Spatio-Temporal Coverage Optimization of Sensor Networks

Thèse

Vahab Akbarzadeh

Doctorat en génie électrique
Philosophiæ doctor (Ph.D.)

Québec, Canada

Résumé

Les réseaux de capteurs sont formés d'un ensemble de dispositifs capables de prendre individuellement des mesures d'un environnement particulier et d'échanger de l'information afin d'obtenir une représentation de haut niveau sur les activités en cours dans la zone d'intérêt. Une telle détection distribuée, avec de nombreux appareils situés à proximité des phénomènes d'intérêt, est pertinente dans des domaines tels que la surveillance, l'agriculture, l'observation environnementale, la surveillance industrielle, etc. Nous proposons dans cette thèse plusieurs approches pour effectuer l'optimisation des opérations spatio-temporelles de ces dispositifs, en déterminant où les placer dans l'environnement et comment les contrôler au fil du temps afin de détecter les cibles mobiles d'intérêt.

La première nouveauté consiste en un modèle de détection réaliste représentant la couverture d'un réseau de capteurs dans son environnement. Nous proposons pour cela un modèle 3D probabiliste de la capacité de détection d'un capteur sur ses abords. Ce modèle inègre également de l'information sur l'environnement grâce à l'évaluation de la visibilité selon le champ de vision. À partir de ce modèle de détection, l'optimisation spatiale est effectuée par la recherche du meilleur emplacement et l'orientation de chaque capteur du réseau. Pour ce faire, nous proposons un nouvel algorithme basé sur la descente du gradient qui a été favorablement comparée avec d'autres méthodes génériques d'optimisation «boites noires»sous l'aspect de la couverture du terrain, tout en étant plus efficace en terme de calculs.

Une fois que les capteurs placés dans l'environnement, l'optimisation temporelle consiste à bien couvrir un groupe de cibles mobiles dans l'environnement. D'abord, on effectue la prédiction de la position future des cibles mobiles détectées par les capteurs. La prédiction se fait soit à l'aide de l'historique des autres cibles qui ont traversé le même environnement (prédiction à long terme), ou seulement en utilisant les déplacements précédents de la même cible (prédiction à court terme). Nous proposons de nouveaux algorithmes dans chaque catégorie qui performent mieux ou produits des résultats comparables par rapport aux méthodes existantes. Une fois que les futurs emplacements de cibles sont prédits, les paramètres des capteurs sont optimisés afin que les cibles soient correctement couvertes pendant un certain temps, selon les prédictions. À cet effet, nous proposons une méthode heuristique pour faire un contrôle de capteurs, qui se base sur les prévisions probabilistes de trajectoire des cibles et égale-

ment sur la couverture probabiliste des capteurs des cibles. Et pour terminer, les méthodes d'optimisation spatiales et temporelles proposées ont été intégrées et appliquées avec succès, ce qui démontre une approche complète et efficace pour l'optimisation spatio-temporelle des réseaux de capteurs.

Abstract

Sensor networks consist in a set of devices able to individually capture information on a given environment and to exchange information in order to obtain a higher level representation on the activities going on in the area of interest. Such a distributed sensing with many devices close to the phenomena of interest is of great interest in domains such as surveillance, agriculture, environmental monitoring, industrial monitoring, etc. We are proposing in this thesis several approaches to achieve spatiotemporal optimization of the operations of these devices, by determining where to place them in the environment and how to control them over time in order to sense the moving targets of interest.

The first novelty consists in a realistic sensing model representing the coverage of a sensor network in its environment. We are proposing for that a probabilistic 3D model of sensing capacity of a sensor over its surrounding area. This model also includes information on the environment through the evaluation of line-of-sight visibility. From this sensing model, spatial optimization is conducted by searching for the best location and direction of each sensor making a network. For that purpose, we are proposing a new algorithm based on gradient descent, which has been favourably compared to other generic black box optimization methods in term of performance, while being more effective when considering processing requirements.

Once the sensors are placed in the environment, the temporal optimization consists in covering well a group of moving targets in the environment. That starts by predicting the future location of the mobile targets detected by the sensors. The prediction is done either by using the history of other targets who traversed the same environment (long term prediction), or only by using the previous displacements of the same target (short term prediction). We are proposing new algorithms under each category which outperformed or produced comparable results when compared to existing methods. Once future locations of targets are predicted, the parameters of the sensors are optimized so that targets are properly covered in some future time according to the predictions. For that purpose, we are proposing a heuristics for making such sensor control, which deals with both the probabilistic targets trajectory predictions and probabilistic coverage of sensors over the targets. In the final stage, both spatial and temporal optimization method have been successfully integrated and applied, demonstrating a complete and effective pipeline for spatiotemporal optimization of sensor networks.

Contents

Résumé iii

Abstract v

Contents vii

List of Tables ix

List of Figures xi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Research Objectives . . . 5
- 1.3 Outline . . . 7

2 Sensor Model 9
- 2.1 Sensing Model . . . 9
- 2.2 Coverage Formulation . . . 13
- 2.3 Visibility Function . . . 15
- 2.4 Probabilistic Membership Functions . . . 16
- 2.5 Zoom Factor . . . 18
- 2.6 Conclusion . . . 20

3 Spatial Optimization 21
- 3.1 Placement Problem . . . 21
- 3.2 Spatial Optimization Problem . . . 25
- 3.3 Implemented Optimization Methods . . . 25
- 3.4 Gradient Descent Method . . . 27
- 3.5 Experiments . . . 34
- 3.6 Results . . . 38
- 3.7 Conclusion . . . 40

4 Trajectory Prediction 43
- 4.1 Trajectory Prediction . . . 44
- 4.2 Trajectory Prediction Problem . . . 47
- 4.3 Short Term Trajectory Prediction . . . 48
- 4.4 Long Term Trajectory Prediction . . . 53
- 4.5 Experiments . . . 62

4.6 Results . . . 65
4.7 Conclusion . . . 69

5 Sensor Control **71**
5.1 Sensor Control . . . 71
5.2 Sensor Control Problem . . . 73
5.3 Greedy Sensor-wise Coverage Optimization (GSCO) . . . 75
5.4 Experiments and Results . . . 77
5.5 Conclusion . . . 81

6 Experiments with the Integrated System **83**
6.1 Dataset . . . 83
6.2 Spatial Optimization . . . 85
6.3 Trajectory Prediction . . . 85
6.4 Combination of Short and Long Term Prediction Methods . . . 91
6.5 Conclusion . . . 92

7 Conclusion **95**
7.1 Research Contributions . . . 96
7.2 Future work . . . 99

A Calculation of Derivatives **101**

B List of Publications **107**
B.1 Journal . . . 107
B.2 Conference . . . 107

Bibliography **109**

List of Tables

3.1 The parameter values for a realistic model of a sensor. 35
3.2 Information concerning the test maps used for the experiments. 36
3.3 The parameter values used for spatial optimization algorithms. 37
3.4 Maximum number of iterations for each method on each map. 37
3.5 Coverage percentage of spatial optimization algorithms with various numbers of sensors. 39

4.1 The parameter values for the trajectory generation in simulated dataset. 63
4.2 The parameter values for the short term trajectory prediction methods. 65
4.3 Average error and standard deviation of error for different short term prediction methods. 66
4.4 Average error and standard deviation of error for different long term prediction methods. 69

5.1 The parameter values for a realistic model of a PTZ camera. 78

6.1 Coverage percentage of the targets for the short term prediction methods. . . . 87
6.2 Coverage percentage of the targets for the long term prediction methods. 90
6.3 Coverage percentage of the targets for combination of short and long term prediction methods. 93

List of Figures

2.1 The free parameters $(x_i, y_i, \theta_i, \xi_i)$ for sensor $\mathbf{s}_i$ inside the environment. 13
2.2 A simple example on the effect of visibility function. 14
2.3 3D example of visibility function behaviour. 16
2.4 The effects of variations of β_d on $\mu_d(\|\mathbf{p}_i - \mathbf{q}\|)$. Assuming that $\alpha_d = 5$. 17
2.5 The effects of variations of β_p on $\mu_p(\theta_i - \angle_p(\mathbf{q} - \mathbf{p}_i))$. Assuming that $\alpha_p = 60$. 17
2.6 Probabilistic coverage model of a sensor on the elevation map. 18
2.7 The relationship between parameters $\alpha_{pi}^{(t)}$, L_h and $f_i^{(t)}$. 19

3.1 Illustration of the temperature function used for the simulated annealing method. 27
3.2 Pseudo-code of sensor placement with simulated annealing, with perturbation of one sensor position at a time. 28
3.3 The proposed gradient descent method for sensor placement optimization. . . . 31
3.4 Simple experiment to show the effect of different parts of the loss function. . . . 33
3.5 Optimizing with different loss functions. 33
3.6 An experiment to show the effect of overlap on coverage over the gradient. . . . 34
3.7 An experiment to show the effect of a coverage gap over the gradient. 34
3.8 Probabilistic coverage model of a sensor. 35
3.9 Part of the Université Laval (UL) map, chosen for the weighted experiments. . 36
3.10 Result of placement on the Université Laval campus map. 40
3.11 Result of placement on the North Carolina rural area map. 41
3.12 Comparison between speed of convergence for different methods on a sample run for map NC-B. 42

4.1 Short term prediction methods. 52
4.2 Problem of the probabilistic short term methods for long prediction steps. . . . 53
4.3 The behaviour of similarity function in one dimension. 56
4.4 The behaviour of similarity function with unknown intermediate observations and fixed bandwidth parameters. 56
4.5 The behaviour of similarity function with unknown intermediate observations and variable bandwidth parameters. 57
4.6 Long term target trajectory prediction process. 61
4.7 Illustration of one target moving in the environment in the simulated dataset. . 63
4.8 MIT Trajectory dataset [80]. 64
4.9 Train Station dataset [95]. 65
4.10 Affect of the sliding window size on the performance of different prediction methods over the MIT trajectory dataset. 67

4.11 Affect of the sliding window size on the performance of different prediction methods over the Train station dataset. 68

5.1 The proposed GSCO algorithm for sensor control. 76
5.2 Accessible coverage map ($\psi_i^{(t)}$) of a sensor. 76
5.3 The location of sensors is represented by the yellow squares on the map. 78
5.4 Coverage percentage for experiments with 50 targets using CMA-ES and GSCO optimization methods. 79
5.5 Coverage percentage for experiments with 100 targets using CMA-ES and GSCO optimization methods. 80
5.6 Coverage percentage for experiments with 150 targets using CMA-ES and GSCO optimization methods. 80

6.1 The elevation map of the Vieux-Québec area used for experiments. 84
6.2 The gates and pedestrian path information for the Vieux-Québec map. 84
6.3 The weight map used for the spatial optimization. 86
6.4 Sensor locations optimized by the CMA-ES method. 86
6.5 The combination of short and long term prediction methods for predicting the future location of a target. 91

Chapter 1

Introduction

1.1 Motivation

With the proliferation of Micro-Electro-Mechanical Systems (MEMS), recent years have seen great advances in deployment and application of Sensor Networks (SN) [73]. SNs consist of a network of sensor devices, where each device can autonomously make measurements in the environment and communicate with other sensors to achieve the goal of delivering valuable information to the end-users [8]. Besides, the sensor nodes in the SN usually have the capability to do preliminary processing, and only transfer small amount of valuable information.

Although each sensor alone can provide valuable information to the end-users, it is usually the combination of the information gathered from different sensors that provides capabilities to achieve several high level tasks. For example, although a single camera can give us useful information about the pedestrians who passed through the field of view of the camera, it is the information gathered from several cameras that can determine the overall trajectory of the pedestrian.

Sensor nodes have three main components: the sensing unit, the processing unit and the communication unit [29]. In the case of battery-powered sensor nodes, there is usually an energy unit which manages the batteries or other power sources (such as solar cells).

Sensors are devices which respond to a specific physical stimulus in their surrounding environment and convert the sensed measurements into digital signals, which can then be communicated to the end-users. The sensing unit consists of one or several sensors, type of which depends on the application of the network. Typical sensor types are thermal, biological, optical, chemical, and magnetic sensors.

The specifications of the processing unit depends on the type of sensors used. Sensors with less computational need (e.g., thermal sensors) are usually equipped with low power processing units (e.g., TmoteSky micro controller [69] with 0.375 nJ/instruction power consumption in

active mode). On the other hand, multimedia sensor units usually require larger amount of processing, and therefore the processing unit is much stronger. The processing model of the application also has an effect on the processing requirements of the sensors. In the applications where the burden of processing is distributed between the sensors (i.e. the bulk of processing is done locally on the sensors), the processing unit should be stronger compared to centralized scenarios, where the main responsibility of sensors is to send the raw data to the base station. The processing units are also equipped with sleep mode capabilities to reduce the power consumption of the unit even further when the sensor is not in operation.

The communication unit consists of different algorithms for the transport layer, network layer and data-link layer. Each of the mentioned layers should fulfill the responsibilities usually assigned to those layers. Besides, they should consider the specific requirements of the sensor networks.

1.1.1 Applications

By their nature, SNs are able to make a redundant sensing close to the phenomena of interest. The large number of sensors deployed in the target region makes SNs a suitable candidate for many applications which need a close monitoring of a physical phenomena. Depending on the environment the sensors are placed in, the networks can be categorized into different groups such as: terrestrial, underground, and underwater.

Terrestrial SNs are the most common networks, consisting of hundreds or even thousands small, battery powered sensor devices. The location of the sensors is either pre-determined, or the sensors are dropped from the plane in harsh environments. Underground SNs [51] are usually buried under the ground to monitor the condition of mines. Underwater SNs [38] have two main challenges. First, they should be designed considering the harsh environmental conditions, which causes the failure rate to be rather high in these sensors. Second, the wireless connection between the sensors is unreliable with limited bandwidth, long propagation delay, and signal fading issues.

From the mobility perspective, the sensors can be either static or mobile. In the static sensors, as their name suggests, the sensors are static in the position and condition they were initially placed in, while in the mobile networks, the sensors have the capability to change their initial conditions, to better adapt to the changing environment. Some networks could be consisted of both static and mobile sensors.

Application of sensor networks can be categorized from different perspectives, but most applications can be categorized under two groups [91]:

- **Monitoring networks:** These systems are usually designed to monitor structures (e.g., buildings or bridges for structural problems), farms (e.g., humidity or fertilizer levels in

the soil), human health (e.g., fire-fighters vital sign monitoring), harsh environments (e.g., eruptions of volcanos), or child education (e.g., sensor-enhanced toys to monitor child's learning process).

- **Tracking networks:** As the name suggests, in this group the goal is to track animals (e.g., migration of different species), humans (e.g., surveillance networks) or objects (e.g., vehicles for traffic purposes).

As an example, for the case of using SNs to monitor structures, sensors are usually put in place to [54]: 1. characterize structural vibration characteristics, 2. validate design models, or 3. better understand nonlinear structural responses resulting from seismic loadings. The result of measurements made by sensor help to automate the task of assessing structural health.

Another monitoring example is the systems which monitor the humidity and fertilizer levels in the soil. These systems help the farmers to find the optimal time to water or fertilize the farm, which can have great economical as well as environmental effects. Abd El-kader et al. [31] have shown that the cost of the deployed SN can be recovered only after one year from the savings made in the resources used (water and fertilizer), besides the increase in the yield size and quality.

FireLine [11] is a heart rate monitoring system, which used to monitor a fire fighter's heart rate in real-time to detect any abnormality and stress. The system can send alarm signals to the base station if the heart rate of the person goes above a mentioned threshold. The sensors are embedded inside a shirt that the fire fighter wears under his clothes.

Volcanic Monitoring [83] was another system deployed in Volcán Reventador in northern Ecuador to capture seismic and infrasonic (low-frequency acoustic) signals. As a result of close monitoring, these systems facilitates scientific studies of wave propagation phenomena and volcanic source mechanisms.

Under the category of tracking systems, ZebraNet [92] is a system composed of locating sensors attached to zebra collars. The goal of the system is to track animal migration patterns. Several collars were attached to animals at the Sweetwaters game reserve in central Kenya. Although monitored animals showed extra movements in the first week of monitoring, after this time period there was no significant difference between the movements of collared and non-collared animals.

Traffic surveillance [24] is an example of intelligent transportation systems designed to detect and track vehicles using acoustic and magnetic sensors. Currently the detection system in place uses the inductive loop systems. Authors have shown that the performance of the magnetic sensors produced superior results compared to the acoustic sensors and comparable to that of well maintained inductive loop system, in terms of vehicle detection rate. Besides the

magnetic sensor systems have higher configuration flexibility, which made the system scalable and deployable everywhere in the traffic networks.

1.1.2 Performance Measurements

Generally speaking, performance refers to the capability of a SN to perform the operation it was designed for in the first place. Depending on the type of sensors and the application, different performance criteria could be defined for a sensor network. For example, in some applications, sensors are battery powered and therefore energy consumption becomes a critical issue. The reason is that if the energy of a sensor is depleted, the sensor becomes unusable. In these types of networks, the routing algorithms, the sleeping mechanisms, and the location of base station have vital role in the overall life time of the network. The reason is that sensor consumes assigned amount of energy every time the sensor is sensing or sending the information to other nodes, and therefore communication of sensors with each other and with the base station should be optimized.

Connectivity is another performance criteria which plays an important role in wireless sensor networks. In these networks, there need to be a guarantee that all the nodes in the network can communicate with the base station. In other words, the network should be connected so that all the measurements made by the sensors would reach the base station, otherwise the unreachable sensors are useless.

There are other performance criteria such as response time, temporal accuracy, security, etc., but the most important performance criteria which is applicable to all types of sensor networks is the *coverage*. The initial assumption is that the SN is being used to sense a phenomena, therefore coverage is a fundamental performance criteria for measuring the quality of network in sensing the environment [56]. Informally, coverage means that each location in a region of interest should be within the sensing range of at least one of the sensors. In other words, each sensor covers part of the environment, and the coverage of the whole network is the aggregation of the coverage of the individual sensors. Depending on the SN application, coverage can refer to monitoring an entire area, observing a set of targets, or looking for breach among a barrier.

Maximal coverage is usually attained through optimizing placement. Placement (or spatial optimization) is an example of a more general problem of configuring sensor parameters. Depending on the application of SN and the sensor type being used, each sensor has a number of parameters that must be determined, (e.g., latitude, longitude and orientation of each sensor). Placement is the process of determining these parameters to obtain different goals (such as coverage, connectivity, etc.). Spatial optimization with the goal of coverage maximization is one of the most important criteria in SN deployment and will be the focus of the first part of this thesis.

The placement problem focuses to optimize SN deployment from the spatial perspective. The

other important aspect which should be considered is the temporal perspective. The difference comes from the nature of the phenomena the sensors are used to cover. In other words, sensors are sometimes placed in the environment to cover a phenomena which is static (e.g., door of a building). In these problems the parameters of the sensors are optimized once and then they will stay unchanged through time.

The other possibility is that the phenomena has a temporal aspect and changes over time (e.g., tracking people walking in the field). To optimize the coverage of the network form the temporal perspective, the parameters of the sensors should be controllable over time. These networks are sometimes referred to as sensor and actuator networks [7]. The goal of temporal coverage optimization (which will be the focus in the second part of the thesis) is to derive a mechanism which effectively and efficiently uses the sensing capabilities of sensors during a time period, for the goal of maximizing the coverage of the network over a phenomena which is changing over the time (targets moving in the field in this thesis).

Approaches based on human control of the sensor parameters are impractical, as controlling a large number of sensors is economically inefficient. Besides, humans attention level is variable over time, and tasks as such are repetitive which results in further lack of attention in human operators. Therefore, there is a great amount of interest to perform the mentioned tasks using an automatic system. The system should be able to perform all the tasks necessary to cover the moving targets automatically. These tasks include the detection and identification of targets, determining the current location and predicting the future location of targets, and finally modifying the parameters of the sensors to maximize coverage over the targets. In this thesis we focus on predicting the future positions of targets and optimizing the parameters of sensors to cover the moving targets.

1.2 Research Objectives

The research question is: How to optimize location and working parameters of sensors of a sensor network in order to maximize coverage over a number of targets moving in the environment.

Before we go into the details of the objectives, we should make clear some of the assumptions we are going to make about different parts of the system:

1. **Sensors:** are stationary devices capable of detecting the presence of targets in the environment. They are stationary meaning that once they are placed in the environment, they cannot move and change their location. On the other hand, the sensors have some controllable operating parameters. In other words, although their location is fixed through time, they have some controllable sensing parameters (e.g., sensing direction) which is adjustable through time.

A sensor can cover a target if the target is within the sensing area of the sensor. Besides, if the sensor covers the target, the sensor can estimate the identity and location of target with complete accuracy.

Although connectivity and energy consumption are common research problems when dealing with SNs, they are outside the scope of this thesis. In other words, we assume that sensors have uninterruptible connectivity and unlimited energy supply.

2. **Environment:** is an area with predefined boundaries, which is fixed during the operation of the system. All the interaction between sensors and targets happen inside the environment. The environment can contain natural and man-made objects which might interfere with the coverage of sensors.

3. **Target:** is a mobile entity in the environment. Each target appears in the environment at a specific time, makes several movements, and finally disappears from the environment. The target is sensed (covered) by a sensor if its location is within the coverage area of a sensor. The system knows the identity of each target once it is detected by one sensor, and there will not be no confusion between two targets, even if one of them goes out of the coverage of network for some time. The goal of the system is to cover the targets while they are present in the environment.

Now that some of the assumptions are clarified, we can focus more on the objectives of this thesis. As mentioned before, the final goal of the system is to have maximal coverage over moving targets. For that purpose, the best location for each sensor should be first determined in the environment. The reason is that although the controllable parameters of the sensors can be changed later, during the operation of the network, the location of each sensor is the only parameter which should be defined before the networks is put into operation.

To find the best location of the sensors, we need to have a model for the coverage area of each sensor. The reason is that the best location for sensors should guarantee that the coverage of two sensors is not "wasted" by them covering the same part of the environment, while some other part of the environment is non-covered. Therefore the coverage of each sensor should be accurately modelled.

The placement of the sensors (what we call spatial optimization) is the only phase which should be done before the operation of sensors. Once sensors are in place, the network can start its operation and cover the targets. In this phase, the targets should be first detected in the environment. As soon as a target enters the coverage area of a sensor, it is assumed that the target is detected and its location is known. From there on, the task of the system is to keep the target under coverage until the target exists the environment.

To keep the targets under coverage, the system should predict the future location of each target. The reason is that at each time step, the system should know the future location of

targets to plan the change of sensor parameters for the future time step. Once the future location of targets predicted, the system can adjust the controllable parameters of sensors. The system then iterates between the trajectory prediction and sensing parameter adjustment steps, to keep the coverage over the targets as they displace in the environment. The specific tasks of this objective are as follows:

1. **Sensor coverage model** defines an adjustable sensor coverage model that allows modelling coverage area of different types of sensors, while taking into to account the effect of environmental obstacles on the coverage region of each sensor.
2. **Spatial optimization** aims at finding the best location and direction of sensors, so that the final network has maximal coverage over the whole environment. The algorithm should be able to provide proportional coverage over the environment. Meaning it should be able to provide more complete coverage over more important parts of the environment. Besides, it should be scalable to large environments.
3. **Target trajectory prediction** looks for predicting the future location of the targets as accurately as possible with adjustable prediction horizon, assuming a number of targets for which the location and identification is known.
4. **Sensor control algorithm** defines a mechanism to optimize the variable parameters of the sensors, to have maximal coverage over the prediction produced in the target trajectory prediction step.

1.3 Outline

This thesis is organized in seven chapters. In chapter 2, we propose a new model for the coverage area of sensors. The novelty of the model is in the combination of the probabilistic sensing model with terrain information. Therefore, the model gives more realistic estimates for the coverage region of sensors with line-of-sight coverage capabilities. In chapter 3, we propose an algorithm based on the gradient descent method to optimize the location and direction of sensors, so that the final network has maximal coverage over the environment. The main advantages of the algorithm are low computational demand and scalability to a large number of sensors.

Then we switch to temporal optimization in 4, where several methods are proposed to predict the trajectory of mobile targets. Depending on the prediction horizon, the proposed methods are divided into two groups of short and long term methods. Later, in chapter 5, the focus is on the sensor control problem. In this chapter, we propose the Maximum Coverage Strategy algorithm to change the direction of stationary sensors. The proposed algorithm produces competitive results compared to more sophisticated optimization algorithms, with less amount of computation.

Next, in chapter 6, all the previously mentioned algorithms are integrated into a complete system and applied in a totally new environment. In this experiments, the sensors are first placed in the environment, and then the direction of sensors is moved according to the predictions made over the movement of the targets. Finally, in chapter 7 we conclude the thesis by presenting the scientific contributions of the thesis, and some of the possible applications as the future work.

Chapter 2

Sensor Model

Sensor are designed to sense a phenomenon. This is the principal and most important responsibility of the sensors. In order to simulate and evaluate the behaviour of sensors in different conditions, the sensing capability of the sensors should be modelled. For the sensing model, the goal is to get a more realistic model for the coverage area of the sensor in the environment. In other words, the preference is over the models which rely on more realistic assumptions so that difference between the coverage of the network in the simulated setting and later in the real experiments would be as small as possible.

For the rest of the chapter, in section 2.1 we cover some of the models proposed for the sensing model in the literature, in section 2.2 we provide the formulation for the coverage function of our sensor model, which itself includes the visibility, distance and orientation membership functions. The visibility function is explained in 2.3, and the membership functions for distance and orientation are given in section 2.4. Finally, in section 2.5, we provide the coverage model for a specific type of sensor which we will use later in the rest of thesis.

2.1 Sensing Model

The sensing model has two components; the sensor model and the environment model. This division comes from the fact that the coverage of a sensor over a location in the environment does not only depend on the characteristics of the sensor alone; but also the environmental factors take an important role in the final coverage of the network. Sensor models can also be categorized according to different perspectives. Two are of particular interest for our work: sensing distance and coverage direction.

2.1.1 Sensing Distance

The main issue in the sensing distance is to establish the relationship between the detection ability of a sensor and the distance it has with the object of interest. The conventional

approach assumes a binary coverage for each sensor [37], meaning that the sensor can only cover the locations which are within a specific range from its location. This range is usually referred to as the sensing range of the sensor. More precisely, for the coverage $C(.,.)$ between sensor $\mathbf{S}$ and location $\mathbf{q}$ in the environment we define the following coverage function:

$$C(\mathbf{S},\mathbf{q}) = \begin{cases} 1 & \|\mathbf{S}-\mathbf{q}\| \leq r \\ 0 & \text{otherwise} \end{cases}, \tag{2.1}$$

where $\|\mathbf{S}-\mathbf{q}\|$ is the euclidean distance between $\mathbf{S}$ and $\mathbf{q}$, and r is the sensing range of the sensor.

A more reasonable assumption is that the sensing ability of the sensors diminish as the distance increase from the sensor. These methods are usually referred to as the probabilistic sensing models, as the underlying assumption is that the probability of detecting an event in a specific location diminishes as the distance between the event and the sensor increases. Many different methods have been proposed under the probabilistic sensing model approach.

Zhu et al. [98] proposed the following equations for the coverage model of the sensor:

$$C(\mathbf{S},\mathbf{q}) = \begin{cases} 1 & \|\mathbf{S}-\mathbf{q}\| \leq r - r_e \\ e^{-\lambda\alpha^{\beta}} & r - r_e \leq \|\mathbf{S}-\mathbf{q}\| \leq r + r_e \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

where r_e is the critical distance around the sensing range of the sensor, and λ, β, and α are the three parameters of the decaying function.

The model proposed by Liu et al. [53], assumes a ring coverage around the sensor location. The model is based on the assumption that the detectability of an event by a sensor is related to the energy emitted by the event and therefore suggested the following formula:

$$C(\mathbf{S},\mathbf{q}) = \begin{cases} \frac{\alpha}{\|\mathbf{S}-\mathbf{q}\|^{\beta}} & r_1 \leq \|\mathbf{S}-\mathbf{q}\| \leq r_2 \\ 0 & \text{otherwise} \end{cases}, \tag{2.3}$$

where r_1 and r_2 define the range of sensor detection capability, α is the energy of the event happening at location $\mathbf{q}$, and β is the signal decaying parameter.

Finally, the model proposed by Ahmed et al. [3] was based on a staircase model for the coverage of sensors in their surrounding. The model can be viewed as an extension of the binary model. Unlike the binary model in which the coverage goes from one to zero in one step, in this model the coverage diminishes at different distances, until it gradually reaches

zero. The following is a four step model of coverage:

$$C(\mathbf{S},\mathbf{q}) = \begin{cases} 1 & 0 \leq \|\mathbf{S}-\mathbf{q}\| \leq r_1 \\ \alpha_1 & r_1 \leq \|\mathbf{S}-\mathbf{q}\| \leq r_2 \\ \alpha_2 & r_2 \leq \|\mathbf{S}-\mathbf{q}\| \leq r_3 \\ \alpha_3 & r_3 \leq \|\mathbf{S}-\mathbf{q}\| \leq r_4 \\ 0 & \text{otherwise} \end{cases}, \tag{2.4}$$

where $0 \leq \alpha_3 \leq \alpha_2 \leq \alpha_1 \leq 1$ are the coverage degrees at different levels, and $0 \leq r_1 \leq r_2 \leq r_3 \leq r_4$ are different distances at which the coverage decrease.

2.1.2 Sensing Direction

A common assumption [41, 62] is that each sensor can sense a circular area around itself having a radius known as the sensing range. This is the simplest assumption for the coverage region of a sensor. Although, this assumption of omni-directional sensing ability is valid for some type of sensors, such as temperature, humidity and magnetic sensors, this assumption does not hold true for many types of sensors, such as cameras, and ultrasound sensors, which have a directional sensing area.

Jing et al. [4] proposed a sector shaped model for the coverage of sensors. This model resembles in the definition to the omni-directional model. The only difference is that sensor has a sensing direction D, and a sensing angle α. In other words, the sensor can only cover a sector of the circle defined by the sensing direction and the sensing angle. More precisely, an event is detected by the sensor if the following two conditions are satisfied, between sensor $\mathbf{S}$ and location $\mathbf{q}$:

$$\|\mathbf{S}-\mathbf{q}\| \leq r, \tag{2.5}$$

$$\cos(\frac{\alpha}{2}) \leq \cos\angle(\mathbf{S}-\mathbf{q}), \tag{2.6}$$

Another coverage model was proposed in [1], where Adriaens et al. modelled the coverage region of a sensor as a isosceles triangle. The parameters for this model included the coverage angle α and maximum sensing distance (the length of each side of the triangle).

A 3D directional coverage model was proposed by Ma et al. [55]. In the new model, the sensing direction of the sensor has two parameters. Notice that in the 3D space, the direction has three parameters, which are usually refereed to as pan, tilt, and yaw. While the pan and tilt parameters are important, the yaw degree do not change the sensing area of the sensor. Also the sensing angle has two parameters, to define the maximum sensing range along the pan and tilt angles. Therefore the final model has four parameters, the direction along the pan and tilt angles, and the maximum degree the sensor can cover along each of those angles.

Recently, Peng et al. [66] also proposed a 3D coverage model for directional sensors. Besides, they proposed an algorithm to detect the coverage holes in the environment, and another algorithm based on the virtual potential fields was proposed to remedy the coverage holes.

Notice all the directional models presented here, have binary coverage using the definition given in the previous section.

2.1.3 Environment Model

As mentioned before, the coverage of a sensor over the environment depends on the coverage model of the sensor and the model with which the environment is represented. Here we review some of the models used for the environment.

The model most commonly used for the environment is a flat 2D space [10, 37]. In this model sensors can be placed anywhere in the environment, and the coverage region of the sensors only depend on the sensor model. This is the most simplistic model for the environment and this simplification usually results in an overestimation for the performance of the network. In other words, in a realistic model the effect of the obstacles and topography of the terrain should also be taken into consideration.

Sometimes the 2D environment is divided into a regular grid, and the coverage is calculated over the grid cells. This is usually referred to as the grid based coverage [21]. The important factor in the grid based coverage is the size of each grid cell. The coverage calculated over the environment with small grid cells, can closely simulate the result over the actual 2D map, while larger grid cells are less accurate.

The problems of the simple model encouraged authors to consider more realistic models for the environment. For example, authors in [30, 52], used some square shaped obstacles to model the interference that the environment can make over the coverage region of each sensor. The problem is that square obstacles cannot accurately represent all types of obstacles present in the environment, especially they are not suitable to represent the topographical information.

Mittal et al. [59], studied the effect of moving obstacles on the coverage of a visual sensor network. In this study, the dynamical visibility constraints are modelled for simple circle obstacles, therefore the final result could represent the effect of humans moving within the scene of several sensors.

A configuration less studied in the literature is the coverage model in 3D environments. This is the most realistic model, but due to computational complexities of the placement problem, the approach has been considered only by few [60]. For example, in [68] authors proposed to model the volume under the surface of ocean as a 3D environment for sensor placement. Due to computational complexities, sensor models used for these environments are restricted to omni-directional with binary coverage models.

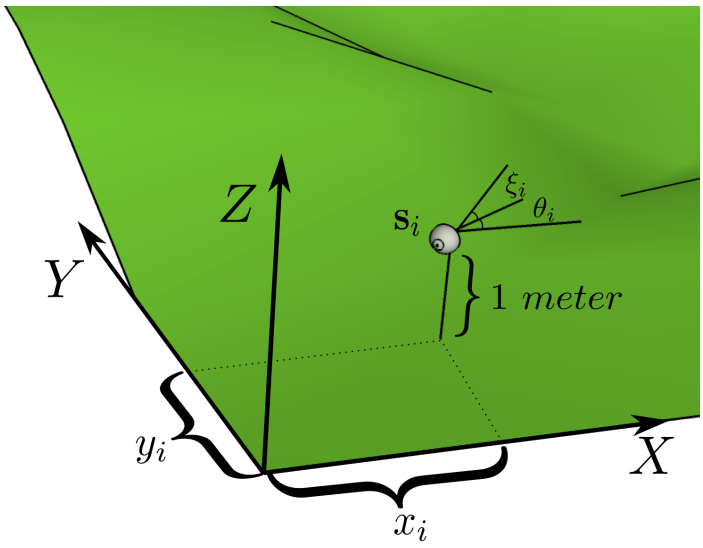


Figure 2.1: The free parameters $(x_i, y_i, \theta_i, \xi_i)$ for sensor $\mathbf{s}_i$ inside the environment.

2.1.4 Proposed Model

In this chapter, we are proposing a probabilistic coverage model for directional sensors in a 2.5D environment. The proposed model has several advantages over other models presented in the literature. First, the model is applicable to both directional and omni-directional sensors. Second, the coverage ability of the model decays gradually both with respect to distance and angle. Finally, the model assumes a topological terrain model for the environment.

2.2 Coverage Formulation

The sensing model mainly depends on distance, orientation, and visibility. We first assume that all sensors are positioned at a certain constant height τ above the ground level. The sensor position is thus described by a 3D point $\mathbf{p} = (x, y, z)$, where (x, y) are free parameters and $z = g(x, y) + \tau$ is constrained by the terrain elevation $g(.,.)$ at position (x, y), as defined by a Digital Elevation Model (DEM) provided by a Geographic Information System (GIS). We further assume that the anisotropic properties of sensors are fully defined by a pan angle θ around the vertical axis and a tilt angle ξ around horizontal axis. Given the DEM, a sensor network $N = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n\}$ of n sensors is thus fully specified by $4n$ free parameters $\mathbf{s}_i = (\mathbf{p}_i, \theta_i, \xi_i)$, $i = 1, 2, \ldots, n$, with $\mathbf{p}_i = (x_i, y_i)$ (see Figure 2.1). The typical resolution of DEMs in this thesis is one square meter.

Now the coverage $C(\mathbf{s}_i, \mathbf{q})$ of sensor $\mathbf{s}_i$ at point $\mathbf{q}$ in the environment can be defined as a function of distance $d(\mathbf{s}_i, \mathbf{q}) = \|\mathbf{p}_i - \mathbf{q}\|$, pan angle $p(\mathbf{s}_i, \mathbf{q}) = \angle_p(\mathbf{q} - \mathbf{p}_i) - \theta_i$, tilt angle

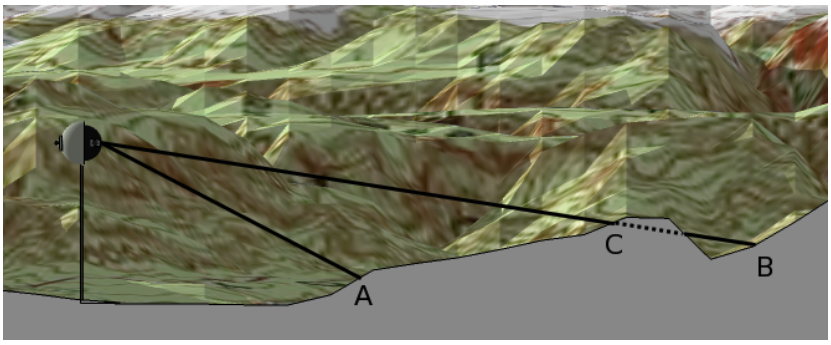


Figure 2.2: The effect of visibility function. Here sensor and point **A** are inter-visible, therefore $v(\mathbf{s}_i, \mathbf{A}) = 1$, but it is not the case for $\mathbf{s}_i$ and **B**, because the line of sight is obscured at point **C**.

