}})</td>
<td>probability of a false negative</td>
</tr>
<tr>
<td>(q)</td>
<td>index into agents</td>
</tr>
<tr>
<td>(Q)</td>
<td>Number of agents</td>
</tr>
<tr>
<td>(r)</td>
<td>radius of polygon</td>
</tr>
<tr>
<td>(R_e)</td>
<td>reward</td>
</tr>
</tbody>
</table>
$R$  
measurement covariance

$s^n$  
distance along edge the nth particle has moved

$\dot{s}^n$  
velocity of a particle

$S$  
average distance between nodes on graph

$S_{avg}$  
average distance between targets on the roundtrip path

$S_g$  
side length of the gth polygon

$T$  
roundtrip duration

$T_{df}$  
time it takes mode to encounter intersection

$T_Q$  
time between sightings for Q agents

$T_r$  
time it takes to traverse subdivision

$T_s$  
timestep mode comes into existence

$u_k$  
time of particle filter

$u^m_k$  
Entropy for target m at timestep k

$U$  
number of edges on given node minus 1

$v_e$  
velocity of target on $e$

$v_{e0}$  
average velocity of a particle along $e^n$

$V_a$  
UAV airspeed

$V_e^k$  
Dijkstra value of an edge at timestep k

$V_{e,w,k}$  
Weighted Dijkstra value for given edge at timestep k

$w^n_k$  
weight of the nth particle at timestep k

$x^n_k$  
nth particle at time k

$x^m_{k,n}$  
nth particle of target m at timestep k

$y_k$  
measurement at timestep k

$y^m_{1:k}$  
not sure if m should be involved

$Y_k$  
inverse of $P$

$E$  
Edges connecting the nodes in $G$

$E^{1:f}$  
edges 1-F on map

$G$  
Road Network

$N$  
Nodes in $G$

$N^{1:C}$  
x,y coordinates for nodes 1-C

$X^m_j$  
set of particles for the jth target.

$X_0$  
Initial set of particles

$X^m_{1:B}$  
particle filter for target m in bins 1-B

$\alpha$  
scale factor between $S_{node}$ and $S_{points}$

$\gamma_k$  
time of system at time k

$\gamma^m_k$  
Nonlinear weighting for target m at timestep k

$\gamma_{max}$  
maximum entropy

$\delta_t$  
timestep duration

$\Delta_t$  
$k - k_0$ alternatively $T/\delta_t$

$\varepsilon$  
maximum allowed change in policy between epochs

$\eta_k$  
sensor measurement noise

$\eta_{u,k}$  
entropy normalizing factor

$\eta_w$  
normalizing factor for particle weights

$\nu$  
particle velocity noise
σ²
σ²_p
σ²_{p,k}
σ²_{p,k_0}
σ²_v
σ_τ
τ_k

position variance
positional variance at timestep k
positional variance at initialization or average minimum variance
variance of particle velocity noise
standard deviation in the sensor measurement noise
target location at time k
CHAPTER 1. INTRODUCTION

Multiple-target tracking has a wide array of applications ranging from air traffic control [1] to following shoppers in a store [2]. Many approaches exist to track moving objects, vehicles, and pedestrians. Algorithms of particular interest include Multiple Hypothesis Tracking (MHT) [3], Probability Hypothesis Density (PHD) filters [4], Recursive RANSAC (R-RANSAC) [5], and their variants. Most applications of these algorithms constrain the area of interest to the field of view of the sensor deployed. Targets that move out of the field of view are usually forgotten and considered a new target when seen again.

Other research, where the area of regard is larger than the field of view of an UAV’s sensor, sometimes pose the situation as a search problem, as in [6] and [7]. It has been shown that incorporating additional information, such as road network information, improves the target’s state estimate [8]. Another powerful technique, employed by [6] and [9], is to incorporate negative information. Traditional localization would only update the target location belief if it were viewed. However, if the target is nowhere to be seen within the UAV’s field of view, this still gives some information about its location.

As an illustration, consider the case where a target could be in one of two possible locations. Searching one will reveal that the target is indeed there or must be at the other location. This negative information update proves valuable when an UAV cannot follow all of the targets all of the time.

We describe a method for incorporating road map information as well as a negative update to track multiple vehicles in an area larger than the UAV’s field of view in the presence of clutter and missed detections. We demonstrate the effectiveness of this approach despite the temporal sparsity of positive target measurements.

The tracker’s ability to maintain target certainty is defined in an theorem developed in this thesis. This theorem identifies the particle filter’s average-entropy lower-bound using the size and
complexity of the map, the number of targets tracked and the number of active UAVs. The lower bound of entropy is inversely proportional to the upper bound of the target location certainty. By knowing the target location certainty upper bound for a given number of UAVs, we can predict the number of UAVs needed to achieve a defined performance threshold for any given scenario. To the best of our knowledge no other work has developed an approach to address these issues.

This thesis builds on a previous conference publication [10] where we described the Rao-Blackwellized particle filter (RBPF) and the exhaustive receding horizon (ERHC) path planner. This thesis expands those results by incorporating a neural net path planner trained with deep reinforcement learning (deep-RL) and developing a theorem that gives the lower bound for the average entropy of the particle filter.

My contributions to this research consist of developing the path planners and the theorem for calculating the lower-bound of average-entropy described in this paper as well as integrating all parts described in this thesis together both in simulation and hardware.

The unmanned aerial vehicle (UAV) motion is controlled utilizing the road network constraint and the particle filter, which provides probability density information about the predicted target location. Numerical simulations demonstrate the comparable effectiveness between different controllers and show that the controllers improve target estimate certainty when compared to a random search pattern. Hardware results, completed in a motion capture room, show the efficacy of an end-to-end solution.

The remainder of the paper is organized as follows: Chapter 2 describes the RBPF as well as Visual MTT. The RBPF uses Visual MTT to update target estimates. The Path Planners described in Chapter 3 use the RBPF as a knowledge base. Chapter 4 defines an algorithm for identifying the lower bound for Entropy in the RBPF based on the resources available. Results comparing the path planners in simulation and demonstrating the effectiveness of the ERHC in hardware are shown in Chapter 5. Finally conclusions and future work are presented in Chapter 6.
CHAPTER 2. RELATED WORK

There are two components critical to my research that were developed by others, the RBPF and Visual-MTT. The RBPF presented in Section 2.1 is first defined in [10] and is used to estimate target locations. The theorem described in Chapter 4 uses the best estimate of the RBPF for entropy analysis. The second component, Visual-MTT, provides the target location in the camera frame as well as a target identification value based on when targets are first seen. Visual-MTT is described in Section 2.2 and is used to obtain the hardware results discussed in Chapter 5.

2.1 Rao-Blackwell Particle Filter

The RBPF is a multilayered particle filter. The top level particles each represent different possible states. In this fashion the particle filter is able to perform data association and reliably identify targets even after they pass outside the FOV. This filter is valuable when tracking multiple targets that move in and out of the field of view. The filter is also able to handle targets that appear identical to the agent’s sensor.

This section uses writing taken directly from a conference paper [10]. While the writing and RBPF development is primarily that of a colleague, I aided in the editing and also debugged the filter while integrating it with the other components used in this thesis. In Subsection 2.1.1 we describe a particle filter used to track a single target. We build on that particle filter in Subsection 2.1.2 in order to track multiple targets when the area of regard is greater than the sensor’s field of view.

2.1.1 Target Tracking

Consider the problem of tracking a single vehicle constrained to a road network using a UAV. The solution described in this subsection is used as a building block in the complete archi-
Figure 2.1: The single-target particle filter maintains a finite number of hypotheses, even after the vehicle has traveled some distance since being seen. Particles are plotted as transparent dots to indicate density. The blue diamond shows true target position, the star shows the UAV position, and the dashed circle delineates the UAV’s field-of-view $\mathcal{F}$.

architecture of tracking multiple vehicles with unknown data correspondence presented in Subsection 2.1.2.

**Single-Target Particle Filter**

The UAV encodes its belief of the target location using a particle filter (PF). Also known as Sequential Monte Carlo, the PF is a nonparametric implementation of the Bayes filter [11]. In contrast to a Kalman filter, the PF easily describes multimodal distributions, and it cleanly handles nonlinear motion and measurement models. These features are especially helpful in our scenario, where a target vehicle could be on any one of a number of roads after passing through an intersection. Figure 2.1 illustrates this scenario where the UAV has not seen the target for some time, and multiple good hypotheses exist.

Let $x$ denote the state of the target. We encode the initial belief of the state as a probability density function (pdf) $p(x_0)$ and draw the initial set of particles $\mathcal{X}_0$ from this distribution,

$$\mathcal{X}_0 \sim p(x_0).$$  \hfill (2.1)
Each particle is denoted as $x^n_k$, where the superscript $n$ denotes the $n$th particle and the subscript $k$ denotes time index. The set of particles at time $k$ is denoted as $\mathcal{X}_k = \{x^n_k \mid n = 1 \ldots N\}$. In this paper we chose a uniform distribution for $p(x_0)$, implying no prior knowledge about where the target may initially be in the search area.

Prediction is performed by sampling from the proposal distribution as

$$x^n_k \sim p(x_k \mid x^n_{k-1}). \tag{2.2}$$

When a measurement $y_k$ is received, each particle is assigned an importance factor as the ratio of the target distribution to the proposal distribution

$$w^n_k = \frac{p(x^n_k \mid y_{1:k})}{p(x^n_k \mid y_{1:k-1})}, \tag{2.3}$$

where $y_{1:k} = \{y_1, \ldots, y_k\}$ is the set of measurements. By applying Bayes’ rule to the numerator and factoring, we see that the importance factor, or weight, is proportional to the likelihood of the current measurement, given the particle’s current state,

$$w^n_k \propto p(y_k \mid x^n_k). \tag{2.4}$$

With the added constraint that all weights must sum to one, the proportionality is sufficient to calculate each particle’s weight. At each time instant, the particles are resampled with probability proportional to their weights, and their weights reset to the initial value $p_0 = \frac{1}{N}$.

We employ two techniques to better fit the posterior distribution $p(x_k \mid y_{1:k})$. First is the low variance resampling technique described in [12]. While resampling is necessary it can remove good particles and lead to particle deprivation. Low variance resampling helps mitigate this issue.

Another technique is to resample only as often as is beneficial, known as selective resampling [13]. The idea behind selective resampling is that if the particles were sampled from the true posterior, they would all have equal importance. The deviation from the true posterior can then be estimated by calculating the number of effective particles, a metric introduced by [14]

$$N_{eff} = \frac{1}{\sum_{n=1}^{N} (w^n)^2}. \tag{2.5}$$
Selective Resampling provides a way to determine when resampling is necessary. For example, the particles could be resampled when $N_{eff}$ drops below the threshold $\frac{2N}{3}$.

