# Viewpoint Planning Framework for Single Guard Robot in Indoor Environment

**(屋内環境における見守りロボットの視点プラニングに関する研究)**

**July, 2015**

**Doctor of Engineering**

**Igi Ardiyanto**

イギ　アルディ ヤント

**Toyohashi University of Technology**

DATE : 2015/07/15

| | | | |
|---|---|---|---|
| Dept. of Computer Science and Engineering | | ID | D105319 |
| Name | IGI ARDIYANTO | | |

| | |
|---|---|
| Supervisor | Prof. JUN MIURA |

# Abstract

| | |
|---|---|
| Title | Viewpoint Planning Framework for Single Guard Robot in Indoor Environment |

For supporting human life, a robot needs to be close and interacts with human. Generally, those closeness and interactions enforce the robot to have abilities for recognizing human and having the space awareness. For a specific need, the robot also has to be equipped with a specific ability, too. This ability is often imitated from the human behavior when one faces the same task or situation.

The goal of this thesis is to make a guard robot which imitates the job of the guardians. The robot is given a task to keep an eye on a person inside an indoor environment like the museum, gallery, or exhibition room. Besides the main task of the robot which is to watch the person, the robot also should be aware of the space in the environment in order to take its advantages. For example, the robot can increase the efficiency of the batteries by stopping at the point which has a large coverage. Another benefit is that the stopping robot can reduce noises and blurs in the image frames due to the robot's instability when the robot moves.

To accomplish the above problem, this thesis describes a global planning algorithm for a single guard robot in an indoor environment. The planner is used for the guard robot to continuously and effectively watch a certain object such as a person. Our proposed planner exploits the topological features of the environment, by extracting a set of viewpoints using a generalized coverage solver. We subsequently search for escaping gaps from which the target may go out of the robot's sight. We then plan the action for the robot based on a geodesic motion model and escaping gaps. A stochastic approach and a greedy method are presented to choose an optimal action. Particularly, a particle-based approach combined with a chance constraint bound is utilized for ensuring that the target person is always under the robot visibility. Experimental results using a 3D simulator and the real robot are provided to show the effectiveness and feasibility of our algorithm.

# Contents

# List of Figures

# List of Tables

# List of Theorems

# Chapter 1

# Introduction

## 1.1 Research Backgrounds

Recently, robotic technologies have been pushed forward and greatly demanded in many applications. Its utilization is broad, ranged from home appliances to industrial sectors. There are couples of rationale behind the exertion of the robots; to alleviate the human's jobs, to substitute the human on hazardous works, or to exploit its precision for some specific tasks. When a robot is required to replace or to support the human in a certain task, it implies that the robot needs to imitate, fully or partially, what the human workers do. This imitation can be in the form of actions, procedures, or even the human thinking perspective to finish the job. With regards to such problem, here we present one example of the task on which a robotic framework is expected to solve, as described by the following problem setting.

Let us imagine an indoor environment such as museum, office, gallery, or exhibition room. Suppose there is a *Very Important Person* (VIP, e.g. Minister, Chief, or Officer) visiting the museum and needs to be guarded. Ordinarily, a group of guardians (or so-called "securities") are assigned to do the guarding job. The most important task of these guardians is to watch over the VIP for the entire time, yet they should not disturb and restrict his/her mobility. Additional duties may be added as well, such as documenting the overall activities of the VIP.

Now we aim to substitute the human guardians with the robotic platform. The task imitation for the above case is straightforward, the robot should mimic the behavior of the human guardians. Nevertheless, several considerations need to be contemplated:

1. The number of robots used for guarding the VIP eventually becomes one important factor, if the cost is restricted.
2. The robot has limitation on the battery capacities, affecting its working time duration.
3. If documenting the activity of the VIP is included as the robot task, it is preferable to highly reduce the robot's instability (e.g. due to its excessive movement), to acquire a *less-noise*

video or image.

4. The robot should be *non-intrusive*, i.e. it must not disrupt and alter the current activity of the VIP.

Considering the above restrictions, we propose a novel planning algorithm for a single guard robot to imitate the job of the human guardians. Since the robot price tends to be very expensive, here we emphasize the use of single robot for guarding purpose as indicated by point (1) in the above consideration. The robot task is then minimally rephrased as follows:

**Definition 1.1.1** (Guard Robot)**.** *The guard robot should maintain the visibility towards the target person, while minimizing its movement.*

The term "minimizing its movement" is raised, basically for tackling the problems specified by point (2) and (3). By minimizing the robot movement, we expect to reduce the energy which may lead to a longer duration of the robot working time. It also means the robot tends to be in an idle condition. As the result, a *less-noise* video of the target person (when we decide to take a documentation) can be obtained due to the stability improvement.

Based on the above expectations, our main idea is to make the robot moves only when it is needed, i.e. whenever the target is predicted to leave the robot's field-of-view, and mostly stays at a certain location which holds a wide view. We then introduce *viewpoint* terminology in our proposed guard robot planner. The viewpoint is described as a point from which the idle robot can safely and steadily watch over the target person for a long time.

The viewpoints are utilized to assist the action planning processes. Our strategy is to move the robot from one viewpoint to another. By exploiting the viewpoints, we expect to reduce the search spaces of the robot. We also introduce a concept of *escaping gaps* to lessen the target person prediction space, which makes the guard robot problem become more tractable. Using this approach, the robot will not disturb the target person since it attempts to "see from a distance", granting the consideration pointed by point (4) above.

## 1.2 Related Work

### 1.2.1 Art Gallery Problem

In the matter of guarding an indoor environment, the *Art Gallery Problem* (AGP) is related to our proposed guard robot. The AGP is defined as a problem of discovering a minimum number of guards, typically in the form of sensors or cameras, such that it covers all interiors of the environment. The AGP has been widely studied, especially by computational geometry communities (e.g., [1], [2], [3], and [4]).

The use of viewpoints in our proposed approach can be perceived as another form of the AGP. In conjunction with the planner, our proposed algorithm creates a dynamic version of the AGP, with a single robot be in charge of guarding the entire environment by visiting the viewpoints as needed. Contrarily, the original AGP statically puts a set of guards for the same purposes.

### 1.2.2 Pursuit-evasion

The classical pursuit-evasion problem aims to make the pursuer(s) capture the escaping evader(s). This problem is also highly addressed by a vast number of researches, e.g., [5], [6], [7], [8], and [9]. By the nature, one may guess that our guard robot is a variant of pursuit-evasion problem where the evader (target person) does not try to escape. One notable thing is that in our guard robot problem, the game does not necessarily end when the target is already “captured”. Instead, the robot will continuously and optimally act to cover the target until it leaves the environment.

### 1.2.3 Watchman Route Problem

This problem aims to plan the shortest route on which a robot or watchman can inspect every point inside a polygon. The watchman route problem (WRP) is also popular among the robotic and computational geometry researchers, e.g., [10], [11], and [12]. At a glance, our proposed problem resembles the classical WRP, except the guard robot has a target to be “followed" and its additional objective is to minimize the movement.

### 1.2.4 Person Following Robot

The most closely related works to our guard robot problem is the person following robot, which has a long time history in the robotic researches (e.g., [13], [14], [15], and [16]). This algorithm is also suitable for solving the proposed problem. The main difference is that the person following algorithm continuously makes the robot move and attempts to be within a certain distance towards the target. As the opposite, our approach versatilely tries to understand the environmental topology and only moves when it is necessary to keep the target person under its vicinity. By this strategy, it is expected to reduce the energy used by the robot.

## 1.3 Contributions

Our contributions mainly lay on the introduction of a novel framework and the cooperation between the topological viewpoints and an on-line planning strategy for solving the guard robot problem. It is also worth to note that we are raising a distinctively new variant of robotic problem, compared with the previously mentioned related works. This new task includes the problem of keeping a target under the robot visibility while reducing its movement.

Besides the contributions related to the viewpoint planning above, we also devote several novelties on its building blocks, as follows:

1. In **coverage problem**, we introduce a generalized algorithm for solving the coverage area under arbitrary cost function, which leads to the unified solver for coping with several famous problems (e.g. *Art Gallery Problem*, *Sensor Coverage*, and *Robot Coverage*).
2. In **motion planning problem**, we propose a potential based randomized tree algorithm for optimizing the robot movement.
3. In **person tracking problem**, an incremental improvement is introduced by applying an upper body-based detection and tracking for ensuring the robustness of person tracking algorithm.