$t(\mathbf{s}_i, \mathbf{q}) = \angle_t(\mathbf{q} - \mathbf{p}_i) - \xi_i$, and visibility $v(\mathbf{s}_i, \mathbf{q})$ from the sensor:

$$C(\mathbf{s}_i, \mathbf{q}) = f[\mu_d(\|\mathbf{p}_i - \mathbf{q}\|), \mu_p(\angle_p(\mathbf{q} - \mathbf{p}_i) - \theta_i), \mu_t(\angle_t(\mathbf{q} - \mathbf{p}_i) - \xi_i), v(\mathbf{p}_i, \mathbf{q})], \tag{2.7}$$

where $\angle_p(\mathbf{q} - \mathbf{p}_i) = \arctan\left(\frac{y_\mathbf{q} - y_{\mathbf{p}_i}}{x_\mathbf{q} - x_{\mathbf{p}_i}}\right)$ is the angle between the sensor $\mathbf{s}_i$ and the point $\mathbf{q}$ within the **XY** plane, and $\angle_t(\mathbf{q} - \mathbf{p}_i) = \arctan\left(\frac{z_\mathbf{q} - z_{\mathbf{p}_i}}{\|\mathbf{p}_i - \mathbf{q}\|}\right)$ is the angle between the sensor $\mathbf{s}_i$ and the point $\mathbf{q}$ within the **XZ** plane. In other words, for $\mathbf{q}$ to be covered by sensor $\mathbf{s}_i$, we need to take into account its sensing range, sensing angles, and visibility. Let $\mu_d, \mu_p, \mu_t \in [0, 1]$ represent some membership functions of the mentioned coverage conditions, then Equation 2.7 can be rewritten as multiplication of these memberships:

$$C(\mathbf{s}_i, \mathbf{q}) = \mu_d(\|\mathbf{q} - \mathbf{p}_i\|) \cdot \mu_p(\angle_p(\mathbf{q} - \mathbf{p}_i) - \theta_i) \cdot \mu_t(\angle_t(\mathbf{q} - \mathbf{p}_i) - \xi_i) \cdot v(\mathbf{p}_i, \mathbf{q}) \tag{2.8}$$

Function $v(\mathbf{p}_i, \mathbf{q})$ is binary. Given a sensor position $\mathbf{p}_i$, if the line of sight between sensor $\mathbf{s}_i$ and $\mathbf{q}$ is obscured, then we assume that the coverage cannot be met ($v = 0$), otherwise the visibility condition is fully respected ($v = 1$), (see Figure 2.2). In our experiment, we assume that all sensors are one metre above the ground ($\tau = 1$). Memberships μ_d, μ_p, and μ_t need to be defined according to their parameters. At each position $\mathbf{q} \in \Xi$ of environment Ξ, the coverage for a single sensor is thus the multiplication of the three above conditions. The coverage function is probabilistic as each of the membership functions provide the probability that an object of interest at point $\mathbf{q}$ is detected by the sensor $\mathbf{s}_i$. Therefore, $C(\mathbf{s}_i, \mathbf{q})$ represents the probability of coverage while $1 - C(\mathbf{s}_i, \mathbf{q})$ gives the probability of non-coverage. If more than one sensor covers $\mathbf{q}$, then a way to compute the local network coverage C_l is:

$$C_l(N, \mathbf{q}) = 1 - \prod_{i=1,\ldots,n} (1 - C(\mathbf{s}_i, \mathbf{q}))\,. \tag{2.9}$$

This formulation is based on the assumption that the coverage of several sensors with respect to one position in the environment is independent from each other. This assumption roots

in the probabilistic coverage definition of each sensor. More precisely, assume that a location $\mathbf{q}$ in the environment is covered by the sensor $\mathbf{s}_1$ with the probability $C(\mathbf{s}_1, \mathbf{q})$. This model follows a Bernoulli probability distribution, for which we define a random variable $X_{\mathbf{s}_1\mathbf{q}}$. The random variable takes the value of one ($P(X_{\mathbf{s}_1\mathbf{q}} = 1) = C(\mathbf{s}_1, \mathbf{q})$) if the location is covered by sensor, and value of zero otherwise ($P(X_{\mathbf{s}_1\mathbf{q}} = 0) = 1 - C(\mathbf{s}_1, \mathbf{q})$). Now assume that the location is covered also by another sensor $\mathbf{s}_2$ with probability $C(\mathbf{s}_2, \mathbf{q})$. Therefore, the location $\mathbf{q}$ is covered by the network if at least one of the sensors cover the location, or we could say that the location is not covered if none of the sensors cover the location. More precisely, the probability of the coverage is:

$$C_l(\{\mathbf{s}_1, \mathbf{s}_2\}, \mathbf{q}) = 1 - [P(X_{\mathbf{s}_1\mathbf{q}} = 0, X_{\mathbf{s}_2\mathbf{q}} = 0)], \tag{2.10}$$

but we assumed that the coverage of the two sensors over the location is independent from each other so we have:

$$C_l(\{\mathbf{s}_1, \mathbf{s}_2\}, \mathbf{q}) = 1 - [P(X_{\mathbf{s}_1\mathbf{q}} = 0)\, P(X_{\mathbf{s}_2\mathbf{q}} = 0)], \tag{2.11}$$

$$= 1 - [(1 - C(\mathbf{s}_1, \mathbf{q}))\, (1 - C(\mathbf{s}_2, \mathbf{q}))], \tag{2.12}$$

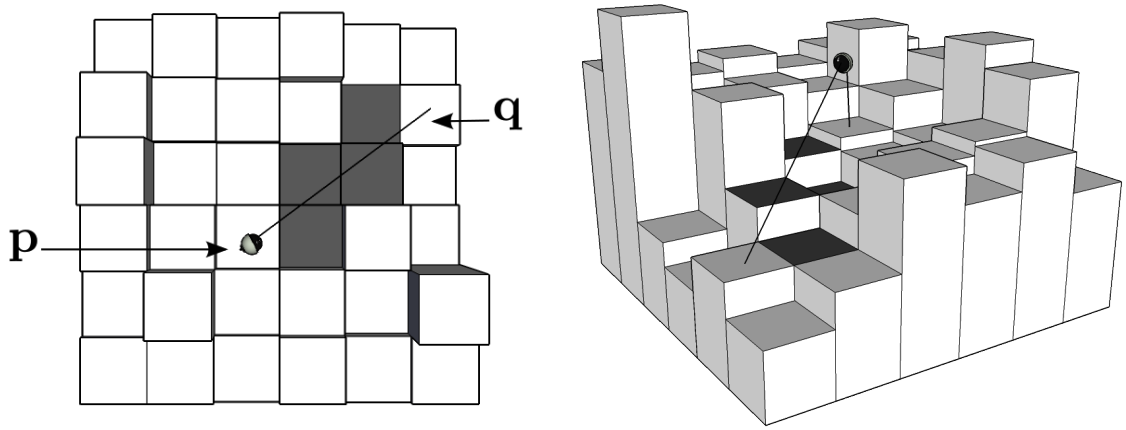
which can be generalized for n sensors to the formula proposed in Eq. 2.9.

2.3 Visibility Function

Visibility function calculates the visible area for each sensor. The main factor which affects the visibility between two points is the elevation of all points in the straight line connecting those two points. This information is provided by a DEM, which is basically a two dimensional matrix, where each cell stores the elevation of the corresponding location in the real environment (see Fig.2.3).

In order to calculate visibility between two points $\mathbf{p}$ and $\mathbf{q}$, the list of all cells in the matrix which intersects with the line of sight between those two points should be first calculated. Then each point in the list is checked versus the line connecting points $\mathbf{p}$ and $\mathbf{q}$. If the elevation of an intermediate point is more than the elevation of the line at that point, then points $\mathbf{p}$ and $\mathbf{q}$ are not inter-visible, otherwise they are inter-visible.

Assuming that point $\mathbf{p}$ represents the location of a sensor in the environment, first the elevation of point $\mathbf{p}$ is increased by the elevation of the sensor and then the mentioned visibility calculation process is repeated between point $\mathbf{p}$ and all other points in its vicinity.



(a) Elevation map (b) Coverage map

Figure 2.3: Visibility function behaviour. Assuming that the visibility between two points $\mathbf{p}$ and $\mathbf{q}$ is of question, the grey cells represent the set of cells whose elevation should be compared with the straight line between points $\mathbf{p}$ and $\mathbf{q}$.

2.4 Probabilistic Membership Functions

The membership functions μ_d, μ_p, and μ_t can be defined as crisp function, with value of 1 when the position is within a fixed sensing range or angle of view, and otherwise 0.

$$\mu_d(\|\mathbf{p}_i - \mathbf{q}\|) = \begin{cases} 1 & \|\mathbf{p}_i - \mathbf{q}\| \leq d_{max} \\ 0 & \text{otherwise} \end{cases}, \tag{2.13}$$

$$\mu_p(\angle_p(\mathbf{q} - \mathbf{p}_i) - \theta_i) = \begin{cases} 1 & (\angle_p(\mathbf{q} - \mathbf{p}_i) - \theta_i) \in [-a, a] \\ 0 & \text{otherwise} \end{cases}, \tag{2.14}$$

$$\mu_t(\angle_t(\mathbf{q} - \mathbf{p}_i) - \xi_i) = \begin{cases} 1 & (\angle_t(\mathbf{q} - \mathbf{p}_i) - \xi_i) \in [-b, b] \\ 0 & \text{otherwise} \end{cases}. \tag{2.15}$$

However, such functions used in a coverage function provide essentially a binary 0/1 signal, which is not a realistic performance model of real sensors. Moreover, for a method such as gradient descent, differentiable coverage functions are needed. Therefore, we propose to use probabilistic membership functions that provide a monotonically decreasing membership value over distance and relative angle of position to sensor. Hence, these functions have two benefits: first, they better comply with the performance of the real sensors, and second, they are differentiable.

We propose to use sigmoid function for distance membership function:

$$\mu_d(\|\mathbf{p}_i - \mathbf{q}\|) = 1 - \frac{1}{1 + \exp\left(-\beta_d(\|\mathbf{p}_i - \mathbf{q}\| - \alpha_d)\right)}, \tag{2.16}$$

with β_d and α_d as the parameters configuring the membership function (see Figure 2.4). Parameter β_d can be approximated using experimental observations on sensor behaviours.

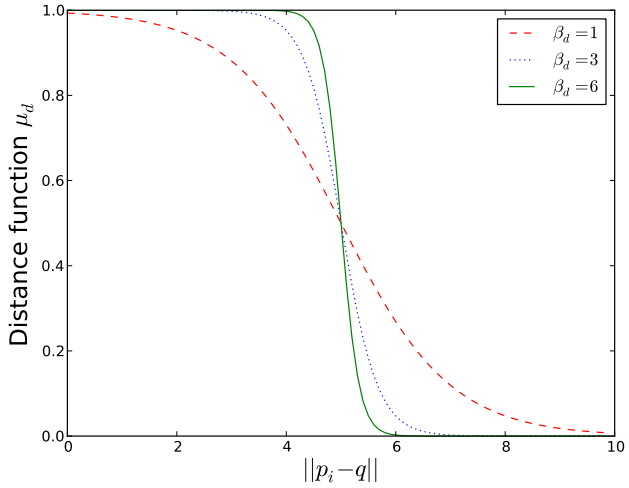


Figure 2.4: The effects of variations of β_d on $\mu_d(\|\mathbf{p}_i - \mathbf{q}\|)$. Assuming that $\alpha_d = 5$.

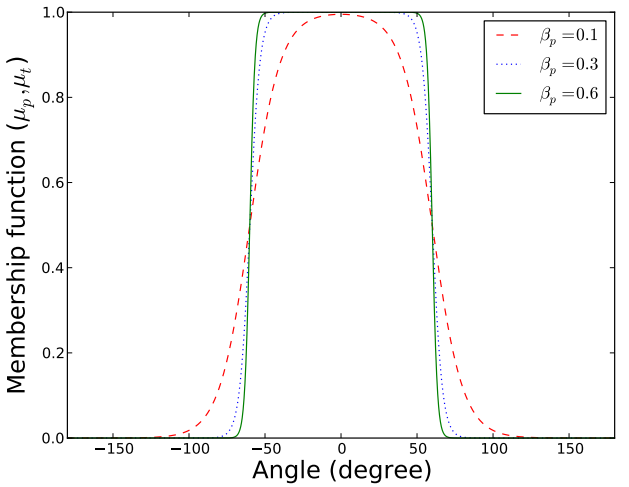
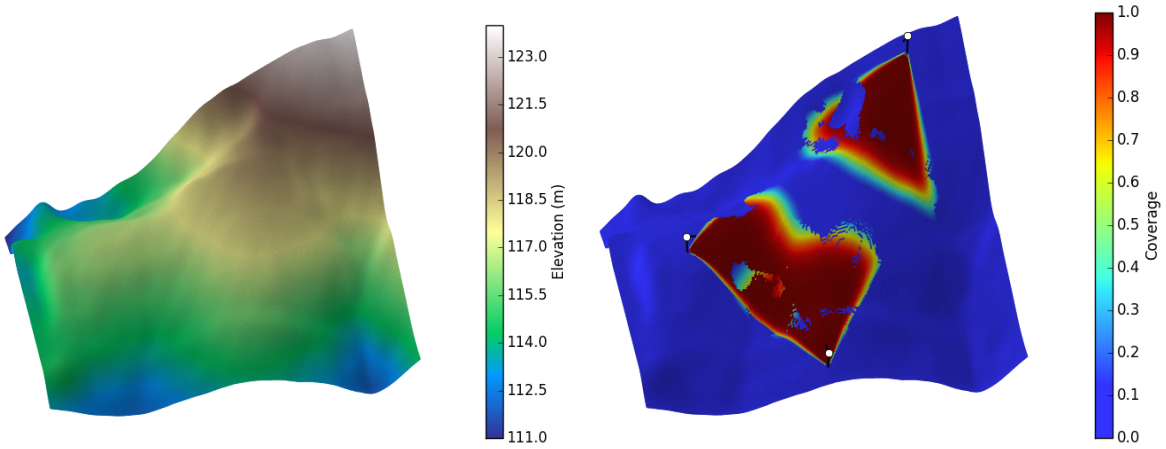


Figure 2.5: The effects of variations of β_p on $\mu_p(\theta_i - \angle_p(\mathbf{q} - \mathbf{p}_i))$. Assuming that $\alpha_p = 60$.

As shown in the figure, parameter β_d controls the slope of the function and α_{di} determines the distance where the sensor has 50% of its maximum coverage. We suggest the sigmoid function because of its resemblance to the Cumulative Distribution Function of the normal distribution. In other words, if we assume that the sensing range of a sensor is defined by a normal distribution (rather than a dirac delta function in case of binary coverage) then the sigmoid function would represent the coverage region of the sensor (instead of the unit step function in case of binary coverage).



(a) Elevation map (b) Coverage map

Figure 2.6: Probabilistic coverage model of a sensor. (**a**) The underlying elevation map used for the experiment; (**b**) Assuming there are three sensors on the map (each shown with a white circle above the ground), the colour shows the different degree of coverage for the whole network. The effect of the visibility function can be seen by the non-visible areas within the coverage region of each sensor. The values used for the sensor models are: β_d=0.2, $\alpha_d = 100$, $\beta_p = 0.5$, $\alpha_p = 60$, $\beta_t = 0.5$, and $\alpha_t = 30$.

As for the pan angle membership functions, we propose another sigmoid function:

$$\mu_p(\underbrace{\angle_p(\mathbf{q}-\mathbf{p}_i)-\theta_i}_{\gamma_i}) = \frac{1}{1+\exp\left(-\beta_p(\gamma_i+\alpha_p)\right)} - \frac{1}{1+\exp\left(-\beta_p(\gamma_i-\alpha_p)\right)}, \tag{2.17}$$

where α_p controls the "width" of the function and β_p controls the slope of the function at the boundaries (see Figure 2.5). Note that the proposed function has the range [-180,180] degrees. Therefore, any calculated angle should be brought into this range accordingly. In the same way, membership function μ_t is defined as:

$$\mu_t(\underbrace{\angle_t(\mathbf{q}-\mathbf{p}_i)-\xi_i}_{\zeta_i}) = \frac{1}{1+\exp\left(-\beta_t(\zeta_i+\alpha_t)\right)} - \frac{1}{1+\exp\left(-\beta_t(\zeta_i-\alpha_t)\right)}, \tag{2.18}$$

with a range in [-90,90].

2.5 Zoom Factor

Cameras could be viewed as a special type of sensors. The sensor model explained in the previous section closely resembles the coverage region of the conventional cameras over the environment. Later in chapter 4, we will focus on a special type of sensors, called the Pan-Tilt-Zoom (PTZ) cameras. These cameras can change their coverage region in three ways by panning, tilting and zooming. The pan and tilt capabilities are well explained by the

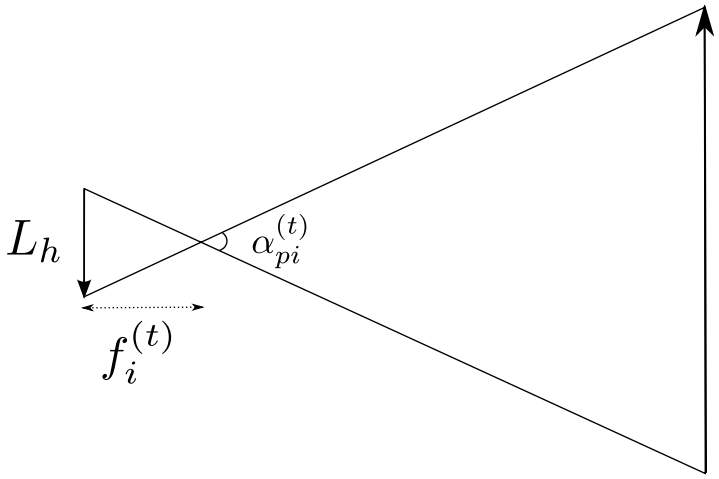


Figure 2.7: The relationship between parameters $\alpha_{pi}^{(t)}$, L_h and $f_i^{(t)}$.

general sensor model we defined in the previous section. Focal length (zoom) is the additional parameter of the PTZ cameras which we define using the $f_i^{(t)}$ parameter. This parameter represents the focal length of the camera (sensor) $\mathbf{s}_i$ at time t.

Notice that the formulation given in the previous section was designed for the spatial optimization problem. In the temporal optimization problem, we need that the properties of the sensors be variable through time, and our formulation should reflect this fact. As a result, in the temporal optimization problem the α_d would change to $\alpha_{di}^{(t)}$, which defines the α_d parameter of sensor $\mathbf{s}_i$ at time t. In the same way we would define the parameters $\alpha_{pi}^{(t)}$, $\alpha_{ti}^{(t)}$, $\theta_i^{(t)}$, and $\xi_i^{(t)}$. Therefore, the state of each camera is defined by $(\mathbf{p}_i, \theta_i^{(t)}, \xi_i^{(t)}, f_i^{(t)})$. Next, we define how the newly defined focal length parameters affects the other parameters for the sensor coverage model.

The pan and tilt membership functions are related to the focal length value through the $\alpha_{pi}^{(t)}$ and $\alpha_{ti}^{(t)}$ parameters. More precisely, these parameters define the angle of view for the camera, so we have:

$$\alpha_{pi}^{(t)} = 2\arctan(L_h, 2f_i^{(t)}), \tag{2.19}$$

$$\alpha_{ti}^{(t)} = 2\arctan(L_v, 2f_i^{(t)}), \tag{2.20}$$

where L_h and L_v are the horizontal and vertical sizes of the camera's sensor, respectively. The pinhole model of the camera and the relationship between the parameters in Eq. 2.19, is shown in Fig. 2.7. The same figure could be drawn for the parameters in Eq. 2.20.

The distance membership function is also related to the focal length value $f_i^{(t)}$ through the following formula:

$$\alpha_{di}^{(t)} = \left[\frac{\alpha_{d_{max}} - \alpha_{d_{min}}}{f_{max} - f_{min}}(f_i^{(t)} - f_{min})\right] + \alpha_{d_{min}}, \tag{2.21}$$

where $\alpha_{d_{max}}$ and $\alpha_{d_{min}}$ are the maximum and minimum value that $\alpha_{di}^{(t)}$ can take. In practice, the value of these parameters depends on the camera type, size of the targets, the minimum number of pixels inside the images needed for accurate detection of target, and so forth.

2.6 Conclusion

This chapter proposes a novel coverage model for sensors in the environment. The novelty of this model lies in the integration of terrain information (elevation maps) with a probabilistic sensor model. The visibility function was added to the sensor model to take in to account the interference of the obstacles on the coverage region of each sensor. Besides, the proposed probabilistic sensing model is applicable to both directional and omni-directional sensors, and therefore can model both sensor types. The decay in coverage of sensors is both valid with respect to distance and angle, modelled with independent parameters. Thus, the final model can adapt to different types of sensors. The proposed functions are also differentiable. As we will show later, this is of great importance in derivation based optimization algorithms. Finally, we presented a model for a specific type of sensors, that is the Pan-Tilt-Zoom (PTZ) cameras.

Chapter 3

Spatial Optimization

In order to evaluate the performance of the sensor network on the assigned task, a measure of performance should be defined. The most commonly used performance criteria is the coverage. In the previous chapter, we modelled the coverage of a sensor over the environment. Based on the sensing model, different coverage goals could be defined. A natural step from there is to place the sensors in the environment considering the coverage goal, as the placement of the sensor can greatly affect the overall coverage of the network. That is what we call spatial optimization, the goal being to find the best placement for the sensors (e.g. location and direction) so that the final network has maximal coverage over the environment.

The chapter is organized as follows. In section 3.1, different coverage goals and optimization methods proposed for the spatial optimization problem are presented. Then in section 3.2, the spatial optimization problem is defined based on the sensor model proposed in chapter 2. In section 3.3, two classical optimization methods are proposed for the placement problem. Next, in section 3.4, a new algorithm is explained for the placement problem which is based on the gradient descent method. The performance of the proposed algorithm is compared with other optimization algorithms in section 3.5, and the results are presented in section 3.6.

3.1 Placement Problem

Once the sensing model is defined, the coverage goal of the whole network should be determined. The coverage goal is closely related to the phenomena under study. For example, in a surveillance system of a building, some specific parts (e.g. entrances) are more important than others and therefore should receive a more thorough coverage. Based on the defined coverage goal, the placement optimization algorithms try to find the best parameters for the sensors so that the final network has a maximal coverage over the environment. In this section, we cover some of the common coverage goals and placement optimization algorithms mentioned in the literature.

3.1.1 Coverage Goal

Flat coverage (or field coverage) is the most common coverage goal. In this approach, all the locations in the environment have the same importance for the coverage problem [10]. In its simplest version, the goal is to get all the locations in the environment within the coverage region of at least one sensor. The measure of performance in these experiments is the ratio of the covered area to the overall deployment area.

Some applications which require high reliability for the final coverage, put the requirement that each location in the environment should be within the coverage range of at least k sensors. This is usually known as the k-coverage problem [41, 82]. The redundant coverage of the environment is justified by the possibility that some of the sensors might break while in operation (in binary coverage model as explained in the previous chapter), or in the probabilistic model, one of the sensors might not detect the event, and having coverage by more than one sensor over one location, increases the probability of detecting the event.

In the point coverage models [19] (also called weighted coverage [5]), some specific locations in the environment (such as buildings, doors, etc.) need more complete coverage than other locations. In most of the literature, the focus has been on the stationary targets, where the important locations usually model the buildings, side walks, etc. A subcategory of point coverage models are the barrier coverage models where the goal is to detect the intruders penetrating into a restricted environment [49].

Some of the literature has focused on the concepts of maximal breach path and maximal support path [57]. In the maximal breach path, the goal is to find the path between two points inside the environment which is less covered by the sensors inside the field. Conversely, the maximal support path looks for the path which has maximum coverage by sensors. The argument is that these two paths provide an estimate for best or worst coverage provided by the SN. The maximal breach path defines the vulnerability of the network with respect to intruders, and so is a valuable source of information for surveillance networks.

3.1.2 Spatial Optimization

Assuming that the coverage goal, sensing, and environmental model for a problem has been defined, it is the responsibility of the spatial optimization algorithm to find the best location (and direction if sensors are directional) of the sensors. The goal with spatial optimization is to get the placement with the least amount of computation and with the final coverage as high as possible.

One classical problem studied in computational geometry and closely related to SN placement problem is the “Art Gallery Problem” [65]. In a 2D version of the problem, the objective is to cover the area of a polygon with a number of cameras. The minimum number of cameras

and location of each camera constitute a solution to the problem. One of the solutions to this problem is achieved by dividing the area into non overlapping triangles and putting camera on the vertices of triangles. Although the problem can be solved in polynomial time for 2D problem, the extension of the problem to 3D makes it a NP-hard problem. One of the differences between this problem and SN placement problem which makes the solutions of Art galleries problem inapplicable to coverage in SNs is that in this problem it is assumed that each camera has unlimited sensing range, unless there is an obstacle in between. As we saw in the previous section, this assumption does not hold true for SNs, because usually it is assumed that sensors have a defined maximum sensing range.

Voronoi diagrams and Delaunay triangulation have also been used to optimize the location of sensors [9]. Considering the structure of Voronoi diagrams, each sensor is responsible to cover its Voronoi cell, because all of the points inside each cell are closer to the sensor generating that cell compared to other sensors. It can be easily concluded that if all Voronoi vertices of a Voronoi cell are within the coverage range of sensor, then there are no coverage holes inside that SN. Extending this idea, Wang et al. [78] proposed an effective heuristic to estimate the relative size of coverage holes using the distance between each sensor and its furthest Voronoi vertex. Using the same heuristic, a simple approach to heal the Voronoi holes is to move each sensor toward its farthest Voronoi vertex.

Voronoi diagram has also been used in the construction of maximal breach path, because points on Voronoi edges have maximum distance to closest sensors. In the same way, maximal support path lies on the edges of Delaunay triangles because Delaunay triangulation produces triangles with minimum edge length. A simple search algorithm (e.g. breadth-first-search) is used to find the optimal path for each problem.

Another approach is based on the virtual potential fields [40]. In this strategy, sensors are moved by the repulsive forces that they sense from other sensors and the obstacles in the environment. These repulsive forces tend to spread sensors across the environment. At the same time, sensors sense a viscous friction force, which helps the sensors reach a static equilibrium.

Besides the approaches based on computational geometry concepts, methods proposed for placement optimization can be classified into two categories: exact methods and heuristic-based approaches. A group of approaches have considered the placement problem as a special case of the maximum coverage problem [2, 43]. In these approaches, the problem is formulated so that a greedy algorithm can produce near-optimal results with approximation boundaries. Other algorithms, such as integer linear programming [39] and binary integer programming [94] methods, have also been employed for the placement problem.

A wide variety of meta-heuristic methods have also been applied to the placement problem, ranging from genetic algorithm [44], evolution strategies [6], evolution algorithm with specialized operators [75, 90, 97], swarm optimization [81, 96] and simulated annealing [25]. An issue

with meta-heuristic algorithms is their high computational cost, because these optimization algorithms usually need many evaluations of candidate solutions through simulations, which requires high processing resources. This issue makes these algorithms unsuitable for on-line applications with limited computing resources, where the position and orientation of sensors should be adapted after deployment and during utilization. The computational requirement also limits the size of the networks for which a solution can be found in a reasonable time.

In another set of approaches, spatial phenomena are modelled using Gaussian processes. As a solution for the placement problem under this assumption, sensors could be placed in locations with highest entropy [28], or maximum mutual information [48]. Krause et al. [48] have shown a polynomial-time algorithm for the placement problem, which is within a constant factor of the optimal result. Even though the complexity of this algorithm is reduced to $O(kn)$ for k sensors in n possible sensor locations ($n \gg k$), it can still remain inapplicable for large environment where n is proportional to the map area.

There has been some work conducted on the usage of the classical gradient descent method for the sensor placement problem. Cortes et al. [27] proposed a distributed mechanism for maximal coverage in multi-robot sensor systems. They showed that if the coverage performance between sensor and target is based on Euclidean distance, the optimal coverage could be achieved by moving each omni-directional sensor in the direction of its Voronoi cell centroid. In this approach, authors have assumed an omnidirectional coverage area for each sensor, therefore the "dominance region" of each sensor is simulated by its Voronoi cell.

Schwager et al. [72] have proposed another gradient descent method for the unmanned aerial vehicle (UAV) placement problem, but their approach relies on a two dimensional view of the environment, and the covered area of each sensor is strictly binary. As the mentioned approach is designed for placement of the UAVs, each pixel of the camera can cover different area sizes depending on the elevation of the UAV. Therefore, the main notion of coverage in the cost function is the pixel per unit area that the optimization method is trying to minimize.

3.1.3 Proposed Method

In this chapter, we are proposing to use gradient descent (GD) as an optimization method for the sensor placement problem with a weighted coverage goal. At each step of the algorithm, we calculate the analytical derivatives of the coverage function with respect to the position and orientation of each sensor and try to move them in a way that maximizes the overall coverage of the network. Besides the system was designed with weighted coverage goal, therefore, different locations in the environment can have different importance for the coverage task.

Unlike previous approaches on sensor placement optimization using the gradient descent method [27, 72], we use a realistic model for the terrain and a directional probabilistic sensing model for the coverage of each sensor (as defined in the previous chapter). We show that in

addition to its simplicity, our method can produce results comparable to more sophisticated black box optimization methods, without the need for large computational resources.

3.2 Spatial Optimization Problem

The coverage goal of the system is weighted coverage, as a result, each position $\mathbf{q}$ is also attributed with another parameter $w_q \in [0, \infty)$. This parameter defines the importance of location $\mathbf{q}$ for the coverage task. Therefore, higher values of w_q represent higher importance of the location $\mathbf{q}$ in the goal coverage problem. In the same way, we extend the previous coverage formulation (as proposed in Eq. 2.9) to encounter the effect of weights and define the global coverage of network N over the environment Ξ as:

$$C_g(N, \Xi) = \frac{1}{\sum_{\mathbf{q} \in \Xi} w_q} \sum_{\mathbf{q} \in \Xi} w_q C_l(N, \mathbf{q}). \tag{3.1}$$

The goal of the spatial optimization is to find the setting for the parameters of the sensors which maximizes the coverage of the final network over the environment. More precisely, the goal is to find a real-valued vector composed of four values per sensor $(x_i, y_i, \theta_i, \xi_i)$, so as to maximize the global sensor network coverage $C_g(N, \Xi)$ (Eq. 3.1) over a given elevation map Ξ:

$$N = \{(x_1, y_1, \theta_1, \xi_1), (x_2, y_2, \theta_2, , \xi_2), \ldots, (x_n, y_n, \theta_n, \xi_n)\}, \tag{3.2}$$

$$N^* = \underset{N}{\operatorname{argmax}}\, C_g(N, \Xi). \tag{3.3}$$

Note that the NP-hardness of the mentioned placement problem could be verified by comparison with the maximum coverage problem, which is known to be NP-hard [46]. In this comparison, each sensor at a specific position and orientation is a sample for a subset that covers a set of locations in the environment, and we want to find the k sensor positions that allow a maximum coverage when used together. In this problem, the assumption we are making is more general, as the position and orientation of sensors are continuous and coverage of a sensor for each location is probabilistic.

3.3 Implemented Optimization Methods

From this model for sensor coverage model, we compare three spatial optimization schemes: an adaptation of Simulated Annealing (SA) [47] for sensor placement, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [36], an evolutionary algorithm for real-valued optimization, and finally our proposed Gradient Descent (GD) method.

We briefly explain each method in the following sections.

3.3.1 Simulated Annealing Method on Single Sensor

SA [47] is a stochastic optimization algorithm. With a generic probabilistic heuristic approach, simulated annealing may escape local optimum and converge to global optimum, and thus may be more effective for a global optimization problem of a given function in a large search space. Our implementation of SA is described in Fig. 3.2. It requires the definition of the temperature function (temperature(t)), and the setting of two parameters (M, σ)

- Parameter M defines the maximum iterations for simulated annealing. The larger the M, the more time consuming the optimization, and the more likely the global optimum can be reached.

- σ defines candidate generator, i.e., the size of neighbourhood where subsequent solutions will be generated. An essential requirement for σ is that it must provide a sufficiently short path from the initial state to any state which may be the global optimum. Another issue is that σ should be selected so that the search path avoids becoming trapped in a local minimum, i.e., it must be large enough to cross local minima in an effort to reach the global optimum.

- Temperature function (temperature(t)) defines the probability of accepting a move in simulated annealing. Initially, the temperature(t) is set to a high value, then it is decreased at each step according to some annealing schedule, and finally ends with temperature(t) $\to 0$ towards the end of the allocated maximum steps, M. When the temperature is high, the probability of accepting a move will be high. When the cooling rate is low, the probability of accepting a move decreases. The idea is that the system is expected to wander initially towards a broad region of the search space containing good solutions, ignoring small features of the energy function; then drift towards low-energy regions that become narrower and narrower. To satisfy the conditions above, the temperature(t) is defined as an exponential decay function (see Fig. 3.1) as follows:

$$\text{temperature}(t) = \frac{1}{2} \exp\left[-\frac{2 \ln 2 \times t}{M}\right]. \tag{3.4}$$

3.3.2 CMA-ES Evolutionary Algorithm

CMA-ES [36] is an evolutionary algorithm known for its good performance and stability [34]. It updates the covariance matrix of the distribution to learn a second order model of the underlying objective function, similar to the approximation of the inverse Hessian matrix in the Quasi-Newton method in classical optimization. However, it does not require analytical derivatives of the partial derivatives.

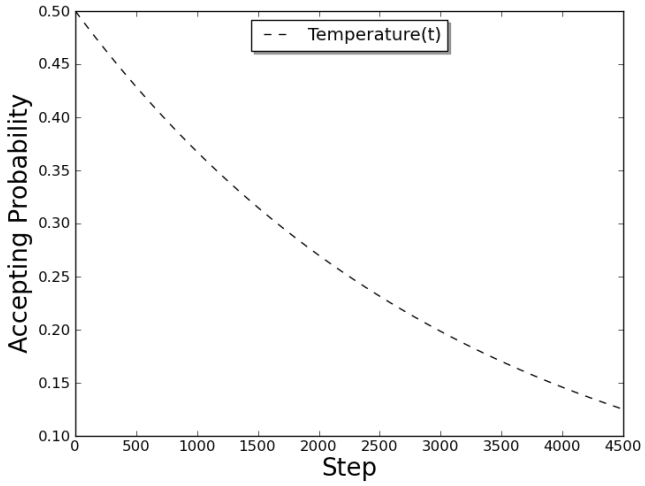


Figure 3.1: Illustration of the temperature function used for the simulated annealing method. Here, it is assumed that the maximum number of iterations (M) is 4550.

CMA-ES was chosen among different population-based stochastic optimization methods, as it was shown several times that it has overall superior performance compared to other optimization methods on a variety of standard continuous black-box optimization benchmarks (as an example in [58])

For sensor placement optimization, the position and orientation of the sensors can be encoded inside an individual, and a population of individuals can be evolved through generations. At the end of the evolution, the individual with the best coverage is chosen as the final solution.

The algorithm's parameters include the number of parents (μ), the number of offspring (λ), the mutation factor (σ), and the number of generations through which the algorithm runs. In each generation of the algorithm, a collection of the best μ candidate solutions are selected from the set of λ offsprings of the previous generation. These solutions are then used to update the distribution parameters, which will eventually generate the offsprings for the next generation.

3.4 Gradient Descent Method

Gradient descent (GD) is a classical numerical optimization method used to find the local optimum of an error function. At each step of the algorithm, the gradient of the error function (loss function in our formulation) is calculated and the free parameter of the system is updated to make a small step in the opposite direction to the gradient [15]. Next, the gradient is recalculated for the new solution, and this step is repeated until a maximum number of iterations is reached or the size of the gradient falls beneath a threshold.