To calculate $N_{eff}$, a particle must keep track of its importance factor through each measurement update until resampling occurs, i.e.,

$$w^n_k = \eta_w w^{n-1}_k p(y_k | x^n_k),$$

(2.6)

where $\eta_w$ is a normalizing factor for the particle weights. In practice, these two techniques greatly reduce the chance that good particles are lost during resampling.

**Road Constraint**

Let $F(x)$ denote the field-of-view of the camera in the inertial frame when the UAV is at state $x$. Any time a target is outside the UAV’s field-of-view, its state can only be estimated using prediction. If the target could move unconstrained on the ground plane, the estimate would quickly disperse and become unusable. Constraining the target to a road network allows the UAV to accurately predict the possible places the target could go, even when it has not been seen for some time (see Figure 2.1).

We model the road network constraint as a directed graph

$$\hat{G} = (\mathcal{N}, \mathcal{E}),$$

(2.7)

where edges $\mathcal{E}$ represent road segments, and nodes $\mathcal{N}$ represent intersections or corners with known Cartesian coordinates. In the remainder of this paper we will not distinguish between the graph and its embedding in $\mathbb{R}^2$, i.e., $n \in \mathcal{N}$ will represent a point in $\mathbb{R}^2$ representing the inertial north-east coordinates of the node, and $e \in \mathcal{E}$ represents an inertially defined line in $\mathbb{R}^2$ with length $\text{len}(e)$.

Each particle encodes its current edge $e$ and how far along the edge it has traveled, denoted by path parameter $s$. Therefore the $n^{th}$ particle is given by,

$$x^n = (e^n, s^n), \quad e^n \in \mathcal{E}, \quad s^n \in \mathbb{R}.$$  

(2.8)
We use the notation $\mathcal{G}(x^n)$ to denote the real-world 2D Cartesian location associated with particle $x^n$.

**Target Motion Model**

The dynamic model of the target motion defines the proposal distribution shown in Equation (2.2). While virtually any dynamic model works with this architecture, this paper uses a constant velocity model with random perturbations. The particle’s position $s^n$ is propagated along the road as

$$\dot{s}^n = v_{e^n} + v,$$

where $v_{e^n}$ is some nominal velocity for the road segment $e^n$, (e.g., 15 m/s) and $v \sim \mathcal{N}(0, \sigma_v^2)$ is additive Gaussian white noise with standard deviation $\sigma_v$.

When a particle reaches an intersection (i.e., the end of an edge), in other words, if

$$s^n > \|e^n\|,$$  

then $e^n$ is randomly assigned with equal probability to one of the edges leaving that node, excluding the edge that returns to the previous node (i.e., no U-turns).

**Measurement Model**

When a sensor measurement is detected, the measurement likelihood model is a mixture of a Gaussian distribution corresponding to a true measurement coming from a target that is in the camera field-of-view $\mathbb{F}$ and a uniform distribution corresponding to a false alarm, specifically,

$$p(y_k | x_k^n) = (1 - P_{FA}) \frac{1}{2\pi\sigma_T} \exp\left(-\frac{1}{2\sigma_T^2} \|\mathcal{G}(x_k^n) - y_k\|^2\right) + \frac{P_{FA}}{A_R},$$

where $P_{FA}$ is the probability of false alarm, $\sigma_T$ is the standard deviation of the measurement noise, and $A_R$ is the 2D area spanned by the road network.
Negative Update

The lack of a sensor measurement is also information that can be used to update the target likelihood map because it indicates that the target is not in the sensor field-of-view $\mathcal{F}$. In this case, the negative measurement model is a mixture of two uniform distributions. The negative measurement model is then a mixture of two uniform distributions,

$$p(z_k | x^n_k) = \begin{cases} \frac{1-P_{FA}}{A_F}, & \mathcal{G}(x^n_k) \in \mathcal{F} \\ \frac{P_{FA}}{A_R-A_F}, & \text{Otherwise} \end{cases}$$

(2.12)

where $A_F$ is the area of the UAV’s field-of-view. When using a camera fixed with respect to the UAV body frame, $\mathcal{F}$ and consequently $A_F$, become a function of the UAV altitude and attitude. For the purposes of this paper, we assume a constant $P_{FA}$.

2.1.2 Data Association

Subsection 2.1.1 describes tracking a single target in the presence of clutter and missed detections. In this subsection, we extend the filter to track multiple vehicles.

Known Data Correspondence

Extending single-target tracking to multiple targets would be trivial if sensor measurements could give perfect data correspondence. That is if the sensor reported both the location and ID of the target. We assume that each target’s motion is independent of other targets, implying that the joint distribution can be factored as

$$p(\mathcal{X}^{1:M} | y_{1:k}) = \prod_{m=1}^{M} p(\mathcal{X}_k^m | y_{1:k})$$

(2.13)

where $M$ is the number of targets to be tracked, and $\mathcal{X}_k^m$ is the set of particles estimating the location of the $m^{th}$ target. In the case of perfect data correspondence the UAV simply maintains a separate particle filter for each target. As a positive measurement is received, it would only be
applied to the corresponding target. Negative measurements would be applied to the entire bank of trackers.

Unfortunately, it can be very difficult to visually differentiate two similar looking vehicles and so perfect data correspondence is not possible. Instead, we implement a Rao-Blackwellized Particle Filter (RBPF) to handle the data association in a manner similar to [9] and [15].

**Rao-Blackwellized Particle Filter**

Let \(c_{1:k}\) be the history of data associations; that is, \(c_k = m\) indicates that the measurement at timestep \(k\) corresponds to target \(m\), where \(m \in 1 \ldots M\) and \(M\) is the number of targets. If we let \(c_k\) be a random variable, then the joint distribution given a certain measurement is

\[
p(c_{1:k}, \mathcal{X}_{1:M}^k | y_{1:k}) = p(c_{1:k} | \mathcal{X}_{1:M}^k, y_{1:k}) \prod_{m=1}^M p(\mathcal{X}_m^k | y_{1:k}). \tag{2.14}
\]

We can approximate the right-hand side of Equation (2.14) using a Rao-Blackwellized particle filter. In this filter, each particle maintains its own joint target location distribution described in Equation (2.13), given a certain history of data associations. Collectively, the particles approximate the distribution over the history of correspondences.

Typically, the state is factored such that an optimal, closed form filter is used to reduce the dimensionality of the problem [16]. Common choices are the Kalman filter or the Hidden Markov Model (HMM) [9]. We found that we had sufficient computational power for each particle to maintain a bank of PF trackers as described in Section ?? and therefore chose not to discretize the problem to fit an HMM. The computational cost of this formulation is \(O(HMN)\), where \(M\) and \(N\) are as defined above, and \(H\) is the number of history particles. Our approach has an additional benefit that with a continuous state space, the road network of interest could be expanded without increasing the number of discrete states or reducing the discretization resolution, as would have been necessary with an HMM. Additionally, we are not bound to a linear Gaussian model, as with a Kalman filter.
Data Association Sampling

Assuming that targets are otherwise indistinguishable, data association must be determined from the estimated state of the targets. One approach is maximum likelihood (ML) association, where the best fit is chosen as

\[ \hat{c}_k = \arg\max_m p(y_k \mid c_k = m, \hat{c}_{1:k-1}, \mathcal{X}_k^m, y_{1:k}). \]  

(2.15)

We instead use Data Association Sampling (DAS) [12], where data associations are sampled from a categorical distribution according to their likelihoods,

\[ p_{c_k=m} \propto p(y_k \mid c_k = m, \hat{c}_{1:k-1}, \mathcal{X}_k^m, y_{1:k}). \]  

(2.16)

This can be approximated by summing the measurement likelihood over all particles for a given target and normalizing,

\[ p_{c_k=m} \approx \eta \sum_{n=1}^N p(y_k \mid x_k^{m,n}), \]  

(2.17)

where Equation (2.11) is used as the summand and \( x_k^{m,n} \) is the \( n \)th particle for the \( m \)th target at timestep \( k \). This approach can better retain multiple data association histories that have similar likelihood until they can be discriminated using later measurements.

The RBPF allows the UAV to properly associate measurements of targets, even when they leave and re-enter its field-of-view. However, these estimation techniques alone are not sufficient to maintain a good estimate of where all the targets are at any given time. The information from the filter must be used to feed a path planning algorithm so that the UAV can position itself to maximize target location certainty. The next section describes our approach to tracking and following multiple targets.

2.2 Visual MTT

Visual MTT, shown in Figure 2.2, is a tool developed at BYU for the purpose of tracking multiple targets with a camera. There are two components that manage this, the Visual Frontend and Recursive Random Sampling Consensus (R-RANSAC) [5]. The Visual Frontend processes
2.2.1 Visual Frontend

The Visual Frontend processes the video feed and identifies the objects in the feed that are moving. Early iterations of this had to use a static camera feed [17] but it has advanced to the point where it is able to identify the movement in the image due to camera movement and from there determine which objects in the camera are actually moving. There are general parameters for how to format the video feed as an input shown in Table 2.1. The two critical parameters are the frame stride and the resize scale. The frame stride determines how many frames are skipped. If the image has minimal changes between frames then a higher frame stride is advised to help prevent noise in feature motion. The resize scale allows a lower resolution image to be analyzed. This can be useful if the native resolution of the image is large and processing power is limited. If the scale is too small however, then it runs the risk of losing features.

The Frontend processing component has three different tools that can be turned on or off to identify targets: feature motion, color detection, and You Only Look Once (YOLO). For the purposes of our project we used feature motion and the color detector. Feature motion identifies
features that diverge from the movement of the the rest of the camera frame while the color detector is able to put a mask on the image that ignores anything that is not in the desired hue range. The color detector consistently detected the red vehicles amid the background.

Table 2.1: The general processing parameters are used to modify the feed from the camera to maximize the capabilities of all the Frontend modules that are activated. Frame stride determines how many frames are skipped at a time during processing while the resize scale determines the resolution of the image being processed.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Stride</td>
<td>1</td>
</tr>
<tr>
<td>Resize Scale</td>
<td>1.0</td>
</tr>
<tr>
<td>Published Video Scale</td>
<td>1.0</td>
</tr>
<tr>
<td>Published Frame Stride</td>
<td>1</td>
</tr>
<tr>
<td>Text Scale</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 2.2: The Feature Motion module uses the motion of the features to determine which features should be passed into R-RANSAC for track identification. The velocities referred to are pixel velocities across the image not the inertial frame velocity of the targets.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{R \text{ pos}}$</td>
<td>.007</td>
</tr>
<tr>
<td>$\sigma_{R \text{ Vel}}$</td>
<td>.0014</td>
</tr>
<tr>
<td>Minimum Feature Velocity</td>
<td>.004</td>
</tr>
<tr>
<td>Maximum Feature Velocity</td>
<td>.039</td>
</tr>
</tbody>
</table>

The Feature Motion Detector identifies the features in the image and by comparing those features across multiple frames identifies targets in the image. This is done by finding features that move independently from the rest of the image and grouping those features together with others that are in close proximity with similar velocity. If there are not sufficient features in the image or the frame stride is too low then the feature detector will be unable to perform well and will produce false positives and false negatives. The hardware results had a number of constraints due
to the relative speeds of the targets and the UAV and the size of the motion-capture room the tests were conducted in. The small room size constrained the UAV to a low velocity. The UAV needed to travel at greater speeds than the vehicles which further limited their velocities. Frame strides greater than one resulted in missed detections due to the hardware constraints. However, the slow speed of the UAV caused excessive noise when using a frame stride of one. These constraints and the noise they caused required us to use the color detector to identify the vehicles.