Each contribution will be explained further on the following chapters related to each problem.

## 1.4 Thesis Organization

This thesis is organized as follows. We first describe a mathematical definition of our guard robot, concept of *viewpoints* and *escaping gaps* utilized in our approach, as well as the strategy for simplifying the guard robot problem and its proposed building blocks in chapter 2. The next chapter explains the extraction of viewpoints by utilizing the topological features and a heuristic linear programming method. Chapter 4 describes the viewpoint planning algorithm, which elaborates both deterministic and probabilistic approaches, and utilizes a geodesic motion model for exhibiting both the robot and the target person movements. Chapter 5 and 6 respectively elucidate supporting algorithms for completing implementation of the viewpoint planning algorithm, that is the path planning for the mobile robot and the person tracking algorithm. The proposed viewpoint planning approach is then verified on various experiments in chapter 7. In the end, the conclusion and future directions of our research are provided.

# Chapter 2

# Defining Guard Robot Problem

## 2.1 Problem Definition

Consider a typical indoor workspace $\mathcal{Q} = \{\mathcal{Q}_f, \mathcal{Q}_n\} \subseteq \mathbb{R}^m$, $m \geq 2$, constrained by walls and possibly obstacles. $\mathcal{Q}_f$ and $\mathcal{Q}_n$ are respectively perceived as "passable" and "non-passable" regions for the robot. Let $\{q_t^r, q_t^h\} \in \mathcal{Q}$ denote the robot and the target person state at a certain time $t$. The robot is equipped with sensors to sweep the free space, creating a continuous boundary which depicts the "visible" area for the robot, denoted by $\oint B(q_t^r)$. The guard robot problem is subsequently described by following mathematical expression

$$\text{Minimize} \quad \int_0^\infty \left| \frac{\partial (q_t^r)}{\partial t} \right| dt \tag{2.1}$$

$$\text{s.t.} \quad \oint B(q_t^r) \bigcap \overline{q_t^h q_t^r} = \emptyset \tag{2.2}$$

$$\{q_t^r, q_t^h\} \in \mathcal{Q}_f, \tag{2.3}$$

where $\overline{q_t^h q_t^r}$ denotes a straight line connecting $q_t^h$ and $q_t^r$.

The above expression implies we attempt to reduce the total movement of the robot, indicated by summation of the first order derivative of the robot state over the time in eq. (2.1), while the main constraint is to keep the target $q_t^h$ within the visibility boundary of the robot (eq. (2.2)). Obviously, the robot state's derivative can be approximated by discrete-time dynamic model

$$\frac{\partial (q_t^r)}{\partial t} \approx f(u_t, \varepsilon_t) : q_t^r \mapsto q_{t+1}^r, \quad t \in [0, \infty], \tag{2.4}$$

where $u_t \in \mathcal{U}$ is the control input and $\varepsilon_t$ denotes the uncertainty. Hence, eq. (2.1) gives an implication that we are minimizing controls applied to the robot.

Solving the above optimization problem analytically is challenging, due to following reasons:

1. The future information of the target state $q_t^h$ is not available at $t > 0$, assuming $t = 0$ is the

current state, unless it is carefully modeled according to the environment (still, the uncertainty will be large).

2. The robot states, dynamics, and sensors may enclose the uncertainty too.
3. Since the visibility boundary of the robot is unique for each state $q_t^r$, it means the optimization should examine the constraint over a huge space of all probable future states of both the robot and the target person.
4. The robot goal space is very large too, whereas the entire workspace could become a destination for the robot to move, as long as it satisfies the visibility constraint.

## 2.2 Simplifying The Guard Robot Problem

As mentioned above, the guard robot problem encounters some challenges. In a nutshell, those problems are a very large robot goal space and the future information uncertainty of the robot and the target person. Here we contemplate to ease those challenges using the following approaches.

### 2.2.1 Polygonal representation of the environment

We aim to relax the workspace $\mathcal{Q} = \{\mathcal{Q}_f, \mathcal{Q}_n\}$ into a simpler 2D planar polygonal form, as the environment map tends to have a complex shape. A set of procedures for simplifying $\mathcal{Q}$ is then implemented (similar steps can be found in [17]), as follows:

1. **Binarization**. A binary map $\mathcal{I}(q)$ is obtained by mapping each state $q \in \mathcal{Q}$, as follows

$$\mathcal{I}(q) = \begin{cases} 1 & \text{for } \forall q \in \mathcal{Q}_f \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

2. **Smoothing**. For reducing noises in the map, morphological operations are performed, using *opening* and *closing* operators.
3. **Contour extraction**. An algorithm introduced by Suzuki, *et al.* [18] is subsequently engaged to extract the contour from the binary map $\mathcal{I}(q)$, yielding an outer contour $\oint B^{outer}$ and (possibly) $k$-inner contours $\oint B_k^{hole}$. Both are a set of closed segment chains.

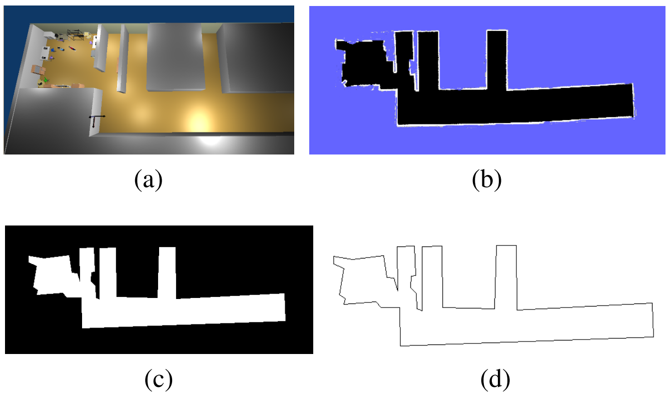


**Figure 2.1:** Simplifying environment map: (a) original environment, (b) map from SLAM algorithm, (c) binarized map, (d) extracted polygon.

4. **Line segments simplification**. Finally, Douglas-Pecker algorithm [19] is used for simplifying the contours $\oint B^{outer}$ and $\oint B_k^{hole}$. It produces a closed, connected polygon $\mathcal{P}$ with the outer boundary $\delta\mathcal{P}$ and $k$-inner boundaries $\delta\mathcal{H}_k$ where $k$ denotes the number of holes[1] inside $\mathcal{P}$. A point $p$ satisfying

$$\{p \in \{\mathcal{P} \cap \neg(\bigcup^{k} \mathcal{H}_k)\}\}, \tag{2.6}$$

is called the *interior point* of $\mathcal{P}$. From now eq. 2.6 is written by $p \in \mathcal{P}$ to describe the interior point $p$, for the sake of simplicity. The state $q \in \mathcal{Q}$ is also interchangeable with $p$ such that $q \in \mathcal{P}$ has the same meaning with $p \in \mathcal{P}$.

Figure 2.1 shows the instance of simplified map created from the original environment. The white area in Fig. 2.1c represents the free space for the robot and target person to move.

### 2.2.2 Visibility polygon

Visibility polygon becomes one of fundamental problems in our guard robot, since we deal with the visibility constraint of the target person towards the robot. The visibility polygon of a point is given by following definition

**Definition 2.2.1** (Visibility Polygon)**.** *Let $q$ be a point inside $\mathcal{P}$, another point $s \in \mathcal{P}$ is considered visible from $q$ if $\overline{qs} \subset \mathcal{P}$, where $\overline{qs}$ is a line segment.*

[1]A geometrical term to define a closed polygon which is not connected to the exterior boundary.

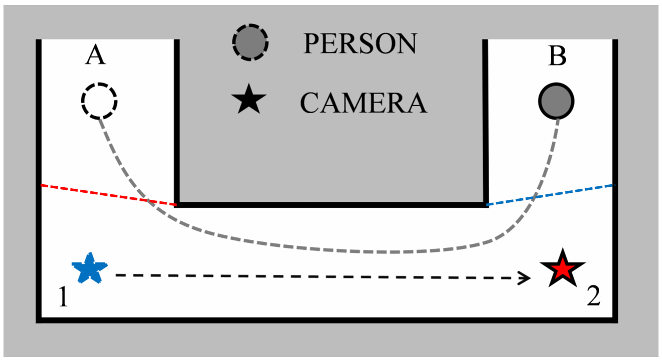


**Figure 2.2:** Illustration of camera switching to keep the target under visibility. When the person moves from A until the blue line (camera 1 visibility), camera 1 does the “watching" task over the person. The “watching" task is then taken over by camera 2 from the blue line until the person reaches B.

