ALGORITHMS FOR DISTRIBUTED AND MOBILE SENSING

İbrahim Volkan İşler

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2004

Kostas Daniilidis and Sampath Kannan
Supervisor of Dissertation

Benjamin Pierce
Graduate Group Chairperson

To my love, Hilal.

Acknowledgements

Those who shed their light on me throughout my studies:

My advisors Kostas Daniilidis and Sampath Kannan for their support and guidance. For trusting me. For all the effort they put in me.

Sanjeev Khanna for being a true mentor.

I have had the privilege of working with George Pappas, Jianbo Shi and Camillo Taylor. I learned a lot from them.

Vijay Kumar, the head of the GRASP Lab, for keeping an eye on us.

I have enjoyed numerous discussions with Jean Gallier, Sudipto Guha, Max Mintz, Val Tannen. Thank you for always being there.

Calin for being the best office mate and John for being the best conference buddy.

The Turkish Mafia: Yelkan, Tolga and Gurhan. Franjo and other friends at CIS. Ameesh, Mirko and the next generation of GRASP for carrying the flag. Mike Felker, Gail Shannon and other CIS staff who work hard to make our lives easy.

My parents and my sister for their continuous support and endless love.

Hilal Nakiboglu for taking the path of life with me and holding my hand. Kuzu, sensiz ne yapardim bilemiyorum.

And the source of that light..

ABSTRACT

ALGORITHMS FOR DISTRIBUTED AND MOBILE SENSING

İbrahim Volkan İşler

Kostas Daniilidis and Sampath Kannan

Sensing remote, complex and large environments is an important task that arises in diverse applications including planetary exploration, monitoring forest fires and the surveillance of large factories. Currently, automation of such sensing tasks in complex environments is achieved either by deploying many stationary sensors to the environment, or by mounting a sensor on a mobile device and using the device to sense the environment.

The Eighties and Nineties witnessed tremendous advances in both distributed and mobile sensing technologies. To take advantage of these technologies, it is crucial to design algorithms to perform sensing tasks in an autonomous fashion. In this dissertation, we study four fundamental sensing problems that arise in sensing complex environments with distributed and mobile systems.

For mobile sensing systems we study exploration and pursuit-evasion problems. In the *exploration* problem, the goal is to design a strategy for a mobile robot so that the robot sees every point in an unknown environment as quickly as possible. In the *pursuit-evasion* problem, the goal is to design a strategy for a pursuer to capture an adversarial evader.

For distributed sensing systems we study placement and assignment problems. In the *placement* problem, the goal is to place sensors to an environment so that every point in the environment is in the range of at least one sensor. The *assignment* problem deals with the issue of assigning targets to sensors in a network, so that overall error in estimating the position of the targets is minimized.

We present algorithms to perform these sensing tasks in an efficient fashion. Performance guarantees of the algorithms are mathematically proven and evaluated by simulations.

Contents

Acknowledgements **iv**

1 Introduction **1**

1.1 Modern Sensing Systems . . . 3

1.1.1 Sensor networks . . . 3

1.1.2 Mobile robots . . . 3

1.1.3 Applications of sensing systems . . . 4

1.2 Sensing Problems for Mobile Robots . . . 5

1.2.1 Exploration . . . 5

1.2.2 Pursuit-evasion . . . 6

1.3 Sensing Problems for Sensor Networks . . . 8

1.3.1 Sensor placement . . . 8

1.3.2 Sensor assignment . . . 9

1.4 Methodology . . . 11

1.4.1 Challenges . . . 11

1.4.2 Solution philosophy . . . 12

1.5 Our contribution . . . 14

1.5.1 Exploration . . . 15

1.5.2 Sensor placement . . . 16

1.5.3 Sensor assignment . . . 17

1.5.4 Pursuit-evasion . . . 18

1.6 Outline 19

2 Local Exploration **20**

2.1 The Exploration Problem in Robotics 22

2.2 Online Algorithms and Competitive Analysis 23

2.2.1 Competitive analysis in robot exploration 23

2.2.2 New problem statement 24

2.3 Competitive Algorithms 26

2.3.1 Algorithm UNIREC 26

2.3.2 Algorithm EXPREC 28

2.3.3 The competitive ratio of EXPREC 28

2.4 Probabilistic framework 30

2.5 Simulation Results 35

2.5.1 The underlying distribution is known 36

2.5.2 When the beliefs are wrong 38

2.6 Concluding Remarks 39

3 Sensor placement **41**

3.1 Sampling Based Sensor Placement 42

3.2 Problem Statement 44

3.2.1 VC-dimension and set-cover 44

3.2.2 Visibility and set-cover 46

3.2.3 Related work on VC-dimension 47

3.3 Results On Planar Configurations 48

3.3.1 2DCIRCLE 48

3.3.2 2DCONVEX 50

3.4 Results on 3D Configurations 54

3.4.1 Upper bound 54

3.4.2 Lower bound 55

3.5 Sensor Deployment by Sampling 56

3.6 Placing Sensors on a Circle 58

3.6.1 Visual events 59

3.6.2 The Algorithm PlaceCircleGuards 61

3.6.3 Analysis 62

3.7 Concluding Remarks 63

4 Sensor Assignment 64

4.1 Related Work 65

4.2 The Focus of Attention Problem 67

4.2.1 Problem definition 67

4.2.2 Cameras on a line 68

4.2.3 Range sensors on a circle 80

4.2.4 Discussion: Universal Placement 86

4.2.5 Arbitrary sensor assignment 86

4.3 Error Models 89

4.4 Concluding Remarks 94

5 Pursuit-Evasion Games in Complex Environments 97

5.1 Pursuit-Evasion Problems: The Global Picture 99

5.2 Pursuit-Evasion on Graphs 100

5.3 Pursuit-Evasion in Geometric Environments 103

5.3.1 Capturing the evader 103

5.3.2 Locating the evader 105

6 Pursuit-Evasion on Graphs 107

6.1 Preliminaries 110

6.2 A Winning Strategy with Two Hunters 112

6.3 Hunter-win Graphs 115

6.3.1 Characterization of hunter-win graphs against reactive rabbits 117

6.3.2 Characterization of hunter-win graphs against general rabbits 118
6.4 Global Visibility and Dismantlable Graphs 123
6.4.1 Hunter strategy for dismantlable graphs 127
6.4.2 Extension to non-dismantlable graphs 129
6.5 Extending the Rabbit's Visibility 129
6.6 Capture Time on Hunter-win Graphs 131
6.6.1 A strategy for chordal graphs 132
6.7 Concluding Remarks 134

7 Pursuit-Evasion in Geometric Environments 135
7.1 Randomized Strategies 137
7.1.1 Preliminaries 138
7.2 Locating the Evader 140
7.2.1 The pursuer strategy 141
7.2.2 Lower bounds 144
7.3 Capturing the Evader with Two Pursuers 144
7.3.1 The lion and man problem 146
7.3.2 Lion's strategy 146
7.3.3 The strategy to capture the evader 147
7.3.4 Strategy of Pursuer p_1 149
7.3.5 Strategy of Pursuer p_2 150
7.3.6 Properties of Pursuer p_1's strategy 150
7.3.7 Expected time to capture 152
7.3.8 Capturing the evader with a single pursuer 153
7.3.9 Polygonal rooms with a door 154
7.4 Concluding Remarks 155

8 Conclusion 157
8.1 Summary of Contributions 160

8.2 Future Research Directions 163
8.3 Final Remarks 166

List of Tables

2.1 The input x is the robot's position, y is the location of the corner, v is robot's speed, and τ is the time it takes to make a reconstruction. The command *Reconstruct* denotes the operation of an omnidirectional range acquisition and $circle(x, y)$ is the circle that passes through x and y and has a diameter $dist(x, y)$. Algorithm UNIREC has a competitive ratio of π and algorithm EXPREC has a competitive ratio of 2.23. 27

2.2 The results when the underlying distributions match the beliefs about the distribution. 1000 samples were drawn from the distributions in figure 2.5 (numbers in the first column reference Figure 2.5). Rest of the columns present the average time to see the next edge for the four algorithms presented in this chapter. 36

2.3 The results when the underlying distributions match the beliefs about the distribution. 1000 samples were drawn from the distributions in figure 2.5 (column1). Rest of the columns present the results for running the four algorithms presented in this chapter. 37

2.4 Robot thinks the world is $\mathcal{N}(60, 5)$ but in fact the samples are drawn from uniformly from $[0, \pi]$. 39

3.1 Algorithm PlaceCircleGuards. 62

4.1 A 2-approximation algorithm for assigning cameras on a line to targets. 71

4.2 A (1+ϵ)-approximation algorithm for assigning equidistant cameras on a line to targets. 76
4.3 Greedy algorithm for selecting valid assignments. 87
4.4 A 2-local algorithm for selecting valid assignments. A solution is called a *valid solution* if no two of assignments in the solution contain the same target or camera. 87

5.1 Number of pursuers for pursuit on graphs: n means that the number can be arbitrarily large. 103
5.2 Locating the Evader in a polygon: $\Theta(\sqrt{h} + \log n)$ result can be improved to $\Theta(\sqrt{h} + 1)$ using the result in Section 7.2. 105
5.3 Capturing the Evader in a Polygon. 106

List of Figures

1.1 A search-and-rescue scenario 7

1.2 An object-inspection setup 9

1.3 In a network of active cameras, targets may be assigned to pairs of cameras so that each pair tracks its corresponding target. 10

1.4 Our methodology is based on a two-level abstraction. In this thesis, we focus on the higher level and present distributed and sensing algorithms that explicitly deal with the three sources of complexity inherent in the design of such systems. 14

2.1 The problem description: The robot, located at x, can see the edge E but not E', the next edge. E' makes an angle of θ with the line passing through x and the corner y. The optimal offline strategy is P_1, going directly to the extension of E' when $\theta < \frac{\pi}{2}$ and to go directly to y otherwise. When robot has continuous vision, by following P_2 along the circle whose diameter is $\overline{xy}$, the robot guarantees a competitive ratio of $\frac{\pi}{2}$. 25

2.2 **MDP1** has $2n$ states where $n = \frac{\pi}{2\delta}$ that depends on the sampling parameter δ and $\mathcal{F}$ is the final state. Being in s'_i (resp. s_i) means that the robot is at δi and has just (resp. not) made a reconstruction. . . 32

2.3 **MDP2** has $\frac{n(n+1)}{2}$ states where $n = \frac{\pi}{2\delta}$ that depends on the sampling parameter δ. $\mathcal{F}$ is the final state. Being in state $s_{i,j}$ means that the robot is at $i\delta$ on the circle and the last reconstruction was at $j\delta$. . . 33

2.4 **Left:**The optimal policy for MDP2. The sampling value δ used was 5 degrees, therefore the location i, j in the image above represents the action when the robot is at $i\delta$ and the last reconstruction it made was at $j\delta$. The vertical column is i and the horizontal columns is j. The blue upper right half illustrates the inaccessible states. The green values at the lower right correspond to RECONSTRUCT actions and the red region in between correspond to MOVE action. The distribution P^{θ} is according to the distribution on the **Right**. . . . 35

2.5 The distributions used for experiments: Each bucket represents 5 degrees. **Distribution 1** is generated using a gaussian with mean 60 degrees and variance 5 degrees. **Distribution 2** is uniform between $\frac{\pi}{4}$ and $\frac{3\pi}{4}$. **Distribution 3** is uniform between 0 and π. **Distribution 4** is a bimodal distribution obtained by adding up two gaussians with means $\frac{\pi}{6}$ and $\frac{\pi}{2}$ and a variance of 3 degrees. 35

2.6 Optimal policies for MDP1 and MDP2 for Distribution 3 in figure 2.5. **Left:** The probability of making a reconstruction for MDP1 **Right:** The policy for MDP2 37

2.7 $P^{\theta}(\theta) = \mathcal{N}(60, 5)$, but τ varies according to the values in the horizontal axis. POLICY2 outperforms POLICY1 as τ increases. 38

2.8 **Left:** The samples representing the real state of the world were drawn from $\mathcal{N}(60, 5)$ and the optimal policies were computed for $\mathcal{N}(\mu, 5)$ where μ is the horizontal axis. Each simulation is the average time to see the next edge for 1000 samples. **Right:** Same as left but the samples representing the real state of the world were drawn uniformly from $[0, \frac{\pi}{2}]$. 39

3.1 Interior visibility extends to exterior visibility by turning the polygon inside out. 47

3.2 Points $\{p_1, p_2\}$ can be shattered by four cameras. Each camera is labeled with the subset it can see. In this figure the polygon P is $\triangle ABC$. 49

3.3 Case 1 of Lemma 6: The $\{p_i\}$-cell gets two marks and the $\emptyset$-cell gets one mark. The unbounded $\{p_i\}$-cell is marked $\times$. 51

3.4 The small picture on the left shows a polygon with 5 shattered points $\{p_1, p_2, p_3, p_4, p_5\}$. The figure on the right is a detail showing the intersection pattern of the i-rays. Each label $2, 3, 4, 5$ indicates the cardinality of the subset seen from the region. 53

3.5 57

3.6 p is to the left of p' implies $c \leq a$ and $d \leq b$. 60

4.1 An instance of the Focus of Attention problem where eight cameras are tracking four targets. Two cameras are needed to estimate the position of a target and a camera can not track more than one target. 65

4.2 The Focus of Attention Problem on the line. 69

4.3 A greedy assignment assigns c_1 and c_4 to target t_1 and gets stuck with the pair (c_2, c_3). The optimal assignment in this case is to assign t_1 to (c_1, c_3) and t_2 to (c_2, c_4). 70

4.4 The matchings produced by our algorithm (shown in dotted lines) can be twice as bad as the optimal matching (shown in solid lines) by moving the furthest target to infinity. 72

4.5 **Case (a):** A tracking scenario with targets performing a random walk. **Cases (b)-(d):** Comparison of three algorithms. Histograms of the mean-squared error (MSE) for tracking without reassignment (a), with greedy reassignment (b) and reassignment according to the 2-approximation algorithm (c). 74

4.6 Partitioning the line segment: An edge that connects a camera in block L_i to a camera in block R_j is called an edge of type (i, j). . . . 77

4.7 Assigning two edges of type (i, j) from L_i to R_j. Note that the first two cameras in R_j are already marked – i.e. have already been assigned to a camera in L_k for some $k < i$. 77

4.8 Figure for Lemma 33 78

4.9 Geometric Dilution of Precision (GDOP): Each sensor c_1 and c_2 measures its distance to target with some precision. The error in position estimation is shown for two target locations t_1 and t_2. The error reduces as the angle $\theta = \angle c_1 t_i c_2$ gets closer to $\frac{\pi}{2}$. 81

4.10 Sensors on circle: **Left:**The defective region for sensors c_1 and c_2 is the shaded area defined by arcs $Arc1(c_1, c_2)$ and $Arc2(c_1, c_2)$. **Right:** The defective regions are disjoint. 82

4.11 Three possible cases for the location of target t_2. 83

4.12 **Case (a):** A tracking scenario showing robot and sensor positions. **Cases (b)-(d):** Comparison of three algorithms. Histograms of the mean-squared error (MSE) for tracking without reassignment (a), with greedy reassignment (b) and reassignment according to the 1.42-approximation algorithm (c). The latter reduces both MSE and error variance over the alternative approaches. 85

4.13 **Left:** An instance of FOA. **Right:** The conflict graph for the optimal solution and a 2-local solution: The nodes on the left (resp. right) represent the optimal (resp. 2-local) solution. There is an edge between two nodes if the assignments conflict. In this case, a 2-local solution gives a $\frac{5}{3}$ approximation, showing that the analysis is indeed tight. 88

4.14 **Left:** Simulator snapshot for 2-local assignment trial. **Center, Right:** The number of valid tracks recovered for greedy and 2-local search strategies. In this example, 2-local improved over greedy by on average 15%. 90

4.15 Stereo set-up for depth triangulation given matches 92

4.16 **Left:** Error at X=12.5cm and Z=125cm as a function of vergence angle α ($b = 25, \sigma_d = 14.8\mu, f = 0.6cm$). Cameras verge at this point when $\alpha = 84.2894^o$. **Middle-left:** Error at X=12.5, Z=125 when cameras are verging at this point with changing baseline.$\alpha_l = 180 - tan^{-1}[\frac{Z}{b-X}], \alpha_r = tan^{-1}[\frac{Z}{X}], \sigma_d = 14.8\mu, f = 0.6cm, b \in [30, 100]$. **Middle-right:** Prediction Error for the point X=12.5cm and Z=125cm. Left camera is centered at X=25cm, Z=0cm. The predicted view is at X=baseline, Z=0cm. All cameras verge at the point X=12.5 Z=125cm. $\sigma_d = 14.8\mu, f = 0.6cm$. **Right:** Prediction error along the line X=12.5cm as a function of Z. $\alpha = 84.289^o, b = 25, \sigma_d = 14.8\mu, f = 0.6cm, Z \in [60, 200]$. Cameras verge at Z=125. Each point in the synthetic plot represents 1000 trials. 93

5.1 Pursuit-Evasion on graphs. 101

5.2 Pursuit-Evasion in bounded environments. Visibility refers to the evader's visibility. The circled numbers indicate the number of pursuers known to be sufficient for the corresponding case. An arrow shows a result that carries over to another case. A question mark means that there is no known improvement over the result that carried over. 104

6.1 On this graph, the hunter can not capture the rabbit using a deterministic strategy. 108

6.2 This graph is hunter-win against reactive rabbits but not against general rabbits. 115

6.3 Visualization of the folding procedure for a non-dismantlable graph. The vertices w,v and x are in the residual H. Since there is no edge from w to x, the edges shown with dashed lines can not exist. 123

6.4 This graph is hunter-win against rabbits with local visibility. However, a rabbit with full visibility never gets caught. 125
6.5 Hunter can always stay above the rabbit. The height of a vertex is proportional to its label. 127
6.6 Hunter can make progress every time the rabbit revisits a vertex. . 128
6.7 Every cycle C of a chordal graph has at least two ears. 132
6.8 Chasing the rabbit on chordal graphs. 133

7.1 A single pursuer can not capture an evader using deterministic strategies. 137
7.2 Triangulation of a polygon and its dual tree. 139
7.3 Notation used for Lemma 57. Each vertex of the tree corresponds to a triangle in the triangulation tree. 141
7.4 **Left:** An instance of the simulator showing the triangulation of the environment as well as the hiding location of the evader. **Middle:** The histogram of the number of rounds required to locate evader in 1000 simulations. **Right:** The histogram of the time-steps required to locate evader in 1000 simulations. 143
7.5 For any randomized pursuer strategy, the expected time to capture the evader in this star with hooks is $O(n \cdot diam(P))$. 144
7.6 Lion's strategy 145
7.7 Pocket with respect to c and v 147
7.8 The extended lion's move 149
7.9 A bad case for searching in unknown environments. 156

8.1 Researchers at the Intel Research Laboratory at Berkeley, the College of the Atlantic in Bar Harbor and the University of California at Berkeley deployed wireless sensor networks on Great Duck Island, Maine (http://www.greatduckisland.net/). 157

8.2 Examples of recent commercial robots (a) Roomba vacuum cleaner (b) Packbot exploration robot (c) Patrolbot surveillance robot 158

Chapter 1

Introduction

The desire to extend one's presence and sense distant worlds dwells at the depths of the human spirit. Since their earliest days, humans have tried numerous methods to fulfill this desire. A common method to extend one's presence is to use an agent (e.g. another person) to sense a distant environment. This, of course, requires a medium through which one can communicate with this agent.

So far, most of the effort in this design has concentrated on establishing a communication medium. Earlier methods include using pigeons to carry, or smoke signals to decode messages between two parties. In 1837, Samuel Morse invented the telegraph, marking the beginning of the modern information age. Today, using wireless communication, we can transfer enormous amounts of data through the air and even space.

Even though the search for better communication media still continues, recently part of the scientific community has turned its attention to the other component – the sensing agent. It is desirable to replace humans as sensing agents for various reasons. First of all, it is difficult to deploy them to many environments such as other planets, ocean bottoms, and volcanos. The second reason is best explained by an example. Consider the task of monitoring forest fires. This is a sensing task where we need to observe a very large area and the events we would like to detect happen very infrequently. Even though people have sophisticated sensing capabilities, their sensing range is quite limited. Therefore complete sensory coverage of a large area requires many people to be actively engaged in the task. This is a big waste of resources. Finally, it is difficult for humans to make very precise measurements without using additional equipment. Therefore, they are not sufficient alone for tasks such as map building, where such precise measurements are required.

As a result, scientists and engineers have designed many different sensors which can often overcome the sensing limitations of humans. On one hand, we have sensors that can sense large but simple environments. Examples of such sensors include antennas which listen to the radiation emitted from distant stars, and radars that scan the skies. On the other hand, we have sensors which can obtain detailed measurements but which have limited ranges. The typical example of such a sensor is a standard camera. Currently, sensing expansive and complex environments such as a large factory or a forest presents a considerable challenge. This is because, such environments necessitate taking detailed measurements *and* covering a large area.

In this thesis, we focus on sensing tasks where we would like to sense a large, complex environment. The accomplishment of such tasks becomes useful and necessary in a number of applications. For instance, one might want to cover a building with cameras for security. A forest can be covered with heat sensors to detect fires. Currently, there are two basic methods for performing sensing tasks in complex environments. As demonstrated in the examples above, the first method is to deploy many stationary sensors to cover the environment. The second method is to mount the sensor on a mobile device and use the device to sense the environment. For example, we can mount a camera on a mobile robot, using it to patrol a building and detect intruders.

In the next section, we present an overview of systems for sensing complex environments and typical applications where such systems are used.

1.1 Modern Sensing Systems

At the moment, there are two primary methods for sensing complex environments, sensor networks and mobile robot systems.

1.1.1 Sensor networks

Deploying a large number of sensors to cover a complex environment is a common technique. For example, cameras are commonly installed in hotels, supermarkets and casinos for surveillance purposes. Real-time images from these cameras are gathered in a security office where a single officer can remotely sense the entire building. However, the number of sensors in these systems are limited to the order of hundreds [1]. Sensor-networks are the next-step in the evolution of such systems.

The term sensor-network usually refers to a network of small, inexpensive sensing devices with certain communication and computation capabilities. The motivation for building these devices is primarily an economic one. Since they are inexpensive, it is not cost-prohibitive to deploy thousands of them, say, in a forest and use the underlying network to accomplish sensing tasks such as detecting fires or monitoring wildlife. The technological and scientific issues underlying the development and deployment of such networks are very challenging. Currently, there are many research groups studying these issues (e.g. [61, 30]).

1.1.2 Mobile robots

Another way of achieving remote sensing is to mount sensors on a mobile device such as a robot. The field of robotics witnessed tremendous advances in the last decades. At the moment, robotics technology is in a relatively mature state. Robots have been used in assembly lines for decades. Today, it is possible to purchase a mobile

[1]The Greektown Casino in downtown Detroit, MI, has 1,024 Panasonic cameras (data from panasonic.com).

robot, with basic sensing and motion capabilities, off the shelf. In fact, while this thesis is being written, the rovers Spirit and Opportunity have been roaming on the surface of Mars for months.

Typically robots are used as sensing agents for mapping tasks such as exploring planets, ocean floors and volcanos. However, as the robots continue to become more cost effective, we expect that their use in other applications such as surveillance will become much more widespread in the near future.

1.1.3 Applications of sensing systems

The most typical application of sensing systems considered in this thesis is *surveillance*. We can deploy a sensor network to detect intruders in a building. Alternatively, we can use mobile robots to patrol the building and detect the intruders.

Another application is *object inspection*. We can place cameras on a conveyor belt to inspect products or use a camera mounted on a robot arm for inspection. A similar application is *environment monitoring*. As mentioned before, we can deploy a sensor-network on a forest to detect fires or to monitor wild-life. A network deployed on a bridge can monitor its structural properties.

An emerging application is *location aware computing* where various devices use sensors deployed in an environment such as a campus to localize themselves. Such devices can be used to track the elderly or children; or to collect statistics about the use of facilities. In general, *tracking* objects or people is a general application of sensing systems.

It is worth noting that the use of sensor-networks and mobile robots as sensing agents is not mutually exclusive. For example, we can use a mobile robot to deploy a sensor network or we can use a sensor-network to localize our mobile robots.

1.2 Sensing Problems for Mobile Robots

In this section, we introduce two fundamental problems that arise in design of sensing systems which uses mobile robots. These problems are exploration and pursuit-evasion.

1.2.1 Exploration

For many different sensing tasks, we need to sense every point in an *unknown* environment. If we use a mobile robot as a sensing agent to accomplish this task, we need to solve the *exploration problem*: generate a trajectory for the robot so that the robot senses every point in an unknown environment as quickly as possible.

Solving the exploration problem is an important prerequisite for many different sensing tasks. It is required for almost any mapping [104] task – ranging from building a 2D layout of an apartment to drawing a 3D map of an ocean floor. For object inspection, it is often desirable to sense an object entirely and if the object is unknown, this brings up an exploration problem. Robotic deminers and vacuum-cleaners need to sense every point on the floor or ground and often times they do not have a prior map of the area.

Exploration is related to the path-planning problem in robotics which is the problem of finding a path between two points which avoids obstacles in the environment. Path-planning is a well-studied problem [68]. However, the exploration problem differs from path-planning in two important aspects.

The first aspect is coverage. Most of the time, the robot does not have to visit all points in the environment in order to sense all the points. For example, a robot equipped with an omni-directional camera does not even have to move to see the walls of a convex building. However, visiting all points in the environment may be necessary for some applications such as vacuum-cleaning a floor or demining a field. Even if this is the case, designing an optimal strategy is a difficult problem. As an example, consider an application where we use a robot to build a 3D map of an apartment. Imagine our robot is equipped with a stereo camera pair and suppose all the the rooms in the apartment are convex. In this case, we can represent the apartment by a graph whose vertices correspond to rooms and corridors and whose edges correspond to doors. If this graph is available to us, then the exploration problem reduces to the well-known traveling salesperson problem [71]: Find the shortest tour that visits all the vertices. Unfortunately, finding such a tour is an NP-complete problem even when the input graphs are restricted to be planar [34].

The second aspect is online planning. In most applications, our robot must explore the environment without any prior information about it. For example, suppose we would like use our robots as scouts to build the map of an enemy facility. This is similar to the above problem of mapping an apartment. However, in this scenario, it is unlikely that we have a graph that represents the environment *a priori*. Therefore, our robot must solve the exploration problem in an online fashion and generate its trajectory as the map becomes available. This seems like a hopeless goal: can we find a motion plan to see every point in the environment quickly but without knowing the environment?

1.2.2 Pursuit-evasion

In the exploration problem, usually the environment we would like to sense is static. It does not change during the course of exploration. However, many sensing tasks require sensing a moving object. As an example, consider a search-and-rescue mission where we would like to find a lost wanderer in a park or a desert. To accomplish this mission, it is not enough to sense every point in the desert. Suppose that while we were exploring some region A, the wanderer was located in another region, say C (see Figure 1.1). It is possible that when we move to the next region, say $B \neq C$, our target wanders off to A! Hence, we have to revisit the region A in order to find the target.

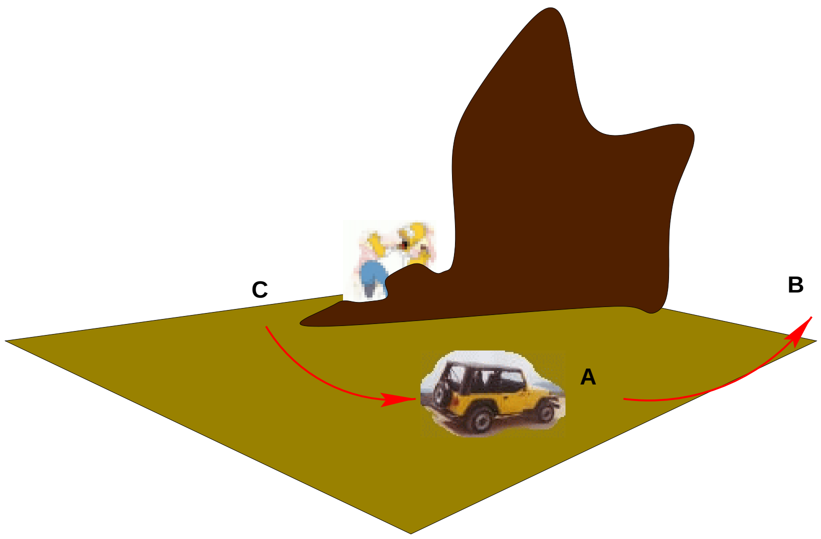


Figure 1.1: A search-and-rescue scenario

Such sensing tasks are best modeled as pursuit-evasion games. In a pursuit-evasion game, one or more pursuers try to capture an evader who, in turn, tries to avoid capture indefinitely. In the search-and-rescue example above, we can model the wanderer as an evader and define capture as sensing or finding the evader. The main ingredient of a pursuit-evasion game is the presence of an adversarial evader who actively avoids capture. Due to this aspect, a pursuit strategy is usually more sophisticated than a search strategy where the target's motion is independent from the pursuer's.

Obtaining pursuit-strategies is important for surveillance applications. For example, we may use a mobile robot to detect intruders before they spend too much time in a building. In these applications, the intruders are indeed adversarial, therefore a pursuit-evasion formulation is natural. In the search-and-rescue setting, even though the victim is not adversarial, a pursuit strategy is still desirable as it guarantees a rescue regardless of the victim's actions.

1.3 Sensing Problems for Sensor Networks

As mentioned before, the alternative to use a mobile robot to accomplish sensing tasks is to deploy many static sensors onto the environment. In this section, we introduce two fundamental problems, placement and assignment, that arise in the design of such systems.

1.3.1 Sensor placement

In Section 1.2.1, we formulated the problem of covering a possibly unknown area with a mobile agent as the exploration problem. In many applications, sensing a point once is not sufficient. We may need to sense all points in the environment at all times. For example, when we use a mobile robot for *surveillance*, if the environment is large, the time between two consecutive visits to sense the same location may also be large. This is undesirable, as it may give a lot of time to the burglar. For a similar reason, detecting forest fires using a few mobile robots may be inadequate.

In such applications, the alternative is to deploy many sensors so that every point in the environment is sensed by at least one sensor at all times. We call the problem of finding such a deployment the *sensor placement* problem.

In most sensor placement instances, it is reasonable to assume that the environment is known. For example, when installing cameras for surveillance, we may already have a floor map of the building. The main issue in such cases is to find the minimum number of sensors and a placement of these sensors to cover a given area. Solving such sensor placement problems is also important for *quality control/object inspection* tasks. For example, a factory which builds automobile parts may design an inspection setup where the object is placed on a controlled stand and the part is inspected by cameras (Figure 1.2). In this scenario, a model of the environment (i.e. a CAD model of the object) would be available.

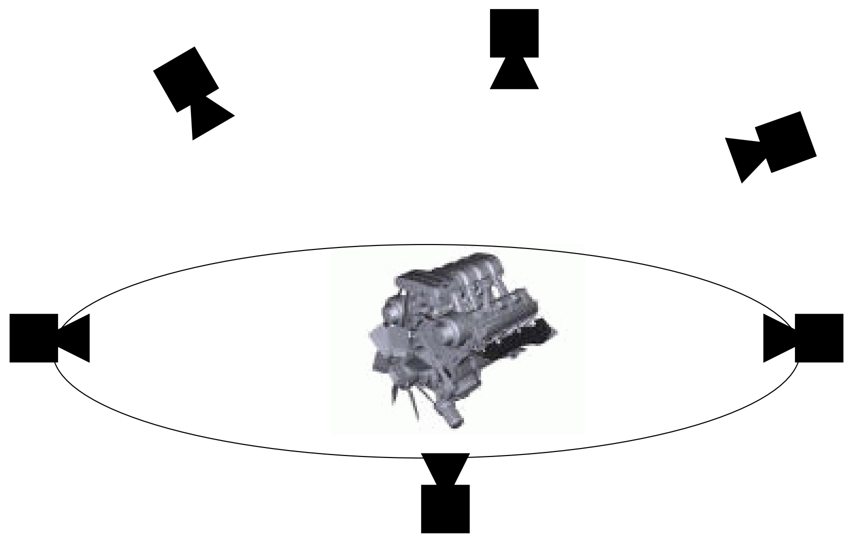


Figure 1.2: An object-inspection setup

In the object inspection example above, we have a lot of control on the environment. We can place the object as well as the sensors precisely. However, there are many applications where the sensors can not be placed precisely. For instance, a sensor network can be deployed on a forest by dropping the sensors from an airplane. In military applications, special cannons can be used to scatter sensors on a foreign territory. In such scenarios where the deployment is subject to unpredictable, large disturbances, sensor may fall onto almost random locations. The main placement question becomes "How many random sensors must be placed to ensure coverage of the environment?"

1.3.2 Sensor assignment

Once a sensor network has been deployed, one of its primary uses is tracking. In an airport, we may want to track passengers to monitor suspicious activity. However, target tracking need not be adversarial. For example, a sensor network can track mobile robots so that the mobile robots can localize themselves. For location aware computing, devices may query our sensors for measurements.

In some of these applications, we have to explicitly assign sensors to targets to

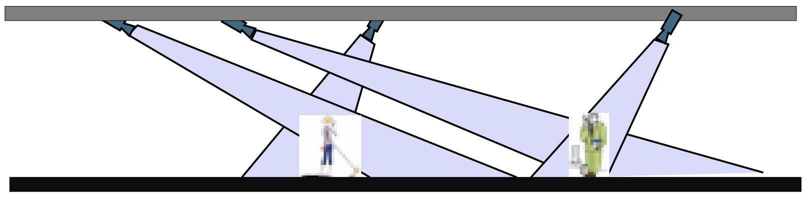


Figure 1.3: In a network of active cameras, targets may be assigned to pairs of cameras so that each pair tracks its corresponding target.

be tracked. For example, if our sensor network is composed of active cameras, the cameras may have to turn towards and zoom onto the target, so that the target is at the center of the image (Figure 1.3). In case of a sensor network with passive sensors, we may want to turn on only a few sensors to save energy. For location-aware computing and localization applications, the devices or robots may be assigned to specific sensors to minimize the communication load across the network. These are all instances of the *sensor assignment problem* where the sensors are explicitly assigned to targets in order to track them.

In most applications, a typical device used for sensing can not estimate the position of a target by itself. For example, at least two cameras are needed to estimate the position of a target. In addition, a single sensor may not always participate in tracking more than one target. In the camera example, if a camera zooms on to the target, there can be at most one target within the field of view of the camera.

For most sensors, the error in estimating the position of the targets is directly dependent on the location of the targets. Therefore, the main objective in the sensor assignment problem is to come up with a sensor assignment scheme so that the overall error in estimating the position of the targets is minimized. As we will see in Chapter 4, this is a challenging combinatorial problem.

1.4 Methodology

In this section, we first discuss some of the challenges in solving sensing problems introduced in the previous section. Afterwards, we describe our approach for solving these problems.

1.4.1 Challenges

In this thesis, we study four fundamental sensing problems: exploration, pursuit-evasion, sensor placement and assignment. In order to design efficient algorithms to solve these problems, we must deal with the following sources of complexity.

Environment complexity: The world we live in has a complex geometric structure. Quite often, designing sensing algorithms in complex environments is a difficult task. Consider the following problem: Given a polygon (which represents an environment), place a minimum number of omnidirectional cameras so that the entire boundary of the polygon is covered by the cameras. That is, any point on the boundary must be visible from at least one camera.

Now suppose that the polygon is convex. In this case, we would need only one camera, regardless of the actual shape of the polygon. However, as the shape of the polygon and hence, the structure of the environment, gets more complicated, finding the minimum number of cameras becomes a difficult problem. In fact, it is known that [83], the general problem is NP-hard, even if the input polygons do not contain any obstacles inside.

In general, in order to obtain efficient algorithms for solving sensing problems we need methods to deal with the complexity of the environment.

Sensing complexity: For many sensing tasks, we must design algorithms which much process the set of points sensed from a particular location in the environment. Let us revisit the camera placement problem discussed above and call the set of points visible from a point p, $V(p)$. To design an efficient algorithm for the placement problem, we must design algorithms that consider the sets $V(p)$ for all p. The structure of these sets can be quite complex and this complexity makes sensing planning problems challenging.

As an example, consider the exploration problem. Suppose the environment is represented by a simple polygon and does not contain any obstacles. First, consider the exploration problem when we have a robot equipped with only a touch sensor. In this case, the problem is quite easy. The robot can start from an arbitrary point on the boundary and explore the polygon by following the boundary. Next, imagine that our robot is equipped with a camera. For this robot, exploring the environment optimally is a much more challenging problem. As illustrated in these scenarios, even when the underlying environments are the same, the sensing complexity has a direct impact on the difficulty of the sensing problems.

Number of agents: In order to sense large, complex environments quite often a single sensing agent is not sufficient, especially if the sensors are static. In case of mobile robots, usually sensing tasks can be performed more efficiently by using teams of robots. Unfortunately, the problem formulations for these multi-agent sensing problems are much more complex then the single agent version – making the design of algorithms for such scenarios more challenging.

For example, in sensor-placement problem, we can easily find the best positions for a few cameras. However, as the number of cameras increases, the combinatorial complexity of the problem becomes a major obstacle in designing efficient algorithms.

1.4.2 Solution philosophy

In this thesis we study planning problems that arise in sensing large, complex environments. In particular, we study the exploration and pursuit-evasion problems for mobile robots (introduced in Section 1.2) and planning and assignment problems for sensor networks (introduced in Section 1.3). We present efficient algorithms for these problems which addresses the three sources of complexity discussed in the previous section: environment complexity, sensing complexity and combinatorial complexity due to the large number of agents.

One approach for solving these problems is to study them in a bottom-up fashion. This method starts with a detailed agent model. In case of networks of cameras, a detailed model would address issues such as errors due to calibration, limitations due to field-of-view constraints, uncertainty in sensing due to sub-pixel errors. For robots, a detailed model could include forces applied to the motors, uncertainty due to odometry, and non-holonomic constraints.

The next step in the bottom-up approach is to design algorithms for these complex agent models that addresses the three sources of complexity presented in Section 1.4.1. The advantage of such a bottom-up approach is that it is very realistic and the problem formulations are closer to the real life scenarios. Unfortunately, solving the problems that arise from these formulations is often very difficult. Hence, even though there are approaches in the literature to solve these problems in this fashion, usually these solutions are based on heuristics without any guarantees about the quality of the solution.

In this thesis, we adopt a philosophy based on a two-level abstraction. Instead of focusing on a particular, complex sensing-agent model, we choose a simple and generic model. For example, we assume that robots can be represented as points whose motion is not subject to any differential constraints. Similarly, we assume that the cameras are omnidirectional without any restrictions on their field of view.

Our approach is illustrated in Figure 1.4. At the higher level, the agent model is simple but we explicitly deal with the three issues that arise in the design of distributed and sensing mobile systems for complex environments. Device specific issues are dealt at a lower layer where the agent model is sophisticated but none of issues related to the environment complexity is studied.

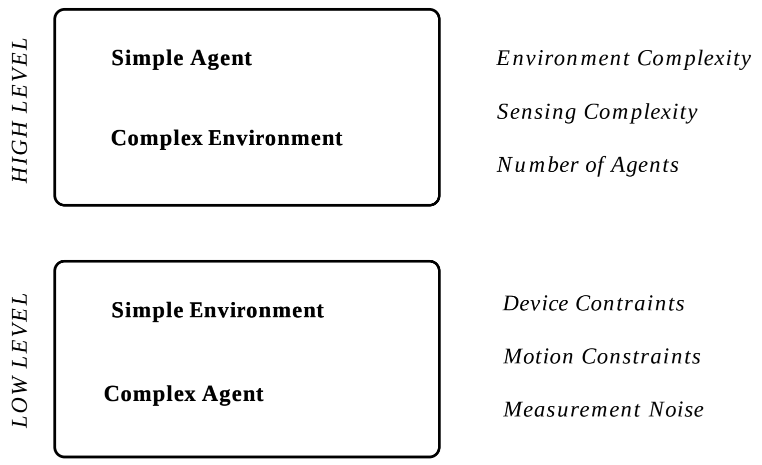


Figure 1.4: Our methodology is based on a two-level abstraction. In this thesis, we focus on the higher level and present distributed and sensing algorithms that explicitly deal with the three sources of complexity inherent in the design of such systems.

Note that the traditional robotics approach mostly focuses on the lower level. Consequently many problems in the domain of this level are well-studied. On the hand, a rigorous treatment of issues at the higher level is still lacking. This is the main focus of this thesis.

1.5 Our contribution

The methodology described in the previous section allows us to focus on combinatorial and geometric challenges in designing sensing systems for complex environments. By ignoring low-level issues, we offer solutions that have the following qualities:

1. **Worst-case guarantees:** In all problems, we provide solutions that have quality guarantees for all instances of the problem. For example, in Chapter 6, we present a pursuit strategy that works against any evader strategy on any graph. Such worst-case guarantees are rare in robotics and providing such guarantees is a novel contribution of this thesis.