The Color detector uses the parameters shown in Table 2.3. It first masks the image with a range in HSV. This mask is further refined by determining a minimum and maximum blob size to get rid of noise on the image and prevent two targets in close proximity from being seen as a single target. Features are then identified in the masked image and combined with all other active Frontend modules.

Table 2.3: The color detector applies a mask out everything in the image but the desired hue range. The blob size further refines the mask by eliminating noise in the subsequent image.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>min hue</td>
<td>0</td>
</tr>
<tr>
<td>max hue</td>
<td>37</td>
</tr>
<tr>
<td>min sat</td>
<td>201</td>
</tr>
<tr>
<td>max sat</td>
<td>255</td>
</tr>
<tr>
<td>min val</td>
<td>84</td>
</tr>
<tr>
<td>max val</td>
<td>219</td>
</tr>
<tr>
<td>morph iterations</td>
<td>1</td>
</tr>
<tr>
<td>morph kernel</td>
<td>2</td>
</tr>
<tr>
<td>min blob size</td>
<td>1</td>
</tr>
<tr>
<td>max blob size</td>
<td>200000</td>
</tr>
</tbody>
</table>

Whichever modules are used in the Frontend, the resulting features form the input for R-RANSAC which determines whether the current detection belongs to a known target track. These detections are labelled with a track ID tying them together with previous detections if they are of the same target or giving them a new ID to be used in the future.
2.2.2 R-RANSAC

Visual-MTT uses R-RANSAC to convert the feature detections from the Frontend into tracks [5]. Tracks are target identifications that are able to persist even if the target temporarily leaves the UAV’s field of view. R-RANSAC does this by estimating the time-evolving signals using recursive least-squares. In order to optimize the track information for our system, we used the parameters in Table 2.4.

R-RANSAC works by first identifying new models using RANSAC. RANSAC first creates hypothesis for new models from random samples. The hypothesis with the most inliers is chosen. RANSAC assumes the target moves at a constant velocity so only two points are necessary to form a hypothesis. R-RANSAC keeps track of the models created at each time-step. It then propagates the models forward and uses the homography convert them into the current frame. The homography is the transformation that maps features in one image to features in another image. It allows R-RANSAC to account for both rotation and translation in the image frame. Existing models that have inliers in the current frame become good models. Tracks are models that are in view in the current video frame.
Table 2.4: Parameters used in Visual MTT R-RANSAC to identify features for target tracking.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Parameters</td>
<td></td>
</tr>
<tr>
<td>( \delta_t )</td>
<td>0</td>
</tr>
<tr>
<td>Model Merging Parameters</td>
<td></td>
</tr>
<tr>
<td>( \tau ) Vel Percent Diff</td>
<td>.25</td>
</tr>
<tr>
<td>( \tau ) Vel Abs Diff</td>
<td>1.28</td>
</tr>
<tr>
<td>( \tau ) Angle Abs Diff</td>
<td>15</td>
</tr>
<tr>
<td>( \tau ) Xpos Abs Diff</td>
<td>.07</td>
</tr>
<tr>
<td>( \tau ) Ypos Abs Diff</td>
<td>.07</td>
</tr>
<tr>
<td>Track Parameters</td>
<td></td>
</tr>
<tr>
<td>( \tau \rho )</td>
<td>.93</td>
</tr>
<tr>
<td>( \tau ) CMD</td>
<td>4</td>
</tr>
<tr>
<td>( \tau ) Vmax</td>
<td>.154</td>
</tr>
<tr>
<td>( \tau ) T</td>
<td>1</td>
</tr>
<tr>
<td>RRANSAC Propagation Specific Parameters</td>
<td></td>
</tr>
<tr>
<td>( N_w )</td>
<td>10</td>
</tr>
<tr>
<td>( M )</td>
<td>30</td>
</tr>
<tr>
<td>( \tau ) R</td>
<td>.4</td>
</tr>
<tr>
<td>Model Pruning Parameters</td>
<td></td>
</tr>
<tr>
<td>( \text{ell} )</td>
<td>100</td>
</tr>
<tr>
<td>guided sampling threshold</td>
<td>1.43</td>
</tr>
<tr>
<td>( \tau ) R RANSAC</td>
<td>.37</td>
</tr>
<tr>
<td>( \Gamma )</td>
<td>.0007</td>
</tr>
<tr>
<td>Model Pruning Parameters</td>
<td></td>
</tr>
<tr>
<td>surveillance region</td>
<td>.99</td>
</tr>
<tr>
<td>( \tau ) CMD prune</td>
<td>10</td>
</tr>
<tr>
<td>Motion Model Specific Parameters</td>
<td></td>
</tr>
<tr>
<td>( \sigma ) Q vel</td>
<td>3.0</td>
</tr>
<tr>
<td>( \alpha ) Q vel</td>
<td>.5</td>
</tr>
<tr>
<td>( \sigma ) Q jrk</td>
<td>.0001</td>
</tr>
<tr>
<td>( \alpha ) Q jrk</td>
<td>.0001</td>
</tr>
</tbody>
</table>
CHAPTER 3. SINGLE AGENT PREDICTIVE PATH PLANNING

When tracking multiple targets, the UAV should not simply find and follow one of them, but should spend time monitoring each target. We propose a path planning algorithm to maximize the information gain over all targets as the UAV flies above the road network. Our approach uses Dijkstra’s algorithm as a path planning framework. Augmenting the algorithm to account for particle movement makes the decision process viable on complex maps.

3.1 Dijkstra’s Algorithm

Dijkstra’s algorithm finds the shortest path between two locations in a graph [18]. We apply Dijkstra’s algorithm by building a spanning tree of the road network, from which the UAV identifies the shortest path to any desired location. This paper modifies Dijkstra’s algorithm to find a path of a desired length that will maximize the information gained during flight. A naïve approach for maximizing information takes the number of particles on an edge and divides by the length of the edge $||G(e)||$

$$V_k^e = \frac{1}{||G(e)||} \sum_{m=1}^{M} \sum_{n=1}^{N} \delta(e^{m,n}, e),$$  \hspace{1cm} (3.1)

where

$$\delta(e^{m,n}, e) = \begin{cases} 
1, & \text{if } e^{m,n} = e \\
0, & \text{otherwise}.
\end{cases}$$  \hspace{1cm} (3.2)

The UAV then feeds the edge values $V_k^e$ into a EHRC to find the path that maximizes target surveillance.

Using this method presents a severe vulnerability because targets with low entropy have closely spaced particles. In the event of a large disparity between the entropy of targets the UAV will prioritize following the target with the lowest entropy to the detriment of all other targets. This situation is illustrated in Figure 3.1 where all the particles for the blue target are located on one
Figure 3.1: Using the naïve value function in Equation (3.1), the UAV exclusively tracks a single target because of the closely packed particles of that target. Ideally the UAV should weight the target with highest entropy more heavily. Particles are plotted as transparent dots to indicate density. The diamonds shows true target positions, the star shows the UAV position, and the dashed circle delineates the UAV’s field-of-view $F$. One target is represented in blue while the other is green.

We avoid the above scenario by implementing target weighting based on the entropy of a target estimate, $u^m$. Entropy-based weighting allows the UAV to prioritize tracking high entropy targets without having the entropy of other targets grow unchecked. The entropy [19] of particle $x^m_k$ is given by

$$u_k^m = \sum_{n=1}^{N} p(x_k^{m,n} \mid y_{1:k}) \log p(x_k^{m,n} \mid y_{1:k}).$$ (3.3)
To get the target weighting, first normalize the entropy via $\eta_{u,k}$ and then apply the sigmoid function

$$
\gamma^m_k = \frac{1}{1 + e^{-a(\eta_{u,k}v^m_k - 0.5)}},
$$

(3.4)

where $a$ is a gain defining how strongly target weights get pushed apart by small differences in entropy. Targets with higher entropy are given higher weights. The resulting weighted edge value is

$$
V^e_{w,k} = \frac{1}{|S(e)|} \sum_{m=1}^{M} \gamma^m_k \sum_{n=1}^{N} \delta(e^{m,n}, e),
$$

(3.5)

resolving the issue shown in Figure 3.1 that used Equation (3.1). In Figure 3.2, using Equation (3.5), the UAV pursues the blue target as it has a greater weight than the green target. Without entropy weighting, the UAV would return to track the green target, allowing knowledge of the blue target to dissipate. With entropy weighting the UAV attempts to maintain equal certainty across all targets.

We next extend this strategy to receding horizon control by propagating the particles forward until the time the UAV will be traversing each edge.

### 3.2 Exhaustive Receding Horizon Control

The UAV needs to account for particle movement while path planning in order to achieve an optimal path. The EHRC creates a tree structure where each branch represents a potential path for the UAV. Branch weights are determined by the weighted edge values $V^e_{w,k}$ along that branch path. Particle propagation must be taken into account since the edge value is a function of time and the UAV does not travel instantaneously down the path. The tree is built recursively where a breadth first search is performed down the tree structure while accounting for particle movement. Each point of exploration of the breadth first search is referred to as a lookahead step.

In each lookahead step, a branch is created for each edge leaving the current node. The edge value, Equation (3.5), is added to that branch, and the particles on that edge are removed from further consideration along that path. All remaining particles are propagated forward by the amount of time it takes the UAV to traverse that edge. This process repeats for each lookahead step until the maximum number of lookaheads are performed. The branch with the highest value
Figure 3.2: Using edge values weighted by target entropy, the UAV prioritizes the blue particles over the green since the entropy of blue is larger than that of green.

is selected, and the UAV traverses the first edge of that path before repeating the procedure to calculate the next edge to traverse.

The computational cost of this algorithm is $O(MNd^L)$, where $M$ is the number of targets, $N$ is the number of particles per target, $d$ is maximum number of edges leaving a node, and $L$ is the number of lookahead steps. This path planning technique was shown in simulation to be effective in choosing good paths.
3.3 Deep-RL

We also implemented a path planner that uses a neural net trained using deep reinforcement learning. Reinforcement learning allows the net to learn what decisions are good and bad based solely off the objective function. An objective function takes the current state of the system and returns a scalar based on how closely that state reflects the system objective. The neural net offers an on-line computational advantage over other path planners since the execution of the neural net is $O(1)$ after training. While this is an ideal computational cost, the trade off comes because the neural net must be trained which, depending on design, map size and number of targets, can take an unacceptably long time. In addition, tuning the training parameters can be a non-trivial process.