The visibility polygon $\mathbb{V}(q)$ is then defined as

$$\mathbb{V}(q) = \{\forall s \in \mathcal{P} \mid \overline{qs} \subset \mathcal{P}\}. \tag{2.7}$$

### 2.2.3 Concept of viewpoints for reducing the robot goal space

Let us take a look on Fig. 2.2. When static guards are used (e.g. cameras), the art gallery problem (AGP) can be adopted for solving the target person visibility problem. By a finite number of cameras, the AGP holds a guarantee to cover the entire building. Thereafter, to watch over a person inside it, one can easily switch his attention to the camera on which the person is visible. Figure 2.2 portrays the camera switching process when a person moves from A to B. This also applies to all other cameras.

By substituting each camera with a virtual point, let us imagine a robot is used to “see" the same person. The camera switching process now becomes the robot movement from the virtual point 1 to 2, to hold the same visibility towards the person. These virtual points are named *viewpoints*. The guard robot problem subsequently becomes the problem of determining the feasible viewpoint for the robot to watch over the target person.

The main point of the above illustration is, we basically expect to reduce the large goal space of the guard robot into a small set of viewpoints $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, which hold the same visibility guarantee towards the target person. This property becomes the basis of our proposed approach for the guard robot problem.

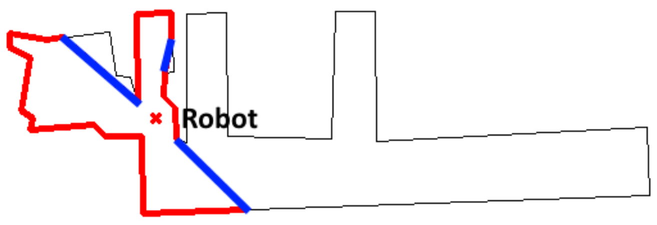


**Figure 2.3:** Escaping gaps representation, denoted by blue lines. Together with the red lines, it forms the visibility polygon of the robot (marked by the red cross).

## 2.2.4 Concept of escaping gaps

As we restrict our problem to an indoor environment, we can exploit the fact that the person movement should not violate the environment restrictions, e.g. walls. Accordingly, the possible scenarios for the robot to lose its visibility towards the target are when the target passes *escaping gaps*.

Escaping gaps are described as a set of locations at which the target may vanish from the robot's view. It has a similar idea with the popular term *frontiers* for exploration of an unknown space (e.g. [5]).

**Definition 2.2.2** (Escaping Gaps)**.** *Recall $\delta\mathcal{P}$ as boundary of the workspace polygon $\mathcal{P}$ and $q$ is a point inside $\mathcal{P}$. Let $\delta\mathcal{V}(q_t^r)$ denotes boundary of the visibility polygon of the robot at current time. The escaping gaps $\mathcal{G}$ is then defined as*

$$\mathcal{G} = \{\forall q | q \in (\delta\mathcal{V}(q_t^r) \bigcap \neg\delta\mathcal{P})\}. \tag{2.8}$$

Figure 2.3 visually describes eq. (2.8). The blue line in Fig. 2.3 represents escaping gaps which lie on the visibility polygon but do not lie on the workspace boundary. Typically, the escaping gaps $\mathcal{G}$ are in the form of a set of lines, as shown in Fig. 2.3. These lines are subsequently discretized into a set of points, such that every point in $\mathcal{G}$ is separated by a minimum distance $d_g$ (currently, $d_g = 0.25$ meters).

While the viewpoints are important for simplifying the goal space for the robot, the escaping gaps are indispensable for relaxing the visibility constraint and the prediction space of the target person. Let $\forall\psi \in \Psi$ be all possible future paths which can be taken by the target person bounded by $\delta\mathcal{P}$. The path space is possibly infinite unless it is well modeled, denoted by $\Psi_{model} \subset \Psi$. Let $\Psi_g \subset \Psi, \forall g \in \mathcal{G}$ be the target person paths towards escaping gaps. Since $\Psi$ is bounded, the only possible paths breaking the visibility constraint are $\Psi_g$, or, $\Psi_g \cap \Psi_{model}$ when the model is used.

There are two possible outcomes when the model is used:

- $(\Psi_g \cap \Psi_{model}) \subseteq \Psi_g$, which is basically the path inside the model leading to escaping gaps;
- $(\Psi_g \cap \Psi_{model}) = \emptyset$, which means the target person path never leaves $\mathbb{V}(q_0^r)$.

Subsequently, it is enough to generalize $\Psi_g$ as the possible paths breaking the visibility constraint. We are then able to raise the following definition.

**Definition 2.2.3** (Worst-case Assumption)**.** *Under the assumption that the target person $q_t^h$ cannot pass through $\delta\mathcal{P}$ and it is initially inside the robot visibility ($q_0^h \in \mathbb{V}(q_0^r)$), any future action taken by the target person is guaranteed under visibility of the stationary guard robot $\mathbb{V}(q_0^r)$, except for the worst-case scenarios, i.e. it is leaving through escaping gaps $\mathcal{G}$.*

Definition 2.2.3 gives the following consequences:

1. It implies we can set our focus only on predicting the worst-case scenarios instead of making a very accurate model for the target movement, to keep the target under visibility of the robot. Hence, we only need to have a reliable prediction of those worst-case conditions.
2. The robot does not need to take any action unless the target is predicted to leave the visibility scope via escaping gaps. This behavior directly fulfills our purpose, i.e. to reduce the robot movement.

Now, we have already reduced the prediction space of the target person, and the rest is to predict its movement towards escaping gaps. However, carrying out a long prediction of the human future movement is error-prone. Moreover, there is no guarantee that the target person always has an intention to go through escaping gaps. Hence, we updates the target movement in an iterative fashion to obtain a reliable prediction.

The action of the robot accordingly can be made based on the above prediction of the worst-case conditions. In connection with the concept of viewpoints, the robot actions are either to go to a viewpoint which covers the possible escaping gap efficiently, or, to stay in the current robot states (e.g. by making $u_t = 0$, which is preferable) if the target is predicted will never pass through the escaping gaps.

### 2.2.5 Proposed framework for the guard robot

Using the above concepts, we re-establish the guard robot problem as follows

$$\begin{aligned}
\min \quad & \textstyle\sum u_t, \qquad t \in [0, \infty] && (2.9)\\
\text{s.t.} \quad & \{q_t^h \in \psi_g\} \in \mathbb{V}(q_t^r \in \psi_r), && (2.10)\\
& \psi_r \simeq f(u, \varepsilon) : [q_0^r, v] \mapsto \mathbb{R}^m, && (2.11)\\
& \forall \psi_g \in \Psi_g, && (2.12)\\
& \forall v \in \{q_0^r, \mathcal{V}\}, && (2.13)\\
& \{q_t^r, q_t^h\} \in \mathcal{P}. && (2.14)
\end{aligned}$$

The above formulation principally tells the robot to choose the viewpoint destination which minimizes the future control $u_t$ (eq. (2.9)) and always holds the target visibility towards the robot (eq. (2.10)). The constraint in eq. (2.10) indicates each state of the person staying on its path towards the escaping gap $\psi_g$ (see eq. (2.12)) must be visible from the robot at anytime $t$. Function $f(\cdot)$ in eq. (2.11) represents the robot kinematic model for moving to viewpoint $v$, parameterizing the robot path $\psi_r$, in order to keep track and encounter the targeted person as suggested by eq. (2.10). Equation (2.13) gives the robot a choice to stay at the current state $q_0^r$ (i.e, $u_t = 0$) when it is deemed the target will never leave the robot visibility (e.g. the environment is completely convex or the target does not move at all).

We then propose a framework for solving the guard robot problem by dividing it into two stages: off-line and on-line stages. In the off-line stage, we examine the environment to get viewpoints. We then use those viewpoints to make action plans for the robot real-time, according to the current prediction of the robot and the target states. Figure 2.4 shows our proposed framework. The forthcoming sections will describe the detail of viewpoint extraction and how the action plans are executed in response to the predicted target movement.

