AFRL-AFOSR-UK-TR-2024-0008

Towards Game-Theory for Tracking (ToGT4T)

SIMON MASKELL
THE UNIVERSITY OF LIVERPOOL
BROWNLOW HILL
LIVERPOOL, , L69 7ZX
GBR

12/12/2023
Final Technical Report

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515 Box 14, APO AE 09421

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
20231212	Final	START DATE 20220701	END DATE 20230630

4. TITLE AND SUBTITLE
Towards Game-Theory for Tracking (ToGT4T)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
	FA8655-22-1-7021	61102F
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Simon Maskell, Rahul Savani

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
THE UNIVERSITY OF LIVERPOOL
BROWNLOW HILL
LIVERPOOL L69 7ZX
GBR

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
EOARD UNIT 4515 APO AE 09421-4515	AFRL/AFOSR IOE	AFRL-AFOSR-UK-TR-2024-0008

12. DISTRIBUTION/AVAILABILITY STATEMENT
A Distribution Unlimited: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This is the final report for the 'Towards Game Theory for Tracking' project, funded by AFOSR via grant number FA8655-22-1-7021. A particle filter was used to estimate states of a moving target. The assumed behaviour of the target included the capacity to attempt to evade being detected by the sensor. The experimental results show some ability to identify evasive behaviour. In situations involving pronounced obscuration, the are too few detections to make meaningful inferences. Recommended future work includes consideration of a more sophisticated particle filter, Monte-Carlo roll-out and reinforcement learning.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	c. THIS PAGE		
U	U	U	SAR	43

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
CHARLTON LEWIS	3142356045

Standard Form 298 (Rev.5/2020)
Prescribed by ANSI Std. Z39.18

Towards Game Theory for Tracking: Final Report

Yifan Zhou and Simon Maskell

Research Title: Towards Game Theory for Tracking
PI: Prof Simon Maskell, University of Liverpool
Grant Number: FA8655-22-1-7021
Period of Performance: 1 July 2022 - 30 June 2023

September 2023

Contents

List of Figures

List of Tables

Executive Summary

This is the final report for the 'Towards Game Theory for Tracking' project, funded by AFOSR via grant number FA8655-22-1-7021.

A particle filter was used to estimate states of a target moving from one source node to the destination node along a weighted road network. The weights were defined to make it possible for the assumed behaviour of the target to involve evading being detected by the sensor. The experimental results show some ability to estimate whether a target is being evasive and some robustness to the presence of missed detections. However, in situations involving pronounced obscuration, the are too few detections to make meaningful inferences.

It is recommended that a forthcoming publication capture the work undertaken to date together with future work that considers further experiments, multiple targets, a more sophisticated particle filter and the use of Monte-Carlo roll-out in the planner. Longer term, it is identified that it would be interesting to explore the use of reinforcement learning as a solution methodology.

Section 1

Introduction

1.1 Contractual Context

This is the final report for the 'Towards Game Theory for Tracking' project, funded by AFOSR via grant number FA8655-22-1-7021.

1.2 Report Organisation

The report follows the required structure such that after this introductory section, section 2 describes the methods, assumptions and procedures. The results and discussion are then described in section 3 while conclusions and recommendations are given in section 4.

Section 2

Methods, Assumptions and Procedures

2.1 Assumptions

This report assumes scenarios that a target moves from a source node to a destination node along a road network following a behaviour policy. We assume there is a sensor that moves with a constant velocity model. The sensor can detect the geodetic position (i.e. latitude-longitude) of the target when the target is visible. The road network and the position of the sensor are both known to the target.

It is assumed that the detection mechanism is such that the sensor creates a ray which, when it hits the target results in a detection such that the target can be obscured by any object between the sensor and the target. It is assumed that such objects are buildings and terrain. It is also assumed that the rays are drawn at same time and omni-directional which means the detection is collected regardless of the target's position at a precise time point. In the reported results, it is assumed that if the target is visible, the probability of detection is 100% (though this is actually a parameter and can be changed).

The target has the information of the road network and the exact position of the sensor such that the visibility of the sensor can be computed by the target for online path planning. The target has two behaviours: moving to the destination quickly; avoiding being detected by the sensor. Two models are therefore introduced for the behaviours: the shortest path and the path gives minimum cost when transitioning the visible area incurs an additional cost. Dijkstra's algorithm is applied for both models to find the path with minimum cost. However, the target can change which model to follow at any time.

For the tracker, the source node of the target is known, but the destination node is not. A number of confuser destinations are defined which could be any node other than the true destination node. In order to limit the computational cost for the experiments, the number of confuser destinations is limited and pre-defined.

2.2 Methods

2.2.1 Three-dimensional City Model

In order to generate the visibility map using a ray-tracing algorithm, a 3D city model needs to be generated. The 3D model considered here consists of two parts: the terrain and the buildings. The raw terrain data was acquired from United States Geological Survey (USGS). The NASA SRTM3 SRTMGL1 data (which belongs to the NASA SRTM Collections) was downloaded[1]. The data is free to use (for academics). The terrain data is provided in geoTiff format and then converted into point clouds. In this report, an area of roughly 3×3 km (centered on the University of Liverpool) is considered in all the experiments. The point cloud was connected by using Delaunay triangulation to extract triangles. There are a total of $126,100$ triangles and $63,570$ points in the terrain model. An exemplar visualisation of the terrain model is shown in figure 2.1.

In terms of the buildings, unfortunately there is no public database that contains precise building information. An alternative solution adopted here is to download the locations and classes of the buildings via the OpenStreetMap (OSM) API, and then fill the heights of the buildings using pre-defined rules. In this case, there are 101 classes of buildings. Some examples of the class names and the associated pre-defined heights are:

- Default: 10m
- Apartment: 30m
- Cathedral: 10m
- Terrace: 10m

A total of $181,218$ buildings were obtained from the OSM data. Note that each building can be represented by an arbitrary number of points. The point cloud of each building is defined by a number of polygons and are added onto the terrain model (above the terrain). Linear interpolation is used to create a dense mesh of the terrain to obtain the height of the ground where buildings locate.

As the result of the pre-processing described above, the city model that results comprises $488,275$ triangles and $293,216$ points. An exemplar visualisation of the city model is shown in figure 2.2.