Generalization is always a major concern in machine learning as it is impractical to train the neural net on all possible data permutations. Overfitting is one of the main difficulties that impede a neural net’s ability to generalize. Overfitting occurs when the neural net finds a relationship in the training data that allows it to discard data critical to the decision making process. This can occur in scenarios where there is a data-set bias [20] or the neural net trains longer than necessary. We avoid data-set bias by training the neural net on randomly generated scenarios. A failure to generalize can result in overconfidence as the net will perform well under specific conditions but will make poor decisions on scenarios it sees for the first time which can sometimes lead to critical failures.

This issue is minimized in our scenario since the natural progression of the particle filter is towards a uniform distribution of particles across the road network. This means even if the neural net encounters a scenario that results in the agent making the poorest decision possible. The agent will not get stuck in a continual loop of poor decisions since as time passes and the agent fails to properly track all targets the entropy of the particle filter will rise and within a short period of time the neural net will be back in a state it knows how to handle properly. While poor decisions are not ideal, this does pose a significant advantage over using neural nets that could enter into situations that cause critical failure.

This neural net is trained using a proximal policy optimization (PPO). PPO works by taking the loss gradient and uses first order optimizers to maintain a gradient descent thereby optimizing the objective of the net [21]. PPO works by alternating between training a policy network and a value network as shown in Figure 3.3. In other methodologies only one of the two of these
Figure 3.3: Each epoch the reward over random scenarios is recorded. The average reward is kept to track training progress. The dataset of rewards is then trained over the shuffled dataset using the value network as a reference. Once the policy network has been trained the value network is trained on the same dataset using the updated policy network as a reference.

is trained with either a known policy or a fixed evaluation of the system state. The policy loss function is described as,

$$L_P^t(\theta) = \mathbb{E} \left[ \min \left( r_t(\theta)\hat{A}_t, \operatorname{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_t \right) \right],$$

(3.6)

where $r_t$ is the probability ratio at time $t$ and $\hat{A}_t$ is the advantage estimator of the state at time $t$. The probability ratio represents the changes between the new policy and the old policy and the clip function uses the $\varepsilon$ to limit how much the policy can change during each training cycle. The advantage of the state,

$$A_t = -V(s_t) + \lambda V(s_{t+1}) + \ldots + \lambda(T-t)V(s_T),$$

(3.7)

is calculated by looking at the reward of the state by combining the state’s current value $V(s_t)$ with the value of states the current state will lead to up to $T$ timesteps into the future. The weighting between current and future rewards is determined by $\lambda$ where $\lambda = 0$ would give zero weight to
future rewards and $\lambda = 1$ gives equal weight to all rewards evaluated. This is one of many variables that must be tuned to optimize the training cycle. For our scenario we used $\lambda = .8$.

The value loss function is a mean squared error function where the distance is the difference between the estimated value of the state $V_t^{\text{targ}}$ based off the current policy and that given by the value net $V_\theta(s_t)$ which yields

$$L^V F = \left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2.$$  \hspace{1cm} (3.8)

The two loss functions $L^\pi$ and $L^V F$ are then combined in a total loss function which takes the expected value of the difference between the two loss functions

$$L_P^+ + V F(\theta) = \mathbb{E}\left[L_P^\pi(\theta) - c_1 L^V F(\theta)\right],$$  \hspace{1cm} (3.9)

where $c_1$ provides a scaling term to the value loss. This loss function is optimized over hundreds of epochs. By combining the two loss functions together the policy network is able to use the updated value network to find new policies which in turn are used to update the value networks perception of the state. By incrementally training the value net off the policy net and the policy net off the value net the system is able to learn by itself what policies generate good states and what constitutes a good state.

The top layer of our neural net matches the size of the feature space with each subsequent layer using sixty percent of the neurons of the previous layer until the bottom layer as shown in Figure 3.4. The final layer of the policy network is equal to the size of the action space and the final layer of the value network is a single neuron which outputs a scalar value representing the value of the current state. In order to train the neural net, we translate the state of the map, agent positions, and particle data into a numerical vector that is readable by the net. The way data is processed between the input and output of a neural net is called the layout and is determined by the connectivity and number of neurons per layer. We tried a variety of layouts, such as wave form and different monotonically decreasing forms, but found that monotonically decreasing by two thirds each layer performed better or on par with the other methods. This resulted in a total of 1401 nodes. Where each layer of nodes was connected to the subsequent layer by a rectified linear activation unit (ReLU). The ReLU function is linear for values greater than zero and returns zero.
Figure 3.4: The top layer of the neural net is equal to the size of the feature space which in the case of our map is 574. Each subsequent layer is reduced by 40% until the bottom layer which is equal to either the action space in the case of the policy network or a single neuron to estimate the value of the state for the value network.

for all other values. ReLU has superseded Sigmoid and tanh functions as the default activation function [22].

The feature space has five distinct sets of information. First in the state space are the positions of the $Q$ UAVs, $A^q_{ixy}$, in $x,y$ coordinates where $q \in [1, Q]$. Combined with the map layout, the neural net is able to learn which actions in the action space are valid for each UAV’s location.

The particle distribution is the next part of the state space, given by

$$
\mathcal{X}^m_{1:B} = [\mathcal{X}^m_1, \ldots, \mathcal{X}^m_B].
$$

(3.10)

The map is separated into $B$ equal sized bins of 10 meters. The number of particles from target $m$ in $b = 1$ is $\mathcal{X}^m_1$. In this way the path planner knows which actions would be best to minimize the entropy for each individual target.
Figure 3.5: This figure depicts the neural net’s reward function over the training time. Within the first 80 epochs the neural net has reached an average reward of 80 units on random simulations. At this point it is outperforming a random path planner but the results are not competitive. Since the learning has hit a plateau, it takes 500 more epochs to bridge the gap to the point where it is performing at the level shown in Figure 5.3.

To know which target’s entropy we need to minimize we then append a list of each target’s entropy weighting $\gamma_k^m$, i.e.

$$
\gamma_k^{1:M} = [\gamma_k^1, \ldots, \gamma_k^M],
$$  

(3.11) 

allowing the neural net to focus on targets with greater entropy.

Finally, the map layout is included as a list of nodes

$$
\mathcal{N}_{x,y}^{1:C} = [\mathcal{N}_{x}^{1}, \mathcal{N}_{y}^{1}, \ldots, \mathcal{N}_{x}^{C}, \mathcal{N}_{y}^{C}].
$$  

(3.12) 

This provides the map’s $x,y$ coordinates when combined with their connecting edges,

$$
\mathcal{E}^{1:f} = [\mathcal{E}^1, \ldots, \mathcal{E}^f].
$$  

(3.13) 

24
Combining these five sets of information, the feature space is defined as the one dimensional vector,

\[ F_{\text{net}} = [A_{x,y}^1, \ldots , A_{x,y}^Q, \mathcal{X}_{1:B}^1, \ldots , \mathcal{X}_{1:B}^M, \gamma_{1:M}, \gamma_{1:C}, \mathcal{E}^1:f]. \] (3.14)

Each road leaving an intersection is considered a different possible action by the UAV. The action space is therefore equal to the maximum number of edges leaving any node. If we simply used a sorted list of the nodes connected to the current node with the action as an index into that list then the neural net would prioritize lower index values over higher ones since some nodes have shorter lists than others. By using an unordered list of the connected nodes for each node on the map as the action space no index receives an undesired bias.

We trained the net to reasonable levels within a short span using the reward function

\[
R_e = \begin{cases} 
\max (\gamma) - \gamma_k & \text{if } \max (\gamma) - \gamma_k < 31 \\
1.12^{\max (\gamma) - \gamma_k} & \text{otherwise}
\end{cases}.
\] (3.15)

The training cycles after the initial rise only yielded incrementally better results, as seen in Figure 3.5. There are many different variables that go into the speed and accuracy with which a neural net can be trained. Future work would involve finding a way to prevent the neural net from plateauing after 80 epochs and instead continue with its steep learning curve until it maximized its reward.
CHAPTER 4. LOWER BOUND FOR AVERAGE ENTROPY OF RBPF

In this chapter we develop a theorem to find the lower bound $B_L$ of the average entropy of the Rao-Blackwellized Particle Filter (RBPF). The theorem relies on Assumptions 1-7.

**Assumption 1.** *Targets and the UAVs maintain constant velocity with random perturbations.*

The motion model used for this thesis and theorem assumes that both the targets and the UAVs travel at a constant velocity with random perturbations.

**Assumption 2.** *The round-trip path is the ideal layout for the multi-agent path planner.*

**Assumption 3.** *Each target uses an independent stochastic path planner.*

**Assumption 4.** *Targets are equally likely to be at any point on the road network.*

In the scenarios considered, the targets have a uniform likelihood of starting at any location in the map. They maintain a constant velocity and use independent stochastic path planners which results in a uniform likelihood of the targets being at any location on the road network.

**Assumption 5.** *The average distance between any two graph nodes $S$ is equal to the average distance between any two points on the graph multiplied by a scale factor $\alpha_S$.***

The distance between any two points on the graph can be found in one of two ways. If the points are on the same edge then the distance is the Euclidean distance between the points. Otherwise the distance is the minimum distance between the nodes adjacent to the two points plus the distance from the points to the adjacent nodes. Because targets are equally likely to be at any location on the map, averaging these two scenarios results in a distance that is closely related to the average distance between nodes. However as the exact relationship between the average distance between points on a graph and the average distance between nodes is dependent on individual map configuration it must be determined on a map by map basis.
We verified Assumption 5 using Monte Carlo simulations. With 1000 Monte Carlo runs per map over 144 maps that varied from 100\text{m}^2 to 3600\text{m}^2 in size and found that the error between the Dijkstra’s average and the distance between two points was less than two percent of the given map size showing that for a wide variety of maps the scale factor $\alpha$ between $S_{\text{node}}$ and $S_{\text{point}}$ is relatively close to one.

**Assumption 6.** Particle filter resampling occurs at each timestep. The particle filter receives one positive update per sighting and no negative updates.

Negative updates affect the entropy of the filter any time the UAV is in the proximity of particles, but the target is not yet in sight. While we can determine what the average round-trip time is, the round-trip path varies making it difficult to account for how negative updates will affect the particle filter on average.

**Assumption 7.** Mode merging does not occur.

A mode is comprised of particles that are in close proximity and have similar velocities. A detailed description of the evolution of modes is provided in Section 4.3.

We now provide a theorem that defines the lower bound on the average entropy. This theorem uses the degree type $D$ of a mode in the entropy calculation. The degree type is the $D$ percent of the particle filter contained in that mode. For example, when a mode of degree $D = 100\%$ exists, every particle in the filter is in that mode.