## 2.3 Proposed System Architecture

Running solely the viewpoint planning algorithm will bring the robot to nowhere, since it only provides “plan” for the robot. It needs other building blocks for performing a complete mission in a robotic framework. We list two important supporting blocks, namely the path planning and the person tracking algorithms. The path planning algorithm is responsible to translate the plan

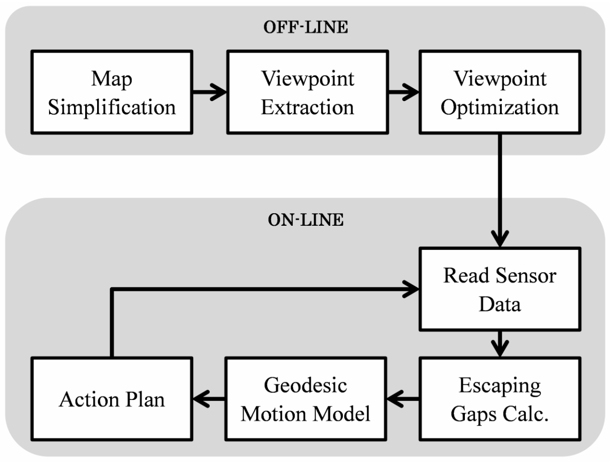


**Figure 2.4:** Block diagram of proposed viewpoint planning algorithm.

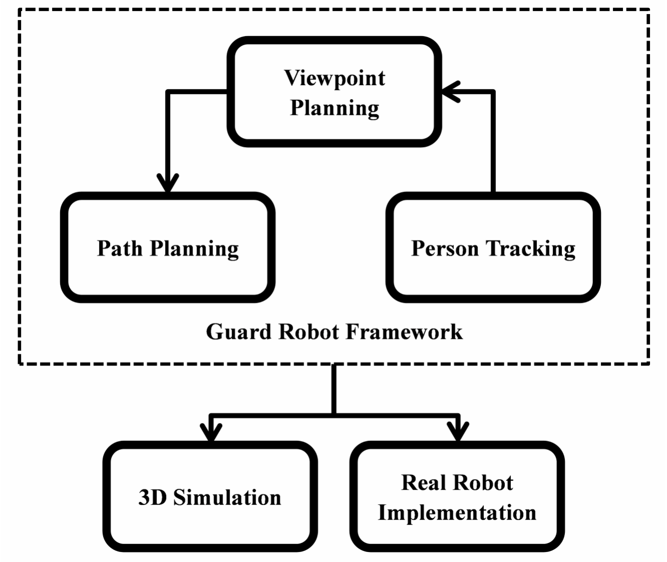


**Figure 2.5:** Block diagram of proposed architecture for the guard robot.

from the viewpoint planning into a sequence of robot motions, while the person tracking algorithm provides information of the target person states. Figure 2.5 shows our proposed architecture for performing a complete mission of the guard robot problem.

# Chapter 3

# Viewpoint Extraction via Generalized Coverage Solver

In this chapter, we describe a method for retrieving the viewpoints needed by the guard robot problem. As explained beforehand, the concept of viewpoints is a variant of the broader problem class, called coverage problem. Instead of exploiting a specialized algorithm to be used only by our guard robot, here we propose a broader algorithm, namely “Generalized Coverage Solver”, which can cover a huge class of the coverage problems, such as the *Art Gallery Problem*, *Sensor Placement Problem*, and *Robot Coverage Problem*. To conform with such wide problem class and to avoid any confusion, in this chapter we interchange the term “viewpoint” and “guard”.

## 3.1 Introduction

Suppose we want to put several surveillance cameras to see the entire building. Due to the budget limitation, we also want to reduce the number of camera as minimum as possible. The problem now becomes how many cameras should be used and where it should be placed. This question is basically the essence of the *Art Gallery Problem*.

The Art Gallery Problem, as appeared in several textbooks (*e.g.* [1], [20], and [21]), is a classical problem which inquiries the minimum number of guards which should be placed in a polygon ensuring the full coverage of the entire polygon. It is closely related to the sensor coverage and sensor placement problem, so we are not surprised for finding several works which blend both problems into one topic, such as [22], [23], and [24].

There are a lot of practical cases in the real world which rely on the Art Gallery or sensor coverage problem and its variants. The surveillance camera placement above is one example. The other related cases are how to determine the efficient sensor placement and coverage problem in a *sensor network* (*e.g.* [23] and [24]) and *multi agent deployment* for the building inspection [25]. The robots are also used, instead of the cameras or sensors, for the area coverage [26] and distinguishing landmarks [3]. There also exist a work using a mobile robot for doing a 3D mapping and scan matching based on the Art Gallery Problem ([27] and [28]). Therefore, several works consider the terrain of the polygon or environment to do the guarding tasks *(e.g.* [29], [30], and [31]).

Each problem above has a unique characteristic. For instance, the cameras can be located at any place of the building, even on the edge of the walls or the corners, depend on the camera model. Yet, there are some occasions that we prefer to put the camera, such as the omnidirectional camera, on the middle of the room's ceiling. Subsequently, the guard robot cannot be placed at the edge of the walls, instead, it should work and move inside or in the interior of the room. This placement preference is also applicable for the 3D mapping and the terrain guards, where some locations will be more important and interesting to see or to guard than the others. While those problems are usually solved for each cases (such as vertex guards [32], edge guards, and interior guards [4]), here we aim to establish a generalized framework which can be applied for every cases which includes the placement preferences.

### 3.1.1 Related work

The early result of the Art Gallery Problem was published by Chvatal [33] in which it is proclaimed that $(\frac{n}{3})$ guards are adequate for covering any polygon with $n$ vertices, which later was proved by Fisk [34]. Using Fisk's proof, Avis and Toussaint [35] then developed an $O(n\log n)$ algorithm for assigning the guard positions. These classical works assume a simple polygon without holes.

Recently, a vast effort has been promoted to deal with the Art Gallery Problem and coverage problem variants. Pinciu [2] proposed a coloring algorithm to find the connected guards in an art gallery. Gonzales-Banos *et al.* [22] presented an algorithm for a restricted version of the sensor placement problem (with a connection to the Art Gallery Problem) using a randomized approach, by applying a large set of guards and optimizing them using *hitting set* approach.

Since the optimal guarding problem is NP-Hard [36], some approximation algorithms then appeared, such as [37] and [38]. Gosh [37] provided a logarithmic approximation ratios for calculating a minimum number of vertex guards for a polygon. Then, Desphande *et al.* [38] proposed an $O(\log n)$-approximation for point-guard problem.

Later, heuristic-based approaches were then introduced. The authors in [39] adopted a greedy strategy to form a set of heuristic-based algorithm with the polygon partition methods. Bottino *et al.* [4] then created a new heuristic for the Art Gallery Problem, however the algorithm was restricted to cover only the edges of a polygon.

The most recent works made attempts for seeking an exact solution of the Art Gallery Problem via Integer Linear Programming (ILP) approach. Couto *et al.* [32] introduced an exact algorithm for the problem limited to the vertex guards. In [40], the authors utilized a finite set of *so-called* witnesses and guard candidates, and employed a linear relaxation of the primal-dual formulations iteratively to find the integer solution of the Art Gallery Problem. It was then improved in [41] by introducing the combination of Linear Programming (LP) and Difference of Convex (DC) programming. While both approaches performed good results in several instances, they failed to converge to the integer solutions in many other samples. Moreover, their approaches were basically restricted to the original issue of the Art Gallery Problem which did not take into account the guard positions.

In short, the existing algorithms for solving the Art Gallery Problem and sensor coverage above suffered from the following drawbacks:

a) They are applied to a simple or certain class of polygon;

b) They are only applicable for limited problems;

c) Lack of real applications on the real environment conditions.

### 3.1.2 Our contributions

Our objective is to develop a unified framework for solving the coverage problem in different kind of settings and applications as mentioned in the beginning of this chapter, by generalizing the original Art Gallery Problem. At the same time, we want to overcome the drawback of the existing methods. Here we propose *Generalized Coverage Solver* (*GC-Solver*) to achieve those purposes.

The GC-Solver simplifies the environment map into a polygon (it may contain holes) and exploits the topological features of the map to extract a set of guard candidates which guarantee the full coverage of the environment. Here the boundaries and the skeleton of the map are used as the topological features. Subsequently, the GC-Solver builds the visibility area of each candidate and composes them into an arrangement.