2. **Visibility-based planning:** As mentioned in Section 1.4.1, sets that arise from visibility have a complex structure and this complexity makes visibility-planning tasks quite difficult. On the other hand, vision is the richest perception mode and cameras are becoming increasingly common in sensing tasks. As a result, there is a growing need for algorithms to accomplish visibility-based sensing tasks. To meet this demand, we focus on sensing tasks where vision is the primary sensing mode. This, we believe, is one of the main contributions of this thesis.

To design efficient algorithms for visibility-based planning tasks, we must understand and utilize geometric constraints imposed by the structure of the environment. The study of such constraints will be one of our primary focuses throughout the thesis. For exploration and pursuit-evasion, we will study the connections between shortest-paths in polygonal environments and visibility. For sensor placement, we will study the structure and complexity of sets that arise from visibility. Finally for sensor-assignment, we will utilize specific error models for stereo cameras to obtain efficient assignment algorithms. We believe that the study of such structural properties is of independent interest. and another contribution of the thesis.

We proceed with a summary of our results for individual problems.

1.5.1 Exploration

An online algorithm [31, 12] is an algorithm which does not have access to its entire input in advance. Instead, the input is presented to the online algorithm during its execution. Due to the lack of a prior map of the environment, exploration problem must be solved in an online fashion.

In Chapter 2, we study a local exploration problem: Imagine that a robot standing in an unknown environment sees the end-point of an edge (a wall), but does not know its orientation. We design a sensing strategy that minimizes the time spent in both traveling and reconstruction so as to “see” the unknown edge. This solution can be used by a mapping algorithm which, for example, explores the walls of an apartment one after another.

We first study this problem in the context of online algorithms and present a competitive strategy that guarantees a worst-case performance. However, the edge orientations are usually not arbitrary, especially in indoor environments. Therefore, we also present a probabilistic framework that incorporates prior information about the environment, if it is available.

1.5.2 Sensor placement

In Chapter 3, we first study a deployment scheme where each sensor may fall onto a random point in the environment. This model is relevant to scenarios where the sensors are dropped from a plane or scattered over long distances by a cannon. In these scenarios, the main placement question becomes "How many random sensors must be placed to ensure coverage?"

A set system is a pair (X, R) where X is the base set and R is a collection of some subsets of X. We can obtain an upper-bound for the deployment question above by studying the complexity of set systems which arise from sensor deployment. These are set systems where the base set is the environment we would like to cover. For each candidate sensor location, there is a subset in the set-system that corresponds to the portion of the environment sensed from that location. One measure of the complexity of set systems is their Vapnik-Chervonenkis (VC)-dimension. If the VC-dimension of this set system is bounded, then a small number of sensor suffice to cover the base set.

In Chapter 3, we study external visibility and the deployment of cameras to cover the exterior of a building. We show that the VC-dimension of polyhedral scenes is unbounded. We study two planar cases and prove the exact VC-dimension for both: If the cameras are restricted to lie on a circle, the VC-dimension is 2 and if they lie outside the convex-hull of the polygon, the VC-dimension is 5.

We also study a more controlled placement problem which usually arises in object-inspection settings. Suppose we have a CAD model of a part and we would like to inspect it for defects. A typical setup is to place the object on a turntable and capture images of the object from a single camera, while rotating the object. Equivalently, we may rotate the camera around the object. The sensor-placement question for this scenario is to find the angles for the turntable to stop so that the object is entirely covered. For this problem, we present an approximation algorithm that uses at most one more sensing location than the optimum number.

1.5.3 Sensor assignment

Once a sensor network has been deployed, one of its primary uses is tracking. In an adversarial setting, targets must be tracked in order to monitor suspicious activity. However, target tracking need not be adversarial. For example, a sensor network can track mobile robots so that the mobile robots can localize themselves. Such tracking tasks also arise in location-aware computing where devices query sensors to localize themselves.

A typical device used for sensing in a sensor network can not estimate the position of a target by itself. For example, at least two cameras are needed to estimate the position of a target. In addition, a single sensor may not participate in tracking more than one target. In the camera example, if the cameras zoom onto the target, there can be at most one target within the field of view of the camera.

Motivated by these constraints, we formulate the following *Focus of Attention Problem*: How should disjoint pairs of sensors be assigned to targets so as to minimize the overall error in estimating their positions?

In this Chapter 4, we start with a study of error metrics used for position estimation. We then study the focus of attention problem and present approximation algorithms to solve it.

1.5.4 Pursuit-evasion

Recently, there has been increasing interest in capturing intelligent evaders with certain sensing capabilities contaminating a complex environment. Aside from its obvious applications in search-and-rescue as well as surveillance, this problem provides a natural test-bed for many mobile robot algorithms.

Mainly, there are two approaches for modeling complex environments. One approach is to model the environment by a graph. For example, the nodes of the graph may correspond to physical locations such as rooms and the edges may represent doors. The other approach is to model the environment by a geometric object such as a polygon. For example, the polygon may correspond to a 2D layout of a factory.

In this thesis, we study pursuit-evasion games for both representations. The literature on pursuit-evasion games is vast. Therefore we dedicate Chapter 5 to an overview of existing results. In Chapter 6, we study the following pursuit-evasion game that is played on a graph: One or more pursuers are seeking to capture an evader on a graph. At each round, the evader tries to gather information about the location of the pursuers but it can see them only if they are located on adjacent nodes. Such a local information structure is typical in robotics applications and is the main novelty of our formulation. For this problem, we show that two pursuers suffice for catching evaders with limited visibility with high probability. We also present polynomial time algorithms that decide whether a single pursuer can capture the evader on a given graph.

This graph model allows us to incorporate arbitrary sensing and environment complexity into a graph. However, motion and visibility are coupled. That is, the set of vertices a player can move to is the same as the set of vertices it can see. In typical mobile robot settings, the sensing range is much larger than the set of points the robots can move to in the next time step. Therefore, in Chapter 7 we also study the same game played in a polygonal environment: A player located at point p can see the point q as long as the line segment pq lies entirely within the polygon.

However, players can only move to points that are within unit distance from their current positions.

We present algorithms for one or more pursuers to locate and capture an evader. We consider various models for the motion and the visibility of the evader. A detailed description of these models is deferred to Chapter 5.

1.6 Outline

This thesis contains seven chapters following this introductory chapter. Each chapter can be read independently.

Chapter 2 is dedicated to the exploration problem. Chapter 3 is on sensor placement. In this chapter, present results on VC-dimension of visibility systems and show how these results can be utilized in sensor deployment. We also present an approximation algorithm for placing sensors on a circle. Chapter 4 is on sensor assignment and dedicated to the Focus of Attention problem.

The pursuit-evasion problem is studied in three chapters. In Chapter 5, we give an overview of pursuit-evasion games in complex environments. Usually, a complex environment is either represented by a graph or a polygon. We present results obtained for either representation in Chapter 6 (for graphs) and Chapter 7 (for polygons).

We conclude the thesis with an overview of its contributions and future directions in Chapter 8.

Chapter 2

Local Exploration

Many sensing tasks require robots to sense every point in the environment. This is an important prerequisite for mapping tasks [104] which is one of the primary applications of current robotic systems. In the robotics literature, the term mapping is used to refer to the general task of building environmental layouts from measurements obtained by sensors mounted on mobile robots. In some applications, it is possible that we have some prior information about the environment to be mapped. For example, we may want to use our robot to build a detailed 3D model of an apartment and we may have a floor plan of the apartment beforehand. However, in most applications it is likely that we have no prior information about the environment. In this case, we have to solve the exploration problem: generate a trajectory for our robot, so that it sees *every point* in the environment *as quickly as possible.*

Since the environment is unknown, exploration is an online problem and therefore has been studied extensively in the context of online algorithms. An online algorithm [31, 12] is an algorithm which does not have access to its entire input in advance. Instead, the input is presented to the online algorithm during its execution. However, the online framework has not had much impact in practice as *"Today's approaches are usually greedy, that is, they choose control by greedily maximizing information gain"* [104]. Perhaps an explanation for this gap is hinted by Thrun in [104].

When choosing where to move, various quantities have to be traded off: the expected gain in map information, the time and energy it takes to gain this information, the possible loss of pose information along the way, and so on. Furthermore, the underlying map estimation technique must be able to generate maps in real-time, which is an important restriction that rules out many existing approaches.

As we shall see in Section 2.2, existing online algorithms assume continuous sensing. However, this assumption is violated in common mapping systems: A robot equipped with a laser scanner must stay stationary and obtain a range map before deciding where to go. Similarly, in stereo vision systems where the computation is costly and cannot be pipelined with image grabbing, robots spend time in reconstruction.

To address this issue, we present a local exploration algorithm in this chapter. Imagine that a robot standing in an unknown environment sees the end-point of an edge (e.g. a wall), but does not know its orientation. We design a sensing strategy that minimizes the time spent in both traveling and reconstruction so as to "see" the unknown edge. The novelty of this formulation is in addressing the problem of spending time in range acquisition which has not been accounted for in previous approaches.

Our local exploration algorithm can be used as an efficient subroutine for a robot which builds the model of an unknown building one wall after another. This greedy approach is well justified, because it is known [5] that it is not possible to have a global exploration algorithm with guaranteed performance[1].

We start with an overview of existing approaches within the robotics community.

[1]We shall revisit this result after an overview of online algorithms in Section 2.2.

2.1 The Exploration Problem in Robotics

Mapping is a very active research area and a recent survey of the state of the art can be found in [104]. In the robotics community, most effort is dedicated to localizing the robot during mapping. This is known as the simultaneous localization and mapping (SLAM) problem [73]. A recent overview of the state of the art in SLAM research can be found in [74] in addition to [104].

An additional challenge in unknown environments is the issue of visual coverage, better known as visual exploration. The visual aspect of coverage is emphasized in [36, 101] while area coverage is the focus of [64]. The latter approach can be used to produce a roadmap to sweep a space [22].

As mentioned before, the exploration problem for visual coverage is currently solved using heuristic approaches. Recent related work includes Rekleitis et al. [89] who use two robots for visual exploration. In their method, one robot reconstructs the environment while the other one is utilized for planning the next location of the first robot. Zlot et al. [113] present a multi-robot approach for exploration trying to maximize information gain while minimizing travel costs. Burgard et al. [17] assign a new target point for each of a group of robots so that the cost of reaching these points is minimized and the amount of already explored area is simultaneously maximized. An issue that has received a lot of attention recently is to close loops in the map, i.e. to discover already visited areas [40].

A related problem is the *next best view* problem in Computer Vision [111, 97, 66]. In these approaches, usually the problem of how to move a camera mounted on a gripper to maximize the information gain and the quality of the next picture taken is addressed without any optimality guarantees for the motion of the gripper.

2.2 Online Algorithms and Competitive Analysis

Traditional algorithms typically operate on the entire input. In online problems [31] the input is not known in advance but presented to the algorithm during its operation. One way of measuring the performance of online algorithms is competitive analysis [12]. In competitive analysis, we compare the performance of an online algorithm against the performance of the optimal offline algorithm and consider the worst case ratio. Let $cost_A(\sigma)$ be the cost incurred by an online algorithm A on the input sequence σ. Let OPT be the optimal offline algorithm and let $cost_{OPT}(\sigma)$ be the cost incurred by the optimal offline algorithm on input σ. We say that the online algorithm A is *c-competitive*, if there exits a constant b such that on every input sequence σ,

$$cost_A(\sigma) \leq c \cdot cost_{OPT}(\sigma) + b$$

The *competitive ratio* is the infimum over c such that A is c-competitive. We say that an algorithm is competitive, if it has a constant competitive ratio. In robotics, competitive analysis has been used for various navigation problems as a measure of efficiency [11, 26, 5, 50, 25, 47]. In the context of exploration, the competitive ratio gives us the worst case deviation of the cost of an exploration algorithm from the cost incurred by a robot who has a prior model of the environment and still wants to build a map.

2.2.1 Competitive analysis in robot exploration

A 2-competitive algorithm for rectilinear polygons with bounded number of obstacles has been presented in [25]. For simple polygons without obstacles, a 26.5-competitive algorithm has recently been proposed [47]. For polygons with an arbitrary number of obstacles, it has been shown that there is no competitive strategy [5]. For the local problem of how to look around a corner, which is addressed in this paper, a 1.21-competitive algorithm has been presented [50].

All above algorithms make the continuous visibility assumption that the robot can acquire a 3D view of the environment without any stop or cost for this acquisition. This assumption is violated for range scanners where the robot has to stop and acquire the locally visible 3D-view. It does not apply for omnidirectional visual stereo reconstruction either, because current acquisition times do not allow on the fly computation: the robot can only decide where to move after acquiring the map.

2.2.2 New problem statement

In this chapter, we address local exploration strategies which can be useful in global exploration. We make the following assumptions:

- The 3D-environment consists of vertical edges and walls and thus can be modeled as a polygon in the flatland.
- The robot can localize itself with respect to an acquired view and that it can register these views in the same coordinate system. We assume that there is no uncertainty in the robot's position estimate.
- The robot has an omnidirectional range acquisition system, which means no restrictions in the field of view.
- The robot does not move during range acquisition.
- The circle defined by robot's current position and the vertex adjacent to the edge to be explored (figure 2.1) is free of obstacles.

In this setup, we consider a robot, located at x, seeing the edge E but not E', the next edge (see figure 2.1). Edge E' makes an angle of θ with the line passing through x and the corner y. The optimal offline strategy is to follow path P_1, going directly to the extension of E' when $\theta < \frac{\pi}{2}$ and to go directly to y otherwise. In the online setting it is not possible to follow P_1, because its orientation depends on E' which has

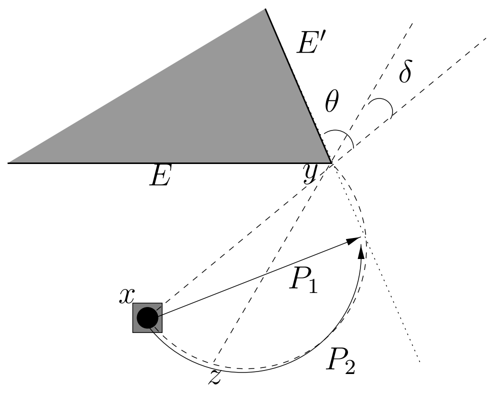


Figure 2.1: The problem description: The robot, located at x, can see the edge E but not E', the next edge. E' makes an angle of θ with the line passing through x and the corner y. The optimal offline strategy is P_1, going directly to the extension of E' when $\theta < \frac{\pi}{2}$ and to go directly to y otherwise. When robot has continuous vision, by following P_2 along the circle whose diameter is $\overline{xy}$, the robot guarantees a competitive ratio of $\frac{\pi}{2}$.

not been seen by the robot yet. When the robot has continuous vision, by following P_2 along the circle whose diameter is $\overline{xy}$, the robot guarantees a competitive ratio of $\frac{\pi}{2}$. This strategy was used in [47] as a part of the global exploration strategy assuming on the fly acquisition.

Here, we introduce a new cost measure for the time it takes to see the next occluded edge as the sum of the time spent in reconstructions plus the time spent in traveling:

$$cost_A(\sigma) = \tau N + \frac{d}{v}, \tag{2.1}$$

where τ is the time it takes to make a reconstruction, N is the number of reconstructions made until next edge is seen, d is the distance traveled, and v is the velocity of the robot. The input σ consists of robot's position x, the position of the corner vertex y, and the angle θ the next edge makes with the robot's line of sight (see figure 2.1). Note that our cost model assumes constant velocity, however it is possible to incorporate more complicated dynamics into equation 2.1.

Our contribution is two-fold:

- In a deterministic set-up with no knowledge about the occluded edge, we present two competitive strategies.

- Assuming a belief about the occluded edge, we propose two formalizations in terms of a Markov Decision Process (MDP) and solve for optimal policies that maximize the overall expected reward.

In simulations, we compare the four algorithms and we find out that the MDP policies outperform the deterministic algorithms when the beliefs are close to reality. This chapter is written in the order described above: competitive algorithms, MDP framework, and experimental analysis.

2.3 Competitive Algorithms

Let x be robot's position, y be the corner, D be the distance from the robot's current position to the corner, v be the speed of the robot, and τ be the time it takes to make a reconstruction. Let $\epsilon\frac{D}{v} = \tau$. That is, the time it takes to make a reconstruction is ϵ times the time it takes to reach the corner. Let $t_{OPT} > 0$ be the time it takes the optimal algorithm to reach the point where it can see the next edge (traversing P_1 in Fig. 2.1).

If $\epsilon \geq 1$, then the robot goes straight to the corner. Since $\frac{D}{v} \leq \tau$, the competitive ratio becomes

$$\frac{\tau + \frac{D}{v}}{\tau + t_{OPT}} \leq \frac{2\tau}{\tau + t_{OPT}} \leq 2.$$

Otherwise, we propose two algorithms given in Table 2.1.

2.3.1 Algorithm UNIREC

Let $\mathcal{C}$ be the circle whose diameter is the line segment that joins the robot to the corner (i.e x to y). Suppose in Fig. 2.1, that during time τ the robot travels to position z on $\mathcal{C}$ without leaving the circle. Let $\delta = \angle xyz = \frac{\tau v}{D}$. Note that $\epsilon = \delta$. The robot will go to the points on $\mathcal{C}$ defined by $\delta, 2\delta, 3\delta, \ldots$ until it sees the next

UNIREC(τ, v, x, y)	**EXPREC**(τ, v, x, y)
$D \leftarrow dist(x, y)$	$D \leftarrow dist(x, y)$
$\mathcal{C} \leftarrow circle(x, y)$	$\mathcal{C} \leftarrow circle(x, y)$
$\delta \leftarrow \frac{\tau v}{D}$	$\delta \leftarrow \frac{\tau v}{D}$
If $\delta > 1$ go to x	If $\delta > 1$ go to x
Otherwise	Otherwise
$i = 1$	$i = 1$
Until the next edge is seen	Until the next edge is seen
Visit $i\delta$	Visit $i\delta$
Reconstruct	*Reconstruct*
$i \leftarrow i + 1$	$i \leftarrow 2i$

Table 2.1: The input x is the robot's position, y is the location of the corner, v is robot's speed, and τ is the time it takes to make a reconstruction. The command *Reconstruct* denotes the operation of an omnidirectional range acquisition and $circle(x, y)$ is the circle that passes through x and y and has a diameter $dist(x, y)$. Algorithm UNIREC has a competitive ratio of π and algorithm EXPREC has a competitive ratio of 2.23.

edge without leaving the circle [2]. Let $\theta \in [0, \frac{\pi}{2}]$ be the actual angle (Fig. 2.1 between the edge and the line that passes through the robot's position and the corner). The competitive ratio of this algorithm reads:

$$C = \frac{\lceil\frac{\theta}{\delta}\rceil\tau + \lceil\frac{\theta}{\theta}\rceil\theta\frac{D}{v}}{\tau + \frac{D}{v}\sin\theta}$$

Since $\frac{D}{v} = \frac{\tau}{\delta}$ we obtain

$$C \;=\; \frac{\lceil\frac{\theta}{\delta}\rceil\tau + \lceil\frac{\theta}{\delta}\rceil\delta\frac{\tau}{\delta}}{\tau + \frac{\tau}{\delta}\sin\theta} = \frac{2\lceil\frac{\theta}{\delta}\rceil}{1 + \frac{\sin\theta}{\delta}} \leq \frac{2(\frac{\theta}{\delta} + 1)}{1 + \frac{\sin\theta}{\delta}}$$

which is increasing with θ. Hence, the worst case is achieved when $\theta = \pi/2$:

$$C \;\leq\; \frac{2(\frac{\pi}{2\delta} + 1)}{1 + \frac{1}{\delta}} = \frac{2\delta + \pi}{\delta + 1}$$

Since $\delta < 1$, the worst case is achieved as $\delta \longrightarrow 0$ and the ratio becomes π.

[2]The reader may wonder why we do not take the short cuts instead, which means compute δ and go straight to the point $(D\cos\delta, D\sin\delta)$ and continue with updating $D \leftarrow D\cos\delta$. Even though this might perform better for some values of τ, it does not improve the competitive ratio for small δ: $D\cos\delta \approx D$ and $D\sin\delta \approx D\delta$

2.3.2 Algorithm EXPREC

It is possible to improve this ratio by modifying the strategy as follows: Instead of visiting $\delta, 2\delta, 3\delta, \ldots$, the robot increases exponentially its steps and visits $\delta, 2\delta, 4\delta, \ldots, 2^i\delta$. Note that during the i^{th}-step robot traverses an angle of $2^{i-1}\delta$ and the total angle traversed so far is $(2^i - 1)\delta$. If θ is the actual angle, the robot sees the next edge as soon as it takes $i = \lceil \log(\frac{\theta}{\delta} + 1) \rceil$ steps. The competitive ratio reads:

$$C = \frac{i\tau + \delta(2^i - 1)\frac{D}{v}}{\tau + \frac{D}{v}\sin\theta}$$

The worst case of the ratio of EXPREC is thus 2.2214. We present the details of this derivation next.

2.3.3 The competitive ratio of EXPREC

In this section we derive the competitive ratio for the algorithm EXPREC. Recall that instead of visiting $\delta, 2\delta, 3\delta, \ldots$, the robot takes exponential jumps and visits $\delta, 2\delta, 4\delta, \ldots, 2^i\delta$. During the i^{th} step robot traverses an angle of $\delta 2^{i-1}$ and the total angle traversed so far is $\delta(2^i - 1)$. Therefore if θ is the real angle, the robot sees the next edge as soon as it takes $i = \lceil \log(\frac{\theta}{\delta} + 1) \rceil$ steps.

Case 1: $\theta \leq \frac{\pi}{4}$

$$
\begin{aligned}
C &= \frac{i\tau + \delta(2^i - 1)\frac{R}{v}}{\tau + \frac{D}{v}\sin\theta} \\
&= \frac{i\tau + \delta(2^i - 1)\frac{\tau}{\delta}}{\tau + \frac{\tau}{\delta}\sin\theta} \\
&= \frac{i + (2^i - 1)}{1 + \frac{\sin\theta}{\delta}} \\
&= \frac{\lceil \log(\frac{\theta}{\delta} + 1)\rceil + 2^{\lceil \log(\frac{\theta}{\delta}+1)\rceil} - 1}{1 + \frac{\sin\theta}{\delta}} \\
&\leq \frac{(\log(\frac{\theta}{\delta} + 1) + 1) + 2^{(\log(\frac{\theta}{\delta}+1)+1)} - 1}{1 + \frac{\sin\theta}{\delta}} \\
&= \frac{\log(\frac{\theta}{\delta} + 1) + 2^{(\log(\frac{\theta}{\delta}+1)+1)}}{1 + \frac{\sin\theta}{\delta}} \\
&= \frac{\log(\frac{\theta}{\delta} + 1) + 2(\frac{\theta}{\delta} + 1)}{1 + \frac{\sin\theta}{\delta}} \\
&= \frac{2(\frac{\theta}{\delta} + 1)}{1 + \frac{\sin\theta}{\delta}} + \frac{\log(\frac{\theta}{\delta} + 1)}{1 + \frac{\sin\theta}{\delta}} \\
&= \frac{\frac{2}{\delta}(\theta + \delta)}{\frac{1}{\delta}(\delta + \sin\theta)} + \frac{\log(\frac{\theta}{\delta} + 1)}{1 + \frac{\sin\theta}{\delta}} \\
&= \frac{2(\theta + \delta)}{(\delta + \sin\theta)} + \frac{\log(\frac{\theta}{\delta} + 1)}{1 + \frac{\sin\theta}{\delta}}
\end{aligned}
$$

This last ratio achieves its maximum value of 2.2214 when $\delta \to 0$ and $\theta = \frac{\pi}{4}$.

Case 2: $\frac{\pi}{4} \leq \theta \leq \frac{\pi}{2}$

If θ is slightly larger than $\frac{\pi}{4}$, the robot takes a huge last step and goes all the way to the corner following the entire half circle. The competitive ratio is:

$$\begin{aligned}
C &\leq \frac{\log(\lceil \frac{\pi}{2\delta} \rceil)\tau + \frac{\pi R}{2v}}{\tau + \frac{D}{v}\sin\frac{\pi}{4}} \\
&= \frac{\log(\lceil \frac{\pi}{2\delta} \rceil)\tau + \frac{\pi\tau}{2\delta}}{\tau + \frac{\tau}{\delta}\sin\frac{\pi}{4}} \\
&= \frac{\log\lceil \frac{\pi}{2\delta} \rceil + \frac{\pi}{2\delta}}{1 + \frac{\sin\frac{\pi}{4}}{\delta}} \\
&= \frac{\log\lceil \frac{\pi}{2\delta} \rceil}{1 + \frac{\sin\frac{\pi}{4}}{\delta}} + \frac{\frac{\pi}{2\delta}}{1 + \frac{\sin\frac{\pi}{4}}{\delta}}
\end{aligned}$$

As $\delta \rightarrow 0$, the first term vanishes and the competitive ratio becomes $\frac{\pi/2}{\sin\frac{\pi}{4}} = 2.2214$

2.4 Probabilistic framework

In most environments, we expect that the robot has some beliefs about the angles formed by vertices in polygonal environments. For example, most angles in man-made environments are rectilinear or in case of doors 180 degrees. In this section, we present a framework that allows us to represent the robot's belief about the environment as a probability distribution and show how to solve for optimal strategies when such beliefs are available.

A finite state Markov Decision Process (MDP) is given by a finite set of states S, a finite set of actions A, transition probabilities $P(r|s,a)$ of arriving at state r when action a is taken from state s, and rewards $R^a_{s,r}$ from arriving at state r from state s via action a. A policy π is a function that takes a state-action pair (s,a) and returns a real number in [0,1], indicating the probability of taking action a when in state s. An optimal policy is a policy which maximizes expected return at each state. Given a finite MDP it is possible to find an optimal policy using dynamic programming or its variants such as Policy Iteration. A comprehensive introduction to MDP can be found in [98, 10].

Suppose we have the distribution $P^{\theta}(\theta)$ for the distribution of the corner angles. For example, one can express the belief that the environment is rectilinear by choosing $P^{\theta}(\theta)$ to be a truncated gaussian with mean 90 degrees and a variance representing the uncertainty of this belief. Another possibility is to keep the histogram of the angles already observed during the exploration and to use this histogram as an approximation for $P^{\theta}(\theta)$. Yet another possibility is to use Monte Carlo Methods [98] for reinforcement learning to incorporate the learning of $P^{\theta}(\theta)$ into the exploration process. Even though obtaining $P^{\theta}(\theta)$ is an interesting problem on its own, from now on we assume that it is given as an input.

One way to model the edge exploration problem is to discretize the circle whose diameter is the line segment joining the robot and the vertex using a resolution parameter δ. Let $n = \lfloor \frac{\pi}{2\delta} \rfloor$ and let us double use the notation $\delta, 2\delta, 3\delta, \ldots n\delta$ for both the stops on the circle as well as the angle between the initial and current positions of the robot whose apex is at the corner.

An MDP model, we will call MDP1, is presented in figure 2.2. State s_i represents the state of the robot when it is located at $i\delta$ and has not made a reconstruction yet. At each s_i it can either decide to move to s_{i+1} or make a reconstruction. When it makes a reconstruction it either sees the next edge in which case it goes to the state $\mathcal{F}$ and remains there or cannot see it yet. The latter case is represented by the state s'_i. From s'_i the only reasonable action is to move. Note that we chose to discretize the circle defined by the robots location and the corner, instead of discretizing the whole plane. The advantage of this approach is the drastic reduction in the number of states which means a reduction in the memory requirements and running time of the algorithm.

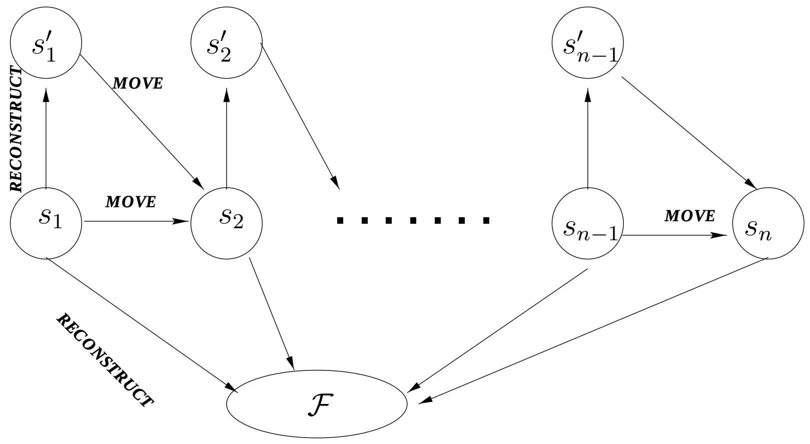


Figure 2.2: **MDP1** has $2n$ states where $n = \frac{\pi}{2\delta}$ that depends on the sampling parameter δ and $\mathcal{F}$ is the final state. Being in s'_i (resp. s_i) means that the robot is at δi and has just (resp. not) made a reconstruction.

The actions are Rec and Mov for reconstruct and move respectively. The transition probabilities are determined by the distribution $P^\theta(\theta)$:

$$\begin{aligned} P(\mathcal{F}|s_i, Rec) &= P^\theta(\theta \leq i\delta) \\ P(s'_i|s_i, Rec) &= P^\theta(\theta > i\delta) \\ P(s_{i+1}|s_i, Mov) &= 1 \\ P(s_{i+1}|s'_i, Mov) &= 1 \end{aligned}$$

All other probabilities are zero. Note that even though we assumed that the robot has complete control of its motion by letting $P(s_{i+1,j}|s_{i,j}, Mov) = 1$, one can easily incorporate uncertainty in motion using an appropriate uncertainty model.

The rewards, $R^a_{s_i,s_j}$, represent the immediate reward received upon arriving state s_j from state s_i as a result of action a. Since we are dealing with costs, we use negative costs as rewards we want to maximize.

$$\begin{aligned} R^{Mov}_{s_i,s_{i+1}} &= -\frac{\delta D}{v} \\ R^{Mov}_{s'_i,s_{i+1}} &= -\frac{\delta D}{v} \\ R^{Rec}_{s_i,s'_i} &= -\tau \\ R^{Rec}_{s_i,\mathcal{F}} &= -\tau \end{aligned}$$

Given a distribution P^θ, we compute the optimal policy that maximizes the expected reward using the well known policy iteration algorithm [98, pp98]. Policy iteration is known for its fast convergence properties in practice and this was indeed the case for our problem. For MDP1, we observed that the optimal policies move until enough reward is accumulated and start reconstructing afterwards. For example, if $P^\theta = \mathcal{N}(\mu, \sigma)$, the optimal algorithm turns out to move an angle of $\mu + \sigma$ and then to reconstruct at each step afterwards.

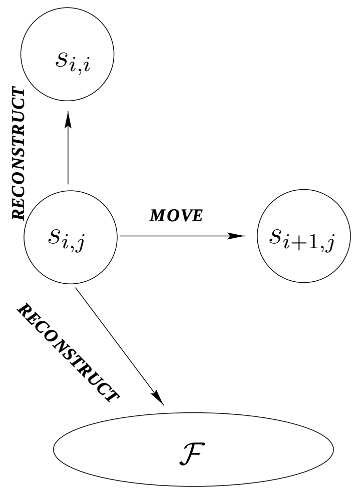


Figure 2.3: **MDP2** has $\frac{n(n+1)}{2}$ states where $n = \frac{\pi}{2\delta}$ that depends on the sampling parameter δ. $\mathcal{F}$ is the final state. Being in state $s_{i,j}$ means that the robot is at $i\delta$ on the circle and the last reconstruction was at $j\delta$.

It is possible to obtain a better performance by remembering the last reconstruction made. Let s_{ij} represent the information that the robot is standing at $i\delta$ and the last reconstruction it made was at $j\delta$. Figure 2.3 illustrates the transitions for state $s_{i,j}$. The transition probabilities and rewards for this new MDP, which we call MDP2, are given by:

$$\begin{aligned} P(\mathcal{F}|s_{i,j}, Rec) &= P^{\theta}(j\delta \leq \theta \leq i\delta) \\ P(s_{i,i}|s_{i,j}, Rec) &= 1 - P(\mathcal{F}|s_i, Rec) \\ P(s_{i+1,j}|s_{i,j}, Mov) &= 1 \end{aligned}$$

All other probabilities are zero.

$$\begin{aligned} R^{Mov}_{s_{i,j},s_{i+1,j}} &= -\frac{\delta D}{v} \\ R^{Rec}s_{ij}, s_{i,i} &= -\tau \\ R^{Rec}s_{i,j}, \mathcal{F} &= -\tau \end{aligned}$$

The drawback of this approach is the increase in the number of states, and hence the running time of the algorithm. The former policy based on MDP1 requires $2n$ states, whereas the number of states for MDP2 is $\frac{n(n+1)}{2}$. Note that states s_{ij} with $i < j$ are not well defined. The power of MDP2 is illustrated in figure 2.4, where the figure on the left illustrates the optimal policy for the bimodal distribution P^{θ} on the right. Based on MDP1, in contrast, the robot moves until enough reward accumulates and reconstructs afterwards. This strategy does not exploit the low probability region as MDP2. We further illustrate the optimal policies for MDP1 and MDP2 for various distributions in the next section.

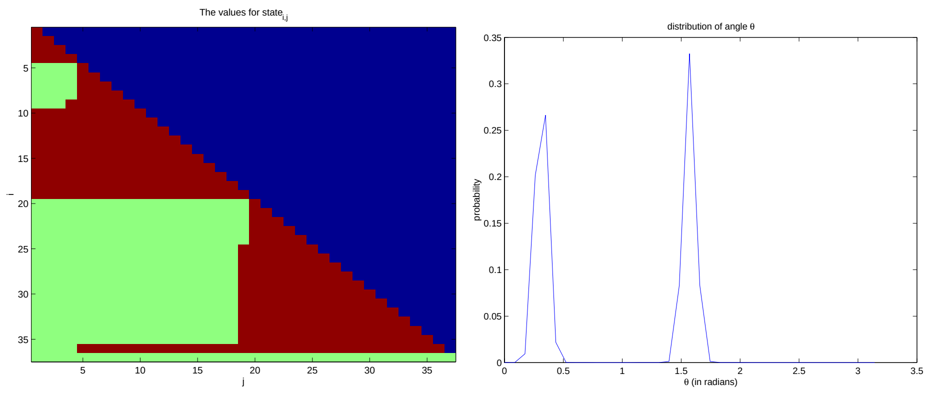


Figure 2.4: **Left:**The optimal policy for MDP2. The sampling value δ used was 5 degrees, therefore the location i, j in the image above represents the action when the robot is at $i\delta$ and the last reconstruction it made was at $j\delta$. The vertical column is i and the horizontal columns is j. The blue upper right half illustrates the inaccessible states. The green values at the lower right correspond to RECONSTRUCT actions and the red region in between correspond to MOVE action. The distribution P^{θ} is according to the distribution on the **Right**.

2.5 Simulation Results

In this section we compare the four algorithms we describe in this chapter. **UNIREC** and **EXPREC** are the two competitive algorithms described in table 2.1. We will refer to the optimal policy of MDP1 summarized in figure 2.2 as **POLICY1** and the optimal policy of MDP2 summarized in figure 2.3 as **POLICY2**.

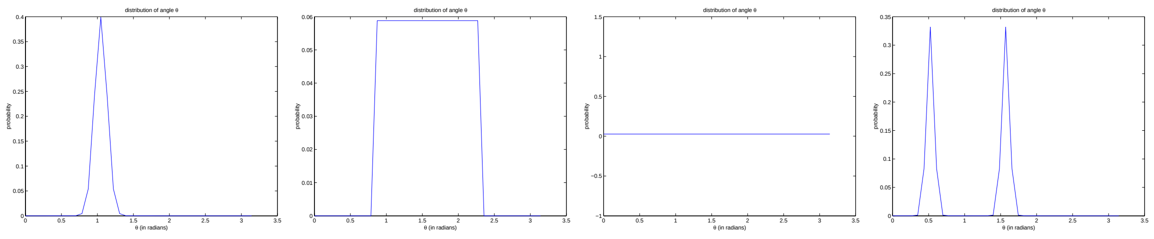


Figure 2.5: The distributions used for experiments: Each bucket represents 5 degrees. **Distribution 1** is generated using a gaussian with mean 60 degrees and variance 5 degrees. **Distribution 2** is uniform between $\frac{\pi}{4}$ and $\frac{3\pi}{4}$. **Distribution 3** is uniform between 0 and π. **Distribution 4** is a bimodal distribution obtained by adding up two gaussians with means $\frac{\pi}{6}$ and $\frac{\pi}{2}$ and a variance of 3 degrees.

2.5.1 The underlying distribution is known

The algorithms **UNIREC** and **EXPREC** have performance guarantees regardless of the distribution P^{θ}. In this section, we try to answer the question: Is it really worth solving for optimal policies, even when P^{θ} is available? The answer turns out to be yes, as the following experiments show.

We compare the results for MDPs built using the exact distribution of θ with the competitive algorithms. In other words, the instances of the simulations were generated from the distributions in figure 2.5 and the same distributions were used to build the MDPs. The sampling parameter for all the MDPs we used is 5 degrees which is equal to the bucket sizes of the distributions.

In the following experiment, summarized in table 2.2, the robot stands on the wall, 10m away from the corner. This aligns the line of sight of robot with the visible edge, allowing us to use the full range of $[0, \pi]$ for θ. Hence, $D = 10m$. Each reconstruction takes 2 seconds and the robot moves with a speed of $0.5m/s$. The time it takes to reach the corner is 20 seconds, therefore $\delta = 0.1$ for algorithms UNIREC and EXPREC.

P^{θ}	**UNIREC**	**EXPREC**	**POLICY1**	**POLICY2**
1	43.98	39.40	26.72	28.93
2	57.22	39.42	41.77	35.16
3	48.17	34.88	48.53	34.82
4	43.06	37.63	37.69	27.39

Table 2.2: The results when the underlying distributions match the beliefs about the distribution. 1000 samples were drawn from the distributions in figure 2.5 (numbers in the first column reference Figure 2.5). Rest of the columns present the average time to see the next edge for the four algorithms presented in this chapter.

Note that Distribution 3, which is uniform in $[0, \pi]$, represents the case when there is no apriori information about the environment. The policies for this case are presented in figure 2.6. In this case, all MDP1 can do is to move until enough probability is accumulated and to reconstruct at every step afterwards, as it has no memory of the previous reconstruction. MDP2, in contrast, prefers to move further after a recent reconstruction.

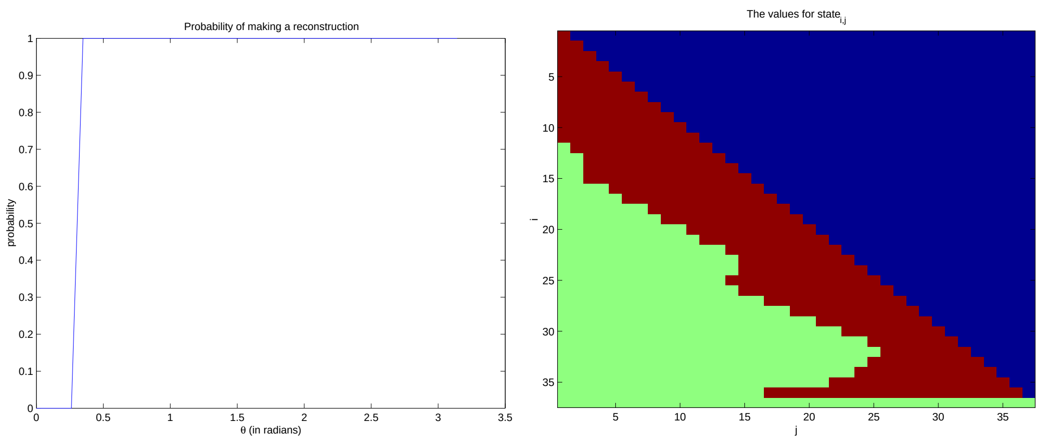


Figure 2.6: Optimal policies for MDP1 and MDP2 for Distribution 3 in figure 2.5. **Left:** The probability of making a reconstruction for MDP1 **Right:** The policy for MDP2

The next experiment is the same as the previous one other than the reconstruction time $\tau = 10$ seconds and therefore $\delta = 0.5$ and the results are presented in table 2.3.

P^θ	**UNIREC**	**EXPREC**	**POLICY1**	**POLICY2**
1	53.96	40.94	36.99	37.43
2	68.64	46.49	47.86	53.63
3	61.30	40.82	76.21	46.95
4	56.92	35.24	68.25	43.16

Table 2.3: The results when the underlying distributions match the beliefs about the distribution. 1000 samples were drawn from the distributions in figure 2.5 (column1). Rest of the columns present the results for running the four algorithms presented in this chapter.

Using results in table 2.2 and table 2.3 we see that if the underlying distribution is available, the optimal policies outperform the competitive algorithms for some distributions. Another observation is that when the reconstruction is costly ($\tau = 2$

vs $\tau = 10$) the number of reconstructions become really significant and POLICY2 starts outperforming POLICY1. To illustrate this further we ran simulations that keep the distribution constant but vary the reconstruction time and the results are shown in figure 2.7.