2.2.2 Road Networks

2.2.2.1 Representation by node-edge

The road network is extracted using OSM data. Only "highway" (which is the main keyword used for identifying any kind of roads or streets) is used as the keyword to download the data: this means that the data does not include some other kinds of paths (such as those for pedestrians). A JSON parser in MATLAB is then used to extract the nodes and edges. Every node has a geodetic (latitude-longitude) coordinate. The edges are represented by a pair of nodes using a tuple such that, for example, edge (1, 2) is the edge between node 1 and 2. One-way traffic is not considered: every edge is considered to be bi-directional. Each edge then has two attributes: length and weight. As the name assumes, length is the euclidean distance between the two end nodes. Weight can be

[1] https://lpdaac.usgs.gov/products/srtmgl1v003/

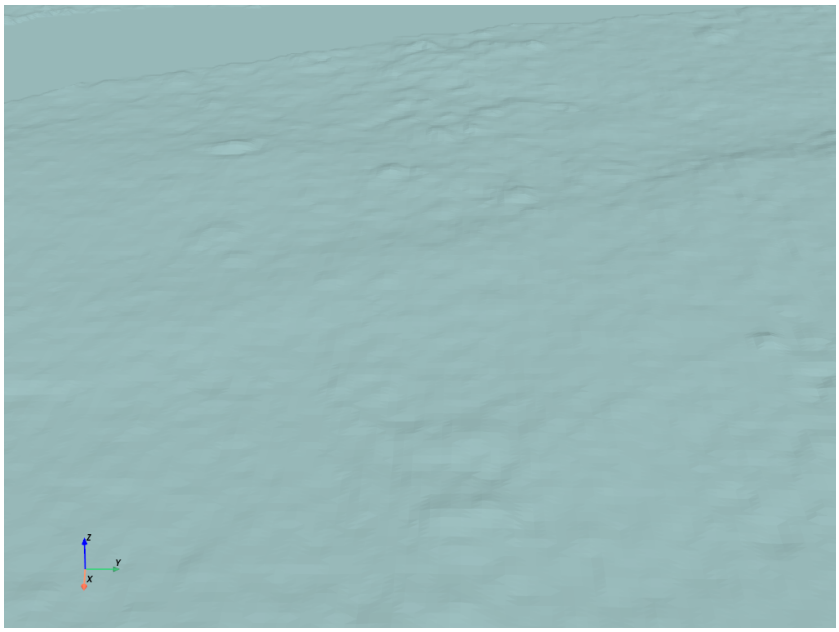


Figure 2.1: The 3D model of the terrain centered by the University of Liverpool.

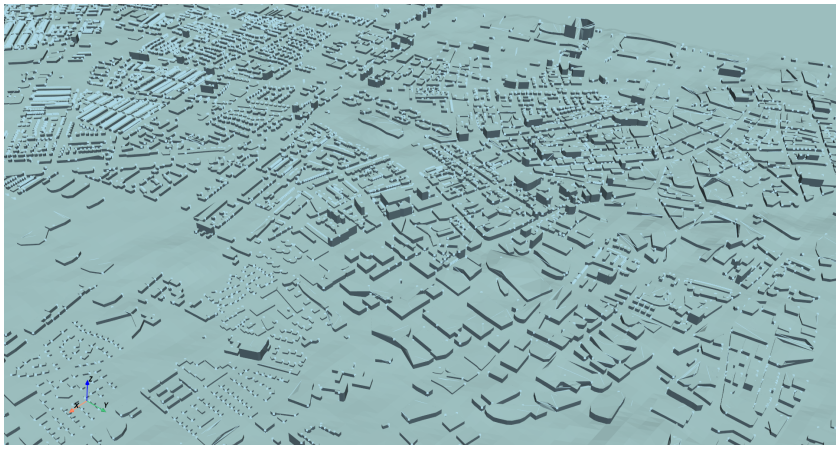


Figure 2.2: The 3D model of the terrain and buildings centered by the University of Liverpool.

Figure 2.3: The road network represented by edges and nodes.

defined and updated at any time. The weights are particularly used to calculate the path between two nodes using Dijkstra's algorithm. When calculating the shortest path between two given node, weight is considered to be equal to the length. The implementation of the graph capitalises on `networkx` [2], which is a network analysis tool in Python.

The road network of the considered area is shown in figure 2.3.

2.2.2.2 Representation by points

As well as the node-edge representation, the road network is also represented by points along the edges and at the nodes. The points are extracted based on the node-edge representation. To do so, the geodetic node coordinates are firstly converted to North-East-Down coordinates with a pre-defined centre point. As the edges are always straight lines by definition, the points along the edges are calculated by applying the equation of straight lines. The maximum distance between two extracted points can be adjusted. In the cases considered herein, the maximum distance is around 10 metres. This results in $182,225$ points being used to describe the city road network considered. Note that, in addition to the node-edge representation, each point has altitude information (which is crucial to calculating the visibility information).

The road network is shown in figure 2.4.

2.2.3 Visibility Map and Ray-tracing

The visibility map includes all the points on the road network that can be observed by the sensor. Generating the visibility map makes use of the point representation of the road network and the ray-tracing algorithm.

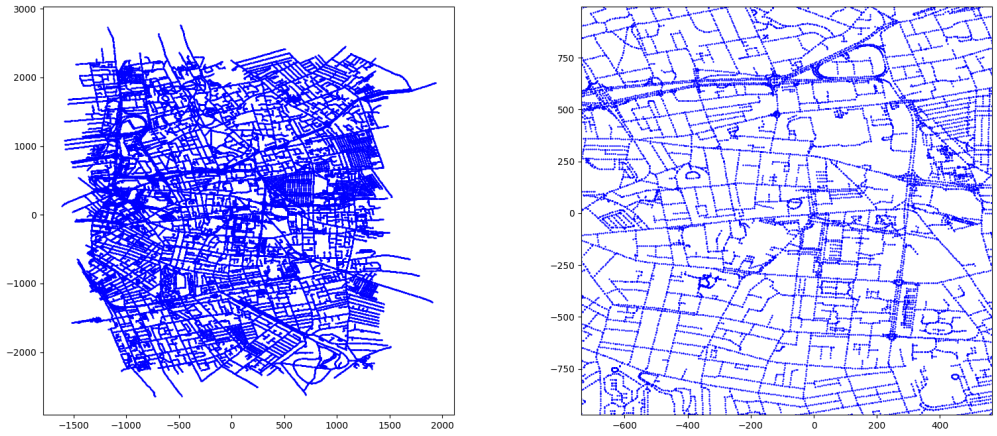


Figure 2.4: The road network represented by points. (left) total network. (right) magnified version.

The basic idea of ray-tracing is to check if a straight line which is defined by the start and end points intersects with triangle(s). A well-known algorithm for performing this check is called the Möller–Trumbore intersection algorithm. By giving one ray and many triangles, this algorithm calculates the intersections with all the triangles. The algorithm performs all checks and is therefore slow in a large environment. A preliminary evaluation by the authors showed that while using MATLAB to implementation such an algorithm, obtaining one visibility map takes longer than 5 hours using one CPU core.

To reduce the run-time, two approaches were considered. One is to implement the algorithm in a more sophisticated way in C++ and enable multi-threading. While this did give an improvement in run-time, the large number of triangles considered meant that this approach had limited utility. Another approach is to exploit a spatial data structure relevant to the 3D model and so to use a tree structure to store the triangles. There are a number of candidate tree structures and open-source implementations available. `Pyvista` [3] is a Python package that helps to generate the aforementioned 3D city models. It used OCtree and is implemented in C++. By using `Pyvista`, the total runtime for one visibility map can be reduced to 4 minutes (with the same computer as considered with the MATLAB implementation mentioned above). Although this is a great improvement, it is still insufficient to enable real-time processing. Fortunately, Intel's `embree` library [5] has been developed by Intel with the aim of highlighting the ray-tracing ability of Intel chips. `embree` provides more a sophisticated implementation with better tree structures. It uses a Bounding volume hierarchy (BVH) tree (which is the state-of-the-art and widely used in the computer graphics community). Using this library, generating one visibility map takes around 0.5 seconds at the first iteration (when the library builds the BVH tree) and takes around 0.2 seconds to generate subsequent maps (if the 3D model is identical). We note that this library only supports Intel/AMD CPUs.