Initialize sensor network with random positions uniformly distributed in placement domain (assuming domain to be in $\mathbf{p}_i \in [0,1]^2$), and random orientations,

$$\begin{aligned} x_i &\sim \text{Unif}(0,1),\ i = 1,\ldots,n, \\ y_i &\sim \text{Unif}(0,1),\ i = 1,\ldots,n, \\ \theta_i &\sim \text{Unif}(0,1),\ i = 1,\ldots,n, \\ \xi_i &\sim \text{Unif}(0,1),\ i = 1,\ldots,n, \\ N &= \{(\mathbf{p}_1,\theta_1,\xi_1),(\mathbf{p}_2,\theta_2,\xi_2),\ldots,(\mathbf{p}_n,\theta_n,\xi_n)\}. \end{aligned}$$

Assess performance of initial sensor network, $f = C_g(N,\Xi)$.
Set best sensor network and best performance to the initial one, $N_{\text{best}} = N,\ f_{\text{best}} = f$.
for $t = 1,\ldots,M$ **do**
  Select a random sensor with uniform distribution, $s \sim \text{DiscUnif}(n)$.
  Apply a random perturbation of sensor position $\mathbf{p}_s = (x_s, y_s)$ and orientation (θ_s, ξ_s) to generate new candidate sensor network placement N',

$$r_x \sim \mathcal{N}(0,\sigma),\ r_y \sim \mathcal{N}(0,\sigma),\ r_\theta \sim \mathcal{N}(0,\sigma),\ r_\xi \sim \mathcal{N}(0,\sigma),$$

$$x'_s = x_s + r_x,\ y'_s = y_s + r_y,\ \theta'_s = \theta_s + r_\theta, \xi'_s = \xi_s + r_\xi,$$

$$N' = \{(\mathbf{p}_1,\theta_1,\xi_1),\ldots,(\mathbf{p}'_s,\theta'_s,\xi'_s),\ldots,(\mathbf{p}_n,\theta_n,\xi_n)\}.$$

  Evaluate performance of new candidate sensor network N', $f' = C_g(N',\Xi)$.
  if $f' > f$, the new sensor network is better than current one, **then**
    Accept new sensor network, $N = N',\ f = f'$.
    if $f' > f_{\text{best}}$, new sensor network is better than best so far, **then**
      Set best sensor network and best performance to the current one, $N_{\text{best}} = N',\ f_{\text{best}} = f'$.
    end if
  else
    Get temperature of current iteration, $E_i = \text{temperature}(t)$.
    if $e_i < E_i,\ e_i \sim \text{Unif}(0,1)$, current sensor network is accepted given the temperature, **then**
      Accept new sensor network, $N = N',\ f = f'$.
    end if
  end if
end for
Return best sensor network found, N_{best}, as final result.

Figure 3.2: Pseudo-code of sensor placement with simulated annealing, with perturbation of one sensor position at a time.

3.4.1 Loss Function

Eq. 3.1 defines the objective function for a maximization problem, while the GD method is defined as a minimization method. Therefore, we define the loss function $L(N,\Xi)$ as the negative of the coverage function to form a minimization problem. More precisely, the loss

function for network N and environment Ξ is given by:

$$L(N,\Xi) = \frac{1}{\sum_{\mathbf{q}\in\Xi} w_{\mathbf{q}}}\left[\underbrace{\sum_{\mathbf{q}\in\Xi} w_{\mathbf{q}} L_v(N,\mathbf{q})}_{\text{visible loss}} + \nu \underbrace{\sum_{\mathbf{q}\in\Xi} w_{\mathbf{q}} L_u(N,\mathbf{q})}_{\text{non-visible loss}}\right], \tag{3.5}$$

where $L_v(N,\mathbf{q})$ is the loss incurred by the sensors of network N for which point $\mathbf{q}$ is visible and $L_u(N,\mathbf{q})$ is the loss incurred by the sensors of network N for which point $\mathbf{q}$ is non-visible, ant the ν parameter records the weighting of the non-visible component against the visible loss component. With respect to the two parts of the loss function, we take the visibility function away from the formula using two disjoint sets $U_{\mathbf{q}}$ and $V_{\mathbf{q}}$. Here, $V_{\mathbf{q}}$ contains all of the sensors from which location $\mathbf{q}$ is visible, and sensors in $U_{\mathbf{q}}$ contain all of the sensors from which location $\mathbf{q}$ is not visible. More precisely:

$$U_{\mathbf{q}} = \{\mathbf{s}\in N \mid v(\mathbf{s},\mathbf{q}) = 0\}, \tag{3.6}$$

$$V_{\mathbf{q}} = \{\mathbf{s}\in N \mid v(\mathbf{s},\mathbf{q}) = 1\}. \tag{3.7}$$

The visible loss is thus defined as:

$$\begin{aligned} L_v(N,\mathbf{q}) &= \underbrace{1 - \left[1 - \prod_{\mathbf{s}_i\in V_{\mathbf{q}}} \underbrace{(1 - C(\mathbf{s}_i,\mathbf{q}))}_{L_v(\mathbf{s}_i,\mathbf{q}))}\right]}_{C_l(N,\mathbf{q})} \\ &= \prod_{\mathbf{s}_i\in V_{\mathbf{q}}} \left[1 - \mu_d(\|\mathbf{p}_i - \mathbf{q}\|)\cdot\mu_p(\theta_i - \angle_p(\mathbf{q}-\mathbf{p}_i))\cdot\mu_t(\xi_i - \angle_t(\mathbf{q}-\mathbf{p}_i))\right] \\ &= \prod_{\mathbf{s}_i\in V_{\mathbf{q}}} \left[1 - \mu_{di}\cdot\mu_{pi}\cdot\mu_{ti}\right] \\ &= \prod_{\mathbf{s}_i\in V_{\mathbf{q}}} L_v(\mathbf{s}_i,\mathbf{q}) \end{aligned} \tag{3.8}$$

where, for simplicity, we denote $\mu_d(\|\mathbf{p}_i - \mathbf{q}\|)$ as μ_{di}, $\mu_p(\theta_i - \angle_p(\mathbf{q}-\mathbf{p}_i))$ as μ_{pi}, and $\mu_t(\xi_i - \angle_t(\mathbf{q}-\mathbf{p}_i))$ as μ_{ti}.

To explain the loss function rather informally, the visible loss is the loss incurred by the visible points; similarly the non-visible loss is the loss incurred by the non-visible points. The optimization algorithm tries to maximize the coverage of the visible points through the visible loss and tries to minimize the number of non-visible points through the non-visible loss. Although these two goals seem similar (and they do have similar effects in most of the cases), they are complementary. We will later see in section 3.4.3 how they complement each other.

The visible loss for the whole network comes to one minus the global coverage defined in Eq. 3.1, which is the coverage function used here as the parameter criterion. However, this measure provides no useful feedback for non-visible positions. Still, these positions are as important as the visible positions in the optimization problem, and their effect should be

taken into consideration. In order to provide a better signal to the optimization algorithm, we add a non-visible component to the loss function, which adds the effect of the positions in the environment that are not visible by any sensor or poorly covered by some sensors.

Non-visible loss between sensor $\mathbf{s}_i$ and location $\mathbf{q}$ is defined as the difference between the current coverage (given by visible loss $L_v(N, \mathbf{q})$) of point $\mathbf{q}$ and the coverage that it would have if it was visible from sensor $\mathbf{s}_i$:

$$\begin{aligned} L_u(\mathbf{s}_i, \mathbf{q}) &= L_v(N, \mathbf{q}) - [L_v(N, \mathbf{q}) \cdot (1 - \mu_{di} \cdot \mu_{pi} \cdot \mu_{ti})] \\ &= \mu_{di} \cdot \mu_{pi} \cdot \mu_{ti} \cdot L_v(N, \mathbf{q}) \end{aligned} \tag{3.9}$$

Accordingly, the non-visible loss for a network N and location $\mathbf{q}$ is:

$$L_u(N, \mathbf{q}) = \sum_{\mathbf{s}_i \in U_\mathbf{q}} L_u(\mathbf{s}_i, \mathbf{q}) = \sum_{\mathbf{s}_i \in U_\mathbf{q}} \mu_{di} \cdot \mu_{pi} \cdot \mu_{ti} \cdot L_v(N, \mathbf{q}) \tag{3.10}$$

3.4.2 Analytical Gradient Descent on Distance, Pan Angle and Tilt Angle

In this section, we calculate the analytical gradient on distance, pan angle and tilt angle for the sensor placement optimization problem.

Given independent variables x_i, y_i, θ_i and ξ_i, we need to obtain partial derivatives of the loss function for all of these variables (see Appendix A for more details). Let:

$$\mathrm{f}_i(N, \mathbf{q}) = \prod_{\mathbf{s}_j \in V_\mathbf{q} \setminus \{\mathbf{s}_i\}} [1 - \mu_{dj} \cdot \mu_{pj} \cdot \mu_{tj}] \tag{3.11}$$

As there are four free parameters for each sensor, we show the derivative of the overall loss with respect to a generic parameter ψ of sensor $\mathbf{s}_i$. ψ can be any of the four parameters (*i.e.*, x, y, θ, or ξ). The derivations are as follows:

$$\begin{aligned} \frac{\partial L(N, \Xi)}{\partial \psi_i} &= \frac{1}{\sum_{\mathbf{q} \in \Xi} w_\mathbf{q}} \sum_{\mathbf{q} \in \Xi} w_\mathbf{q} \left[\frac{\partial L_v(N, \mathbf{q})}{\partial \psi_i} + \nu \frac{\partial L_u(N, \mathbf{q})}{\partial \psi_i} \right] \\ &= \frac{1}{\sum_{\mathbf{q} \in \Xi} w_\mathbf{q}} \sum_{\mathbf{q} \in \Xi} w_\mathbf{q} \left[\frac{\partial \left[\mathrm{f}_i \cdot (1 - \mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}) \right]}{\partial \psi_i} + \nu \frac{\partial \left[\sum_{\mathbf{s}_j \in U_\mathbf{q}} \mu_{dj} \cdot \mu_{pj} \cdot \mu_{tj} \cdot L_v(N, \mathbf{q}) \right]}{\partial \psi_i} \right] \\ &= \frac{1}{\sum_{\mathbf{q} \in \Xi} w_\mathbf{q}} \sum_{\mathbf{q} \in \Xi} w_\mathbf{q} \ \mathrm{g}_\psi(\mathbf{s}_i, \mathbf{q}) \end{aligned} \tag{3.12}$$

where,

$$\mathrm{g}_\psi(\mathbf{s}_i, \mathbf{q}) = \begin{cases} -\mathrm{f}_i \frac{\partial [\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}]}{\partial \psi_i} & \text{if } v(\mathbf{s}_i, \mathbf{q}) = 1 \\ \nu \ L_v(N, \mathbf{q}) \ \frac{\partial [\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}]}{\partial \psi_i} & \text{if } v(\mathbf{s}_i, \mathbf{q}) = 0 \end{cases} \tag{3.13}$$

The derivative of the loss function with respect to parameter ψ can be divided into two parts. The visible loss is constant when the position $\mathbf{q}$ is not visible from the sensor $\mathbf{s}_i$, and therefore, the derivative equals zero. Similarly, the non-visible loss has a derivative of zero when sensor $\mathbf{s}_i$ can see position $\mathbf{q}$. Therefore, the derivative was divided into two equations using the $\mathrm{g}_\psi(\mathbf{s}_i, \mathbf{q})$ function.

Therefore, in each step of the GD method, all of the gradients are calculated with respect to the free parameters of each sensor (*i.e.*, x_i, y_i, θ_i and ξ_i), and the position and orientation of all sensors are updated using the pseudocode shown in Algorithm 3.3.

In this algorithm, $\psi_i^{(t)}$ is a free parameter of sensor i at iteration t and η_ψ is the learning rate for the generic parameter ψ. Furthermore, ω is the momentum parameter used to help the algorithm escape from local minima. The momentum is a classical method used for neural network optimization to encourage faster convergence of the gradient descent algorithm [71]. It increases the step sizes taken for a free parameter if the gradient points in the same direction for several iterations. Therefore, the algorithm can ignore small features in the loss function surface and skip shallow local minima.

Given a sensor network N and the environment Ξ as the input:
for $t = 1, \ldots, max\_iter$ **do**
  for all $\mathbf{s}_i \in N$ **do**
    $\Delta\psi_i^{(t)} = \eta_\psi \frac{\partial L(N,\Xi)}{\partial \psi_i} \; + \; \omega \Delta\psi_i^{(t-1)}, \; \forall \psi \in \{x, y, \theta, \xi\}$
    $\psi_i^{(t+1)} = \psi_i^{(t)} \; - \; \Delta\psi_i^{(t)}, \; \forall \psi \in \{x, y, \theta, \xi\}$
  end for
end for

Figure 3.3: The proposed gradient descent method for sensor placement optimization.

The $\mathrm{g}_\psi(\mathbf{s}_i, \mathbf{q})$ function needs the value for the derivative of $[\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}]$ with respect to the free parameters. We have calculated this derivative for all free parameters and reported the result in Eq. 3.14 to Eq. 3.17 (see Appendix A for more details).

$$\begin{aligned}
\frac{\partial \left[\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}\right]}{\partial x_i} \;=\; & \Bigg[\mathrm{dsg}(\phi_{di}, -\beta_d, -\alpha_d) \cdot \frac{(x_i - x_q)}{\phi_{di}} \cdot \mu_{pi} \cdot \mu_{ti} \\
& + \left[\mathrm{dsg}(\phi_{pi}, -\beta_p, -\alpha_p) - \mathrm{dsg}(\phi_{pi}, -\beta_p, \alpha_p)\right] \cdot \frac{y_\mathbf{q} - y_i}{\phi_{di}^2} \cdot \mu_{di} \cdot \mu_{ti} \\
& + \left[\mathrm{dsg}(\phi_{ti}, -\beta_t, -\alpha_t) - \mathrm{dsg}(\phi_{ti}, -\beta_t, \alpha_t)\right] \cdot \frac{(z_i - z_\mathbf{q})(x_i - x_\mathbf{q})}{\phi_{di}[\phi_{di}^2 + (z_i - z_\mathbf{q})^2]} \cdot \mu_{di} \cdot \mu_{pi} \Bigg]
\end{aligned} \tag{3.14}$$

$$\frac{\partial\left[\mu_{di}\cdot\mu_{pi}\cdot\mu_{ti}\right]}{\partial y_i} = \Bigg[\mathrm{dsg}(\phi_{di},-\beta_d,-\alpha_d)\cdot\frac{(y_i-y_q)}{\phi_{di}}\cdot\mu_{pi}\cdot\mu_{ti} + \left[\mathrm{dsg}(\phi_{pi},-\beta_p,-\alpha_p)-\mathrm{dsg}(\phi_{pi},-\beta_p,\alpha_p)\right]\cdot\frac{x_i-x_{\mathbf{q}}}{\phi_{di}^2}\cdot\mu_{di}\cdot\mu_{ti} + \left[\mathrm{dsg}(\phi_{ti},-\beta_t,-\alpha_t)-\mathrm{dsg}(\phi_{ti},-\beta_t,\alpha_t)\right]\cdot\frac{(z_i-z_{\mathbf{q}})(y_i-y_{\mathbf{q}})}{\phi_{di}[\phi_{di}^2+(z_i-z_{\mathbf{q}})^2]}\cdot\mu_{di}\cdot\mu_{pi}\Bigg] \tag{3.15}$$

$$\frac{\partial\left[\mu_{di}\cdot\mu_{pi}\cdot\mu_{ti}\right]}{\partial\theta_i} = \left[\mathrm{dsg}(\phi_{pi},-\beta_p,\alpha_p)-\mathrm{dsg}(\phi_{pi},-\beta_p,-\alpha_p)\right]\cdot\mu_{di}\cdot\mu_{ti}, \tag{3.16}$$

$$\frac{\partial\left[\mu_{di}\cdot\mu_{pi}\cdot\mu_{ti}\right]}{\partial\xi_i} = \left[\mathrm{dsg}(\phi_{ti},-\beta_t,\alpha_t)-\mathrm{dsg}(\phi_{ti},-\beta_t,-\alpha_t)\right]\cdot\mu_{di}\cdot\mu_{pi} \tag{3.17}$$

In these equations $\mathrm{dsg}(\delta,\beta,\alpha)$ is an auxiliary function used to simplify the equations:

$$\mathrm{dsg}(\delta,\beta,\alpha)=\frac{\beta\exp(\beta(\delta+\alpha))}{(1+\exp(\beta(\delta+\alpha)))^2}=\beta\ \mathrm{sig}(\delta,\beta,\alpha)[1-\mathrm{sig}(\delta,\beta,\alpha)], \tag{3.18}$$

which corresponds to the derivative of the so-called sigmoid function:

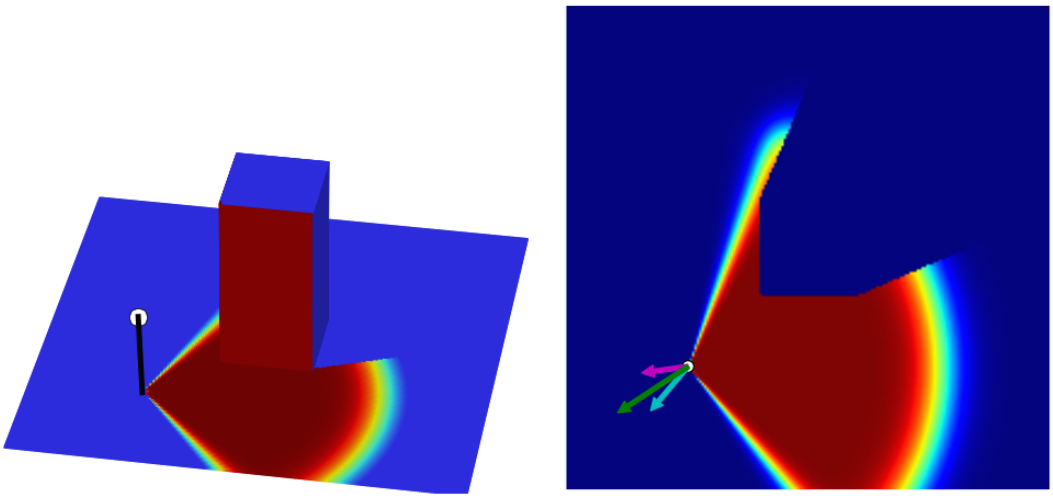
$$\mathrm{sig}(\delta,\beta,\alpha)=\frac{1}{1+\exp(\beta(\delta+\alpha))}. \tag{3.19}$$

Although we presented the algorithm in a centralized fashion, it is distributed in nature. Each sensor only needs to know its own position, orientation and visible area, in addition to the overlap between its covered area and other sensors in its neighbourhood to calculate its derivatives. Therefore, if we are using mobile sensors capable of communicating with their neighbours (which is essential in most SNs), they can transfer the required information, and each sensor can compute its movement independently.

3.4.3 Sanity Checks

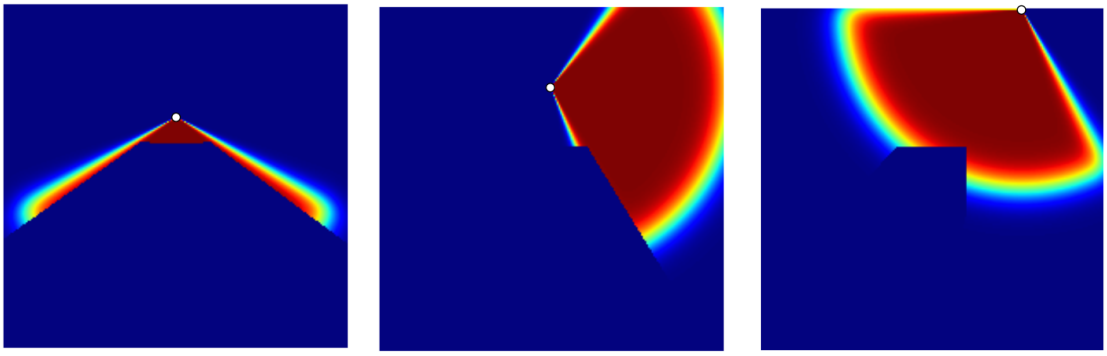
In order to check the behaviour of the GD optimization algorithm in real settings, we performed some simple experiments to check if the method performs reasonably in these simple settings, so that we can later extend the experiments to more complex experiments.

In the first experiment, we examine the effectiveness of the non-visible loss on the overall coverage achieved by the GD algorithm. The setting contains one sensor in a map that has a cube-shaped obstacle in the middle (see Fig. 3.4). The values used to generate the coverage region of each sensor are the same as Fig. 2.6. Two sets of experiments were performed on the map to optimize the location of the sensors. In the first experiment, we use only the visible loss part of the loss function (Eq. (3.5)). In the second experiment, the combination of both visible and non-visible losses were used for optimization (see Fig. 3.5). In these experiments, the final coverage percentage achieved by the visible and combined loss were 23.2% and 27.4%, respectively. Therefore, we observe that even in the simple example shown, the inclusion of the non-visible loss (in the overall loss function) helps in producing better final results.



(a) 3D view (b) 2D view

Figure 3.4: Simple experiment to show the effect of different parts of the loss function, with a 3D view (**a**), and 2D view (**b**) of the covered region of the sensor in a flat environment having a cubic obstacle in the centre. In (**b**), the cyan arrow shows the non-visible loss gradient, the magenta arrow shows the visible loss gradient and the green arrow is the combined loss gradient aggregating the effect of the two mentioned losses.

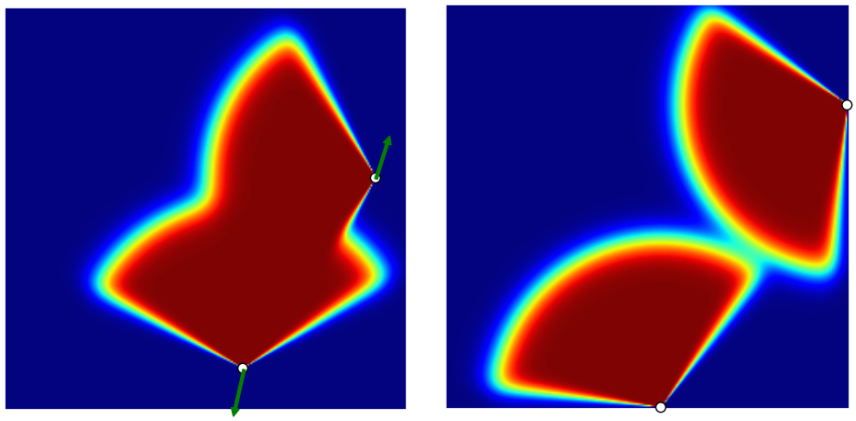


(a) Initial position (b) Visible loss result (c) Combined loss result

Figure 3.5: Optimizing with different loss functions: (**a**) the initial position of the sensor over a map (facing one of the sides of the obstacle); (**b**) the final result of the optimization using only the visible loss with a final coverage of 23.29%; (**c**) the final result of the optimization performed using both the visible and non-visible loss parts with a final coverage of 27.47%.

In the second experiment, we evaluated the effect of overlap between the coverages of two sensors over the gradient calculated for each one (see Fig. 3.6). In this experiment, two sensors are initially placed in a flat environment (no obstacle), where there is an overlap between the coverage of the two sensors. The gradient calculated for each sensor makes the sensors move away from each other to reach a final position where the total coverage is maximum.

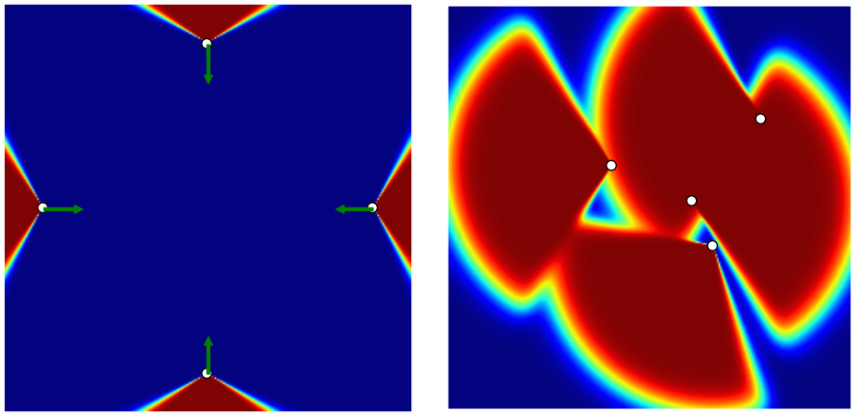
In the final experiment, we show how a coverage gap can be filled with the GD method (see Fig. 3.7). In this experiment, there are four sensors in a flat environment, each heading



(a) Initial positions (b) Final positions

Figure 3.6: An experiment to show the effect of overlap on coverage over the gradient; (**a**) the initial position of the sensors and the green arrow shows the gradient of the loss function with respect to each sensor; (**b**) the final position of the sensors after the optimization.

outward, so creating a coverage gap in the centre. The goal is to determine whether the GD method can detect this gap and displace the sensors in order to cover it. As shown in Fig. 3.7b, the sensors have moved to cover most of the gap.



(a) Initial position (b) Final position

Figure 3.7: An experiment to show the effect of a coverage gap over the gradient; (**a**) initial positions of the sensors, with the green arrow showing the gradient with respect to each sensor; (**b**) final position of the sensors after the optimization has been completed. As can be seen the initial coverage gap in the centre has been filled by the sensors.

3.5 Experiments

In order to evaluate the performance of the proposed GD method, we compared it with two other optimization methods, namely simulated annealing (SA) and covariance matrix adaptation evolution strategy (CMA-ES), which we briefly summarized before. Next, we

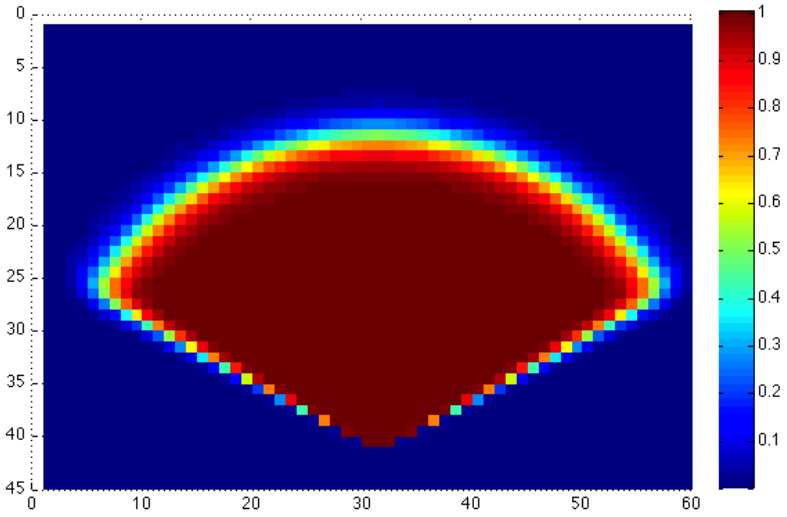


Figure 3.8: Probabilistic coverage model of a sensor. Assuming that a sensor is positioned at (30,40) heading upward, the colour shows different degrees of coverage for points inside the map.

explain the parameters of the sensor model based on the model proposed in chapter 2.

3.5.1 Sensor Model Parameters

For a reasonable model of a sensor, we propose to use the parameters shown in Table 3.1. With these values, the sensors have 50% of the maximum coverage at 30m or at sensing angle of 120^o. The coverage region of a sensor with these parameters is shown in Fig. 3.8.

Table 3.1: The parameter values for a realistic model of a sensor which has a 50% of the maximum coverage at 30m or span of view of 120^o.

Parameter	α_d	β_d	α_p	β_p	α_t	β_t
Value	30	1	60	1	30	1

3.5.2 Maps

To conduct our experiments, we first selected a mountainous area in North Carolina, USA. The data was provided by a raster layer map in the "OSGeo Edu" dataset (Available at `http://grass.osgeo.org/download/sample-data/`). More specifically, we focused on a portion of the map that covers a small watershed in a rural area near Raleigh, the capital city of North Carolina. The coordinate system of the map is the NC State Plane (Lambert Conformal Conic projection), metric units and North American Datum (NAD83) geodetic datum. We used four portions of the map for our experiments. The information concerning different

selected portions of the map is presented in Table 3.2. Testing the optimization methods with different map sizes, allows the scalability of each method to be verified.

We also tested the optimization algorithms over a map of Université Laval campus. The map of the area is shown in Fig. 3.9. In this experiment, which is an example of a surveillance system for the campus, the goal is two-fold. First, we want to test the performance of different methods in the presence of man-made obstacles (*i.e.*, buildings). Second, the target area is weighted, meaning that each pixel is attributed with a different weight ($w_{\mathbf{q}}$) as described in section 3.2. For this experiment, we assume that the top of the buildings have low importance in the total coverage ($w_{\mathbf{q}} = 0.1$), the streets have an average importance ($w_{\mathbf{q}} = 0.4$) and the ground level where the pedestrians walk have the highest importance ($w_{\mathbf{q}} = 0.8$). The specifications for the coordinates of the campus maps are also provided in Table 3.2.

Table 3.2: Information concerning the test maps used for the experiments.

Method	**NC-A**	**NC-B**	**NC-C**	**NC-D**	**UL-A**	**UL-B**
Highest elevation (m)	130.65	127.81	131.6	131.55	100.0	107.3
Lowest elevation (m)	125.95	109.3	103.7	103.76	85.25	82.85
No. of Columns	100	200	500	500	100	250
No. of Rows	100	250	250	500	100	200
No. of pixels	10,000	50,000	125,000	250,000	10,000	50,000

Figure 3.9: Part of the Université Laval (UL) map, chosen for the weighted experiments. Here, different parts of the map have different weights. Buildings are shown in red and have a weight of $w_{\mathbf{q}} = 0.1$, the ground is represented in green and has a weight of $w_{\mathbf{q}} = 0.8$ and the streets are shown in black and have a weight of $w_{\mathbf{q}} = 0.4$.

3.5.3 Optimization Algorithm Settings

For SA, the perturbations for positions and orientations follow a Gaussian distribution with standard deviation σ_{sa}. The optimal value for σ_{sa} has been established by trial and error (over

the range $\sigma_{sa} \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.2\}$), and set to $\sigma_{sa} = 0.01$ for each map. CMA-ES is run following recommendations of its author [35], with a population of $\lambda = \lfloor 4+3\log(N) \rfloor$ offspring and $\mu = \lfloor \frac{\lambda}{2} \rfloor$ parents. Here, N is the dimensionality of the given problem, determined by the number of sensors in each map. A mutation factor $\sigma_{cma} = 0.167$ is also used. For the GD method, the list of parameters include the learning rates η_x and η_y for positions, the learning rate η_θ for pan angles, the learning rate η_ξ for tilt angles and the weighting parameter ν. The values of these parameters used in the experiments have been found by trial and error, with the same values used over all maps. For that, we performed a grid search for all of the parameters over the NC-A map. More precisely, we had a grid search over parameters $\eta \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$, $\nu \in \{0, 0.5, 1, 1.5, 2\}$ and $\omega \in \{0, 0.5, 1\}$. Then, we chose the best five selections of the parameters and among them chose the one that performed best (on average) on all of the other maps. All parameters used for the experiments are summarized in Table 3.3.

Table 3.3: The parameter values used for simulated annealing, gradient descent and covariance matrix adaptation evolution strategy (CMA-ES) methods.

	SA	**CMA-ES**	**GD**					
Parameter	σ_{sa}	σ_{cma}	η_x	η_y	η_θ	η_ξ	ν	ω
Value	0.01	0.167	0.05	0.05	0.5	0.005	1	0.5

The most computationally demanding part of each algorithm lies in calculating the overall coverage for a candidate solution. Therefore, to allow fair comparison between different optimization methods regarding computation requirements, we put a limit on the number of candidate network coverage calculations each method can do. SA and GD methods perform one coverage calculation per iteration; therefore, the maximum number of iterations for these methods equal the maximum number of coverage calculations for each map. The CMA-ES method calculates coverage for each of its offspring at each generation, so this algorithm proceeds for $\frac{t_{max}}{\lambda}$ iterations on each map, where t_{max} is the maximum number of iterations and λ is the number of offspring. We have reported the maximum iteration for each map and algorithm in Table 3.4. The maximum number of iterations define the stop criterion for each algorithm on each map.

Table 3.4: Maximum number of iterations for each method on each map.

Method	**NC-A**	**NC-B**	**NC-C**	**NC-D**	**UL-A**	**UL-B**
Simulated Annealing (SA)	6000	30,000	103,500	144,000	6000	30,000
Gradient Descent (GD)	6000	30,000	103,500	144,000	6000	30,000
CMA-ES	400	1500	4500	6000	400	1500

In the GD method, we added another criteria for the termination of the algorithm. In this algorithm, if the fitness of the candidate solution is not improved after a specific number of iterations, the algorithm will stop and report the best solution found so far. In our implementation, this number of iterations is set to be 50.

For the CMA-ES and SA methods, each sensor placement optimization scheme has been run 30 times, from which are estimated the average and the standard deviation of each method. CPU times are also averaged over the 30 runs, in order to compare the resources required by each method to produce a solution. These time values, reported in Table 3.5, have been evaluated by running the methods on a single Intel i7 core running at 2.8 GHz.

Comparatively, the running time of GD on a specific map is much less than the other two methods. Therefore, we implemented a restart mechanism for our experiments. More precisely, we accomplished this by calculating, on average, how many runs GD is able to perform with restart to be comparable in terms of the number of evaluations calculated. In stochastic optimization, restarting consists in making several runs of the algorithm and using the best solution of these runs as the final result. Joined to a stop criterion that halts the optimization process as soon as it converges, this leads to better results than one single long run. SA and CMA-ES are executed 30 times, so GD is executed for 30× the average number of runs with restart executed on each map. In Table 3.5, for the GD method, we have reported the average coverage percentage between the 30 repetitions with restart as "GD with restart" and the average between all runs as "GD single run average".

Each optimization method begins the optimization process from an initial network setting. We take the random distribution as our initial setting for the optimization methods. Random distribution is the simplest scenario, meaning that all sensors are randomly distributed in the environment.

All optimization programs are implemented in the Python language. The CMA-ES implementation was taken from Distributed Evolutionary Algorithms in Python (DEAP), available at `http://deap.gel.ulaval.ca` [32]; a Python library for evolutionary algorithms developed at Université Laval.

3.6 Results

Here, we compare the performance of optimization methods on the mentioned test maps. Each optimization method was run 30 times, except GD, which was run for 30× the average number of runs with restart, from which the average coverage percentage and standard deviation were recorded. A good optimization method should have high coverage and low standard deviation. The results are reported in Table 3.5. In the experiments, each scheme has been run 30 times, with coverage averages and the corresponding standard deviations reported. Note that 100%

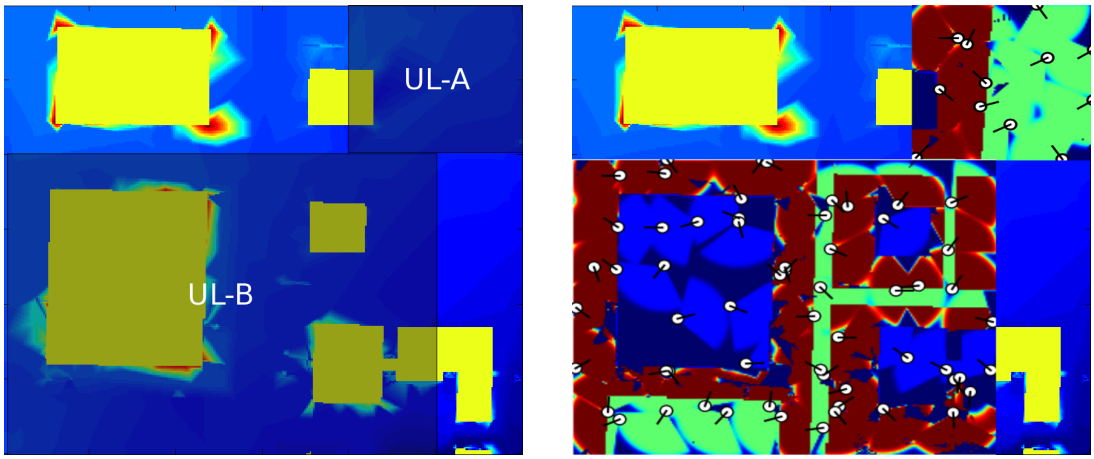
coverage is not possible with a finite number of sensors, given its probabilistic nature. Fig. 3.10 and Fig. 3.11 also present the best results obtained by the GD method for each map. There are several aspects that should be pointed out in Fig. 3.10b. First, the coverage of each sensor is scaled by the weight of the area that the sensor is placed on (the same way that the coverage of sensors are weighted in Eq. (3.1)). This gives us the ability to differentiate between areas with different weights. For example, on the top of the buildings (where the weight is smallest), the sensors have blue coverage, or on the roads, the coverage of sensors is green. Next, notice that the area which has lower weights (e.g., the top of the buildings) is not covered as well as the areas with higher weights (e.g., the ground level). This makes sense as the algorithm has balanced the coverage capacity and put more emphasis on the more important areas.