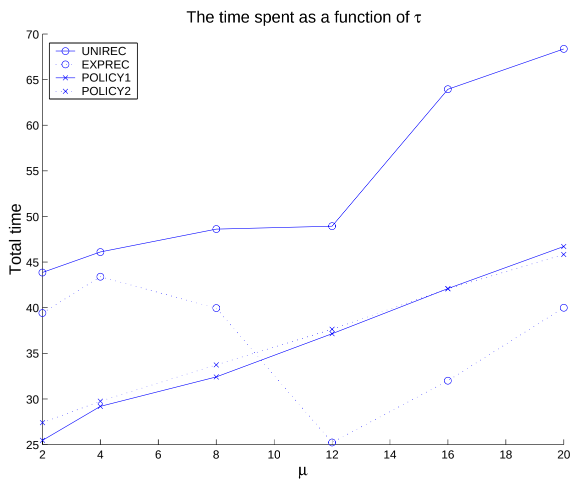


Figure 2.7: $P^{\theta}(\theta) = \mathcal{N}(60, 5)$, but τ varies according to the values in the horizontal axis. POLICY2 outperforms POLICY1 as τ increases.

2.5.2 When the beliefs are wrong

In order to illustrate what happens when the robot's beliefs do not match the environment, we use a different distribution to draw samples for the experiment than the one we use to find the optimal policies for the MDPs. For example, for $\mu = 40$ in the left plot of figure 2.8, we computed the optimal policies for $\mathcal{N}(40, 5)$ and then used 1000 samples from $\mathcal{N}(60, 5)$ for simulations, in order to create a discrepancy between the robot's beliefs and the state of the world.

As in the previous section, in the following experiments the robot stands on the wall, 10m away from the corner. Each reconstruction takes 2 seconds and the speed of the robot is $0.5m/s$. The time it takes to reach the corner is 20 seconds, therefore $\delta = 0.1$ for algorithms UNIREC and EXPREC.

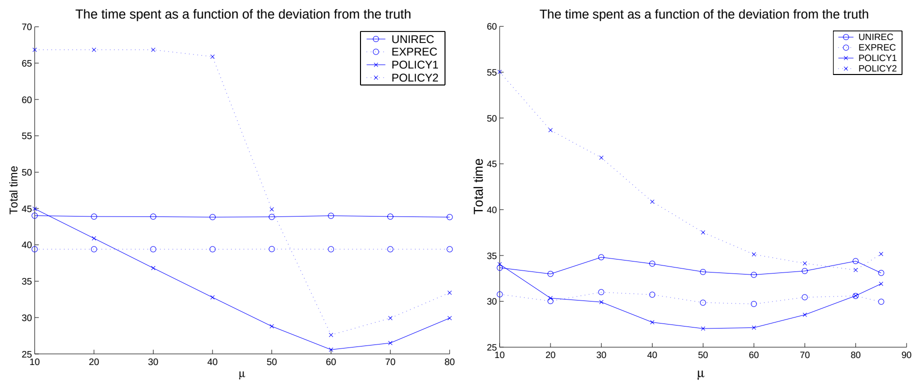


Figure 2.8: **Left:** The samples representing the real state of the world were drawn from $\mathcal{N}(60, 5)$ and the optimal policies were computed for $\mathcal{N}(\mu, 5)$ where μ is the horizontal axis. Each simulation is the average time to see the next edge for 1000 samples. **Right:** Same as left but the samples representing the real state of the world were drawn uniformly from $[0, \frac{\pi}{2}]$.

UNIREC	**EXPREC**	**POLICY1**	**POLICY2**
47.44	34.43	38.16	50.87

Table 2.4: Robot thinks the world is $\mathcal{N}(60, 5)$ but in fact the samples are drawn from uniformly from $[0, \pi]$.

As expected, when the beliefs are wrong, the performance of the algorithms UNIREC and EXPREC do not get affected, since they do not assume any distribution for the input. However, the results in figure 2.8 and table 2.4 suggest that MDP2 is more sensitive to errors in the underlying beliefs than MDP1. This is because MDP2 has a more specialized policy than MDP1.

2.6 Concluding Remarks

In this chapter, we have studied the problem of how to look around a corner in a polygonal environment given that we want to minimize the time spent in traveling as well as in reconstruction. We addressed local optimality regarding the visibility of the next occluded edge. Our strategy can accelerate heuristic planning for global exploration.

We presented competitive algorithms for the local exploration problem. These algorithms guarantee bounded deviation from the optimal behavior, even if no information about the environment is available apriori. For the cases where such information is available, we presented a formalization based on Markov Decision Processes. After this formalization, optimal strategies can be computed using well-known techniques. In our future work we plan the following thrusts: to incorporate uncertainty in the position estimates of the robot, to relax the circle discretization and search for a more efficient state-action tessellation of the plane, to study the local problem in 3D by generalizing the form of the occluding contour, and ultimately, to solve the global exploration problem efficiently.

Chapter 3

Sensor placement

Perhaps the most common method for continuously sensing a large, complex environment is to place a large number of sensors in the environment. A typical example is a system that uses surveillance cameras in a big hotel. One of first fundamental issues that arise when designing such a system is placement: How many sensors do we need in order to cover the environment? Where should we place these sensors?

Our ability to answer the placement question relies heavily on the sensor and the environment models. For example, covering a square region of area one with sensors which can detect motion within a disk of radius ϵ is an easy problem. On the other hand, covering a polygonal region with a minimum number of cameras is an NP-hard problem [83]. In fact, solving visibility-based coverage problems is one of the main challenges in computational geometry research.

In this chapter, we present two results that advance our understanding of such placement problems.

First, we study set systems that arise from visibility-based placement problems. A set system is a pair (X, R) where X is the base set and R is a collection of some subsets of X. For camera placement, we consider set systems where the base set is the environment we would like to cover. For each candidate sensor location, there is a subset in the set-system that corresponds to the portion of the environment sensed from that location. For sensor-placement applications, it is important to understand how complex such set systems can become. The complexity of set systems can be measured through their Vapnik-Chervonenkis (VC)-dimension. In addition, the study of VC-dimension allows us to answer the question: How many random samples (sensors) are required to cover a given scene? This question arises in deployment scenarios where the sensors are dropped onto a foreign territory and the location of the sensors can not be controlled precisely due to disturbances such as wind and the motion of the plane.

Second, in Section 3.6, we present an algorithm for placing cameras on a circle in such a way that ensures coverage of a known polygonal object. This task is motivated by the following scenario: Suppose we have a CAD model of a part and we would like to inspect it for defects. A typical setup is to place the object on a turntable and capture images of the object from a single camera, while rotating the object. Equivalently, we may rotate the camera around the object. The sensor-placement question for this scenario is to find the angles for the turntable to stop so that the object is entirely covered. For this problem, we present an approximation algorithm that uses at most one more sensing location than the optimal number of sensing locations.

We start with the results on VC-dimension of visibility-based set systems.

3.1 Sampling Based Sensor Placement

Imagine a known 3D polyhedral environment where a set of cameras has to be placed in such a way that every point in the environment is visible. The 2D version is known as the art gallery problem [83, 94, 84] and sufficiency results exist for several versions of this problem. For example, $\lfloor \frac{n}{3} \rfloor$ cameras can cover any simple polygon. However, such results are inapplicable in robotic and image-based rendering applications where the environments can be very complex with millions of vertices. A further application is placing antennas for line-of-sight broadband communication [1]. Imagine that backbones end at each neighborhood and that communication inside 1km can be achieved with line-of-sight laser beams that can carry from 10Mbps up to 1.25 Gbps bandwidth. Assuming that a consumer can put a receiving antenna at the window of her studio or even on a kiosk in a street, the coverage problem becomes a visibility problem where the cameras become arrays of distributing antennas.

In this work, we consider aspects of the problem of minimizing the number of viewpoints without sacrificing the goal of complete visibility. The particular scenarios we are addressing are surveillance, object inspection, and image based rendering. In the case of surveillance, we need a complete coverage at any time so that no event will be missed. This is the reason why coverage with one mobile guard (shortest watchman route - solvable in polynomial time [21]) is not applicable. In case of object inspection [93], we know the prior geometry of an object, and we need the minimal number of views so that the object will be checked regarding defects. In this scenario, the object might be placed on a turntable and we ask then for the minimal number of rotations. The objects might even be medical organs which have to be imaged from very few viewpoints of an endoscope guided by a robot manipulator. In the case of image based rendering, we might have a prior rough map of the environment but we need to obtain a detailed reconstruction with a range sensor. In other cases, we have a geometric model, but we do not know anything about the color or texture of an object. In all these cases, it is important that the rendered environment does not have any holes because of originally uncovered areas.

We are not going to address here the equally important problem of unknown environments or objects related to model building tasks. Several algorithms exist for exploring unknown environments [47] building upon fundamental results in the on-line traversal of graphs [25] or on Markov processes for modeling partially known dynamic scenes [70]. Significant contributions have been also achieved in the Next Best View planning problem [100, 66, 45] for surface acquisition. However, the results proven in this chapter have implications for the unknown case as well: Choosing sensor locations randomly is a method frequently used for sensor placement in unknown environments [16]. The VC-dimension theory enables us to answer the question: *How many random samples (sensors) do we need in order to cover a given region?* Our results, together with the theory of ϵ-nets, provide an upper bound to the answer to this question when omni-directional cameras are used as sensors.

It is well known [83] that the minimal guard coverage problem is NP-hard. To study the existence of approximation algorithms, we can consider minimal guard coverage as an instance of the set-cover problem. The general version of minimum set-cover cannot be approximated with a ratio better than $\log n$. However, we do not know whether any set-cover instance can be realized as a visibility system. A powerful interface between set-cover and the particular geometric setup is the Vapnik-Chervonenkis (VC) dimension which enables us to quantitatively bound how general a set system is.

In this chapter, we present new bounds on the VC-dimension for three instances of the problem. In section 3.2, we formalize the problem statement and describe our contribution to the state of the art. Next, we deal with 2D configurations and prove new bounds on the VC-dimension for cameras on a circle (subsection 3.3.1). Specifically, we prove that if the cameras are restricted to lie on a circle, the VC-dimension is 2 and if they remain outside the convex hull of the object, the VC-dimension is 5 (subsection 3.3.2). In section 3.4, we prove the main result that the VC-dimension for 3D configurations is unbounded. We conclude with a summary in Section 3.7.

3.2 Problem Statement

3.2.1 VC-dimension and set-cover

A set system is a pair $(X, \mathcal{R})$ where X is a set and $\mathcal{R}$ is a collection of subsets $R \subseteq X$.

Definition 1 *Given a set system $(X, \mathcal{R})$, let A be a subset of X. We say A is shattered by $\mathcal{R}$ if $\forall Y \subseteq A$, $\exists R \in \mathcal{R}$ such that $R \cap A = Y$. The VC-dimension of $(X, \mathcal{R})$ is the cardinality of the largest set that can be shattered by $\mathcal{R}$ [109].*

The VC-dimension, introduced first in supervised learning for pattern classification, plays an important role also in randomized and geometric algorithms [76, 3]. As an example we mention the related notion of an ϵ-net: If the set system $(X, \mathcal{R})$ has a constant VC-dimension d, then with high probability, a small number $(O(\frac{d}{\epsilon} \log \frac{1}{\epsilon}))$ of points sampled from X intersects all the subsets in R whose sizes are greater than $\epsilon|X|$ (such a sample is called an ϵ-net). Another useful property is that if $(X, \mathcal{R})$ has a constant VC-dimension d, then the number of subsets in $\mathcal{R}$ is $O(n^d)$ where $n = |X|$. A related result [108] that implicitly deals with ϵ-nets is the existence of polyhedra that require arbitrary number of guards even if every point inside the polyhedron can see an area equal to ϵ fraction of the total interior area, using the notion of ϵ-good polygons introduced in path planning [62].

Given a set system, the minimum *set-cover* problem asks for a minimum cardinality set $\mathcal{S} \subseteq \mathcal{R}$ such that $\bigcup_{R \in \mathcal{S}} R = X$. The *hitting set* problem is the dual of set-cover and its decision version reads: We are given a set X and a collection of sets $\mathcal{R}$ where each $R \in \mathcal{R}$ is a subset of X. We are also given a number k. The question is whether there is a subset $H \subset X$ such that $|H| \leq k$ and for each $R \in \mathcal{R}$, $R \cap H \neq \emptyset$.

Both problems are known to be NP-complete and can be approximated to within a log factor of the maximum set sizes (in either the primal or the dual system) and not better [95]. For sets systems with finite VC-dimension d, however, Brönnimann and Goodrich presented an algorithm which returns a solution whose size is at most $O(d \cdot \log OPT \cdot OPT)$ [15]. Here, OPT is the cardinality of the optimal solution. This is a significant improvement on the previous approximation, when the cardinality of the optimal solution is smaller than the cardinality of the maximum set size in the set system.

3.2.2 Visibility and set-cover

In this chapter we will address the problem of minimal guard coverage or camera placement. An *instance* of the camera placement problem is: Given a polygon or a polyhedron P and a specification of possible camera locations, find a minimum set-cover of the system $(P, \{V(c_i)\})$, where $V(c_i)$ is the set of points visible on P from camera c_i and the index i varies over all possible camera locations. The definition of $V(c_i)$ can capture any optical constraints on what a camera can see. We will refer to the specification of possible camera locations as a *setup*. We say a set S of cameras *cover* P if $\bigcup_{c_i \in S} V(c_i) = P$. Depending on the application, P may refer to the boundary of the object or it may be extended to include the interior of the environment as well.

Camera placement can also be seen as a particular case of the hitting set problem. The set X is the set of possible camera locations. For each point p on the boundary of the polyhedron P, there is a set R_p consisting of all camera locations that can see p. The hitting set problem assumes a finite set X and we have to implicitly deal with this issue when we attempt to pose camera placement as such a problem. Typically, we do so by using a sampling technique or by showing that a finite set of extremal points need to be considered.

Throughout this section we will represent cameras with their projection centers c_i and say that c_i sees the point $p \in P$ if the only intersection of the line segment pc_i with P is p. We extend the notion of visibility to sets as follows: We say that a camera sees a set of points ω if it can see all the points in ω. The following notation will be useful for VC-dimension proofs: Let $P_m = \{p_1, \ldots, p_m\}$ be m points on P. We say that camera c sees the subset $\omega \subseteq P_m$ if c can see all points in ω but no point in $P_m \setminus \omega$.

By the *VC-dimension of a setup*, we will refer to the VC-dimension of the maximum number of points that can be shattered over all instances of the camera placement problem for a specific setup. For example, if there are no restrictions on cameras

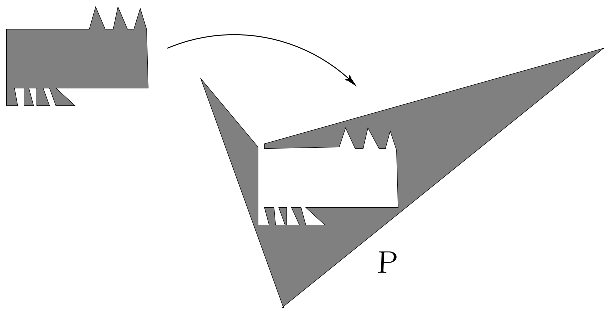


Figure 3.1: Interior visibility extends to exterior visibility by turning the polygon inside out.

and we want to cover simple polygons, we would like to find the VC-dimension of the set system $(P, \{V(c_i)\})$ as P varies over the set of all simple polygons. Therefore, in order to give a lower bound m on the VC-dimension of a setup it suffices to present one instance where m points are shattered, but for an upper bound m one needs to show that there exists no instance such that $m + 1$ points can be shattered.

3.2.3 Related work on VC-dimension

In general, it is possible to consider visibility systems as set systems and camera placement as a set-covering problem [35]. The general version of the minimum set-cover problem cannot be approximated better than a factor of $\log n$. However, as mentioned in the first section it is not clear that general set-cover instances can be realized by visibility systems.

Valtr proved that the VC-dimension of the system $(P, \{V(x) \mid \forall x \in P\})$, where P is a simple polygon and $V(x)$ is the visibility polygon of point x in P, is bounded between 6 and 23 [107]. This result applies to all 2D configurations we consider, simply by turning the polygon inside-out (see figure 3.1, also [94]). He also established a bound of $O(\log h)$ for polygons with h holes. A similar result for visibility under angle and distance constraints has been obtained in [37].

On the other hand, it is not clear how to exploit a bounded VC-dimension to obtain an improved approximation algorithm. Approximation algorithms for minimum guard coverage have been considered [35, 28, 37] for different versions of the problem. However there is still a gap between the inapproximability results and existing algorithms.

As mentioned before, the minimum set-cover of set systems with bounded VC-dimension can be approximated within a logarithmic factor of the optimal value [15]. However, this by itself does not improve on the existing $\log n$ approximations, as the optimum can be as big as $n/3$ [83]. Nevertheless, this algorithm was used in [37] to get rid of the dependency of the approximation factor to the number of samples (rather than the number of vertices). In fact, obtaining a constant approximation algorithm for guard placement in polygons without holes is one of main open problems in the field of art galleries.

For the setup where cameras are restricted to lie on a circle, an approximation algorithm that returns at most one more camera than the minimum necessary is presented in Section 3.6 A similar algorithm can be found in [78] for the polygon separation problem. Placing cameras outside the convex hull of a polygon is related to hitting lines with points [65, 14]. Unfortunately there are no improved algorithms for this restricted case either.

3.3 Results On Planar Configurations

3.3.1 2DCIRCLE

Consider a setup in which we want to cover a polygon P using cameras restricted to a circle C around P. Note that not all polygons are completely visible from a circle outside. In this section and the next, we restrict ourselves to the points on P that are visible from at least one point on the circle. Polygons that are entirely visible from the viewing circle are called externally visible polygons [43].

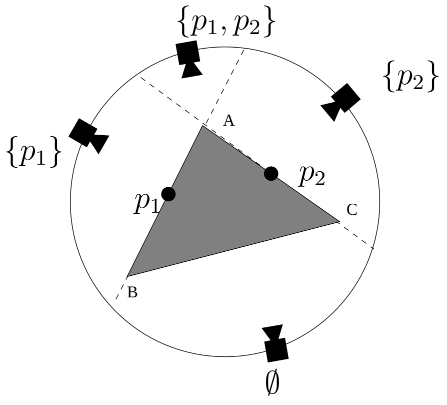


Figure 3.2: Points $\{p_1, p_2\}$ can be shattered by four cameras. Each camera is labeled with the subset it can see. In this figure the polygon P is $\triangle ABC$.

Definition 2 *We define 2DCIRCLE as a setup where a set of cameras whose locations are restricted to a circle C are to cover a simple polygon that is contained in C.*

We need the following technical lemma before proving our main proposition. Its proof is omitted because it is straightforward.

Lemma 3 *Each point p on the polygon P is visible along a continuous arc on the circle C and nowhere else.*

We now prove the following proposition.

Proposition 4 *The VC-dimension of 2DCIRCLE is exactly 2.*

Proof: For any m points on the polygon P, the m visibility arcs have $2m$ endpoints and hence there are only $2m$ combinatorially distinct camera positions. Since 2^m cameras are necessary to shatter m points, we need $2^m \geq 2m$ which is only true for $m \leq 2$.

The lower bound is proven by the example in Figure 3.2 where the points p_1 and p_2 are shattered by the 4 cameras shown. ∎

3.3.2 2DCONVEX

Let us now relax the restriction on camera locations so that we allow cameras anywhere outside the convex hull of the polygon.

Definition 5 *We define 2DCONVEX as a setup where a set of cameras located outside the convex hull of a simple polygon P are to cover P.*

The upper bound on the VC-dimension of 2DCONVEX slightly increases but it is still a small constant significantly less than the upper bound for the general case, 23.

Lemma 6 *The VC-dimension of 2DCONVEX is less than or equal to 5.*

Proof: Suppose that $Q = \{p_1, \ldots, p_6\}$ is a set of 6 points shattered on a polygon P. Let C be the boundary of conv$P$, the convex hull of P, and let E be the exterior of C, i.e. $E = R^2 \setminus (\text{conv}P)$. For each $i = 1, \ldots, 6$, a point of E sees the point p_i if and only if it lies between a certain pair of disjoint open half-lines emanating from C. We call these two open half-lines *the i-rays* and refer to the point on C from which they emanate as their endpoints. The 12 i-rays, $i = 1, \ldots, 6$, partition E into *cells*. Formally, cells are maximal connected components of the plane after the removal of *convP* and of the 12 rays.

We can also obtain the cells in the following way. One by one and in an arbitrary order, we remove the 12 rays from E. After the removal of the first ray there is still one component (cell). Each other ray divides every intersected cell into two smaller cells. Thus, if the ray intersects k of the previously removed rays, its removal increases the number of cells at most by $k + 1$. Since there are at most $\binom{12}{2} - 6 = 60$ intersections between the 12 rays (the two i-rays are disjoint for each i), the number of cells in the final arrangement is at most $60 + 11 = 71$.

It follows from the construction that cameras placed at different locations in the same cell see the same subset ω of $Q = \{p_1, \ldots, p_6\}$. We then call the cell an *ω-cell*.

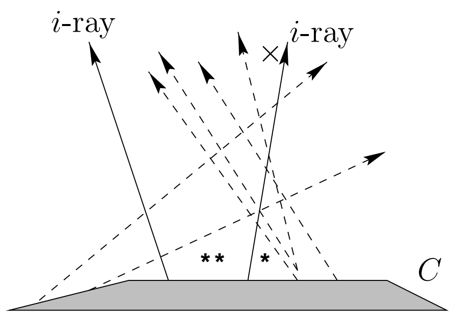


Figure 3.3: Case 1 of Lemma 6: The $\{p_i\}$-cell gets two marks and the $\emptyset$-cell gets one mark. The unbounded $\{p_i\}$-cell is marked $\times$.

To get a contradiction it suffices to show that there are no $2^6 - 1 = 63$ cells seeing distinct nonempty subsets of Q.

First, suppose that one of the points p_i lies on C. Then the two i-rays are parts of lines tangent to C. It is easily verified that each j-ray, $j \neq i$, is disjoint from at least one of the two i-rays in this case. Whenever a j-ray is disjoint from an i-ray, $j \neq i$, this decreases the number of cells in the final arrangement by 1. Thus, the number of cells is at most $71 - 10 = 61$ in this case, which is not enough.

Thus, we may suppose that each point p_i lies inside conv$P$. Whenever an i-ray is disjoint from a j-ray, $i \neq j$, for technical reasons we "create" a new, "abstract" cell and associate it with this pair of disjoint rays. The total number of "real" and "abstract" cells is exactly 71, provided no three rays intersect in a single point (otherwise it is smaller).

We suppose that every camera c_ω is placed in an unbounded cell whenever it is possible. We remove the camera $c_\emptyset$ and 63 cameras in 63 different cells remain. We say that a cell is *empty*, if it contains no camera. All "abstract" cells are empty.

We now describe a procedure which distributes 18 auxiliary marks $1, 1, 1, 2, 2, 2, \ldots, 6, 6, 6$ in some of the empty cells in such a way that at most two marks are placed in one cell. It will follow that at least $18/2 = 9$ cells are empty.

For each $i = 1, \ldots, 6$, the three marks i are distributed as follows. We say that an i-ray and a j-ray form an *i-pair*, if they are disjoint. We distinguish several cases.

Case 1: There is at most one i-pair. One of the i-rays is intersected by all j-rays for $j \neq i$. Hence its endpoint sees no point $p_j, j \neq i$. We place two marks i in the adjacent $\{p_i\}$-cell and one mark i in the adjacent $\emptyset$-cell (see figure 3.3). Note that the cell with two marks i is empty because it is bounded and there is an unbounded $\{p_i\}$-cell in this case.

Case 2: There are two i-pairs.

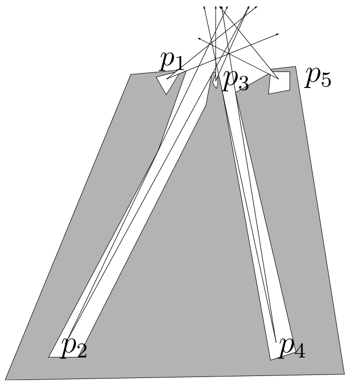
One mark i is placed in each of the two abstract cells associated with the i-pairs. The remaining, third mark i is put in an $\emptyset$-cell chosen as follows.

Subcase 2a: No point $p_j, j \neq i$, is visible from the endpoint e of one of the i-rays. In this case we put the third mark i in the $\emptyset$-cell adjacent to e.

Subcase 2b: Condition in Subcase 2a is not valid. In this case, note that each i-ray must be responsible for exactly one i-pair. Otherwise, one of the i-rays would intersect all other j-rays and the endpoint of this i-ray would only see p_i and this is covered in Subcase 2a. Let i_1 and i_2 be the two i-rays. Suppose that the endpoint of i_1 sees some $p_j, j \neq i$. Then the wedge of visibility of p_j must intersect i_1 in a bounded interval starting at the endpoint. For all other $k \neq j$, the k-rays intersect i_1 and it is immediate that this intersection is a bounded interval again. Thus, as we go to infinity along i_1, we must have the case that only p_i is visible on one side and no point is visible on the other side. We put the third mark i to this unbounded $\emptyset$-cell adjacent to i_1. Note that this $\emptyset$-cell is not adjacent to C, as can be verified from the analogous reasoning for i_2.

Case 3: There are three or more i-pairs. We put a mark i in three abstract cells associated with distinct i-pairs.

We now check that no cell receives more than two marks. Any abstract cell is associated with a pair of disjoint rays and may receive at most two marks corresponding to these two rays. A $\{p_i\}$-cell receives at most two marks i. An $\emptyset$-cell adjacent to C may receive at most two marks corresponding to the two rays having the endpoints on the boundary of the cell. An $\emptyset$-cell non-adjacent to C may receive



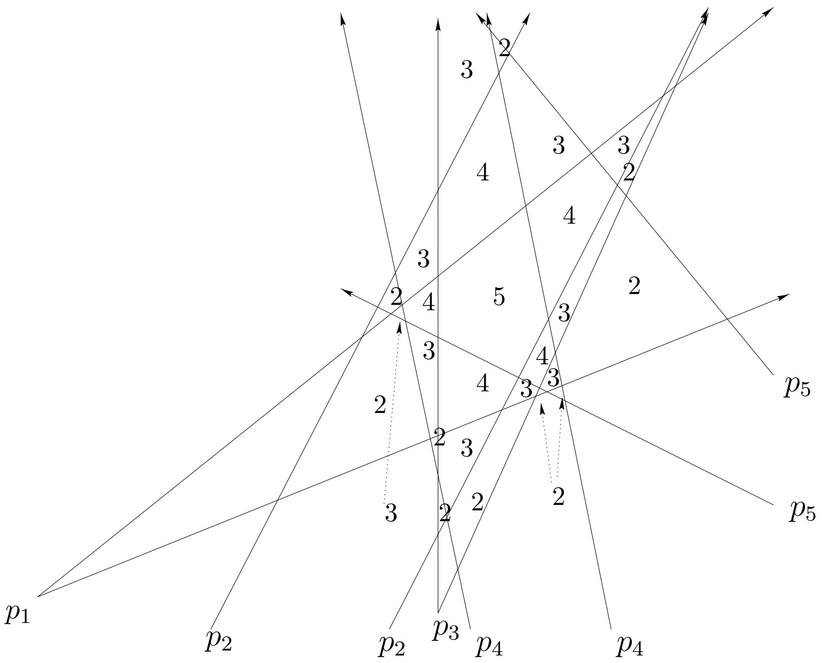


Figure 3.4: The small picture on the left shows a polygon with 5 shattered points $\{p_1, p_2, p_3, p_4, p_5\}$. The figure on the right is a detail showing the intersection pattern of the i-rays. Each label 2, 3, 4, 5 indicates the cardinality of the subset seen from the region.

at most two marks corresponding to rays forming unbounded edges of the cell. All other cells receive no marks.

Hence, no cell receives more than two marks. Therefore there are at least $18/2 = 9$ cells with at least one mark. They are all empty. Thus, at most $71 - 9 = 62$ cells are nonempty. They cannot contain all the 63 cameras $c_\omega, \emptyset \subset \omega \subseteq Q$. ∎

Note that relaxing the camera locations from 2DCIRCLE to 2DCONVEX indeed increases the VC-dimension, as we see in the following lemma.

Lemma 7 *The VC-dimension of 2DCONVEX is greater than or equal to 5.*

Proof: An instance where five points are shattered is presented in figure 3.4. ∎

The result of this section is summarized in the following theorem which follows from lemma 6 and lemma 7:

Theorem 8 *The VC-dimension of 2DCONVEX is exactly 5.*

Remark: If we further remove the restriction that the cameras are outside the convex hull, then the best known bound is 23 and the reader is referred to [107].

3.4 Results on 3D Configurations

In this section, we consider the following setup which arises in typical tele-immersive applications:

Definition 9 *We define* $3DSPHERE$ *as a setup where we are given a polyhedron* $\mathcal{P}$*, and a viewing sphere* $\mathcal{S}$ *such that* $\mathcal{P}$ *is totally contained in* $\mathcal{S}$*.*

We show that even when the viewing region is restricted to a sphere, there are polyhedra with n vertices such that $\Theta(\log n)$ points on the polyhedron can be shattered from the viewing sphere $\mathcal{S}$ that contains $\mathcal{P}$. Namely, we prove the following theorem:

Theorem 10 *The VC-dimension of 3DSPHERE is* $\Theta(\log n)$*.*

In the next two subsections, we present the upper and lower bounds for the VC-dimension of 3DSPHERE, in lemmas 11 and 12, respectively.

3.4.1 Upper bound

In this section we present an upper bound on the VC-dimension of 3DSPHERE. In [88], Platinga and Dyer define *aspects* as changes in the topology of the image of a polyhedron. After presenting a catalogue of events (structural changes in the image of the polyhedron), they construct the viewing space partition, VSP, which is a partition of the viewpoint space into maximal regions of constant aspect. They also present tight bounds for the number of regions in VSP. They show that the size of the VSP for a general (i.e. non-convex) polyhedron under orthographic projection is $\Theta(n^6)$ and their model for the orthographic projection is exactly the same as 3DSPHERE with $\mathcal{S}$ at infinity.

Lemma 11 *The VC-dimension of 3DSPHERE is* $O(\log n)$*.*

Proof: Let $P_m = \{p_1, \ldots, p_m\}$ be any m points to be shattered on a polyhedron. If we define an aspect as appearance/disappearance of $p_i, i = 1, \ldots, m$, and restrict the camera locations to a sphere that contains the polyhedron, we can use the catalogue of events in [88] to show that the size of the VSP for this new notion of aspects is still $\Theta(n^6)$. However, in order to shatter m points, one needs 2^m distinct partitions. Since we must have $2^m = O(n^6)$, we have $m = O(\log n)$ which gives us the desired upper bound. ∎

3.4.2 Lower bound

In this section we show that the upper bound $\log n$ on the VC-dimension of 3DSPHERE is indeed tight, our main result is stated in the following lemma for theorem 10.

Lemma 12 *For any given m, there exists a polyhedron* $\mathcal{P}$ *with* $\Theta(2^m)$ *vertices such that there are m marked points on* $\mathcal{P}$ *that can be shattered from* 2^m *cameras on the viewing sphere* $\mathcal{S}$*.*

Proof: We take a regular simplex (tetrahedron) T inside a viewing sphere S. Let F be a facet (2-dimensional face) of T. We take a set M of m points slightly above the center of F. Further, we place 2^m cameras $c_\omega, \omega \subseteq M$, on S visible from each point of M. We choose them so that no camera lies on a line determined by two points of M.

Then, for any $p \in \omega \subseteq M$, we choose a segment $s(p, \omega)$ having one endpoint on the face F and intersecting the segment pc_ω and no other segments $p'c_{\omega'}, p' \in M, \omega' \subseteq M$. Clearly, we may choose the segments $s(p, \omega)$ pairwise disjoint. Then we replace each $s(p, \omega)$ by a very thin simplex $S(p, \omega)$ growing out of the face F and intersecting the segment pc_ω and no other segments $p'c_{\omega'}, p' \in M, \omega' \subseteq M$. If the simplices $S(p, \omega)$ are thin enough, then they are pairwise disjoint.

Let P be the union of T with the $m \cdot 2^{m-1}$ simplices $S(p, \omega)$. Then P shatters the set M and has $m \cdot 2^{m+1} + 4$ vertices. It has a facet F' obtained from F by the removal of $m \cdot 2^{m-1}$ triangular holes. We arbitrarily triangulate F'. If we now perturb the vertices of F', then P may be changed to a simplicial polyhedron $\mathcal{P}$ such that F' is replaced by triangular facets corresponding to the chosen triangulation of F'. ∎

Proof of theorem 10: Theorem 10 is a direct consequence of lemmas 11 and 12. ∎

3.5 Sensor Deployment by Sampling

In this section, we describe how VC-dimension properties of set systems can be utilized in sensor-network deployment. Consider a scenario where deployment has to be accomplished in one step and no intermediate information becomes available during deployment. Further, suppose we can not place the sensors precisely. This can happen, for example, when we deploy the network from an airplane. In the worst case, we may have no control over the deployment and each sensor can fall onto a random location. If this is the case, the crucial question becomes: How many random samples are needed to cover the environment?

In answering this question, the first step is to model the device constraints, typically using geometric objects. For example, one can model range constraints by representing sensors with disks or field of view constrains using triangles. Line-of-sight or visibility type of constraints can be modeled with star shaped polygons.[1] We will need the following definition:

Definition 13 *$N \subseteq X$ is an ϵ-net for $(X, \mathcal{R})$, if for all $R \in \mathcal{R}$ with $|R| \geq \epsilon$, we have $N \cap R \neq \emptyset$.*

[1]A star-shaped polygon is a polygon which contains at least one point from which the entire boundary is visible.

Now suppose we would like to cover a square region A of unit area with disks of area ϵ. Consider the set system (X, R), where X is the (infinite) set of all disks with centers inside A and for every point p in A, there is a subset $R(p)$ in R that contains the disks that intersect p. Note that $R(p)$ itself can be represented by a disk of area ϵ. Therefore an ϵ-net for (X, R) intersects all the subsets in R, meaning that all points in A are covered by the ϵ-net. It is worth emphasizing that all the sets mentioned above are infinite. The crucial property here is that the set system itself has a *constant* VC-dimension:

Theorem 14 ([42]) *Let d be the VC-dimension of $(X, \mathcal{R})$ and $\delta > 0$. Then, $O(\frac{d}{\epsilon}\log\frac{d}{\epsilon} + \frac{1}{\epsilon}\log\frac{1}{\delta})$ points drawn at random from $\mathcal{R}$ is an ϵ-net with probability at least $1 - \delta$.*

Therefore using theorem 14, we can sample a constant number of points (for a given ϵ), independent of the size of X (which is infinite most of the time) and obtain an $\epsilon-$net with high probability.

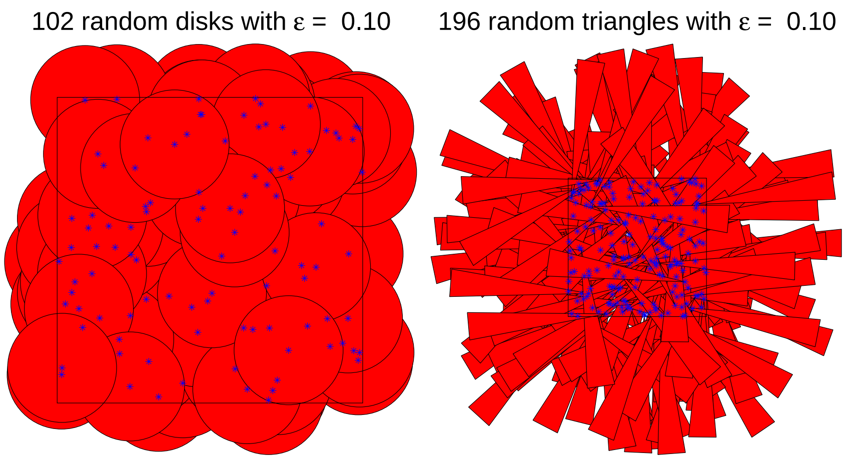

102 random disks with ε = 0.10
196 random triangles with ε = 0.10

Figure 3.5: Coverage with sampling

Figure 3.5 shows some examples of coverage with random samples. In the left figure a unit area is covered with disks of radius $\epsilon = 0.1$. In this simulation

$d/\epsilon \log d/\epsilon = 102$ disks (with $d = 3$) were placed at locations chosen uniformly at random from $[0, 1]^2$. In the right figure, a similar experiment was performed for identical triangles that cover the same area. However in this case, 196 sensors are required as the triangle set system is a more complicated one with VC-dimension 5.

We conclude this section by giving examples of set systems with bounded VC-dimension: Half-plane set systems have VC-dimension 3, arbitrary convex k-gons have VC-dimension $2k + 1$ which can be obtained easily. However, determining the VC-dimension of set system can be rather tricky. A good example is [107] where the VC-dimension of visibility set systems for simply connected regions is shown to be a constant.

Of course, there are cases where we can have much more control over the deployment. In such cases, we can have a better estimate for the number of sensors. In the next section, we present an algorithm that computes almost the optimal number of sensors required for covering an object from a circle.

3.6 Placing Sensors on a Circle

In this section we present an efficient placement algorithm for the setup $2DCIRCLE$. The algorithm PlaceCircleGuards places guards so that every point p on the polygon that is visible from at least one point on the circle is seen from some guard.

As shown in Lemma 3 the region of C from which any particular point p on P is visible is a continuous arc.

Definition 15 *Let q be a point on P. The visibility arc of q, A_q, is the set of points on C that can see q. A_q is called a minimal arc if there is no point p on P such that the arc A_p is properly contained in A_q.*

Alternatively, the guard placement problem can be viewed as the problem of hitting the set of minimal arcs.

It is easy to see that hitting intervals on a line using minimum number of points can be solved optimally by a greedy algorithm: Order the intervals with respect to their right endpoints. Place a point to the leftmost right-end of an interval that is not covered.

In order to hit the set of minimal arcs we start with an arbitrary point on c which moves clockwise on C. We will maintain two lists:

- **Covered:** Contains all the portion of the boundary of P that is currently covered. Initially empty.
- **Active:** The parts of the boundary that have been seen by c but not currently covered. Initially contains the segments observed from the initial position of c.

For now, assume continuous motion of c. The algorithm proceeds as follows:

```
When $c$ moves to the point $c'$ on $V$
If new boundary becomes visible from $c'$, add it to *Active.*
If a point in *Active* is not visible from $c'$,
    Place guard $g$ at $c'$.
    Add the part of the boundary visible from $g$ to *Covered.*
    Remove the part of the boundary visible from $g$ from *Active.*
$c \leftarrow c'$
Repeat until *Covered* $= P$.
```

Obviously, we can not move c to all points on C. The main challenge in this algorithm is to find a small finite set of "events" of interest along C. Let us define appearance/disappearance of a point p on P as c moves in clockwise direction as a *visual event.* In the next section we show that some visual events necessarily occur before others. Therefore we can consider finitely many locations of c.

3.6.1 Visual events

By Lemma 3 for each point p there is an arc $A_p = \widehat{ab}$ on which p is visible. We call a the *entrance* of the arc and b the*exit.*

Definition 16 *Let e be a line segment (either an edge or a portion of an edge) on the boundary of P. Then the weak visibility region of e is defined as $\bigcup_{p \in e} A_p$.*

Let e be a line segment on the boundary of P. We can rotate P so that e is horizontal and the exterior of P is locally above e. We will call this the canonical orientation of P with respect to e. It is clear that in this orientation, the interior of e can only be seen above the horizon defined by e. For two points a, b on the viewing circle C and above this horizon we say that $a \leq b$ if a appears clockwise of or coincides with b.

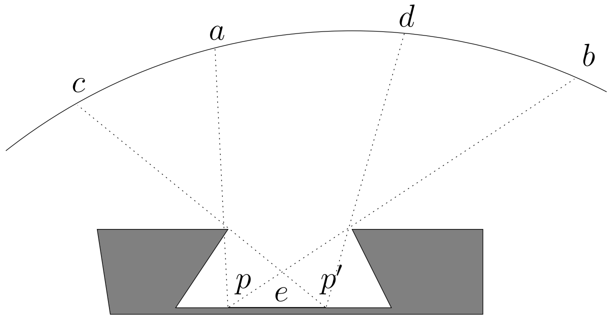


Figure 3.6: p is to the left of p' implies $c \leq a$ and $d \leq b$.

Lemma 17 *Let P be oriented canonically with respect to line segment e on the boundary. Let p, p' be points in the interior of e with p to the left of p'. Let $A_p = \widehat{ab}$ and $A_{p'} = \widehat{cd}$ (see figure 3.6). Then $c \leq a$ and $d \leq b$.*

Proof: Suppose for contradiction that $a < c$. Then there is a point a' between a and c that can see p but cannot see p'. Clearly the boundary of the polygon must intersect the segment $\overline{a'p'}$. By definition the boundary of the polygon cannot intersect the wedge formed by c, d, and p' which lies to the right of this segment. Thus the boundary of P must intersect $\overline{a'p'}$ from the left. This will force it to first intersect the segment $\overline{a'p}$ contradicting the assumption that p is visible from a'. Thus we have a contradiction. A symmetric argument shows that $d \leq b$. ∎

A useful corollary of Lemma 17 is that if $[p, q]$ is a segment on the boundary of P, then in any clockwise traversal starting from some point that does not see $[a, b]$, b appears before any point in (a, b) and a disappears after all points in (a, b).