A *non-unicost Set Covering Problem* (SCP)-based algorithm is then adopted to obtain a set of suboptimal guard candidates based on the arrangement. The usage of the *non-unicost* SCP allows us to attach the weight function to each guard candidate, so that the guard choices can be adjusted to the problem preferences (*i.e.* vertex guards, interior guards, cost function-based guards, *etc.*).

The GC-Solver then continues to optimize the guard candidates using a *hybrid probabilistic*-based algorithm. By alternating the *non-unicost* SCP and the *probabilistic* optimization, the GC-Solver thus tries to acquire the optimal guards.

Main contributions of this work are three-fold. First, our approach serves a generalized and comprehensive framework for the Art Gallery Problem which binds a broad range of coverage applications. Second, we introduce the utilization of the topological features to fetch the potential guard candidates. Lastly, we also initiate combination of the *non-unicost* SCP and the *probabilistic* optimization for obtaining the optimal set of guards. To the best of our knowledge, it is the first method pursuing such unified coverage problem.

## 3.2 Generalized Coverage Problem

This section describes the framework of our approach for solving the coverage problem. The term “generalized” here is used for emphasizing our intention to develop a coverage algorithm which binds a diverse applications, in contrast to the original Art Gallery Problem.

### 3.2.1 Algorithm overview

Our proposed method consists of two large portions: extracting the location of the guard candidates, and optimizing them to obtain the optimal position of the guards. The first part involves the process of simplifying the given environment or map, from which we draws up a set of guard candidates. The topological features is then employed to guarantee the full coverage of the entire environment, by using the concept of visibility polygon.

The second part incorporates the optimization procedures of the obtained guard candidates. Here we propose the usage of a *non-unicost* Set Cover Problem alternated by a probabilistic optimization technique, to ensure the optimality of the guards in accordance with the various problem settings (*i.e.* different applications of the coverage problem). Figure 3.1 shows the pipeline

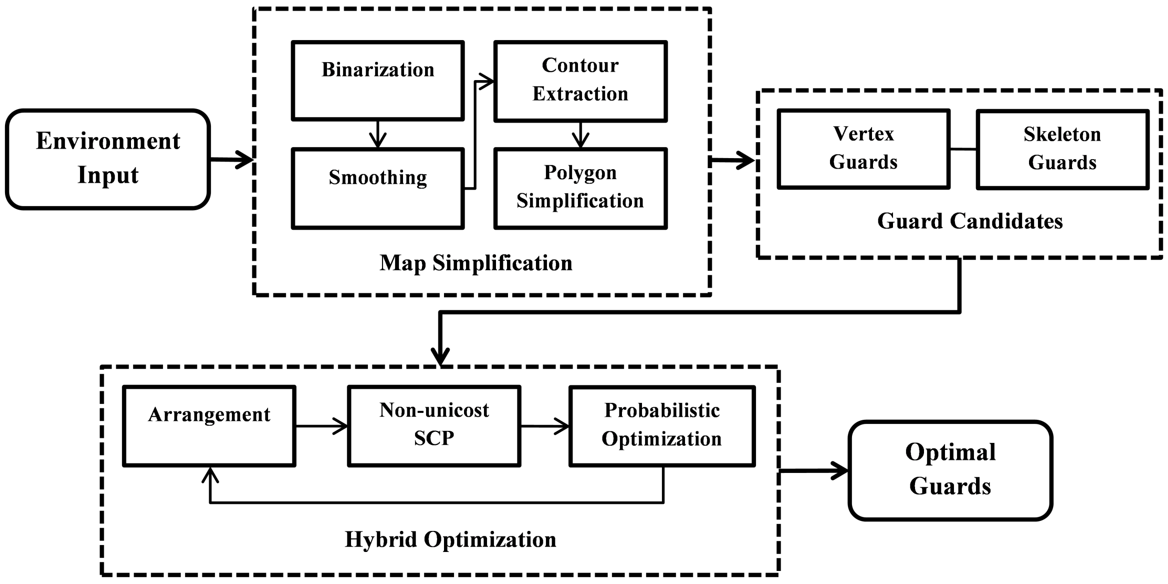


**Figure 3.1:** Block diagram of the proposed algorithm for the coverage problem.

of our proposed algorithm.

### 3.2.2 Planar polygon extraction from the environment map

We follow the procedure explained in section 2.2.1 for extracting a planar polygon representation from the environment map. Let us recall the notation $\mathcal{P}$ as the polygon, with $\delta\mathcal{P}$ and $\delta\mathcal{H}$ representing its outer and inner boundaries, and $\mathbb{V}(p)$ as the visibility polygon of a point $p \in \mathcal{P}$.

### 3.2.3 Guard candidates from the ensemble features

The term “guard” is defined as a location on which we set an entity for “seeing” or covering the area in the environment (in this case, the polygon). The physical form of the entity can be a sensor, camera, base station, or even a robot. Intuitively, a person will put the guard at the position which has a wide view, or at the location which is difficult to see. For instance, a security camera is usually placed on the top corner of a room, or, if we have an omnidirectional camera, we will placed it on the middle of ceiling which has maximum field-of-view (*e.g.* at the intersection of corridors).

For solving the coverage problem, we first extract the possible location of the guards using the human intuition as mentioned above. Here the topological features of the environment are adopted and selected as the candidates, as follows:

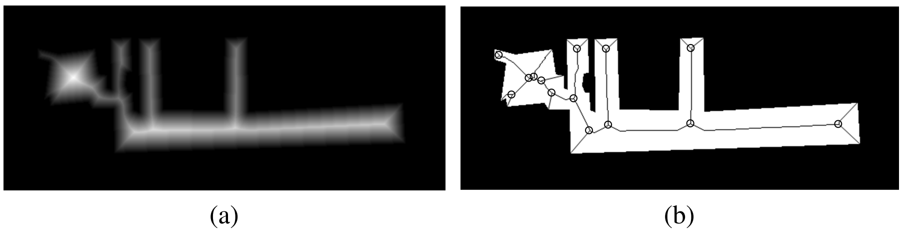


Figure 3.2: Skeleton guards extraction: (a) distance transform of the map, (b) obtaining skeleton vertices. The circles denote the skeleton guards.

#### 3.2.3.1 Vertex guards

The polygon vertices are among the important features in a polygon. Several previous researches on the Art Gallery Problem also employed these features, such as [2], [37], and [4]. In reality, a vertex of a polygon represents a corner of a room on which the sensor or camera is placed. To obtain the vertex guards, we simply take out all vertices of the polygon's boundary, including its holes (if any).

#### 3.2.3.2 Skeleton vertices as the interior guards

In the real problem setting such as a building, we can naturally determine the place in which we will get a wider view. Intuitively, a person will say that an intersection of corridors in a building grants wider view, compared with the wall or the corner of the room. Based on this reason, we deem it is necessary for us to take account the topological shape of the environment for the coverage problem. Thus, we use skeletonization technique for capturing topology of the polygon.

For obtaining the skeleton vertices, we first build a skeleton map using laplacian of distance transform technique [17] (in contrast to the straight skeleton in [42]). We construct a distance transform map $\mathcal{D}$ (Fig. 3.2a) given by

$$\mathcal{D}(p) = \begin{cases} \|p - p'\| & \text{for } p \in \mathcal{P}, p' \in \{\delta\mathcal{P}, \delta\mathcal{H}\} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where $p'$ is the nearest non-passable point to $p$.

We then apply a laplacian filter to $\mathcal{D}$, to get the skeleton map $\mathcal{K}$, denoted by

$$\mathcal{K}(p) = \frac{\partial^2 \mathcal{D}}{\partial p_x^2} + \frac{\partial^2 \mathcal{D}}{\partial p_y^2}, \tag{3.2}$$

where $p_x$ and $p_y$ respectively denote the *x-axis* and *y-axis* of the point $p$. $\mathcal{K}$ is then binarized by a threshold. The skeleton guards are then acquired by taking out all junctions and endpoints of the skeleton (Fig. 3.2b). Both types of guards (vertices and skeleton) are then coalesced, producing a set of guard candidates $\mathcal{V}$.