Some examples of different generated visibility maps resulting from different altitudes for the sensor are shown in figure 2.5. It can be observed that when the altitude increases, the visibility

area becomes larger. Note that sensor altitude is expressed in terms of height above sea level and the average altitude of Liverpool city centre is around 50m.

2.2.4 Path Planning

As mentioned in section 2.1, the target can choose one of two models when the choice of next edge should be made. In this report, a simple planning strategy is used which identifies the path that obtains the smallest cumulative cost using Dijkstra's algorithm. When the cost of each edge equals to the length of the edge, this strategy is equivalent to the shortest path (Model 1). To hide from the sensor (Model 2), one can increase the cost of the edge if it is visible to the sensor. Since the lengths of the edge vary, it is not fair to attach the same cost penalty when only a part of a long edge is visible. To address this, we capitalise on the point representation, such that one edge is comprised of a number of points where that number is correlated with its length. The weight of the edge is therefore computed as follows:

$$W_e = |e| + N^{vis}(e) \cdot C \tag{2.1}$$

where $|e|$ is the length of the edge (e) and $N^{vis}(e)$ is the number of visible points on the edge. C is the penalty applied for the path including each visible point. If $C = 0$, Model 2 and Model 1 are the same and both models consider the shortest path. In the subsequent experiments, $C = 1000$ for Model 2.

This formulation of the cost ensures that the visibility is reflected in the cost, as well as there being a difference between two edges that have differ in terms of how much of the edges are visible.

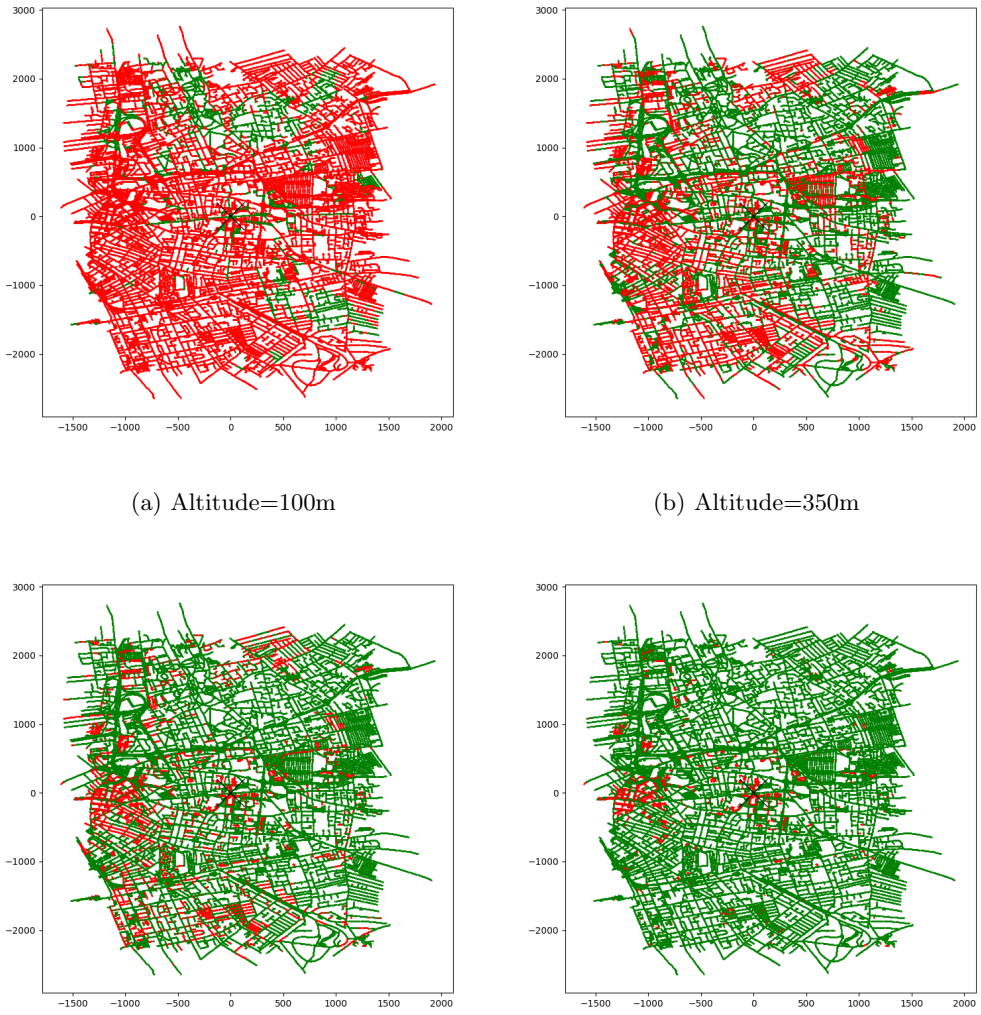
2.2.5 Particle Filter (PF) and Sequential Monte-Carlo (SMC) Sampler

PF [1] uses a number of particles to model the probability density over time of a target. It is particularly useful when the probability density is non-linear and non-Gaussian. As the target state considered in this report contains the categorical behaviour (e.g. destination, edge identity and model index) of the target and since the model we will assume considers a decision making model for the target's behaviour, the state is obviously non-linear and non-Gaussian such that the PF is an appropriate choice of algorithm to use.

2.2.5.1 State Vector

The state vector of the target is denoted by $S_t = \{r, \dot{r}, e, n, d, s, m\}$, where the parameters are explained as follows.

1. Range (r): The range of the target from the node n on edge e.
2. Speed ($\dot{r}$): The speed of the target. It is a scalar and the direction is from node n, to the other end node on edge e.
3. Edge (e): The edge that the target is currently on. If the target is on a node exactly, e is the next planned edge.
4. Node (n): The current starting node of the target.
5. Destination (d): The destination of the target.



(a) Altitude=100m

(b) Altitude=350m

(c) Altitude=1050m

(d) Altitude=5050m

Figure 2.5: The visibility map generated when the sensor locates at (0,0) with different altitudes. Green indicates that a point is visible to the sensor. Red indicates that a point is not visible to the sensor.

6. Source (s): The source node of the target.

7. Model index (m): The model of the target that is used to plan the path (see section 2.1).

2.2.5.2 State Prediction and Transition Model

The states of the particles are predicted at the beginning of each time-step. The first step is to plan the future path. Path planning runs Dijkstra's algorithm considers the target behaviour (and is parameterised by n_t, d and the corresponding graph indicated by m_t).