Lemma 18 *The weak visibility region of a line segment on P is a continuous arc.*

Proof: Let p and p' be two points on a line segment on the boundary of P. Assume that we have canonical orientation and p lies to the left of p'. We show the lemma by showing that one of the following must hold: Either A_p and $A_{p'}$ intersect or, for any point x on C which lies strictly between $A_{p'}$ and A_p x can see some point on e between p and p'.

Suppose A_p and $A_{p'}$ do not intersect. Then by Lemma 17 $A_{p'} = \widehat{cd}$ is to the left of A_p. Let the segments $\overline{dp'}$ and $\overline{ap}$ intersect at point o_1 and the segments $\overline{cp'}$ and $\overline{bp}$ intersect at point o_2. Suppose x is point between d and a and x is completely blocked from seeing the segment $\overline{pp'}$. Then either there is a portion of the boundary of P in the sector defined by d, a, and o_1 or there is a portion of the boundary of P in the sector defined by p, p', and o_2. Either of these possibilities leads to a contradiction with respect to the known visibilities. ∎

3.6.2 The Algorithm PlaceCircleGuards

In this section we present the algorithm *PlaceCircleGuards* that mimics the greedy hitting set algorithm outlined in the previous section.

Proposition 19 *PlaceCircleGuards uses at most one more guard than optimal.*

Proof: After PlaceCircleGuards has placed the first guard at point a on C we remove all arcs that pass through a. At this point the C can be "opened" at a and the remaining arcs can be viewed as intervals on a line. PlaceCircleGuards now mimics the IntervalHittingSet algorithm which is optimal for this problem. ∎

PlaceCircleGuards(C, P)
Input: A circle C and a polygon P
Output: a set of points, G, on C such that G covers P.

```
$G \leftarrow \emptyset$
pick an arbitrary point $p$ on $C$
$Active \leftarrow$ {chains on $P$ visible from $p$}
$Covered \leftarrow \emptyset$
$Q_a \leftarrow \emptyset$
for each vertex $v$ of $P$
    Find the visibility arc $A_v$ of $v$
    $Q_a \leftarrow Q_a \cup \{A_v\}$
Sort $Q_a$ according to their exits
(*from now on we keep $Q_a$ always sorted*)
while $Covered \neq P$
    for each chain $s \in Active$
        Let $a, b$ be the endpoints of $s$
        Find the visibility arcs $A_a$ and $A_b$
        $Q_a \leftarrow Q_a \cup \{A_a, A_b\}$
    $p \leftarrow Head(Q_a).exit$
    $G \leftarrow G \cup \{p\}$
    remove all the arcs that intersect $p$ from $Q_a$
    add the portions of boundary visible from $p$ to $Covered$.
```

Table 3.1: Algorithm PlaceCircleGuards.

3.6.3 Analysis

In this section, we show that the running time of PlaceCircleGuards is polynomial by showing that the number of line segments in the lists *Covered* and *Active* is bounded by a polynomial.

Definition 20 *We say a guard g splits a line segment s if s splits into two disjoint line segments s_1 and s_2 after the removal of the portion of s visible from g. s_1 and s_2 are called the children of s.*

Lemma 21 *Let e be an edge of P. During the course of the algorithm PlaceCircleGuards e may be split at most once and none of the children of e is split afterwards.*

Proof: Initially an edge may be split if the locations of the current set of guards do not intersect the visibility arcs of its endpoints. However, as the algorithm proceeds clockwise, none of the children will be split due to lemma 17. ∎

Corollary 22 *At any instance of the algorithm PlaceCircleGuards the size of the list that contains visible line segments is at most* $2n$.

3.7 Concluding Remarks

Visibility of a polygonal or polyhedral configuration can be cast as a set system with subsets defined by the visibility region of each camera. The minimal guard coverage problem can then be formulated as a minimum set-cover problem. The constraints imposed by the geometry of the setup can be captured with the VC-dimension of the visibility set system. It was known [107] that the upper bound of the VC-dimension for polygons is 23. In this work, we improved this bound for two cases of exterior visibility: cameras on a circle containing the polygon and cameras outside the convex hull of a polygon. The circle case has significant practical implications because it minimizes the number of stations of a turn-table or a laser-scanner pedestal in 3D-modeling and object inspection. For this case, we present an algorithm that uses at most one more sensor than the optimal number of sensors necessary to capture the object. The placement of cameras outside the convex hull is significant in surveillance and image based rendering. For this case as well as the case of arbitrary placement inside a polygon the existence of approximation algorithms with constant ratio is still an open problem.

In the 3D case, we showed that for any n, a polyhedron with n vertices can be constructed such that the VC-dimension of its exterior visibility is $\Theta(\log n)$. These results provide insights for sampling based sensor deployment algorithms.

Chapter 4

Sensor Assignment

Primary applications of sensor networks are in surveillance and monitoring tasks where targets are tracked by the sensors. In this chapter, we study a fundamental problem that arises in such scenarios: How should we assign sensors to targets to minimize the error in estimating the position of the targets?

In an adversarial setting, targets must be tracked in order to monitor suspicious activity. However, target tracking need not be adversarial. For example, a sensor network can track mobile robots so that the mobile robots can localize themselves. Such tracking tasks also arise in location-aware computing [102] where devices query sensors to localize themselves.

Unfortunately, the sensors used for these tasks are inherently limited, and individually incapable of estimating the target state. Without additional constraints, a minimum of two bearing sensors (such as cameras) are required to estimate the position of a target. For range sensors, three are required to localize a target (although this can be reduced to two using filtering techniques). Noting that the measurements provided by these sensors are also corrupted by noise, we realize that the choice of which measurements to combine can greatly influence the accuracy of our tracking estimates.

Consider a distributed set of such sensors charged with tracking groups of targets

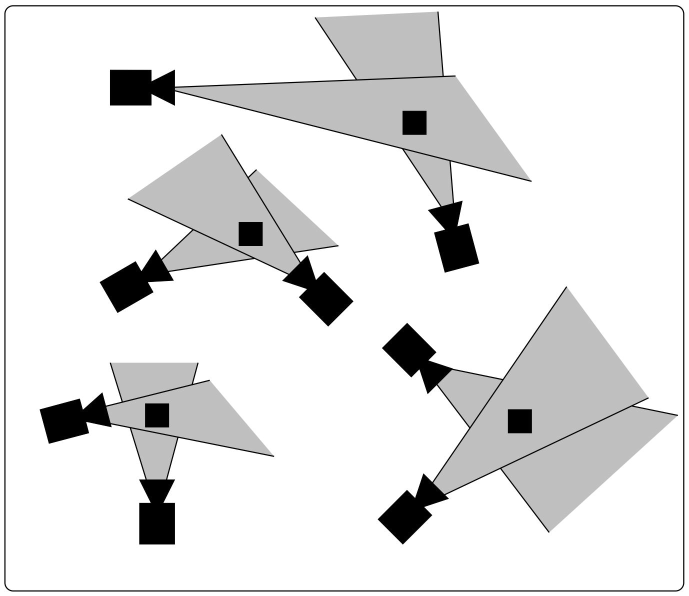


Figure 4.1: An instance of the Focus of Attention problem where eight cameras are tracking four targets. Two cameras are needed to estimate the position of a target and a camera can not track more than one target.

as illustrated in Figure 4.1. It would be unrealistic to assume that each sensor could track multiple targets or that the network possessed unlimited computational power and bandwidth. With this in mind, our problem can be viewed as an optimal allocation of resources for target tracking. How should pairs of sensors be assigned to targets so that the sum of errors in target position estimates is minimized? We refer to this as the *focus of attention* problem for distributed sensors.

4.1 Related Work

Since the measurements of multiple sensors are combined to estimate target pose, our work relates strongly to research in sensor fusion. Fusing measurements from multiple sensors for improving tracking performance has been the subject of significant research [7]. However, the focus has been on combining measurements from sensors (radars, laser range-finders, etc.) individually capable of estimating the target state (position, velocity, etc.). As our sensors require the fusion of pairs of measurements, we desire instead an optimal assignment of *disjoint sensor pairs* to targets. This added dimension changes the complexity of the problem entirely, and distinguishes our work from previous approaches.

Within the robotics community, Durrant-Whyte *et al.* pioneered work in sensor fusion and robot localization. This yielded significant improvements to methods used in mobile robot navigation, localization and mapping [75, 27]. Thrun *et al.* have also contributed significant research to these areas [103, 105]. However, our work distinguishes itself from traditional data fusion techniques in that the sensors themselves are actively managed to improve the quality of the measurements obtained *prior* to the data fusion phase, resulting in corresponding improvements in state estimation.

There has been other related research under the heading of sensor networks. Cortes *et al.* investigated the issue of sensor coverage [23]. In their model a single sensor is sufficient to cover a point. However, the quality of coverage decreases with distance. Since in their model the sensors are mounted on mobile robots, their work focused on the movement of sensors while ensuring optimal coverage. Our work begins where the sensor coverage problem leaves off and is applicable when multiple sensors are required for monitoring a single region. Jung and Sukhatme examined a heterogeneous network of static and mobile sensors for target tracking [60]. Using a region based approach, each robot attempts to maximize the number of tracked targets per region. In contrast to our work, data fusion issues were not considered. Lastly, Horling *et al.* [48] focused on network management optimization to ensure target observability and synchronized sensor observations in order to better estimate target position. In sharp contrast, our approach optimizes explicit sensor error metrics to obtain an optimal or near optimal sensor-target assignment.

4.2 The Focus of Attention Problem

The following are the standard definitions used for analysis of approximation algorithms [46] that will be used in the chapter:

Definition 23 *A polynomial time algorithm, $\mathcal{A}$, is said to be an α-approximation algorithm, if for every problem instance I, $\mathcal{A}$ produces a solution within a factor of α of the optimal solution.*

We say a problem is *inapproximable*, if for any α, obtaining an α-approximation algorithm for the problem is NP-hard.

Definition 24 *A polynomial-time approximation scheme (PTAS) is a family of algorithms $\mathcal{A}_\epsilon : \epsilon > 0$ such that for each $\epsilon > 0$, $\mathcal{A}_\epsilon$ is a $(1+\epsilon)-$approximation algorithm which runs in polynomial time in input size for fixed ϵ.*

4.2.1 Problem definition

The focus of attention problem (FOA) is formally defined as follows: The input consists of n targets, $2n$ sensors and a cost function $c(i, j, k)$ which indicates the cost of tracking target k using sensors i and j where $i, j \in [1 \dots 2n]$ and $k \in [1 \dots n]$. In the sequel, this cost represents the expected error associated with a position estimate obtained by fusing the information from sensors i and j. We are required to output an assignment: a set of n triples such that each target is tracked by two sensors, no sensor is used to track more than one target and the sum of errors associated with triples is minimized.

FOA is closely related to the following problem [34]:

Definition 25 (3D-Assignment) *Given three sets X, Y and W and a cost function $c : X \times Y \times W \rightarrow N$, find an assignment A (that is a subset of $X \times Y \times W$ such that every element of $X \cup Y \cup W$ belongs to exactly one element of A) such that $\sum_{(i,j,k) \in A} c(i, j, k)$ is minimized.*

3D-Assignment (3DA) is NP-hard [24] and inapproximable [18]. It is easy to see that any instance of 3DA can be reduced to an instance of FOA just by setting $c_{FOA}(i,j,k) = c_{3DA}(i,j,k)$ whenever $c_{3DA}(i,j,k)$ is defined and infinite otherwise. Moreover, since this reduction is approximation preserving, FOA with arbitrary costs is not approximable as well.

However, usually the error is not arbitrary but a function of the camera/target geometry. As we shall see in the preceding sections, it is possible to obtain efficient algorithms for specific geometries and error functions. We start with cameras placed on a single line.

4.2.2 Cameras on a line

In this section, we consider collinear cameras located on line l tracking targets on the plane. The error associated with cameras i and j tracking target k is $\frac{Z_k}{l_{ij}}$ where Z_k is the vertical distance of the target k to the line l and l_{ij} is the baseline, the distance between the two cameras (see Figure 4.2). This metric can be used to gauge the error in the stereo reconstruction [1] and gives a good approximation when the targets are not too close to the cameras. Note that this error metric fails if the targets are very close to the line l, therefore in this section we assume that there exists a minimum clearance δ such that $Z_i > \delta$, for all targets i.

Suppose that the cameras are sorted from left to right and let c_i be the coordinate of the i^{th} camera. The following lemma enables us to separate matching cameras from matching targets to pairs.

Lemma 26 *Let Z_i be the depths of targets, $Z_1 \leq Z_2 \leq \ldots \leq Z_n$ and l_i be the baselines in an optimal assignment sorted such that $l_1 \leq l_2 \leq \ldots \leq l_n$. There exists an optimal matching such that the target at depth Z_i is assigned to the pair with*

[1]In fact, a better approximation is $\frac{Z_k^2}{l_{ij}}$. However, when all the cameras are collinear, the depth of a target is the same for all cameras and therefore for simplicity we assume that the depths are squared and the error is $\frac{Z_k}{l_{ij}}$. The derivation of this error measure can be found in Section 4.3.

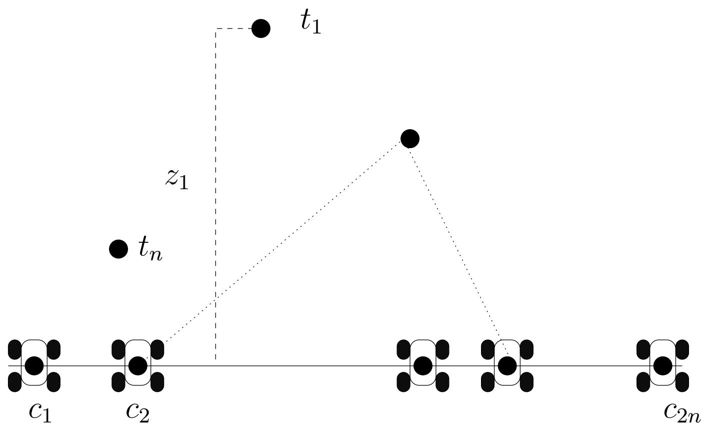


Figure 4.2: The Focus of Attention Problem on the line.

baseline l_i.

Proof: Suppose not. Then, in the optimal solution there exists two assignments (Z_i, l_j) and (Z_k, l_i) such that $Z_i > Z_k$ and $l_j < l_i$. Consider a new assignment obtained by modifying the optimal assignment by switching the targets of the two assignments. Since we have,

$$\begin{aligned} \frac{Z_i}{l_j} + \frac{Z_k}{l_i} &> \frac{Z_i}{l_i} + \frac{Z_k}{l_j} \\ \frac{(Z_i - Z_k)}{l_j} &> \frac{(Z_i - Z_k)}{l_i} \end{aligned}$$

this new assignment produces less error than the optimal solution – a contradiction! ∎

Lemma 26 implies that once we choose pairs of cameras, the targets can be assigned to these pairs in a sorted fashion where further targets are assigned to pairs with larger baselines.

Performance of known heuristics

It is easy to see that a greedy assignment that assigns the furthest target the maximum available baseline can be arbitrarily far from optimal value: Consider the setting in Figure 4.3 with four cameras where the two cameras in the middle are very close

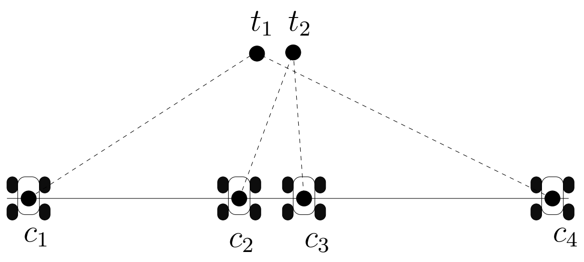


Figure 4.3: A greedy assignment assigns c_1 and c_4 to target t_1 and gets stuck with the pair (c_2, c_3). The optimal assignment in this case is to assign t_1 to (c_1, c_3) and t_2 to (c_2, c_4).

to each other. In this configuration, the greedy algorithm can produce an assignment that is arbitrarily more costly than the optimal assignment: $(t_1, c_1, c_3), (t_2, c_2, c_4)$.

Perhaps not so obvious is the performance of the following algorithm: Find a matching between the cameras that maximizes the sum of the baselines and assign these pairs to targets. This algorithm, which we call *Match-Assign*, gives a 3/2 approximation for the 3D-Assignment problem when the cost of a triple is the perimeter of the triangle formed by the points in the triple [24].

The Match-Assign Algorithm can also be arbitrarily bad: Suppose there is one target at $Z = \mathcal{Z}$ and $n-1$ targets at $Z = \epsilon$. Each camera c_i is located at $x = \frac{i}{2n-1}$, $i = [1, \ldots, 2n]$.

First consider the matching (c_1, c_{2n}) and (c_{4i-2}, c_{4i}), (c_{4i-1}, c_{4i+1}) for $i = 1, 2, \ldots, (2n-2)/4$. The cost of this matching is $1 + \frac{2(n-1)}{2n-1} \approx 2$ and the total error is $\mathcal{Z} + (n-1)\frac{2\epsilon}{2n-1}$.

Next, consider the matching that matches c_i with c_{i+4}. The cost of this matching is also $\frac{4n}{2n-1} \approx 2$ but the total error is $(2n-1)\frac{\mathcal{Z}}{4} + (n-1)\frac{(2n-1)\epsilon}{4}$.

Therefore, two matchings with equal sum of baselines may lead to errors such that one can be made arbitrarily larger than the other and the Match-Assign algorithm cannot be used to obtain a good approximation.

A 2-Approximation Algorithm

In this section we present a 2-approximation algorithm for the previous assignment problem. The algorithm simply assigns camera i to camera $n+i$ and these pairs are then assigned to the targets according to Lemma 26. The algorithm is summarized in Table 4.1. Let l_i be the baselines generated by our algorithm. Let OPT be any optimal solution and l_j^* be the baselines in OPT. The following lemmas show that we can find a one-to-one correspondence between l_i and l_j^* such that l_i are longer than half of their corresponding pairs in the optimal solution.

```
CamerasOnALine($c_1, \ldots, c_{2n}$: camera positions,
        $z_1, \ldots, z_n$: target depths )
for $i$=1 to n
   $p_i \leftarrow (c_i, c_{n+i})$
$\{p'_i\} \leftarrow$ Sort $\{p_i\}$ so that $p'_i$ is the $i^{th}$ largest baseline
$\{z'_i\} \leftarrow$ Sort $\{z_i\}$ so that $z'_i$ is the $i^{th}$ largest depth
for $i$=1 to n
   assign $p'_i$ to $z'_i$
```

Table 4.1: A 2-approximation algorithm for assigning cameras on a line to targets.

Lemma 27 *For all i, there exists an index j such that $l_i \geq l_j^*$.*

Proof: Let k be the the pair such that $|(c_k, c_{n+k})| = l_i$.

Let $A = \{c_k, c_{k+1}, \ldots, c_{n+k}\}$. Since $|A| = n + 1$, in the optimal matching there must be two cameras in A that match with each other and the baseline of that match is at most l_i. ∎

Lemma 28 *Let $S = \{l_1, \ldots, l_n\}$ and $OPT = \{l_1^*, \ldots, l_n^*\}$. For any $A \subseteq S, |A| = k$, there exists a subset $B \subseteq OPT, |B| = k$ and a bijection $\sigma_k : A \rightarrow B$ such that $l_i \geq \sigma_k(l_i)/2$ for all $l_i \in A$.*

Proof: The lemma is proven by induction on $|B| = k$.

Basis: Existence of σ_1 for $k = 1$ is a corollary of Lemma 27.

Inductive Step: Let c_i and c_j be the leftmost and rightmost cameras used by the edges in A. Without loss of generality, assume that $|c_i c_{n+i}| \geq |c_j c_{n+j}|$. Let Y be the subset of pairs in OPT that matches cameras in the set $C = \{c_i, c_{i+1}, \ldots, c_j\}$.

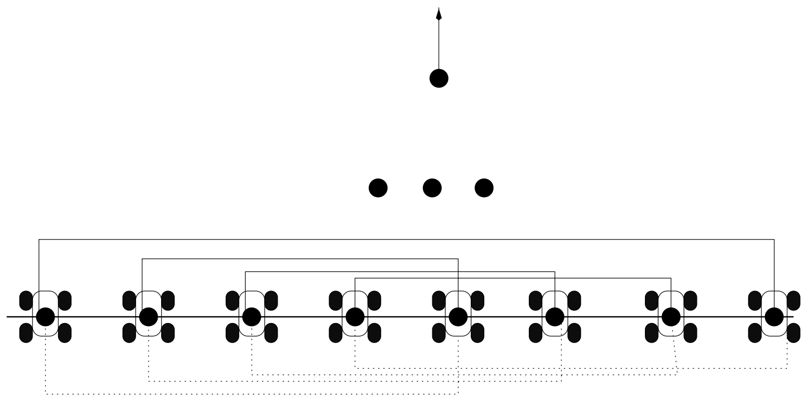


Figure 4.4: The matchings produced by our algorithm (shown in dotted lines) can be twice as bad as the optimal matching (shown in solid lines) by moving the furthest target to infinity.

We first observe that $|Y| \geq k$. This is because $|C| \geq n + k$ and hence at most $n - k$ cameras in C could be matched by OPT to cameras outside C.

The longest edge in B is easily seen to be at most $2|c_i c_{n+i}|$. We now recursively compute σ_{k-1} for $A' = A \setminus \{(c_i, c_{n+i})\}$. Let B' be the range of σ_{k-1}. Since $|Y| \geq k$, Y must have at least one pair, say l^*, not in B'. We match this pair to (c_i, c_{n+i}):

$$\sigma_k(l) = \begin{cases} \sigma_{k-1}(l), & \text{if } l \in A' \quad (4.1) \\ l^*, & \text{if } l = (c_i, c_{n+i}) \quad (4.2) \end{cases}$$

■

Therefore by Lemma 28 there exists a mapping σ from S to the optimal matching such that $l_i \geq \frac{\sigma(l_i)}{2}, \forall l_i \in S$ which gives us the desired approximation guarantee. This analysis is tight, there are instances where our algorithm can be twice as costly as the optimal cost:

The tight example consists of $n/4$ cameras at $x = 0$, $n/4$ cameras at $x = 1 - \epsilon$, $n/4$ cameras at $x = 1 + \epsilon$ and $n/4$ cameras at $x = 2$. There is one target at $Z = \mathcal{Z}$ and $n - 1$ targets at $Z = \epsilon$ (see Figure 4.4).

The optimal cost in this case is $\frac{\mathcal{Z}}{2} + (n-2)\frac{\epsilon}{1+\epsilon} + \frac{\epsilon}{2\epsilon}$. This is achieved by matching c_1 to c_{2n} and $c_{\frac{n}{4}+1}$ to $c_{\frac{3n}{4}}$ and imitating our algorithm otherwise.

Our cost in this case is is $\frac{\mathcal{Z}}{1+\epsilon} + (n-1)\frac{\epsilon}{1+\epsilon}$ which becomes twice the optimal value as $\mathcal{Z}$ grows to infinity.

We summarize the main result of this section in the following theorem.

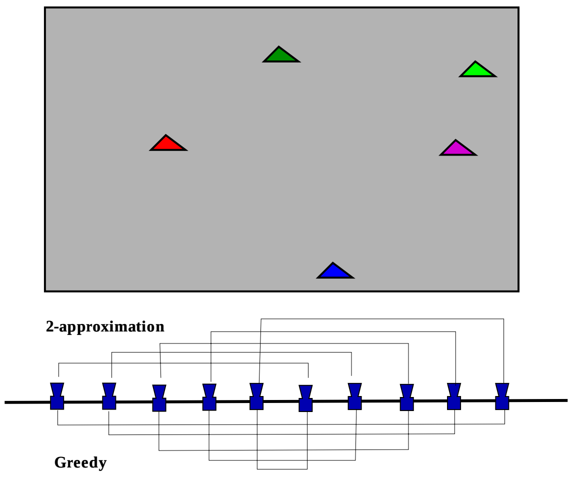
Theorem 29 *There exists an $O(n \log n)$-time algorithm that simultaneously gives a 2-approximation to minimizing the sum of errors metric as well as minimizing the maximum error metric when the cameras are aligned and the cost of assigning cameras i and j to target k is $\frac{Z_k}{l_{ij}}$ where l_{ij} is the distance between the cameras and Z_k is the distance of target k to the line that passes through the cameras.*

Simulation results

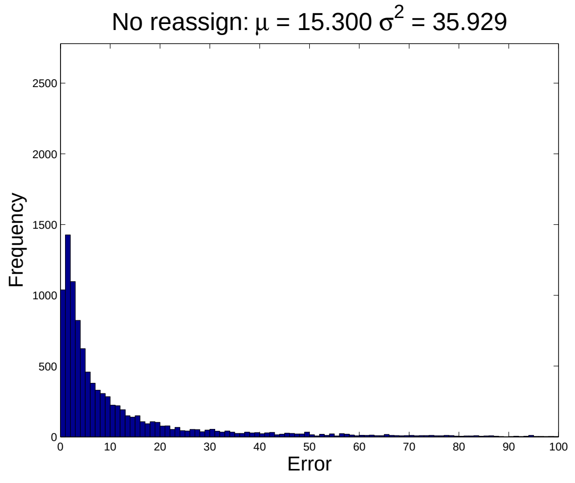
In this section, we present simulation results that contrast the performance of the 2-approximation algorithm empirically with a static and a dynamic/greedy approach. The simulation models the target tracking task as outlined in Section 4.2.2. Specifically, we consider 10 cameras charged with tracking 5 targets performing a random walk as shown in Figure 4.5 (a). The sensors measure bearings to targets. We assume that the sensor locations are known without error, and each sensor is constrained to tracking a single target at any given time. Measurements from pairs of sensors are then merged (via triangulation) to obtain an estimate of the position of the target. We modeled this scenario for two different algorithms.

Algorithm 1 initially assigned each target to the best available pair and kept this assignment fixed throughout the simulation. Algorithm 2 employed a greedy reassignment strategy whereas Algorithm 3 employed the 2-approximation algorithm presented in Section 4.2.2. In this approach, sensor pairs communicated target position estimates (requiring $O(n)$ communications), and sensor pair-target assignments were dynamically updated as necessary.

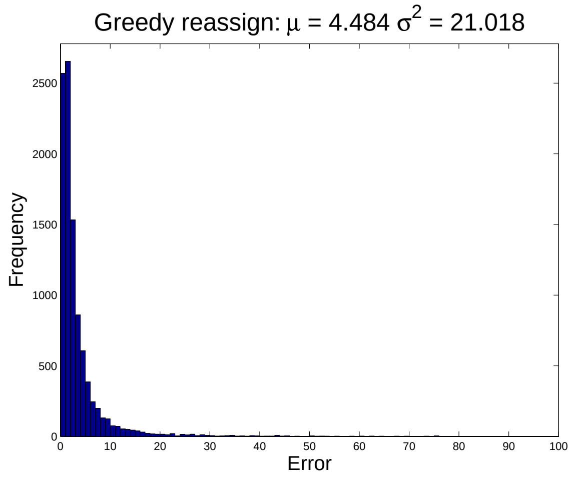
We simulated the performance of these three algorithms for 10000 iterations. The error in bearing was simulated by drawing samples from zero mean Gaussian with $\sigma = 1^\circ$. The histograms of average error for the methods is shown in Figure 4.5



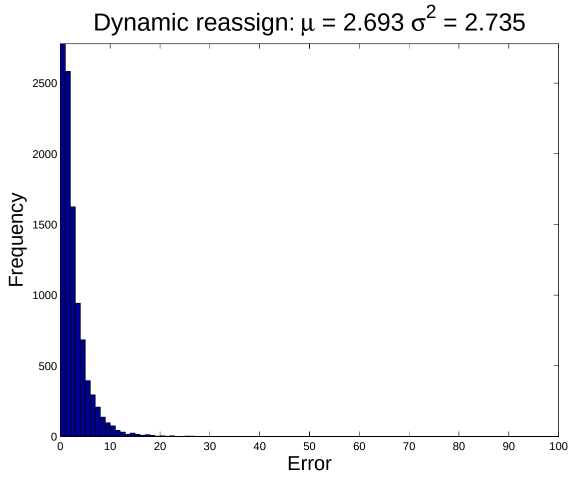
(a) Simulator



(b) No reassignment



(c) Greedy reassignment



(d) 2-approximation algorithm

Figure 4.5: **Case (a):** A tracking scenario with targets performing a random walk. **Cases (b)-(d):** Comparison of three algorithms. Histograms of the mean-squared error (MSE) for tracking without reassignment (a), with greedy reassignment (b) and reassignment according to the 2-approximation algorithm (c).

(b-d). The mean squared error ($\mu = 2.69$), and the variance of the error ($\sigma^2 = 2.73$), produced by the 2-approximation algorithm is significantly lower than both the greedy ($\mu = 4.48, \sigma^2 = 21.02$) and the static ($\mu = 15.30, \sigma^2 = 35.93$) approaches.

A PTAS for equidistant cameras

Our next result is a PTAS for equidistant cameras on the line. Specifically, we will show that given any $\epsilon > 0$, there is an algorithm to compute a $(1+O(\epsilon))$-approximate solution in $n^{O(1/\epsilon^4)}$ time (polynomial for any fixed positive ϵ). Without any loss of generality assume that the distance between two consecutive cameras is 1, hence the length of the line segment is $2n - 1$.

Note that we can view any solution as a matching between the cameras such that the weight on a matching edge (c_i, c_j) is equal to $\frac{Z_k}{|c_i c_j|}$ if and only if cameras c_i and c_j are assigned to the target k in the solution. We start with an overview of our PTAS. There are three main ingredients of our scheme:

- In any optimal matching, a camera in the left half (first n cameras) must be paired with a camera in the right half (last n cameras).

- Fix any optimal matching OPT. We distinguish between two types of camera pairings (edges) in OPT. We call an edge e *short* if its length is at most ϵn, and call it *long* otherwise. We show that all but a small fraction of the error in OPT is incident on the long edges.

- Finally, we show how to find a matching that allows us to approximate the length of each long edge in OPT to within a factor of $(1 - \epsilon)$ and each short edge to within a factor of $1/2$. Since much of the error is concentrated on long edges, this suffices to get an overall $(1 + O(\epsilon))$-approximation.

In what follows, we formally develop this ideas.

Lemma 30 *In an optimal matching the leftmost n cameras match with the rightmost n cameras.*

Proof: Assume c_i is matched to c_j, $i, j \leq n$ in an optimal solution, OPT. This implies that among the rightmost n cameras at least two of them match with each other, say c_k and c_l. But then, this matching can be improved by pairing c_i with c_k and c_j with c_l which contradicts the optimality of OPT. ∎

```
EquidistCamsOnALine($c_1, \ldots, c_{2n}$: camera positions,
      $z_1, \ldots, z_n$: target depths, $\epsilon$: error parameter)

$p \leftarrow \epsilon^2 n$; $q \leftarrow 1/\epsilon^2$
Partition leftmost cameras into $L_1, \ldots, L_q$ (Figure 4.6)
Partition rightmost cameras into $R_1, \ldots, R_q$
$\mathcal{M} \leftarrow$ set of all matchings of $\{L_i\}$ and $\{R_i\}$
for each matching $M$ in $\mathcal{M}$
   $\{p_i\} \leftarrow$ Camera pairing produced by ComputeCameraPairs($M$)
   $\{p'_i\} \leftarrow$ Sort $\{p_i\}$ so that $p'_i$ is the $i^{th}$ largest baseline
   $\{z'_i\} \leftarrow$ Sort $\{z_i\}$ so that $z'_i$ is the $i^{th}$ largest depth
   Error($M$) $\leftarrow$ error produced by assigning $p'_i$ to $z'_i$
return the best assignment

Subroutine ComputeCameraPairs(M: A partition matching)
$S \leftarrow \emptyset$
unmark all cameras
for $i = 1$ to $q$
   for $j = 1$ to $q$
      $m \leftarrow$ weight of matching edge type $(i, j)$
      for i = 1 to m
         $c_1 \leftarrow$ leftmost unmarked camera in $L_i$
         $c_2 \leftarrow$ leftmost unmarked camera in $R_j$
         $S \leftarrow S \cup \{(c_1, c_2)\}$
         mark $c_1$ and $c_2$
return $S$
```

Table 4.2: A (1+ϵ)-approximation algorithm for assigning equidistant cameras on a line to targets.

Let $p = \epsilon^2 n$ and $q = 1/\epsilon^2$. Partition the n points on the left into equal sized blocks $L_1, \ldots, L_q$ so that each block has p consecutive cameras. Similarly, we partition the points on the right into equal sized blocks $R_1, \ldots, R_q$. Consider a camera pairing (x, y) in OPT. We call it of type (i, j) if x is in L_i and y is in R_j.

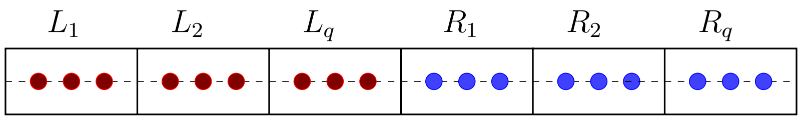


Figure 4.6: Partitioning the line segment: An edge that connects a camera in block L_i to a camera in block R_j is called an edge of type (i, j).

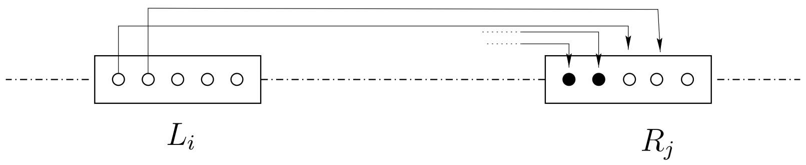


Figure 4.7: Assigning two edges of type (i, j) from L_i to R_j. Note that the first two cameras in R_j are already marked – i.e. have already been assigned to a camera in L_k for some $k < i$.

Recall that an edge is called short if its length is no more than ϵn.

Lemma 31 *The number of short edges is at most ϵn.*

Proof: The lemma follows from the fact that the short edges may involve at most $1/\epsilon$ left blocks connected to the $1/\epsilon$ right blocks. ∎

There are at most q^2 (i.e. constant, for a given ϵ) different types of matching edges in OPT. We can assume from here on that we know the number of edges of each type in OPT. This is easily done by enumerating over all possible $O(n^{q^2})$ vectors of matching edge types. Note that we are not enumerating over all possible matchings of cameras (the number all such matchings is n factorial). Instead, the enumeration is over all possible types of edges. This gives us a $q \times q$ matrix T where the entry $T(i, j)$ is equal to the number of edges of type (i, j).

Given the matrix T, we use the following scheme to match the cameras. Initially, all cameras are unmarked. Starting with the block L_1, we do the following for each block L_i. Suppose $T(i, 1) = x_1, \ldots, T(i, q) = x_q$. We first pair the x_1 leftmost cameras in L_i to x_1 leftmost unmarked cameras in R_1. We mark all paired cameras. Then we pair x_2 leftmost unmarked cameras in L_i to x_2 leftmost unmarked cameras in R_2 and so on. See Figure 4.7.

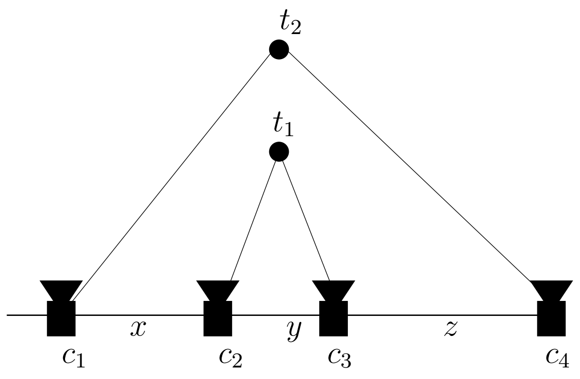


Figure 4.8: Figure for Lemma 33

Next, we show that the matching scheme described above produces camera pairs that are not much shorter than the camera pairs in OPT.

Lemma 32 *Given the matrix T that specifies number of edges of each type in OPT, the matching scheme above decreases the length of each short edge by a factor $1/2$ at most, and each long edge by a factor of $(1-\epsilon)$ at most.*

Proof: First observe that lengths of any two edges pairing a camera in some block L_i to another block R_j differ from each other by at most an additive $\epsilon^2 n$ term. This immediately gives us the error bound for long edges as well as for short edges whose length exceeds $\epsilon^2 n$. All that remains to consider are edges that match vertices in L_q to R_1. Observe that in our greedy matching scheme, we pair cameras in L_q only after all cameras in $L_1, ..., L_{q-1}$ have been paired. Thus we pair cameras in L_q to the rightmost possible cameras in R_1, doing at least as well as the pairing in OPT for this group. ∎

Lemma 33 *Let c_1, c_2, c_3 and c_4 be four cameras ordered from left to right, $x = |c_1c_2|$, $y = |c_2c_3|$, $z = |c_3c_4|$ with $z > x$. In addition, let t_1 and t_2 be two targets at distances z_1 and z_2 respectively (Figure 4.8). If (c_1, c_4, t_2) and (c_2, c_3, t_1) are triples in an optimal assignment then:*

$$\frac{Z_1}{y} \leq Z_2 \frac{(x+y)}{(x+y+z)(y+z)}$$

Proof: Consider the assignment obtained by crossing the pairs: (c_1, c_3, t_1) and (c_2, c_4, t_2) (see Figure 4.8). Due to optimality we have

$$\frac{Z_1}{y} + \frac{Z_2}{x+y+z} \leq \frac{Z_1}{x+y} + \frac{Z_2}{y+z}$$

and the lemma follows by simple algebraic manipulation. ∎

Finally, we show that all but a small fraction of the error weight in an optimal matching is incident on the long edges.

Lemma 34 *Let the weight (error) on an edge e for an assignment be $\frac{Z_i}{|e|}$ where Z_i is the depth of the target assigned to this edge and $|e|$ is the distance between the cameras connected by e. In any optimal assignment, the total weight on the short edges is at most an 64ϵ fraction of the overall weight.*

Proof: Let M and N be the leftmost and rightmost $\frac{3n}{4}$ cameras respectively. In an optimal matching, due to Lemma 30, the edges in M match with rightmost n edges and at least $\frac{n}{2}$ of them are in N. Let $B = \{b_1, \ldots, b_{\frac{n}{2}}\}$ be the set of any $\frac{n}{2}$ "big" edges that match cameras from M to cameras in N and $S = \{s_1, \ldots, s_k\}$ be the set of "small" edges. By Lemma 31, $k \leq \epsilon n$.

Partition B into $\frac{n}{2k} \geq \frac{1}{2\epsilon}$ groups B_i of size k arbitrarily.

We pick any group B_j and match the edges $b_i \in B_j$ to edges in S arbitrarily. Let Z_i^s and Z_i^b be the depths of targets assigned to s_i and b_i respectively. By Lemma 33 with $x + y \leq n + \epsilon n$, $x + y + z \geq \frac{n}{2}$ and $y + z \geq \frac{n}{4}$ we get:

$$\frac{Z_i^s}{s_i} \leq Z_i^b \frac{(n + \epsilon n)}{\frac{n}{2}\frac{n}{4}} = Z_i^b \frac{8(1+\epsilon)}{n} \leq \frac{16 Z_i^b}{n}$$

Let $w(S)$ be the total error in set S. Since a baseline can be of length at most $2n-1$, by summing up over the elements in S, we get $w(S) \leq 32 w(B_j)$.

Therefore we conclude:

$$w(B) \geq \frac{w(B_i)}{2\epsilon} \geq \frac{w(S)}{64\epsilon}$$

since the total weight is greater than $w(B)$, the lemma follows. ■

Theorem 35 *There exists a PTAS for assigning equidistant cameras on a line.*

Proof: The matching described ensures that short edges in OPT are reduced by at most a factor of 2 and long edges are within a factor of $(1+\epsilon)$. Using Lemma 34 above, by combining these matchings, we get an overall $1+O(\epsilon)$-approximation. ■

4.2.3 Range sensors on a circle

In this section, we consider range-sensors located on a circle $\mathcal{C}$ at equidistant intervals, tracking targets that are located inside $\mathcal{C}$. The error associated with a pair of range sensors (c_1, c_2) and a target t is approximated by $\frac{1}{\sin\theta}$ where $\theta = \angle c_1 t c_2$ (Figure 4.9). This is the Geometric Dilution of Precision (GDOP) for sensors that measure distances from the targets. A brief discussion on the derivation of this error measure can be found in Section 4.3.

In practice three range sensors are required for explicit target localization. However, target-tracking need not be an adversarial task. Consider a team of mobile robots negotiating a sensor network. Pairs of sensor measurements could be paired with heading information to enable localization. In this application, identifying optimal pairs would prove useful for providing optimal position estimates while minimizing network transmissions.