#### 3.2.3.3 Coverage guarantee

Before an optimization process is carried out for the set of guard candidates above, we want to show that the mixture of the guard candidates itself have already been able to cover the entire polygon, even it is not the optimal one. In other words, this property is giving a clue to the optimization part that it should always return a solution (*i.e.* full coverage of the polygon). The following proposition is used for exhibiting the coverage guarantee of the guard candidates.

**Proposition 3.2.1** (Coverage Guarantee: Vertices)**.** *All vertices (including the holes, if any) of a planar polygon are always adequate for covering the entire polygon.*

**Proof** (Coverage Guarantee: Vertices)**.** *One of possible ways to prove this proposition is by using the set theory over the established theorems. For the polygon with holes, O'Rourke [1] stated* $\frac{n+2h}{3}$ ***vertex guards*** *are sufficient for covering a polygon with n vertices and h holes (see theorem 5.1 of [1]). Let* $\mathcal{G}_{vt}$ *be a set of all vertices (including the holes) of a polygon with cardinality n, and* $\mathcal{V}_{rk}$ *be the set of covering vertices in O'Rourke theorem with cardinality* $\frac{n+2h}{3}$*. We begin with proving the set cardinality,* $\frac{n+2h}{3} \leq n$*. The problem can be written as*

$$\begin{aligned} \frac{n+2h}{3} \leq n \Longleftrightarrow \frac{2h}{3} \leq \frac{2n}{3}, \\ \Longleftrightarrow h \leq n. \end{aligned} \tag{3.3}$$

- *If* $h = 0$*, since* $n \geq 3$ *(a polygon is composed by at least three vertices), then* $h \leq n$ *is held.*
- *If* $h > 0$*, since* $n > 3h > h$ *(a hole has at least three vertices), then eq. (3.3) is held.*

*Thus the inequality holds for any number of holes. Here we have proved that the statement* $\frac{n+2h}{3} \leq n$ *is true. As the consequence,* $\mathcal{V}_{rk} \subset \mathcal{V}_{vt}$*. It suggests there exists a set of element in* $\mathcal{V}_{vt}$ *which cover the entire polygon. Notice that for* $h = 0$*, the problem becomes the Chvatal theorem [33].*

Consequently, the guard candidates $\mathcal{V}$ which is a combination of the vertices and skeleton guards, retains the same coverage guarantee.

**Proposition 3.2.2** (Coverage Guarantee: Vertices and Skeletons)**.** *The coverage of the combination of guard candidates $\mathcal{V}$ over the polygon $\mathcal{P}$ is always guaranteed.*

**Proof** (Coverage Guarantee: Vertices and Skeletons)**.** *Let $\mathcal{V}_{vt}$ and $\mathcal{V}_{sk}$ respectively be the vertex and skeleton guards. Subsequently, the guard combination $\mathcal{V}$ can be denoted as $\mathcal{V} = \mathcal{V}_{vt} \bigcup \mathcal{V}_{sk}$. From proposition 3.2.1, we know that $\forall v_{vt} \in \mathcal{V}_{vt}$ holds the coverage guarantee. Using the set theory, since $\mathcal{V}_{vt} \subset \mathcal{V}$, it implies $\forall v_{vt} \in \mathcal{V}$. Thus, $\mathcal{V}$ retains the same coverage guarantee as $\mathcal{V}_{vt}$.*

## 3.3 Hybrid optimization for coverage problem

After the guard candidates are determined, the next step is to optimize the number of guards for covering the entire area of $\mathcal{P}$. Even for the original issue of the Art Gallery Problem which does not consider the placement of the guards; it is already an NP-Hard problem [36]. Here we try to evade the problem by applying a *hybrid optimization* approach. First, we transform the coverage problem into a *Set Covering Problem*, a family of *Integer Linear Programming*. A heuristically *probabilistic optimization* is then administered to obtain the optimal guard positions.

### 3.3.1 Arrangement of the guard's visibility

Before we bring the coverage problem into a *Set Covering Problem*, we need to understand its prerequisite of the transformation, that is the *polygon arrangement*. We borrow the definition of the *arrangement* from [43]. Given a finite set of guard candidates $\mathcal{V}$, from which we acquire a set of visibility polygon $\mathbb{V}(\mathcal{V})$, the *arrangement* $\mathcal{A}(\mathcal{V})$ is then defined as the subdivision of the plane created by the intersection of all vertices of $\mathbb{V}(\mathcal{V})$, such that $\mathcal{A}(\mathcal{V}) : \mathbb{V}(\mathcal{V}) \mapsto \mathcal{F}_c$. Each subdivision area of $\mathcal{A}(\mathcal{V})$ is called a *face*, denoted by $f_c \in \mathcal{F}_c$. Figure 3.3 shows the definition of the arrangement and *face*.

Reciprocally, we can construct the visibility polygon of a guard $\mathbb{V}(v_1)$ as a set of *faces* $f_c$

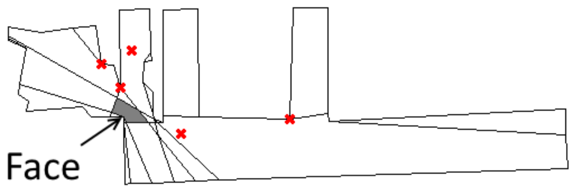


**Figure 3.3:** Arrangement of a set of guards. The red crosses represent the guards. All edges are the result of combining the visibility polygon of all guards. The gray area is one of the *faces* created by the arrangement.

"seen" by the guard $v_1 \in \mathcal{V}$, such that

$$\mathbb{V}(v_1) \approx \{ \bigcup_{\forall f_c \in \mathcal{F}_{c1}} f_c | \mathcal{F}_{c1} \subset \mathcal{F}_c \}, \tag{3.4}$$

where

$$\mathcal{F}_{c1} = \{\forall f_c \in \mathbb{V}(v_1)\}. \tag{3.5}$$

It then raises a definition of the polygonal coverage, as follows

**Definition 3.3.1** (Polygonal Coverage)**.** *The coverage of a polygon $\mathcal{P}$ by a finite set of guards $\mathcal{G}$ is guaranteed under circumstances*

$$\mathcal{P} = \bigcup_{\forall f_c \in \mathcal{F}_c} f_c, \tag{3.6}$$

*where $\mathcal{F}_c$ are composed by the arrangement of* $\mathbb{V}(\mathcal{V})$.

In other words, all *faces* will always cover the polygon as long as its composing set of guard $\mathcal{V}$ also has a full coverage. Now we aim for the optimal number of $\mathcal{V}$ which satisfies definition (3.3.1).

### 3.3.2 The Art Gallery Problem as the Set Covering Problem

Equation (3.4) and (3.6) give two consequences:

a) There may exist a *face* "seen" by more than one guard, or geometrically

$$\mathbb{V}(v_1) \bigcap \mathbb{V}(v_2) \approx \mathcal{F}_{c1} \bigcap \mathcal{F}_{c2} \neq \emptyset, \quad \{v_1, v_2\} \in \mathcal{V}. \tag{3.7}$$

It simply means some guards may cover the same area.

b) Summation of all *faces* in $\mathcal{F}_c$ should cover the entire polygon $\mathcal{P}$, in order to comply with definition (3.3.1).

These consequences lead to the problem of assigning the smallest set of guards $\mathcal{V}$ such that its summation of *faces* in the $\mathcal{A}(\mathcal{V})$ satisfies equation (3.6). Accordingly, we are able to bring the coverage problem into an *Integer Linear Programming* formulation, more precisely, a *Set Covering Problem*.

Given an $M \times N$ matrix $\mathbf{A}$, the *Set Covering Problem* (SCP) is defined as a problem of discovering a subset of the columns of $\mathbf{A}$ which covers all rows at a minimum cost [44]. Using the SCP formulation, the original Art Gallery Problem can be defined as follows

$$\text{Minimize} \quad \sum_{n=1}^{N} v_n, \quad \text{for } v_n \in \mathcal{V} \tag{3.8}$$

$$\text{s.t.} \quad \sum_{n=1}^{N} a_{mn} v_n \geq 1, \quad \{m = 1, \ldots, M\} \tag{3.9}$$

$$v_n \in \{0,1\}, \quad \{n = 1, \ldots, N\} \tag{3.10}$$

$$a_{mn} \in \{0,1\}, \quad a_{mn} \in \mathbf{A}. \tag{3.11}$$

Here, $M$ and $N$ respectively denote the number of *faces* of the arrangement and that of the guard candidates.