The second step for state prediction is to propagate the range (r) as follows:

$$\begin{bmatrix} r_{t+1} \\ \dot{r}_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \Delta T \end{bmatrix} \cdot \begin{bmatrix} r_t \\ \dot{r}_t \end{bmatrix} + \mathcal{N}\ (0, Q) \tag{2.2}$$

$$Q = \begin{bmatrix} \frac{\Delta T^4}{4} & \frac{\Delta T^3}{2} \\ \frac{\Delta T^3}{2} & \Delta T^2 \end{bmatrix} \cdot \sigma^2 \tag{2.3}$$

where ΔT is the time interval between two time-steps; $\mathcal{N}(.)$ is Gaussian noise; Q is the variance of the process noise and $\sigma^2 = 10^{-6}$ for all experiments in this report.

If r_{t+1} is smaller than the length of the edge ($|e_t^i|$), the target stays on the edge such that other parameters are not modified. If r_{t+1} equals to the length of the edge, the target is considered to enter the next edge with r being reset to 0 and n being updated. If r_{t+1} is larger than the length of the edge, the target enters the next edge with a residual: $\tilde{r}_{t+1} = r_{t+1} - |e_t^i|$ and n being updated. It is also possible that $\tilde{r}_{t+1}$ is still larger than the length of next edge (e.g. when the edge is short or the time step is long). In this case, this process will iterate along the planned path, for edges $e_t^1, e_t^2, ...$, until $\tilde{r}$ is smaller than or equal to the length of an edge.

2.2.5.3 Measurement Model

The measurement model is a non-linear transformation from the range (r) and edge (e) to a geodetic coordinate (y). The measurement noise considers a Gaussian noise on both the latitude and longitude axis. The way the measurement is calculated is as follows:

$$y_k = \begin{bmatrix} e^{lon}(s) \\ e^{lat}(s) \end{bmatrix} + \begin{bmatrix} r \cdot cos\theta \\ r \cdot sin\theta \end{bmatrix} + \begin{bmatrix} \mathcal{N}(0, 10^{-7}) \\ \mathcal{N}(0, 10^{-7}) \end{bmatrix} \tag{2.4}$$

where

$$\theta = tan^{-1} \frac{e^{lon}(e) - e^{lon}(s)}{e^{lat}(e) - e^{lat}(s)} \tag{2.5}$$

where: $e^{lon}(s)$ is the **lon**gitude of the **s**tart node of edge (s); $e^{lon}(e)$ is the **lon**gitude of the **e**nd node of edge (e); $e^{lat}(s)$ is the **lat**itude of the **s**tart node of edge (s); $e^{lat}(e)$ is the **lat**itude of the **e**nd node of edge (e); θ is the direction of the edge.

2.2.5.4 Extracting Estimates of the Target State

The process of extracting estimates of the target state has two aspects.

The calculation of the estimated position uses the weighted mean of the measurements over all the particles.

However, since some of the parameters (e.g. e, n, d, s, m) in the state vector are categorical, for each of these parameters, the sum of the normalised weights is first calculated. Then a vote based on the maximum sum is made to decide the estimate to report.

2.2.5.5 SMC Sampler for Sampling Destinations and Models

Since the measurement model is only associated with the current edge and range, the updater in the particle filter only updates the particle weights based on these two values. This means that the particles that indicate the true destination may not have the highest weight and can therefore be removed by the resampler at early time steps. A mechanism has been introduced to recover the particles and to add diversity to the destinations considered by the filter. The SMC sampler adopted is that initially described in [4]. In this paper, the particles re-simulate the destination based on which edge it is on. Candidate destinations are required to start from the source and pass through the current edge following the shortest path.

In this report, because the path planning should be done at every iteration, the potential destinations and intermediate edges are numerous. The limit on candidate destinations considered in [4] has therefore been removed such that all destinations (i.e. confuser and true destinations) can be candidates. In this case, the probability of conducting re-simulation is reduced to ensure the tracker remains stable.

The same idea is used for model selection. The tracker starts by sampling the model with equal probabilities for each model. At each time step, the model index can be re-simulated with the planned path being changed for the particle if the model changes. This process is similar to what is called Monte-Carlo roll-out in reinforcement learning. However only one future time step is considered.

2.3 Procedure

The overarching procedure of the proposed algorithm is shown in figure 2.6 and includes two parts.

The first part initialises the environment, simulates the sensor movement, generates all the ground-truth and detection points. It runs only once at the start. The building and terrain information is downloaded and processed into structured data given the area of interest. The code is implemented in MATLAB with data that is stored locally. The simulator simulates the movement of the sensor and the target based on all the aforementioned rules. It also makes use of the visibility map generator and graph processor for planning the paths.

The second part of the procedure runs iteratively over time. The state vector of the particle filter is initialised as follows:

- $r_0 = 0$
- $\dot{r}_0 = |\mathcal{N}(0, 10^{-6})|$
- e is the next planned edge according to d, s and m.
- n is the source node: $n = s$.
- d is the drawn from all the destinations using multinominal sampling (with equal weights).
- s is the source node.

- m is the drawn from 1,2 using binominal distribution with equal weights.

At each subsequent time-step, each particle predicts the state of the target first. The PF updates the particles if a detection is received and outputs an estimation of the target. During the prediction, the visibility map is re-calculated and the road network graph is updated if the position of the sensor changes. If not, the paths calculated by the graph processor are recorded for use at subsequent time steps. By exploiting the property of the shortest (or minimum cost) path, once a path is discovered, the shortest (or minimum cost) paths between any two intermediate nodes are also determined. For example, if the shortest path between node A and H is

$$A \rightarrow C \rightarrow D \rightarrow F \rightarrow H$$

the shortest path between C and H is therefore

$$C \rightarrow D \rightarrow F \rightarrow H$$

This property is also used in Monte-carlo tree search and it is helpful to reduce repetitive calculation if the graph remains identical.

The updating process follows the standard updating method in a PF. Particles are resampled if the effective sample size (ESS) falls below a threshold at the end of an iteration: Particle re-simulation is useful to ensure diversity in the particles. As will be shown in the results, the improvement offered by re-simulation is particularly significant when encountering missed detections for several consecutive time steps.