Table 3.5: Coverage percentage and standard deviation on the target areas with various numbers of sensors. Results in bold denote the best results that are statistically significant according to the Wilcoxon–Mann–Whitney test (pairwise compared with the other results with p-value of 0.05). As the range of the time requirements for different algorithms is large, time is reported either in seconds (s), minutes (m), hours (h) or days (d).

Method	**NC-A**	**NC-B**	**NC-C**	**NC-D**	**UL-A**	**UL-B**
Number of sensors	12	60	150	300	12	60
Search dimension	48	240	600	1,200	48	240
SA average	88.4%	89.4%	89.0%	84.9%	89.1%	83.6%
SA SD	1.6%	0.6%	0.4%	0.5%	1.7%	2.5%
SA CPU time	17.7 m	10.2 h	4.1 d	11.1 d	15.1 m	10.1 h
CMA-ES average	**90.2**%	**92.4**%	91.7%	60.3 %	90.4%	**87.6**%
CMA-ES SD	1.9%	0.7%	0.6 %	2.3 %	1.5%	1.0%
CMA-ES CPU time	18.0 m	11.0 h	4.4 d	11.4 d	15.9 m	11.1 h
GD with restart	88.6%	92.1%	**95.1**%	**91.7**%	**91.4** %	83.2%
GD single run average	85.0%	90.7%	94.3%	90.3%	86.7%	76.2%
GD single run SD	0.9%	0.2%	0.1%	0.3%	1.0%	1.6%
GD average # of restart	13	19	112	165	15	27
GD CPU time single run	85 s	33.9 m	52.0 m	1.5 h	72 s	22.2 m

Results indicate that CMA-ES outperformed other methods on three maps and GD performed better on the other three maps. In general, the performance of the three methods is very similar on smaller maps, but on larger maps, the difference is significant. CMA-ES performed better on smaller maps, while GD with restart performed better on larger maps. Still, the difference between the performance of CMA-ES and GD on smaller maps ($\sim 1\%$) is much less than the difference on larger maps ($\sim 30\%$), except for one small map (UL-B) where the difference is larger ($\sim 4\%$). The larger difference of the performance on UL-B map could be related to the abrupt changes of the elevation and visibility on this map, which itself is caused by the abundance of buildings on this map. These changes make the derivative of the objective function discontinuous in the search space; therefore, it will be harder for the GD method to escape local minima. The main difference occurs on the largest map (NC-D) where CMA-ES

is unable to obtain a better estimation than the initial random positions. It is well-known that CMA-ES does not scale well in high dimensionality [70]. The reason is that large maps generate high dimensional search spaces (*i.e.*, hundreds of dimensions), and estimating the covariance matrix from a relatively small sample set is brittle. Therefore, the CMA-ES with full covariance matrix is only usable for problems with a small number of sensors (less than 250).



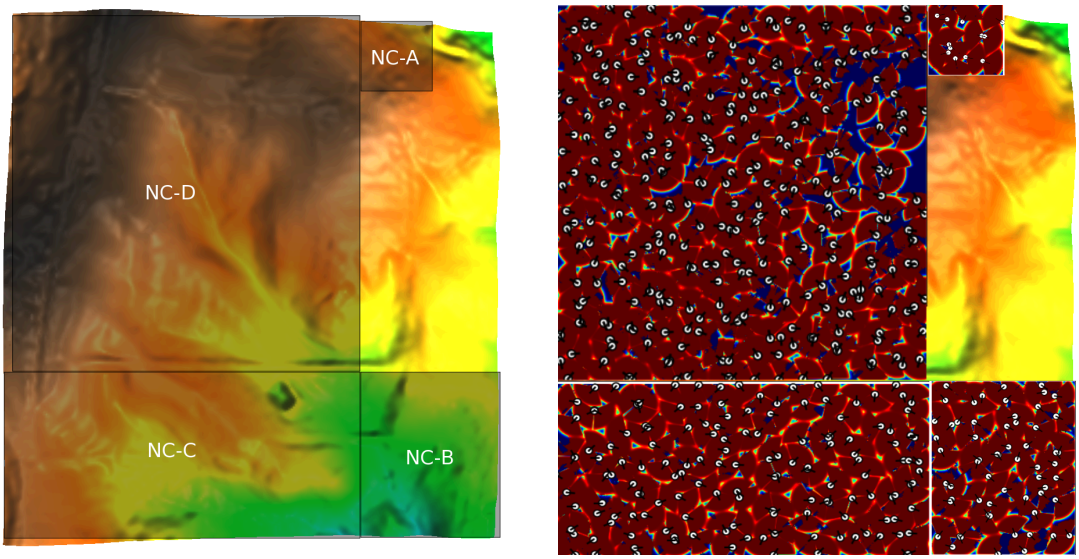
(**a**) Campus maps (**b**) Placement results with gradient descent

Figure 3.10: Result of placement on the Université Laval campus map: (**a**) two sub-parts of the maps used for the experiments, the specifications concerning each map are given in Table 3.2; (**b**) position of the sensors in the best placement obtained by the GD method. Here, white circles represent the position of sensors on the map and the black line connected to each circle shows the direction of each sensor. Different colours present different degrees of coverage using the same colour map as Fig. 3.8.

The other major difference is attributed to the computational demand of different algorithms. SA and CMA-ES consume roughly the same amount of computational power, while GD requires between 13- to 165-times less computation time. For example, Fig. 3.12 compares the speed of convergence between different methods on map NC-B.

3.7 Conclusion

This chapter presented an analytical gradient descent (GD) algorithm for optimizing sensor placement (spatial optimization). The algorithm was implemented with a realistic model for the environment and a probabilistic model for the sensors. Other optimization methods (CMA-ES and SA) were also implemented and compared with the GD method. In comparison, CMA-ES performed slightly better on smaller maps, while GD performed significantly better on larger maps. Another advantage of the GD method lies in its processing time, as on the tested maps, its performance was between 13- to 165-times superior to that of the other two methods. The final advantage of the algorithm is related to its distributed nature. Although,



(**a**) North Carolina maps (**b**) Placement results with gradient descent

Figure 3.11: Result of placement on the North Carolina rural area map: (**a**) four sub-parts of the maps used for the experiments (the specification concerning each map is given in Table 3.2); (**b**) position of the sensors in the best placement obtained by the GD method.

we have tested the algorithm in a centralized fashion, GD has the capability to be executed in a distributed fashion, where each sensor only needs to know the position and coverage of neighbour sensors.

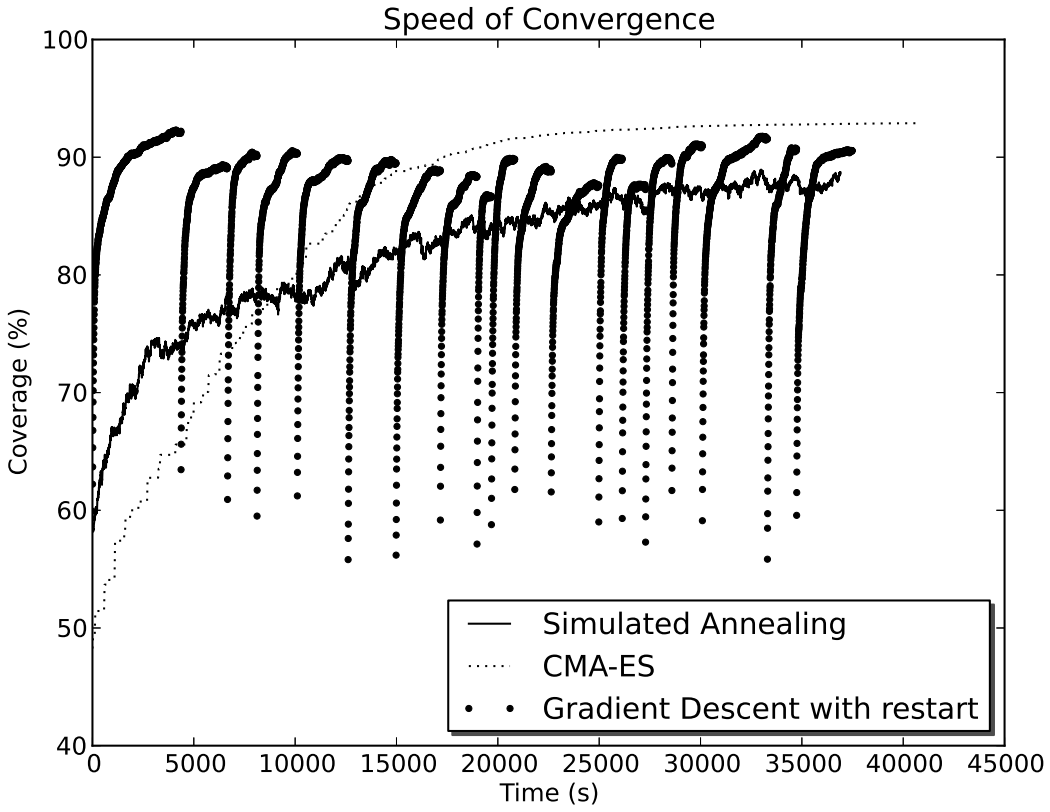


Figure 3.12: Comparison between speed of convergence for different methods on a sample run for map NC-B.

Chapter 4

Trajectory Prediction

In the temporal optimization, the assumption is that the sensors have already been placed in the environment, using the spatial optimization algorithms presented in chapter 3. Still, the sensors have some parameters (e.g. direction) which can be changed over time during the operation of the network. The goal of the temporal optimization is to find a control mechanism to automatically change the variable parameters of the sensors, so that the final network has the maximal coverage over phenomena which is changing over time. Here, we focus on the problem of covering a number of moving targets in the environment.

For this purpose, first the targets should be detected, and their location be estimated with good accuracy. Next, the trajectory of each target should be predicted for a future time step. The reason for the prediction step is that all the mentioned steps (target detection, location estimation, coverage optimization) will take some time. Therefore, the algorithm should plan to cover the targets before their movement. This shows the importance of the trajectory prediction step. In this chapter we look at the algorithms proposed for the trajectory prediction step, and in the next chapter we will look at the control mechanisms available to change the parameters of the sensors.

In section 4.1 we look at the algorithms proposed for the trajectory prediction problem in the literature. The approaches are categorized under short term and long term prediction methods. In section 4.2, we define the problem of trajectory prediction and propose a performance criteria. In section 4.3, we go over some short term prediction methods and propose some probabilistic prediction methods for this problem. In section 4.4, long term prediction methods are studied and kernel density estimation method is proposed for the problem. Next, section 4.5 presents the experiments settings, and finally in section 4.6 results under both categories are presented.

4.1 Trajectory Prediction

Trajectory prediction is the process of estimating the future location of the target moving inside an environment. The trajectory is usually modelled as a sequence of observed locations of the target. Therefore, in trajectory prediction, the goal is to predict the future location of a target, given the history of its previous locations. A simple approach for this problem is to perform estimation using only the previous trajectory of the same target. We refer to the approaches in this category as the short term prediction methods, as in these methods, the prediction step can not be arbitrarily large. Another possible approach is to use the trajectory of other targets who traversed the same environment. We refer to these approaches as the long term prediction methods. Next, we cover some of the methods proposed in the literature under each of the two mentioned groups.

4.1.1 Short Term Prediction

In the context of sensor networks for target tracking, there has been some work on the trajectory prediction of targets. Xu et al. proposed an algorithm in [86, 87, 88] to predict the future location of a target using three heuristics (constant, average, and exponential average). In the constant heuristic, the assumption is that the targets keeps the current direction and speed in the future. In the average heuristic, the average of the previous movements is used for prediction and in the exponential average heuristic, the exponential average of the recent movements predicts the future movement. In all the mentioned methods, the future location is predicted as a single point, and does not include the prediction uncertainty into the model.

Fuemmeler et al. [33], considered the problem from the perspective of finding the best scheduling mechanism for the sleep period of the sensors. They formulated the problem as an example of partially observable Markov decision process (POMDP), and considered different tradeoffs between energy savings and the tracking errors. In a similar work, Yeow et al. [89], considered the trajectory prediction problem as an example of Markov decision process (MDP) and applied Q-learning to solve it.

Brooks et al. compared in [16] the performance of three methods for trajectory prediction in SNs. The compared methods were pheromone routing, extended Kalman Filter, and belief networks. The authors did not provide any comparison between the performance of different methods over a single setting. In a similar work, Chong et al. [26], proposed a distributed Kalman filter mechanism to track several targets moving in the environment. The main focus of the work is to minimize the communication cost between sensors by sending state information of targets only when necessary.

More recently, Jiang et. al. [42], proposed two prediction methods to estimate the location of moving targets. They called the two methods kinematics-based prediction, and probability-based prediction. In kinematics-based prediction, the assumption is that the target will keep

the same direction and speed as the previous movement. This corresponds the constant heuristic of [87, 88]. For probability-based prediction, authors assumed that target's displacement is represented in polar coordinates. In these coordinates, the uncertainty of displacement size is represented by a normal distribution, and for the displacement direction a linear model was used to represent uncertainty. A weaker point that require justification is that, once calculating the parameters of the normal distribution, authors mentioned that they have used the "68-95-99.7 rule" of normal distribution (instead of a more straightforward method such as maximum likelihood).

4.1.2 Long Term Prediction

The main problem with all the short term prediction approaches is that the future location of the target cannot be estimated for far future time steps. Specifically, as the prediction time increases the uncertainty of the prediction, it also increases to the limit that it becomes no more useful for the application. Another possible approach is to use the history of previous targets who have already traversed the same environment for predicting the trajectory of the new targets.

One approach for long term trajectory prediction uses hidden Markov models (HMM). In these approaches the set of previous trajectories is summarized into the HMM model, and the similarity of a new trajectory is calculated with each of the HMM states from the HMM emission probabilities it produces. Two types of HMM models can be distinguished for trajectory prediction: 1) behavioural models, where the HMM is designed to model high level intentional states (e.g., interactions between peoples) [64, 85], and 2) physical models, where states correspond to positions of the target represented as line segments [74], mixture of Gaussians [13], Voronoi diagrams [76], etc.

HMM methods can be seen as parametric approaches for trajectory prediction, where the hidden states, emission and transition probabilities constitute the parameters of the system. The problem of this approach is that it is not clear what is a good representation for hidden states (e.g. line segment, probabilistic distribution). Moreover, depending on the chosen hidden states, the emission probability between a state and an observation must still be defined.

The problem of defining parameters for parametric approaches (e.g., HMM), gives rise to a non-parametric approaches. In non-parametric approaches, instead of summarizing previous trajectories into states, a similarity function is defined between two trajectories. Here, the two trajectories are that of a new target and another trajectory from the set of trajectories of previous targets. We exploit the fact that the new target will (more probably) have the same trajectory as those with which it has a higher similarity.

Compared to parametric methods, non-parametric methods have desirable properties for tra-

jectory prediction, such as their ability to work in an online fashion. In other words, with parametric methods, the addition of new trajectories in the history dataset implies the re-estimation of all system parameters, while this is not the case for non-parametric methods. This is an important property for the trajectory prediction problem, because new trajectories are detected all the time and they should be added to the history dataset. Non-parametric methods have the advantage of being insensitive to the addition of new trajectories, as each prediction is computed on-demand over the history dataset, with little or no other computation required beforehand. However, the computation required is proportional to the history dataset size, such that for a large set of trajectories observed, one should limit computation by restricting the size of the history dataset used in some way (e.g., through a sliding window). Specifically, in the case where the obstacles in the environment are mobile, the combination of online trajectory addition and the sliding variable becomes important, as the most recent trajectories should be used for prediction of new trajectories.

In non-parametric approaches, the main problem is the definition of the similarity function between two trajectories. There are several approaches proposed to solve this problem. As a simple example Nanni et al. [61] proposed to use average Euclidean distance between two trajectories as a measure of dissimilarity between them. The problem with this approach is that the results will be poor if the trajectories are not well aligned [67]. In a similar but more accurate approach Bashir et al. [12] calculated the dissimilarity between two trajectories by using the Euclidean distance between the Principal Component Analysis (PCA) coefficients of each trajectory.

Dynamic Time Wrapping (DTW) [45], Longest Common Subsequence (LCSS) [77], and Edit Distance on Real sequence (EDR) [22] are the three methods which calculate the similarity by alignment between the subsequent trajectories. The basic idea behind the three algorithms is sequence alignment. These methods use a dynamic programming approach to align two trajectories such that the length of the portion which matches between the two trajectories is maximized. It has been shown that these methods produce stable results for trajectory prediction problem [93], as they are robust to noise in the input data [17].

The concept of applying density estimation on trajectories has already been proposed by Chen et al. [20], but in their work the distance function between trajectories was defined on the absolute locations and without including the displacements. Kernel Density Estimation (KDE) is a well-known non-parametric density estimation method, but there has been very limited applications of the KDE method for trajectory prediction. KDE was usually used for the problem of anomaly detection among trajectories. Laxhammar et al. [50] was among the first to propose the application of KDE for the problem of anomaly detection in trajectory datasets. In their work, they assumed trajectories have a Markovian property that the next location of each target depends only on its previous two locations. Later Oliver [63] also applied KDE for anomaly detection in trajectory datasets.

4.1.3 Proposed Methods

In this chapter we are proposing methods under both the short term and long term prediction algorithms. For the short term prediction method, we are proposing the probabilistic exponential average, and the multimodal methods, and also propose to use maximum likelihood to learn the parameters of the unimodal method.

As for the long term prediction methods, we propose to use KDE, to model the similarity between two trajectories, and to use that similarity to predict the future locations of targets. We also propose a mechanism to adapt the bandwidth parameters of the KDE method for the case of missing observations in real datasets.

4.2 Trajectory Prediction Problem

Trajectory prediction is the process of estimating the future location of a given target using the information of its previous movements (in short term trajectory prediction) and the movements that other targets have made in the same environment (in long term trajectory prediction). More precisely, assume $\mathbf{T}_j^{(t)}$ is an on-going trajectory of target j in the environment Ξ, currently being observed at time t, where $\mathbf{T}_j^{(t)} = (\mathbf{L}_j^{(t)}, \Delta\mathbf{L}_j^{(t)})$. Here, $\mathbf{L}_j^{(t)} = (x_j^{(t)}, y_j^{(t)})$ represents the location of the target and $\Delta\mathbf{L}_j^{(t)} = (\Delta x_j^{(t)}, \Delta y_j^{(t)})$ its previous time step displacement.

We wish to measure the location of the target after s time steps (i.e. $\widehat{\mathbf{T}_j^{(t+s)}}$) as accurately as possible. Here we propose a probabilistic model for the future location of each target. More precisely, for a target $\mathbf{T}_j$ at time $t+s$ we define a discrete random variable $X_j^{(t+s)}$ over the sample space of all the locations $\mathbf{q} \in \Xi$. $X_j^{(t+s)}$ is defined by a probability mass function $P(X_j^{(t+s)} = \mathbf{q})$ which returns the probability of target $\mathbf{T}_j$ being at location $\mathbf{q}$ at time $t+s$. For brevity, we note this measure as $\mathcal{P}_{j\mathbf{q}}^{(t+s)}$. The goal is to incrementally learn the probabilistic model for each target as it moves within the environment.

The performance of the trajectory prediction method can be evaluated through the average (or expected value of) distance between the prediction and the actual location of the target. More precisely, prediction error for the location of a test trajectory $\mathbf{T}_j$ at time $t+s$ is defined as follows:

$$E_j^{(t+s)} = \sum_{\mathbf{q}\in\Xi} \mathcal{P}_{j\mathbf{q}}^{(t+s)} \left\| \mathbf{q} - \mathbf{L}_j^{(t+s)} \right\|_2, \tag{4.1}$$

which is the sum over the Euclidean distance between all possible positions of the environment and the actual location of the target at time $t+s$, weighted by the predicted probability $\mathcal{P}_{j\mathbf{q}}^{(t+s)}$ that the target will be at position $\mathbf{q}$. The goal of the prediction method is to find the prediction mechanism which gives the minimum value for the prediction error:

$$\mathcal{P}_{j\mathbf{q}}^{*(t+s)} = \underset{\mathcal{P}_{j\mathbf{q}}^{(t+s)}}{\operatorname{argmin}} \; E_j^{(t+s)}. \tag{4.2}$$

The result of the trajectory prediction at time t is a prediction map $\mathbf{R}^{(t)}$. The prediction map, sums up the result of prediction for different targets which are moving in the environment. So that, if the prediction method predicts that two targets are moving into one location with high probability, then that location is getting a high value in the prediction map. More precisely, for each element $r_{\mathbf{q}}^{(t)} \in \mathbf{R}^{(t)}$ we have:

$$r_{\mathbf{q}}^{(t)} = \sum_{j} \mathcal{P}_{j\mathbf{q}}^{*(t+s)}. \tag{4.3}$$

4.3 Short Term Trajectory Prediction

In this section we are looking to predict the future location of a target considering the previous movements of the same target in the environment. The main idea behind the short term prediction methods is that the target tend to proceed with the same direction within its movement. The main problem with this approach is that the uncertainty related to the prediction grows linearly with the prediction step. We will elaborate more on this issue at section 4.3.8. But for now, we assume that the prediction horizon is restricted to one step ahead. In other words, at each time step, the location of all present targets are predicted for the next time step.

The short term methods are limited for predicting one time step into future. Therefore, the prediction method can be defined in terms of the displacement that the target will make at each time step. The displacement vector of each target $\mathbf{T}_j$ at time t is represented by $\Delta\mathbf{L}_j^{(t)} = \mathbf{L}_j^{(t)} - \mathbf{L}_j^{(t-1)}$. The goal is to predict the next displacement $\widehat{\Delta\mathbf{L}}_j^{(t+1)}$ using the history of past movements $\{\Delta\mathbf{L}_j^{(t_j+1)}, \Delta\mathbf{L}_j^{(t_j+2)}, \ldots, \Delta\mathbf{L}_j^{(t)}\}$.

4.3.1 Constant Prediction

The simplest assumption for predicting the displacement in the next time step is that the targets keeps the current direction and speed in the next displacement. This was proposed as the constant method in [86, 87, 88]:

$$\widehat{\Delta\mathbf{L}}_j^{(t+1)} = \Delta\mathbf{L}_j^{(t)}. \tag{4.4}$$

4.3.2 Average Prediction

Another approach would be to take the average of all the previous movements that the target has made in the environment. We will refer to this method as the average method [86, 87, 88]:

$$\widehat{\Delta\mathbf{L}}_j^{(t+1)} = \frac{1}{n} \sum_{s=t_j+1}^{t} \Delta\mathbf{L}_j^{(s)}, \tag{4.5}$$

where $n = t - t_j$.

4.3.3 Exponential Average Prediction

One of the problems related to the average method is that it gives equal weight to all previous movements. For example, if a target has moved towards the West and then suddenly changed direction and moved towards the East, this method keeps predicting the next position within the West side, until the East movements can outweigh the West movements. Therefore, it is clear that more recent movements are more important. Therefore in the third method, Xu et al. [86] proposed the exponential average method:

$$\widehat{\Delta\mathbf{L}}_j^{(t+1)} = (1-\eta)\,\widehat{\Delta\mathbf{L}}_j^{(t)} + \eta\,\Delta\mathbf{L}_j^{(t)}, \tag{4.6}$$

where $\eta \in [0,1]$ is the learning rate

4.3.4 Probabilistic Exponential Average Prediction

All the three mentioned prediction methods has a similar shape for the final result. In all these methods the final result is a vector, so these methods gives maximum probability to one location in the environment and all neighbour locations obtain zero probability. In terms of the formulation we developed in section 4.2 for the probabilistic prediction, we have $\mathcal{P}_{j\mathbf{q}^*}^{(t+1)} = 1$ for one location $\mathbf{q}^*$ and zero for all other locations. This represents a simple binary probability function; we can further improve these approaches by using a bivariate Gaussian distribution around the predicted location:

$$\mathcal{P}_{j\mathbf{q}}^{(t+1)} = \mathcal{N}_2(\mathbf{q}; \widehat{\Delta\mathbf{L}}_j^{(t+1)}, \mathbf{\Sigma}), \tag{4.7}$$

where $\mathcal{P}_{j\mathbf{q}}^{(t+1)}$ is the predicted probability that the target moves to location $\mathbf{q}$ in the next time step, $\widehat{\Delta\mathbf{L}}_j^{(t+1)}$ is the mean of the bivariate distribution which we calculated in (4.6), and $\mathbf{\Sigma}$ is the covariance matrix. Here, for simplicity we assume $\mathbf{\Sigma} = \sigma_p^2\mathbf{I}$. From now on, we refer to this prediction method as the probabilistic exponential average approach.

4.3.5 Multimodal Prediction

We can further improve the probabilistic exponential average prediction, by addressing the following problem. Assume the target has moved to the West for some time and then moved towards the North. After the transition, the predicted location would be towards the North-West for some time until it is gradually directed towards the North. While the target has never moved towards the North-West, and still, after the prediction has moved North, there is no probability of the target moving towards the West, while it is more reasonable that the probability of a West movement would be larger than a probability of a movement in other directions (e.g. East or South). To solve this problem, we define the probabilistic measure for all the previous movements and then sum up their values for all the locations $\mathbf{q}$. Therefore, in each time step, the probabilistic model of the last movement $\mathcal{P}_{j\mathbf{q}}^{(t)}$ for location $\mathbf{q}$ is added

to the prediction based on the recent movement of the target with respect to that location $R_{j\mathbf{q}}^{(t)}$, to predict the probability for the future time step. We call this approach the multimodal approach and it is formalized as follows:

$$\mathcal{P}_{j\mathbf{q}}^{(t+1)} = (1-\eta)\,\mathcal{P}_{j\mathbf{q}}^{(t)} + \eta\, R_{j\mathbf{q}}^{(t)}, \tag{4.8}$$

$$R_{j\mathbf{q}}^{(t)} = \mathcal{N}_2(\mathbf{q}; \Delta\mathbf{L}_j^{(t)}, \mathbf{\Sigma}), \tag{4.9}$$

As can be seen, each movement is creating a Gaussian distribution (4.9), and the effect of these distributions is incrementally added through time (4.8). This way, if the target has already moved toward a direction before, the probability of that direction would always be higher than other directions towards which the target has never moved.

We can observe that the probabilistic exponential average method and the multimodal method, both make an exponential average between the observed displacements. In the case of probabilistic exponential average the average is taken between the displacements, while in the multimodal the average is taken between the probability distributions used to represent each displacement.

4.3.6 Unimodal Prediction

So far, we have assumed that the covariance matrix is constant for all the movements. Another approach is to estimate the mean and covariance matrix of the next movement using the previous movements. In this approach, all the previous displacements are weighted and the parameters are calculated using the maximum likelihood estimation. We call this approach the unimodal approach. More precisely, assuming that $\theta(t+1) = (\widehat{\Delta\mathbf{L}}_j^{(t+1)}, \mathbf{\Sigma}_j(t+1))$ are the parameters of the bivariate normal distribution that we want to estimate, we have:

$$\widehat{\theta}(t+1) = \underset{\theta(t+1)}{\operatorname{argmax}} \prod_{s=t_j+1}^{t} \mathcal{N}_2(\eta^{t-s}\Delta\mathbf{L}_j^{(s)} | \theta(t+1)), \tag{4.10}$$

$$P_{j\mathbf{q}}^{(t+1)} = \mathcal{N}_2(\mathbf{q}; \widehat{\theta}(t+1)). \tag{4.11}$$

Note that the estimation is done in polar coordinate systems. In other words, the distance r and angle θ between target position $\mathbf{L}_j^{(t)}$ and location $\mathbf{q}$ are used for calculations in this space. The reason is that, it is usually assumed that the target's movement consists of a direction and a step size. These parameters are more easily represented in a polar coordinate system than in a cartesian system.

4.3.7 Kalman Filter Prediction

Another possible approach for next movement prediction is the classical Kalman Filter method. In this approach the mean and covariance matrix is updated at each time step using the following formulas:

Correction step

$$\mathbf{K} = \mathbf{\Sigma}_{j-}(t)(\mathbf{\Sigma}_{j-}(t) + \mathbf{\Sigma}_1)^{-1}, \tag{4.12}$$

$$\widehat{\Delta\mathbf{L}_j}^{(t)} = \Delta\mathbf{L}_{j-}^{(t)} + \mathbf{K}(\Delta\mathbf{L}_j^{(t)} - \Delta\mathbf{L}_{j-}^{(t)}), \tag{4.13}$$

$$\widehat{\mathbf{\Sigma}_j}(t) = (\mathbf{I} - \mathbf{K})\Delta\mathbf{L}_{j-}^{(t)}, \tag{4.14}$$

Prediction step

$$\Delta\mathbf{L}_{j-}^{(t+1)} = \widehat{\Delta\mathbf{L}_j}^{(t)}, \tag{4.15}$$

$$\mathbf{\Sigma}_{j-}(t+1) = \widehat{\mathbf{\Sigma}_j}(t) + \mathbf{\Sigma}_2. \tag{4.16}$$

In the correction step, our prediction for the future movement of the target is combined with the new measurement (the recent displacement of the sensor) that we observe. Within this step $\mathbf{\Sigma}_1$ represents the noise we have in our measurements. In our experiments, we assumed that we observe the movements of the targets with infinite accuracy, therefore we have $\mathbf{\Sigma}_1 = \mathbf{0}$. $\mathbf{K}$ is called the Kalman gain, and measures the relative certainty between the new measurement and the current estimate. In the prediction step (as it names suggests), we expand our estimation into the next time step. In this step $\mathbf{\Sigma}_2 = \sigma_p^2\mathbf{I}$ is the process noise. Adding all the mentioned modifications our formulas boils down to:

$$\Delta\mathbf{L}_{j-}^{(t+1)} = \Delta\mathbf{L}_j^{(t)}, \tag{4.17}$$

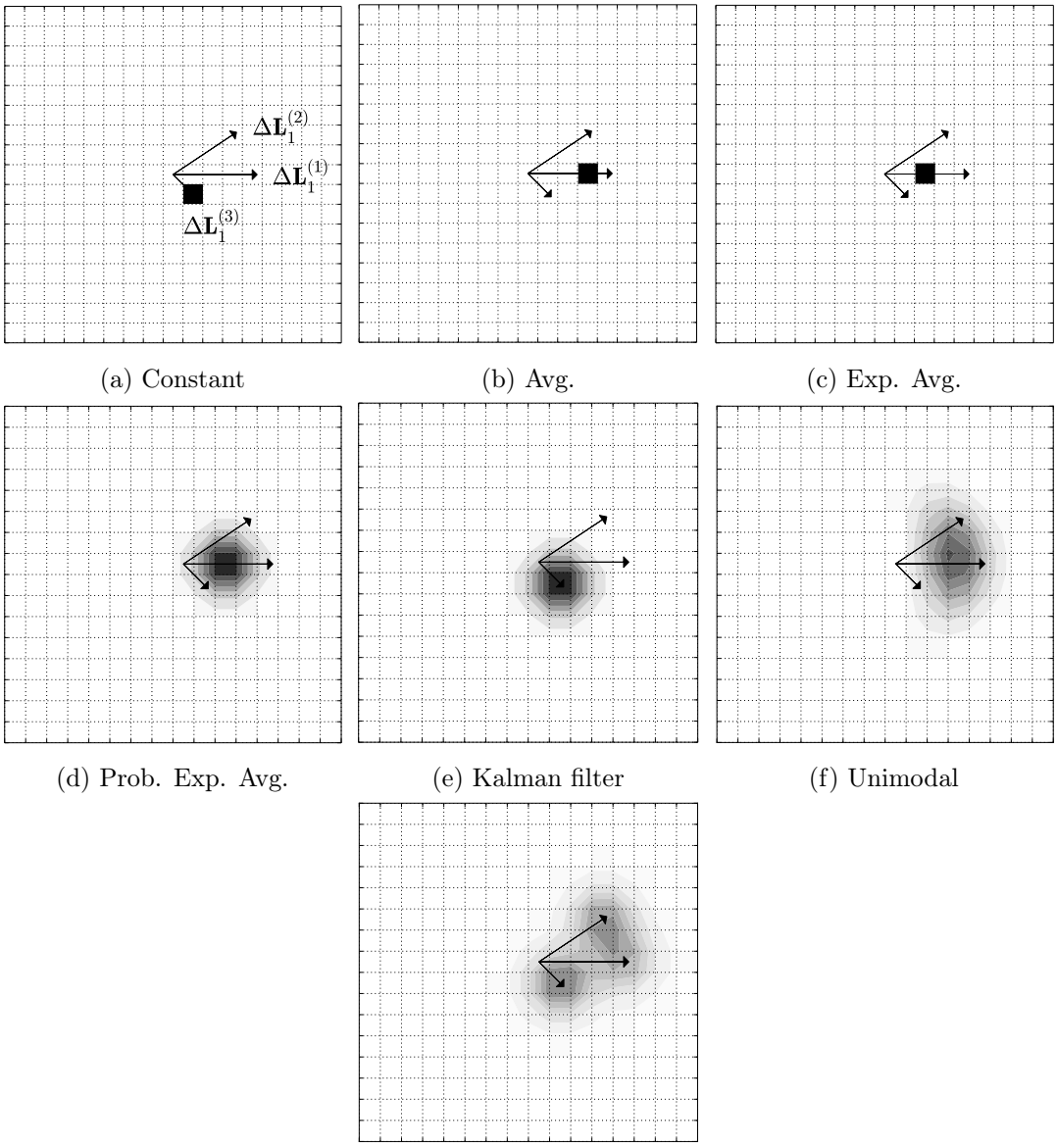
$$\mathbf{\Sigma}_{j-}(t+1) = \mathbf{\Sigma}_2, \tag{4.18}$$

$$\mathcal{P}_{j\mathbf{q}}^{(t)} = \mathcal{N}_2(\mathbf{q}; \Delta\mathbf{L}_{j-}^{(t+1)}, \mathbf{\Sigma}_{j-}(t+1)), \tag{4.19}$$

This way the Kalman Filter can be viewed as a probabilistic exponential average method where all the weight is given to the latest movement. In other words, the prediction of the Kalman Filter is a Gaussian distribution centred on the recent movement. All these prediction methods are presented in Fig. 4.1.

4.3.8 Shortcoming of Short Term Prediction Methods

As mentioned in the beginning of the section, in short term prediction methods, we restrict ourselves for prediction of one step into the future. The reason is that the uncertainty (variance) of the prediction increases linearly with the number of steps of prediction into the future.

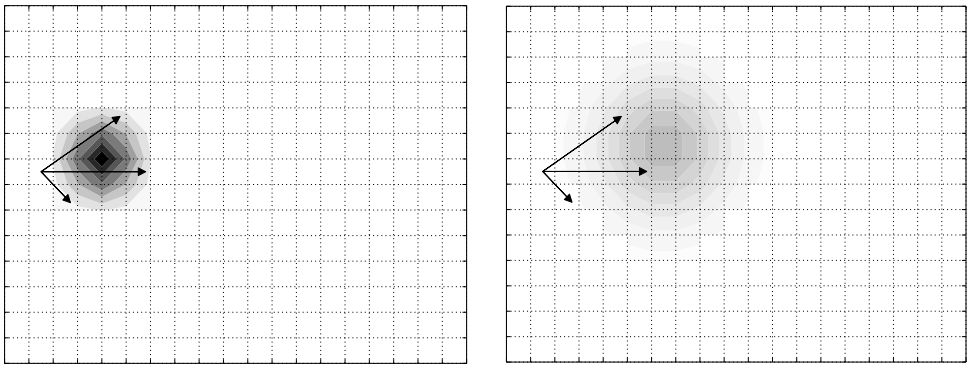


(a) Constant (b) Avg. (c) Exp. Avg.

(d) Prob. Exp. Avg. (e) Kalman filter (f) Unimodal

(g) Multimodal

Figure 4.1: Assume that target $\mathbf{T}_1$ was detected in the first time step $t_1 = 0$, and currently we are at $t = 3$. Therefore, we have the history of target $\mathbf{T}_1$ movements for the past three time steps $\Delta\mathbf{L}_1 = \{\Delta\mathbf{L}_1^{(1)}, \Delta\mathbf{L}_1^{(2)}, \Delta\mathbf{L}_1^{(3)}\}$. The goal is to predict the next movement. The probability of next movement is shown for different prediction methods.



(a) One step prediction (b) Two steps prediction

Figure 4.2: Problem of the probabilistic short term methods for long prediction steps.