**Theorem 1.** The lower bound for average entropy of a RBPF governed by Assumptions 1-7 is

$$B_L = \frac{1}{T_Q} \sum_{k=0}^{T_0} \mathbb{E}[u_k], \quad (4.1)$$

where

$$T_Q = \frac{T}{Q \delta_t} \quad (4.2)$$

is the average minimum number of timesteps between target sightings (average minimum round-trip time $T$ divided by the number of UAVs $Q$ and the timestep duration $\delta_t$)

$$\mathbb{E}[u_k] = - \sum_{D \in D} \sum_{j=0}^{J} P_{d,k} \varphi_k[b^j] \log \left( \frac{d \varphi_k[b^j]}{\varphi_k[b^j]} \right) \quad (4.3)$$
Figure 4.1: In the case of five equally-spaced targets the shortest round-trip distance is a regular polygon. These four polygons represent all possible polygon sizes. Taking the side length $S_{\text{avg}}$ gives us the average distance between targets on the minimum round-trip path shown in Equation (4.5)

is the expected entropy, where $\varphi_k[b^h]$ is the probability of a target existing in a discrete section $b^h$ of the map, $D$ is the degree type, and $P_D$ is the probability of a mode of degree type $D$ existing.

The proof depends on five lemmas which are presented in the subsequent sections. Lemma 1 identifies the average minimum round-trip time $T$ to visit all targets on the map. The variance growth of the RBPF is defined in Lemma 2 and the average variance after a positive update is defined in Lemma 3. The relationship between variance and entropy in a single mode is in Lemma 4. And the probability $P_D$ of a mode of degree type $D$ existing is given in Lemma 5.

4.1 Minimum Time to All Targets

In this section we provide a formula for calculating the average minimum-time to visit all the targets on the road network.

**Lemma 1.** Under Assumptions 1, 4, and 5 the average minimum round-trip time to visit all targets is

$$T = \frac{S_{\text{avg}} M}{V_a},$$  \hspace{1cm} (4.4)

where $V_a$ is the airspeed of the UAV, and $S_{\text{avg}}$, given by

$$S_{\text{avg}} = \frac{1}{G} \sum_{g=1}^{G} S_g,$$  \hspace{1cm} (4.5)
is the average distance between targets on the round-trip path with \( G = M - 1 \), and

\[
S_g = 2\alpha S \sqrt{\left(1 - \cos \frac{2\pi}{M}\right) \left(1 - \cos \frac{2g\pi}{M}\right)},
\]

(4.6)

where \( \alpha \) is the scale factor between the average distance between points and the average distance between nodes on a graph from Assumption 5.

Proof. Dijkstra's algorithm gives the minimum distance from a node to any other node on the graph [23]. By taking the average of the vector of distances returned by Dijkstra's algorithm we get the average distance from a given node to any other node. Repeating this process for each node and taking the average of the results gives us the average distance from any node to any other node on the graph, designated as \( S \).

By Assumption 4 and 5 each target is on average \( \alpha S \) meters apart when travelling along the road network. The minimal path that visits \( M \) targets that are \( S \) meters apart can be approximated by a regular polygon with a target located at each vertex. The distance between vertices is equally likely to be \( S \). Since the diagonals and the edges of a regular polygon are not equidistant this results in a list of equally likely polygons of varying sizes.

The perimeters of this list of polygons are averaged together to estimate the average round-trip distance. A regular polygon is symmetric and the average round-trip distance can be determined from the diagonals and edges of a single vertex. There are \( G = M - 1 \) connections from one vertex to all other vertices on the polygon.

Let the radius \( r \) be the distance between any vertex and the center of the polygon. When the side length \( S_g \) and the number of vertices \( M \) are known the law of cosines \( c^2 = a^2 + b^2 - 2ab \cos(\theta_c) \) can be used to calculate the radius and the length of the line between any two vertices. Using a single side of the polygon and two radii to form an isosceles triangle the side length is then

\[
S_g^2 = r^2 + r^2 - 2r^2 \cos\left(\frac{2\pi}{M}\right).
\]

(4.7)

Solving for the radius \( r \) yields

\[
r = \frac{S_g}{\sqrt{2 \left(1 - \cos \frac{2\pi}{M}\right)}}.
\]

(4.8)
The line connecting any two vertices is the third side of an isosceles triangle where \( r \) forms the other two sides with an angle between them as \( 2g\pi/M \). Using this information, the line is calculated by
\[
d_g^2 = r^2 + r^2 - 2rr \cos \frac{2g\pi}{M},
\]
which yields
\[
d_g = r \sqrt{2 \left( 1 - \cos \frac{2g\pi}{M} \right)}. \tag{4.10}
\]
Substituting Equation (4.8) into Equation (4.10) gives
\[
d_g = \frac{S_g}{\sqrt{2 \left( 1 - \cos \frac{2\pi}{M} \right) \sqrt{2 \left( 1 - \cos \frac{2g\pi}{M} \right)}}}. \tag{4.11}
\]
For each possible polygon, we set \( d_g \) equal to \( \alpha S \) in Equation (4.11) and then solve for the side length, \( S_g \) to obtain Equation (4.6). This identifies the side lengths of the \( G \) potential polygons. The average side length is given by Equation (4.5).

The average time duration of a round trip across all targets is then given by the average distance divided by the UAV’s airspeed as shown in Equation (4.4).

Figure 4.1 illustrates the set of possible polygons when there are five targets. Since any two targets are an average distance of \( S \) apart, the lines between two vertices \( (d_1, d_2, d_3, \text{ and } d_4) \) remain a set distance, but the side lengths \( S_g \) for \( g \in [1, 4] \) change.

4.2 Single Mode Entropy

In this section we provide a formula for calculating the particle filter’s entropy when there is only one mode.

The entropy of a mode at time \( k \) can be calculated from its variance. This is because entropy in the particle filter is based off the particle filter density, which is directly related to the variance when there is only one mode. Because our measurement model uses Gaussian noise in the velocity,
the modes in our particle filter are also Gaussian. Entropy is determined from the particle’s density, represented by the positional variance \( \sigma_{p,k}^2 \). The mode’s position in the graph is irrelevant.

Until a mode encounters an intersection causing it to split into two or more separate modes its behaviour and variance are the same as would occur in a Kalman Filter. To predict \( \sigma_{p,k}^2 \) at timestep \( k \) we need to know how the mode’s variance grows over time and what the variance will be after a positive measurement update. For the sake of this theorem the variance right after incorporating a positive measurement update is referred to as the minimum variance.

We give the calculation for the single mode entropy by first describing the variance growth of the RBPF between updates (Lemma 2), then computing the average variance after a positive update (Lemma 3), and finally providing the relationship between variance and entropy in a single mode (Lemma 4).

**Lemma 2.** The growth of the positional variance between sightings for a mode \( \sigma_{p,k}^2 \), under Assumptions 1 and 6, at timestep \( k \) is

\[
\sigma_{p,k}^2 = \sigma_{p,k_0}^2 + \sigma_v^2 \delta_t^2 \Delta_k,
\]

where \( k_0 \) is the time of the last sighting, \( \delta_t \) is the timestep duration, and \( \Delta_k = (k - k_0) \).

**Proof.** The behaviour of a single mode between updates matches that of a Kalman filter. Therefore, we use the Kalman filter’s time propagation equation to predict a mode’s variance growth. At timestep \( k_0 \) the Kalman filter’s co-variance matrix is

\[
P_{k_0} = \begin{bmatrix}
\sigma_{p,k_0}^2 & 0 \\
0 & \sigma_v^2
\end{bmatrix}
\]

(4.13)

and the state transition matrix for a constant-velocity target (Assumption 1) is

\[
F = \begin{bmatrix}
1 & \delta_t \\
0 & 1
\end{bmatrix}.
\]

(4.14)

The time-propagated covariance is

\[
P_{k+1} = FP_kF^\top.
\]

(4.15)
If our particles were initialized with constant velocities and resampling only took place at $k_0$ then

$$P_{k_0+1} = \begin{bmatrix} \sigma_{p,k_0}^2 + \delta_t^2 \sigma_v^2 & \delta_t \sigma_v^2 \\ \delta_t \sigma_v^2 & \sigma_v^2 \end{bmatrix}.$$  

(4.16)

Iteratively applying Equation (4.15) from timestep $k_0$ to $k$ yields

$$P_k = \begin{bmatrix} \sigma_{p,k_0}^2 + (\sigma_v^2 \delta_t^2) \sum_{m=0}^{\Delta_k} 2m + 1 & \delta_t \sigma_v^2 \sum_{m=0}^{\Delta_k} 2m + 1 \\ \delta_t \sigma_v^2 \sum_{m=0}^{\Delta_k} 2m + 1 & \sigma_v^2 \end{bmatrix}.$$  

(4.17)

However, under Assumption 6, resampling occurs at every timestep during which the velocity for each particle is reset. Because the velocity at time $k$ is independent of its velocity at time $k-1$ the off-diagonal terms in the covariance matrix (Equation (4.16)) become zero at each timestep. This simplifies Equation (4.17) to

$$P_k = \begin{bmatrix} \sigma_{p,k_0}^2 + \sigma_v^2 \delta_t^2 \Delta_k & 0 \\ 0 & \sigma_v^2 \end{bmatrix},$$  

(4.18)

where the first element in the matrix is the positional variance $P_k[0,0] = \sigma_{p,k}^2$ and is given by Equation (4.12).

To verify that Equation (4.12) provides an accurate estimate of the positional variance growth, we ran 100 Monte Carlo simulations for various values of $\sigma_v^2$. We compared the simulated positional variance with the variance calculated using Equation (4.12). The simulated variance is calculated by starting the simulator with a known variance $\sigma_{p,k_0}^2$ and then allowing the simulation to run for 55s without any updates.

When comparing the actual variance with our estimate, we found that the error in our estimate was expressed as a Gaussian centered around the true positional variance. The Gaussian’s variance increased proportionally with increasing values of $\sigma_v^2$. This is shown in Figure 4.2 where
Figure 4.2: This graph shows the error between the positional variance estimate given in Equation (4.12) and the positional variance computed in simulation when running without a positive sighting for $t = 55s$. For each value of $\sigma_v^2$ the error in $\sigma_p^2$ is calculated over 100 Monte Carlo runs.

the error Gaussian for increasing values of $\sigma_v^2$ are compared. As expected increasing $\sigma_v^2$ resulted in increasing error variance in the $\sigma_p^2$ estimate.

When plotting the predicted variance against the average of our Monte Carlo runs for a single $\sigma_v^2$ we get the results shown in Figure 4.3. As can be seen, the estimate exactly matches the simulation average, validating Equation (4.12).