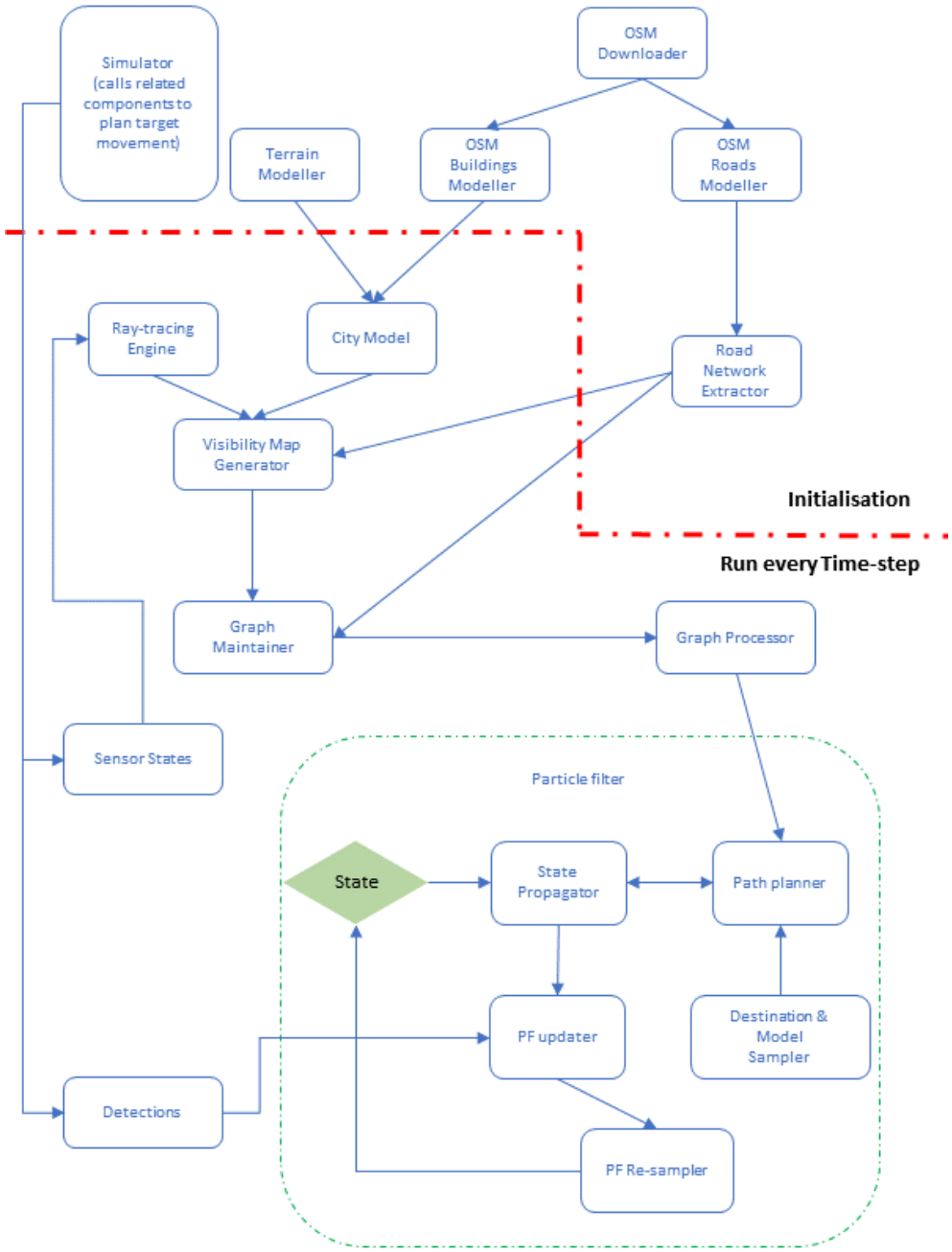


Figure 2.6: A summary of the algorithmic procedure.

Section 3

Results and Discussion

In this section, the following four considerations will be investigated:

1. The ability to estimate the path of a target that aims to avoid being detected by a static sensor (i.e. a target that always follows Model 2 rather than Model 1).

2. The distribution of the particles after encountering a number of consecutive missed detections when the target always follows Model 2.

3. The ability to estimate the path of a target when it switches between two models while the sensor is static.

4. How a moving sensor changes the estimated paths of the target when the target follows Model 2.

In order to make sure that the investigations are clear, the source and the destination nodes of the target are always fixed. The confuser destinations are randomly chosen, because they will not influence how the true paths are being planned. Considering such random confuser destination makes it possible to examine the robustness of the tracker. The default setting of the experiments are reported in table 3.1 with any departures from these settings highlighted in the descriptions or captions in the following subsections.

3.1 Visual Output

The visual output of the code used in this report is shown in figure 3.1. The visual output consists of four views and the description of each of these views is as follows:

- **Global view** displays the entire road network. It includes 1) the positions of the particles; 2) the true path (which is generated by the simulator); 3) paths planned by Model 1 and 2 (which start from the next node to the destination); 4) the position of the true destination; 5) the positions of the confuser destinations; 6) the position of the sensor; 7) the ellipse described the uncertainty in the estimated destination.

Variable	Value	Remarks
Number of confuser destinations	30	-
Number of particles	500	-
Source	$52.426, -2.975$	(Latitude, Longitude); The nearest node is used as the source node.
True Destination	$53.391, -2.957$	(Latitude, Longitude); The nearest node is used as the destination node.
Speed	5×10^{-4}	Geodetic coordination.
Sensor Initial State	250,0,350	(North, East, Altitude).

Table 3.1: The default settings for the experiments.

- **Zoomed view** magnifies the global view by 3 times. This area is centered on the estimated target. It is used to clearly show the particles around the estimated target and the position of the current measurement.
- **Model weights** shows the summation of the weights for each model. This graph mainly reflects which model is dominating the tracker at the current time-step.
- **Destination distribution** shows the count of particles per destination. This graph reflects estimation of the destination. The true destination is highlighted in purple.

3.2 Static Sensor and Constant Model

In this section, a static sensor is considered and the behavioural model for the target is considered to be constant. When using the default experimental settings and considering a target that always follows Model 1 (i.e. takes the shortest path to the destination) which is shown in figure 3.4(a), it takes 79 time steps to reach the destination and the number of misseddetection is small. Figure 3.2 shows the RMSE of the tracker estimation (as quantified in the measurement space). The green line shows no significant fluctuations all the time even around times when missed detections are encountered. To reduce the area that can be seen by the sensor, the altitude of the sensor was lowered to 150 then to 100 metres. When the altitude is 150 metres, many consecutive missed detections occur between time step 30 and 42. During these periods, the RMSE increases significantly. When the target is detected, the RMSE recovers: the particle distribution is able to recover and represent the true distribution. This case is shown in figure 3.3. When the altitude of the sensor further reduces to 100 metres, due to the small number of detections, the tracker cannot successfully track the target. Although the filter re-acquires the target at time step 42, the track is subsequently lost for the rest of the time steps.

If the target's behaviour model is always model 2 (which tries to avoid being detected by the sensor), the path is as shown in figure 3.4(b). Recall that the sensor visibility is shown in figure 2.5(b). It can be noted that the South-West area of the map is largely invisible to the sensor. For this reason, the target takes a path through this area. The total time steps used to get to the destination is 178 (which is much more than it takes using Model 1). The final track is shown in figure 3.5. It can be seen that the track is obviously incorrect in the South-West of the road