For example in Fig. 4.2 we have shown the prediction of the probabilistic exponential average method for one and two steps in the future. As can be seen, even increasing the prediction step from one to two, increases the prediction uncertainty to a level that the prediction is much less usable. Therefore, it is clear that the short term prediction methods cannot be used for arbitrary prediction steps, and that is the reason we look further into the long term prediction methods in the next section.

4.4 Long Term Trajectory Prediction

In the short term prediction methods explained in the previous section, only the previous displacements of each target is used to predict its future location. As mentioned before, the main limitation of the short term prediction methods is that the prediction step is limited to one time step for these methods.

In this section we cover another group of methods proposed for the trajectory prediction problem, namely the long term prediction methods. In these methods, the trajectory of other targets who traversed the same environment is used to predict the trajectory of a new target. The information obtained from other targets help the system to predict the trajectory of new targets for longer prediction steps.

The point that should be clarified more is the reason for longer prediction steps. In other words, why does the system needs to predict the location of the targets longer into the future. For the response, notice the time requirement for different steps of any temporal optimization algorithm:

1. The prediction algorithm needs some time to predict the future location of each target based on its current locations (and possibly the history of previous targets).

2. When the prediction method has made its estimations, the sensor control method uses

these estimations to calculate the best parameter setting for each sensor and then move the direction of sensors.

Therefore there is a limitation between consecutive states of the system. Specifically, assume that at some time t, the prediction step takes s_1 time, and the sensor control step takes s_2 time. Having this, the system should make the prediction and optimization for time $t+s_1+s_2$. It is because each step depends on the result from the previous step, and all the steps should be done in sequence. We combine all these constraints and at each time t we make the prediction, optimization and movement for time $t+s$ where $s = s_1+s_2$. This parameter should be adjusted appropriately for different choices of prediction and sensor control algorithms. In other words, performing the sensor control at each time step (i.e. $s = 1$), which was assumed in the short term prediction methods, is not plausible in some real settings.

Target coverage problem is an example of a scheduling problem. If the coverage of the sensors is planned based on the current location of the targets (or their location in near future as in the short term prediction methods), the overall coverage of the network over the targets might converge to a suboptimal solution in long term. That is, a specific setting of the sensors might give good coverage in the near future while the long term coverage of the setting is inferior to another setting which gives lower coverage in short term but better results in long term. Predicting the location of the targets longer into the future will provide more information for the control mechanism of the sensors, and helps the algorithm to avoid locally optimal solutions. This is another reason which favours the application of long term prediction methods.

In the long term prediction methods, we use the history of the displacements of other targets who traversed the same environment. More precisely, assume that when a new target $\mathbf{T}_j$ arrives in the environment at time t_j, we have a dataset $\mathbb{T}$ of $j-1$ previously observed target trajectories in the environment, $\mathbb{T} = \{\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{j-1}\}$. The goal is to use the history $\mathbb{T}$ to predict the future location of $\mathbf{T}_j$ as it moves in the environment.

4.4.1 Similarity Function

We propose to do the prediction using the similarity measure between the trajectories. The principal assumption is that the similarity between the previous locations and displacements of two trajectories is directly related to the closeness of their future locations. The main problem is how to define the similarity between two trajectories.

More formally, at a time step t, we wish to measure the similarity $\mathrm{S}(\mathbf{T}_j^{(t)}, \mathbf{T}_m)$ between the current state $\mathbf{T}_j^{(t)}$ of an on-going trajectory and the whole history of another trajectory $\mathbf{T}_m \in \mathbb{T}$. Having this, we form a four dimensional space, where the state of a trajectory at one time step is represented by a point in this space, consisting of its current location $\mathbf{L}_j^{(t)} =$

$(x_j^{(t)}, y_j^{(t)})$ and its most recent displacement $\Delta\mathbf{L}_j^{(t)} = (\Delta x_j^{(t)}, \Delta y_j^{(t)})$. As a result we have $\mathbf{T}_j^{(t)} = \{x_j^{(t)}, y_j^{(t)}, \Delta x_j^{(t)}, \Delta y_j^{(t)}\}$. Note that the displacement was added to the state in order to differentiate trajectories that pass through a given location in the environment, but with different directions or velocities. The prediction algorithms uses the dataset $\mathbb{T}$ to get a better estimate of the trajectory of the newly arrived targets.

4.4.2 Kernel Density Estimation (KDE)

In the mentioned four dimensional space we estimate the density of trajectory $\mathbf{T}_m$, and we define the similarity between the state of trajectory $\mathbf{T}_j$ at time t, as its value within the density estimated for the trajectory $\mathbf{T}_m$. More formally we have:

$$S(\mathbf{T}_j^{(t)}, \mathbf{T}_m) = \widehat{F}_{\mathbf{T}_m}(\mathbf{T}_j^{(t)}), \tag{4.20}$$

where $S(.,.)$ is the similarity function, and $\widehat{F}_{\mathbf{T}_m}(\mathbf{T}_j^{(t)})$ is the density estimated for trajectory $\mathbf{T}_m$ evaluated at the point $\mathbf{T}_j^{(t)}$. For density estimation, we use the nonparametric multivariate kernel method. More precisely we have:

$$\widehat{F}_{\mathbf{T}_m}(\mathbf{z}) = \frac{1}{n_m} \sum_{t=1}^{n_m} \mathbf{K}(\mathbf{T}_m^{(t)}, \mathbf{z}, \mathbf{h}_m), \tag{4.21}$$

where n_m is the total number of displacements of target $\mathbf{T}_m$, and $\mathbf{h}_m = \{h_1, h_2, h_3, h_4\}$ are the bandwidth parameters for trajectory $\mathbf{T}_m$ along the four mentioned dimensions. These parameters define the length-scale of the function for location and displacement, respectively. Informally speaking, h_1, and h_2 defines how far two locations should be in the environment to be considered far (along the x and y dimensions), and h_3, and h_4 defines how much the directions of the two displacements should be diverged before they are considered different (along the Δx and Δy dimensions).

As kernel function $\mathbf{K}(.,.,.)$, we use the Gaussian kernel defined as:

$$\mathbf{K}(\mathbf{z}, \mathbf{z}', \mathbf{h}) = \prod_{d=1}^{4} \frac{1}{h_d} K\left(\frac{z_d - z'_d}{h_d}\right), \tag{4.22}$$

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right). \tag{4.23}$$

4.4.2.1 The Problem of Missing Observations

The mentioned similarity function is defined in four dimensions and therefore difficult to visualize. As a result we present the behaviour of the similarity function in a one dimensional space (Fig. 4.3). In this figure, there are two consecutive observations at positions 1 and 5, and we observe that the similarity is higher close to those points. In the figure, the dotted lines represent the underlying normal distributions we mapped over each observation.

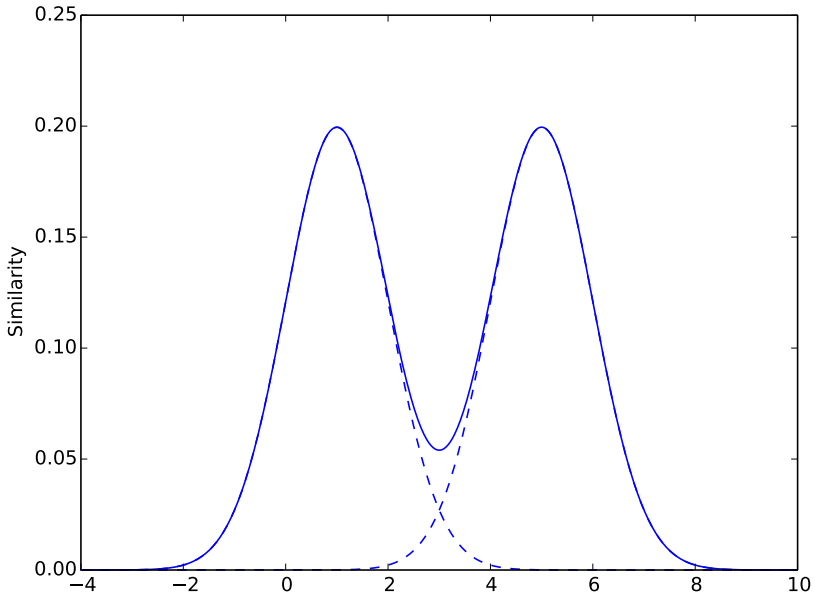


Figure 4.3: The behaviour of similarity function in one dimension.

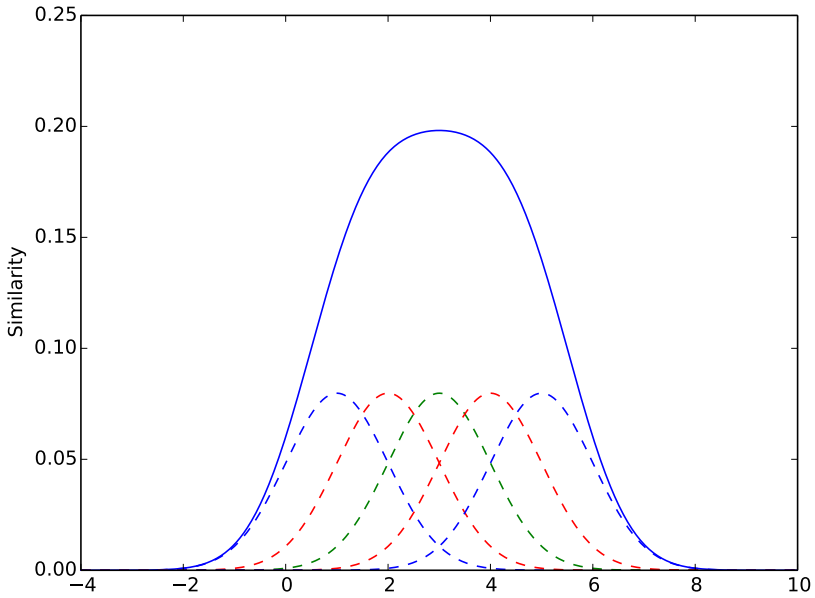


Figure 4.4: The behaviour of similarity function with unknown intermediate observations and fixed bandwidth parameters.

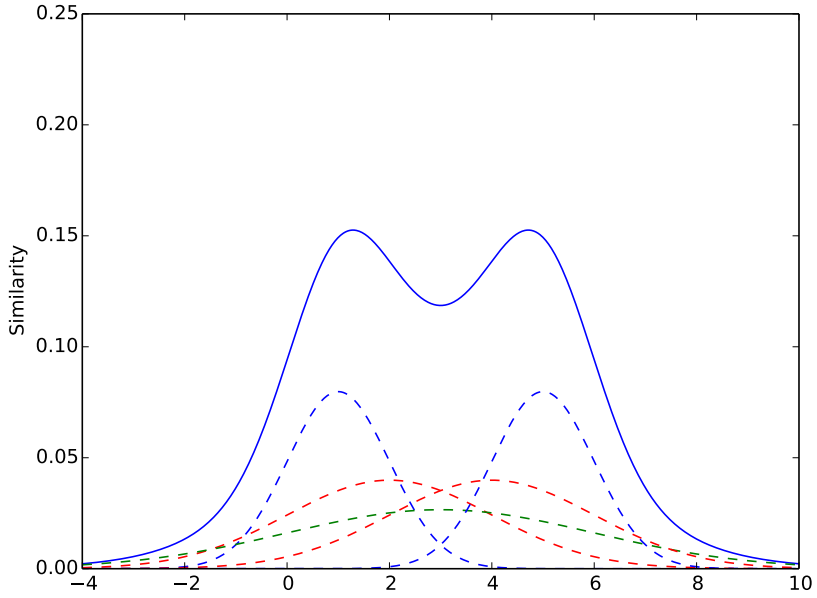


Figure 4.5: The behaviour of similarity function with unknown intermediate observations and variable bandwidth parameters.

The assumption behind the similarity function is that the trajectory of the history target $\mathbf{T}_m$, has been observed at all time steps. As we saw in the previous section, this is not the case in most real datasets. In reality the time step difference between consecutive observations of the same trajectory might be different. For example, assume that for the data presented in Fig. 4.3, the time difference between the two observations is four time steps (instead of one). The current model is not compatible with this assumption, and therefore should be modified. One possibility is to evenly divide the space between the two observations and add each interpolated point as an observation [63]. In this model all the newly created observations would have equal bandwidth parameters $\mathbf{h}$. This model is presented in Fig. 4.4.

The problem with this approach is that it assumes that the intermediate observations (which we didn't actually observe), have the same bandwidth parameter as the actual observations we made. For the example given in Fig. 4.4, the similarity at point 3 is higher than that of point 5, while we had an observation at point 5, but none at point 3.

For that reason, we propose a model in which the bandwidth parameter of the interpolated observations increase linearly as the interpolated point gets further away from an actual observation. Fig. 4.5 represents our model for the example given before. As can be seen, the bandwidth parameter is the same for the points where we had an observations (points 1 and 5), and the bandwidth parameter increases as we get away from the observations. In the same way, we modify the similarity equation (Eq. 4.21), so that the bandwidth parameter is a function of the observation, and therefore not fixed for all points. The modified version of

the similarity equation (Eq. 4.21) is as follows:

$$\widehat{F}_{\mathbf{T}_m}(\mathbf{z}) = \frac{1}{n_m} \sum_{t=1}^{n_m} \mathbf{K}(\mathbf{T}_m^{(t)}, \mathbf{z}, \mathbf{h}_m^{(t)}). \tag{4.24}$$

where in the new formula the bandwidth parameter $\mathbf{h}_m^{(t)}$ is defined for each trajectory m at each time step t.

4.4.2.2 Parameter Estimation

The only parameters of the KDE method that should be decided are the bandwidth parameters. As previously mentioned, the bandwidth parameters define the distance at which two points are considered close. We calculate the bandwidth parameter using the maximum likelihood leave-one-out cross-validation method:

$$\widehat{F}_{\mathbf{T}_m - i}(\mathbf{T}_m) = \frac{1}{n_m - 1} \sum_{t=1, t\neq i}^{n_m} \mathbf{K}(\mathbf{T}_m^{(t)}, \mathbf{T}_m^{(i)}, \mathbf{h}_m). \tag{4.25}$$

In other words, in order to evaluate a parameter $\mathbf{h}_m$ for trajectory $\mathbf{T}_m$, only one of the displacements of the trajectory ($\mathbf{T}_m^{(i)}$) is removed from the summation and the likelihood of the similarity function is evaluated at that point. For a given bandwidth parameter, the process is repeated for all displacements in the trajectory. Therefore the final process is:

$$\prod_{i=1}^{n_m} \widehat{F}_{\mathbf{T}_m - i}(\mathbf{T}_m) = \frac{1}{(n_m - 1)^{n_m}} \prod_{i=1}^{n_m} \sum_{t=1, t\neq i}^{n_m} \mathbf{K}(\mathbf{T}_m^{(t)}, \mathbf{T}_m^{(i)}, \mathbf{h}_m). \tag{4.26}$$

In the given formula the multiplicands are usually small and the final product leads to underflow. Therefore it is more convenient to maximize the log of the likelihood:

$$\sum_{i=1}^{n_m} \log \left[\widehat{F}_{\mathbf{T}_m - i}(\mathbf{T}_m) \right] = -n_m \log (n_m - 1) + \sum_{i=1}^{n_m} \log \left[\sum_{t=1, t\neq i}^{n_m} \mathbf{K}(\mathbf{T}_m^{(t)}, \mathbf{T}_m^{(i)}, \mathbf{h}_m) \right]. \tag{4.27}$$

Taking the derivative of the given formula with respect to the bandwidth parameters leads to complex equations, which cannot be solved in a closed form. Therefore, in our experiments, we adjusted the parameters of the method by a linear search conducted within the range $[1, 20]$ with the precision of 0.5, and the best value was chosen as the value of the bandwidth parameter for each trajectory.

Notice the mentioned formula defines the bandwidth parameter for the observed points. For each interpolated point, we define a level property $L(\mathbf{T}_j^{(t)})$, which measures the minimum distance between the mentioned point and the closest observed point. For the example given

in Fig. 4.4, points 2 and 4 have level 2, and point 3 has level 3. The observed points (1, and 5) both have level 1. Using this formula the bandwidth parameter is defined as:

$$\mathbf{h}_j^{(t)} = L(\mathbf{T}_j^{(t)}) \times \mathbf{h}_j, \tag{4.28}$$

where $\mathbf{h}_j$ is the base bandwidth parameter calculated using the method proposed in Eq. 4.27.

Next, we will explain two other similarity based prediction methods, which we will later use to evaluate and compare the performance of the KDE method.

4.4.3 Longest common subsequence (LCSS)

The proposed similarity measure given in Eq. 4.20 is compared with the well-known similarity measure called Longest Common Subsequence (LCSS) algorithm [77]. It consists in finding the longest similar subsequence between two trajectories using dynamic programming. More precisely, for two trajectories that end with states $\mathbf{T}_i^{(x)}$ and $\mathbf{T}_j^{(y)}$, the recursive $LCSS$ function is defined as:

$$LCSS_{\epsilon,\delta}(\mathbf{T}_i^{(x)}, \mathbf{T}_j^{(y)}) = \begin{cases} 0 & \text{if } x < 1 \text{ or } y < 1 \\ 1 + LCSS_{\epsilon,\delta}(\mathbf{T}_i^{(x-1)}, \mathbf{T}_j^{(y-1)})) & \\ \qquad \text{if } \|\mathbf{L}_i^{(x)} - \mathbf{L}_j^{(y)}\|_2 < \epsilon \text{ and } |x-y| < \delta & \\ \max(LCSS_{\epsilon,\delta}(\mathbf{T}_i^{(x)}, \mathbf{T}_j^{(y-1)}), LCSS_{\epsilon,\delta}(\mathbf{T}_i^{(x-1)}, \mathbf{T}_j^{(y)})) & \\ \qquad \text{otherwise} & \end{cases}, \tag{4.29}$$

where ϵ defines the maximum acceptable distances between corresponding target positions, and δ is the maximum allowable time warp. From this definition, the similarity between two trajectories is defined as:

$$S(\mathbf{T}_i, \mathbf{T}_j) = \frac{LCSS_{\epsilon,\delta}(\mathbf{T}_i^{(n_i)}, \mathbf{T}_j^{(n_j)})}{\min(n_i, n_j)}. \tag{4.30}$$

The hyper-parameters δ and ϵ of the LCSS method were optimized by trial-and-error through a grid search. The value of the ϵ parameter was set equal to one. The δ parameter was defined as a function of the length of the two trajectories that the LCSS algorithm was applied to. The value for the δ parameter was defined as 20% of the maximum length of the two trajectories, which is in line with the suggestion made by Vlachos et al. [77].

4.4.4 Principal Component Analysis Based Similarity

We also compared our proposal with a second approach to evaluate the similarity between different trajectories, using the Euclidean distance between the PCA coefficients of the trajectories [12]. For this purpose, all trajectories are first resampled to the median size of the

trajectories in the dataset, in order to resize trajectories to the same length. Then, a Principal Component Analysis (PCA) is applied to these trajectories (PCA is applied separately to x and y coordinates of each trajectory). The number of coefficients kept from the PCA was selected in order to retain at least 95 % of the variance (denoted by K_x and K_y for each dataset). From this, the following similarity measure is devised:

$$S(\mathbf{T}_i, \mathbf{T}_j) = \frac{1}{\sum_{k=1}^{K_x}(\gamma_x^k)^2 + \sum_{k=1}^{K_y}(\gamma_y^k)^2 + 1}, \tag{4.31}$$

where γ_x^k and γ_y^k are the distance between the k-th PCA coefficient of the two trajectories along the x and y axes.

In our experiments, we observed that matching a full trajectory $\mathbf{T}_i$ with part of another trajectory $\mathbf{T}_j$ using the PCA method produces poor results. Therefore, in Eq. 4.31, we evaluated the similarity between the sub-trajectory of the two trajectories. In other words, assuming that the test target $\mathbf{T}_j$ is at the t-th displacement; when computing the similarity between this target and another target $\mathbf{T}_i$, we consider only the first t displacements of the two trajectories for the PCA calculation.

4.4.5 Prediction using Similarity Function

Once the similarities have been calculated using any of the methods presented (KDE, PCA, or LCSS), at each time step, the similarity function behaves as a weighting parameter that can be used to predict the future locations of target $\mathbf{T}_j^{(t)}$ as a weighted sum of the trajectory of the previous targets. More precisely we have:

$$o_{ji}^{(t)} = \frac{S(\mathbf{T}_j^{(t)}, \mathbf{T}_i)}{\sum_{\mathbf{T}_m \in \mathbb{T}} S(\mathbf{T}_j^{(t)}, \mathbf{T}_m)}, \tag{4.32}$$

where $o_{ji}^{(t)}$ is the normalized similarity between the displacement of target $\mathbf{T}_j$ at time t and the whole trajectory $\mathbf{T}_i \in \mathbb{T}$. Using this formulation, the future location of a target is a discrete probability distribution over environment Ξ, where the probability of the target associated with $\mathbf{T}_j$ being at each location $\mathbf{q} \in \Xi$ after s time steps from current time t is defined as:

$$\mathcal{P}_{j\mathbf{q}}^{(t+s)} = \sum_{\mathbf{T}_m \in \mathbb{T}} o_{jm}^{(t)} \mathbf{1}(\mathbf{T}_m^{(\upsilon^*+s)}, \mathbf{q}), \tag{4.33}$$

where function $\mathbf{1}(\mathbf{T}_m^{(\upsilon^*+s)}, \mathbf{q})$ returns one if the target that generated $\mathbf{T}_m$ was at location $\mathbf{q}$ at time $\upsilon^* + s$, and zero otherwise.

In order to predict the next locations of trajectory $\mathbf{T}_j^{(t)}$, at each time step, among all of the displacements $\mathbf{T}_m^{(\upsilon)}$ of a sample trajectory $\mathbf{T}_m$, the displacement which is most similar with

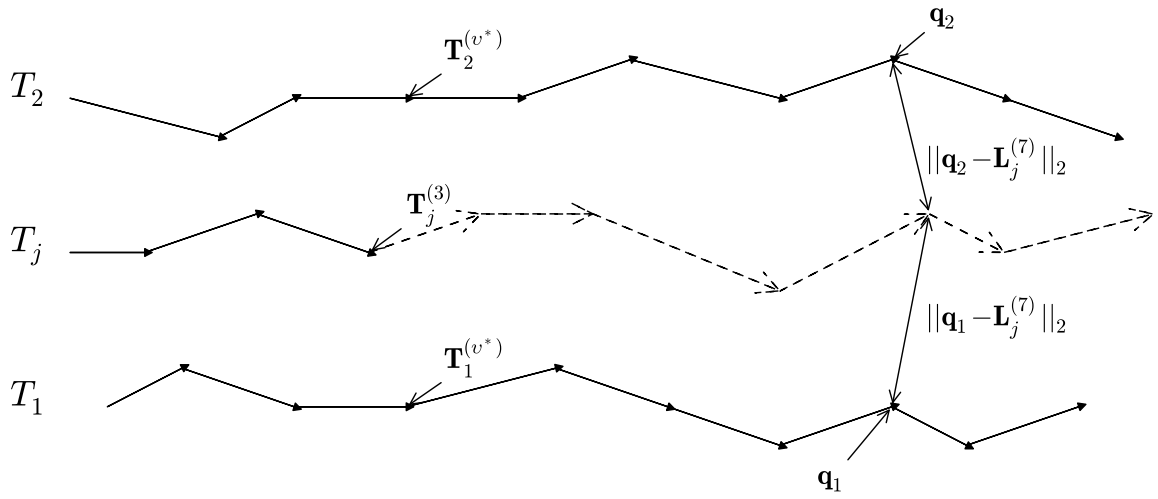


Figure 4.6: Long term target trajectory prediction process.

the current displacement $\mathbf{T}_j^{(t)}$ of target $\mathbf{T}_j$ should be found. This most similar displacement occurs at time step:

$$v^* = \underset{v}{\operatorname{argmax}}\, S(\mathbf{T}_m^{(v)}, \mathbf{T}_j^{(t)}). \tag{4.34}$$

The whole process of trajectory prediction and performance evaluation is shown for a simple example in Fig. 4.6. As before, $\mathbf{T}_j$ is the test trajectory for which we want to predict future locations, and we have the history of two reference trajectories $\mathbf{T}_1$ and $\mathbf{T}_2$. We will use the reference trajectories for predicting the location of the test trajectory. Target $\mathbf{T}_j$ is currently at the third time step, and we want to know its location after four time steps, $s = 4$.

First, the similarity between the last displacement of the test trajectory ($\mathbf{T}_j^{(3)}$) and the reference trajectories is calculated. Let us assume that $S(\mathbf{T}_j^{(3)}, \mathbf{T}_1) = 0.2$, and that $S(\mathbf{T}_j^{(3)}, \mathbf{T}_2) = 0.3$. Therefore (using Eq. 4.32) we have $o_{j1}^{(3)} = 0.4$, and $o_{j2}^{(3)} = 0.6$. Next, we use Eq. 4.34 to find the displacements in the reference trajectories that maximize similarity with the current displacement of the test trajectory ($\mathbf{T}_1^{(v^*)}$, $\mathbf{T}_2^{(v^*)}$). A non-zero prediction probability is assigned to the two reference trajectories, ($\mathcal{P}_{j\mathbf{q}_1}^{(7)}$, $\mathcal{P}_{j\mathbf{q}_2}^{(7)}$), which are used to predict the future locations of the test trajectory ($\mathbf{T}_j^{(7)}$).

To evaluate the performance, the distance between our prediction and the actual location of the test trajectory is calculated ($\|\mathbf{q}_1 - \mathbf{L}_j^{(7)}\|_2$,$\|\mathbf{q}_2 - \mathbf{L}_j^{(7)}\|_2$). Having this, the performance of our prediction would be $E_j^{(7)} = 0.4 \times \|\mathbf{q}_1 - \mathbf{L}_j^{(7)}\|_2 + 0.6 \times \|\mathbf{q}_2 - \mathbf{L}_j^{(7)}\|_2$.

4.5 Experiments

In this section we focus of evaluating the performance of the different trajectory prediction methods presented in sections 4.3 and 4.4. For that purpose, three different experimental settings are tested, one with simulated trajectories we have designed and two configurations based on datasets of real targets tracked in 2D.

4.5.1 Simulated Dataset

In order to make the experiments for the trajectory prediction, we need an environment in which the targets displace and make the trajectories. Next, we explain the environment and the targets we simulated to perform the experiments.

4.5.1.1 Map

For the experiment we used a map of Université Laval campus, in Québec, Canada. The map is a larger section of the map presented in section 3.5.2. The new map has 300 rows and 300 columns, with a resolution of one meter per dimension.

4.5.1.2 Targets

In our experiments each target is a simulation of a pedestrian walking on the campus. The target enters the environment from one gate of a building, walks inside the environment and exits from another gate (see Fig. 4.7). In this figure, the blue colour represents the buildings, green represents the street and parkings, and red is the ground. The trajectory of each target is generated based on the current location and the temporary goal locations defined for each target. The temporary goal is generated based on the shortest path between the initial gate and the final gate on the pedestrian path of the campus. Here, the pedestrian path is represented as a graph with intersections on the path being the vertices of the graph and the paths themselves being the edges. Each intersection on the shortest path between the two gates can be a temporary goal for the target.

At each time step, the next location of a target is randomly chosen from a distribution which itself is produced by the multiplication of two other distributions, namely the beta distribution and the Gaussian distribution. More precisely, at each time step, a Gaussian distribution is applied on the angle between the target's location and its temporary goal and a beta distribution is applied on the distance to determine the step size. The step size is multiplied by the maximum step size v, to get the step size at each time step. The multiplication between these two distributions is normalized to sum up to one and then used to determine the probability of each location in the environment to be selected as the next location of the target. Beta distribution has two parameters α and β and the Gaussian distribution has the parameters μ and σ_a. The values for trajectory generation parameters are summarized in

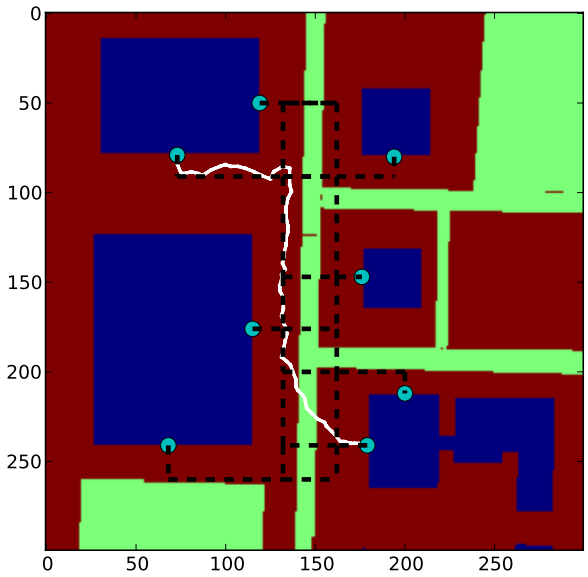


Figure 4.7: There are eight gates on the campus map. Each gate is represented by a cyan circle. Targets can enter from any of the eight gates, walk around the campus and exit the environment through another gate. The trajectory of a sample target is also shown using the white line. The pedestrian paths inside the campus are shown using the dashed black lines. The target follows the path to move between gates.

Table 4.1. For our experiments we generated 1000 targets. Given that the maximum step size v is set to 5 for our experiments here, and assuming that pedestrians walk with the speed of 1.4 m/s, each time step should be roughly 3.5 seconds.

Table 4.1: The parameter values for the trajectory generation in simulated dataset.

Parameter	Value	Parameter	Value
α	2	β	2
μ	0	σ_a^2	125
v	5		

4.5.2 MIT Trajectory Dataset

The MIT Trajectory dataset [80] contains information about trajectories of 40 453 targets moving in a parking lot. The information was gathered using a single camera for five days.

In most real datasets the time difference between consecutive displacements of trajectories is not constant. This is also the case for the two real datasets (MIT and Train station) we present here. In our experiments, we used only the trajectories for which the time difference between consecutive displacements is less than 10 time steps. If the time difference is more

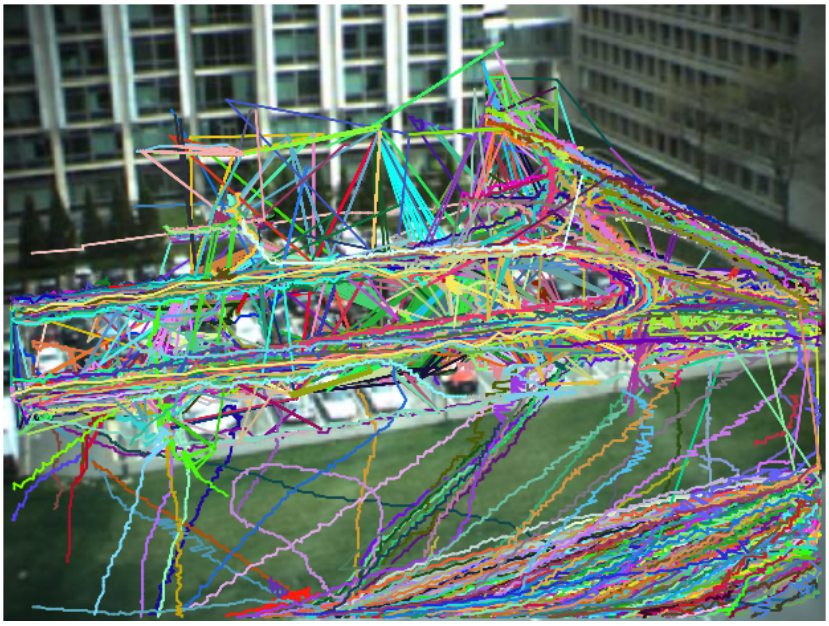


Figure 4.8: MIT Trajectory dataset [80]. Trajectories were generated from real targets moving in a parking lot within five days. A thousand of the 36 075 trajectories used for experiments are shown in the image.

than 10 time steps, we split the trajectory into two trajectories, and treat each one as an independent trajectory. The reason for this selection is that if the time difference is more than 10 time steps the two parts of the trajectory can diverge so much that it is not reasonable to consider them as a single trajectory.

As the other criteria, we only select the trajectories whose length is at least 35 steps. The reason is that in real datasets, there are some short trajectories which do not add useful information for the prediction of other trajectories, and therefore we remove them from the dataset.

Using the mentioned criteria we selected 36 075 among all the trajectories in the dataset. The parking lot and 1000 of the chosen trajectories used for our experiments is presented in Fig. 4.8.

4.5.3 Train Station Dataset

The Train Station dataset [95] contains the trajectories of 47 866 targets moving in the New York Grand Central Station. The data comes from a 30 minutes video with resolution 480×720 pixels at 24 fps. Similarly to the parking lot dataset, only those trajectories who satisfied the criteria of time difference between consecutive displacements were extracted and used for the simulations resulting in a subset of 46 741 such trajectories. The train station and 1000 of these trajectories are shown in Fig. 4.9.

Figure 4.9: Train Station dataset [95]. The trajectories were generated from the targets moving in the train station within 30 minutes. A thousand of the 46 741 trajectories used for experiments are shown in the image.

Table 4.2: The parameter values for the short term trajectory prediction methods.

Parameter	Value	Parameter	Value
η	0.8	σ_p^2	1

4.6 Results

In this section we present the performance of different short term and long term prediction methods over the three datasets mentioned in the previous section. The performance of different methods is evaluated using the expected distance criteria defined in Eq. 4.1.

4.6.1 Short Term Methods

Here, we compare the performance of seven short term prediction methods, namely, constant, average, exponential average, probabilistic exponential average, Kalman filter, unimodal, and multimodal methods. The main parameters of the short term methods are the uncertainty assigned to each displacement σ_p^2, and the learning rate η. The value of these parameters are summarized in Table 4.2.

The final results of application of each method for the three mentioned datasets are presented in Table 4.3. As we can see the exponential average method produced the best results in simulated and MIT datasets, while the constant method produced the best result in the Train station dataset. Although in the Train station dataset, the results of the exponential average

Table 4.3: Average error and standard deviation of error for different prediction methods. Error values are calculated using Eq. 4.1, smaller values are better.

	Dataset	Simulated	MIT	Train Station
Constant	Avg. err.	$3.24m$	$1.75p$	$0.78p$
	Stdev.	1.1	1.3	0.5
Average	Avg. err.	$3.74m$	$2.16p$	$0.86p$
	Stdev.	1.3	1.7	0.1
Exponential average	Avg. err.	$2.94m$	$1.57p$	$0.81p$
	Stdev.	1.8	1.2	0.1
Probabilistic exponential avg.	Avg. err.	$3.68m$	$3.16p$	$2.61p$
	Stdev.	1.2	1.5	0.1
Unimodal	Avg. err.	$4.61m$	$6.4p$	$3.48p$
	Stdev.	2.5	4.5	0.8
Kalman Filter	Avg. err.	$3.93m$	$3.41p$	$2.67p$
	Stdev.	1.3	1.8	0.1
Mutimodal	Avg. err.	$4.31m$	$3.43p$	$2.77p$
	Stdev.	1.3	1.6	0.5

is competitive with that of the constant method.

In general non-probabilistic methods (constant, average and exponential average) produced better results compared to the probabilistic methods (probabilistic exponential average, unimodal, Kalman filter, and multimodal). Looking at the results of predictions produced by each method in Fig. 4.1, we observe that the predictions of non-probabilistic methods are more concentrated than the probabilistic methods. Therefore, the reason for the superior performance of these methods with respect to the expected distance metric could be related to the centralization of the prediction. In other words, in the methods who produced more dispersed predictions even if the actual displacement lies in the section which was given a weight by the prediction method, the final result would be inferior of more concentrated methods, even if the next displacement is not completely aligned with the prediction.

We will see later in chapter 6, that probabilistic methods produce better coverage over the targets compared to non-probabilistic prediction methods. Therefore, we can conclude that the expected distance criteria, is not a very informative to compare the performance of different prediction methods once they are used in the context of temporal optimization in SNs.

4.6.2 Long Term Methods

The performance of the proposed Kernel Density Estimation (KDE) method is compared with the two other mentioned methods (PCA and LCSS). The same set of experiments that were used to test the performance of the short term methods were used to test the long term

methods. The only difference between the two experiments is that for the short term methods, the prediction horizon is one time step. For the long term methods, the prediction horizon can be arbitrarily large, as the prediction relies on the trajectory of other targets. To evaluate the performance of the mentioned long term methods, we performed two set of experiments with different prediction horizons of ($s = 5$) and ($s = 10$) time steps.