For simplicity, assume there are $4n$ sensors and $2n$ targets. Let S be the set of pairs generated by matching sensor i with sensor $i+n$ which is 90 degrees away clockwise from i. Assign the targets arbitrarily to pairs.

For two sensors c_1 and c_2, let x be a point inside C such that $\angle c_1 x c_2 = \frac{3\pi}{4}$ (see Figure 4.10). Let $Arc1(c_1, c_2)$ be the arc defined by c_1, c_2 and x and $Arc2(c_1, c_2)$ be

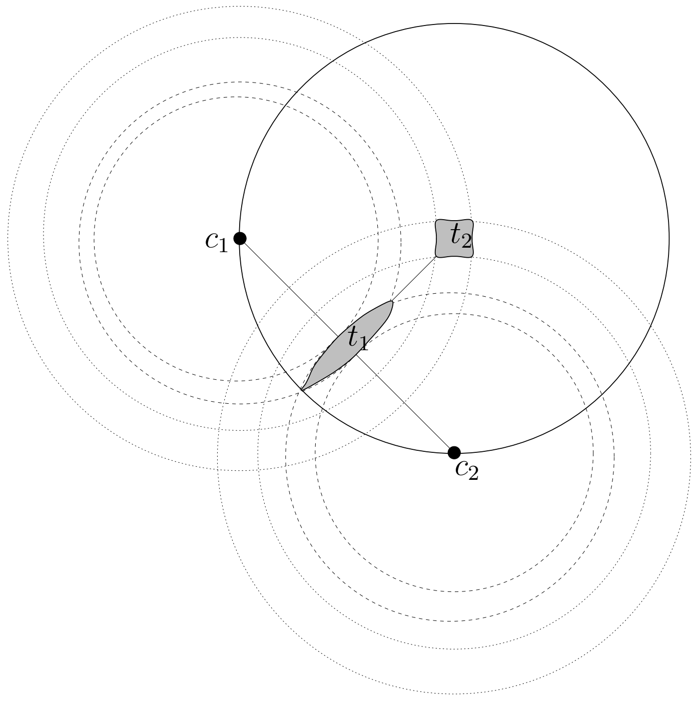


Figure 4.9: Geometric Dilution of Precision (GDOP): Each sensor c_1 and c_2 measures its distance to target with some precision. The error in position estimation is shown for two target locations t_1 and t_2. The error reduces as the angle $\theta = \angle c_1 t_i c_2$ gets closer to $\frac{\pi}{2}$.

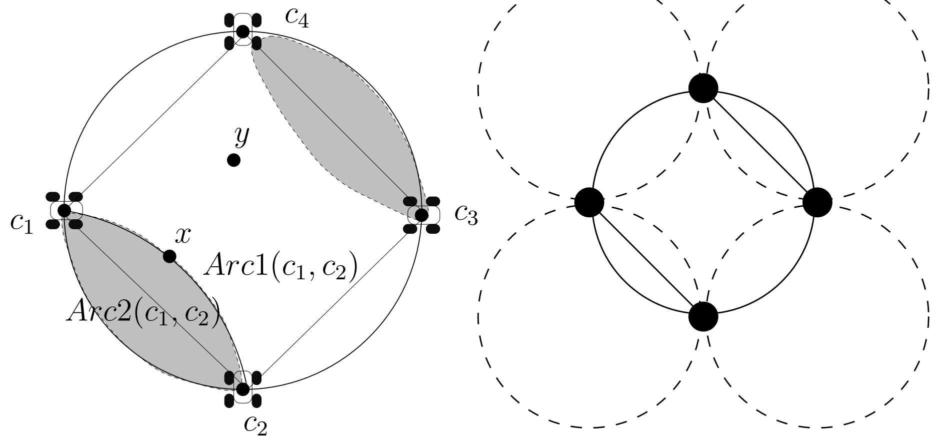


Figure 4.10: Sensors on circle: **Left:**The defective region for sensors c_1 and c_2 is the shaded area defined by arcs $Arc1(c_1, c_2)$ and $Arc2(c_1, c_2)$. **Right:** The defective regions are disjoint.

the arc axially symmetric with respect to the the chord c_1c_2. Note that $Arc2$ lies on $\mathcal{C}$.

Let A_{ij} be the region inside $Arc1(c_i, c_j)$ and $Arc2(c_i, c_j)$. Since any target outside the region A_{ij} is viewed by an angle less than $\frac{3\pi}{4}$ and greater than $\frac{\pi}{4}$ degrees from (c_i, c_j), we call the region A_{ij} a *defective* region for the pair (c_i, c_j). This angle is enough to guarantee a 1.42-approximation since $1/\sin(\frac{3\pi}{4}) < 1.42$ and the least error possible in this metric is 1. A target is *good*, if it is assigned to sensors c_1 and c_2 and located outside the defective region of (c_1, c_2). We summarize the properties of defective regions in the following propositions, which can be proven using basic geometric formulas.

Proposition 36 *Any target outside the defective region of sensors c_1 and c_2 is viewed by an angle less than $\frac{3\pi}{4}$ and greater than $\frac{\pi}{4}$ from c_1 and c_2.*

Proposition 37 *Let c_1, c_2, c_3 and c_4 be four sensors $\frac{\pi}{2}$ degrees apart. Defective regions of $(c_1, c_2), (c_2, c_3), (c_3, c_4)$ and (c_4, c_1) are disjoint (Figure 4.10 right).*

Having assigned the targets to sensors $\frac{\pi}{2}$ degrees apart we proceed as follows: We scan the pairs assigned to each target t_i. Suppose the current pair is (c_1, c_2).

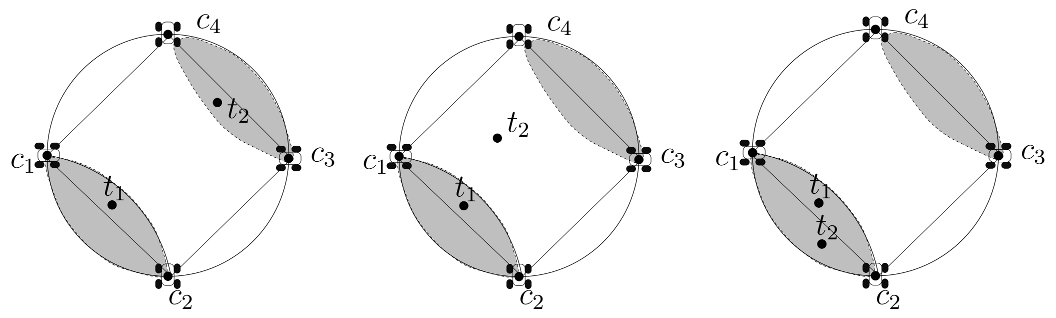


Figure 4.11: Three possible cases for the location of target t_2.

Now suppose that t_1 assigned to (c_1, c_2) is defective (i.e. in the defective region A_{12} of c_1 and c_2). Consider the pair (c_3, c_4), such that c_3 (resp. c_4) is the antipodal of c_1 (resp. c_3) and the target t_2 assigned to (c_3, c_4). Figure 4.11 illustrates the three possibilities based on the location of t_2.

- if $t_2 \in A_{34}$, we swap targets: the new assignment is (c_1, c_2, t_2) and (c_3, c_4, t_1).
- if t_2 is good and outside A_{12} again we swap targets: the new assignment is (c_1, c_2, t_2) and (c_3, c_4, t_1).
- if t_2 is good and inside A_{12}, we swap pairs: the new assignment is (c_1, c_4, t_1) and (c_2, c_3, t_2).

The reason we picked the angle as $\frac{3\pi}{4}$ is to make the defective regions disjoint: As the right illustration in Figure 4.10 shows, by construction the defective regions only intersect at the sensors. This makes each assignment have an error of 1.42 at most. In addition, once an assignment is modified we never return to it. Therefore this algorithm gives a 1.42-approximation for $1/\sin\theta$ error metric.

The main result of this section is summarized in the following theorem:

Theorem 38 *There exists an $O(n)$-time algorithm that simultaneously gives a 1.42-approximation to minimizing the sum of errors metric as well as minimizing the maximum error metric when the $4n$ sensors are equally spaced on a circle and the cost of assigning sensors i and j to target k is $\frac{1}{\sin\angle ikj}$.*

Simulation results

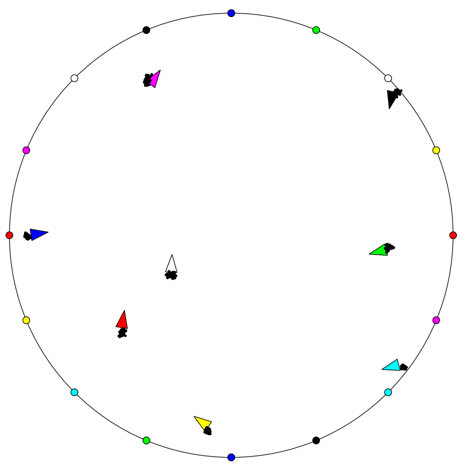
In this section, we demonstrate the utility of the algorithm for range sensors on circle for a cooperative localization task.

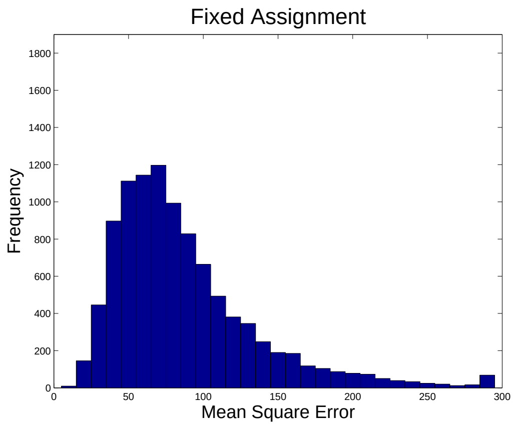
Target tracking need not be adversarial. In this simulation, n robots are operating within a sensor network defined by $2n$ range sensors on a circle. The robots rely on pairs of sensor measurements to estimate position. The resulting pair of position estimates are then merged with odometry information to eliminate pose ambiguity. Both the sensor and odometry measurements are corrupted with random Gaussian noise.

Again, three algorithms were modeled. Each initiated with a globally optimal assignment of sensor pairs to targets. In the first, this assignment was maintained throughout the simulation. The second employed a greedy reassignment strategy at each time step. Finally, Algorithm 3 followed the 1.42-approximation presented in Section 4.2.3. In this case, the sensor pairs assigned to every two targets were chosen from the initial four sensors assigned to the targets. Localization then proceeded with each robot transmitting a position estimate to its assigned sensor pair. The sensor pair in turn transmitted range measurements to the target. These measurements, the knowledge of sensor positions and odometry measurements allowed each robot to estimate its position at each timestep. The procedure then iterated.

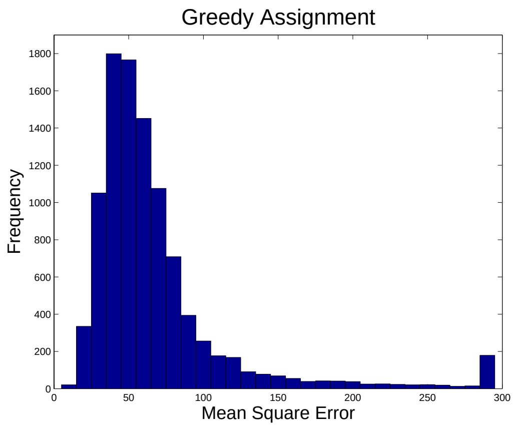
Localization performance for the three algorithms is reflected in Figure 4.12. In this example, 8 robots were tracked by 16 sensors over 10,000 time steps. The robots localized while following pseudo-random trajectories through the network. Results indicate significant improvements in localization performance can be achieved by intelligently assigning targets to sensors.

Both our 1.42 approximation (MSE=68.4, $\sigma^2 = 29.3$) as well as greedy reassignment (MSE=70.9, $\sigma^2 = 66.8$) outperformed static assignment (MSE=87.5, $\sigma^2 = 49.7$). It is interesting to note that while greedy reassignment and our 1.42 approximation have nearly identical performance with respect to mean square error,

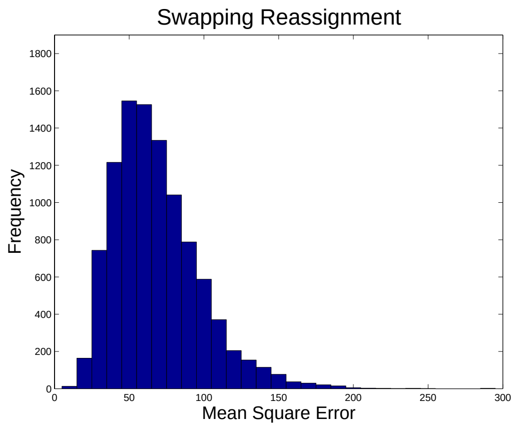

(a) Simulator



(b) No reassignment



(c) Greedy reassignment



(d) 1.42-approximation

Figure 4.12: **Case (a):** A tracking scenario showing robot and sensor positions. **Cases (b)-(d):** Comparison of three algorithms. Histograms of the mean-squared error (MSE) for tracking without reassignment (a), with greedy reassignment (b) and reassignment according to the 1.42-approximation algorithm (c). The latter reduces both MSE and error variance over the alternative approaches.

the performance guarantee of our approach results in a significantly lower variance. It also scales far better computationally ($O(n)$ vs. $O(n^3)$) and requires far fewer reassignments (2,206 vs. 61,241 out of 80,000 possible assignments).

4.2.4 Discussion: Universal Placement

Note that the analysis above shows that the equidistant placement for $\frac{1}{\sin\theta}$ metric is *universal*: When the cameras are placed equidistantly, no matter where the targets are located, our algorithm guarantees a 1.42-approximation to the optimal matchings generated by *any* placement of sensors on circle.

Similarly, a universal placement for cameras on a line segment $[x, y]$ for the Z/b metric would be to put half of the cameras on x and the other half on y, which guarantees an optimal assignment for this metric.

4.2.5 Arbitrary sensor assignment

The inapproximability of FOA for general sensor assignment lead us to repose it as its "dual" maximization problem. To do this, we define the notion of a *valid track*. An assignment (c_i, c_j, t_k) is considered a valid track if $Err(c_i, c_j, t_k) \leq \delta_0$, where δ_0 represents an acceptable error threshold predefined by the user. The problem then becomes: Given a set of sensors C with $c_i \in C$, a set of targets T with $t \in T$, and an error threshold δ_0, construct a set of disjoint assignments A, where $(c_i, c_j, t_k) \in A$ iff $Err(c_i, c_j, t_k) \leq \delta_0$, such that $|A|$ is maximized.

In addition to arbitrary error metrics, this formulation allows us to deal with occlusions in the scene: If target t_j is not visible from camera c_i, we delete all triples that contain both c_i and t_j from the set of valid tracks.

When the error metric is arbitrary, this problem is equivalent to *Maximum 3-Set Packing*[2], which is known to be NP-hard [34]. It is also known that a simple greedy

[2]Given a 3-set system (S, C) – a set S and a collection C of size 3 subsets of S, find a maximum cardinality collection of disjoint sets in C.

```
GreedyAssign(A = $\{(c_i, c_j, t_k)\}$: valid assignments)
$S \leftarrow \emptyset$
for all assignments $a \in A$
   if $a$ does not conflict with any assignment in $S$
      $S \leftarrow S \cup \{a\}$
return $S$
```

Table 4.3: Greedy algorithm for selecting valid assignments.

heuristic (Table 4.3) which adds any set to the solution as long as it does not conflict is within a factor of 3 of the optimal solution. A *2-locally-optimal* solution is defined as a maximal solution that cannot be improved further by removing any item from the current solution, and attempting to insert two non-conflicting items (Table 4.4). It has been shown that any 2-locally optimal solution provides a $\frac{5}{3}$ approximation [41, 112]. If $N = O(n^3)$ is the number of valid tracks, the greedy algorithm can be implemented in $O(N)$ time, whereas finding a 2-locally optimal solution takes $O(N^2)$ time. The details of the latter implementation can be found in [41].

```
2-LocalAssign(A = $\{(c_i, c_j, c_k)\}$: valid assignments)
$S \leftarrow GreedyAssign(A)$
improved $\leftarrow$ true
while improved
   improved $\leftarrow$ false
   for all assignments $a \in S$
      for all assignments $b, c \in A \setminus S$
         if $S \setminus \{a\} \cup \{b, c\}$ is a valid solution
            $S \leftarrow (S \setminus \{a\}) \cup \{b, c\}$
            improved $\leftarrow$ true
return $S$
```

Table 4.4: A 2-local algorithm for selecting valid assignments. A solution is called a *valid solution* if no two of assignments in the solution contain the same target or camera.

One might suspect that a 2-locally optimal solution would yield an approximation factor better than $\frac{5}{3}$ for restricted error metrics. However, this is not the case, even for equidistant cameras on the line: Consider the example in Figure 4.13 with cameras on a line, Z/b as our error metric and an error threshold $\delta_0 = 1$. There are five targets, $t_1, \ldots, t_5$ and $z_1 = 9, z_2 = 7,\ z_3 = 5, z_4 = 3$ and $z_5 = 1$. Ten cameras c_1 to c_{10} are located at $x = 1, \ldots, 10$. Optimum packing is five targets with $(t_1, 1, 10), (t_2, 2, 9)$, $(t_3, 3, 8), (t_4, 4, 7)$ and $(t_5, 5, 6)$, represented by the nodes of the conflict graph shown on the left in Figure 4.13. Suppose our solution is $(t4, 3, 6)$, $(t5, 8, 9)$ and $(t3, 5, 10)$. Note that it is not possible to remove a triple from this solution and insert two, therefore it is 2-locally optimal. But this implies a $\frac{5}{3}$ approximation, which shows that the analysis is tight. It is possible to generalize this example to $5k$ targets, just by replicating k instances of the same example and putting them on the top of each other. Thus, the $\frac{5}{3}$ lower bound is tight even for equi-spaced cameras on a line.

In the next section, we investigate the utility of the greedy and 2-local algorithms through simulations.

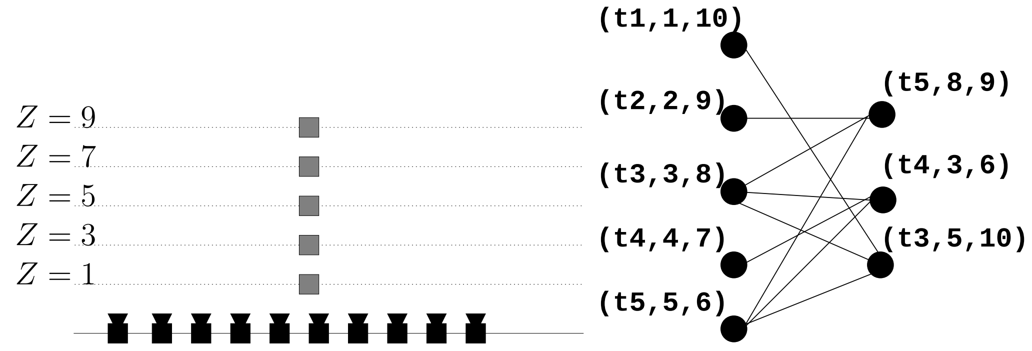


Figure 4.13: **Left:** An instance of FOA. **Right:** The conflict graph for the optimal solution and a 2-local solution: The nodes on the left (resp. right) represent the optimal (resp. 2-local) solution. There is an edge between two nodes if the assignments conflict. In this case, a 2-local solution gives a $\frac{5}{3}$ approximation, showing that the analysis is indeed tight.

Simulation results

In this last simulation, we examined the arbitrary sensor placement problem as outlined in Section 4.2.5. For this example, 20 cameras were distributed roughly uniformly on the plane and charged with tracking 10 targets. Here, the objective was to maximize the number of valid tracks, in contrast to the error minimization objective of previous simulations. Targets followed random trajectories, and were tracked in simulation using particle filters. The respective particle sets were employed to generate a numerical error metric for the targets as discussed in [96].

Two algorithms were investigated for this maximization approach. The first employed a greedy assignment strategy, and the second a 2-locally optimal approach as discussed in Section 4.2.5. The latter took the greedy solution as input, and as a consequence could only improve on its performance. Reassignment was made for both algorithms at each time-step. Several trials were conducted corresponding to sparse and dense solution sets. Data from a representative trial can be found in Figure 4.14.

In each trial, the 2-local solution improved over greedy by 5-15%. As expected, the larger improvements corresponded to dense solution sets - i.e. when there were more opportunities for finding local improvements. These results are by no means encompassing, and provide only insights into expected performance which is a function of too many variables to address here. However, they imply that unless the guarantee of improved performance is critical, the greater computational complexity of 2-local may not be warranted by the expected performance improvement over greedy for real-time applications.

4.3 Error Models

In this section, we present a brief discussion on the error functions used in this chapter. In order to derive analytical functions for error estimation, we propagate the

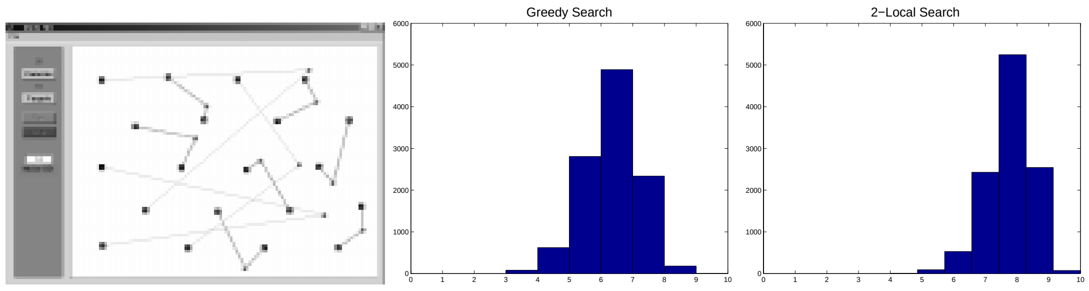


Figure 4.14: **Left:** Simulator snapshot for 2-local assignment trial. **Center, Right:** The number of valid tracks recovered for greedy and 2-local search strategies. In this example, 2-local improved over greedy by on average 15%.

error in the observables to position estimation by using a first order approximation of the function used for estimation. For example, in stereo reconstruction, the observables are the pixel coordinates of the target in the images. The value we would like to estimate is the depth (i.e. distance from the camera) of the target.

Suppose we have a function $y = f(x)$ that is used for estimating the position of the targets, y, given the observable x. Here, both x and y can be vectors. If the error is unbiased, a small error ϵ in the measurement x propagates to our estimation as $y' = f(x + \epsilon) \approx f(x) + J\epsilon$ where J is the Jacobian of f (See [33], pp. 42–43). A common approach is to use the determinant of the covariance of the transformed variable y as an analytical error function:

$$Cov_y = J \cdot Cov_x \cdot J^T \tag{4.3}$$

Omnidirectional Stereo cameras

Given a point $X = [x\ y\ z]$ in 3D, let (u_1, v_1) and (u_2, v_2) be the coordinates (in pixels) of the projection of X onto the image plane of two cameras. If the two images are rectified (parallel), we have $v_1 = v_2$. It is commonly assumed that the observation in v is error-free. Hence, the depth of the world point, z, can be estimated by

$$z = \frac{bf}{|u_1 - u_2|} \tag{4.4}$$

where f is the focal length of the cameras, b is the baseline and the quantity $d = |u_1 - u_2|$ is called the disparity. By defining $\delta = d/f$ and taking the derivatives with respect to δ, we obtain the standard deviation of the depth as:

$$\sigma_z = \frac{Z^2}{b}\sigma_\delta \tag{4.5}$$

Hence, the propagation of error in disparity measurements to the depth estimation is proportional to the quantity $\frac{z^2}{b}$.

Range sensors

Let $\vec{r_1} = [x_1, y_1]$ and $\vec{r_2} = [x_2, y_2]$ be the coordinates of two sensors that measure their distances, r_1 and r_2, to the target located at $\vec{r} = [x, y]$. Hence, we have the measurements:

$$r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}, \qquad i = 1, 2 \tag{4.6}$$

By taking derivatives with respect to x and y, the determinant of the jacobian can be shown to be:

$$|J| = \frac{|\vec{r_1}||\vec{r_2}|}{\vec{r_1} \times \vec{r_2}} = \sin(\theta) \tag{4.7}$$

where θ is the angle between the two vectors $\vec{r} - \vec{r_1}$ and $\vec{r} - \vec{r_2}$.

General stereo setups

In this section we investigate the error in depth produced during the triangulation step of stereo reconstruction. Consider the stereo setup in Figure 4.15 where two cameras (left and right) are verging with angles α_l and α_r. The origin of the world coincides with the image center of the right camera. The positive X axis of the world is the ray $\vec{O_r O_l}$. We assume that the vertical disparity is exactly zero and given by the calibration parameters used for the image rectification. This means that we constrain ourselves to the "flatland" ($X - Z$ plane) and consider only variations in vergence and baseline. Given a world point P, our measurement of depth can be

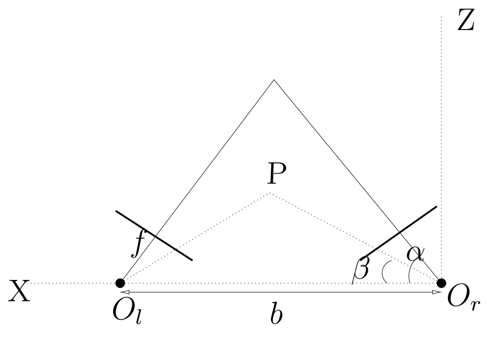


Figure 4.15: Stereo set-up for depth triangulation given matches

expressed as a function of baseline b, vergence angles α_l, α_r, focal length f and the world coordinates (X,Z):

$$x_r = \frac{f\ (X \tan\alpha_r - Z)}{X + Z\tan\alpha_r}$$
$$x_l = \frac{f\ (Z + (b-x)\tan\alpha_l)}{b - X - Z\tan\alpha_l}$$

The actual triangulation algorithm uses the projection matrices from calibration but since the vertical disparity is assumed to be zero we can solve the system above instead.

If we replace x_l with $f\tan\beta_l$ and x_r with $f\tan\beta_r$ the triangulated point reads

$$\begin{aligned} X &= \frac{b(\sin(\beta_r - \beta_l + \alpha_l - \alpha_r) - \sin(\beta_r + \beta_l - \alpha_l - \alpha_r)}{\sin(-\beta_l + \beta_r - \alpha_r + \alpha_l)} \\ Z &= \frac{b(\cos(\beta_r - \beta_l + \alpha_l - \alpha_r) - \cos(\beta_r + \beta_l - \alpha_l - \alpha_r)}{\sin(-\beta_l + \beta_r - \alpha_r + \alpha_l)} \end{aligned}$$

In dense area based approaches we can assume that x_l is exact and that matching errors reflect only in the disparity $d = x_r - x_l$. The variance of the pixel disparity can be approximately propagated into the depth variance as follows:

$$\sigma_Z^2 = (\frac{\partial Z}{\partial d}\sigma_d)^2. \tag{4.8}$$

According to Cramer-Rao if our estimator is unbiased the above expression gives the lower bound on the Z-variance. In the remainder of this section we are going to vary the ground truth depth, the vergence angles, and the baseline. For the non-varying parameters each time we assume values equal to the calibration parameters of our set-up. The focal length is 6 mm and consequently the pixel size is 15μm. The average depth point for a typical scene is Z=125cm. We consider only the cases where the cameras are verging symmetrically($\alpha_r = \alpha$, $\alpha_l = 180^o - \alpha$). All angles are shown in degrees.

We compute $(\frac{\partial Z}{\partial d})^2$ and plot the algebraically predicted **relative** depth error σ_Z^2/Z^2 which we call the theoretical error. To verify our formulae we also run a synthetic experiment 1000 times adding Gaussian noise $\mathcal{N}(0, \sigma_d)$ and taking the ensemble average for the relative error for the same depth point $E[\|\Delta Z\|^2]/Z^2$ which we call synthetic error.

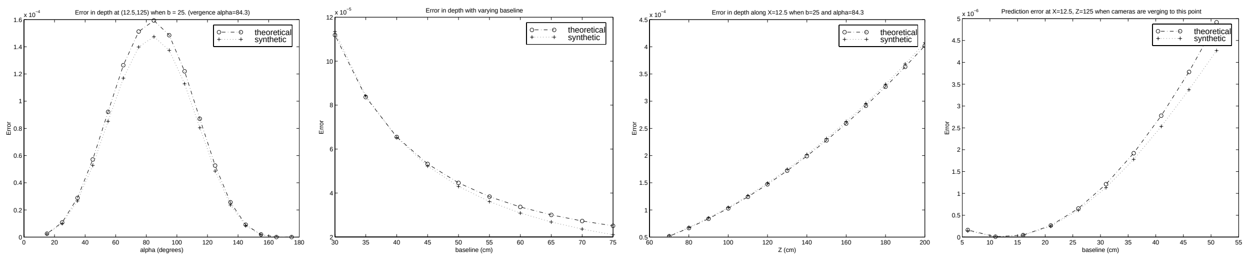


Figure 4.16: **Left:** Error at X=12.5cm and Z=125cm as a function of vergence angle α ($b = 25, \sigma_d = 14.8\mu, f = 0.6cm$). Cameras verge at this point when $\alpha = 84.2894^o$. **Middle-left:** Error at X=12.5, Z=125 when cameras are verging at this point with changing baseline.$\alpha_l = 180 - tan^{-1}[\frac{Z}{b-X}], \alpha_r = tan^{-1}[\frac{Z}{X}], \sigma_d = 14.8\mu, f = 0.6cm, b \in [30, 100]$. **Middle-right:** Prediction Error for the point X=12.5cm and Z=125cm. Left camera is centered at X=25cm, Z=0cm. The predicted view is at X=baseline, Z=0cm. All cameras verge at the point X=12.5 Z=125cm. $\sigma_d = 14.8\mu, f = 0.6cm$. **Right:** Prediction error along the line X=12.5cm as a function of Z. $\alpha = 84.289^o, b = 25, \sigma_d = 14.8\mu, f = 0.6cm, Z \in [60, 200]$. Cameras verge at Z=125. Each point in the synthetic plot represents 1000 trials.

In the first experiment we study the error in reconstructing a fixed world point when the fixation point of the cameras changes. This is equivalent to the question: Given a person fixed at a position, should the cameras verge in front or behind the person? Regarding correspondence we know that the cameras should fixate on the person so that the disparity range is limited. However, regarding the triangulation step we observe both theoretically and synthetically that the error in depth is maximized when the cameras verge to the point of interest (Figure 4.16-left). Basu [90] showed that we could obtain the opposite effect if the resolution were not constant over the image but foveating.

In the second experiment, we keep the cameras fixating at the same point but increase the base-line. As intuitively expected (Fig. 4.16-middle-left), the relative depth error at the fixated point decreases with increasing baseline.

In the third experiment, we assume that the stereo set-up is fixed and we just let the considered point in the scene move along the symmetry axis $X = \frac{b}{2}$. As expected (Fig. 4.16-middle-right) the relative depth error increases as we move away from the cameras.

For the algebraic prediction of the second error metric, we assume that the novel view is in the plane XZ with center at the point $(X = b_v, Z = c_v)$ and vergence angle α_v. We compute the projection x_v of the ground-truth point (X, Z) on the new image plane and the projection $\hat{x}_v$ of the reconstructed point $(\hat{X}, \hat{Z})$. We then take the derivative of the difference $x_v - \hat{x}_v$ with respect to the disparity. The variance of this error metric is then

$$E[|x_v - \hat{x}_v|^2] = (\frac{\partial(x_v - \hat{x}_v)}{\partial d}\sigma_d)^2$$

To study the discrepancy in the novel view at a point on the symmetry axis (Fig. 4.16-right), we let the virtual camera move along the baseline and we observe that the minimum of the error metric for the novel views occurs when the virtual camera lies in the middle of the baseline.

4.4 Concluding Remarks

In this chapter, we have introduced the focus of attention problem and studied the algorithmic aspects of managing the attention of a group of distributed sensors. We observed that for a general cost metric, the problem is NP-hard and not well approximable. However, for constrained geometric cases we were able to exploit relations between the sensor geometry and corresponding error metrics. From this, we obtained: a 2-approximation for stereo cameras constrained to the same baseline, a PTAS solution for the same geometry when the cameras are spaced equidistantly, and a 1.42-approximation for $4n$-range sensors equi-spaced on the circle. For arbitrary sensor placement, we reposed the problem in a maximization vein. Using results from maximum set-packing, we obtained a $\frac{5}{3}$-approximate solution. This was implemented in simulation, and its performance contrasted against a greedy approach.

Adjusting the focus of attention of the sensors reduces the overall tracking error significantly. This is true when applied to both direct and filtered estimation problems, as demonstrated in sections 4.2.2 and 4.2.3, respectively. The 2-approximation for stereo cameras and the 1.42-approximation for range sensors have several desirable attributes. Their matchings have twofold approximation guarantees; the sums of errors are bounded, as are the individual target errors. Additionally, they are readily implemented, and are inexpensive both computationally ($O(n \log n)$ and $O(n)$, respectively) and in terms of network communications ($O(n)$). In simulation, both showed significant improvements in performance over greedy/static assignment strategies. The constraints to geometry are restrictive but still useful, and we are currently working to extend these to additional configurations.

Empirical results for arbitrary sensor placement simulations indicate on average a 5-15% improvement for the $\frac{5}{3}$-approximate solution over a greedy approach. However, the former is more expensive computationally (quadratic vs. linear in the number of valid tracks). As a consequence, a greedy strategy may be preferred for real-time applications.

While we feel these results provide a solid theoretical footing for the FOA problem, there is still significant room for future work. Using our error metrics, sensor configurations are limited to restricted geometries. This is addressed to a point by our $\frac{5}{3}$-approximate solution for arbitrary sensor placement. However, it relies upon an alternate objective function (maximizing low error tracks rather than minimizing the total error). Strategies for minimizing the tracking error under general sensor placement is the topic of ongoing work.

In the general FOA formulation, we have not addressed the subject of occlusions. We have assumed that the individual targets are neither occluded by one another, nor by clutter in the environment. Such an assumption is unrealistic. However, the estimates from each sensor/target triple need not be used as direct estimates for target positions. Rather, they can serve as inputs to traditional Bayesian filters (i.e. Kalman or particle) which yield estimates of the target pose. The latter has shown robustness for tracking features in a cluttered image [51]. For the arbitrary sensor placement problem, general visibility constraints (range, occlusions, *etc.*) can also be accommodated by merely eliminating the corresponding triples from the feasible set of assignments.

Finally, we have only considered the case where each sensor is constrained to track a single target during each time step. Such an approach is relevant for pan-tilt-zoom cameras, or when the computational requirements of the tracking algorithm support only a single target assignment. We have not addressed the case where multiple targets are visible within a single camera's field of view. We hope to answer such questions in our future research.

Chapter 5

Pursuit-Evasion Games in Complex Environments

Many sensing require finding a moving object. As described in Section 1.2, this is a challenging task for mobile robots who can not see all the points in the environment at all times. As an example, consider an application where a mobile robot is used to find intruders in a building. In this application, the intruders will try to avoid being seen by our robot. Hence, our robot must move in such a way that no matter how the intruders move, our robot will eventually see them. As another application, consider a search-and-rescue mission where we would like to find a lost wanderer in a desert. Such a person may be quite unpredictable. Therefore in this scenario as well we need a motion plan to eventually find him – no matter how he moves.

Such problems are best formulated as pursuit-evasion games which are among the fundamental problems studied by robotics researchers. In a pursuit-evasion game, one or more pursuers try to capture an evader who, in turn, tries to avoid capture indefinitely. A typical example is the homicidal chauffeur game where a driver wants to collide with a pedestrian and the goal is to determine conditions under which he can (not) do so. Among the numerous applications of this game are collision avoidance and air traffic control.

There are many different versions of pursuit evasion games based on the following characteristics:

- *Environment where the game is played:* Examples include polygonal environments, airspace, a grid, or a graph.
- *Information available to the the players:* Do they know each others' positions all the time? Does the pursuer know evader's strategy? Is there noise in the observations?
- *Models of players' motion:* Is player motion physically modeled using dynamical systems with bounds on inputs or do they move discretely on graphs, one edge at a time ?
- *Definition of capture:* In some games, the pursuer captures the evader if the distance between them is less than a threshold. In other games, the pursuers must intercept, see or surround the evader in order to capture it.

The main ingredient of a pursuit-evasion game is the presence of an adversarial evader who actively avoids capture. Due the this aspect, the solutions to pursuit-evasion problems are game theoretic and distinguish themselves from their target finding counterparts (e.g. [91, 106]) where the evader's motion is independent from the pursuer's.

Recently, there has been increasing interest in capturing intelligent evaders with certain sensing capabilities who are contaminating a complex environment. Aside from its obvious applications in search-and-rescue and surveillance, this problem provides a natural test-bed for many mobile robot algorithms.

As stated in [38], complex environments impose geometric free space constraints and pursuit-evasion problems in these environment inherit the complexity of motion planning. An additional source of complexity is visibility[1]. If the players have

[1]By *visibility of a point x*, we mean the set of points visible from x. Since visibility is a symmetric relation in this work, this also is the set of points that can see x.

line of sight visibility, then they can exploit occlusions in the environment. Therefore, geometric complexity also imposes restrictions on the information available to the players. Addressing these issues require an understanding of the combinatorial aspects of the game.

There are two main approaches for modeling complex environments. A common approach is to model the environment with a graph whose nodes correspond to physical locations such as rooms. Alternatively the environment can be represented by a polygon (possibly with holes) which usually corresponds to a 2D layout of the actual environment. In the next sections, we present an overview of results from the literature for both kinds of representations. We have included the results presented in chapters 6 and 7 in the overview for completeness.

5.1 Pursuit-Evasion Problems: The Global Picture

In this section we present a taxonomy of pursuit-evasion games based on the evader's speed and visibility. In the sequel, we will assume that the pursuer has global visibility (i.e. knows the position of the evader at all times). This is well-justified when the game is played in a finite environment (finite graphs or bounded polygons). In such environments, randomization gives the pursuer the power of global visibility – the power of knowing the evader's location at all times:

Let P_G be a pursuer with global visibility and suppose there exists a winning strategy for P_G. This means that, no matter what the evader does, the game ends in finite time, say T. Let S be the set of all possible evader strategies that last at most T steps. In case of graphs, S may be the set of all paths of length at most T. In case of polygons, the evader strategy must be discretized before enumeration. For example, in visibility-based games, the evader's location can be mapped to an appropriate vertex of the polygon.

The pursuer with local visibility, say P_L, picks an evader strategy uniformly at random from S and simulates the strategy of P_G against the chosen evader strategy. If he can correctly guess the (future) moves of the evader, he wins the game. Since $|S|$ is finite, the pursuer will eventuall make a correct guess in finite time with high probability – simply by repeating the trials. Of course, this strategy may cause an exponential increase in the time to capture the evader. Therefore, in most of the existing literature, this approach is not taken for designing pursuer strategies. However, since it simplifies the taxonomy, we have found this assumption useful.

Further, this randomized guessing strategy does not hold if the environment is unbounded. For example, consider the following pursuit-evasion problem [72]: The input is a pair $(P, d)$where P is the polygonal room the game is played in and d is a *door*, a point marked on the boundary of P. The goal is to devise a strategy for the pursuer to eventually see the evader, in such a way that the evader can not escape through the door. In this game, the pursuer can not afford a wrong guess and therefore the technique just described does not apply.

5.2 Pursuit-Evasion on Graphs

In this section, we present an overview of pursuit-evasion games on graphs. We assume that the pursuer has global visibility (as justified in the previous section). We also assume that the speeds of the players are normalized so that the pursuer has unit speed: It can move from vertex u to v if (u, v) is an edge.

Hence, we can characterize pursuit-evasion games on graphs based on two parameters: the evader's visibility and speed.

Evader's visibility: Global visibility means that the evader knows the location of the pursuer(s) at all times. By local visibility, we refer to the case where the evader can see only the neighborhood of its current location. Local visibility can be generalized to i-visibility meaning that the evader can see the vertices which are at most distance i from its current location.

Evader's speed: The two well studied cases are infinite speed and unit speed. The only way to capture an infinitely fast evader is to surround it, otherwise in no time it can escape to any vertex v if there is a path from its current location to v that avoids the pursuers. Unit speed means that the evader's speed is the same as the pursuers'. The players can move to a vertex in the next round only if that vertex is adjacent to their current location.

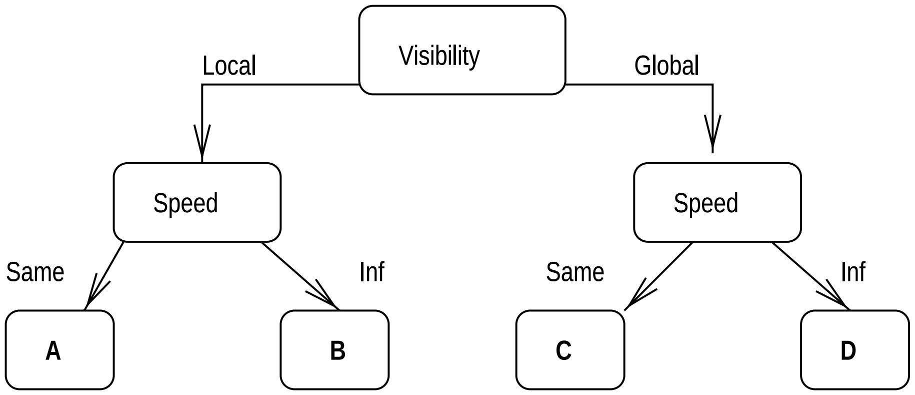


Figure 5.1: Pursuit-Evasion on graphs.