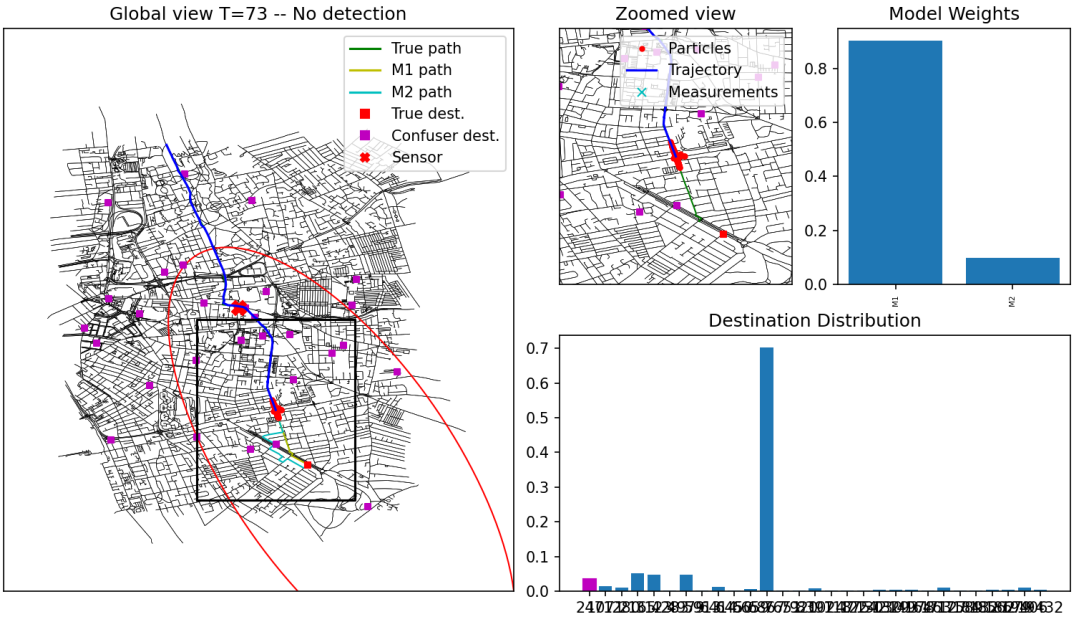


Figure 3.1: An example of the visual output of the program.

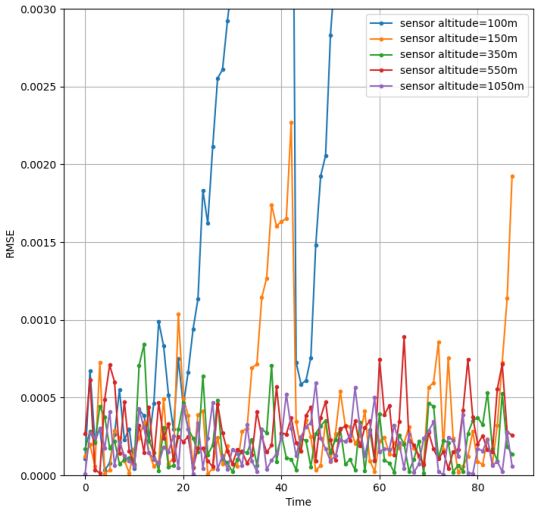
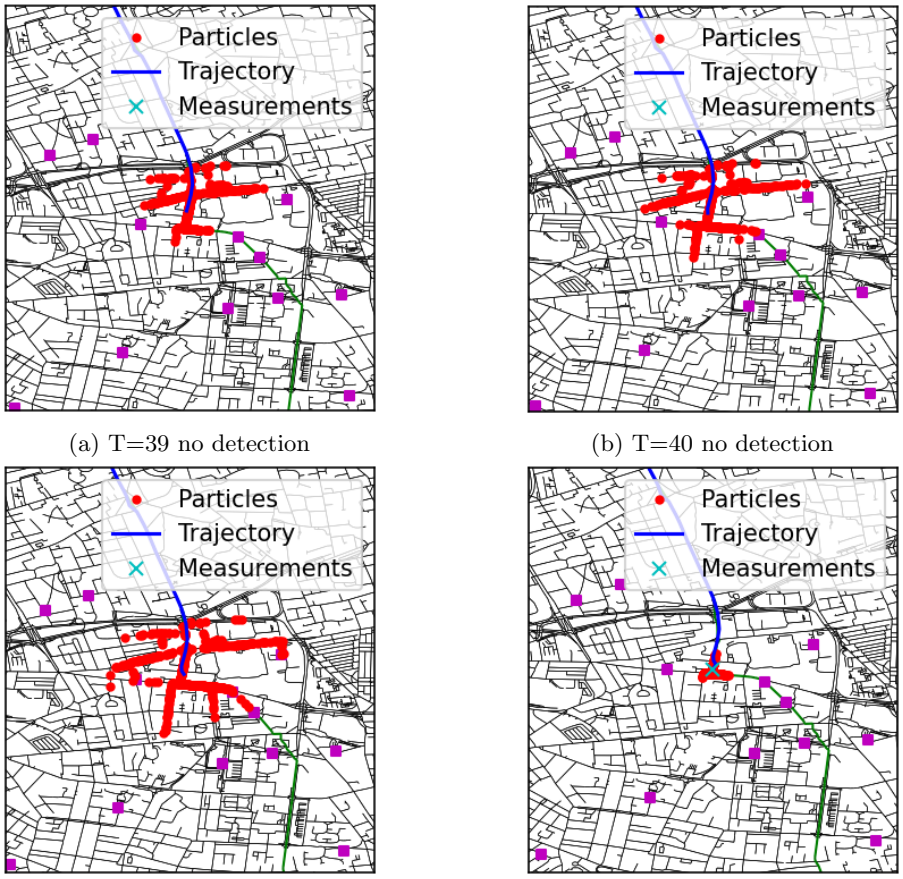


Figure 3.2: The RMSE of the tracker when the static sensor is placed at different altitudes. The target always follows Model 1.



(a) T=39 no detection

(b) T=40 no detection

(c) T=41 no detection

(d) T=42 detection is marked by cyan cross

Figure 3.3: An example of the particle distribution when missed detections occur and a detection is received at the final time step.

network (when no detections are received). Fortunately, the estimation exhibits some ability to recover from the small number of detections that are eventually received. However, this is mainly because (fortunately) there are a few particles that happen to arrive at the destination.

(a) Target always follows Model 1. (b) Target always follows Model 2.

Figure 3.4: The true path (green) of the target following two models. The position of the sensor is marked by the red cross. Note that the source node is located at the Northern end of the map and the true destination node is located towards the Southern end of the map.

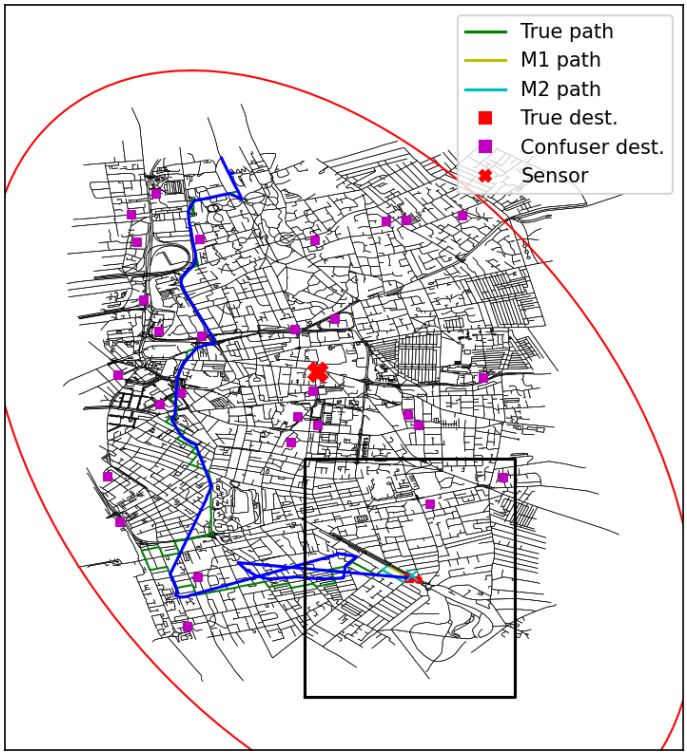


Figure 3.5: The final tracker output when the target always considers Model 2.