The similarity based prediction method presented here is categorized as non-parametric method. Therefore, the performance of the mentioned methods depend on the size of the history of reference trajectories used for prediction. We performed some experiments and observed that as the size of the history of reference trajectories goes beyond 1000 trajectories, the increase in size does not significantly affect the performance of the prediction method (see Figs. 4.10, 4.11). Therefore, we keep the trajectory history as a sliding window with the size of 1000. That means that at each time step, in order to predict the trajectory of a new target, only the trajectories of the previous 1000 targets are used. In this manner, the prediction time for each trajectory remains manageable for the large datasets (e.g. MIT and Train Station).

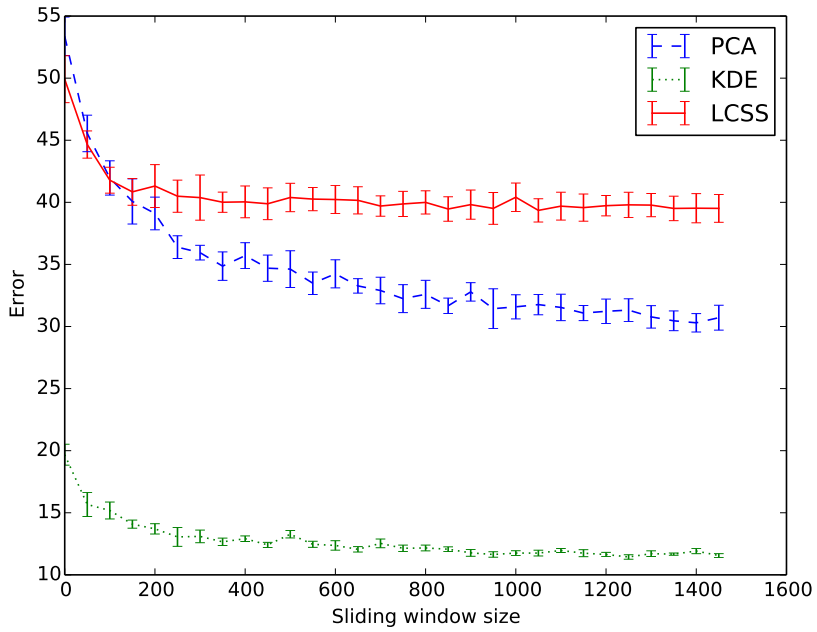


Figure 4.10: Affect of the sliding window size on the performance of different prediction methods over the MIT trajectory dataset.

The performances of the KDE, LCSS, and PCA prediction methods are reported in Table 4.4. Each method has been run 30 times, where each run uses a different order for processing of the trajectories. Therefore, for each run for KDE, LCSS, and PCA a different set of trajectories have been used for the prediction of each trajectory. Reported results are average performances over 30 runs. In this table, the average distance between the prediction made by each method and the actual location of the target is reported (as defined by Eq. 4.1).

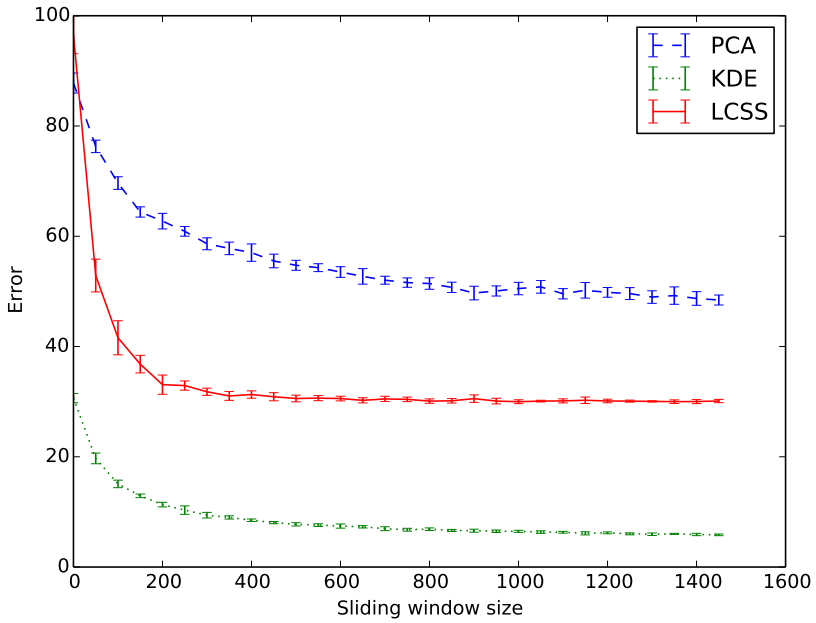


Figure 4.11: Affect of the sliding window size on the performance of different prediction methods over the Train station dataset.

It can be seen that the KDE method outperformed the LCSS and PCA methods considering the error measure on all datasets and with all simulation settings. The reason for the strong performance of the KDE method is twofold. First, the KDE method is the only method which considers both the location and the displacement in the similarity function. Therefore, the inclusion of the displacement in the similarity function can significantly change the behaviour of the similarity function in different scenarios. Second, KDE is the only probabilistic method among the three. The probabilistic foundations of the method, helps it to make smoother decisions over the similarity between two trajectories. For example, the parameters of the LCSS method (ϵ, δ) takes binary decisions for matching the subparts of two trajectories. Meaning that if the distance between two parts are less than the mentioned parameter values the two parts are similar and not otherwise. On the other hand, the similarity measure defined by the probability density function of the KDE method changes more smoothly from one to zero.

Another explanation is that KDE has a mechanism to deal with missing data points by taking into account the uncertainty (i.e., variance) over the estimated location of the target, while the two other methods rely only in the interpolated positions. That could be the reason that the performance of different methods is closer on the simulated dataset (in which there is no unobserved trajectory) compared to the two other real datasets where there are unobserved points.

In all settings, increasing the prediction horizon from ($s = 5$) to ($s = 10$) has increased the

Table 4.4: Average error and standard deviation of error for different prediction methods. Error values are calculated using Eq. 4.1, smaller values are better. Bold values represent statistically significant results according to the Wilcoxon signed-rank test (p-value=0.05) when compared pairwise with other methods.

	Dataset	Simulated	MIT	Train Station
Prediction horizon = 5				
KDE	Avg.	**11.68**m	**10.39**p	**12.45**p
	Stdev.	0.2	0.6	0.4
LCSS	Avg.	35.4m	29.84p	36.65p
	Stdev.	0.3	0.8	0.7
PCA	Avg.	23.42m	39.56p	54.29p
	Stdev.	0.5	0.6	0.6
Prediction horizon = 10				
KDE	Avg.	**33.36**m	**22.98**p	**19.65**p
	Stdev.	0.6	0.6	0.5
LCSS	Avg.	72.43m	43.57p	39.58p
	Stdev.	1.0	0.9	0.4
PCA	Avg.	50.18m	49.52p	58.53p
	Stdev.	0.9	0.5	0.7

expected error. This is reasonable given that all methods predict further into the future with less certainty. The interesting point is that for each dataset, increasing the prediction time step has roughly the same effect over the different methods (i.e., the increase of the expected error is comparable for different methods over one dataset). This suggests that the properties of the dataset (i.e. trajectories of the dataset) has a more significant impact over long term prediction accuracy than the prediction methods. For example, the simulated dataset has the largest drop in performance between the short ($s = 5$) and long ($s = 10$) prediction time steps. This could be related to the fact that the targets change direction more frequently in the simulated dataset than the other two datasets. In this dataset, targets share part of their trajectory with many other targets, unlike the other two datasets, where each target usually keeps the initial direction of its movement.

We can also observe that in the simulated dataset the PCA method outperformed the LCSS method, while in the other two datasets the performance of the LCSS method was superior. Therefore, it is clear that the characteristics of each dataset has an important effect on the final performance of each method.

4.7 Conclusion

In this chapter we have presented different methods to predict the future location of a moving object. The proposed methods are categorized under short term and long term methods. The

main difference between the two groups is that in the short term methods the only the previous displacements of each targets are used to predict its future location while in the long term prediction methods, the history of other targets who traversed the same environment are also used in the prediction process.

For the short term prediction methods we proposed two probabilistic methods namely the probabilistic exponential average and the multimodal methods. Besides, we used the maximum likelihood to optimize the parameters of the unimodal prediction method. The strong point of the multimodal approach is its capability to handle several assumptions at the same time (e.g. zig-zag movement) and to adapt to non-Gaussian movement models.

For long term prediction methods, we have presented a model to measure similarity between trajectories of targets moving inside an environment using the kernel density estimation. Besides, we presented a mechanism to adapt the bandwidth parameter of the kernel density estimation for the problem of missing observations which is a common problem in real datasets. The strong performance of the method is related to its probabilistic nature and consideration of the displacement in the similarity function.

Under each category, experiments were conducted on one simulated and two real datasets. The proposed models were compared with other commonly used methods, using the expected distance as the performance measure. In the long term prediction methods, the KDE method outperformed both of the other methods over all tested datasets and simulation settings. In the case of short term methods, the results were inferior to non-probabilistic prediction methods. As we will see later in chapter 6, the probabilistic short term prediction methods outperform the non-probabilistic methods when used to support sensor control.

Chapter 5

Sensor Control

Temporal optimization (as we consider in this thesis) consists of two main problems, the trajectory prediction and the sensor control. In trajectory prediction (which we covered in chapter 4) the goal is to predict the future location of moving targets. In this chapter we focus on the sensor control problem, for which the goal is to modify the parameters (e.g. pan, tilt, zoom) of the sensors, based on the prediction given by the trajectory prediction algorithm, so that the final network has a maximal coverage over the targets as they move within the environment. Coverage degree over the targets and the computation time of the algorithm are the two main performance criteria which should be considered for designing a sensor control algorithm.

In section 5.1 we cover other methods proposed in the literature for the sensor control problem. In section 5.2 the problem is formulated, and in section 5.3 the Greedy Sensor-wise Coverage Optimization (GSCO) algorithm is proposed for the problem. In section 5.4, we perform some experiments and compare the performance of the GSCO algorithm with another well-known optimization algorithm. Finally, section 5.5 concludes the chapter.

5.1 Sensor Control

Once the future location of the targets is predicted, it is the role of the sensor control algorithm to calculate the best parameters for each of the sensors to have a maximum coverage over the targets. The sensor control problem is closely related to the placement problem in SNs with a weighted coverage goal (as mentioned in chapter 3).

There are two differences between the placement problem and the sensor control algorithm. The main difference is the time requirements of the two algorithms. In placement problem, the running time of the algorithm can be arbitrarily long, as the placement is a one time process. In other words, once the best location and direction for sensors is determined there is no need to run the algorithm again, therefore the running time of the algorithm is less of

a critical issue. On the other hand, the frequency of the run of sensor control algorithm is much higher. Besides, the system is online, and the sensor control algorithm should return the results before the time step expires, otherwise the results are of little use. Therefore, in sensor control algorithms, the computation time becomes an important issue.

The second difference is that, in placement problem, the goal is to find the best location and direction for the sensors, so that the final configuration has maximum coverage over the environment. On the other hand, in the sensor control problem, the location of the sensors are fixed, but their directions are variable. Therefore, sensor control can be viewed as spatial coverage optimization problem for the sensors with fixed location, but variable directions over a weighted environment.

The same problem was discussed in the literature by different authors. For example, Cai et al. [18] proposed the application of the cover set algorithm to the spatial optimization of the directional sensors with fixed locations. In each iteration of the greedy method they proposed for the problem, the target location that can be covered by the minimum number of sensor directions is selected and added to the solution set. Authors also proposed a distributed version of the algorithm for large scale applications. In the distributed approach each sensor has only access to coverage information of its neighbours. The method proposed by the authors does not work with probabilistic prediction for location of the targets, while in many applications, the trajectory prediction produces different certainties for the future location of a target.

Similarly, Ai et al. proposed in [4] the maximum coverage by minimum sensors method which follows the same principles as the one proposed in [18]. The proposed centralized greedy algorithm iteratively selects a sensor and calculates the number of targets which could change state from uncovered to covered using the sensor. Next, the direction which maximizes the coverage over uncovered targets is selected. The iteration continues until all the targets are covered or all the sensors have been selected. The main problem with the proposed approach is that the coverage of sensors over the targets is binary, therefore, the algorithm is not applicable if the sensor has partial coverage over the target.

In [23], Chen et al. defined a weight for each of the targets and each direction the sensor can take, which are called the target weight and orientation weight respectively. The target weight is calculated based on the number of sensors that can cover a target. Orientation weights are defined based on the total number of targets in the environment, the number of targets the orientation can cover, and the target weight for each of the targets in the coverage range. Wang et al. [79] proposed a priority based target coverage algorithm, where the importance of targets is different for the system. A genetic algorithm was applied on the direction of sensors to find the direction which best covers all the targets.

The main drawback of both algorithms is that they do not take into consideration any limitation on the movement speed of the sensors. In other words, they assume that sensors move

from any direction to another direction in one time step. The other drawback of the proposed methods is that they only consider one variable parameter per sensor, and therefore iterate over all possible directions of the sensors. This approach is not reasonable when a sensor has two or three variable parameters, as the computation needed to iterate over all possible combinations is cumbersome.

5.1.1 Proposed Method

In this chapter, we introduce the Greedy Sensor-wise Coverage Optimization (GSCO) algorithm. The algorithm is specifically designed to find the best direction for sensors, to cover prediction of targets' location in the environment. The input to the algorithm handles different degrees of certainty for target locations, works with probabilistic coverage model of sensors, and finally accepts limitation for moving speed of sensors. Besides the computational need of the algorithm is very reasonable.

5.2 Sensor Control Problem

In temporal optimization, the goal is to cover a number of targets moving in the environment using a SN. From here on, we focus on Pan-Tilt-Zoom cameras (PTZ) (as defined in Chapter 2) as our sensors. The pan, tilt, and zoom properties of the sensors are variable through time, and this provides the capability for the sensors to change their coverage direction and to provide a maximal coverage over the moving targets. To provide this coverage, first the targets should be detected in the environment, their trajectories should be modelled and finally the parameters of the sensors should be modified so that the targets remain under coverage while they are present in the environment. In general we focus on two parts of the mentioned problem. In the previous chapter we focused on the trajectory prediction problem and in this chapter the focus is on the sensor control problem.

Assume that we have a sensor network $N = \{\mathbf{s}_i | i = 1, \ldots, n\}$ consisting of n sensors $\mathbf{s}_i$ with the pan, tilt, and zoom control parameters. We extend the previous sensors model presented in Chapter 2, in the way that each sensor is also attributed with a time parameters $t = \{1, 2, \ldots, M\}$, where M is the total simulation time. The time parameter was added, because in temporal optimization (as its name suggests), the state of the sensor is changing over time, and the time parameter models this effect. In this setting, the state of each sensor at time t is defined as $\mathbf{s}_i^{(t)} = (\mathbf{p}_i, \theta_i^{(t)}, \xi_i^{(t)}, f_i^{(t)})$. As it can be seen, the location of the sensor is fixed through time ($\mathbf{p}_i$), but the pan, tilt and zoom parameters are variable ($\theta_i^{(t)}, \xi_i^{(t)}, f_i^{(t)}$).

The goal of the network N is to optimally cover J moving targets $\mathbb{T} = \{\mathbf{T}_j | j = 1, 2, \ldots, J\}$. Each target $\mathbf{T}_j$ is first detected at time t_j in the environment, then it makes n_j displacements, until it exits the environment at time $t_j + n_j$. Therefore the trajectory of each target can be represented as: $\mathbf{T}_j = \{\mathbf{T}_j^{(t)} | t = t_j, \ldots, t_j + n_j\}$.

In the sensor control problem, the goal is to find the configuration for the variable parameters of the sensors (e.g. pan, tilt, and zoom), to have maximal coverage over the moving targets in the environment. Sensor control phase is based on the result of the trajectory prediction, which is given to this phase as a prediction map $\mathbf{R}^{(t)}$ (as defined in section 4.2).

The commands that the network applies to sensor $\mathbf{s}_i$ through time is defined by $\mathbf{\Pi_i} = \{\pi_i^{(t)} | t = 1, \ldots, M\}$, where $\pi_i^{(t)} = (\Delta_\theta, \Delta_\xi, \Delta_f)$ is the command given to sensor $\mathbf{s}_i$ at time t for updating its pan, tilt, and zoom values. The set of commands for the n sensors is summarized as $\mathbb{\Pi} = \{\mathbf{\Pi}_i | i = 1, \ldots, n\}$. Each sensor also has a set of characteristic parameters which define the maximum and minimum value of each control parameters, along with their associated speed of change:

$$\langle (\theta_{min}, \theta_{max}, v_\theta), (\xi_{min}, \xi_{max}, v_\xi), (f_{min}, f_{max}, v_f) \rangle. \tag{5.1}$$

It is clear that the given command at each time step should satisfy each sensor's movement constraints. More precisely we have as constraints:

$$|\Delta_\theta| \leq v_\theta, \quad \theta_{min} \leq \underbrace{\theta_i^{(t)} + \Delta_\theta}_{\theta_i^{(t+1)}} \leq \theta_{max}, \tag{5.2}$$

$$|\Delta_\xi| \leq v_\xi, \quad \xi_{min} \leq \underbrace{\xi_i^{(t)} + \Delta_\xi}_{\xi_i^{(t+1)}} \leq \xi_{max}, \tag{5.3}$$

$$|\Delta_f| \leq v_f, \quad f_{min} \leq \underbrace{f_i^{(t)} + \Delta_f}_{f_i^{(t+1)}} \leq f_{max}. \tag{5.4}$$

where the boundaries for each sensor was defined in Eq. 5.1.

We define the following objective function for a given command set as the coverage it provides for all of the targets through time:

$$O(\mathbb{\Pi}) = \sum_{t=1}^{M} \sum_{\mathbf{q} \in \Xi} r_{\mathbf{q}}^{(t)} C_l(N, \mathbf{q}), \tag{5.5}$$

and the goal is to find a command set $\mathbb{\Pi}^*$ which maximizes this objective function:

$$\mathbb{\Pi}^* = \underset{\mathbb{\Pi}}{\operatorname{argmax}}\ O(\mathbb{\Pi}). \tag{5.6}$$

Notice in this formula, the trajectory prediction step defines the weights assigned to different locations in the environment through the weight parameter $r_{\mathbf{q}}^{(t)}$, and the sensor control step finds the command set $\mathbb{\Pi}^*$ based on that prediction. Therefore, both the prediction and control steps, affect the result of objective function. The defined objective function provides an estimate on the actual coverage that the network provides over the targets. The actual coverage is calculated using the following formula:

$$C_g(\mathbb{\Pi}) = \frac{1}{MJ|\Xi|} \sum_{j=1}^{J} \sum_{t=1}^{M} \sum_{\mathbf{q} \in \Xi} \mathbf{1}(\mathbf{L}_j^{(t)}, \mathbf{q})\ C_l(N, \mathbf{q}), \tag{5.7}$$

where the function $\mathbf{1}(\mathbf{L}_j^{(t)}, \mathbf{q})$ returns one if target $\mathbf{T}_j$ is at location $\mathbf{q}$ at time t, and zero otherwise. The formula provides the average coverage of the network over all the targets during the whole simulation time.

5.3 Greedy Sensor-wise Coverage Optimization (GSCO)

The result of the prediction step is given to the sensor control step as a prediction map $\mathbf{R}^{(t)}$. The value of each location within the prediction map defines the importance of that location for the coverage task. There are several points which should be considered once designing the sensor control algorithm. First, the prediction map contains probabilities and not binary values as the predicted location of the targets. Therefore, the algorithm should be able to handle probabilistic predictions. Second, the sensors themselves have probabilistic coverage over the targets and it should be taken into consideration. Third, the movement of sensors is limited for each time step. These limitations put a boundary on the sensor's parameter values for the next time step. Finally, each sensor has several variable parameters and therefore a simple iteration over all possible combination of values is usually not possible.

The problem has online time requirements, therefore we are aiming to have an algorithm which has linear complexity with respect to number of sensors in the environment. So that we can get some guarantees on the running time of the algorithm in real settings.

Considering all the mentioned criteria, we propose the GSCO algorithm. The main idea behind the GSCO algorithm is that target locations which have higher prediction value, and at the same time, can be covered by fewer number of sensors should be given higher priority in the coverage task. In other words, in the GSCO algorithm, at each time step, the prediction map $\mathbf{R}^{(t)}$ is divided by the accessible coverage map $\Psi_{\mathbf{q}}^{(t)}$ (which defines the number of sensors that can cover any location in the environment), to define a weight map which determines the importance of each location for the coverage task. This weight map is later used to iteratively optimize the direction of all sensors in the environment.

The algorithm is shown in Fig. 5.1. Different steps of the algorithm are explained in more details below:

- Calculate $\psi_i^{(t)}$: For each sensor calculate the accessible coverage map $\psi_i^{(t)}$. This map holds the locations in the environment that the sensor $\mathbf{s}_i$ can effectively cover in the future. For its calculation, we take into consideration the current parameters $(\theta_i^{(t)}, \xi_i^{(t)}, f_i^{(t)})$, and the maximum speed (v_θ, v_ξ, v_f) of each sensor.

Given a sensor network N, the prediction map $\mathbf{R}^{(t)}$, and the environment Ξ as the input:
while $N \neq \emptyset$ **do**
  for all $\mathbf{s}_i \in N$ **do**
    Calculate ψ_i
    $\Psi_{\mathbf{q}} = \Psi_{\mathbf{q}} + \psi_{i\mathbf{q}}, \quad \forall \mathbf{q} \in \Xi$
  end for
  $\mathbf{U}_{\mathbf{q}} = \frac{\mathbf{R}_{\mathbf{q}}^{(t)}}{\Psi_{\mathbf{q}}}, \quad \forall \mathbf{q} \in \Xi$
  $\mathbf{s}_j = Rand(N)$
  Optimize parameters of $\mathbf{s}_j$ using $\mathbf{U}$.
  for all $\mathbf{q} \in \Xi$ **do**
    $r_{\mathbf{q}}^{(t)} = r_{\mathbf{q}}^{(t)} - (r_{\mathbf{q}}^{(t)} . C(\mathbf{s}_j^{(t)}, \mathbf{q}))$
  end for
  $\mathbf{N} = \mathbf{N} \setminus \mathbf{s}_j$
end while

Figure 5.1: The proposed GSCO algorithm for sensor control.

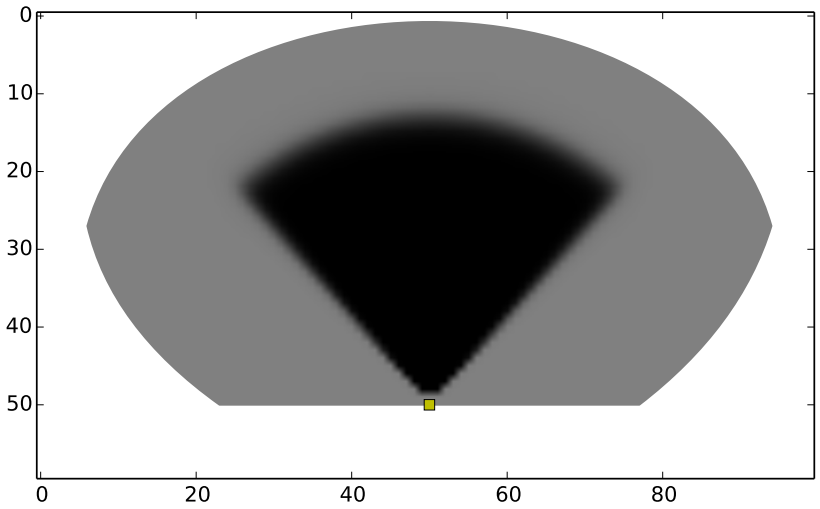


Figure 5.2: Assuming that a sensor is positioned at (50,50) heading upward, the dark shaded area shows the current coverage of the sensor and the light shaded area shows the accessible coverage map ($\psi_i^{(t)}$) of the sensor in the next time step.

The other parameter used for the calculation of $\psi_i^{(t)}$ was a threshold parameter ϕ. This threshold was defined because (theoretically based on Eq. 2.9), the coverage of a sensor over all the locations from which it is visible is non-zero, while the coverage over most of these locations is close to zero. Therefore, by using the ϕ threshold, we only consider points for which the sensor can significantly change the coverage of the points it is covering. For an example of $\psi_i^{(t)}$, look at Fig. 5.2, where we have shown the current coverage of a sensor and its accessible coverage map. More precisely, we have:

$$\psi_i^{(t)} = \{\mathbf{q} \in \Xi \mid C(\mathbf{s}_i^{(t)}, \mathbf{q}) > \phi\}. \tag{5.8}$$

- $\Psi_{\mathbf{q}}^{(t)} = \Psi_{\mathbf{q}}^{(t)} + \psi_{i\mathbf{q}}^{(t)}$: Using $\psi_i^{(t)}$, calculate the global accessible coverage map $\Psi^{(t)}$ for the environment. Each cell $\mathbf{q}$ in $\Psi^{(t)}$ has a value $\Psi_{\mathbf{q}}^{(t)}$ representing the effective accessible coverage of all sensors over this location. The $\Psi^{(t)}$ value for each location shows the potential of that location to be covered by different sensors. Notice that the value of different locations in the possible coverage map are real values (unlike previous work (e.g. as in [23, 79]) in which the values are integers). This is a natural consequence of the probabilistic coverage model we used for our sensors.

- $\mathbf{U}_{\mathbf{q}}^{(t)} = \frac{\mathbf{R}_{\mathbf{q}}^{(t)}}{\Psi_{\mathbf{q}}^{(t)}}$: Create a weight map $\mathbf{U}^{(t)}$. In the new weight map, the weight of the locations which can be covered by more than one sensor is proportionally reduced. This is the core of the algorithm, which assigns more weight to location that can be covered by less number of sensors.

- $\mathbf{s}_j = Rand(N)$: Randomly choose a sensor $\mathbf{s}_j$ from the sensors whose direction haven't been optimized yet.

- Optimize parameters of $\mathbf{s}_j$ using $\mathbf{U}^{(t)}$: The optimization is done using random sampling. Random sampling was shown to be more effective search method compared to other simple methods such as grid search [14], therefore we chose it as our optimization algorithm. For that purpose ω samples are randomly chosen from the range of possible pan, tilt and zoom factors. The parameter set which achieves the best final coverage is chosen as the final direction of the sensor. The important point is that the evaluation for different parameter sets for a sensor is based on the weight map $\mathbf{U}^{(t)}$.

- $r_{\mathbf{q}}^{(t)} = r_{\mathbf{q}}^{(t)} - (r_{\mathbf{q}}^{(t)} . C(\mathbf{s}_j^{(t)}, \mathbf{q}))$: Once the direction and zoom level of the sensor $\mathbf{s}_j$ is found, its coverage is proportionally decreased from the prediction map. For example, assume that location $\mathbf{q}$ has the value of $(r_{\mathbf{q}}^{(t)} = 0.8)$ in the prediction map. Also assume that the direction of sensor $\mathbf{s}_j$ is fixed and the coverage of the sensor over the location is $C(\mathbf{s}_j^{(t)}, \mathbf{q}) = 0.7$. Having this, the new value of the location in the prediction map becomes: $r_{\mathbf{q}}^{(t)} = 0.8 - 0.8 \times 0.7 = 0.24$.

- $\mathbf{N} = \mathbf{N} \setminus \mathbf{s}_j$: Remove the sensor $\mathbf{s}_j$ (for which the direction has been optimized) from the set of sensors.

The iteration continues until the parameters of all the sensors is determined.

5.4 Experiments and Results

In the experimental section we compare the performance of the proposed GSCO algorithm with the CMA-ES method. Notice we could not compare our method with other methods mentioned in section 5.1, given that none of the mentioned algorithms used a probabilistic

Table 5.1: The parameter values for a realistic model of a PTZ camera.

Parameter	Value	Parameter	Value
$L_h(m)$	5.37×10^{-3}	$L_v(m)$	4.04×10^{-3}
$\alpha_{d_{max}}(m)$	50	$\alpha_{d_{min}}(m)$	25
$f_{min}(m)$	4.7×10^{-3}	$f_{max}(m)$	9.4×10^{-3}
β_d,β_p and β_t	1	$\tau(m)$	1
ξ_{min}	-90^o	ξ_{max}	90^o
v_θ	$\pm 30^{o/t}$	v_ξ	$\pm 5^{o/t}$
$v_f(m/t)$	$\pm 1.33 \times 10^{-3}$	θ_{max}	360^o
θ_{min}	0^o		

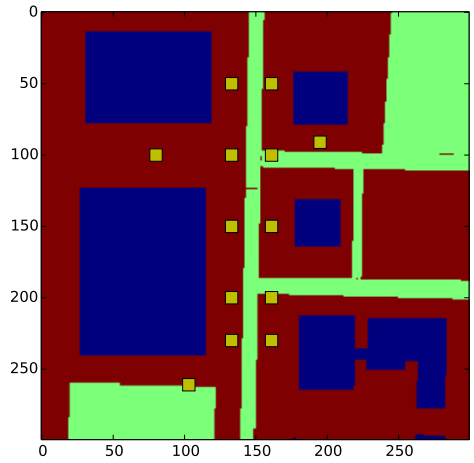


Figure 5.3: The location of sensors is represented by the yellow squares on the map.

coverage model for sensors, and therefore our sensor coverage model is not applicable in those algorithms.

To perform the experiments, we used the simulated dataset explained in section 4.5.1. Notice that the dataset lacks the information for the sensors. The position of the sensors is presented in Fig. 5.3. In this figure the sensor locations are represented by the yellow squares.

Sensors placed in the environment are the Pan-Tilt-Zoom cameras. The specification of the cameras are explained in section 2.5. For a reasonable model of a camera, we propose to use the parameters shown in Table 5.1.

5.4.1 Sensor Control

We use the probabilistic exponential average method (section 4.3.4) as the prediction method to determine the future location of targets. Then, the mentioned two optimization methods find the best commands for the sensors. For the parameters of the CMA-ES method, we used the values suggested by the authors of the method [36]. The threshold parameter of the GSCO

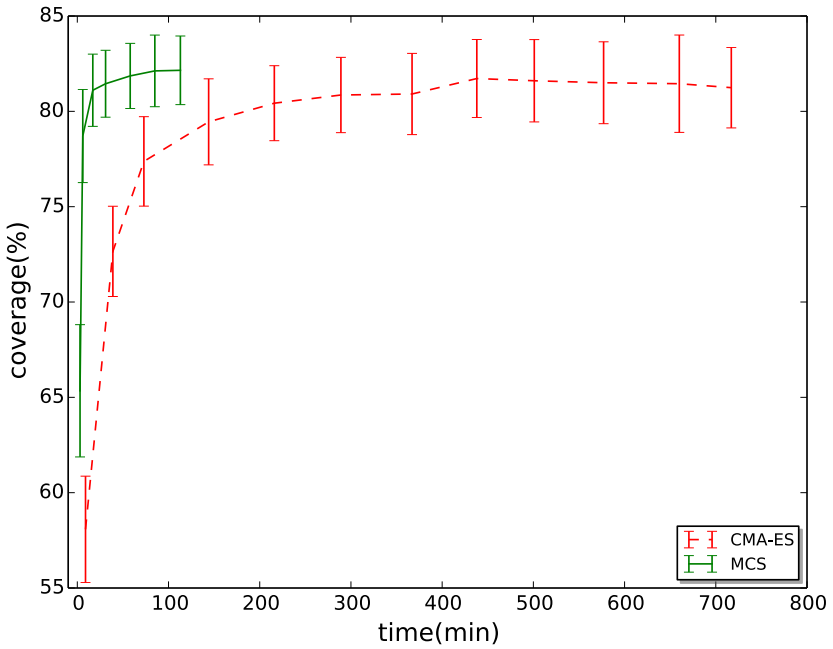


Figure 5.4: Coverage percentage for experiments with 50 targets using CMA-ES and GSCO optimization methods.

algorithm (ϕ) was set to 0.5 for the experiments presented here. The value was chosen by trial and error, and the algorithm appears quite robust to the choice of this parameter.

To test different scenarios, we produced simulations with different density of targets. Therefore, at each time step, a number of targets equal to the ratio between the total number of targets to the simulation time enter the environment. For the experiments, we created simulations with 50, 100, and 150 targets in the map, with a total simulation time of $M = 200$ time steps. We produced 30 different simulation settings with each parameter set, and performed the experiments on them. The performance measure was previously defined (Eq. 5.7) as the average coverage the network provides over all the targets through the whole simulation time.

One of the critical parameters for the CMA-ES algorithm is the maximum number of iterations that the algorithm is allowed to run for. This parameter directly affects the final performance and the running time of the algorithm. As it is a critical measure, we tested CMA-ES method with different values for the maximum number of iterations, ranging from 1 to 100. More precisely, the iteration numbers are $[1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$.

The same argument applies to the number of random samples ω taken by the GSCO algorithm. We performed several experiments, in which the ω parameter has the range of $[1, 10, 50, 100, 200, 300, 400]$. The results are presented in Figs. 5.4, 5.5, and 5.6. Each variance bar in the figures represents the result of the CMA-ES algorithm with one specific maximum number of iterations or the GSCO algorithm with a specific ω parameter. In these experi-

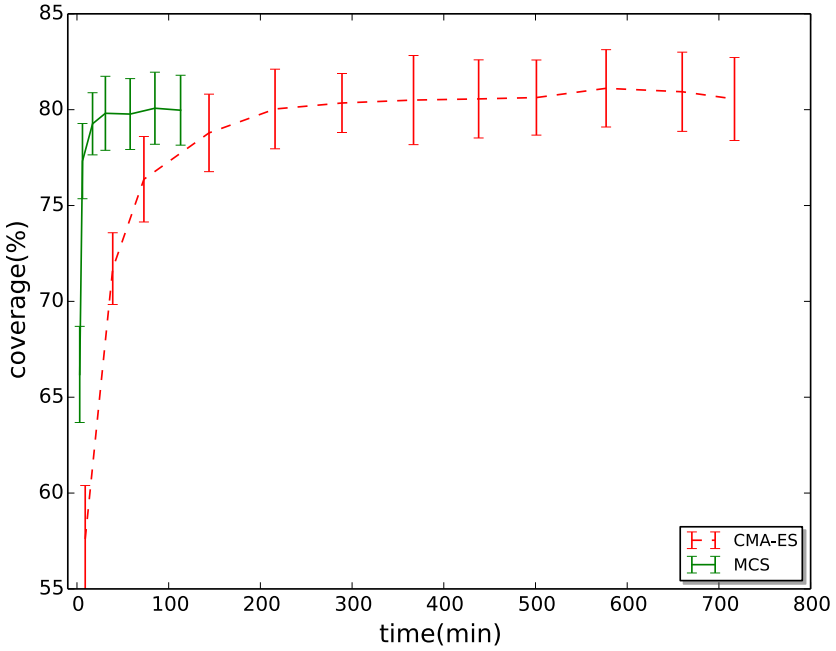


Figure 5.5: Coverage percentage for experiments with 100 targets using CMA-ES and GSCO optimization methods.

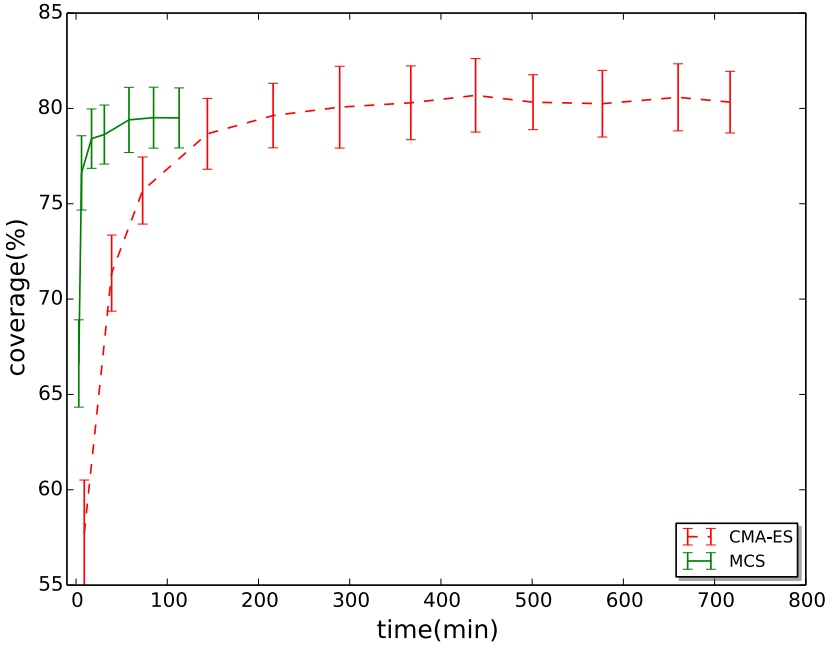


Figure 5.6: Coverage percentage for experiments with 150 targets using CMA-ES and GSCO optimization methods.

ments, we are looking for small running times and large coverages. Therefore, better algorithm should produce results closer to the top-left corner of the figure.