In the next lemma we incorporate positive updates to calculate the average minimum variance for a mode. We set the timestep duration to specifically be $\Delta_k = (T / \delta)$, which is the number of timesteps between positive updates, where the round-trip time $T$ is given by Lemma 1.

Because the particles in our system are constrained to the road network they are effectively constrained to 1D space. The UAV measures the target’s position along the road segment, making the measurement covariance $R$ a scalar.
Figure 4.3: The predicted variance growth calculated using Equation (4.12) and the average variance from 100 Monte Carlo simulations are plotted for a \( t = [0, 55] \) s time window. As expected from the results in Figure 4.2 the two plots line up perfectly.

**Lemma 3.** The average minimum variance \( \sigma_{p_0}^2 \) for a mode is

\[
\sigma_{p_0}^2 = -\Delta_k \delta_\gamma^2 \sigma_v^2 + \sqrt{(\Delta_k \delta_\gamma^2 \sigma_v^2)^2 + 4R\Delta_k \delta_\gamma^2 \sigma_v^2}. 
\]

(4.19)

**Proof.** The minimum variance occurs after a positive update and is a function of both the variance of the mode prior to the update and the measurement covariance. The relationship between the variance before and after an update can be expressed in the information update equation from the information filter [24]

\[
Y_k = P_k^{-1} + H^\top R^{-1} H, 
\]

(4.20)
where $H = [1 \ 0]$ is the observation model. Substituting $H$ and Equation (4.18) into Equation (4.20) yields

$$
Y_k = \left[ \begin{array}{cc}
\left( \sigma_{p_0}^2 + \sigma_v^2 \Delta_k^2 \right)^{-1} & 0 \\
0 & \sigma_v^{-2}
\end{array} \right] + \left[ \begin{array}{cc}
R^{-1} & 0 \\
0 & 0
\end{array} \right].
$$

(4.21)

We note that $Y_k[0,0] = \sigma_{p_0}^{-2}$ and use Equation (4.21) to solve for the minimum variance,

$$
\frac{1}{\sigma_{p_0}^2} = \frac{1}{\sigma_{p_0}^2 + \Delta_k \delta^2 \sigma_v^2} + \frac{1}{R}
$$

$$
\sigma_{p_0}^2 = \frac{R(\sigma_{p_0}^2 + \Delta_k \delta^2 \sigma_v^2)}{R + \sigma_{p_0}^2 + \Delta_k \delta^2 \sigma_v^2}
$$

$$
R(\sigma_{p_0}^2 + \Delta_k \delta^2 \sigma_v^2) = (R + \sigma_{p_0}^2 + \Delta_k \delta^2 \sigma_v^2)\sigma_{p_0}^2
$$

$$
0 = \sigma_{p_0}^4 + \sigma_{p_0}^2 \Delta_k \delta^2 \sigma_v^2 - R\Delta_k \delta^2 \sigma_v^2.
$$

The minimum variance is obtained from the positive half of the quadratic formula, since the variance is positive definite, and yields Equation (4.19).

The relationship between the minimum variance, the measurement covariance, and the size of the map is shown in Figure 4.4. This graph plots the map’s edge distance compared against the average minimum variance for different measurement covariance $R$ values. If the UAV has low trust in its sensors, represented by a high $R$ value, then the particle cloud after a positive update will be less concentrated than if the UAV has a small measurement covariance value, representing a high degree of trust in the sensor measurements. The size of the map determines the duration between sightings as given in Lemma 1. Larger map edges (shown along the X-axis) allow for more time between positive updates, resulting in a greater $\sigma_p^2$ prior to and after the positive update. These two factors combined with the mode’s variance growth rate allow for a perfect estimate of the minimum variance of the mode.

**Lemma 4.** When a particle filter is confined to a single mode, the entropy of the particle filter is

$$
\bar{u}_k = \sum_{j=0}^{J} \varphi_k[b^j] \log \varphi_k[b^j],
$$

(4.22)
Figure 4.4: This graph plots the relationship between the size of the map and the average minimum variance for different measurement covariance values. Increasing $R$ shows less trust in the sensors and results in less effective positive updates. Increasing the map size (X-axis) allows for more time between positive updates and also results in a larger minimum variance $\sigma_p^2$.

where the map is discretized into $J$ bins of one meter length

$$b^{0:J} = [0, 1, 2, \ldots, L_{map}], \quad (4.23)$$

$L_{map}$ is the length of the map, and $\varphi_k[b^i]$ is the probability of a target existing in section $b^i$ of the map.

Proof. Since the particle cloud in a single mode system is contiguous we can abstract away the map by saying that the particle cloud lies along a single line where the length of the line is equal to the distance connecting the two furthest points on the map $L_{map}$. The map is discretized into $J$ bins of a uniform length (for our simulation results this was set to one meter). The single mode has
Figure 4.5: Comparison of simulated entropy generated by 100 Monte Carlo runs with the estimate of the entropy from Equation (4.22). This graph shows that for a single mode the estimate matches the simulation’s average.

A Gaussian distribution. Therefore the probability that a target is within bin \( h \) is

\[ \phi_k \left[ b^j \right] = \frac{1}{\sqrt{2\pi\sigma^2_{p,k}}} \exp \left( -\frac{(b^j - \mu)^2}{2\sigma^2_{p,k}} \right). \] \hspace{1cm} (4.24)

Replacing \( p(x^m_{k,n_1}|y^m_{1,k}) \) with \( \phi_k \left[ b^j \right] \) in Equation 3.3 yields Equation (4.22).

In Figure 4.5 we show 100 Monte Carlo simulations and compare the average entropy with the entropy calculated in Equation (4.22). The plots line up closely showing that for a single mode, the algorithm provides an excellent estimate in the short term, with a slight divergence as time progresses.
To make this algorithm viable for more than the trivial case we need to account for multimodal distributions, which is addressed in the next section.

4.3 Estimated Number of Modes

To estimate a mode’s probability of existence $P_D$ for each degree type $D$ over time, the map is analyzed to understand how long a single mode will traverse each map’s edge and how many modes it will split into once it leaves the edge. Each edge may be considered separately to understand the modes behavior. However, we will exploit the similarities found in many maps by grouping edges (or contiguous edges that don’t split into multiple road segments) if they have the same length and relate to the rest of the map in an identical fashion. These groups are termed subdivisions. In the case of the 3x3 map, shown in Figure 3.2 the map is composed of sixteen directed edges. The edges can be grouped into three subdivisions as shown in Figure 4.6a by the blue, red, and purple colored arrows.

A map’s subdivisions are then converted into the tree structure shown in Figure 4.6b. The first set of branches show the probability that a target will be initialized in each subdivision while the second set of branches show which subdivisions are accessible from the current subdivision, how many modes will be spawned upon leaving the subdivision, and the percent of the current mode that will exist in each subsequent mode. In the case of a stochastic path planner, the parent mode will divide into equal portions when splitting into child modes.

When a mode is initialized it has a degree type $D = 100\%$. The way it splits into child modes is dependent on the subdivision it was initialized in. Each subsequent mode’s $P_D$ is dependent on the $P_D$ of its immediate parent and the time it takes for the parent to traverse its subdivision.
Though complex, mode probabilities deterministically depend on the properties of the subdivision in which it was initialized.

**Lemma 5.** Under Assumptions 1, 3, and 7, the probability \( P_D \) of each mode of degree type \( D \) existing at timestep \( k \) is given by Algorithm 2.

**Proof.** When a mode is initialized in a given subdivision at timestep \( k_0 \) it has a uniform likelihood of being anywhere in that subdivision. As a result the earliest that mode can leave the subdivision is at \( k_0 + 1 \) (assuming it was initialized at the end of the subdivision) and the latest is at \( k_0 + T_r \), where \( T_r \) is the time it takes a target to traverse the subdivision (assuming it was initialized at the beginning of the subdivision). After initialization, the progress of a mode is deterministic. Therefore the linear probability of a mode encountering its next intersection at timestep \( k \) is

\[
w(k, T_s, T_r) = \begin{cases} 
0 & k < T_s \\
\frac{k-T_s}{T_r} & T_s \leq k < T_s + T_r \\
1 & \text{otherwise}
\end{cases} \tag{4.25}
\]

where \( T_s \) is the timestep that the mode came into existence and \( T_r \) is the time it takes to traverse the initial subdivision of the mode’s first parent.

Given Equation (4.25) and Assumption 3 we can extrapolate the probability of existence for the children of a specific mode

\[
W \left( k, T_r, T_s, [T_{d_1}, \ldots, T_{d_F}] \right) = w(k, T_s, T_r) - \sum_{f=0}^{F-1} \frac{1}{F} w(k, T_s + T_{d_f}, T_r), \tag{4.26}
\]

where \( T_s \) is the time the child modes come into existence and \([T_{d_1}, \ldots, T_{d_F}]\) are the times it takes each of the \( F \) child modes to traverse their respective subdivisions. The child modes’ probabilities are divided by \( F \) since, with Assumption 3, particles will traverse each edge with equal probability. Using these parameters, Equation (4.26) first predicts the probability of the child modes coming into existence with \( w(k, T_s, T_r) \). Then it iterates over the \( F \) child modes to sum their probabilities of ceasing to exist. Combining these two halves of Equation (4.26) gives the overall probability of existence at any time \( k \) for the child modes.
Algorithm 1 GetModeSplitLevel(e, P\text{init}, T_r, D, T_s, \text{Map}, k)

Require: the current subdivision sd, Probability of first parent P\text{init} \in \mathbb{R}, the time it takes to traverse the first parent’s subdivision T_r, the current mode degree D, the time the current mode came into existence T_s, the Map in tree format (shown by Figure 4.6b), and the current timestep k.

Ensure: The mode probability for each of the children of the current mode P_D and the list of child modes organized by degree D added to the unprocessed modes list U.

1: \( U \leftarrow \{\} \)
2: \begin{algorithmic}
   \FOR {option \in \text{options}}
   \STATE \( D_{\text{next}} \leftarrow D / \text{len(Map[option].neighbors)} \)
   \STATE \( T_o \leftarrow \text{Map[sd].offsets[option]} \)
   \STATE \( T_d \leftarrow \text{Map[option].offsets} \)
   \STATE \( P_D[D_{\text{next}}] \leftarrow P\text{init}DW(k, T_r, T_s + T_o, T_d) \)
   \STATE \text{unprocessed}[D_{\text{next}}] \text{ append } [P\text{init}, T_r, T_s + T_o, \text{option}] \)
   \ENDFOR
3: \textbf{return} \( P_D, U \)

Algorithm 2 GetModeProbabilities(D_c, U_c, k, \text{Map}, T_Q)

Require: the list of initial mode degrees D_c, the list of initial modes U_c, the current timestep k, the map in tree format shown in Figure 4.6b, the maximum number of timesteps T_Q.

Ensure: the probability for each mode degree type P_D.