3.3 Switching between two Models

While in section 3.2 the behavioural model remains the same over the entire experiment, this is actually not necessary to plan the future path for each particle. To enable evaluating such a property of the scenario, we consider that the target switches its behaviour model at two time steps. The target starts with Model 1 and then switches to Model 2 at time step 25. Then, at time step step 100, the target switches back to model 1 and then takes the shortest path to the destination. The true target path is shown in figure 3.6(a). Compared to figure 3.4(a) and (b), it is evident that the target follows the fastest path at first, and then diverts to the area when the sensor's view of the target is obscured. After the target switches back to model 1, the target follows a new (shortest) path to the destination. The aim is that this scenario shows the effect of online planning. Note that the target takes a total of 145 time steps to reach the destination.

(a) The target switches models and the sensor is static.

(b) The target switches models and the sensor moves.

Figure 3.6: The true path (green) of the target that switches between two models and the sensor movement is also considered. The position of the sensor's final position is marked by the red cross.

Figure 3.7 shows the RMSEs at each time step. When the target switches to model 2 at time step 25, there is not a marked increase in the RMSE. The reason is that the particles can still predict the target states given their candidate paths. However, the RMSE does increase after time step 80 since at this point the variance across the particles accumulates and that influences the accuracy significantly. When the target is detected afterwards, the track recovers, and, with the help of re-simulation, the RMSE is reduced.

By monitoring the sum of the weights per model index (see figure 3.8), the tracker can be seen as a classifier for the target's behaviour. The output of the classifier appears broadly consistent with the ground-truth: the curve for Model 2 is always above the curve for Model 1 between time steps 25 and 100. However, it is also obvious that the classification probability decays unexpectedly.

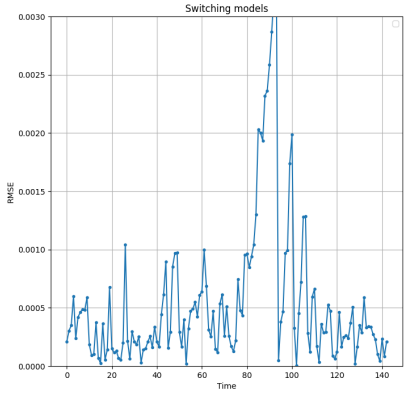


Figure 3.7: The RMSEs at each time step when the target switches models at time steps 25 and 100. The sensor is static.

It seems that the missed detections cause the two probabilities to equalise. This is because the model index re-simulation is applied in the prediction step and is purely random, such that the model assumes no mechanism to preserve the historical information on model index. This issue particularly matters when the target is invisible for many consecutive time steps (since such an event is unlikely under the assumed switching model).

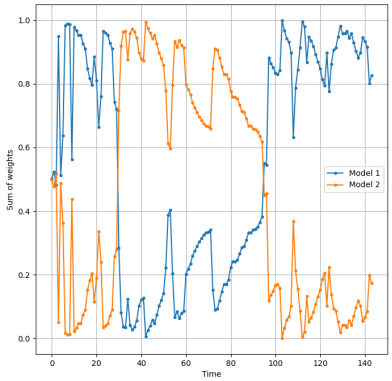


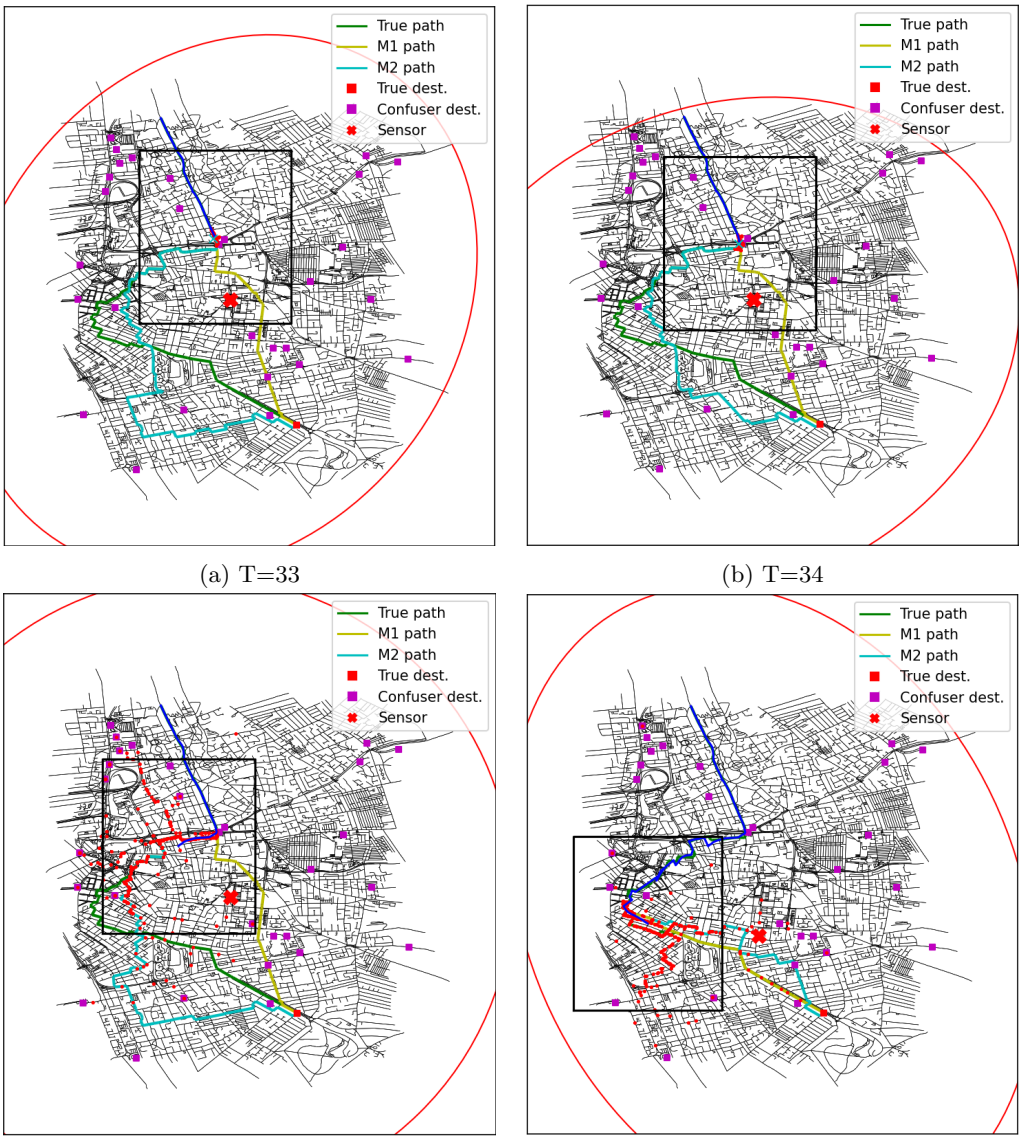
Figure 3.8: The sum of the weights of two models as a function of time. The target switches models at time steps 25 and 100.