We observe that in all the experiments, the GSCO algorithm converged faster than the CMA-ES algorithm. This is an important aspect as the convergence time is crucial for sensor control algorithms, given that the results should be produced in an online fashion. This is particularly important in the current case where the prediction is done using a short term prediction method, because the system needs to update the direction of the sensors at each time step, and therefore computation time becomes even more important.

Notice the runtimes presented here measure the overall simulation time (i.e., $M = 200$ time steps). Therefore, the overall time should be divided by the total simulation time to get the computation time of the algorithm for each time step.

In terms of coverage performance, we observe that in the case of 50 targets, the GSCO algorithm performed better than the CMA-ES algorithm, while for the 100 and 150 targets cases, the performance of the CMA-ES was slightly better. Although in all the cases the performance difference between the two algorithms is less than 1%, and not statistically significant according to a Wilcoxon signed-rank test (p-value=0.05).

We also observe that as the number of targets increases the overall coverage percentage decreases, which is also reasonable, as covering more targets is a more difficult task. The point is that the decrease in the performance of the GSCO algorithm is more significant than the CMA-ES algorithm. The reason could be that, the CMA-ES algorithm solves the sensor control problem as a general optimization algorithm and therefore the number of targets has less important effect on the algorithm, while the performance of the GSCO algorithm is directly related to the heuristic used to design the algorithm. It could be that the heuristic is more applicable to less crowded scenes with lower number of targets.

We can argue that GSCO algorithm has a demonstrated advantage over CMA-ES for the problem of sensor control. Indeed, the final result of the two algorithms in terms of the coverage is similar while the GSCO algorithm converges faster and produces the result in less time, which is important for online control systems. Therefore, the GSCO algorithm is suitable for the systems which have real-time constraints.

5.5 Conclusion

In this chapter we have presented the GSCO algorithm to optimize the parameters of the sensors in the temporal optimization problem. The proposed algorithm has several advantages compared to other algorithms proposed in the literature, such as the capability to work with the probabilistic sensing model for sensors, to handle different certainties for target locations and to model the limitation of sensor movements.

We also compared the proposed method with a general purpose optimization algorithm (CMA-ES), and observed that our algorithm produces the results in a much shorter time in all the cases considered. Besides, the results of our algorithm was superior to that of the CMA-ES algorithm in simulations with smaller number of targets and competitive in simulations with more targets. Therefore the GSCO algorithm can satisfy the real-time requirements of systems without compromising on the performance in terms of the overall coverage of the network.

Chapter 6

Experiments with the Integrated System

In the previous chapters, we have presented several methods on how to place sensors in an environment (in chapter 3), how to predict the location of the targets moving in that environment (chapter 4) and finally how to control sensors to better cover the moving targets (chapter 5). All these methods are based on the probabilistic coverage model we proposed for sensors in chapter 2. In this chapter, we are integrating all those parts into a complete system which first places the sensors in the environment, and then moves the direction of sensors according to the predictions made over the movement of the targets.

For the rest of the chapter, in section 6.1 we introduce the new dataset we used for our experiments, section 6.2 covers the placement of the sensors in the environment, section 6.3 presents the performance of different trajectory prediction methods we presented, namely, the short term, the long term and the combination of the two. All the experiments are done using the sensor control algorithm we proposed in chapter 5. Finally, section 6.5 concludes the chapter.

6.1 Dataset

For the experiments we use a new simulated dataset that we have not used in our previous experiments. The simulated dataset uses the map of part of the Vieux-Québec district in Quebec City, Canada. The map has 200 rows and 250 columns with a resolution of one meter per dimension. The elevation map for the area is shown in Fig. 6.1. From the figure, we can distinguish the buildings which have blue colour from the ground level which is red.

In the new dataset, targets represent the pedestrians walking between buildings on the map. For the experiments, we added the information of the gates and the pedestrian paths on the map (see Fig. 6.2). In this map, the cyan circles represent the gates of the buildings and

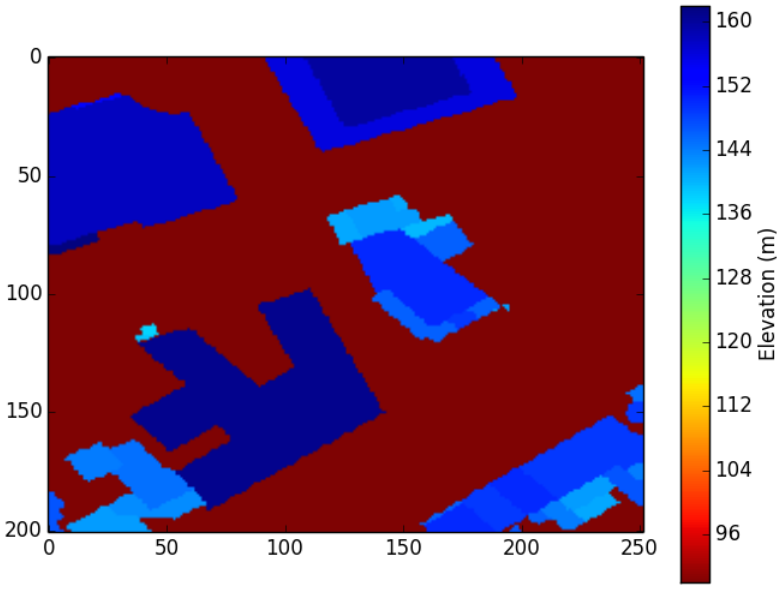


Figure 6.1: The elevation map of the Vieux-Québec area used for experiments.

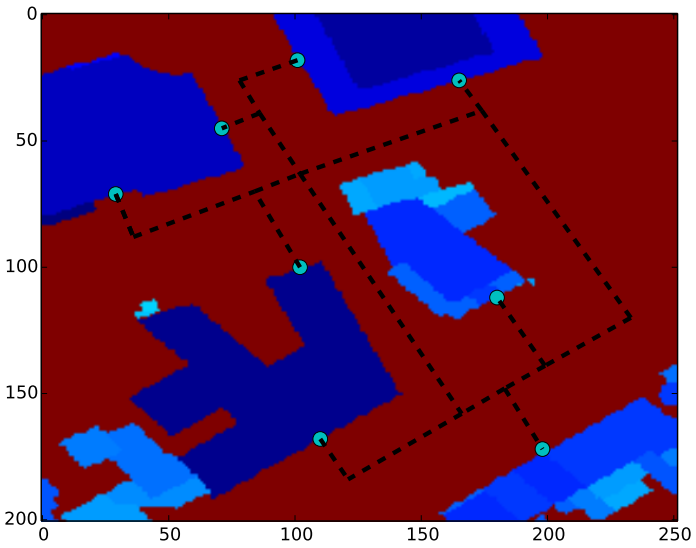


Figure 6.2: The gates and pedestrian path information for the Vieux-Québec map.

the dashed black lines are the pedestrian paths. The trajectory of the targets were generated using the methodology presented in section 4.5.1.2.

The model used for the sensors is the PTZ camera model we proposed in section 2.5. The parameter values used for the model were already given in Table 5.1. We have fixed the number of cameras to 10 for this map.

6.2 Spatial Optimization

The first part of the experiment requires that the location of the sensors be optimized for the environment. Different spatial optimization methods were explained in chapter 3. We observed that for smaller number of sensors, the CMA-ES method performed better than the GD method. Therefore, the location of the sensors in the experiments were optimized using the CMA-ES algorithm.

Notice that the optimization algorithm we used here is slightly different from the one explained in chapter 3. Before, sensors had the three parameters of location, pan, and tilt angles. Here, the sensors are the PTZ cameras, therefore, each sensor has an additional parameter of the focal length. We add the new parameter to the list of parameters and perform the optimization as before. More precisely, in the new experiment, each sensor has the parameters $\mathbf{s}_i = (\mathbf{p}_i, \theta_i, \xi_i, f_i)$, $i = 1, 2, \ldots, 10$, with $\mathbf{p}_i = (x_i, y_i)$. Also notice that the variable parameters do not have a time parameter attributed to them, as we are currently in the spatial optimization phase, which is a one time process.

We follow the weighted coverage model for the experiments. Meaning some parts of the environment could have higher weight than others. As the goal of the system is to cover moving targets, we can use the history of previous targets displacements as the weight map. For that purpose, a weight map was created by adding the trajectory of 1000 targets who have traversed the environment. The location of targets in the trajectories should be summed over each other so that locations which were occupied by more targets get more weight in the final weight map. The weight map is shown in Fig. 6.3. In the figure, locations with lighter colour has higher weight.

The final location of the sensors is shown in Fig. 6.4. The overall coverage of the network over the weight map is 67%. In this figure we observe that the optimization algorithm put more sensors in the area where there was higher weight which is reasonable. In other words, the central, top and bottom corridors were completely covered by the sensors, while the right corridor, which has a smaller weight is not completely covered. In the next section, we use the location of sensors optimized in this section to perform temporal optimization and cover the targets moving in the environment.

6.3 Trajectory Prediction

In the chapter 4, we mentioned two categories of trajectory prediction methods, namely the short term and long term prediction methods, and we mentioned several methods under each category. In that chapter, we compared the performance of different methods using the expected distance measure (Eq. 4.1). In this section, we do the comparison, in the context of the temporal optimization. Meaning the coverage that each method can provide over moving

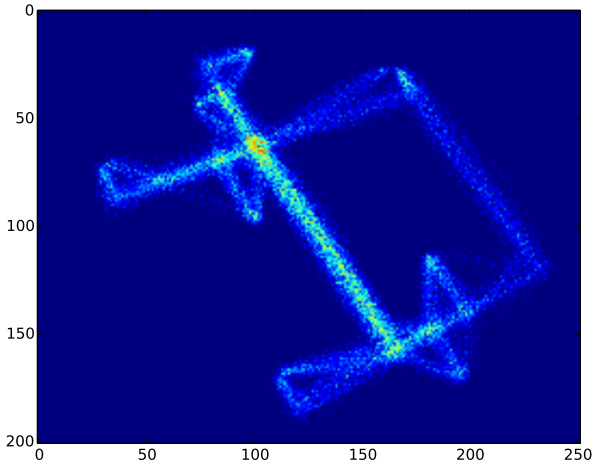


Figure 6.3: The weight map used for the spatial optimization.

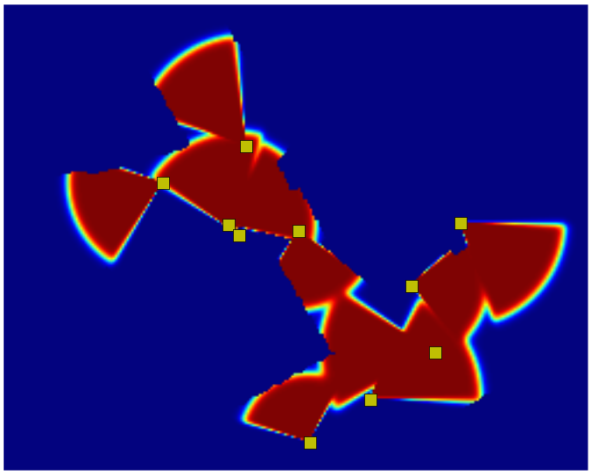


Figure 6.4: Sensor locations optimized by the CMA-ES method.

targets in the environment (Eq. 5.7).

The experiments are done over the map presented in section 6.1. For that purpose different datasets with different ratios between the number of targets and simulation time was created. More precisely, we have three set of experiments each with 30 set of random target trajectories. The number of targets in each set is 20, 50 and 80 targets, and the total simulation time in all the cases is 200 time steps.

For the sensor control algorithm, we use the GSCO algorithm explained in chapter 5. As we saw before, the GSCO algorithm has competitive performance in general compared to the CMA-ES algorithm, while the computation time is much smaller. In the experiments presented here the number of targets is generally small compared to experiments of chapter

Table 6.1: Coverage percentage of the targets with various numbers of targets for the short term prediction methods. For each scenario, 30 sets of random paths were generated, with average coverage of the network, and the standard deviation reported.

	No. of sensors	10	10	10
	No. of targets	20	50	80
	Simulation time	200	200	200
	Avg. cov.	78.45%	78.99%	77.44%
Constant	Stdev.	4.0	3.5	3.2
	Avg. cov.	75.94%	78.07%	76.13%
Average	Stdev.	4.9	2.8	2.5
	Avg. cov.	76.04%	77.44%	76.70%
Exponential average	Stdev.	4.4	3.5	3.1
	Avg. cov.	82.01%	81.11%	79.55%
Unimodal	Stdev.	4.1	2.9	2.2
	Avg. cov.	82.28%	81.90%	79.40%
Kalman filter	Stdev.	3.7	2.6	2.6
	Avg. cov.	82.26%	81.60%	79.57%
Probabilistic exponential avg.	Stdev.	4.3	2.5	2.5
	Avg. cov.	82.88%	82.10%	79.71%
Multimodal	Stdev.	4.5	2.9	2.4
	Avg. cov.	66.35%	67.47%	66.69%
Fixed sensors	Stdev.	7.6	5.5	3.4

5, and we observed that the GSCO algorithm outperforms the CMA-ES algorithm with small number of targets. Therefore, in all the experiments presented here, the performance of GSCO should be superior to that of CMA-ES.

The only parameter of the GSCO algorithm is the number of random samples, ω, generated to optimize the direction of each sensor, which we set the value to 200. The reason is that, in the experiments of previous chapter, we observe that the performance of the GSCO algorithm did not change significantly when the number of random samples generated was increased from 200 to 400.

6.3.1 Short Term Prediction Methods

In this section, we compare the performance of the seven different short term prediction methods explained in section 4.3, namely, constant, average, exponential average, probabilistic exponential average, multimodal, Kalman filter, and unimodal. Besides, we have performed another set of experiments, with the fixed (as in Fig. 6.4) direction for the sensors. The results of the fixed (non-adaptive) setting works as a baseline result to evaluate the effectiveness of the temporal optimization approach in general. The variable parameters of the methods is the same as before (Table 4.2).

The final results of the experiments are presented in Table 6.1. For all the results presented in this chapter the average coverage of the network over the moving targets (Eq. 5.7) was used as a measure of performance. Looking at the results, we observe three levels of performance, meaning that there is statistical performance difference between different levels but not within each level. The lowest level is the fixed sensors. As expected the non-adaptable sensor configurations provide lowest coverage level over the targets. Therefore, we can conclude that usage of the temporal optimization with adaptive sensors can improve the coverage of the final network. This is an important point, as it shows the effectiveness of the temporal optimization methods in improving the overall coverage of the network over the moving targets.

The second level contains the non probabilistic prediction methods, namely, the constant, average, and exponential average methods. The interesting observation is that the constant method produced the best results among the tested non probabilistic prediction methods. Although one should mention that the difference is non statistically significant.

The third level consists of the probabilistic prediction methods, which produced the best results. Among the four prediction methods, they all produced competitive results over all simulation settings. We observe that the results of the multimodal approach is slightly better than the three other methods. The multimodal and probabilistic exponential avg. and unimodal methods produced very similar results on the simulations with 80 targets, while the performance of the multimodal approach was slightly better in the simulations with 20 and 50 targets.

In chapter 4, we observed that the performance of the probabilistic methods was inferior to that of the non probabilistic methods (with respect to the expected distance measure), while here the probabilistic approaches have a superior performance compared to the other three methods. Therefore, the results of the expected distance measure cannot be directly used to measure the performance of different methods once used in the context of temporal optimization.

The weak point of the unimodal and probabilistic exponential avg. approaches concerns the direction changes, as it take some time for both methods to modify their prediction and adjust to new directions. On the other hand, the direction change of the Kalman Filter method is extremely fast, as this method basically uses the latest displacement for prediction. Therefore, the weakness of the Kalman filter method is the lack of memory.

The multimodal method makes a good balance between the two requirement, as the prediction in this method gets updated by the change of direction, and at the same time, the method keeps some of the history of the recent displacements, and therefore is not completely dependant on the latest movement.

The running time for all different methods was roughly 130 minutes. With respect to the

computation time, there is not a significant difference between different prediction methods. The reason is that, the total simulation time of the algorithms is almost completely due to the computations of the coverage of the network over the targets, and the sensor control method (GSCO). In other words, in all cases, the trajectory prediction methods have a negligible effect on the total simulation time.

6.3.2 Long Term Prediction Methods

The same set of experiments that were used to test the performance of the short term methods, were used to test the long term methods (KDE, LCSS, and PCA as explained in section 4.4). The first difference between the two experiments is that for the short term methods, the prediction step is one time step. On the other hand, for the long term methods, the prediction step can be arbitrarily large, as the prediction relies on the trajectory of other targets. To evaluate the performance of the mentioned methods, we performed two set of experiments with different prediction steps of ($s = 5$) and ($s = 10$) time steps. The total simulation time is 200 time steps, which means that the trajectory prediction and the sensor control algorithms are performed 40 and 20 times for each case, respectively.

The other difference is that the long term methods require to have access to a dataset of previous trajectories. Notice that the number of trajectories in our test sets is limited (20 to 80), therefore we cannot use the previous trajectories in the same set to predict the trajectory of a new target (the same way we implemented in the previous chapter). Thus, we need an extra dataset for the prediction of trajectories. We used the dataset of 1000 trajectories (which we previously used in the spatial optimization phase) for that purpose.

The results of the experiments are presented in Table 6.2. It can be seen that the KDE method outperformed the LCSS and PCA methods over all the simulations with the prediction step of five. When the prediction step equals ten there is less difference between the performance of the three methods. The reason could be that as the prediction step increases, the prediction process becomes more difficult, and the difference between the prediction methods would be less significant. In general, for all methods, the results are better when the prediction step equals five compared to ten.

The computational time is linearly related to the prediction step. This is also reasonable, as smaller prediction steps require more trajectory prediction and sensor control. The KDE method is more computationally expensive compared to the other two methods.

From the other perspective, the computational time increases with the number of targets in the environment. The reason is that in long term prediction methods, the trajectory prediction is a costly computation (unlike short term methods), because the trajectory of the new target should be compared with all the trajectories in the dataset. While in the short term methods, the prediction step is rather fast, and the overall computational time is related to the coverage

Table 6.2: Coverage percentage of the targets with various numbers of targets for the global prediction methods. For each scenario, 30 sets of random paths were generated, with average coverage of the network, standard deviation, and CPU time.

	No. of sensors	10	10	10
	No. of targets	20	50	80
	Simulation time	200	200	200
Prediction time = 5				
KDE	Avg. cov.	**73.51**%	**73.66**%	**72.98**%
	Stdev.	5.2	2.3	2.7
	CPU time	28 m	33 m	38 m
LCSS	Avg. cov.	68.54%	70.05%	69.07%
	Stdev.	4.6	2.5	2.5
	CPU time	10 m	17 m	25 m
PCA	Avg. cov.	70.22%	70.56%	70.21%
	Stdev.	5.4	2.9	2.7
	CPU time	10 m	18 m	25 m
Prediction time = 10				
KDE	Avg. cov.	66.26%	68.59%	68.51%
	Stdev.	5.7	2.8	2.4
	CPU time	10 m	16 m	17 m
LCSS	Avg. cov.	65.46%	67.11%	67.16%
	Stdev.	4.8	2.8	2.5
	CPU time	5 m	10 m	12 m
PCA	Avg. cov.	65.70%	68.48%	68.03%
	Stdev.	5.2	3.2	2.8
	CPU time	4 m	10 m	13 m

calculation and sensor control which should be performed in all time steps.

Compared to the results produced by the short term methods, all results of the long term methods are inferior to that of the short term methods. But in general the computational requirement is much lower compared to the short term prediction methods. Therefore, we can conclude that long term methods are a feasible solution, when the computational resources are scarce. Remember this was the original reason we proposed the long term prediction methods in the first place.

Compared to the fixed sensor results, we observe that for the prediction of five time steps, there is an improvement over the results of fixed sensors, while in the ten time step prediction time, the results are equal or worse than the fixed sensor case. Therefore, from our experiments, we can conclude that when the coverage of the fixed sensors placement is relatively good over the environment, the long term prediction methods with large prediction steps cannot improve the coverage of the targets.

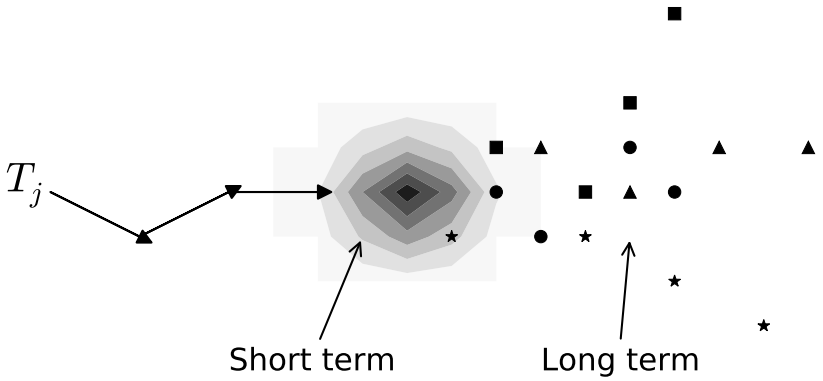


Figure 6.5: The combination of short and long term prediction methods for predicting the future location of a target.

6.4 Combination of Short and Long Term Prediction Methods

In this approach the prediction of a short term and that of a long term trajectory prediction methods are overlaid on top of each other. The reason is that the short term prediction methods are better at predicting the location of the target in the next time step, therefore they are vital for keeping maximal coverage over targets all the time. On the other hand, the long term prediction methods better represent the possible paths that the target might take in the future. Therefore, they can better inform possible candidate sensors to change direction and get ready to anticipate target trajectories.

In Fig. 6.5, we have presented a possible combination of short and long term prediction methods. In this figure, target $\mathbf{T}_j$ has made three displacements in the environment and now we are making predictions for its future locations. As mentioned before, the short term method is used to predict the next location of the target, here we presented the result of the probabilistic exponential average method for the short term prediction.

The long term method is used for longer prediction steps. Here, the long prediction step is five. Therefore, the next location is predicted with the short term method and locations which are 2 to 5 time steps in the future are predicted using the long term methods. In the presented scenario, target $\mathbf{T}_j$ has similarity with four other trajectories, the location of each is represented with a different icon. As you can see, each similar trajectory predicts a different trajectory for the future of the trajectory. Depending on the similarity of $\mathbf{T}_j$ with each of the mentioned trajectories, the predicted locations with the long term method have different weights.

Following the terminology introduced in section 4.2, the probability of target j being at location $\mathbf{q}$ at time $(t+s)$ (i.e., $\mathcal{P}_{j\mathbf{q}}^{(t+s)}$) is given by:

$$\mathcal{P}_{j\mathbf{q}}^{(t+s)} = \begin{cases} \alpha \cdot \mathcal{PS}_{j\mathbf{q}}^{(t+1)} & \text{if } s \text{ equals 1} \\ (1-\alpha) \cdot \mathcal{PL}_{j\mathbf{q}}^{(t+s)} & \text{if } s \text{ is greater than 1} \end{cases} \tag{6.1}$$

where $\mathcal{PS}_{j\mathbf{q}}^{(t+1)}$ is the result of prediction made by the short term prediction method, and $\mathcal{PL}_{j\mathbf{q}}^{(t+s)}$ is the result of long term prediction method. Notice that the sole concatenation of the two distributions does not produce a probability distribution per se, as each of the short and long term methods produce a separate distributions. Therefore, each distribution should be scaled so that the final result sums up to one. The α parameter has the responsibility of normalizing the whole distribution. Here we used the value of 0.5 for the α parameter, but notice that different values can be used to put more emphasis one the short term or long term methods.

As a final experiment, we combined the multimodal and KDE methods. The reason was that the KDE method produced the best result among the long term prediction methods, and multimodal was the best among the short term methods (although not statistically better). The prediction step of the multimodal method is set to one, and for the KDE approach is five.

We have presented the results of the combined method in Table 6.3. We compared the results with the multimodal approach as it produced the best results among all the methods previously tested. We can see that the combined method, has improved the performance of the multimodal approach.

Although in the combined method we have tried to combine the strong points of both short and long term methods, one should note that the shortcoming of this approach is related to the computational time. The reason is that for this prediction model, the results of both prediction methods should be computed and then combined with each other.

This can be another benefit of the long term prediction methods. So far, we have presented the long term prediction methods as a solution for systems in which the computational demand is of concern. Here, we observe that the long term prediction methods can be added to the short term methods to improve their performance.

6.5 Conclusion

In this chapter, we have presented a complete integration of different subsystems we have proposed in previous chapters. In a completely new environment, we first optimized the placement of the sensors using the method proposed in chapter 3, based on the sensor model presented in chapter 2. Next, the performance of different trajectory prediction methods proposed in chapter 4 was compared in a combination with the sensor control method proposed in chapter 5.

Table 6.3: Coverage percentage of the targets for combination of short and long term prediction methods.

	No. of sensors	10	10	10
	No. of targets	20	50	80
	Simulation time	200	200	200
	Avg. cov.	82.88%	82.10%	79.71%
Multimodal	Stdev.	4.5	2.9	2.4
	Avg. cov.	**84.08**%	**83.23**%	**80.84**%
Multimodal + KDE	Stdev.	3.2	2.4	2.4

In general we observed that the temporal optimization methods (combination of trajectory prediction and sensor control) outperformed the static cameras in terms of coverage. This is a great achievement as it shows that the extra computation needed to perform different prediction and optimization algorithms will eventually improve the coverage of the network over the moving targets.

Among the short term prediction methods, the probabilistic methods produced better results in the context of temporal optimization and coverage over moving targets. This is in contrast to the results we observed previously, that the non-probabilistic methods performed better under the expected distance measure. There was not a significant difference between different probabilistic methods, but the multimodal approach performed slightly better than the other methods.

As for the long term prediction methods, they were the least computationally demanding methods. Among the proposed methods, KDE produced better results than the other two for short prediction time steps. The performance of all methods were equal or inferior to fixed sensor model for long prediction steps. Therefore we can conclude if the computational resources are scarce, it is better to use fix sensor model over the adaptive model with long prediction steps.

As the final step, we observed that the combination of short and long term prediction methods would produce the best results compared to all other prediction methods. Therefore, the combination can actually bring the benefits of both approaches into one method.

Chapter 7

Conclusion

In this thesis, we have presented a probabilistic model for the sensing region of sensors, considering the effect of obstacles and terrain information using the line-of-sight coverage region. We also proposed a placement optimization mechanism based on the gradient descent method to find the best location and direction for sensors in the environment.

In the temporal optimization part we focused on covering the moving targets in the environment. In this part, we assumed that the location of the sensors has already been optimized but their direction should be adjusted based on the location of mobile targets. For that, we proposed several methods to predict the future location of targets using either the history of displacements of the same target or other targets. Then we used that prediction, and converted the temporal problem of target coverage into an optimization problem. For this problem we proposed a heuristic algorithm to optimize the parameters of the sensors so that the network can keep maximum coverage over the moving targets.

As all the experiments in the thesis were done in simulated environments, one of the main criteria throughout the thesis was to make realistic assumptions (e.g., coverage region of sensors, or movement model of targets) about different parts of the system. Besides, one of the major requirements for the problem tackled in the thesis (coverage of targets) is the real-time constraints for the approaches proposed to solve it. Therefore, we always tried to keep the computation time as one of the major performance measures for all the methods proposed and tested.

This work provides a complete system for the deployment of SNs. The system will make the work of people interested in the deployment of SNs much simpler, as the performance of a real system can be evaluated even before a real sensor is put in place. The users can evaluate the performance of system under different conditions, and see how different decisions (e.g. algorithms) can affect the final goal of the system.

7.1 Research Contributions

This thesis deals with the problems of sensor model, sensor placement, trajectory prediction and sensor control. The main contributions of this thesis in each of the mentioned fields are categorized as follows.

7.1.1 Sensor Model

Proposed a novel realistic sensing model, relying on probabilistic 3D sensor coverage and line-of-sight visibility. In the first part, we studied the models proposed to represent the coverage region of a sensor. Most of the proposed sensor models make oversimplified assumptions about the coverage region of the sensors with respect to environmental factors. These approaches cannot deal with environmental factors such as terrain topography, and usually assume an omnidirectional disk sensing model for each sensor. In fact, under the assumptions of uniform disk sensing model, it has been shown that optimal coverage can be deterministically achieved with a regular placement of sensors [10].

Facing this challenge, we follow a more flexible avenue for a realistic modelling of sensing. Our approach differs from previous methods in the following three ways:

- Most previous schemes only consider 2D environments and ignore the effects of elevation, whereas our method takes into account the 2.5D terrain information. In our approach, the environment is defined using a Geographic Information System (GIS), which is an information system designed to store, manipulate and analyze geographically referenced data [84].
- Most previous schemes assume omnidirectional sensors, whereas our method allows for constraints to be applied on sensors, such as limited sensing angles and range.
- Previous schemes implement mainly binary coverage, i.e., a point can only be classified as covered or uncovered, whereas our method applies probabilistic coverage in both sensing distance and sensing angle.

In summary, deterministic approaches assume omni-directional sensors with binary coverage on a flat terrain, while our probabilistic approach assumes directional sensors, for which omnidirectional sensors are a special case, with probabilistic coverage on a realistic spatial model of the environment. In order to tackle these problems, we develop a novel probabilistic coverage function for sensor placement that takes into account the above mentioned issues.

The probabilistic approach proposed here for modelling the coverage area of line-of-sight based sensors, provides a baseline to model the behaviour of other types of devices who make measurements in the environment. In general, using probabilistic reasoning to measure the per-

formance of sensor can be a huge benefit for the development of sensing technologies, as the performance of the sensors in simulation and reality would resemble as much as possible.

7.1.2 Placement Problem

Developed a fast and scalable placement optimization algorithm based on gradient descent method. Once the coverage model of the sensor is defined considering the mentioned criteria, it is the responsibility of the optimization algorithm to find the placement of the sensors. Therefore, in the next step, we are proposing to use gradient descent (GD) as an optimization method for the sensor placement problem. At each step of the algorithm, we calculate the analytical derivatives of the coverage function with respect to the position and orientation of each sensor and try to move them in a way that maximizes the overall coverage of the network.

Unlike previous approaches on sensor placement optimization using the gradient descent method [27, 72], we use a realistic model for the terrain and a directional probabilistic sensing model for the coverage of each sensor [5]. We have shown that in addition to its simplicity, our method can produce results comparable to more sophisticated black box optimization methods, without the need for large computational resources.

7.1.3 Trajectory Prediction

The final goal of the system is to cover the moving targets in the environment. Therefore, once the best location for the sensors has been determined in the placement phase, trajectory prediction aims at guessing the next location of the targets with maximum accuracy. There are two general approaches for that purpose, namely, short term and long term prediction methods. In the short term method, only the previous history of each target is used to predict its future location, while in the long term prediction, the history of other targets is also used to further increase the prediction horizon for each target.

Proposed a multi-hypothesis probabilistic short term trajectory prediction method. We have proposed several probabilistic short term prediction methods, and compared their performance with other methods proposed in the literature. More precisely, we have proposed the probabilistic exponential average method, the multimodal methods, and also used maximum likelihood to learn the parameters of the unimodal method. All these methods produced superior results compared to non probabilistic methods [86, 87, 88].

Derived a similarity function between spatio-temporal instances from Kernel Density Estimation, and then used it as a long term trajectory prediction method : As for the long term prediction methods, we have proposed to use the kernel density estimation (KDE) as a similarity measure between trajectories. the proposed model was compared

with two other commonly used methods, namely PCA [12] and LCSS [77]. Results showed that although the proposed approach is slightly more expensive in term of computation time, it outperformed both of the other methods. The strong performance of the method is related to its probabilistic nature and consideration of the displacement in the similarity function.

Improved the performance of trajectory prediction by combining the short and long term trajectory prediction methods. As a final approach, we integrated the short and long term prediction methods to have the benefits of both approaches. For that purpose, we have combined the Multimodal and KDE prediction methods, and observed that the combination produces superior results compared to individual approaches.

7.1.4 Sensor Control

Proposed a Greedy heuristic approach for sensor control. Finally, to optimize the sensing parameters of the sensors we have proposed the Greedy Sensor-wise Coverage Optimization (GSCO). The proposed algorithm follows a greedy approach for the optimization process. For that purpose, an accessible coverage map is defined for each sensor, which determines all the locations that a sensor can cover at the next time step. Then, for each iteration of the algorithm, all the locations in the environment are reweighed to give more importance to the locations which can be covered by less number of sensors. These new weights are used to optimize the parameters of the sensors using a random search.

Our method has the following advantages over the methods proposed in the literature (e.g., [18]):

- Methods from the literature do not consider any limitation on the movement speed of sensor direction, while in our approach limitation of movement speed is critical in determining the possible coverage map of each sensor.
- In all other methods, the coverage of sensors over the environment was binary, while it is a probabilistic coverage in our sensor model.
- Other methods consider only one variable parameter per sensor, and therefore iterate over all possible directions of the sensors. This approach is not reasonable when a sensor has two or three variable parameters, as iteration over all possible combinations is cumbersome.
- Some of the other methods do not work with probabilistic prediction for location of the targets, while in our problem, the trajectory prediction produces different certainties for the future location of a target. As a result our algorithm is capable of working in weighted environment.

7.2 Future work

In this thesis we covered the sensor network application, mostly in the context of PTZ camera networks. With the emerge of Internet of Things (IoT), the number of sensors in many devices connected to each other is increasing. A great potential application of the algorithms proposed here is to evaluate their performance for the networks consisting of different types of sensors with different coverage definitions.

In chapter 3 we proposed the GD algorithm for placement of sensors. The proposed placement approach has implications in other fields. For example, in the observer sitting problem which is usually addressed in the geomatics community, could be viewed as a special case of the placement problem we presented here. In both problems the goal is to find the best location for sensors (observers) so that the combination coverage of them covers certain parts of the environment.

In chapter 4 the Multimodal method was proposed to predict the trajectories of humans, while the strong points of this approach is to handle several assumptions at the same time (such as zig-zag movement). This movement pattern is not usually true for humans, and therefore the strong advantages of this method were not apparent in the experiments performed in this thesis. A possible future work is the application of this method for prediction of trajectories which have zig-zag movement patterns.

In chapter 4 the KDE based similarity function was proposed to measure the similarity between trajectories. The proposed similarity measure is of great interest beyond the applications mentioned in this thesis. For example, many applications including (but not limited to) speech/handwritten/gesture recognition, computer graphics, protein sequence alignment, etc., heavily depend on a similarity function between spatio-temporal instances. Therefore, we believe that besides the mentioned application, the proposed method could have applications in a great variety of domains.