1: \begin{algorithmic}
\FOR {D \in D_c}
   \FOR {line \in U_c}
   \STATE \( sd \leftarrow \text{line.sd} \)
   \STATE \( P\text{init} \leftarrow \text{line.prob} \)
   \STATE \( T_r \leftarrow \text{line.range} \)
   \STATE \( T_s \leftarrow \text{line.start} \)
   \IF {T_s < T_Q}
   \STATE \( \text{ModeProb}, U_c \leftarrow \text{GetModeSplitLevel}(sd, P\text{init}, T_r, D, T_s, \text{Map}, k) \)
   \ENDIF
\ENDFOR
\ENDFOR
\IF {U_c \neq \emptyset}
\STATE \( D_n \leftarrow U_c.\text{keys} \)
\STATE \text{ModeProb} \leftarrow \text{GetModeProbabilities}(D_n, U_c, k, \text{Map}, T_Q) \)
\ENDIF
\textbf{return} \text{ModeProb}
\end{algorithmic}
Figure 4.7: Probability of mode existence $P_D$ for map 4.6a over time $t = [0, 20]$ s. The probability of a mode that contains $D$ percent of the particle filter is represented by the different lines plotted. As time progresses the probabilities of larger modes decrease as they evolve into smaller modes with increasing likelihood.

Algorithm 1 takes a single mode and calculates the probability of existence for each of its children $P_{\text{child}}$ using Equation (4.26). The $P_D$ for the mode degree type $D$ of the children is incremented in line 6 by combining $P_{\text{child}}$ with the probability of the first parent $P_{\text{init}}$ and normalizing by the mode’s degree. The algorithm then returns the $P_D$ for the modes processed as well as a list of the child modes appended to the unprocessed modes list.

Algorithm 2 recursively calculates the probability $P_D$ that each mode degree type $D$ exists across a given duration. It calculates all unprocessed modes and inputs the ones that are born before timestep $T_Q$ into Algorithm 1. The unprocessed modes returned from Algorithm 1 are appended to the unprocessed mode list. This recursion repeats until there are no remaining unprocessed modes within the desired time window. Algorithm 2 ensures each possible mode is passed into Algorithm
Algorithm 1 calculates the mode’s $P_D$. Therefore, Algorithm 2 provides the probability of $P_D$ for each mode degree type over a set duration $T_Q$.

We verified Lemma 5 by comparing it with results from 10000 Monte Carlo runs on the map in Figure 4.6a. For each Monte Carlo run a mode of degree type $D = 100\%$ was initialized on a random location on the map and then allowed to traverse the map, propagating and splitting for a set time period $T_Q = 20/\delta_t$. The expected number of modes of degree type $D$ obtained by averaging the mode count from the Monte Carlo runs was converted into the probability

$$P_D = \frac{\mathbb{E}[D]}{D}. \quad (4.27)$$

The resulting $P_D$ matched the results of Algorithm 2. Plotting $P_D$ for all possible degree types yields Figure 4.7.

On large maps with many intersections there is a possibility that between target sightings the modes in the particle filter will merge as well as split over time. Mode merging is a complex subject as it can result in non-Gaussian structures depending on how the modes line up with each other as they merge. Mode merging is not analyzed in this paper but will be considered in future work.

### 4.4 Theorem for Lower Bound of Average Entropy

We are now prepared to prove Theorem 1.

**Proof.** Lemma 1 gives the round-trip time to visit each target. Under Assumption 2 the ideal multi-agent path planner will evenly space the UAVs around the roundtrip path. The time between target sightings is obtained by taking the round-trip time given by Equation (4.4) and dividing by the number of active UAVs resulting in Equation (4.2).

The entropy of a single mode system is provided by Equation (4.22), from Lemma 4, where $\phi_k(b^h)$ assumes $D = 100\%$ to properly account for the all the particles belonging to that one mode. The mode degree then determines the percentage of $\phi_k(B^h)$ that is used to calculate the entropy.
converting Equation (4.22) into

\[
u_{d,k} = \sum_{h=0}^{H} -d \phi_k[b^h] \log(d \phi_k[b^h])
\]  

(4.28)

for a single mode of degree \(d\).

The expected number of modes of degree type \(D\) is

\[
E[D] = P_D \frac{1}{D},
\]  

(4.29)

which is the probability that that modes of type \(D\) exists divided by the mode degree. For example, if there is a 100\% probability that modes of type \(D = 50\%\) exist, then there will be two modes that each contain half of the particles and \(E[50\%] = 2\).

The Expected entropy of a mode degree type \(D\) is the expected number of modes of that degree, multiplied with the entropy of modes of that degree. Since variance growth is identical across all modes every mode of a given degree will have identical entropy. To calculate the expected entropy of the system we sum over the expected entropy for each mode degree type giving

\[
E[u_k] = \sum_{d \in D} P_{d,k} \frac{1}{d} \sum_{h=0}^{H} -d \phi_k[b^h] \log(d \phi_k[b^h]),
\]  

(4.30)

which simplifies into Equation (4.3). Lemma 5 provides the probability \(P_D\) of each mode type existing. The lower bound for the average entropy is then the average of Equation (4.3) for \(k = [0, T_Q]\), given by Equation (4.1).

This theorem represents the best case scenario using a perfect multi-agent path planner. In cases with sub-optimal path planners it is possible that more UAVs would be required to maintain the desired lower bound of average entropy.

While entropy is a useful metric it is not intuitive. To make Theorem 1 more intuitive we provide Corollary 1 as a means of transforming the lower bound of average entropy into the upper bound for the average variance in location certainty per target estimate. The variance in location certainty is the positional variance \(\sigma_p^2\) in a single mode. For example if there are five modes each
with a positional variance of 3 meters then we know that the target is in one of five locations with a 3 meter variance in each of the five estimates. This corollary uses an additional assumption.

**Assumption 8.** The number of modes in the particle filter when at the lower bound of average entropy is known as $D_B$.

The positional variance $\sigma_p^2$, number, and degree of modes are used when calculating entropy. Because of this, if the exact number and type of modes is unknown there is a many-to-many relationship between the entropy and the positional variance in the modes of the system. However, if the number of modes are known then there is a one-to-one relationship between entropy and the positional variance in each mode of the RBPF under Assumptions 1, 6, 7, and 8.

**Corollary 1.** If the number of modes of each degree is known (Assumption 8) then the upper bound on the average variance in target location certainty per estimate can be calculated by solving for $\sigma_p^2$ in

$$B_L = -\sum_{d \in D} \sum_{j=0}^{J} \frac{1}{\sqrt{2\pi \sigma_p^2}} \exp \left( -\frac{(b_j - \mu)}{2\sigma_p^2} \right) \log \left( D \frac{1}{\sqrt{2\pi \sigma_p^2}} \exp \left( -\frac{(b_j - \mu)}{2\sigma_p^2} \right) \right).$$

(4.31)

**Proof.** If the number of modes of each degree type is known then $P_D = 1$ for each degree type in Equation (4.3). Substituting in Equation (4.24) for $\phi$ yields

$$u = \sum_{d \in D} \sum_{j=0}^{J} \frac{1}{\sqrt{2\pi \sigma_p^2}} \exp \left( -\frac{(b_j - \mu)}{2\sigma_p^2} \right) \log \left( d \frac{1}{\sqrt{2\pi \sigma_p^2}} \exp \left( -\frac{(b_j - \mu)}{2\sigma_p^2} \right) \right).$$

(4.32)

The upper bound for the location certainty per target estimate is the positional variance of the RBPF when the entropy is equal to $B_L$. To get the variance corresponding to $B_L$ we set $u = B_L$ and numerically solve Equation (4.32) for $\sigma_p^2$.

As an example, consider the map shown in Figure 3.1 with ten targets. It is possible to determine the lower bound for entropy based off the number of UAVs deployed. With just one UAV the lower bound for entropy calculated in Equation (4.1) is $B_L = 2.89$. If there is only one
mode present of degree type $D = 1$ then we can find the upper bound on the target location certainty by solving Equation (4.31) where $D = 1$. This shows that the upper bound on the target location certainty is 19 meters. If this mission requires an upper bound of 16 meters, we know that one UAV is insufficient. Setting $\sigma_p^2 = 16$ in Equation (4.31) and solving for $B_L$ we get the required lower bound on entropy to meet mission requirements. To get the number of UAVs required for the mission we solve for $Q$ in Equation (4.1). By rounding up to the nearest integer we find that a minimum of three UAVs are required to maintain the required average location certainty in target estimates.
CHAPTER 5. RESULTS

We verified the results of this thesis both in simulation and on hardware. The simulation results compared the different path planners with an ideal path planner and showed that the lower bound obtained from Theorem 1 held for each planner. The hardware results showed that an end-to-end solution for multi-target tracking performed well using the ERHC planner.

5.1 Simulation Results

In this section, simulation results are provided using the 3x3 road network in Figure 5.1 with two targets, each travelling at a nominal ten meters per second. The UAV flies at 40 meters per second. In the top level of the filter, ten particles estimate data association histories. Each top level particle has one tracking PF per target (totalling 20 tracking filters) and each tracking PF has 500 particles. The target weighting sigmoid in Equation (3.4) uses gain $a = 10$ and the measurement sensor noise is $R = 5$.

Figure 5.2 shows how the combined entropy of the filter evolves as the UAV tries to find and follow both targets. In region A, the UAV has not found either target. The plot shows some decline in entropy as negative updates are applied and areas are ruled out. Region B shows the time after the first target has been found and priority switches to finding the second target. In Region C, the UAV tries to balance time between following each target to minimize total entropy. Rapid increases in entropy result when targets reach an intersection and hypotheses split. Steep declines in entropy result from positive measurements of the target and negative measurements ruling out hypotheses.

The simulation was run 1000 times using four different controllers and the entropy was averaged over all the runs. The controllers compared were the EHRC from Section 3.2, deep-RL planner from Section 3.3, an ideal planner with perfect knowledge of the target locations, and a stochastic planner that randomly decides paths whenever it encounters intersections.
Figure 5.1: Simulation tracking two targets on a 3x3 city block map. Here the UAV is drawn toward the blue target because its estimate has higher entropy.

Figure 5.3 shows that the different path planners have varying degrees of success. The EHRC approaches the ideal case, the deep-RL planner coming in a close second, and the random path planner falls far behind.

The lower bound line is a result of Theorem 1 described in Chapter 3 that tells us how well the UAVs would perform under ideal conditions. In ideal conditions whenever a UAV has to guess as to which location is most likely to have the target the UAV makes the right choice. As a result we can see the ideal case quickly converges onto the lower bound of entropy. All other path planners will minimize the entropy of the system as best they can, but lacking perfect knowledge they will come short.
The EHRC yields the next best results over time as can be expected when doing an exhaustive path analysis. Where this planner falls short is during the initial search for the targets. When particles are distributed uniformly over the map the exhaustive path planner is forced to make random decisions. In comparison, the neural net initially does better by learning an efficient search pattern. In both of these cases the UAV has limited information about the target location and so cannot do as well as the ideal planner.