3.4 Moving Sensor with a Target that is Switching Between two Models

To further exploit the ability of online path planning and visibility map generation, the sensor is set to move following a constant velocity in this section.

In the first experiment, the target is still performing behaviour switching. The velocity of the sensor is 10 metres per second due South. Figure 3.6(b) shows the true path of the target and the final position of the sensor. Comparing to figure 3.6 (a), it is apparent that the initial paths are the same (since in both cases, the target follows model 1). After several time steps, it is apparent that because the sensor moves towards the South, the target postpones its diversion accordingly (since it does not need to avoid areas that would be visibile to the sensor). However, the path moves further to the West as the sensor moves South. Note that at the end of the simulation, the target switches back to model 1 and then moves along the shortest path to the destination.

More detailed examples of how the online path planing works are shown in figure 3.9. All four images consider using Model 2 (which is influenced by the visibility map) throughout: the planned paths are shown by cyan lines. The differences between figure 3.9 (a) and (b), show the path changes immediately when the sensor moves to a new position. Figure 3.9 (c) and (d) show different planned paths associated with the state that is estimated at a time when there is no detection. It is obvious that the planned path is very different compared with Figure 3.9(a) since the change of the sensor position is pronounced.



(a) T=33 (b) T=34

(c) T=46 (d) T=95

Figure 3.9: Tracking results and the planned future path at different time steps. The target switches models and the sensor moves with a constant velocity.

Section 4

Conclusions and Recommendations

4.1 Conclusions

In this report, a particle filter was used to estimate states of a target moving from one source node to the destination node along a weighted road network. A movable sensor is involved in the implementation whose visibility is also modelled given an approximated 3D model of a real city. Further investigation has considered the target adopting online path planning that avoids being detected by the movable sensor. Moreover, the model which the target is following is also integrated in the tracker which can then classify the probability of whether the target was deliberately avoiding the sensor. The experimental results show that the track can estimate the position of the target as well as the destination and the behaviour model. The particle filter has shown robustness to consecutive missed detections. However, it is also observable that in extreme cases, that there are too many missed detections and the model classification becomes inaccurate with the methods considered.

4.2 Recommendations

The following recommendations emerged while implementing the algorithm and conducting the experiments and would sensibly be considered in the context of an open-literature paper that builds on the work described herein:

1. Further experiments could be conducted to assess the impact of, for example, using fewer particles or more frequent model switching.

2. The method could be extended to consider a multi-target tracking scenario.

3. The particle filter considered could be improved to use an improved proposal distribution and, more specifically, a fixed-lag variant of the particle filter. This would enable the filter to revisit historic states in the light of subsequent data. This is likely to result in significant benefits when considering the processing of the first detection after prolonged periods during which no detections are received.

4. The current use of Dijkstra's algorithm could be replaced with Monte-Carlo roll-out. This would allow the path planning to assume future changes to the behavioural model adopted by the target and future movement of the sensor, neither of which are considered at present.

In the longer term, rather than assuming a model for the target that involves planning given unambiguous knowledge of the sensor's future motion, one could seek to consider a target that itself models the uncertainty associated with the planning undertaken by the sensor. It is likely that extending the approach in this way would lead to an intractable solution. An alternative would be to use reinforcement learning to attempt to learn a recursive mapping of historic data that can inform decision making by the target. Adopting an approach involving two agents, with each attempting to learn how to outplay each other would be an interesting avenue to explore.

Acknowledgements

The authors are grateful to the guidance provided by the AFRL researchers involved in monitoring progress and by Sean O'Rourke in particular. The authors are also grateful for assistance provided by colleagues at the University of Liverpool and, in particular, by Lyudmil Vladimirov, Dipanjan Saha, Michael Wright and Al Phillips.

Bibliography

[1] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[2] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[3] C. Bane Sullivan and Alexander Kaszynski. PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, may 2019.

[4] Lyudmil Vladimirov and Simon Maskell. A smc sampler for joint tracking and destination estimation from noisy data. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2020.

[5] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. Embree: a kernel framework for efficient cpu ray tracing. *ACM Transactions on Graphics (TOG)*, 33(4):1–8, 2014.

List of Symbols, Abbreviations and Acronyms

United States Geological Survey (USGS)

Open Street Map (OSM)

Application Programming Interface (API)

Bounding volume hierarchy (BVH)

Particle Filter (PF)

Sequential Monte-Carlo (SMC)

Effective Sample Size (ESS)

Root Mean Square Error (RMSE)

Probability Density Function (PDF)

Report Documentation Page

1.Report Date	2.Report Type	3.Dates Covered	
		a.Start Date	b.End Date
September 2023	Final Report	1 July 2022	30 June 2023
4.Title and Subtitle			
Towards Game Theory for Tracking: Final Report			
5a.Contract Number	5b.Grant Number	5c.Program Element Number	
	FA8655-22-1-7021		
5d.Project Number	5e.Task Number	5f.Work Unit Number	
6.Authors			
Yifan Zhou and Simon Maskell			
7.Performing Organization Name(s) and Address(es)		8.Performing Organization Report Number	
University of Liverpool Brownlow Hill, Liverpool, L69 9GJ, UK		ToGT4T/1	
9.Sponsoring/Monitoring Agency Name(s) and Address(es)		10.Sponsor/Monitor's Acronyms	11.Sponsor/Monitor's Report Number(s)
EOARD, UNIT 4515, APO AE 09421-4515		AFRL/AFOSR IOE	FA8655-22-1-7021
12.Distribution/Availability Statement			
A. Distribution Unlimited PB Public Release			
13.Supplementary Notes			
None			
14.Abstract			
This is the final report for the 'Towards Game Theory for Tracking' project, funded by AFOSR via grant number FA8655-22-1-7021. A particle filter was used to estimate states of a moving target. The assumed behaviour of the target included the capacity to attempt to evade being detected by the sensor. The experimental results show some ability to identify evasive behaviour. In situations involving pronounced obscuration, the are too few detections to make meaningful inferences. Recommended future work includes consideration of a more sophisticated particle filter, Monte-Carlo roll-out and reinforcement learning.			
15.Subject Terms			
Tracking, Estimating intent, Counter-Deception			
16.Security Classification of:		17.Limitation of Abstract	18.Number of pages
a.Report	b.Abstract	c.This Page		
U	U	U	SAR	42 (exc. this page)
19a.Name of Responsible Person		19b.Phone Number (include area code)	
David Lewis		314 235 6011	