From the above minimization, the SCP formulation for the Art Gallery Problem is obtained by making the guards to be the sets used for covering and imposing the *faces* of the arrangement as the elements to be covered. The guard placement inside the polygon is modeled by testing it using a binary condition ($v_n = 1$ if the guard is included into the set). Henceforth, a face $row(a_{mn})$ is set to 1 if it is seen by the guard $v_n$ (see eq. (3.11)). Inequality of eq. (3.9) assures that a certain row must be covered by at least one column, or in other words, a *face* should be covered by at least one guard. It will guarantee that the entire polygon $\mathcal{P}$ is fully covered.

## 3.3.3 Non-unicost Set Covering Problem

According to eq. (3.8), each guard candidate is treated the same, *i.e.* it does not matter where the selected guards are chosen from, whether it lies at the interior or the vertices of the polygon, as long as it can cover the entire at the most minimum number. This is exactly what the original Art Gallery Problem aims for.

Such formulation of eq. (3.8) is often called *unicost* Set Covering Problem, indicating each guard is eligible to be chosen with the same cost. Nevertheless, in many cases of the coverage problem, some guards may become more alluring than the others. An interesting instance will be a network provider should calculate the different land cost for placing each Base Tranceiver Station (BTS), while still covers the whole area.

In order to achieve a broad range of the coverage application, here we propose the usage of *non-unicost* Set Covering Problem. It allows us to assign different cost for each guard. Subsequently, we modify eq. (3.8) as follows

$$\text{Minimize} \qquad \sum^{N} c(v)v, \quad \text{for } v \in \mathcal{V}. \tag{3.12}$$

Equation (3.12) introduces a cost function $c(v)$, from which we can manage how important a guard will be. The formulation in eq. (3.8) becomes the special case of eq. (3.12) where all costs are equal. We will describe the cost function $c(v)$ further in the experimental section along with the examples, including how it will affect and alter the coverage problem.

### 3.3.4 Hybrid probabilistic guard optimization

The output of the *non-unicost* SCP is self-explanatory, *i.e.* it attains the optimal combination among the input set $\mathcal{V}$. However, it does not imply that the result is also the optimal one for the coverage of a polygon. This matter arises since there is no guarantee whether the optimal guards are already in the input set $\mathcal{V}$ or not; we only ensure the coverage as suggested by proposition 3.2.2.

We then come up with a strategy to cope with it. Essentially, we break down the optimization process into two parts: *initial* and *iterative* optimization. In the initial optimization, we reduce the guard candidates composed in 3.2.3 using the steps mentioned in section 3.3.1 to 3.3.3, yielding an *initial upper bound* of the guards. Afterward, we make attempt to reduce the guards further by using an iterative optimization. Here we examine the area which has mutual guards coverage using a probabilistic randomized search, alternated by the *non-unicost* SCP. This technique is expected to cut down the mutual guards coverage by an alternative point guard.

#### 3.3.4.1 Initial optimization

Let $\mathcal{V}_0$ be the initial set of guard candidates obtained in section 3.2.3. We first optimize the guard candidates by

a) Constructing the arrangement $\mathcal{A}(\mathcal{V}_0)$;

b) Building the cost function of the guards $c(\mathcal{V}_0)$;

c) Solving the *non-unicost* SCP to get a set of *pre-optimized* guards $\mathcal{V}_{opt}$. The cardinality of $\mathcal{V}_{opt}$ is then called *initial upper bound*[1].

The *initial upper bound* $|\mathcal{V}_{opt}|$ of the possible optimal guards from the set is acquired using the above steps. To make it compact, the initial upper bound $|\mathcal{V}_{opt}|$ is now denoted by $U$. Since the algorithm is continued by an iterative optimization, the notation $\mathcal{V}_{opt}$ will be constantly used to show the set of the optimal guards found so far.

#### 3.3.4.2 Iterative optimization

Our basic idea is to examine each optimized guard for further possible reduction, by analyzing the *faces* of its arrangement. Given a *face* $f_c \subset \mathcal{A}(\mathcal{V}_{opt})$ "seen" by a set of guards $\mathcal{V}_{par} \subset \mathcal{V}_{opt}$, the following definition is then applicable.

**Definition 3.3.2** (Faces Visibility). *For a face $f_c$ seen by all $v \in \mathcal{V}_{par}$, it implies that all point guard $v \in \mathcal{V}_{par}$ are inside the visibility polygon of any point $p \in f_c$ ,*

$$\left\{\forall v \in \mathcal{V}_{par} | f_c \subset \mathbb{V}(v)\right\} \implies \left\{\forall p \in f_c | (\forall v \in \mathcal{V}_{par}) \in \mathbb{V}(p)\right\}. \tag{3.13}$$

The above definition is an extension of definition (2.2.1), from which we want to show the possibility of "seeing" a set of guards by a point inside a *face*. It does not necessarily imply that all area of $\mathbb{V}(\mathcal{V}_{par})$ will be covered by a point $p \in f_c$, yet it exhibits the likelihood of "replacing" several guards to one point. Hence, definition (3.3.2) becomes the stepping stone for our proposed approach for finding the optimal guards.

[1]This naming convention suggests an attempt to decrease the cardinality, lower than this upper bound

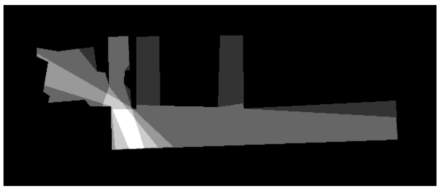


**Figure 3.4:** Face cost map of the arrangement in Fig. 3.3. Brighter *face* means it is seen by more guards.

By making use of the definition above, here we build a *face cost map* which shows how the *faces* are parameterized by the guards, *i.e.* a *face* will have a higher value when it is covered by more number of guards. Let $h(f_c)$ be the *face cost map function* defined by

$$h(f_c) = |\mathcal{V}_{par}|, \quad \forall f_c \in \mathcal{F}_c. \tag{3.14}$$

Subsequently, we use $h(f_c)$ as a distribution function for sampling a set of *auxiliary guard candidates*. We currently sample the auxiliary guard candidates $\mathcal{V}_{sampling}$ from $\mathcal{F}_c$ using $h(f_c)$, as much as two times of $|\mathcal{V}_{opt}|$. It is expected that we will obtain more samples on the *face* covered by more guards. Figure 3.4 shows the definition of the *face cost map*.

Both $\mathcal{V}_{opt}$ and $\mathcal{V}_{sampling}$ are concatenated forming a combined set of guards, from which the *non-unicost* SCP is then solved using the same steps as the one in the *initial optimization*, yielding a new set of optimized guards $\mathcal{V}_r$. The above processes of constructing the face cost map, sampling the *auxiliary guard* candidates, and solving the *non-unicost* SCP are accordingly alternated for the iterative optimization procedures.

A heuristic approach is then exerted for examining the optimality of the guards produced under current iteration. In principal, we wish the reduction of the cardinality (guard number), or a better aggregated guard cost $c(v_n)$ as demanded by eq. (3.12). We commence from the *non-unicost* SCP results (*i.e.* $\mathcal{V}_r$). There are three possible outputs which corresponds to the cardinality of optimized guards $\mathcal{V}_r$ and its leverage to the next iteration :

a) $|\mathcal{V}_r| < U$. This is what we expect for, subsequently the next iteration will start using this new guard set $\mathcal{V}_r$ and the upper bound $U$ is lowered to $|\mathcal{V}_r|$.

b) $|\mathcal{V}_r| > U$. It means the minimization in eq. (3.12) generates a set of guards which has lower aggregated cost, despite of having a higher cardinality. Please note that this type of output unlikely happens when the uniform cost function is used (*e.g.* the original AGP), considering proposition (3.3.1) which will be presented later.

c) $|\mathcal{V}_r| = U$. It means $\mathcal{V}_r$ has a better aggregated cost with the same cardinality. We will particularly discuss this type of output later.