Combinations of these aspects is shown in Figure 5.1. Next, we describe the state of the art for each case.

Box A – Zero Visibility/Same Speed: It has been known for a while [6] that in the 0-visibility case, a single pursuer can capture the evader: A simple random walk suffices to capture in time $O(n^2m)$ where n and m denote the number of vertices and edges respectively. Recently Adler et al. presented a strategy that enables a single pursuer to capture an evader with 0-visibility in $O(n \log n)$ time [2]. They also showed that this bound is tight by presenting a matching evader strategy. In fact, the same results apply even if the rabbit has infinite speed. Hence this problem is closed.

Local visibility (i.e. 1-visibility) is studied in this thesis (Chapter 6). It is shown that two hunters suffice on any graph and an algorithmic characterization of graphs on which a single hunter suffices is presented. However, the single-pursuer strategy on these graphs may take possibly exponential time and the main open problem is whether there exists pursuer strategies that capture the evader in expected polynomial time.

If the evader has 2-visibility, we show that there are graphs that require $\tilde{O}(\sqrt{n})$ pursuers. However, beyond 1-visibility is not well-studied.

Box B – Zero Visibility/Faster Evader: If the evader has no visibility, then its speed is irrelevant and the $O(n \log n)$ capture time presented in [2] applies. The intermediate cases (e.g. the evader is k times faster where k is a constant) are not well-studied – perhaps due to the lack of a natural model for speed differences on graphs. To the best of our knowledge, there is no previous work on pursuit-evasion games on graphs with this model.

Box C – Global Visibility/Same Speed: This game is known as the *the cops and robbers game* [82, 4, 13]. Structural and algorithmic characterization of graphs on which a single pursuer suffices is known (these graphs are called dismantlable graphs).

Determining the cop-number of a graph is an open problem. Special cases have been studied in the literature. It is known that 3 cops suffice on planar graphs, however this is not tight [4]. Clearly 2 cops are required for a cycle, but it is not known whether there are planar graphs that require three cops.

Box D – Global Visibility/Faster Evader: The variant where the evader can be arbitrarily faster than the pursuer is well-studied [87, 77]. Note that in this case, the pursuers must surround the evader to be able to capture it. It is known that determining the number of pursuers (known as the search number) is NP-Complete [77, 67]. Similar to Box B, there is no previous work on pursuit-evasion games on graphs where the evader is only k times faster where k is a constant.

		Visibility	
		Local	Global
Speed	Same	2 [Ch 6]	n [4]
	∞	(?)	n [77]

Table 5.1: Number of pursuers for pursuit on graphs: n means that the number can be arbitrarily large.

5.3 Pursuit-Evasion in Geometric Environments

In this section, we consider games which take place in planar, bounded (typically polygonal) environments. The main parameter of the environment is its connectivity. A planar region is *simply-connected* if any simple closed curve inside the region can be shrunk to a point. It is *multiply-connected* otherwise.

We will consider players with different visibility constraints. In the sequel, *local visibility* refers to line of sight visibility: two points p and q see each other only if the line segment pq lies entirely inside the polygon.

For the evader's speed we will consider only the two extremes. Either the pursuers will be as fast as the evader or the evader will be infinitely fast. Existing results in the literature for intermediate cases are restricted to very simple environments (e.g. inside a circle).

5.3.1 Capturing the evader

Let us start with games where the pursuer tries to capture the evader. Here, capture means to be at the same point at the same time. First, let's consider the case where the players have the same speed . Different combinations based on environment connectivity and evader's visibility are shown in Boxes A-D of Figure 5.2.

Simply-connected Environments: In Chapter 7, we first present a strategy for two pursuers to capture an evader with global visibility in a simply-connected environment. Afterwards, we show the strategy of the two pursuers can be modified so that a single pursuer can also capture the evader (Figure 5.2, Box B). This clearly

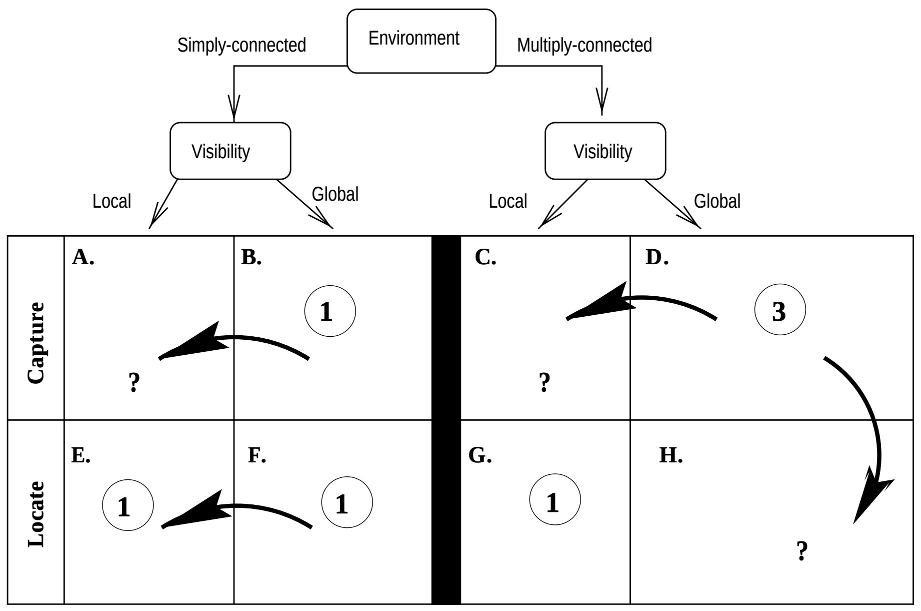


Figure 5.2: Pursuit-Evasion in bounded environments. Visibility refers to the evader's visibility. The circled numbers indicate the number of pursuers known to be sufficient for the corresponding case. An arrow shows a result that carries over to another case. A question mark means that there is no known improvement over the result that carried over.

means that a single pursuer is also enough for capturing an evader with local visibility (Figure 5.2, Box A). However, the capture time for the global visibility case can be exponential in the number of vertices of the polygon. It is not clear if the capture time can be improved for the local visibility case.

Multiply-connected environments: If the players have global visibility and the same speed, it is known that [4] three pursuers can capture the evader in a multiply-connected environment (Figure 5.2, Box D). However, it is not known whether two pursuers suffice or three is necessary. Further, if the pursuers have only local visibility, the capture time may be exponential in the number of vertices of the polygon. It is not known whether this is necessary or if the capture time can be improved, at least for the local visibility case (Figure 5.2, Box C).

5.3.2 Locating the evader

Next, we consider the game where the pursuers try to locate (or see) the evader.

Simply-connected Environments: In Chapter 7, we present a randomized strategy that guarantees with high probability that the evader will be located in any simply connected environment. The strategy works even if the evader has global visibility (Figure 5.2, Boxes E and F). There is not much room for improving the capture time in both cases.

Multiply-connected environments: If the evader has only local-visibility (Figure 5.2, Box G), Phase One of the algorithm described in Section 6.2 can be modified so that a single pursuer can locate the evader in any environment. There is no room for improving the capture time. If the evader has global visibility but the pursuers are as fast as the evader, clearly the result in Box D implies that three pursuers are enough (they can even capture it). However, it is not known whether this can be improved. If the evader is faster, determining the exact number of pursuers is NP-hard for multiply-connected environments [38].

Finally, we summarize known results for different visibility, speed and connectivity combinations. These results are summarized in Tables 5.2 and 5.3 for locating and capturing the evader respectively. For locating an evader with infinite speed, determining the exact number of pursuers is NP-hard for multiply-connected environments [38]. However, at most $\Theta(\sqrt{h} + \log n)$ pursuers are required for any polygonal environment with h holes and n vertices [38]. This number can be improved to $\Theta(\sqrt{h} + 1)$ using the result in Section 7.2.

		Connectedness	
		Simple	**Multiple**
(Visibility, Speed)	**(Global, ∞)**	1 [Sec 7.2]	$\Theta(\sqrt{h} + \log n)$ [38]
	(Global, 1)	1 [Sec 7.2]	3 [4]
	(Local, ∞)	1 [Sec 6.2]	1 Sec 6.2

Table 5.2: Locating the Evader in a polygon: $\Theta(\sqrt{h} + \log n)$ result can be improved to $\Theta(\sqrt{h} + 1)$ using the result in Section 7.2.

Few results are known for capturing the evader. If the pursuer is as fast as the evader, then a single pursuer is enough (Section 7.3). However, what happens as the evader gets faster is not well-understood.

		Connectedness	
		Simple	**Multiple**
(Visibility, Speed)	**(Global, ∞)**	?	?
	(Global, 1)	1 [Sec 7.3]	3 [4]
	(Local, ∞)	?	?
	(Local, 1)	1 [Sec 7.3]	3 [4]

Table 5.3: Capturing the Evader in a Polygon.

In the next two chapters, we present new results for capturing and locating the evader in complex environments, represented either by graphs (Chapter 6) or simply-connected polygons (Chapter 7).

Chapter 6

Pursuit-Evasion on Graphs

A natural representation for a complex environment is to use a graph. For example, the vertices of the graph can correspond to the rooms of a building and the edges may represent corridors.

Aside from robotics, studying pursuit-evasion on graphs have applications in other fields. These games [87, 82, 77, 67, 13, 2] have been studied not only for their applications in network security and protocol design (e.g. [8, 49]) but also for their relations to fundamental properties of graphs such as vertex separation [29]. A remark about the terminology: In the literature, the names pursuer-evader, cop-robber, monster-princess, hunter-rabbit, sheriff-thief have been used somewhat synonymously. Throughout this chapter, we adopt the hunter-rabbit term for it emphasizes the discrete nature of the game [9, 2].

In this chapter, we address a different aspect of the problem that has not received much attention so far. We study the relationship between the information available to the rabbit and the conditions to capture it. The basic model of our game is as follows: The players are located on the nodes of a graph. At every time step, they move to nodes in their neighborhoods (which includes the current node) simultaneously. We say a rabbit is *caught* or *captured* if at the beginning of a time step it occupies the same node as a hunter. We associate the information available to the rabbit with its visibility. If the rabbit has complete information about the location of the hunter(s) during the entire game, we say the rabbit has *full visibility.* On the contrary, if the rabbit has no information about the hunters, then we say it has *no visibility.*

In our present work, we study the game when the rabbit has *local visibility.* That is, it can only see the nodes that are adjacent to its current location. When the hunter is located at an adjacent node, the rabbit has complete information about his location. However, if the hunter is not visible, then the rabbit must infer the hunter's location based on the time and location of their last encounter. Note that this model is different from the "visibility-based pursuit evasion" work [38, 86], where the goal is to eventually "see" an evader which has complete visibility and unbounded speed.

Recently, Adler et al. studied the game when the rabbit has no visibility [2]. They showed that a single hunter can catch the rabbit on any (connected) graph. The full visibility version has also been studied [82, 13]. It is known that under the full-visibility model, the class of graphs on which a single hunter suffices is the class of *dismantlable* graphs. The number of hunters necessary to capture the rabbit on a graph G is known as the cop (hunter) number of G. It is known that [4] the cop number of planar graphs is at most 3 but the cop number of general graphs is still an open question [81, 32].

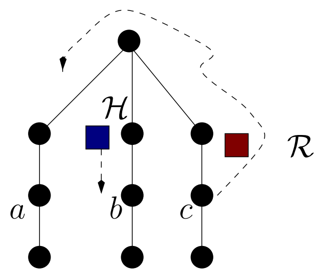


Figure 6.1: On this graph, the hunter can not capture the rabbit using a deterministic strategy.

An interesting aspect of our game is that on most graphs the rabbit can not be captured using a deterministic strategy. A simple example is illustrated in Figure 6.1. Suppose, on this graph, the hunter has a deterministic strategy of visiting the labeled vertices in the order a, b, c. Then, we can design a rabbit strategy that waits until the hunter arrives at b and escapes to a. Afterwards, while the hunter is visiting c, the rabbit escapes to b and it is easy to see that by repeating similar moves, the rabbit can always avoid the hunter. However, on this graph there is a simple randomized strategy for the hunter: Pick one of the leaves at random and visit that leaf!

Therefore, we will focus on randomized strategies. The previous body of work for the full visibility case [82, 13, 81, 32, 4] derandomized the game by forcing the players to move in turns, rabbit followed by the hunter. However, when the players move simultaneously, the game is not well-defined for deterministic strategies even if the players have full visibility: Suppose the game is played on a complete graph. In this case it is easy to see that a single hunter can catch the rabbit simply by guessing its location in the next turn. However, if the hunter's strategy is deterministic, knowing it, the rabbit would never get caught. Similarly, the hunter could always catch the rabbit in a single move if he knew its strategy.

Our results and techniques: Our main result is an algorithmic characterization for the local visibility case. We show that two hunters always suffice on general graphs and present a polynomial time procedure that decides whether a single hunter is sufficient to capture the rabbit on an input graph G. In order to obtain an efficient decision procedure, we establish that the uncertainty in rabbit's knowledge of hunter's location satisfies an interesting monotonicity property. This monotonicity property turns out to be crucial for obtaining a polynomial time characterization.

In the winning strategy for two hunters, a central component is to have one hunter mainly focus on keeping the rabbit on the move. This motivated us to study a natural class of *reactive* rabbit strategies, where the rabbit moves only when the hunter is in its sight. We show that the class of hunter-win graphs (i.e graphs on which a single hunter suffices) against general rabbits is strictly smaller than the class of hunter-win graphs against reactive rabbits. We present a characterization algorithm for reactive rabbits as well.

The characterization algorithms mark pairs of vertices according to certain rules, where the pairs correspond to players' positions. To understand the corresponding hunter strategies on hunter-win graphs, we first present a hunter strategy for the full visibility case. Next, we show that omitting one of the rules from the characterization algorithms yields an algorithm that recognizes graphs that are hunter-win against rabbits with full visibility. Using these two results, we show how the hunter exploits the local visibility if the game is played on a graph G such that on G, the hunter can win against a rabbit with local visibility but not against a rabbit with full visibility.

We note that when the rabbit's visibility is extended to distance 2, there exist graphs for which $\tilde{\Omega}(\sqrt{n})$ hunters are necessary.

This chapter is organized as follows: In Section 6.1, we review necessary concepts that will be used throughout the chapter. In Section 6.2 we present a winning strategy for two hunters on general graphs. Next, we study the graphs on which a single hunter suffices, both for reactive (Section 6.3.1) and general rabbits (Section 6.3.2). Section 6.4 is dedicated to the study of hunter strategies on hunter-win graphs. A gap example distinguishing the power of the two types of rabbit strategies is also presented in Section 6.4. We conclude the paper with a discussion on extensions of our work: In Section 6.5, we study the implications of extending the rabbit's visibility and in Section 6.6 we study the capture time on hunter-win graphs.

6.1 Preliminaries

Throughout the chapter, we use the notation $N(v)$ to denote the set of vertices that are adjacent to v and we always assume that $v \in N(v)$. Unless otherwise stated, n denotes the number of vertices.

The game we study is formally defined as follows: It is played in rounds. In the beginning of a round, suppose a player (either a hunter or a rabbit) is located at vertex v. First, the player checks $N(v)$ and if there is another player located at a vertex $u \in N(v)$, this information is revealed to the player. In this case we say the two players *see* each other. Next, all the players make a decision about where to move and choose a vertex in their neighborhoods. At the end of the round, all players move to their chosen vertex simultaneously. A hunter *catches* the rabbit if they are located on the same vertex.

A *reactive rabbit strategy* is a rabbit strategy where the rabbit is not allowed to move from a vertex v unless the hunter is in $N(v)$. A rabbit strategy is *general* if it is not forced to be reactive. In other words, the rabbit can move even if the hunter is not visible. A *(resp. non-)reactive rabbit* is a rabbit that employs a (resp. non-)reactive strategy. A graph G is *hunter-win against reactive rabbits* if there exists a hunter strategy that catches any reactive rabbit on G with non-zero probability for all possible starting configurations. A graph that is *hunter-win against general rabbits* is defined similarly.

Configuration versus state: For a single hunter game, a *configuration* refers to an ordered pair (h, r) which corresponds to the locations of the hunter and the rabbit respectively. Note that this information may not be available to the rabbit at all times due to its local visibility. A configuration (h, r) is *adjacent* if $h \in N(r)$. We use the notation $< H, r >$ to denote the *state* of the game where r is the location of the rabbit and H corresponds to the set of vertices where the hunter can possibly be located (based on the information available to the rabbit). For the full visibility case, if the current configuration is (h, r), the state is $< \{h\}, r >$. For the zero visibility case, the state is either $< G - \{r\}, r >$ or $< \{r\}, r >$. For the local visibility case that we study, state has a more complex structure, and it evolves over time even when neither the hunter nor the rabbit is in motion.

Suppose u and v are two nodes of a graph G such that $N(u) \subseteq N(v)$. Then, the operation of deleting u from G is called a *folding* of G and we say u *folds onto* v.

A graph is called *dismantlable* if there is a sequence of folds reducing it to a single vertex. We say u *eventually folds onto* v, if there is a sequence $u_0 = u, u_1, \ldots, u_k = v$ such that u_i folds onto u_{i+1}, $0 \leq i < k$. Let G be a dismantlable graph and ψ be a folding sequence reducing G to a single vertex v. We can visualize ψ as a tree T whose vertices are the vertices of G such that when rooted at v every vertex in T is folded onto its parent.

If a graph G is not dismantlable, this means that after a sequence of foldings ψ it reduces to a graph H which can not be folded any further. We refer to the graph H as the *residual graph* of G, or just the *residual*, if G can be inferred from the context. It is known that the residual is unique up to isomorphism [13]. We can visualize the folding process for non-dismantlable graphs as a forest of trees T_h hanging from each vertex $h \in H$ (see Figure 6.3). T_h is composed of vertices that eventually fold onto h and each vertex is folded onto its parent. We define $\psi(u) = w$ if and only if $u \in T_w$, $w \in H$. We note that the tree representation depends on the folding sequence ψ and in general it is not unique.

6.2 A Winning Strategy with Two Hunters

In this section, we present a strategy with two hunters that catches the rabbit on any graph. In general, a single hunter cannot always capture the rabbit. This can be seen by considering a cycle of of length at least 4 as the input graph: The rabbit's strategy is to wait until the hunter becomes visible and move to its neighbor which does not contain the hunter. This strategy guarantees that it will never get caught.

The strategy of the two hunters is divided into epochs that are comprised of two phases. An epoch starts with the hunters located at a predetermined vertex. The first phase starts at time $t = 1$.

In *Phase One*, two hunters move together and their goal is to see the rabbit. To achieve this, the hunters generate a random vertex label $v \in \{1 \ldots n\}$ and move together to v. Afterwards, they wait at v until either $(t \bmod n) = 0$ or the rabbit becomes visible. If the rabbit becomes visible at any time, the first phase is over and the second phase starts. Otherwise, the hunters repeat the same process by generating a new label v.

We claim that the first phase lasts only $n^2 \log n$ steps with high probability. To see this, let $r_1, r_2, \ldots$ be the location of the rabbit at times $n, 2n, 3n, \ldots$ Suppose the hunters have not seen the rabbit until time $i \times n$. At that time, the probability that they generate a label in $N(r_{i+1})$ is at least $\frac{1}{n}$. Since they generate a label after every n steps, the first phase will be over in $n^2 \log n$ steps with high probability.

In *Phase Two* the hunters try to catch the rabbit as follows: Suppose the second phase starts at time $t = t_0$ and let $t_i = t_0 + i$. At that time both hunters H_1 and H_2 are at vertex h and the rabbit is at vertex r, with $r \in N(h)$. For the rest of the second phase, let r_i denote the position of the rabbit at time $t = t_i$ and let us define $r_0 = h$.

The strategy of H_1 is as follows: At time $t = t_i$, he is located at r_{i-1}. With probability $p_1 = \frac{1}{n^2}$, he attacks the rabbit by generating a random neighbor of r_{i-1} and going there in the next step. With probability $1 - p_1$, he chases the rabbit by going to r_i in the next step. The second phase ends with failure if H_1 attacks and misses the rabbit.

The strategy of H_2 is based on the following observation: If H_1 chases the rabbit for more than n steps, the rabbit must revisit a vertex by the pigeonhole principle. Let u be the first vertex revisited and suppose at time t_r, the rabbit visits a vertex $v \in N(u)$ for the first time before revisiting u. The goal of H_2 is to enter v at the same time with the rabbit. To achieve this, first he guesses u, v and t_r. In order to reach u, he chases H_1 by moving to his location in the previous time step until u. Afterwards, H_2 waits until time $t = t_r - 1$ and goes to v from u. We say H_2 is in *chasing mode* if he is following H_1 and he is in *attacking mode* after he arrives at u. The second phase ends with failure if H_2 misses the rabbit when it arrives at v. To summarize, at time $t = t_0$, the hunters are at r_0 and the rabbit is at r_1. When the hunters are chasing, the locations of the rabbit, H_1 and H_2 at time t_i are r_i, r_{i-1}, r_{i-2} respectively. The phase ends when either hunter attacks. If no hunter attacks within n^2 steps, they end the phase and move to the predetermined vertex to start a new epoch.

Next, we state the crucial property of the strategy of the hunters.

Lemma 39 *During Phase Two, the rabbit can not distinguish between the modes of hunter H_2.*

Proof. If the attacking mode starts at time $t = t_1$, the location of H_2 is the same for both modes. If it starts afterwards, we show that if the rabbit sees H_2, it will get caught with non-zero probability.

Suppose the rabbit sees H_2 at time $t = t_2$ which implies $r_2 \in N(r_0)$. In this case, with probability at least $\frac{p_1}{n}$, H_1 can decide to attack from r_0 to r_2 at time $t = t_1$ and catch the rabbit.

Next, suppose the rabbit sees H_2 at time $t > t_2$. If H_2 was in chasing mode at that time, the fact that rabbit sees H_2 implies $r_i \in N(r_{i-2})$. In this case as well, H_1 could decide to attack in the previous step and catch the rabbit with probability $\frac{p_1}{n}$. Therefore H_2 must be invisible to the rabbit during the chasing mode. But, H_2 will also be invisible in the attacking mode because as soon as the rabbit enters a vertex v where it can see H_2, H_2 can catch it by guessing v and the arrival time correctly.

Therefore in order to avoid getting caught, the rabbit must avoid seeing H_2. But then the information available to the rabbit will be same, no matter which mode H_2 is in: H_2 is out of its sight since the beginning of the second phase. $\square$

Lemma 40 *During Phase Two, the hunters succeed with non-zero probability.*

Proof. As discussed previously, after the start of the second phase, the rabbit must revisit a vertex u at time $k \leq n$. If the rabbit does not see H_2 until $t = k$, H_2 can catch it with probability $\frac{1}{n^3}$ at least by guessing $t_r, u, v \leq n$. Note that H_1 will still be chasing the rabbit with probability at least $1 - \frac{k}{n^2} \geq 1 - \frac{1}{n}$. On the other hand, if the rabbit sees H_2, it is caught with probability at least $\frac{1}{n^3} = \min\{\frac{p_1}{n}, \frac{1}{n^3}\}$, by Lemma 39. □

The length of an epoch is $O(n^2 \log n)$: Phase One lasts $O(n^2 \log n)$ time with high probability and Phase Two lasts $\Theta(n^2)$ steps. We have established that in Phase Two, the rabbit is caught with probability at least $\frac{1}{n^3}$. Therefore after $n^3 \log n$ epochs, each of which last $O(n^2 \log n)$ steps at most, the rabbit will be caught, yielding our main result.

Theorem 41 *Two hunters can catch a rabbit with local visibility on any graph with high probability.*

6.3 Hunter-win Graphs

In this section, we start the study of graphs on which a single hunter suffices. An interesting feature of the strategy of two hunters is that one hunter makes the rabbit move constantly, therefore forces it into making mistakes. This suggests that moving when a hunter is not visible may be a disadvantage for the rabbit.

To study this phenomenon we introduce reactive strategies where the rabbit moves only when the hunter is visible and ask the question whether the class of hunter-win graphs against reactive graphs is equivalent to the class of hunter-win graphs against general rabbits. The answer turns out to be negative:

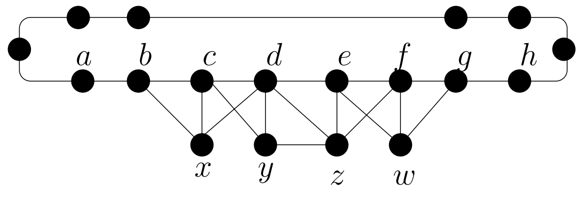


Figure 6.2: This graph is hunter-win against reactive rabbits but not against general rabbits.

The graph in Figure 6.2 is hunter-win against reactive rabbits. The input graph consists of a cycle and the gadget shown in the figure. The hunters strategy is to drive the rabbit into the gadget, by chasing it along the cycle. Once the rabbit is in the gadget, the hunter drives the rabbit to a vertex such that he can reach another vertex (without being seen) whose neighborhood dominates the rabbit's neighborhood. Next, we present the details of the hunter's strategy. In the following, without loss of generality, we assume that the hunter knows the rabbit's next move.

In order to capture the (reactive) rabbit, the hunter first chases it counter-clockwise until the rabbit is at b and the hunter is at a. It can be easily verified that the rabbit can not avoid reaching b without being captured.

If the rabbit moves to x from b, the hunter travels clockwise, arrives at c via y and wins the game (Note that the rabbit, being reactive, will not move in the meantime). Otherwise, if the rabbit moves to c, the hunter moves to b. In the next move, if the rabbit moves to y from c, the hunter travels clockwise, arrives at d through e and wins the game. If the rabbit moves to d from c, then the hunter moves to c.

From d (while the hunter is at c), the rabbit has two options (it will be captured if it goes to y). If it moves to e from d, the hunter goes to y and then to z. The rabbit must then move to w to avoid capture. In this case the hunter goes to f and wins the game. Otherwise, if the rabbit moves to z from d, the hunter travels clockwise again and arrives at e through g and w. From z the rabbit can only go to y, in which case the hunter moves to d from e and wins the game.

Therefore, no matter which strategy it chooses, the hunter can capture a reactive rabbit. However, once it arrives at b, a general rabbit can keep moving in the opposite direction of a until it leaves the gadget. If the length of the cycle is greater than 14, the hunter can not reach the other entrance of the gadget before the rabbit and therefore a general rabbit is safe on this graph.

6.3.1 Characterization of hunter-win graphs against reactive rabbits

In this section, we describe an algorithm that recognizes hunter-win graphs against reactive rabbits. The algorithm marks configurations (h, r) according to the following rules.

```
Algorithm Mark-Reactive:
Mark all configurations $(v, v)$ for every vertex $v$. (Initialization)
Repeat
    Mark $(h, r)$ if for all $r' \in N(r)$, there exists a vertex $h' \in N(h)$ with $(h', r')$ marked. (Stride Rule)
    For all $(h', r)$ that are marked, mark $(h, r)$ for all $h \in N(h') \setminus N(r)$. (Stealth Rule)
Until no further marking is possible.
```

Next, we prove the *soundness* (if all configurations are marked, then the graph is hunter-win) and *completeness* (if the graph is hunter-win, then all configurations will be marked) properties of the marking algorithm.

Soundness: The proof is by induction on the round k in which a configuration is marked.

When $k = 1$ only the configurations (v, v) are marked and the hunter trivially wins the game in these configurations.

Suppose the configurations marked in the first k rounds are sound and consider the configuration (h, r) marked during step $k + 1$. If (h, r) was marked using the stride rule, during the execution of the game, the hunter can force a configuration marked during the k^{th} step with non-zero probability. Hence these configurations are sound. If, on the other hand, the configuration (h, r) is marked by the stealth rule, we observe that the rabbit will remain at vertex r since the hunter is out of its sight and hence hunter can reach the configuration (h', r) which has been marked during the previous steps. Therefore the stealth rule is also sound by the inductive hypothesis.

Completeness: Clearly, if the rabbit is captured the game ends at a marked configuration. Otherwise, we show that the rabbit can always stay in an unmarked configuration and hence never get caught. Suppose there is an unmarked configuration (h, r) and the hunter and the rabbit are at vertices h and r respectively. There are two cases: If $h \in N(r)$, the rabbit must have a move to a vertex r' such that there exists no $h' \in N(h)$ with (h', r') is marked. Otherwise (h, r) would be marked by the stride rule. On the other hand, if $h \notin N(r)$, no matter which vertex h' the hunter moves, (h', r) is unmarked. Otherwise (h, r) would be marked by the stealth rule.

We can now state the result of this section which follows from the soundness and completeness of the marking algorithm.

Theorem 42 *A graph G is hunter-win against reactive rabbits if and only if the algorithm* Mark-Reactive *marks all configurations.*

6.3.2 Characterization of hunter-win graphs against general rabbits

For reactive rabbits, it is easy to see that on a hunter-win graph every rabbit walk can be intercepted (i.e. the rabbit gets caught) by the hunter in $O(n^3)$ steps. However, it is far from being clear that such a polynomial length intercepting walk (i.e a *witness*) exists for non-reactive rabbits. The difficulty is that at any point in time, the rabbit can infer a subset $H \subseteq V$ of possible hunter locations and plan its motion accordingly. This suggests that the state of the game may require specifying arbitrary subsets of vertices, potentially leading to exponential witnesses. Fortunately, we can establish a monotonicity property to establish once again polynomial-size witnesses.

Let $< H, r >$ be the state of the game where H is the set of possible hunter locations when the rabbit is at r. When the rabbit and the hunter are at adjacent vertices r and h respectively, the rabbit knows the hunter's position with certainty and therefore $H = \{h\}$. Now suppose the game starts at configuration (h, r).

Proposition 43 *The hunter can reach an adjacent configuration from any starting configuration* (h, r)*.*

The proof of Proposition 43 is implicit in the strategy presented in Section 6.2. During Phase One, the two-hunters act as one and we showed that their strategy ensures that the hunters and the rabbit will end up in adjacent vertices in n steps with non-zero probability. This means that, no matter which path rabbit takes, there exists a hunter-path of length at most n that leads to an adjacent configuration.

Proposition 44 *A graph G is hunter-win if and only if the hunter wins starting from any adjacent configuration.*

Proof. If the graph is hunter-win, the hunter must win from all starting configurations including the adjacent ones. Conversely, if the hunter can win from any adjacent configuration, then starting from any configuration he can reach an adjacent configuration by Proposition 43, and win the game from here on. □

Therefore by Proposition 44, on a hunter-win graph, we can assume that the game starts from an initial configuration where the players see each other. In addition, without loss of generality, we assume that the rabbit moves so as to maximize the time taken for capture and the hunter moves so as to minimize it.

We can view any hunter-win game as a sequence of rounds $R_1, \ldots, R_p$ where each round starts with the players located at adjacent vertices. Hence, the rabbit has full knowledge of the hunter's position. Clearly, there are at most n^2 rounds and the rounds do not repeat.

Lemma 45 *For the optimal hunter strategy, the length of each round is $O(n^2)$.*

Proof. Partition the round into segments of length $n+1$ each. The rabbit must revisit a vertex r within the same segment. Let $< H_1, r_1 >$ and $< H_2, r_2 >$ be the state of the game during the first and second visits. First, we show that $H_1 \subseteq H_2$. This is because, between r_1 and r_2, the rabbit can not visit any vertex u with $u \in N(h)$, $h \in H_1$: If the hunter is at h, the rabbit would be captured. Next, if $H_1 = H_2$, then the part of the hunter strategy between r_1 and r_2 is redundant and hence the hunter can shorten the game. Therefore as the rabbit keeps visiting the same vertex, its uncertainty is monotonically increasing and after at most n revisits the state of the game becomes $< G - N(r), r >$. In this case, either the rabbit gets caught if it moves or the hunter reveals himself, ending the round. Since the rabbit has to revisit a vertex every n steps and there are at most n revisits, the lemma follows. □

Since the length of a round is $O(n^2)$ and there are n^2 rounds, we conclude that the total length of a hunter-win game is $O(n^4)$.

Our characterization algorithm for general rabbits is based on the existence of such a polynomial size witness. *We will mark only adjacent configurations:* if the adjacent configurations are all marked, by Proposition 44 the hunter wins from all starting configurations. A general rabbit can move even if the hunter is not visible. In order to capture this capability we need to generalize the stealth moves, described next.

Stealth moves

A *k-stealth move* from configuration (h, r) with $h \in N(r)$ to a marked configuration (h', r') is defined as follows: For every rabbit path $P_r = \{r, r_1, \ldots, r_k = r'\}$ of length k, the hunter has a path $P_h = \{h, h_1, \ldots, h_k = h'\}$ such that $h_i \notin N(r_i)$ for $i = 1, \ldots, k-1$, $h_k \in N(r_k)$ and (h_k, r_k) is marked. We refer to P_h as the *stealth path* corresponding to P_r. A configuration (h, r) is marked by the *Stealth Rule* if for all $r' \in N^k(r)$, there exists a k-stealth move to a marked configuration (h', r'). Note that the Stealth Rule for $k = 1$ subsumes the Stride Rule.

Lemma 46 *The markings corresponding to stealth moves are sound.*

Proof. Suppose all previously marked adjacent configurations are sound and consider the next adjacent configuration (h, r) marked by a stealth move of length k. At time $t = 0$ the rabbit is located at r. Since we mark only the adjacent configurations, the state of the game is $< \{h\}, r >$. Take any rabbit path of length k, and suppose at time $t = i$ the rabbit is at vertex r_i. Let $r'_1, \ldots, r'_p$ be the vertices accessible from r_i in the remaining $k - i$ steps and $P_1, \ldots P_p$ be the corresponding stealth paths such that at the end of k steps, P_i ends at vertex h'_i and (h'_i, r'_i) is marked. Let E_i be the event that the hunter has chosen path P_i, $i = 1, \ldots, p$ and let h_i be the i^{th} vertex on P_i. The claim follows from the observation that no matter which path P_i the hunter chooses, the information available to the rabbit is the same, namely hunter was not visible for the last i steps. Therefore the state of the game is $< H, r >$ where $\{h_i | 1 \leq i \leq p\} \subseteq H$. Since the rabbit can not distinguish between the events E_i, no matter which final destination r'_j it chooses, the hunter can be at the corresponding vertex h_j and arrive at the already marked configuration (h'_j, r'_j). $\square$

The stealth moves starting from configuration (h, r) and ending at configuration (h', r') can be computed efficiently by dynamic programming.

We will need an intermediate look-up table T, with $T[h, r, h', r', k] =$ TRUE if and only if for any rabbit path $\{r, r_1, \ldots, r_k = r'\}$ of length k there is a stealth path of length k that starts from h and ends at h'.

The entries of the Table T are filled as follows:

(i) $T[h, r, h', r', 0] =$ TRUE iff $h = h'$, $r = r'$ and $h' \in N(r')$.

(ii) $T[h, r, h', r', 1] =$ TRUE iff $h' \in N(h)$, $r' \in N(r)$ and $h' \in N(r')$.

(iii) $T[h, r, h', r', k + 1] =$ TRUE iff for all $u \in N(r)$ there is a vertex $v \in N(h) \setminus N(u)$ with $T[u, v, h', r', k] =$ TRUE, for $1 \leq k \leq n^2$.

We now present a marking algorithm that uses the look-up table T to compute the stealth moves.

```
Algorithm Mark-General:
Mark all configurations $(v, v)$ for every vertex $v$. (Initialization)
Repeat
    For all configurations $(h, r)$ with $h \in N(r)$, mark $(h, r)$ if there exists an index
    $k \leq n^2$ such that $\forall r' \in N^k(r)$, there exists a vertex $h'$ with $T[h, r, h', r', k] =$ TRUE
    and $(h', r')$ is marked. (Stealth Rule).
Until no further marking is possible.
```

Lemma 47 *If the graph is hunter-win, then the marking algorithm Mark-General will mark all adjacent configurations.*

Proof. Let (h, r) be an adjacent configuration left unmarked after the execution of algorithm Mark-General. We claim that the rabbit can get to an adjacent configuration (h', r') that is unmarked. Suppose not. This means that for any rabbit path $r, r_1, r_2, \ldots, r_k$ there is a hunter path $h, h_1, h_2, \ldots, h_k$ with $h_k \in N(r_k)$ and (h_k, r_k) is marked. By Lemma 45, we have $k \leq n^2$. This implies that (h, r) would be marked by the stealth rule, which gives us the desired contradiction.

Therefore, starting from any unmarked adjacent configuration (h, r), the rabbit can reach another unmarked adjacent configuration. This means that the rabbit will never get caught, since a capture implies that the game enters the configuration (v, v) for some vertex v which is a marked adjacent configuration. □

Theorem 48 *A graph G is hunter-win against general rabbits if and only if the algorithm* Mark-General *marks all adjacent configurations.*

Proof. If all the configurations are marked, G is hunter-win due to the fact that the stealth rule is sound (Lemma 46). Conversely, if there is an unmarked configuration, the rabbit is never caught by Lemma 47. □

6.4 Global Visibility and Dismantlable Graphs

When the rabbit has global visibility, the stealth rule does not make sense. In fact, we will show that the stride rule against reactive rabbits is sound and complete against rabbits with full visibility.

```
Algorithm Mark-FullVisibility:
Mark all configurations $(v, v)$ for every vertex $v$.
Repeat
    Mark $(h, r)$ if for all $r' \in N(r)$, there exists a vertex $h' \in N(h)$ with $(h', r')$ marked. (Stride Rule)
Until no further marking is possible.
```

It turns out that the algorithm Mark-FullVisibility recognizes hunter-win graphs against rabbits with full visibility.

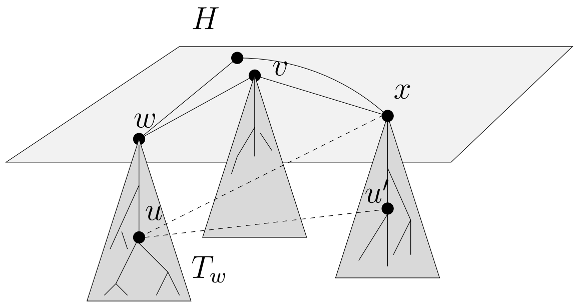


Figure 6.3: Visualization of the folding procedure for a non-dismantlable graph. The vertices w,v and x are in the residual H. Since there is no edge from w to x, the edges shown with dashed lines can not exist.

We will need the following property of non-dismantlable graphs:

Proposition 49 *Let G be a non-dismantlable graph, ψ be a folding sequence and H be the residual. Let x and w be two distinct vertices in H and T_x and T_w be the corresponding folding trees (see Figure 6.3). If there exist a vertex $u \in T_w$ that is adjacent to a vertex $u' \in T_x$, then $x \in N(w)$.*

Proof. Without loss of generality, suppose u was folded before u'. This implies that the parent of u must be adjacent to u'. We replace u with its parent and continue this process of propagating the edge between u and u', which must eventually reach the roots w and x of the corresponding trees. $\square$

Theorem 50 *The algorithm Mark-FullVisibility marks all configurations if and only if the input graph is dismantlable.*

Proof. Suppose the input graph G is dismantlable. We can prove that all configurations will be marked by induction on the order of G. Since G is dismantlable, it must have two vertices u and v with $N(u) \subseteq N(v)$. Let $G' = G - \{u\}$ and run algorithm Mark-FullVisibility on G'. Suppose, inductively, that all configurations in G' are marked. Consider the marking algorithm for G which marks (u, u) first and simulates the marking algorithm on G' afterwards. In addition, whenever (x, v) is marked for a vertex $x \in G'$, we also mark (x, u). This is possible since (x, v) is marked implies that for all $v' \in N(v)$, there exists a vertex $x' \in N(x)$ with (x', v') marked and $N(u) \subseteq N(v)$. Next, we show that all the configurations (x, y) in G' will also get marked in G. Suppose there exists a configuration (x, y) that is marked in G' but not in G. Consider the first such configuration that is discovered in the marking of G. It must be that $u \in N(y)$ and that for all $x' \in x$, (x', u) is not marked at this point. Also, $v \in N(y)$ since $N(u) \subseteq N(v)$. Now using the fact that (x, y) gets marked at this stage in G', we know that there exists $x^{''} \in N(x)$ such that $(x^{''}, v)$ is already marked. But then $(x^{''}, u)$ must also be marked at this point according to the modified marking rule. A contradiction! Thus, any (x, y) marked in G' will also be marked in G. It follows that for any x such that (x, v) is marked in G', we can mark (x, u) in G. It is easy to see that for any x, the configuration (u, x) will also be marked in G since u is adjacent to v and, by the argument above, for all $x' \in N(x)$, (v, x') is marked.