Appendix A

Calculation of Derivatives

The following section provides the calculations of the derivations used in the gradient descent method for the spatial optimization of sensors. This section presents the calculation of the analytical derivative of the membership functions (μ_{di}, μ_{pi} and μ_{ti}) with respect to the four free parameters x_i, y_i, θ_i and ξ_i. The following function is introduced to simplify the math formulations:

$$\mathrm{dsg}(\delta, \beta, \alpha) = \frac{\beta \exp(\beta(\delta + \alpha))}{(1 + \exp(\beta(\delta + \alpha)))^2} = \beta \ \mathrm{sig}(\delta, \beta, \alpha)[1 - \mathrm{sig}(\delta, \beta, \alpha)] \tag{A.1}$$

which corresponds to the derivative of the so-called sigmoid function:

$$\mathrm{sig}(\delta, \beta, \alpha) = \frac{1}{1 + \exp(\beta(\delta + \alpha))} \tag{A.2}$$

First, let us develop the derivatives of membership functions with respect to x_i and y_i, that is:

$$\frac{\partial \left[\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}\right]}{\partial x_i} = \left[\frac{\partial \mu_{di}}{\partial x_i} \cdot \mu_{pi} \cdot \mu_{ti} + \frac{\partial \mu_{pi}}{\partial x_i} \cdot \mu_{di} \cdot \mu_{ti} + \frac{\partial \mu_{ti}}{\partial x_i} \cdot \mu_{di} \cdot \mu_{pi}\right] \tag{A.3}$$

$$\frac{\partial \left[\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}\right]}{\partial y_i} = \left[\frac{\partial \mu_{di}}{\partial y_i} \cdot \mu_{pi} \cdot \mu_{ti} + \frac{\partial \mu_{pi}}{\partial y_i} \cdot \mu_{di} \cdot \mu_{ti} + \frac{\partial \mu_{ti}}{\partial y_i} \cdot \mu_{di} \cdot \mu_{pi}\right] \tag{A.4}$$

For the term $\frac{\partial \mu_{di}}{\partial x_i}$:

$$\phi_{di} = \|\mathbf{p}_i - \mathbf{q}\| = \sqrt{(x_i - x_q)^2 + (y_i - y_q)^2}, \tag{A.5}$$

$$\frac{\partial \mu_{di}}{\partial \phi_{di}} = \frac{-\beta_d \exp(-\beta_d(\phi_{di} - \alpha_d))}{(1 + \exp(-\beta_d(\phi_{di} - \alpha_d)))^2} = \mathrm{dsg}(\phi_{di}, -\beta_d, -\alpha_d) \tag{A.6}$$

To obtain the terms $\frac{\partial \mu_{di}}{\partial x_i}$ and $\frac{\partial \mu_{di}}{\partial y_i}$, we can simply use the chain rule by multiplying the term

$\frac{\partial \mu_{di}}{\partial \phi_{di}}$ with additional terms $\frac{\partial \phi_{di}}{\partial x_i}$ and $\frac{\partial \phi_{di}}{\partial y_i}$, respectively:

$$\frac{\partial \phi_{di}}{\partial x_i} = \frac{\partial \sqrt{(x_i - x_q)^2 + (y_i - y_q)^2}}{\partial x_i} = \frac{(x_i - x_q)}{\sqrt{(x_i - x_q)^2 + (y_i - y_q)^2}} = \frac{(x_i - x_q)}{\phi_{di}} \tag{A.7}$$

$$\frac{\partial \phi_{di}}{\partial y_i} = \frac{\partial \sqrt{(x_i - x_q)^2 + (y_i - y_q)^2}}{\partial y_i} = \frac{(y_i - y_q)}{\sqrt{(x_i - x_q)^2 + (y_i - y_q)^2}} = \frac{(y_i - y_q)}{\phi_{di}} \tag{A.8}$$

Consequently, the derivatives of the membership function μ_{di} can be calculated as:

$$\frac{\partial \mu_{di}}{\partial x_i} = \frac{\partial \mu_{di}}{\partial \phi_{di}} \cdot \frac{\partial \phi_{di}}{\partial x_i} = \mathrm{dsg}(\phi_{di}, -\beta_d, -\alpha_d) \cdot \frac{(x_i - x_q)}{\phi_{di}} \tag{A.9}$$

$$\frac{\partial \mu_{di}}{\partial y_i} = \frac{\partial \mu_{di}}{\partial \phi_{di}} \cdot \frac{\partial \phi_{di}}{\partial y_i} = \mathrm{dsg}(\phi_{di}, -\beta_d, -\alpha_d) \cdot \frac{(y_i - y_q)}{\phi_{di}} \tag{A.10}$$

For the term $\frac{\partial \mu_{pi}}{\partial x_i}$, we first notice that $\mu_{pi} = \mu_p(\phi_{pi})$ where $\phi_{pi} = \angle_p(\mathbf{q} - \mathbf{p}_i) - \theta_i$. Therefore, we have:

$$\frac{\partial \mu_{pi}}{\partial x_i} = \frac{\partial \mu_{pi}}{\partial \phi_{pi}} \cdot \frac{\partial \phi_{pi}}{\partial x_i} \tag{A.11}$$

In the term $\frac{\partial \phi_{pi}}{\partial x_i}$ only the pan angle $\angle_p(\mathbf{q} - \mathbf{p}_i)$ is a function of x_i and y_i:

$$\angle_p(\mathbf{q} - \mathbf{p}_i) = \arctan\left(\frac{y_\mathbf{q} - y_i}{x_\mathbf{q} - x_i}\right) \tag{A.12}$$

and, therefore, we have $\frac{\partial \phi_{pi}}{\partial x_i} = \frac{\partial \angle_p(\mathbf{q} - \mathbf{p}_i)}{\partial x_i}$. The derivative of $\angle_p(\mathbf{q} - \mathbf{p}_i)$ with respect to x_i is thus:

$$\frac{\partial \angle_p(\mathbf{q} - \mathbf{p}_i)}{\partial x_i} = \frac{\partial \angle_p(\mathbf{q} - \mathbf{p}_i)}{\partial \left(\frac{y_\mathbf{q} - y_i}{x_\mathbf{q} - x_i}\right)} \cdot \frac{\partial \left(\frac{y_\mathbf{q} - y_i}{x_\mathbf{q} - x_i}\right)}{\partial x_i} = \frac{1}{\left(\frac{y_\mathbf{q} - y_i}{x_\mathbf{q} - x_i}\right)^2 + 1} \cdot \frac{(y_\mathbf{q} - y_i)}{(x_\mathbf{q} - x_i)^2} = \frac{y_\mathbf{q} - y_i}{\phi_{di}^2} \tag{A.13}$$

Similarly, the derivative of $\angle_p(\mathbf{q} - \mathbf{p}_i)$ with respect to y_i can be obtained:

$$\frac{\partial \angle_p(\mathbf{q} - \mathbf{p}_i)}{\partial y_i} = \frac{\partial \angle_p(\mathbf{q} - \mathbf{p}_i)}{\partial \left(\frac{y_\mathbf{q} - y_i}{x_\mathbf{q} - x_i}\right)} \cdot \frac{\partial \left(\frac{y_\mathbf{q} - y_i}{x_\mathbf{q} - x_i}\right)}{\partial y_i} = -\frac{1}{\left(\frac{y_\mathbf{q} - y_i}{x_\mathbf{q} - x_i}\right)^2 + 1} \cdot \frac{1}{(x_\mathbf{q} - x_i)} = -\frac{x_\mathbf{q} - x_i}{\phi_{di}^2} \tag{A.14}$$

Consequently, we can also develop the terms $\frac{\partial \mu_{pi}}{\partial x_i}$ and $\frac{\partial \mu_{pi}}{\partial y_i}$ as in the following:

$$\begin{aligned}
\frac{\partial \mu_{pi}}{\partial x_i} &= \frac{\partial \mu_{pi}}{\partial \phi_{pi}} \cdot \frac{\partial \phi_{pi}}{\partial x_i} = \frac{\partial \left[\frac{1}{1+\exp(-\beta_p(\phi_{pi}+\alpha_p))} - \frac{1}{1+\exp(-\beta_p(\phi_{pi}-\alpha_p))}\right]}{\partial \phi_{pi}} \cdot \frac{\partial \angle_p(\mathbf{q} - \mathbf{p}_i)}{\partial x_i} \\
&= \left[\frac{\beta_p \exp\left(-\beta_p(\phi_{pi} + \alpha_p)\right))}{(1 + \exp\left(-\beta_p(\phi_{pi} + \alpha_p)\right))^2} - \frac{\beta_p \exp\left(-\beta_p(\phi_{pi} - \alpha_p)\right))}{(1 + \exp\left(-\beta_p(\phi_{pi} - \alpha_p)\right))^2}\right] \cdot \frac{y_\mathbf{q} - y_i}{\phi_{di}^2} \\
&= \left[-\mathrm{dsg}(\phi_{pi}, -\beta_p, \alpha_p) + \mathrm{dsg}(\phi_{pi}, -\beta_p, -\alpha_p)\right] \cdot \frac{y_\mathbf{q} - y_i}{\phi_{di}^2}
\end{aligned} \tag{A.15}$$

$$\frac{\partial \mu_{pi}}{\partial y_i} = \frac{\partial \mu_{pi}}{\partial \phi_{pi}} \cdot \frac{\partial \phi_{pi}}{\partial y_i} = \frac{\partial \left[\frac{1}{1+\exp(-\beta_p(\phi_{pi}+\alpha_p))} - \frac{1}{1+\exp(-\beta_p(\phi_{pi}-\alpha_p))} \right]}{\partial \phi_{pi}} \cdot \frac{\partial \angle_p(\mathbf{q}-\mathbf{p}_i)}{\partial y_i}$$
$$= \left[\frac{\beta_p \exp\left(-\beta_p(\phi_{pi}+\alpha_p)\right))}{(1+\exp\left(-\beta_p(\phi_{pi}+\alpha_p)\right))^2} - \frac{\beta_p \exp\left(-\beta_p(\phi_{pi}-\alpha_p)\right))}{(1+\exp\left(-\beta_p(\phi_{pi}-\alpha_p)\right))^2} \right] \cdot \frac{x_i - x_{\mathbf{q}}}{\phi_{di}^2}$$
$$= \left[-\mathrm{dsg}(\phi_{pi}, -\beta_p, \alpha_p) + \mathrm{dsg}(\phi_{pi}, -\beta_p, -\alpha_p)\right] \cdot \frac{x_i - x_{\mathbf{q}}}{\phi_{di}^2} \tag{A.16}$$

For the term $\frac{\partial \mu_{ti}}{\partial x_i}$, we have $\mu_{ti} = \mu_t(\phi_{ti})$ where $\phi_{ti} = \angle_t(\mathbf{q}-\mathbf{p}_i) - \xi_i$:

$$\frac{\partial \mu_{ti}}{\partial x_i} = \frac{\partial \mu_{ti}}{\partial \phi_{ti}} \cdot \frac{\partial \phi_{ti}}{\partial x_i} \tag{A.17}$$

In the term $\frac{\partial \phi_{ti}}{\partial x_i}$, only the tilt angle $\angle_t(\mathbf{q}-\mathbf{p}_i)$ is a function of x_i and y_i (indirectly through $\mathbf{p}_i$):

$$\angle_t(\mathbf{q}-\mathbf{p}_i) = \arctan\left(\frac{z_{\mathbf{q}} - z_i}{\|\mathbf{p}_i - \mathbf{q}\|}\right) = \arctan\left(\frac{z_{\mathbf{q}} - z_i}{\phi_{di}}\right) \tag{A.18}$$

and, therefore, we have $\frac{\partial \phi_{ti}}{\partial x_i} = \frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial x_i}$. If we do the calculation for $\frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial x_i}$, we have:

$$\frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial x_i} = \frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial \left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)} \cdot \frac{\partial \left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)}{\partial \phi_{di}} \cdot \frac{\partial \phi_{di}}{\partial x_i} = \frac{1}{1+\left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)^2} \cdot \frac{z_i - z_{\mathbf{q}}}{\phi_{di}^2} \cdot \frac{x_i - x_q}{\phi_{di}} \tag{A.19}$$

similarly:

$$\frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial y_i} = \frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial \left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)} \cdot \frac{\partial \left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)}{\partial \phi_{di}} \cdot \frac{\partial \phi_{di}}{\partial y_i} = \frac{1}{1+\left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)^2} \cdot \frac{z_i - z_{\mathbf{q}}}{\phi_{di}^2} \cdot \frac{y_i - y_q}{\phi_{di}} \tag{A.20}$$

and therefore:

$$\frac{\partial \mu_{ti}}{\partial x_i} = \frac{\partial \mu_{ti}}{\partial \phi_{ti}} \cdot \frac{\partial \phi_{ti}}{\partial x_i} = \frac{\partial \left[\frac{1}{1+\exp(-\beta_t(\phi_{ti}+\alpha_t))} - \frac{1}{1+\exp(-\beta_t(\phi_{ti}-\alpha_t))} \right]}{\partial \phi_{ti}} \cdot \frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial x_i}$$
$$= \left[\frac{\beta_t \exp\left(-\beta_t(\phi_{ti}+\alpha_t)\right))}{(1+\exp\left(-\beta_t(\phi_{ti}+\alpha_t)\right))^2} - \frac{\beta_t \exp\left(-\beta_t(\phi_{ti}-\alpha_t)\right))}{(1+\exp\left(-\beta_t(\phi_{ti}-\alpha_t)\right))^2} \right] \cdot \frac{1}{1+\left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)^2} \cdot \frac{z_i - z_{\mathbf{q}}}{\phi_{di}^2} \cdot \frac{x_i - x_q}{\phi_{di}}$$
$$= \left[\mathrm{dsg}(\phi_{ti}, -\beta_t, -\alpha_t) - \mathrm{dsg}(\phi_{ti}, -\beta_t, \alpha_t)\right] \cdot \frac{(z_i - z_{\mathbf{q}})(x_i - x_{\mathbf{q}})}{\phi_{di}[\phi_{di}^2 + (z_i - z_{\mathbf{q}})^2]} \tag{A.21}$$

also, we have:

$$\frac{\partial \mu_{ti}}{\partial y_i} = \frac{\partial \mu_{ti}}{\partial \phi_{ti}} \cdot \frac{\partial \phi_{ti}}{\partial y_i} = \frac{\partial \left[\frac{1}{1+\exp(-\beta_t(\phi_{ti}+\alpha_t))} - \frac{1}{1+\exp(-\beta_t(\phi_{ti}-\alpha_t))} \right]}{\partial \phi_{ti}} \cdot \frac{\partial \angle_t(\mathbf{q}-\mathbf{p}_i)}{\partial y_i}$$
$$= \left[\frac{\beta_p \exp\left(-\beta_t(\phi_{ti}+\alpha_t)\right))}{(1+\exp\left(-\beta_t(\phi_{ti}+\alpha_t)\right))^2} - \frac{\beta_t \exp\left(-\beta_t(\phi_{ti}-\alpha_t)\right))}{(1+\exp\left(-\beta_t(\phi_{ti}-\alpha_t)\right))^2} \right] \cdot \frac{1}{1+\left(\frac{z_{\mathbf{q}}-z_i}{\phi_{di}}\right)^2} \cdot \frac{z_i - z_{\mathbf{q}}}{\phi_{di}^2} \cdot \frac{y_i - y_q}{\phi_{di}}$$
$$= \left[\mathrm{dsg}(\phi_{ti}, -\beta_t, -\alpha_t) - \mathrm{dsg}(\phi_{ti}, -\beta_t, \alpha_t)\right] \cdot \frac{(z_i - z_{\mathbf{q}})(y_i - y_{\mathbf{q}})}{\phi_{di}[\phi_{di}^2 + (z_i - z_{\mathbf{q}})^2]} \tag{A.22}$$

By substituting Equations (A.9), (A.15) and (A.21) in Equation (A.3), we obtain the analytical derivative for the x position of sensor $\mathbf{s}_i$ to the position $\mathbf{q}$ in the environment. The derivative to the y position can be calculated in the same way, substituting Equations (A.10), (A.16) and (A.22) in Equation (A.4).

Let us now develop the derivatives of membership functions with respect to θ_i:

$$\frac{\partial \left[\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}\right]}{\partial \theta_i} = \left[\frac{\partial \mu_{di}}{\partial \theta_i} \cdot \mu_{pi} \cdot \mu_{ti} + \frac{\partial \mu_{pi}}{\partial \theta_i} \cdot \mu_{di} \cdot \mu_{ti} + \frac{\partial \mu_{ti}}{\partial \theta_i} \cdot \mu_{di} \cdot \mu_{pi}\right] \tag{A.23}$$

For the term $\frac{\partial \mu_{di}}{\partial \theta_i}$, the change of θ_i will not affect the membership functions μ_{di} and μ_{ti}, hence:

$$\frac{\partial \mu_{di}}{\partial \theta_i} = \frac{\partial \mu_{ti}}{\partial \theta_i} = 0 \tag{A.24}$$

For the term $\frac{\partial \mu_{pi}}{\partial \theta_i}$:

$$\begin{aligned}
\frac{\partial \mu_{pi}}{\partial \theta_i} &= \frac{\partial \mu_{pi}}{\partial \phi_{pi}} \cdot \frac{\partial \phi_{pi}}{\partial \theta_i} \\
\frac{\partial \mu_{pi}}{\partial x_i} &= \frac{\partial \left[\frac{1}{1+\exp(-\beta_p(\phi_{pi}+\alpha_p))} - \frac{1}{1+\exp(-\beta_p(\phi_{pi}-\alpha_p))}\right]}{\partial \phi_{pi}} \cdot (-1) \\
&= -\left[\frac{\beta_p \exp\left(-\beta_p(\phi_{pi}+\alpha_p)\right))}{(1+\exp\left(-\beta_p(\phi_{pi}+\alpha_p)\right))^2} - \frac{\beta_p \exp\left(-\beta_p(\phi_{pi}-\alpha_p)\right))}{(1+\exp\left(-\beta_p(\phi_{pi}-\alpha_p)\right))^2}\right] \\
&= \mathrm{dsg}(\phi_{pi}, -\beta_p, \alpha_p) - \mathrm{dsg}(\phi_{pi}, -\beta_p, -\alpha_p)
\end{aligned} \tag{A.25}$$

By substituting Equations (A.24) and (A.25) in Equation (A.23), we obtain the analytical derivative for pan orientation of sensor $\mathbf{s}_i$ to the position $\mathbf{q}$ in the environment.

Now, the developments of the derivatives of membership functions with respect to ξ_i:

$$\frac{\partial \left[\mu_{di} \cdot \mu_{pi} \cdot \mu_{ti}\right]}{\partial \xi_i} = \left[\frac{\partial \mu_{di}}{\partial \xi_i} \cdot \mu_{pi} \cdot \mu_{ti} + \frac{\partial \mu_{pi}}{\partial \xi_i} \cdot \mu_{di} \cdot \mu_{ti} + \frac{\partial \mu_{ti}}{\partial \xi_i} \cdot \mu_{di} \cdot \mu_{pi}\right] \tag{A.26}$$

For the term $\frac{\partial \mu_{di}}{\partial \xi_i}$, the change of ξ_i will not affect the membership functions μ_{di}, and μ_{pi}. Hence:

$$\frac{\partial \mu_{di}}{\partial \xi_i} = \frac{\partial \mu_{pi}}{\partial \xi_i} = 0 \tag{A.27}$$

As for the term $\frac{\partial \mu_{ti}}{\partial \xi_i}$:

$$\begin{aligned}
\frac{\partial \mu_{ti}}{\partial \xi_i} = \frac{\partial \mu_{ti}}{\partial \phi_{ti}} \cdot \frac{\partial \phi_{ti}}{\partial \xi_i} &= \frac{\partial \left[\frac{1}{1+\exp(-\beta_t(\phi_{ti}+\alpha_t))} - \frac{1}{1+\exp(-\beta_t(\phi_{ti}-\alpha_t))}\right]}{\partial \phi_{ti}} \cdot (-1) \\
&= -\left[\frac{\beta_p \exp\left(-\beta_t(\phi_{ti}+\alpha_t)\right))}{(1+\exp\left(-\beta_t(\phi_{ti}+\alpha_t)\right))^2} - \frac{\beta_t \exp\left(-\beta_t(\phi_{ti}-\alpha_t)\right))}{(1+\exp\left(-\beta_t(\phi_{ti}-\alpha_t)\right))^2}\right] \\
&= \mathrm{dsg}(\phi_{ti}, -\beta_t, \alpha_t) - \mathrm{dsg}(\phi_{ti}, -\beta_t, -\alpha_t)
\end{aligned} \tag{A.28}$$

By substituting Equations (A.27) and (A.28) in Equation (A.26), we obtain the analytical derivative for the tilt orientation of sensor $\mathbf{s}_i$ to the position $\mathbf{q}$ in the environment.

Appendix B

List of Publications

B.1 Journal

- V. Akbarzadeh, C. Gagné, M. Parizeau, M. Argany, M.A. Mostafavi, "Probabilistic Sensing Model for Sensor Placement Optimization Based on Line-of-Sight Coverage," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 2, pp. 293-303, Feb. 2013.

- V. Akbarzadeh, J.-C. Lévesque, C. Gagné, and M. Parizeau, "Efficient Sensor Placement Optimization Using Gradient Descent and Probabilistic Coverage," *Sensors*, vol. 14, no. 8, pp. 15525-15552, Aug. 2014.

B.2 Conference

- V. Akbarzadeh, A. Ko, C. Gagné, M. Parizeau, "Topography-aware sensor deployment optimization with CMA-ES." *Parallel Problem Solving from Nature, PPSN XI,* pp. 141-150, 2010.

- V. Akbarzadeh, C. Gagné, M. Parizeau, M.A. Mostafavi, "Black-box optimization of sensor placement with elevation maps and probabilistic sensing models," *IEEE International Symposium on Robotic and Sensors Environments (ROSE),* pp. 89-94, 2011.

- V. Akbarzadeh, C. Gagné, M. Parizeau, "Target Trajectory Prediction in PTZ Camera Networks," *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW),* pp. 816-822, 2013.

- V. Akbarzadeh, C. Gagné, M. Parizeau, "Kernel Density Estimation for Target Trajectory Prediction," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

- V. Akbarzadeh, C. Gagné, M. Parizeau, "Sensor Control for Temporal Coverage Optimization," *IEEE Congress on Evolutionary Computation (CEC),* 2016.

Bibliography

[1] J. Adriaens, S. Megerian, and M. Potkonjak. Optimal worst-case coverage of directional field-of-view sensor networks. In *3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, volume 1, pages 336–345, Sept 2006.

[2] P. K. Agarwal, E. Ezra, and S. K. Ganjugunte. Efficient sensor placement for surveillance problems. In *Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS '09, pages 301–314, 2009.

[3] N. Ahmed, S. S. Kanhere, and J. Sanjay. Probabilistic coverage in wireless sensor networks. In *The IEEE Conference on Local Computer Networks*, pages 8 pp.–681, Nov 2005.

[4] J. Ai and A. A. Abouzeid. Coverage by directional sensors in randomly deployed wireless sensor networks. *Journal of Combinatorial Optimization*, 11(1):21–41, 2006.

[5] V. Akbarzadeh, C. Gagné, M. Parizeau, M. Argany, and M.A. Mostafavi. Probabilistic sensing model for sensor placement optimization based on line-of-sight coverage. *IEEE Transactions on Instrumentation and Measurement*, 62(2):293–303, Feb 2013.

[6] V. Akbarzadeh, A.-R. Ko, C. Gagné, and M. Parizeau. Topography-aware sensor deployment optimization with CMA-ES. In *Parallel Problem Solving from Nature, PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 141–150. Springer Berlin Heidelberg, 2010.

[7] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad hoc networks*, 2(4):351–367, 2004.

[8] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[9] M. Argany, M.A Mostafavi, and F. Karimipour. Voronoi-based approaches for geosensor networks coverage determination and optimisation: A survey. In *Voronoi Diagrams in Science and Engineering (ISVD), 2010 International Symposium on*, pages 115–123, June 2010.

[10] X. Bai, S. Kumar, D. Xuan, Z. Yun, and T. H. Lai. Deploying wireless sensors to achieve both coverage and connectivity. In *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '06, pages 131–142, New York, NY, USA, 2006. ACM.

[11] C.R. Baker, K. Armijo, S. Belka, M. Benhabib, V. Bhargava, N. Burkhart, A. Der Minassians, G. Dervisoglu, L. Gutnik, M.B. Haick, C. Ho, M. Koplow, J. Mangold, S. Robinson, M. Rosa, M. Schwartz, C. Sims, H. Stoffregen, A. Waterbury, E.S. Leland, T. Pering, and P.K. Wright. Wireless sensor networks for home health care. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 2, pages 832–837, May 2007.

[12] F. I. Bashir, A. A. Khokhar, and D. Schonfeld. Segmented trajectory based indexing and retrieval of video data. In *International Conference on Image Processing*, volume 2, pages II–623–6 vol.3, Sept 2003.

[13] M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3601–3606 vol.4, 2002.

[14] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Resarch*, 13:281–305, February 2012.

[15] C.M. Bishop. *Pattern recognition and machine learning*. Springer, New York, 2006.

[16] R.R. Brooks, C. Griffin, and D.S. Friedlander. Self-organized distributed sensor network entity tracking. *International Journal of High Performance Computing Applications*, 16(3):207–219, 2002.

[17] D. Buzan, S. Sclaroff, and G. Kollios. Extraction and clustering of motion trajectories in video. In *International Conference on Pattern Recognition*, volume 2, pages 521–524 Vol.2, Aug 2004.

[18] Y. Cai, W. Lou, and M. Li. Cover set problem in directional sensor networks. In *Future Generation Communication and Networking*, volume 1, pages 274–278, Dec 2007.

[19] M. Cardei, M. T. Thai, Y. Li, and W. Wu. Energy-efficient target coverage in wireless sensor networks. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1976–1984 vol. 3, March 2005.

[20] C.-S. Chen, C. F. Eick, and N. J. Rizk. Mining spatial trajectories using non-parametric density functions. In *Machine Learning and Data Mining in Pattern Recognition*, volume 6871 of *Lecture Notes in Computer Science*, pages 496–510. Springer Berlin Heidelberg, 2011.

[21] H. Chen, H. Wu, and N.-F. Tzeng. Grid-based approach for working node selection in wireless sensor networks. In *IEEE International Conference on Communications*, volume 6, pages 3673–3678 Vol.6, June 2004.

[22] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *International Conference on Management of Data*, SIGMOD '05, pages 491–502, New York, NY, USA, 2005. ACM.

[23] U.-R. Chen, B.-S. Chiou, J.-M. Chen, and W. Lin. An adjustable target coverage method in directional sensor networks. In *IEEE Asia-Pacific Services Computing Conference*, pages 174–180. IEEE, 2008.

[24] S.-Y. Cheung and P. P. Varaiya. *Traffic surveillance by wireless sensor networks: Final report*. California PATH Program, Institute of Transportation Studies, University of California at Berkeley, 2007.

[25] P. L. Chiu and F.Y.-S. Lin. A simulated annealing algorithm to support the sensor placement for target location. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 867–870 Vol.2, May 2004.

[26] C.-Y. Chong, F. Zhao, S. Mori, and S. Kumar. Distributed tracking in wireless ad hoc sensor networks. In *Proceedings of the Sixth International Conference of Information Fusion*, volume 1, pages 431–438, July 2003.

[27] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, April 2004.

[28] N.A. Cressie. *Statistics for Spatial Data*. Wiley-Interscience, 1991.

[29] W. W. Dargie and C. Poellabauer. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley & Sons, 2010.

[30] S. S. Dhillon and K. Chakrabarty. Sensor placement for effective coverage and surveillance in distributed sensor networks. In *Wireless Communications and Networking*, volume 3, pages 1609–1614 vol.3, March 2003.

[31] S. M. El-kader and B. M. El-Basioni. Precision farming solution in egypt using the wireless sensor network technology. *Egyptian Informatics Journal*, 14(3):221–233, 2013.

[32] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[33] J. A. Fuemmeler and V. V. Veeravalli. Smart sleeping policies for energy efficient tracking in sensor networks. *IEEE Transactions on Signal Processing*, 56(5):2091–2101, May 2008.

[34] S. Garcia, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *Journal of Heuristics*, 6(15):617–644, 2009.

[35] N. Hansen. The CMA evolution strategy: A tutorial. Available online at `http://www.lri.fr/~hansen/cmatutorial.pdf` (last checked Sept 13th, 2015), 2011.

[36] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159 – 195, 2001.

[37] M. Hefeeda and H. Ahmadi. Energy-efficient protocol for deterministic and probabilistic coverage in sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):579–593, May 2010.

[38] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li. Research challenges and applications for underwater sensor networking. In *Wireless Communications and Networking Conference*, volume 1, pages 228–235, April 2006.

[39] E. Horster and R. Lienhart. Approximating optimal visual sensor placement. In *IEEE International Conference on Multimedia and Expo*, pages 1257–1260, July 2006.

[40] A. Howard, M. J. Matarić, and G. S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Distributed Autonomous Robotic Systems 5*, pages 299–308. Springer Japan, 2002.

[41] C.-F. Huang and Y.-C. Tseng. The coverage problem in a wireless sensor network. In *International Conference on Wireless Sensor Networks and Applications*, WSNA '03, pages 115–121, New York, NY, USA, 2003. ACM.

[42] B. Jiang, B. Ravindran, and H. Cho. Probability-based prediction and sleep scheduling for energy-efficient target tracking in sensor networks. *IEEE Transactions on Mobile Computing*, 12(4):735–747, April 2013.

[43] M. P. Johnson and A. Bar-Noy. Pan and scan: Configuring cameras for coverage. In *IEEE International Conference on Computer Communications*, pages 1071–1079, April 2011.

[44] D. Jourdan and O. L. Weck. Layout optimization for a wireless sensor network using a multi-objective genetic algorithm. In *IEEE Vehicular Technology Conference*, volume 5, pages 2466–2470 Vol.5, 2004.

[45] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 285–289, New York, NY, USA, 2000. ACM.

[46] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[47] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[48] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Resarch*, 9:235–284, June 2008.

[49] S. Kumar, T. H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Annual International Conference on Mobile Computing and Networking*, MobiCom '05, pages 284–298, New York, NY, USA, 2005. ACM.

[50] R. Laxhammar, G. Falkman, and E. Sviestins. Anomaly detection in sea traffic - a comparison of the gaussian mixture model and the kernel density estimator. In *International Conference on Information Fusion*, pages 756–763, July 2009.

[51] M. Li and Y. Liu. Underground structure monitoring with wireless sensor networks. In *International Symposium on Information Processing in Sensor Networks*, pages 69–78, April 2007.

[52] Y.-T. Lin, K. K. Saluja, and S. Megerian. Adaptive cost efficient deployment strategy for homogeneous wireless camera sensors. *Ad Hoc Networks*, 9(5):713 – 726, 2011.

[53] B. Liu and D. Towsley. A study of the coverage of large-scale sensor networks. In *IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 475–483, Oct 2004.

[54] J. P. Lynch. An overview of wireless structural health monitoring for civil structures. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 365(1851):345–372, 2007.

[55] H. Ma, X. Zhang, and A. Ming. A coverage-enhancing method for 3d directional sensor networks. In *IEEE International Conference on Computer Communications*, pages 2791–2795, April 2009.

[56] M. Marks. A survey of multi-objective deployment in wireless sensor networks. *Journal of telecommunications and information technology*, 2010(3):36–41, 2010.

[57] S. Megerian, F. Koushanfar, G. Qu, G. Veltri, and M. Potkonjak. Exposure in wireless sensor networks: Theory and practical solutions. *Wirel. Netw.*, 8(5):443–454, September 2002.

[58] O. Mersmann, M. Preuss, and H. Trautmann. Benchmarking evolutionary algorithms: Towards exploratory landscape analysis. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*, PPSN'10, pages 73–82, Berlin, Heidelberg, 2010. Springer-Verlag.

[59] A. Mittal and L. S. Davis. Visibility analysis and sensor planning in dynamic environments. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision*, volume 3021 of *Lecture Notes in Computer Science*, pages 175–189. Springer Berlin Heidelberg, 2004.

[60] R. Mulligan and H. M. Ammari. Coverage in wireless sensor networks: a survey. *Network Protocols and Algorithms*, 2(2):27–53, 2010.

[61] M. Nanni and D. Pedreschi. Time-focused clustering of trajectories of moving objects. *J. Intell. Inf. Syst.*, 27(3):267–289, November 2006.

[62] S. Olariu and J. V. Nickerson. Protecting with sensor networks: perimeters and axes. In *IEEE Military Communications Conference*, pages 1780–1786 Vol. 3, Oct 2005.

[63] J. B. Oliva. Anomaly detection and modeling of trajectories. Master's thesis, School of Computer Science, Carnegie Mellon University, 8 2012.

[64] N. M. Oliver, B. Rosario, and A. P. Pentland. A bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):831–843, Aug 2000.

[65] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.

[66] J. Peng, J. Jingqi, W. Chengdong, and H. Nan. A coverage detection and re-deployment algorithm in 3d directional sensor networks. In *Chinese Control and Decision Conference*, pages 1137–1142. IEEE, 2015.

[67] C. Piciarelli and G.L. Foresti. On-line trajectory clustering for anomalous events detection. *Pattern Recognition Letters*, 27(15):1835 – 1842, 2006. Vision for Crime Detection and Prevention.

[68] D. Pompili, T. Melodia, and I. F. Akyildiz. Deployment analysis in underwater acoustic wireless sensor networks. In *Proceedings of the 1st ACM International Workshop on Underwater Networks*, WUWNet '06, pages 48–55, New York, NY, USA, 2006. ACM.

[69] V. Potdar, A. Sharif, and E. Chang. Wireless sensor networks: A survey. In *International Conference on Advanced Information Networking and Applications Workshops*, pages 636–641. IEEE, 2009.

[70] R. Ros and N. Hansen. A simple modification in CMA-ES achieving linear time and space complexity. In *International Conference on Parallel Problem Solving from Nature*, pages 296–305, Berlin, Heidelberg, 2008. Springer-Verlag.

[71] D.E. Rumelhart and J.L. McClelland. *Parallel distributed processing: Psychological and biological models*, volume 2. The MIT press, 1986.

[72] M. Schwager, B. J. Julian, M. Angermann, and D. Rus. Eyes in the sky: Decentralized control for the deployment of robotic camera networks. *Proceedings of the IEEE Journal*, 99(9):1541–1561, Sept 2011.

[73] K. Sohraby, D. Minoli, and T. Znati. *Wireless sensor networks: technology, protocols, and applications*. John Wiley & Sons, 2007.

[74] C. Sung, D. Feldman, and D. Rus. Trajectory clustering for motion prediction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1547–1552, Oct 2012.

[75] H. R. Topcuoglu, M. Ermis, and M. Sifyan. Positioning and utilizing sensors on a 3-d terrain part II, solving with a hybrid evolutionary algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 41(4):470–480, July 2011.

[76] D. Vasquez, T. Fraichard, and C. Laugier. Growing hidden Markov models: An incremental tool for learning and predicting human and vehicle motion. *International Journal of Robotics Research*, 28(11-12):1486–1506, 2009.

[77] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *International Conference on Data Engineering*, pages 673–684, 2002.

[78] G. Wang, G. Cao, P. Berman, and T. F. La Porta. Bidding protocols for deploying mobile sensors. *IEEE Transactions on Mobile Computing*, 6(5):563–576, May 2007.

[79] J. Wang, C. Niu, and R. Shen. Priority-based target coverage in directional sensor networks using a genetic algorithm. *Comput. Math. Appl.*, 57(11-12):1915–1922, June 2009.

[80] X. Wang, E. Grimson, G.-W. Ng, and K. T. Ma. Trajectory analysis and semantic region modeling using a nonparametric bayesian model. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.

[81] X. Wang, S. Wang, and J.-J. Ma. An improved co-evolutionary particle swarm optimization for wireless sensor networks with dynamic deployment. *Sensors*, 7(3):354–370, 2007.

[82] Y.-C. Wang and Y.-C. Tseng. Distributed deployment schemes for mobile wireless sensor networks to ensure multilevel coverage. *IEEE Transactions on Parallel and Distributed Systems*, 19(9):1280–1294, Sept 2008.

[83] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, March 2006.

[84] M. Worboys and M. Duckham. *GIS: A Computing Perspective, 2Nd Edition*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

[85] T. Xiang and S. Gong. Beyond tracking: Modelling activity and understanding behaviour. *International Journal of Computer Vision*, 67(1):21–51, April 2006.

[86] Y. Xu and W.-C. Lee. On localized prediction for power efficient object tracking in sensor networks. In *International Conference on Distributed Computing Systems Workshops*, pages 434–439, May 2003.

[87] Y. Xu, J. Winter, and W.-C. Lee. Dual prediction-based reporting for object tracking sensor networks. In *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 154–163, Aug 2004.

[88] Y. Xu, J. Winter, and W.-C. Lee. Prediction-based strategies for energy saving in object tracking sensor networks. In *IEEE International Conference on Mobile Data Management*, pages 346–357, 2004.

[89] W.-L. Yeow, C.-K. Tham, and W.-C. Wong. Energy efficient multiple target tracking in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 56(2):918–928, March 2007.

[90] T.-H. Yi, H.-N. Li, and M. Gu. Optimal sensor placement for health monitoring of high-rise structure based on genetic algorithm. *Mathematical Problems in Engineering*, 2011, 2011.

[91] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networking*, 52(12):2292–2330, August 2008.

[92] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in zebranet. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 227–238, New York, NY, USA, 2004. ACM.

[93] Z. Zhang, K. Huang, and T. Tan. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *International Conference on Pattern Recognition*, volume 3, pages 1135–1138, 2006.

[94] J. Zhao, S.S. Cheung, and T. Nguyen. Optimal camera network configurations for visual tagging. *Selected Topics in Signal Processing*, 2(4):464–479, Aug 2008.

[95] B. Zhou, X. Wang, and X. Tang. Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2871–2878, 2012.

[96] G.-D. Zhou, T.-H. Yi, and H.-N. Li. Sensor placement optimization in structural health monitoring using cluster-in-cluster firefly algorithm. *Advances in Structural Engineering*, 17(8):1103, 2014.

[97] G.-D. Zhou, T.-H. Yi, and H.-N. Li. Wireless sensor placement for bridge health monitoring using a generalized genetic algorithm. *International Journal of Structural Stability and Dynamics*, 14(05):1440011, 2014.

[98] C. Zhu, C. Zheng, L. Shu, and G. Han. A survey on coverage and connectivity issues in wireless sensor networks. *Journal of Network and Computer Applications*, 35(2):619 – 632, 2012. Simulation and Testbeds.