The other main limitation of both these path planners is their inability to scale well. The exhaustive planner’s runtime execution is $O(MNd^L)$ meaning that with a large enough complex map the exhaustive planner will fail to execute in real time. Conversely, regardless of map size the neural net is always $O(1)$. However, the neural net training time scales poorly with increasing map.
Figure 5.3: The ideal scenario and the lower bound are included for comparison. In the ideal scenario the UAV is always aware of the actual positions of the targets and flies directly between the two. The Lower Bound describes the average lower bound for entropy. We can see that the neural net path planner dropped in entropy faster than the exhaustive path planner. This is because the neural net learned efficient search patterns for uniformly distributed particles while the exhaustive path planner only made efficient decisions after it knew the general locations of the targets.

One of the ways the exhaustive planner could be modified to handle large maps would be to take advantage of simulated annealing [25] to reduce the search space required by a large map.

Many other path planners, such as jump point search [26] or Rollout Policy [27], provide alternative efficient ways to plan through large environments, each with their own advantages and disadvantages. It is also possible that modifications to the neural net could reduce the training time to an acceptable level for larger maps. As a proof of concept however, this shows that the Deep-RL planner is a competitive alternative.
Figure 5.4: Snapshot of the scenario implemented in hardware at 185s. The red asterisk represents the quadcopter. The green and blue diamonds are the current locations of each of the two vehicles. As can be seen, the green particle cloud is closely aligned with the actual position of its target. And while there are multiple modes for the blue target representing different estimates, the blue target is closely aligned with one of its modes. The black star and circle represents the estimate and error margin of a target sighting, which when funneled through our data association algorithm, performs a positive update on the green target.

5.2 Hardware

We used hardware to verify the efficacy of both the RBPF and the lower entropy bound algorithm. Flight tests were performed in a motion capture room using the road map shown in Figure 5.4. The figure shows a snapshot from the flight test 185s in. The UAV indicated by the red star has estimates for both targets and is in the process of performing a positive update on one of the targets. The black star represents the target sighting while the black circle shows the measurement covariance. Target one, represented by the green diamond, is within the black circle showing that our error covariance is accurately reflecting our sensor capabilities. The particle clouds represented by the clumps of green and blue stars show that the blue target’s estimate has higher entropy than that of the green target making it the focus of the path planner once the UAV reaches an intersection.

The UAV we used is shown in Figure 5.5. The frame was a DJI Matrice 100. The onboard computer was a TX2 which handled the vision processing as well as ran the ROScopter autopilot along with an F4 board which housed ROSflight. ROScopter used a PID controller with waypoint
following and near constant velocity was managed by saturating the error in the PID loop at .3 meters. An OCAM camera provided the visual feed at a rate of 60hz and a resolution of 640x480. Target detection was handled by inputting the camera data into Visual-MTT [17] with R-RANSAC [5]. This used a color detector to recognize the target and then transformed the camera frame coordinates into the inertial frame assuming a flat earth model. The camera feed had a latency of approximately 0.4 seconds which needed to be accounted for when doing the transformation into the inertial frame. This was accounted for by maintaining a history of the quadcopters location and orientation and using a fixed delay in the quadcopters position when transforming the target coordinates from the camera frame to the inertial frame. By accounting for this latency issue we were able to maintain an error less than half a meter on average for our estimate.

The targets are two small RC cars one of which is shown in Figure 5.6. They were augmented from their original design with an onboard raspberry PI and TX2 respectively. The computers were housed on a custom chassis based off the Donkey Car design [28]. They were then outfitted with silver balls that registered on the motion capture cameras which provided their coordinates in the inertial frame. To make the cars visible to Visual-MTT we wrapped them in red tape and gave the Color Detector the corresponding hue values to mask in the image.
Figure 5.6: The onboard computer is a TX2 running an autopilot based on the coordinates received from the motion capture cameras. The cameras locate the silver dots on the vehicle in order to give an estimate of the vehicle’s position. These coordinates are then converted into NED coordinates. The car drives Dubins paths along the predefined roadmap using an electronic speed controller (ESC) to control the motor.

The kinematics of the cars are of the form

\[
\begin{align*}
\dot{p}_n &= V \cos \theta \\
\dot{p}_e &= V \sin \theta \\
\dot{\theta} &= u,
\end{align*}
\]  

where \(\dot{p}_n\) and \(\dot{p}_e\) are the velocities along the north and east axis, \(V\) is the speed the car is traveling, \(\dot{\theta}\) is the angular velocity in the inertial frame, \(\theta\) is the car’s yaw in the inertial frame and \(u\) is the angle of the wheels.
Figure 5.7: The path of one of the vehicles is illustrated here. The Dubins path shown attempts to minimize the car’s deviation from the road network given the turn radius of the vehicle.

The car kinematics prevent them from executing perfect 90 degree turns and so they cannot perfectly conform to the road network. Dubins paths [29] allow the cars to minimize how much they diverge from the road network. When the desired path has turns with sharper angles than the vehicle is capable of making Dubins paths convert the path into straight lines joined by arcs with a radius equal to the turn radius of the vehicle. The length of the straight lines and the arcs are then chosen so as to minimize the deviations from the original path. Figure 5.7 shows a typical Dubins path for the car as it travels around a square. The car stays within 0.5 meters of the desired track. The cars used an independent stochastic planner that chooses its next direction randomly at each intersection. The planner provides waypoints for the Dubins paths. The vehicles then use a PID controller to maintain a desired heading and velocity provided by the Dubins path planner. Despite the PID controller the cars had a large variance in speed control which in turn impacted their turn radius. The variance in the speed control was largely due to the vehicles attempting to travel at a
The vehicles travelled at an average speed of 0.24 m/s and the quadcopter maintained an average speed of approximately 0.4 m/s. The RBPF showed an impressive ability to maintain accurate estimates and vehicle predictions despite a weak model of the RC cars. In Figure 5.9 the entropy from the hardware test is plotted against the lower bound. The plot is divided into three regions. In region A, the UAV is searching for targets. In region B, the UAV has located the first target and is looking for the second. In region C, the UAV is trying to minimize entropy across both
Figure 5.9: The entropy for the two targets as well as the steady state entropy for the hardware test is shown. In region A, the UAV is searching for targets. In region B, the UAV has located the first target and is looking for the second. In region C, the UAV is trying to minimize entropy across both targets. The steady state entropy of around 3.207 which is just slightly higher than the lower bound of 3.11 units of entropy. This is expected as on a small map the ERHC planner will work comparatively to the ideal planner.

targets. Region C gives us the steady state entropy as once both targets are located the UAV will maintain both estimates. The average steady state entropy was 3.207 which is just barely above the lower bound of 3.11. On this map, using corollary 1, that means target certainty was on average within a variance of 1 meter.
CHAPTER 6. CONCLUSIONS

The RBPF is a two-tier particle filter that uses the top level particles to maintain a history of sightings and thereby perform data association across a set of targets. The lower level particles use the data association provided by the top level particles to perform positive and negative updates on the corresponding lower level filter. In this fashion the most likely top level particle contains the current best estimate for the tracked targets. The RBPF acts as a basis for the rest of the work in this thesis.

We assume the road network is large enough that the UAV can’t keep all targets constantly in their field of view. Therefore, we need path planners. This thesis presents two path planners that use the RBPF to make decisions that maximize target location certainty. At each intersection the ERHC executes an exhaustive depth first search of the possible paths to identify and then execute the first step of the path that maximizes the number of particles encountered with the least distance traversed. The Deep-RL planner in contrast is trained using deep reinforcement learning. Once fully trained the Deep-RL planner is able to identify the best next waypoint in $O(1)$ time.

In simulation we have shown that a neural net trained using Deep-RL is capable of learning efficient map sweeping strategies when target locations are relatively unknown. An ERHC is still more efficient than the neural net trained for this thesis after the initial search and significantly improves tracking performance and target location certainty versus a naïve search pattern. However, it is limited by the computational requirements of exhaustively searching complex maps.

We have also developed a theorem for determining the lower-bound for the average-entropy of the RBPF. Particle Filter entropy is used as a unit of measurement as it gives a way of accurately comparing the precision of complex multi-modal estimates. This gives a reliable way of establishing the resources needed to accomplish mission objectives as well as providing a reliable method of determining the effectiveness of different multi-agent path planners.
We also implemented an end-to-end solution in hardware using a quadcopter as the UAV and two RC cars as the target. Inertial frame coordinates were provided using a motion capture system. The UAV used the ERHC path planner and the targets used independent stochastic path planners. We showed that an end-to-end solution on a small map in hardware is capable of steady state results near the lower bound.

6.1 Future Work

The work in this thesis could be improved in a number of ways. The RBPF could be augmented by employing a higher fidelity motion model for the targets, such as that of [8]. This would allow the RBPF to track vehicles with variable velocities and allow the RBPF to account for movement that deviated from the road network, a clear limitation shown in Figure 5.7.

In addition to improving upon the motion model, the RBPF filter could also be augmented to handle an unknown number of targets similar to the technique used in [12] for estimating the number of landmarks in Simultaneous Localization and Mapping (SLAM). Frequently when attempting to maintain surveillance over an area the number of targets that need to be tracked changes and often targets will enter and leave the area of regard when not in view. By augmenting the RBPF to handle an unknown and variable number of targets we can track targets both in scenarios with changing or unknown conditions. To accomplish this, the RBPF would need to determine whether the target in view is one previously seen or if a new vehicles has entered the system. It would also need a way to determine if vehicles had left the area of regard and so no longer needed to be tracked.

The Deep-RL planner described in Chapter 3 performed well when compared with the EHRC but one of its main limitations is that the neural net must be trained on a specific map. The planner could be enhanced by taking advantage of advances in transfer learning that would allow the neural net to work on multiple maps with a minimal increase in training time. Transfer learning is the discipline in machine learning that allows knowledge learned by a neural net in solving one problem to be leveraged towards similar issues [30]. While many of the advances in transfer learning are implemented in games, tracking varying numbers of targets on road networks have many similarities that could be effectively leveraged using knowledge gleaned from training the neural net on previous maps.
Finally, while Theorem 1 described in Chapter 4 proved extremely effective on the maps discussed in this paper there are a number of ways it could be improved upon. Complex maps will inevitably have long periods between target sightings resulting in different modes merging as well as splitting. If Lemma 5 were augmented to account for mode merging as well as splitting then the theorem would be viable on more complex road networks. The other main limitation posed to the theorem is identified in Assumption 5. Currently $\alpha$ must be calculated for each map, however there is a clear relationship between the average distance between nodes and the average distance between points on a graph. If that relationship were explicitly defined then the theorem could be applied to a wide variety of maps with an exact calculation of the average minimum round-trip time.
REFERENCES


60