Now suppose the input graph is not dismantlable. Let ψ be a sequence of folds reducing G to a residual graph H. For any two vertices $u \in G$ and $v \in H$, we claim that (u, v) is unmarked if $\psi(u) \neq v$. Suppose this is not true and let (u, v) be the first marked configuration such that $\psi(u) \neq v$ (Figure 6.3). Let $w = \psi(u), w \neq v$. Note that v must have a neighbor x such that $x \notin N(w)$, otherwise v would fold onto w. When (u, v) gets marked, there must be a vertex $u' \in N(u)$ such that (u', x) is marked. If $\psi(u') = x$, this would imply $x \in N(w)$ by Proposition 49. So it must be the case that $\psi(u') \neq x$. But then, the fact that (u', x) is marked contradicts with the fact that (u, v) is the first configuration marked with $\psi(u) \neq v$. Therefore, we conclude that if the graph is not dismantlable, the marking process will not mark all configurations. $\square$

As stated earlier, it has been shown that the class of graphs that are hunter-win against rabbits with full visibility are precisely the class of dismantlable graphs [13]. Therefore we obtain:

Corollary 51 *A graph G is hunter-win against rabbits with full visibility if and only if the algorithm Mark-FullVisibility marks all configurations.*

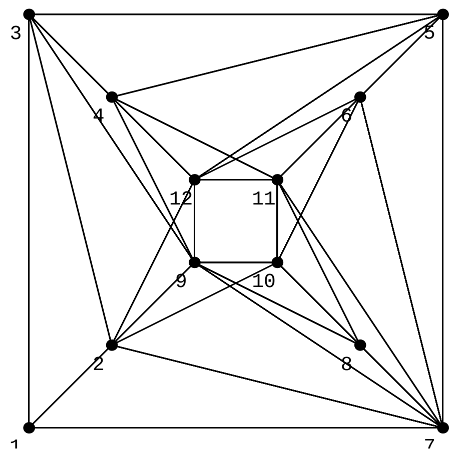


Figure 6.4: This graph is hunter-win against rabbits with local visibility. However, a rabbit with full visibility never gets caught.

We know that there are non-dismantlable graphs that are hunter-win against rabbits with local visibility. An example is shown in Figure 6.4. The labels on the vertices indicate their folding order: First, vertex 1 folds onto vertex 2, afterwards vertex 2 folds onto vertex 9, etc. After folding vertices 1 to 8, vertices 9 to 12 can not be folded, leaving a four-cycle as the residual. Therefore this graph is not dismantlable and consequently it is not hunter-win against rabbits with full visibility. To see that the hunter wins against rabbits with local visibility, let us define the mapping $p : V \to V$ where V is the set of vertices. For $v \in V$ with $1 \leq v \leq 8$, $p(v)$ is the vertex which v folds onto. We define $p(9) = 2$, $p(10) = 8$, $p(11) = 6$ and $p(12) = 4$. The first observation is that the hunter wins the game if he can force the rabbit to go to vertex 1 while he is at vertex 2. Next, we observe that if the rabbit is at vertex $v \neq 1$ and the hunter is at $p(v)$, the rabbit must move to a lower numbered vertex. Now suppose the rabbit is reactive. In this case, it can be verified that for any rabbit location r and for any hunter location $h \notin N(r)$, the hunter has a path to $p(r)$ that does not enter $N(r)$. Therefore, by visiting $p(r)$ repeatedly the hunter can force a reactive rabbit to eventually move to vertex 1 and win the game afterwards.

Hence, the rabbit must have a non-reactive strategy, meaning that it must move when the hunter is not visible. Consider the first time this happens: Suppose the hunter and the rabbit are at vertices h and r with $h \in N(r)$ and the rabbit takes the path $r \to r' \to r''$ such that the hunter is not visible from r'. It can be shown, by enumeration, that for any such vertices h, r, r' and r'', the hunter has a path $h \to h' \to r''$ that captures the rabbit. Therefore the rabbit can not have a non-reactive strategy either and the graph is hunter-win against both types of rabbits.

We conclude this section with an interpretation of Theorem 50: If G is a graph that is hunter-win against rabbits with local visibility but not against rabbits with full-visibility, the hunter captures the rabbit with local visibility using the stealth moves.

6.4.1 Hunter strategy for dismantlable graphs

Given a folding tree T rooted at vertex v, consider the vertex r rabbit is located. We say the hunter is an *ancestor* of the rabbit if he is located on the path from r to v. Suppose the vertices of T are ordered by their deletion times. The hunter strategy is based on the following two lemmas.

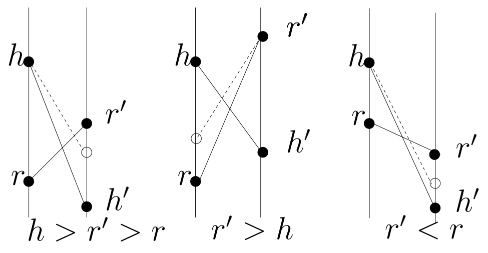


Figure 6.5: Hunter can always stay above the rabbit. The height of a vertex is proportional to its label.

Lemma 52 *Hunter can always maintain ancestry.*

Proof. Suppose the hunter is at vertex h and is an ancestor of the rabbit who is located at vertex r. Let r' be the rabbit's location in the next round. If h is a common ancestor of r and r' on the folding tree T, then the lemma is trivially true. Otherwise, since h is an ancestor of r and (r, r') is an edge, using basic properties of foldings it can be shown that h is adjacent to a vertex on the path that connects r' to the root of T. We show that there is always such a vertex h' with $h' \geq r'$ by a case analysis on r' (See Figure 6.5). Suppose for contradiction $h' < r'$. We will show that h must be adjacent to r' thus allowing the hunter to catch the rabbit in one step.

Case ($h > r' > r$): In this case all the ancestors of h' deleted before h (including r') must have edges to h.

Case ($r' > h$): All the ancestors of r deleted before r' (including h) must have an edge to r'.

Case ($r' < r$): All the ancestors of h' deleted before r (including r') must have an edge to h. □

In fact, not only the hunter can maintain ancestry, but also he can reduce his height in the tree gradually and therefore get closer and closer to the rabbit.

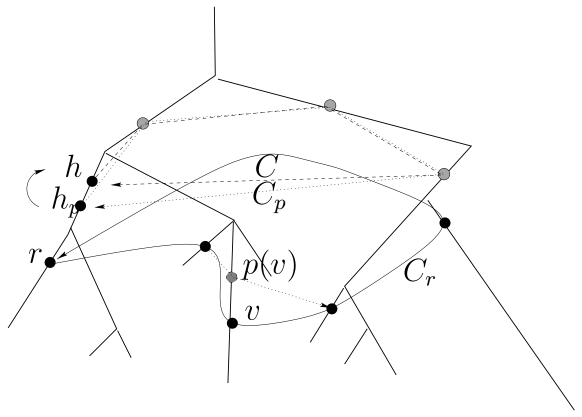


Figure 6.6: Hunter can make progress every time the rabbit revisits a vertex.

Lemma 53 *Every time the rabbit revisits a vertex, the hunter can reduce its height in the tree while maintaining ancestry .*

Proof. Fix any rabbit cycle C_r and let v be the vertex with the lowest label on this cycle and $p(v)$ be its parent (see Figure 6.6). Since v was deleted first, $p(v)$ must have edges to the neighbors of v on the cycle, so we can make a new cycle by replacing v with $p(v)$. We continue this process until the cycle reaches h, the location of the hunter (this must happen since hunter is an ancestor at all times). Let us call this cycle C. Let C_p be the cycle just before C which contains h's child h_p, instead of h. Consider the path $P = \{h\} \cup (C \cap C_p) \cup \{h_p\}$. If the rabbit follows the cycle C_r, hunter can follow the path P and end up at h_p which is lower than h. □

We are now ready to present the hunter strategy on a dismantlable graph G. First, the hunter builds the folding tree T for any folding sequence ψ. Afterwards, he simply guesses the vertex the rabbit will jump to and jumps to the lowest possible ancestor of this vertex (see Figure 6.6). By Lemma 52 he can always remain an ancestor of the rabbit. Further, he can reduce his height in T every time the rabbit revisits a vertex (Lemma 53). Since the tree has a finite height, he can eventually catch the rabbit.

6.4.2 Extension to non-dismantlable graphs

For non-dismantlable graphs, we can extend the notion of ancestry as follows. Suppose the rabbit is at r and the hunter is at h. We say hunter is an ancestor of the rabbit if there is a folding of the vertices such that in the corresponding forest representation, h is located on the path from r to the root of the tree that contains r. Once the hunter establishes ancestry, it is easy to see that Lemma 52 and Lemma 53 still hold –both for reactive and general rabbits. Therefore the hunter can win the game afterwards. Note that the hunter can trivially establish ancestry on dismantlable graphs.

In addition, if we define each vertex as its trivial parent, it is clear that the rabbit wins the game if the hunter can never become an ancestor. Therefore the class of hunter-win graphs is precisely the class of graphs on which the hunter can become an ancestor. One can view the stealth moves as giving the hunter the power to become an ancestor on non-dismantlable but hunter-win graphs such as the one in Figure 6.4.

6.5 Extending the Rabbit's Visibility

Let us define rabbits with *i-visibility* as the rabbits who can see all vertices within distance i. It is known that one hunter always suffices to catch rabbits with 0-visibility [2]. Throughout this chapter, we studied rabbits with 1-visibility and established that 2 hunters always suffice to catch such rabbits. A natural question is how many hunters suffice when the rabbit has i-visibility.

Surprisingly, the number of hunters required for 2-visibility is unbounded: Consider the random bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$ and each edge (u, v) is added with probability $1/\sqrt{n}$.

For an arbitrary vertex u, let x_i be the 0/1 random variable, which takes the value 1 if and only if $(u, i) \in E$. The size of $N(u)$ then becomes a random variable $X = \sum_i x_i$ with the expected value of $E[X] = n \cdot \frac{1}{\sqrt{n}} = \sqrt{n}$.

Using the Chernoff bound (see [79], pp70) with $\delta = 0.5$,

$$Pr[X < (1-\delta)E[X]] < \exp(-E[X]\delta^2/2) = \exp(-\sqrt{n}/8) \tag{6.1}$$

Let E_1 be the event that a vertex has neighborhood of size less than $\sqrt{n}/2$. Using the union bound and Equation 6.1, the probability of E_1 is at most $\frac{n}{\exp(\sqrt{n}/8)}$.

Let us also define the random variable y_i which takes the value 1 if and only if $(u, i) \in E$ and $(v, i) \in E$. Here, $v \neq u$ is an arbitrary vertex. Let $Y = \sum_i y_i$ be the size of the common neighborhood $N(u) \cap N(v)$ with $E[Y] = n \cdot \frac{1}{\sqrt{n}} \cdot \frac{1}{\sqrt{n}} = 1$.

To bound the value of Y, we use the equation (see [79], pp71)

$$Pr[Y > (1+\delta)E[Y]] < 2^{-(1+\delta)E[Y]} = \frac{1}{n^3} \tag{6.2}$$

where δ is chosen such that $(1 + \delta) = 3\log(n)$.

Let E_2 be the event than no two vertices have common neighborhood of size greater than $3\log(n)$. Summing Equation 6.2 over all pairs of vertices and using the union bound, we get the probability of E_2 is at most $\frac{1}{n}$.

The probability that none of the events, E_1 and E_2 happen is at least

$$p = 1 - \frac{n}{e^{\sqrt{n}/8}} - \frac{1}{n} \tag{6.3}$$

Since p becomes non-zero as n grows large, this means that for any (large) n, there exists a graph G^* where every vertex has at least $\frac{\sqrt{n}}{2}$ neighbors and the common neighborhood of any two vertices has size at most $3\log n$.

Now suppose a rabbit with 2-visibility is evading G^*. Note that the rabbit can see the hunters all the time. Without loss of generality, suppose the rabbit is located at a vertex $u \in U$. We can also assume that all the hunters are located in U without any decrease in their power. It easy to see that, on G^*, the number of hunters required is at least $(\frac{\sqrt{n}}{2})/(3 \log n) = \tilde{\Omega}(\sqrt{n})$. Otherwise the rabbit will always have a safe vertex not accessible by the hunters.

6.6 Capture Time on Hunter-win Graphs

When the graph is hunter-win, we know that for any evader strategy, there is a polynomial length hunter strategy for capturing the rabbit. If the hunter had global visibility, then he could watch the rabbit's moves at all times and execute the corresponding strategy. However, even if the hunter has no visibility, he can still capture the rabbit:

Let G be a hunter with global visibility and suppose there exists a winning strategy for G. This means that, no matter what the rabbit does, the game ends in finite time, say T. Let S be the set of all possible rabbit strategies that last at most T steps. Note that S is at most as big as the set of all paths of length at most T.

In order to capture the rabbit, the hunter with local (or no) visibility, say L, picks strategy uniformly at random from S and simulates the strategy of G against the chosen rabbit strategy. If he can correctly guess the (future) moves of the rabbit, he wins the game. Since $|S|$ is finite, the hunter will eventually guess the correct rabbit strategy and capture the rabbit.

Of course, this strategy may cause an exponential increase in the time to capture the evader and it is not clear if this increased capture time is necessary. For example, in Section 6.2, we showed that two hunters can capture the time polynomial in the number of vertices. In the next section, we show that in a large class of graphs, called chordal graphs, a single hunter can capture the rabbit in polynomial time.

6.6.1 A strategy for chordal graphs

In this section, we present a polynomial time pursuer strategy for capturing evaders on chordal graphs.

A graph is *chordal* if every cycle of length at least 4 has a chord – an edge connecting non-consecutive vertices on the cycle.

Definition 54 *Let $C = \{v_1, \ldots, v_k\}$ be a cycle of a chordal graph G. Three vertices v_i, v_{i+1} and v_{i+2} form an ear of C if (v_i, v_{i+2}) is an edge.*

First, we prove the following Lemma:

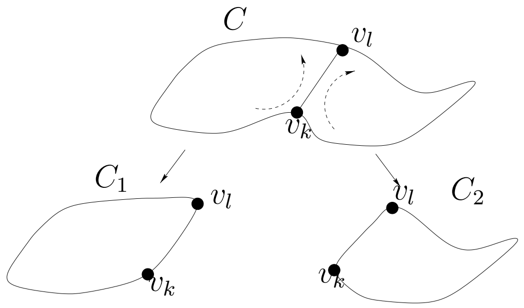


Figure 6.7: Every cycle C of a chordal graph has at least two ears.

Lemma 55 *Every cycle C of a chordal graph has at least two ears.*

Proof: Let $C = v_1, \ldots, v_k$. If $k = 3$ the Lemma is trivially true. Otherwise, since G is chordal, C must have a chord (v_k, v_l) for some k and l. Therefore we can divide C into two cycles C_1 and C_2 that starts and ends at v_k by following C in counter-clockwise and clockwise directions respectively (see Figure 6.7). Further, we can view this process as a tree whose root is C and left (resp. right) child is C_1 (resp C_2). It can be easily verified that the leftmost and the rightmost leaves of this tree give the chords stated in the lemma. ∎

We now present a hunter strategy that catches the rabbit in polynomial time on chordal graphs.

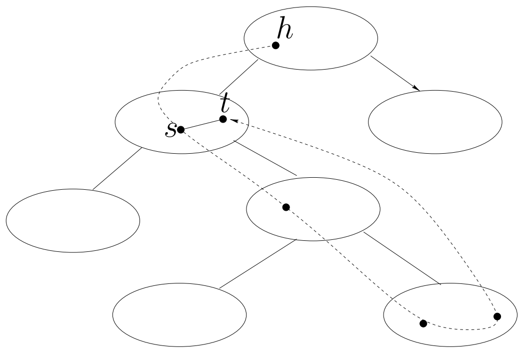


Figure 6.8: Chasing the rabbit on chordal graphs.

The hunter's strategy is divided into phases: In the beginning of a phase, the hunter establishes visibility using the strategy of the previous section. Let h and r be the locations of the hunter and rabbit when they see each other. Consider T, the separator tree of G, rooted such that the separator that contains h is the root of T. The hunter's strategy is to chase the rabbit for t steps and to attack at time t, where t is chosen randomly from 1 to n. Suppose the rabbit jumps up to a vertex t, see Figure 6.8. Since the separators are cliques, there is a vertex s on the cycle such that the edges taken by the rabbit from s on, together with the edge (s,t) is a cycle. Therefore on this cycle there are two indices i and j as defined in Lemma 55 and either $(v_i, v_{i+1}), (v_{i+1}, v_{i+2})$ or $(v_j, v_{j+1}), (v_{j+1}, v_{j+2})$ does not contain the edge (s,t). But this means that, at some point during the chase the hunter was at v_i, the rabbit was at v_{i+1}, the rabbit went to v_{i+2} and $(v_i, v_{i+2}) \in E$. Hence by guessing the time the rabbit will jump up the hunter gets a $\frac{1}{n}$ probability of catching the rabbit at every phase.

Lemma 56 *The hunter catches the rabbit in time $O(n^2 \log n)$ with high probability.*

Proof: It is easy to see that, using the strategy presented in the previous section, the hunter can see the rabbit in time $O(n^2 \log n)$ with high probability. Afterwards, since it catches the rabbit with probability at least $\frac{1}{n}$ at every n steps, by the Chernoff bound it catches the rabbit with high probability after $O(n^2 \log n)$ steps. ■

6.7 Concluding Remarks

In this chapter, we have studied a pursuit-evasion game where the players have only local visibility. We showed that two hunters can catch the rabbit with high probability on any graph. In addition, we presented an algorithmic characterization of graphs on which a single hunter suffices for capture. To the best of our knowledge, this is the only pursuit-evasion game in the literature where the pursuers' strategy explicitly exploits the local visibility of the evader.

An important aspect of the game is the time required to catch the rabbit. For 0-visibility, one hunter succeeds in time $O(n \log n)$ [2]. For 1-visibility we showed that two hunters succeed in $\tilde{O}(n^5)$ time. However, it is not clear whether a single hunter can catch a rabbit on a hunter-win graph in polynomial time. We showed that this is possible for chordal graphs. Our future work includes investigation of this issue for dismantlable as well as general hunter-win graphs.

Chapter 7

Pursuit-Evasion in Geometric Environments

Pursuit-evasion games on graphs allows us to model arbitrary sensing models and environment complexity. It has one drawback though: The motion and visibility are coupled. That is, the set of vertices a player can reach in the next time-step is the same as the set of vertices the player can see. However, in typical mobile robot settings, the sensing range is much larger than the set of points the robots can move in the next time step. Therefore, in this chapter we study the same game played in a polygonal environment.

Perhaps the most well-understood game in this context is the *visibility-based pursuit-evasion* where one or more pursuers try to locate an evader in a polygonal environment [99, 86, 38]. In this game, the evader is very powerful: it has unbounded speed and global visibility, meaning that it knows the location of the pursuers at all times. In [110, 44] the authors study a similar game in a probabilistic framework and propose a greedy algorithm.

The main ingredient of a pursuit-evasion game is the presence of an adversarial evader who actively avoids capture. Due the this aspect, the solutions to pursuit-evasion problems are game theoretic and distinguish themselves from their target finding counterparts (e.g. [91, 106]) where the evader's motion is independent from the pursuer's.

In our present work, we propose randomized pursuer strategies for the visibility based pursuit-evasion problem. Randomization is a powerful technique which allows us to solve many problems that are not solvable by deterministic algorithms and has found wide-spread applications in many areas ranging from computational geometry to cryptography.

As we show in the following sections, it turns out that randomization provides a drastic increase in the power of the pursuers. For example, it is known that there are simply-connected environments where $\Theta(\log n)$ pursuers are required [38] in order to locate the evader with deterministic strategies. In contrast, we show that a single pursuer can locate the evader in any simply-connected environment with high probability, even if the evader knows the pursuer's location at all times and has unbounded speed (Theorem 58). The power of randomized strategies comes from the fact that the evader has no prior knowledge of the random decisions inherent in such strategies. It is worth noting that the randomized strategies work against any evader strategy and require no prior information about the strategy of the evader.

We also address the harder task of capturing the evader. For this problem we present a strategy for two pursuers, one of which is at least as fast as the evader. The strategy is based on the randomized strategy to locate the evader and the solution of a known lion and man problem [92] which is reviewed in Section 7.3.1. The same strategy can be used to capture the evader while protecting a door. This problem was introduced in [72] to model scenarios where the goal is to locate the evader which may leave the polygonal area through a door and win the game.

The two-pursuer strategy can be modified so that a single pursuer can also capture the evader. However, the expected time to capture in this case, though finite, may be significantly longer than the expected time to capture with two pursuers.

7.1 Randomized Strategies

The power of randomization in the context of pursuit-evasion games is nicely illustrated by the example in Figure 7.1. A similar example can be found in [38].

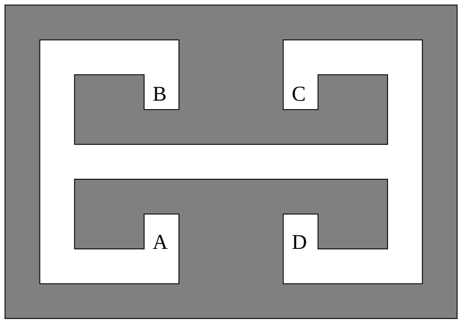


Figure 7.1: A single pursuer can not capture an evader using deterministic strategies.

In this example, a single pursuer can never locate the evader using a deterministic strategy: Let us distinguish four points A, B, C and D as shown in the figure. We will slightly abuse the notation and use these points to refer to the maximal convex region inside the polygon that contains these points as well. Now suppose the pursuer has a deterministic strategy of visiting those points in the order A, B, C, D. In this case, the evader can first hide at B and escape to D while the pursuer is visiting A. Afterwards, it can repeat the same strategy and escape to B when the pursuer is at C. If the pursuer visits the points in a different order, it is easy to see that the evader can find a similar strategy and avoid the pursuer. Therefore, in this polygon one pursuer can never locate the evader.

Now consider the following randomized strategy: Instead of committing to a deterministic strategy, the pursuer selects one of the regions $\{A, B, C, D\}$ uniformly at random and visits that point. It is easy to see that if the pursuer guesses the region where the evader is located, then the evader can not escape and the probability of this desired event is $\frac{1}{4}$. The crucial observation is that since the evader does not know which region the pursuer will visit, it can not choose a strategy based on the order of points visited by the pursuer.

One might suspect that the desired event of locating the evader is not guaranteed since the pursuer can keep guessing a wrong corridor forever. However, this probability can be made arbitrarily small by repeating the same strategy a few times. This is because, if k is the number of trials, the probability of missing in all k trials is only $(\frac{3}{4})^k$ in this example and this probability decreases exponentially in k. In general, if the probability of capture is p, the expected number of rounds to capture is $\frac{1}{p}$. Note that each round is independent. We can obtain the expected time to locate the evader as follows: Since the length of a round is bounded by the time to travel between two furthest points in the polygon (say T), the expected time to capture is $\frac{T}{p}$. We can convert the expected time to a high probability argument by repeating the experiment roughly $\frac{1}{p} \log \frac{1}{p}$ times using the Chernoff bound. For details of this technique the reader is referred to [79].

7.1.1 Preliminaries

Let P be the input polygon including its interior and V be the set of vertices of P. Unless stated otherwise, n denotes the number of vertices of the polygon. Two points $u, v \in P$ can *see* each other if the line segment uv lies entirely in P.

We use $d(u, v)$ to denote the length of the shortest path from u to v that remains inside P. The shortest path has the following property.

Property 1 *The shortest path between any two points u and v inside a polygon P is a polygonal path whose inner vertices are vertices of P.*

The *shortest path tree from a point x in P* is defined as $\cup_{v \in V} d(x, v)$. A polygon is *simply-connected* if any simple closed curve inside the polygon can be shrunk to a point. All the polygons considered in this paper are simply-connected.

The *triangulation* of a polygon is a decomposition of the polygon into triangles by a maximal set of non-intersecting diagonals (see Figure 7.2). The dual of a

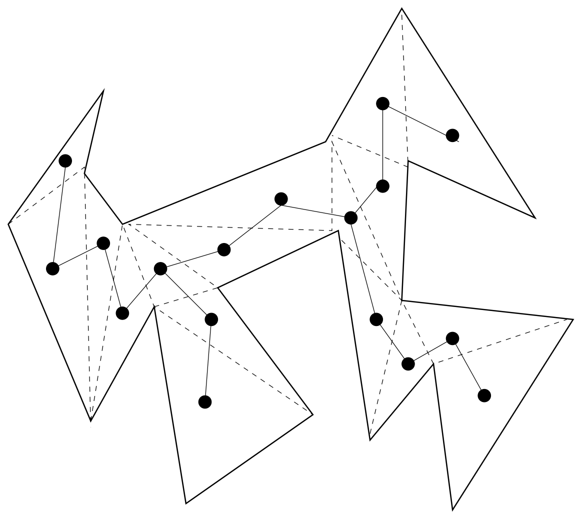


Figure 7.2: Triangulation of a polygon and its dual tree.

triangulation is a graph whose vertices correspond to the triangles. There is an edge between two vertices if the corresponding triangles share a side. It is well known that the triangulation of a simply-connected polygon has exactly $n-2$ triangles. In addition, the dual of the triangulation is a tree [85].

Game formulations

In this paper, we study two pursuit-evasion games. Both games take place in a simply-connected polygon P which is known to all players.

The first game, which we call the *locating game*[1] is defined as follows:

It is played between an evader and a single pursuer. An *evader strategy* is a continuous function $e : [0, \infty) \to P$ such that $e(t)$ denotes the evaders position at time t. The *pursuer strategy*, $p(t)$, is defined similarly.

The pursuer moves with unit speed so that $\dot{p} = 1$. The *diameter* of the polygon P, denoted $diam(P)$, is defined as $\max_{u,v \in P} d(u, v)$. The evader can be arbitrarily faster than the pursuer but it must move continuously.

We assume that in both games, the evader knows the strategy of the pursuer(s)

[1]The general version of this game is known as the visibility-based pursuit evasion game [38]

before the game starts. However, it does not have access to the outcome of the random coin tosses during the execution of the pursuer's strategy. The pursuer, on the other hand, knows nothing about the evader's strategy.

After the game starts, the players can observe their surroundings continuously. Further, at any given time t, the pursuer's location $p(t)$ is revealed to the evader. The pursuer wins the game, if in finite time t^*, it can reach a position such that $p(t^*)$ sees $e(t^*)$. The evader wins the game otherwise, i.e. if, for any given pursuer strategy p, there exists an evader strategy e so that the evader can avoid being seen by the pursuer.

The second game is called the *capture game*:

Let $e(t)$ denote the evader's and $p_i(t)$ denote the i^{th} pursuer's strategy as before. Instead of finding the evader, the pursuers win the capture game if in finite time t^*, they can reach a position such that there exists an i with $p_i(t^*) = e(t^*)$.

In this game, one of the pursuer's is as fast as the evader. Without loss of generality, we assume that $\dot{e} = \dot{p_1} = 1$. Similar to the finding game, the players observe their surroundings continuously and at any given time t, the pursuers' location $p_i(t)$ is revealed to the evader.

We assume that the players move in discrete time intervals and in turns: the evader first, followed by the pursuers.

7.2 Locating the Evader

In this section, we present a randomized algorithm for the visibility-based pursuit-evasion problem [38]:

An evader whose location is unknown to the pursuer is hiding inside a polygon P. The evader has global visibility (i.e. it knows the location of the pursuer at all times) and can be arbitrarily faster than the pursuer. Moreover, the evader is unpredictable – that is, no prior information about its strategy is available. The pursuer's goal is to locate the evader by establishing line-of-sight visibility: He wins the game if he ever sees the evader. The evader wins the the game if it can avoid being located forever.

Next, we show that for any simple polygon P, the pursuer can locate the evader in $O(n \cdot diam(P))$ expected time.

7.2.1 The pursuer strategy

Given polygon P, the pursuer first triangulates the polygon. The pursuer's strategy is divided into rounds of length at most $diam(P)$. Let T be the triangulation tree (see Figure 7.2) rooted at the triangle that contains the pursuer's initial location at the beginning of a round. For any triangle t let $t_1, .., t_k$, $k \leq 3$ be the children of t. We use the notation $T(t)$ to denote the subtree of T rooted at the triangle t. Figure 7.3 is provided for quick reference to the notation used in this section.

The pursuer's strategy relies on the following observation: Suppose the pursuer is inside triangle t and the evader is located inside a triangle contained in $T(t_j)$ for some j. Then, while the pursuer is located at t, the evader can not enter any triangle contained in $T(t_i)$, $i \neq j$ without being seen by the pursuer. This is because the triangle t is a separator for the subtrees $T(t_i)$. Moreover, this property is preserved if the pursuer moves to the triangle t_j.

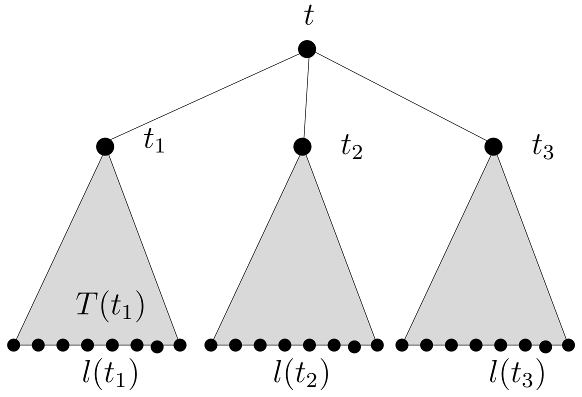


Figure 7.3: Notation used for Lemma 57. Each vertex of the tree corresponds to a triangle in the triangulation tree.

Therefore, had the pursuer known the subtree that contains the evader, he could gradually move towards it while preventing the evader to move from one subtree to another. This process guarantees that the pursuer can enter the triangle which contains the evader and this clearly implies that the evader would be located. Of course, the pursuer does not know where the evader is. This is where we will utilize randomization.

The pursuer will guess the subtree that contains the evader according to the following rule:

Let $l(t)$ denote the number of leaves of the subtree $T(t)$. Suppose the pursuer is located in triangle t and let $t_1, .., t_k$ be the children of t (see Figure 7.3). Let $L = \sum_{i=1}^{k} l(t_i)$. With probability $\frac{l(t_i)}{L}$, the pursuer picks the child t_i and moves there. The round is over whenever the pursuer arrives at a leaf of T.

Next, we show that using this guessing strategy, the pursuer efficiently locates the evader.

Lemma 57 *Let T be the triangulation tree rooted at the triangle that contains the pursuer's initial location in the beginning of the round. At each round, if the pursuer follows the guessing strategy described above, he can locate the evader with probability at least $\frac{1}{L}$.*

Proof. The lemma is proven by induction on the height of T. The basis, where the height of t is 0, corresponds to the case where the input polygon is a triangle. The pursuer trivially locates the evader with probability 1 in this case.

Let $p(t)$ be the probability that the evader is located within a round, starting from triangle t and inductively assume that the lemma is true for all trees of height less than or equal to j.

Given a triangulation tree of height $j+1$, the probability of success starting from the root t is:

$$p(t) \geq \min\left\{\frac{l(t_1)}{L}p(t_1), \ldots, \frac{l(t_k)}{L}p(t_k)\right\} \tag{7.1}$$

Note that all the subtrees $T(t_i)$ have height at most j, therefore by the inductive hypothesis we have $p(t_i) \geq \frac{1}{l(t_i)}$ for all i and the lemma follows. $\square$

Clearly, the number of leaves of any triangulation tree is less than the number of vertices of the polygon, therefore at each round the evader is located with probability at least $\frac{1}{n}$. Moreover, since the length of a round is $diam(P)$, we have the main result of this section:

Theorem 58 *In any simply connected environment P, against any evader strategy, the expected time to locate the evader with a single pursuer is at most $n \cdot diam(P)$ where n is the number of vertices and $diam(P)$ is the diameter of the polygon.*

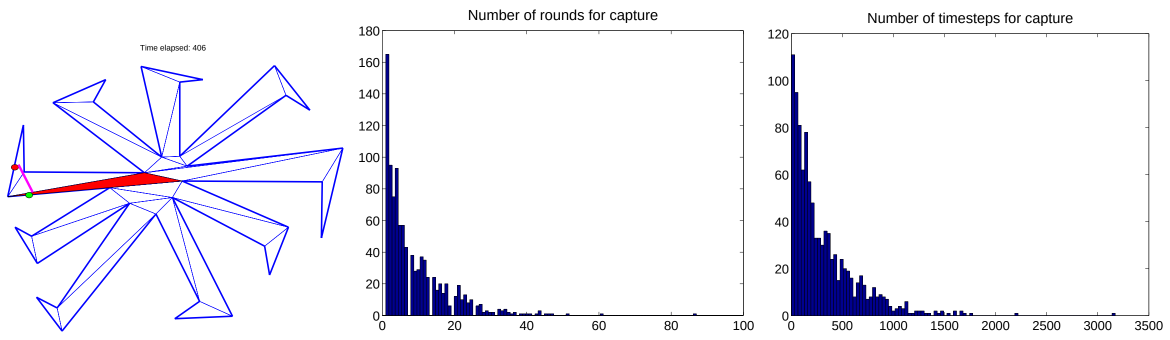


Figure 7.4: **Left:** An instance of the simulator showing the triangulation of the environment as well as the hiding location of the evader. **Middle:** The histogram of the number of rounds required to locate evader in 1000 simulations. **Right:** The histogram of the time-steps required to locate evader in 1000 simulations.

Remark 59 *Any simply-connected polygon can be partitioned into a minimum number of disjoint convex polygons in polynomial time [63, 20]. It is easy to see that the dual of such a partition will be a tree. Therefore, instead of using a triangulation dual, the pursuer can execute the strategy described above using the dual of the convex partition. However, in general this does not improve the expected capture time. For example, for the polygon shown in Figure 7.5, the number of leaves of the triangulation dual is equal to the number of leaves of a minimum convex partition.*

7.2.2 Lower bounds

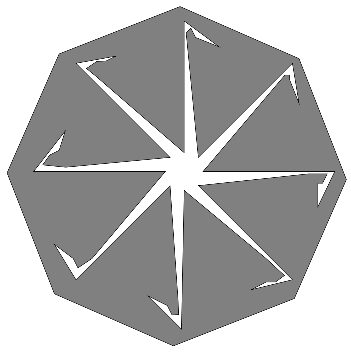


Figure 7.5: For any randomized pursuer strategy, the expected time to capture the evader in this star with hooks is $O(n \cdot diam(P))$.

One might suspect that the expected time to locate an evader can be improved using a more sophisticated strategy. Unfortunately, this is not possible: The polygon in Figure 7.5 is a $k-$star with hooks attached at the end of each spike (in the figure $k = 8$). The evader's strategy is to choose a hook at random and hide there until the end of the game. In order to locate the evader, the expected number of spikes searched by the pursuer is $\frac{k}{2}$ and it takes $diam(P)$ steps to travel from one spike to another. Since the number of vertices is a constant multiple of k, the time it takes to locate the evader is $\Omega(n \cdot diam(P))$. In fact, using the well-known technique due to Yao, this argument can be extended to show that the expected time to capture the evader for *any* randomized pursuer strategy is $\Omega(n \cdot diam(P))$ (see [79] for details).

We present the results of a simulation of the pursuer and the evader strategy for this environment in Figure 7.4.

7.3 Capturing the Evader with Two Pursuers

In this section, we move on to the more challenging task of capturing the evader, defined as moving to the same point as the evader. We assume that there are two pursuers in the game. Later, we will show how to modify their strategy for the case of a single pursuer (at the expense of increasing the expected capture time). We make the following assumptions:

- The pursuers can communicate with each other at all times. [2]
- The evader has global visibility. It is unpredictable, i.e., no prior information about its strategy is available.
- Both pursuers have only line of sight visibility. One pursuer is at least as fast as the evader.
- The game is played in discrete time. In order to simplify presentation, we assume that the players move in turns: first the evader moves, followed by the pursuers.

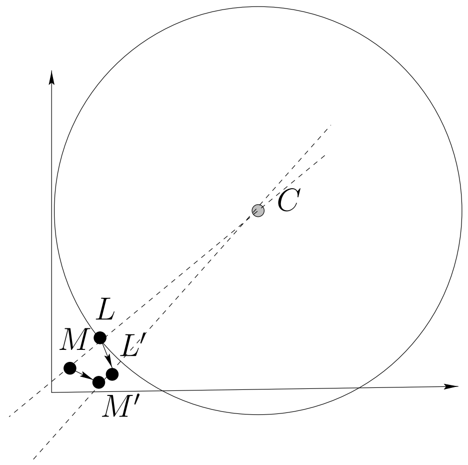


Figure 7.6: Lion's strategy

The strategy of one of the pursuers is based on the solution to a problem known as the *lion and man* problem [92]. We present an extension of this strategy in the case of a (possibly non-convex) polygonal environment. One of the major difficulties for our pursuers is that the evader may not be visible at all times, in which case the

[2]This assumption can be relaxed to line-of-sight communication, i.e. the pursuers can communicate only if they see each other.

lion's strategy is not well-defined. The second pursuer will use the strategy presented in the previous section to tackle this difficulty.

We start with a review of the lion's strategy.

7.3.1 The lion and man problem

The *lion and man* problem with discrete time in the nonnegative quadrant of the plane is attributed to David Gale [39]. Let the initial positions of the lion and man be $L_0 = (x_0, y_0)$ and $M_0 = (x'_0, y'_0)$, respectively. In each round, first the man moves to any point in the quadrant at distance at most 1 from his current position, and then the lion does the same. The lion wins if he moves to the current position of the man. The man wins if he can keep escaping for infinitely many rounds. In [92], Sgall proves that, when both $x'_0 < x_0$ and $y'_0 < y_0$, the lion always catches the man in a finite number of rounds. The number of moves required is bounded by a quadratic function in x_0, y_0 and the slope (or its inverse) of the line segment L_0M_0.

7.3.2 Lion's strategy

Let the initial positions of the lion and man be $L_0 = (x_0, y_0)$ and $M_0 = (x'_0, y'_0)$, respectively. In the beginning of the game, the lion finds a point C on the line M_0L_0 such that L_0 is inside the segment M_0C and the circle with center C, radius $|CL_0|$ and passing through L_0 intersects both axes. Among all possible such circles, it chooses the one whose center is closest to the origin. C remains fixed throughout the game.

Let L and M denote the current positions of the lion and the man respectively (see Figure 7.6). Let M' denote the point the man moves to, $|MM'| \leq 1$. If $|LM'| \leq 1$, the lion catches the man. Otherwise, it moves to a point L' on the line $M'C$ such that $|L'L| = 1$. There are two such points, it chooses the one closer to the man.

Definition 60 *We will refer to this move as the* lion's move from L with respect to

C and M'.

The lion's move maintains the following:

Lemma 61 ([92]) *If the lion does not catch the man in the current move then*

(i) M' has both coordinates strictly smaller than C,

(ii) L' is inside the segment $M'C$, and

(iii) $|L'C|^2 \geq 1 + |LC|^2$.

Proof. See [92]. $\square$

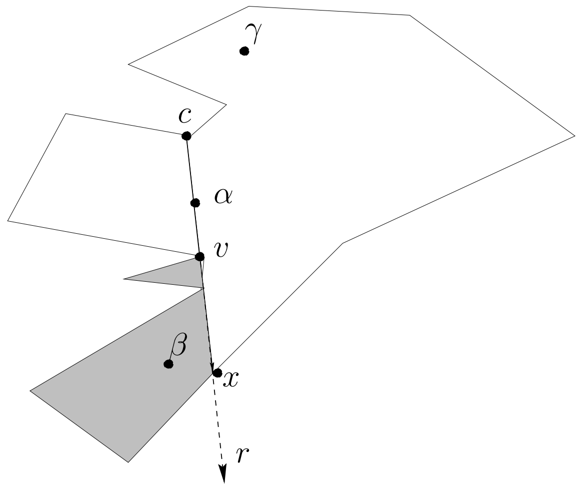


Figure 7.7: Pocket with respect to c and v

7.3.3 The strategy to capture the evader

Let $p_1(t), p_2(t)$ and $e(t)$ denote the locations of the pursuers and the evader, respectively, at time t. In the beginning of the game the two pursuers move together and search for the evader using the strategy described in the previous section. Without loss of generality, we assume that the game starts at $t = 0$ where $p_1(0) = p_2(0) = o$ and $e(0)$ is visible from o. We will sometimes refer to point o as the *origin*. The origin will be fixed until the evader is captured. Let $d_1(t) = d(p_1(t), o)$, $d_2(t) = d(p_2(t), o)$, and $d_e(t) = d(e(t), o)$.

Definition 62 *Suppose $e(t)$ is visible from $p_1(t)$ but $e(t+1)$ is not visible from $p_1(t)$. This means that the shortest path P from $p_1(t)$ to $e(t+1)$ is composed of at least two line segments (Property 1). The first vertex on the path from $p_1(t)$ to $e(t+1)$ is called a* pseudo-blocking vertex.

Let r be the ray starting from a vertex c and passing from another vertex v that is not adjacent to c. In the sequel, c will be the center of the circle for the lion's move and v will be the pseudo-blocking vertex. Consider the first time the ray r leaves the polygon P after it passes through v and let x be the point on $r \cap P$ just before this happens (see Figure 7.7). The line segment vx splits the boundary of the polygon into two chains. The chain which does not contain the point c, together with the line segment vx defines a polygon. We will refer to this polygon as *the pocket with respect to c and v*. The line segment vx is referred to as the *entrance of the pocket.*

We will utilize the following properties of pockets:

Property 2 *Let α be a point on the line segment cv and β be a point in the pocket with respect to c and v. The line segment αv is contained in the shortest path from α to β (Figure 7.7).*

Property 3 *Let R be a pocket with respect to c and v inside a polygon P. Any path from $\beta \in R$ to $\gamma \in P - R$ crosses the entrance of the pocket (Figure 7.7).*

Looking ahead, let us describe how we will utilize these properties: Suppose pursuer p_1 is moving towards the evader and the evader disappears. Let v be the current pseudo-blocking vertex. If p_1 moves towards v, Property 2 implies that it is still moving on the shortest path from the evader to the origin. If the evader becomes visible before p_1 reaches v, Property 3 implies that it must cross the entrance of the pocket and p_1 can continue its strategy (described in the next section) as if the evader has not disappeared.

If the evader is not visible when p_1 arrives at v, then v becomes a *blocking vertex.* At this point, the second pursuer will enter the game.

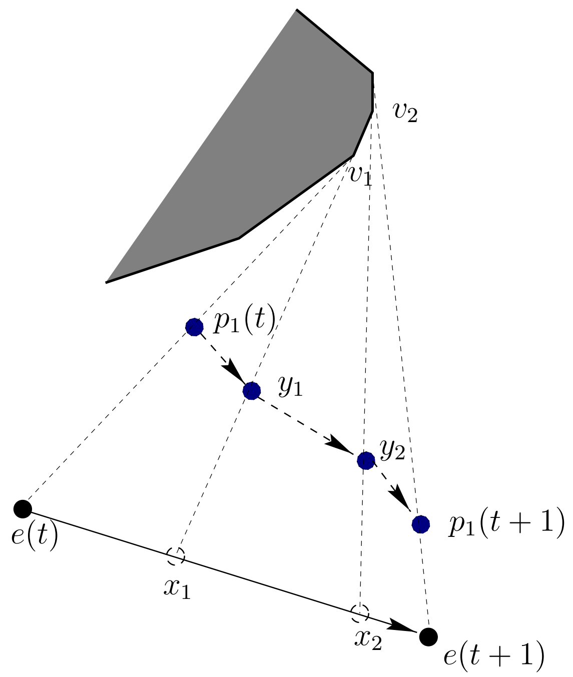


Figure 7.8: The extended lion's move

Next, we present the details of the strategies of p_1 and p_2.

7.3.4 Strategy of Pursuer p_1

As stated earlier, we assume that pursuer p_1 is at least as fast as the evader. At time step t, p_1 moves according to the following strategy:

If the evader is visible, he performs a *extended lion's move* which is defined as follows: Let τ be the shortest path from $e(t)$ to $e(t+1)$. Without loss of generality, p_1 will pretend that the evader followed τ. As a point x moves from $e(t)$ to $e(t+1)$ along τ, the vertices on the shortest path from x to the origin o may change. However, the number of changes is at most n: The first vertex on the shortest path from x to o must be one of the vertices of the polygon. Since τ is the shortest path from $e(t)$ to $e(t+1)$, each vertex of the polygon can be this first vertex for at most one contiguous sub-path in τ. Let $x_1, \ldots, x_{k-1}$ correspond to the points on τ where such changes occur, we define $x_0 = e_t$ and $x_k = e(t+1)$. Let v_i be the first vertex on the shortest path from x_i to o. The extended lion's move consists of $k-1$ phases. During Phase i, $i = 1, \ldots, k$, pursuer p_1 performs the lion's move with respect to v_i and x_i (see Figure 7.8). Note that the time spent by the pursuer in Phase i is equal to the time spent by the evader in traveling from x_{i-1} from x_i.

If the evader was visible in the previous time step, but is not visible any more, let v be the pseudo-blocking vertex. Pursuer p_1 moves towards v until he reaches it. If the evader becomes visible before p_1 arrives at v, he continues with the lion's move. Otherwise, v becomes a blocking vertex.

If the evader is still not visible after p_1 reaches the blocking vertex, he waits for p_2 to report the location of the evader. Let R be the current pocket defined with respect to the blocking vertex and the current center. There are two possibilities.

1) The evader reveals itself to p_1. Then, by Property 3, this must happen before the evader crosses the entrance of R. In this case p_1 continues the game with the lion's move.

2) Pursuer p_2 finds the evader located at e. Let v' be the first vertex on the shortest path from v to e and R' be the pocket with respect to v and v'. In this case, v' becomes a pseudo-blocking vertex, R' becomes the new pocket and p_1 continues his strategy by moving towards v'.

7.3.5 Strategy of Pursuer p_2

The task of pursuer p_2 is to search for the evader when it is not visible to p_1. When the evader disappears from the sight of p_1, pursuer p_2 waits until p_1 reaches the blocking vertex. Afterwards, p_2 locates the evader using the strategy described in the previous section and reports the location of the evader to p_1.

7.3.6 Properties of Pursuer p_1's strategy

Lemma 63 *For all times t, pursuer p_1 maintains the following invariants until the evader is caught:*

(I1) $p_1(t)$ is on the shortest path from o to $e(t)$.

(I2) $d_1(t+1)^2 \geq d_1(t)^2 + \frac{1}{n}$ if $p_1(t) \neq p_1(t+1)$.

Proof of Invariant I1: We prove the invariant by induction. Assume that it holds at time t.

First consider the case where p_1 can see e at time t. Let the first vertex on the shortest path from $e(t)$ to o be u. It follows that $p(t)$ is in the line segment joining u to $e(t)$, since if $p(t)$ is between o and u on the shortest path, he would not be able to see $e(t)$.

Let x denote the evader's position at an arbitrary time in the time interval $[t, t{+}1)$. Suppose when the evader is at x, the first vertex on the shortest path from x to o changes from u to v. Note that p_1 can see the evader until this point. Then, the shortest path from x to o passing through u and the shortest path from x to o passing through v have the same length. This implies that u, v, and x have to be collinear. For otherwise, a shorter path from x to o can be found in the interior of the polygon formed by these two presumed shortest paths from x to o, which is a contradiction.

This implies that either u is an ancestor or a descendant of v in the shortest path tree rooted at o. If u is an ancestor, at the point x where the switch occurs, p_1 could either be on the segment vx in which case it can continue the lion's move in the next phase or p_1 is on the segment uv, in which case e will become invisible to p_1 after x. In this case, p_1 must be either moving towards a pseudo-blocking vertex or waiting at a blocking vertex. In both cases, the invariant is maintained by Property 2. If u is a descendant of v, then p_1 is already on the segment ux and hence on the segment vx. Hence it can continue the lion's move in the next phase. The invariant is therefore maintained as a corollary of Lemma 61.

Otherwise, if p_1 does not see the evader at time t, he must be either waiting at a blocking vertex or moving towards a pseudo-blocking vertex. In both cases, the invariant is maintained by Property 2.

Proof of Invariant I2:

If p_1 is moving towards a pseudo-blocking vertex, his distance to the origin is increasing by 1 and the invariant is maintained.

Next, we show that the extended lion's move maintains the invariant: Suppose the lion's move has $k \leq n$ phases and consider phase i of the extended lion's move where the evader moves from the point x_{i-1} to x_i. Suppose, during this phase the pursuer p_1 moved from point y_{i-1} to y_i (see Figure 7.8) and let v_i be the center of the circle for the lion's move during this phase.

Let $\alpha_i = d(o, y_i) - d(o, y_{i-1})$.

As a corollary of Lemma 61 we have

$d(y_i, o)^2 \geq d(y_{i-1}, o)^2 + \alpha_i^2$.

Summing up over all phases we get the total progress to be $\sum_{i=1}^{k} \alpha_i^2$.

This expression when subject to $\sum_{i=1}^{k} \alpha_i = 1$ is minimized when all $\alpha_1 = \ldots = \alpha_k = \frac{1}{k}$. Therefore we have $d_1(t+1)^2 \geq d_1(t)^2 + \frac{1}{k}$ which implies the invariant I2. ■

The combined strategy of the two pursuers can be viewed as follows: Pursuer p_1 moves only when it knows the shortest path from the evader to the origin o. Performing the lion's move is equivalent to growing a disk inside the polygon whose center is at the origin o and passes through the current location of p_1. By invariant I1, the evader can never enter the disk. Further, the disk is still protected if p_1 does not move. Invariant I2 implies that, whenever p_1 moves, the disk monotonically grows and the evader is eventually squeezed between p_1 and the polygon boundary.

Pursuer p_2 moves only when p_1 does not know the evader's path to the origin. It locates the evader using the randomized strategy given in the previous section and reports its location to p_1 so that p_1, in turn, can keep growing the disk and eventually capture the evader.

7.3.7 Expected time to capture

Let T_1 be the time it takes the faster pursuer (who performs the lion's move) to travel the diameter of the polygon. By Invariant I2 (Lemma 63), this pursuer will capture the evader in nT_1^2 steps. However, in the meantime, the other pursuer may have to search for the evader. Each such search ends in time $T_2 \cdot n \cdot \log n$ with high probability, where T_2 is the time for pursuer p_2 to travel the diameter of the polygon. Moreover, once a vertex becomes a blocking vertex, it will never become a blocking vertex again (it will be included in the ball defined by the origin and p_1), therefore at most n such searches will be necessary. In conclusion, the expected time to capture the evader is $O(nT_1^2 + T_2 \cdot (n^2 \log n))$ with high probability.

7.3.8 Capturing the evader with a single pursuer

Suppose we have only pursuer p_1. In this case, instead of waiting for p_2 to find the evader, p_1 can guess the first vertex on the shortest path from the evader to his current location and move there.

Consider the shortest path tree T from the origin o to the vertices of the polygon. For each vertex v, let $l(v)$ be the number of leaves of the subtree $T(v)$ of T rooted at the vertex v. Then the probability that the pursuer's guess will be successful if he is located at v is at least $\frac{1}{l(v)}$. If the guess is correct and the evader is visible, the pursuer continues with the lion's move. However, in case of a wrong guess the evader may end up in an advantageous location and move towards the origin o, in which case the pursuer must restart the game. Further, if all the guesses are correct, no vertex can be a blocking vertex more than once. Continuing this way we can obtain a worst-case lowerbound on the probability of success. Unfortunately, this bound can be possibly exponentially small in the number of reflex vertices in the environment. However, the expected time to capture the evader is still finite for any simply connected environment and this strategy may still be practical for simple settings.

One might suspect that an analysis similar to the one in Section 7.2 can be applied to prove that the expected time to capture is polynomial. The reason such an analysis does not apply directly is that even if the pursuer and the evader are co-located in a leaf triangle, the capture game still continues and the evader can move to another triangle in the tree. Therefore the number of guesses may exceed the depth of the tree, resulting in a possibly exponential capture time. This poses an interesting trade-off between the pursuer's visibility and the capture time. If the pursuer can somehow track the evader at all times (perhaps using a satellite), then Lemma 63 implies that he could capture the evader in time $O(n \cdot diam(P)^2)$. If this is not possible though, he can either use a second pursuer for locating the evader and still capture it in polynomial time or simultaneously search and capture which results in a much longer capture time.

7.3.9 Polygonal rooms with a door

In [72], Lee et al. studied the following variant of the pursuit-evasion problem: The input is a pair (P, d) where P is the polygonal room the game is played in and d is a *door*, a point marked on the boundary of P. The goal is to devise a strategy for the pursuer to eventually see the evader, in such a way that the evader can not escape through the door. The authors presented a characterization of polygons where a single pursuer with very narrow visibility (represented by a single ray) can locate the evader before it reaches the door.

In a similar scenario, the two pursuer algorithm presented in Section 7.3 can be used to *capture* an evader before it exits through the door. The only modification necessary is the following: Initially, pursuer p_1 is located at the door d and waits until pursuer p_2 locates the evader. Afterwards, he continues with the lion's move with respect to d. This ensures that the evader can never enter the disk whose origin is d and passes through the current location of p_1. Therefore, the door is always protected until the evader is captured.

7.4 Concluding Remarks

In this chapter, we studied the visibility-based pursuit evasion game and showed that using a randomized strategy a single pursuer can locate an unpredictable evader in any simply-connected polygonal environment. The evader may be arbitrarily faster than the pursuer and it may know the location of the pursuer at all times.

The randomized strategy has some desirable properties: First, as shown in [38], there are polygonal environments which require an arbitrary number of pursuers if they are restricted to deterministic strategies. Therefore on such environments, a randomized strategy is mandatory for locating the evader with a single pursuer. Moreover, even if the polygon is deterministically searchable by a single pursuer, it is known that some of these polygons require revisiting parts of the polygon $\Omega(n)$ times [38]. Hence, the expected time to capture with a randomized strategy is comparable to the time to capture with a deterministic strategy. However, the randomized strategies may be preferable to the deterministic strategies, as they do not require complicated data structures and costly preprocessing.

Second, the randomized strategy to locate the evader does not require an exact map of the environment: It is based on the dual graph of the triangulation, therefore it is insensitive to errors in the map of the environment. An interesting research direction is to incorporate the navigation strategies in [106] which require a minimal representation of the environment.

If the environment is entirely unknown, locating the evader may take exponential time. Imagine that the pursuer is searching in a polygon whose triangulation dual corresponds to the skewed tree in Figure 7.9. Suppose the pursuer has no map of the environment, and he is picking the next triangle uniformly at random. If this is the case, the following evader strategy may cause the search to take exponential time. Evader's strategy is to alternate between points A and B. Initially, say, the pursuer is at the root and the evader is hiding at B. To capture the evader, the pursuer must choose to go to the right n times (n = height of the tree), and this would happen with probability $O(2^{-n})$. Therefore the expected time to capture would be exponential in the number of nodes ($= 2n$). Note that whenever the pursuer takes a wrong path the evader can go to A or B (whichever is further to the pursuer) and restart the round.

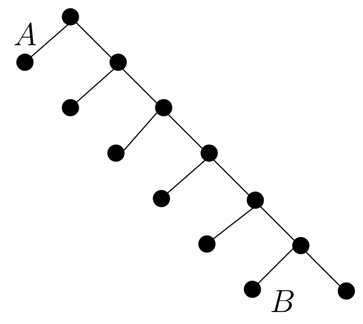


Figure 7.9: A bad case for searching in unknown environments.

Another interesting extension is the case of non-polygonal environments. The randomized strategy can be used to locate the evader in non-polygonal, simply-connected environments. For example, this could be done by replacing the triangulation tree (Lemma 57) with the decomposition studied in [69].

We have also studied the more challenging problem of capturing the evader. For this problem, we presented a strategy for two pursuers (one of which is as fast as the evader) to capture the evader in expected time polynomial in the number of vertices and the diameter of the environment. The strategy can be modified for a single pursuer, however it is not clear whether the expected time to capture remains a polynomial in the number of vertices. We leave this as a future research direction.

Chapter 8

Conclusion

We live in a time of exciting developments in remote sensing systems. On one hand, researchers are deploying networks of sensors onto large, complex environments for monitoring tasks (Figure 8.1). On the other, there are new companies who sell mobile-robot systems for surveillance, exploration and even vacuum-cleaning tasks.

Figure 8.1: Researchers at the Intel Research Laboratory at Berkeley, the College of the Atlantic in Bar Harbor and the University of California at Berkeley deployed wireless sensor networks on Great Duck Island, Maine (http://www.greatduckisland.net/).

We believe that such sensing systems will soon perform in an autonomous fashion

Roomba
Robotic Floorvac
If it's down there, we'll get it.
Check out the entire line of Roomba Robotic Floorvacs.
PatrolBot

(a) http://www.irobot.com/ (b) http://www.packbot.com (c) http://www.mobilerobots.com/

Figure 8.2: Examples of recent commercial robots (a) Roomba vacuum cleaner (b) Packbot exploration robot (c) Patrolbot surveillance robot

and become a part of our lives. Here, the term autonomous is not meant to replace the word intelligent. The core of intelligence lies in the intentions. In other words, the intelligence is in designing “good” objectives. In contrast, what is meant by autonomous here is an action/perception plan so that our robots can achieve these objectives efficiently and without intervention from a human operator.

Even coming up with a mathematically sound specification of an objective is a difficult task. As an example, consider the task of building a map of an environment. In some cases, the environment may be completely unknown. In others, we may have a rough floor plan and want to use our robot to build a detailed 3D model. Now, suppose our goal is to accomplish this task as quickly as possible. To be able to design a plan and to prove that it works we need two models: a model of the environment and a model of our robot.

In modeling the environment, we must address a trade-off between accuracy and modeling complexity. We may want to have as many details as possible (such as cracks along the walls) but for this we must be able to input, manipulate, and output a complex and large environment model.

For the robot, we need a perception model and an action model. These models may span a big range of abstraction. The action model may be very low level, which may involve, for example, specifying the forces to be applied to the motors. We may have slightly higher level commands in the form of "Move 10cm forward". It is also possible to have a programming interface where we can tell our robot to go from point a to point b and expect low-level issues such as collision avoidance to be handled by internal functions. Similarly, for perception we may have a high-level view. For example, our robot may be equipped with a stereo camera pair and we may assume that at any given time we have a 3D model of the portion of the environment seen from our robot. Or, our perception model can be at a lower level which requires us to deal with pixels in the images or the sound waves from a sonar system.

The abstraction level of the environment and the robot model directly impacts our ability to study the action/perception plan. If we adopt a very low level model to accomplish a high level goal, the planning problem can quickly become intractable. As it is the case with many engineering systems, abstraction becomes crucial. To achieve the level of autonomy considered here, we believe that at least a two level abstraction is necessary.

In the high-level, we have a complex model of the environment but a simple model of the robot. For example, the environment can be represented by a polygon with many obstacles inside. The robot model, on the other hand, can be simply a point representing the robot without any kinematic or dynamic constraints on its motion. At this level of abstraction, the goal is to design algorithms that deal with combinatorial issues that arise from environment complexity, sensing complexity or presence of multiple agents.

In the low-level of abstraction, the environment is assumed to be very simple. For example it could be the entire plane without obstacles. The focus is now on the agent model, which can be quite a complex one modeling, for instance, the dynamics of a particular robot. At the low level, the goal is to design algorithms to generate appropriate controls for motion. In addition, the sensing model must be detailed enough to detect obstacles, recognize targets and so on.

The last decade witnessed tremendous advances in the low level of abstraction. We can now buy sophisticated robots for a reasonable price. These robots usually come with a programming interface that gives us many low-level motion and sensing commands. Therefore, the problems in the high level are now the main obstacles in designing autonomous systems. The contributions of this thesis lie in this high-level of system design.

8.1 Summary of Contributions

In this thesis, we focused on distributed and mobile sensing problems for two types of sensing systems: sensor-networks and mobile robots. We identified vision as the main mode of perception. This is because vision provides the most information among the perception modes and hence, cameras are becoming an integral part of robotic systems. Therefore, there is a great need for provable, complete algorithms for sensing systems which utilize vision-based sensors such as cameras or range scanners. However, planning with visibility constraints is a challenging task and such problems are not well-studied in the literature.

In this thesis, we presented solutions to four fundamental problems that arise in design of sensing systems. By viewing sensor-networks as static agents and robots as mobile agents, the problems we study can be categorized as follows:

- In the *exploration* problem, a single mobile agent senses a static environment.
- In the *sensor placement* problem, multiple static agents sense a static environment.
- In the *sensor assignment* problem, multiple static agents sense a dynamic environment.

- In the *pursuit-evasion* problem, multiple mobile agents sense a dynamic environment.

Our contribution for the exploration problem is presented in *Chapter 2*. There, we have studied the problem of how to look around a corner in a polygonal environment in such a way that minimizes the time spent in traveling as well as in reconstruction. We addressed local optimality regarding the visibility of the next occluded edge. This strategy can then be used as a subroutine for building the map of an unknown environment – which is an important application area for robotics.

We presented competitive algorithms for the local exploration problem. These algorithms guarantee bounded deviation from the optimal behavior, even if no information about the environment is available a priori. For the cases where such information is available, we presented a formalization based on Markov Decision Processes. With this formalization, optimal strategies can be computed using well-known techniques.

In *Chapter 3*, we studied the sensor placement problem. Visibility of a polygonal or polyhedral configuration can be cast as a set system with subsets defined by the visibility region of each camera. The minimal guard coverage problem can then be formulated as a minimum set-cover problem. The constraints imposed by the geometry of the setup can be captured with the VC-dimension of the visibility set system. It was known [107] that the upper bound of the VC-dimension for polygons is 23. In Chapter 3, we improved this bound for two cases of exterior visibility: cameras on a circle containing the polygon and cameras outside the convex hull of a polygon. The circle case has significant practical implications because it minimizes the number of stations of a turn-table or a laser-scanner pedestal in 3D-modeling and object inspection. For this case, we present an algorithm that uses at most one more sensor than the optimal number of sensors necessary to capture the object. The placement of cameras outside the convex hull is significant in surveillance and image based rendering. For this case as well as the case of arbitrary placement inside a polygon the existence of approximation algorithms with constant ratio is still an open problem.

In the 3D case, we showed that for any n, a polyhedron with n vertices can be constructed such that the VC-dimension of its exterior visibility is $\Theta(\log n)$. These results provide insights for sampling based sensor deployment algorithms.

In *Chapter 4*, we have introduced the Focus of Attention problem and studied the algorithmic aspects of managing the attention of a group of distributed sensors. We observed that for a general cost metric, the problem is NP-hard and not well approximable. However, for constrained geometric cases we were able to exploit relations between the sensor geometry and corresponding error metrics. From this, we obtained: a 2-approximation for stereo cameras constrained to the same baseline, a PTAS solution for the same geometry when the cameras are spaced equidistantly, and a 1.42-approximation for $4n$-range sensors equi-spaced on the circle. For arbitrary sensor placement, we reposed the problem in a maximization vein. Using results from maximum set-packing, we obtained a $\frac{5}{3}$-approximate solution. This was implemented in simulation, and its performance contrasted against a greedy approach.

Chapter 5 contains an overview of the literature on pursuit-evasion games in complex environments.

In *Chapter 6*, we studied pursuit-evasion games on graphs. As is the case in many robotics applications, we considered the case where the players have only local visibility. We showed that two pursuers can catch the evader with high probability on any graph. In addition, we presented an algorithmic characterization of graphs on which a single pursuer suffices for capture. To the best of our knowledge, this is the only pursuit-evasion game in the literature where the pursuers' strategy explicitly exploits the local visibility of the evader.

An important aspect of the game is the time required to catch the rabbit. For 0-visibility, one hunter succeeds in time $O(n \log n)$ [2]. For 1-visibility we showed that two hunters succeed in $\tilde{O}(n^5)$ time. However, it is not clear whether a single hunter can catch a rabbit on a hunter-win graph in polynomial time. We showed that this is possible for chordal graphs.

Finally, in *Chapter 7*, we studied pursuit-evasion games in geometric environments. First, we studied the visibility-based pursuit evasion game and showed that using a randomized strategy a single pursuer can locate an unpredictable evader in any simply-connected polygonal environment. The evader may be arbitrarily faster than the pursuer and it may know the location of the pursuer at all times.

We also studied the more challenging problem of capturing the evader. For this problem, we presented a strategy for two pursuers (one of which is as fast as the evader) to capture the evader in expected time polynomial in the number of vertices and the diameter of the environment. The strategy can be modified for a single pursuer, however it is not clear whether the expected time to capture remains polynomial in the number of vertices.

8.2 Future Research Directions

In this section, we present future research directions for the problems studied in this thesis.

As mentioned previously, our general approach in the thesis is based on a two-level abstraction. A big open problem is to connect these levels in a provably sound fashion. For example, we obtained pursuit-strategies for point pursuers with no constraints on their motion. Can we extend these strategies for more sophisticated pursuer models?

For some results in the paper, connecting the two levels can be done in a straightforward manner. For example, the probabilistic framework for exploration can be easily extended to handle localization and control uncertainties using Partially Observable Markov Decision Processes [19]. For visibility-based pursuit-evasion, we have preliminary results for extending some of our pursuit strategies to more complex models [52]. Extending the target assignment results beyond the simple models is addressed to a point by our the maximization framework presented in Section 4.2.5.

On the other hand, extending the results on placement, pursuit-evasion for capturing the evader and the general form of the focus of attention problem remains a challenge. Such extensions beyond the simple agent models is an important research problem which will determine the applicability of these results.

We proceed with an overview of individual problems.

Exploration:

The local exploration strategies considered in Chapter 2 can be extended in a couple of directions: Localization uncertainty can be incorporated into the exploration strategy. For the probabilistic strategies, instead of discretizing the circle, it is possible to partition the entire plane. Since this approach is likely to generate a larger state-space, its utility must be explored. Another direction is to study strategies for exploring 3D occluding contours.

Ultimately, the design of global strategies for efficient exploration is a challenging and still open research issue.

Sensor Placement:

Theoretically, the main open problem in camera placement is to close the gap in the optimization version of the art gallery problem: given a simply-connected polygon P, find the minimum number of guards (omnidirectional cameras) required for covering P. For this problem, existence of constant factor approximation algorithms is an open problem.

Of course, in practice, it is difficult to place sensors precisely. They have limited field of views and ranges. These are main research problems that need to be addressed for practical sensor placement algorithms.

Sensor Assignment:

It is important to extend the results in Chapter 4 to more general configurations and error metrics (including error models for more than two sensors tracking a single target). The model in this chapter assumes that the targets do not occlude each other, which may be unrealistic for some applications. Extending this model to incorporate occlusions is a challenging research problem.

Pursuit-Evasion Games:

On graphs, the main open problem is the existence of polynomial length strategies for capturing the rabbit on hunter-win graphs. Other research directions include pursuit evasion games in directed graphs as well as capturing faster rabbits.

There are a number of possible extensions of the algorithms presented in Chapter 7 for simply-connected polygons: An interesting research direction is to incorporate the navigation strategies in [106] which require a minimal representation of the environment. Another interesting extension is the case of non-polygonal environments. The randomized strategy can be used to locate the evader in non-polygonal, simply-connected environments. For example, this could be done by replacing the triangulation tree (Lemma 57) with the decomposition studied in [69].

As noted previously, the expected time to capture the evader in a simply-connected polygon is possibly exponential in the number of vertices. Settling this issue is an important research problem.

None of the strategies described in Chapter 7 work if the polygon is not simply-connected. There are similar issues in higher dimensions. Determining the number of pursuers (and corresponding pursuit strategies) in such environments is a challenging problem. Of course, as the number of pursuers increase, communication requirements become an important issue – making the problem much more challenging.

8.3 Final Remarks

In this thesis we focused on high-level robotics problems where the main challenges are mostly combinatorial and/or geometric. The branch of robotics which deals with these issues is flourishing as a new research field called *Algorithmic Robotics*.

As the robotics systems become more common, we believe that the problems that fall into the domain of algorithmic robotics will be the main obstacles before designing autonomous systems. Hopefully, with efforts from researchers in diverse areas, these obstacles will be overcome and soon we will have fully-autonomous systems performing sophisticated tasks for us!

The results in this thesis appeared in the following publications:

The local exploration problem of Chapter 2 appeared in [55]. The study of VC-dimension of external visibility (Chapter 3) is based on [56]. The algorithm to place guards on a circle has been reported in [54]. Applications of the VC-dimension results to sensor placement has been reported in [53]. Preliminary versions of the results in Chapter 4 appeared in [59, 80].

The results on pursuit-evasion games appeared in two articles: The results on graphs is based on [58] and the results on polygonal environments appeared in [57].

These results are products of research collaborations with my co-authors in these papers. I would like to conclude the thesis by acknowledging their contributions.

Bibliography

[1] A. Acampora. Last mile by laser. *Scientific American*, 287:48–53, July 2002.

[2] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking. Randomized pursuit-evasion in graphs. *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2002.

[3] P. K. Agarwal and S. Sen. Randomized algorithms for geometric optimization problems. *Handbook of Randomization (P. Pardalos et al., eds.), Kluwer Academic Publishers*, 2003.

[4] M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Math*, 8:1–12, 1984.

[5] S. Albers and K. Kursawe. Exploring unknown environments with obstacles. In *In Proc. 9th ACM-SIAM Sympos. Discrete Algorithms, 1998. 13*, 1998.

[6] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovasz, and C. Racko. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[7] Y. Bar-Shalom, X. Li, and T. Kirubarajan. *Estimation with applications to tracking and navigation.* John Wiley, 2001.

[8] T. Basar and P. R. Kumar. On worst case design strategies. *Computers and Mathematics with Applications: Special Issue on Pursuit-Evasion Differential Games*, 13(1-3):239–245, 1987.

[9] P. Bernhard, A.-L. Colomb, and G. P. Papavassilopoulos. Rabbit and hunter game: two discrete stochastic formulations. *Comput. Math. Appl.*, 13(1-3):205–225, 1987.

[10] D. P. Bertsekas. *Dynamic Programming and Optimal Control: 2nd Edition.* Athena Scientific, 2000.

[11] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26(1):110–137, 1997.

[12] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Pres, 1998.

[13] G. Brightwell and P. Winkler. Gibbs measures and dismantlable graphs. *J. Comb. Theory (Series B)*, 78, 2000.

[14] Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding lines and 2-link polygons is apx-hard. In *13th Canadian Conf. on Computational Geometry*, pages 45–48, 2001.

[15] H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[16] N. Bulusu, J. Heidemann, and D. Estrin. Adaptive beacon placement. In *Int. Conf. on Distributed Computing Systems*, 2001.

[17] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proc. of International Conference on Robotics and Automation*, San Francisco, 2000.

[18] R.E. Burkard, R. Rudolf, and G.J. Woeginger. Three-dimensional axial assignment problems with decomposable cost coefficients. *Discrete Applied Mathematics*, 65:123–139, 1996.

[19] A. Cassandra. Optimal policies for partially observable markov decision processes. Technical Report CS-94-14, Brown University, Department of Computer Science, Providence RI, 1994.

[20] B. Chazelle and D. Dobkin. Computational geometry. *Optimal convex decompositions, G. Toussaint (editor)*, pages 63–133, 1985.

[21] W. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.

[22] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized voronoi graph. *The International Journal of Robotics Research*, 19(2):96–125, 2000.

[23] J. Cortés, S. Martínez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1327–1332, Arlington, VA, May 2002.

[24] Y. Crama and F. C. R Spieksma. Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *European J. Oper. Res.*, 60:273–279, 1992.

[25] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment i: The rectilinear case. *Journal of the ACM*, 45:215–245, 1998.

[26] X. Deng and A. Mirzaian. Competitive robot mapping with homogeneous markers. *IEEE Trans. on Robotics and Automation*, 12(4):532–542, 1996.

[27] G. Dissanayake, P. Newman, H. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building. *IEEE Transactions on Robotics and Autonomation*, 17(3):229–241, 2001.

[28] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31:79–113, 2001.

[29] J. A. Ellis, Ivan Hal Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[30] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. Scalable coordination in sensor networks. In *ACM Mobile Computing and Networking (MOBICOM)*, pages 263–270, 1999.

[31] A. Fiat and G. J. Woeginger. *Online algorithms : the state of the art*. Berlin ; New York : Springer, 1998.

[32] S.L. Fitzpatrick and R.J. Nowakowski. Copnumber of graphs with strong isometric dimension two. *Ars Combinatoria*, 59:65–73, 2001.

[33] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2002.

[34] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[35] S. Ghosh. Approximation algorithms for art gallery problems. *Proc. of the Canadian Information Processing Society Congress*, 1987.

[36] H. Gonzalez-Banos, A. Efrat, J. C. Latombe, E. Mao, and T. M. Murali. Planning robot motion strategies for efficient model construction. In *Proc. 9th International Symposium of Robotics Research*, Victoria, Australia, 2001.

[37] H. Gonzalez-Banos and J.C. Latombe. A randomized art-gallery algorithm for sensor placement. *Proc. ACM Symp. on Computational Geometry (SoCG'01)*, pages 232–240, 2001.

[38] L. J. Guibas, J-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications*, 9(4/5):471–, 1999.

[39] R.K. Guy. Unsolved problems in combinatorial games. In *Proc. Symp. Applied Mathematics*, pages 183–189, 1991.

[40] D. Haehnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *In IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS03)*, 2003.

[41] M.M. Halldorsson. Approximating discrete collections via local improvements. *Proc. of Sixth SIAM/ACM Symposium on Discrete Algorithms*, pages 160–169, 1995.

[42] D. Haussler and E. Welzl. ϵ-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[43] C. Hernando, M. E. Houle, and F. Hurtado. On local transformation of polygons with visibility properties. *Theoretical Computer Science*, 289(2):919–937, October 2002.

[44] J. P. Hespanha, G. J. Pappas, and M. Prandini. Greedy control for hybrid pursuit-evasion games. In *Proceedings of the European Control Conference*, pages 2621–2626, Porto, Portugal, September 2001.

[45] V. Hlavac, A. Leonardis, and T. Werner. Automatic selection of reference views for image-based scene representations. *European Conference on Computer Vision*, pages 526–535, 1996.

[46] D. S. Hochbaum. *Approximation algorithms for NP-hard problems.* Boston : PWS Pub. Co., 1997.

[47] Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2002.

[48] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed Sensor Network for Real Time Tracking. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 417–424, Montreal, June 2001. ACM Press.

[49] I.Chatzigiannakis, S.Nikoletseas, and P.Spirakis. An efficient communication strategy for ad-hoc mobile networks. In *Proc. of 15th Symposium on Distributed Computing (DISC'2001)*, pages 285–299, 2001.

[50] C. Icking, R. Klein, and L. Ma. How to look around a corner. *Proc. of the 5th Canadian Information Processing Society Congress*, pages 443–448, 1993.

[51] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29-1:5–28, 1998.

[52] V. Isler, C. Belta, K. Daniilidis, and G.J. Pappas. Stochastic hybrid control for visibility-based pursuit-evasion games. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[53] V. Isler, K. Daniilidis, and S. Kannan. Sampling based sensor-network deployment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[54] V. Isler, S. Kannan, and K. Daniilidis. VC-dimension of exterior visibility of polyhedra. Technical Report MS-CIS-01-34, Computer and Information Science, University of Pennsylvania, 2001.

[55] V. Isler, S. Kannan, and K. Daniilidis. Local exploration algorithms: Online algorithms and probabilistic framework. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Taipei, Taiwan, May 12-17, 2003.

[56] V. Isler, S. Kannan, K. Daniilidis, and P.Valtr. VC-dimension of exterior visibility. *"IEEE Trans. Pattern Analysis and Machine Intelligence"*, 26(5):667–671, 2004.

[57] V. Isler, S. Kannan, and S. Khanna. Locating and capturing an evader in a polygonal environment. In *Sixth International Workshop on the Algorithmic Foundations of Robotics*, 2004.

[58] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with limited visibility. In *"ACM-SIAM Symposium on Discrete Algorithms"*, 2004.

[59] V. Isler, J. Spletzer, S. Khanna, and C.J. Taylor. Target tracking with distributed sensors: The focus of attention problem. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[60] B. Jung and G. Sukhatme. Multi-target tracking using a mobile sensor network. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1050–1055, Washington, DC, May 2002.

[61] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Mobile networking for "smart dust". In *ACM Mobile Computing and Networking (MOBICOM)*, pages 271–278, 1999.

[62] L. E. Kavraki, J.-C. Latombe, and R. Motwani. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57(1):50–60, August 1998.

[63] M. Keil. Decomposing a polygon into simpler components. *SIAM J. Computing*, 14:799–817, 1985.

[64] S. Koenig, C. Tovey, and W. Halliburton. Greedy mapping of terrain. In *Proc. of International Conference on Robotics and Automation*, pages 3594–3599, 2001.

[65] V. S. Anil Kumar, Sunil Arya, and H. Ramesh. Hardness of set cover with intersection 1. In *Automata, Languages and Programming*, pages 624–635, 2000.

[66] K.N. Kutulakos and C.R. Dyer. Recovering shape by purposive viewpoint adjustment. *International Journal of Computer Vision*, 12:113–136, 1994.

[67] A. S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40:224–245, 1993.

[68] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[69] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–201, April 2001.

[70] S. M. LaValle and R. Sharma. On motion planning in changing, partially-predictable environments. *International Journal of Robotics Research*, 16:775–805, 1997.

[71] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

[72] Jae-Ha Lee, Sang-Min Park, and Kyung-Yong Chwa. Searching a polygonal room with one door by a 1-searcher. *International Journal of Computational Geometry and Applications*, 10(2):201–220, 2000.

[73] J. J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *In IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS91)*, 1991.

[74] B. Lisien, D. Morales, D. Silver, G. Kantor, I. Rekleitis, and H. Choset. Hierarchical simultaneous localization and mapping. In *In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS03)*, 2003.

[75] S. Majumder, S. Scheding, and H. Durrant-Whyte. Multi-sensor data fusion for underwater navigation. *Robotics and Autonomous Systems*, 35(1):97–108, 2001.

[76] J. Matoušek. Geometric computation and the art of sampling. In *IEEE Symposium on Foundations of Computer Science – Tutorial*, 1998.

[77] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 1988.

[78] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 296–306, 1992.

[79] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[80] J. Mulligan, V. Isler, and K. Daniilidis. Performance evaluation of stereo for tele-presence. In *Proc. Int. Conf. on Computer Vision*, Vancouver, Canada, Jul. 9-12, 2001. 558-565.

[81] S. Neufeld and R. Nowakowski. A game of cops and robbers played on products of graphs. *Discrete mathematics*, 186:253–268, 1998.

[82] R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Math*, 43:235–239, 1983.

[83] J. O'Rourke. *Art Gallery Theorems And Algorithms*. Oxford University Press, 1987.

[84] J. O'Rourke. Visibility. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 25, pages 467–480. CRC Press LLC, 1997.

[85] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998.

[86] S-M. Park, J-H. Lee, and K-Y. Chwa. Visibility-based pursuit-evasion in a polygonal region by a searcher. *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2076, 2001.

[87] T. D. Parsons. Pursuit evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Application of Graphs*, pages 426–441. Springer Verlag, 1976.

[88] W. Plantinga and C. Dyer. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990.

[89] I. M. Rekleitis, G. Dudek, and E. E. Milios. Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):7–40, 2001.

[90] H. Sahabi and A. Basu. Analysis of error in depth perception with vergence and spatially varying sensing. *Computer Vision and Image Understanding*, 63(3):447–461, May 1996.

[91] A. Sarmiento, R. Murrieta-Cid, and S. Hutchinson. An efficient strategy for rapidly finding an object in a polygonal world. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Optimal Systems (IROS)*, 2003.

[92] Jiři Sgall. Solution of David Gale's lion and man problem. *Theoret. Comput. Sci.*, 259(1-2):663–670, 2001.

[93] R. Sharma and J. Molineros. Computer vision-based augmented reality for guiding manual assembly. *PRESENCE: Teleoperators and Virtual Environments, MIT Press*, 6:292–317, 1997.

[94] T. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80:1384–1399, 1992.

[95] P. Slavik. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25:237–254, 1997.

[96] J. Spletzer and C.J. Taylor. A framework for sensor planning and control with applications to vision guided multi-robot systems. In *Computer Vision and Pattern Recognition Conference*, Kauai, Hawaii, Dec 2001.

[97] I. Stamos and P. K.Allen. Integration of range and image sensing for photo-realistic 3d modeling. In *Proc. of International Conference on Robotics and Automation*, pages 1435–1440, San Fransisco, 2000.

[98] R. S. Sutton and A. G. Barto. *Reinforcement Learning:An Introduction.* The MIT Press, 1998.

[99] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.

[100] K.A. Tarabanis, P.K. Allen, and R.Y. Tsai. A survey of sensor planning in computer vision. *IEEE Trans. Robotics and Automation*, 11:86–104, 1995.

[101] C.J. Taylor and D. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. *IEEE Trans. On Robotics and Automation*, 14(3):147–427, 1998.

[102] S. Teller, J. Chen, and H. Balakrishnan. Pervasive pose-aware applications and infrastructure. *IEEE Computer Graphics and Applications*, Jul/August 2003.

[103] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[104] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002. to appear.

[105] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[106] B. Tovar, S. M. LaValle, and R. Murrieta. Optimal navigation and object finding without geometric maps or localization. In *Proc. IEEE International Conference on Robotics and Automation*, 2003.

[107] P. Valtr. Guarding galleries where no point sees a small area. *Israel Journal of Mathematics*, 104:1–16, 1998.

[108] P. Valtr. On galleries with no bad points. *Discrete & Computational Geometry*, 21:193–200, 1999.

[109] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[110] R. Vidal, O. Shakernia, J. Kim, D. Shim, and S. Sastry. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18:662–669, 2002.

[111] P. Whaite and F. P. Ferrie. Autonomous exploration: Driven by uncertainty. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):193–205, 1997.

[112] G. Yu and O. Goldschmidt. Local optimality and its application on independent sets for k-claw free graphs. *Journal of Combinatorial Optimization*, pages 151–164, 1997.

[113] R. M. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *IEEE International Conference on Robotics and Automation*, May 2002.