We empirically found that after several iterations, the last type of the output of the *non-unicost* SCP above appears in most of the cases. It suggests the cardinality of the guards for the coverage problem is gradually converged. Thus, a *Hausdorff metric* is adopted to ascertain the stopping condition of the iterative optimization process, defined as

$$\begin{aligned} D_{hd}(\mathcal{V}_r, \mathcal{V}_{opt}) &= \max\left\{d(\mathcal{V}_r, \mathcal{V}_{opt}), d(\mathcal{V}_{opt}, \mathcal{V}_r)\right\}, \\ &\text{where} \\ d(\mathcal{V}_r, \mathcal{V}_{opt}) &= \max_{v_r \in \mathcal{V}_r} \min_{v_{opt} \in \mathcal{V}_{opt}} \|v_r - v_{opt}\|. \end{aligned} \tag{3.15}$$

Here, equation (3.15) has a physical meaning that is when the algorithm converges, there should not be much change on the position of the optimized guards *iteration-by-iteration*.

For the last two types of the output of the *non-unicost* SCP above, we merge the $\mathcal{V}_{opt}$ and $\mathcal{V}_r$ to be used in the next iteration. By this strategy, we basically feed the SCP solver all viable set of candidates found so far (*i.e.* has the same cardinality or cost) and let it discovers the best one as the solution. It is expected to avoid the alternating result of the optimal guards[2] and speed up the optimization process.

Using this merging technique, we evoke the following proposition, to prove our statement in the second type of the output above.

**Proposition 3.3.1** (Cardinality of Subsequent Optimal Guards)**.** *For uniform cost function, the cardinality of subsequent optimal guards $|\mathcal{V}_r|$ in the iterative optimization should not exceed the initial upper bound $U$.*

**Proof** (Cardinality of Subsequent Optimal Guards)**.** *(Proof by Contradiction) Assume $|\mathcal{V}_r| > U$ is true. For the iteration $i = 0$ (initial optimization), $\mathcal{V}_r$ is basically equal to $\mathcal{V}_{opt}$, so that $|\mathcal{V}_r|$ is equal to $U$. In the next optimization process, yielded $\mathcal{V}_r$ will be merged with $\mathcal{V}_{opt}$ and the algorithm uses $\mathcal{V}_{opt} \bigcup \mathcal{V}_{sampling}$ as the guard candidates $\mathcal{V}$, which is then utilized for solving the SCP (eq. (3.12)). It means $\mathcal{V}_r \subset \mathcal{V}$. Since eq. (3.12) is a minimization problem with uniform $c(v)$ and the previous $\mathcal{V}_r$ is in the set $\mathcal{V}$, the possible maximum cardinality must be $U$, by means of the Set Cover Problem. However, it contradicts the initial assumption. Therefore we have to conclude that $|\mathcal{V}_r| \leq U$ for the subsequent iteration.*

[2]A condition where two sets of the optimal result show up alternately, which may create an infinite loop.

Someone may wonder that the metric in eq. (3.15) only considers the norm between two sets, regardless of the cost function $c(g)$. This matter is clarified by following proposition

**Proposition 3.3.2** (Convergence of Iterative Optimization)**.** *Under all conditions, the iterative optimization in the probabilistic search always produces*

$$\sum_{v_r \in \mathcal{V}_r} c(v_r) \leq \sum_{v_r^- \in \mathcal{V}_r^-} c(v_r^-), \tag{3.16}$$

*where $\mathcal{V}_r^-$ is the optimal solution in the previous iteration.*

**Proof** (Convergence of Iterative Optimization)**.** *(Proof by Contradiction) The proof construction is basically the same with proposition 3.3.1. Assume $\sum_{v_r \in \mathcal{V}_r} c(v_r) > \sum_{v_r^- \in \mathcal{V}_r^-} c(v_r^-)$ is true. In the iterative optimization, $\mathcal{V}_r^-$ is in the set of $\mathcal{V}$ used for minimization in eq. (3.12), since $\mathcal{V}_r^-$ is merged with $\mathcal{V}_{opt}$ and $\mathcal{V} = \mathcal{V}_{opt} \bigcup \mathcal{V}_{sampling}$. Again, it suggests the minimization result has the total objective value which is equal to at most $\sum_{v_r^- \in \mathcal{V}_r^-} c(v_r^-)$. Yet, it contradicts the initial assumption. Hence we draw a conclusion that the proposition is true.*

The above proposition ensures that our proposed *hybrid probabilistic guard optimization* is probabilistically converged towards the optimal solution. We then call off the iterative process when $D_{hd}(\mathcal{V}_r, \mathcal{V}_{opt})$ is below a threshold.

The iterative optimization is summarized as follows

a) Initialize the cardinality of the given guard candidates;

b) Establish the *face cost map*;

c) Sample a set of auxiliary guard candidates using the *face cost map*;

d) Solve the *non-unicost* SCP;

e) Check the convergence using the *Hausdorff metric*, and repeat.

The complete process of our proposed optimization is shown by Algorithm (1).

**Algorithm 1** Iterative Guard Optimization

```
Require:
 1: $\mathcal{V}_0$: initial set of guard candidates
 2: $|\mathcal{V}_0|$: cardinality of $\mathcal{V}_0$
 3:
Ensure:
 4: $\mathcal{V}_{opt}$: optimized guards
 5:
 6: procedure ITERGUARDOPT($\mathcal{V}_0$)
 7:     // Initial optimization
 8:     $\mathcal{V}_{opt} \leftarrow$ Init($\mathcal{V}_0$)
 9:     $U \leftarrow |\mathcal{V}_{opt}|$                                ▷ initial upper bound
10:     // The real loop starts here
11:     faceCostMap($\mathcal{V}_{opt}$)                                    ▷ eq. (3.14)
12:     $\mathcal{V}_{sampling} \leftarrow$ sample($\mathcal{V}_{opt}, 2|\mathcal{V}_{opt}|$)
13:     $\mathcal{V}_r \leftarrow$ solveSCP($\mathcal{V}_{opt} \bigcup \mathcal{V}_{sampling}$)                 ▷ eq. (3.12)
14:     if $|\mathcal{V}_r| > U$ then                                  ▷ proposition (3.3.1)
15:         $\mathcal{V}_{opt} = \mathcal{V}_{opt} \bigcup \mathcal{V}_r$
16:         go to 12
17:     else if $|\mathcal{V}_r| < U$ then
18:         $\mathcal{V}_{opt} = \mathcal{V}_r$
19:         $U \leftarrow |\mathcal{V}_{opt}|$
20:         go to 11
21:     else
22:         if $D_{hd}(\mathcal{V}_r, \mathcal{V}_{opt}) >$ threshold then                 ▷ Hausdorff
23:             $\mathcal{V}_{opt} = \mathcal{V}_{opt} \bigcup \mathcal{V}_r$
24:             go to 11
25:         else
26:             $\mathcal{V}_{opt} = \mathcal{V}_r$
27:             break
28:         end if
29:     end if
30:     return $\mathcal{V}_{opt}$
31: end procedure
```

### 3.3.5 Remarks

The reader may notice that the result of the *non-unicost* SCP in each iteration is examined by its cardinality. It then raises a question, there exists possibilities to have the $\mathcal{V}_r$ which has more guards than previous $\mathcal{V}_{opt}$, but with a smaller cost. In the real cases, the cost of a guard is often not so cheap (*e.g.* sophisticated camera, robot, *etc.*). Therefore, we consider minimizing the number of guards as our priority.

Another notable thing is the size of $\mathcal{V}_{opt}$ will grow due to the merging technique $\mathcal{V}_{opt} \bigcup \mathcal{V}_r$. Still, we have never empirically underwent any “bloated number of guard" problem, since the optimization process is done on a bounded area (polygon interior) with the *Hausdorff metric*. We can expect the distance between the optimal set on the current iteration and $\mathcal{V}_{opt}$ will gradually decrease.

## 3.4 Experiments and results

The implementation of the GC-Solver is done on a Windows PC (i7 2.4 GHz, 16 GB RAM) using C++ programming language. We extensively utilize the Computational Geometry Algorithms Library (CGAL) [45] for performing the visibility calculation, the arrangement building, and some other geometric computations. To solve the *non-unicost* Set Covering Problem, we use both open source (GLPK [46] and SCIP [47]) and commercial[3] (CPLEX [48] and Gurobi [49]) solvers.

### 3.4.1 Experimental settings

We want to evaluate the effectiveness and generality of our proposed algorithm. For accomplishing that goal, we prepare six different environments as shown by Fig. 3.5, grouped into:

a) Artificial 2D/3D maps without and with holes (Fig. 3.5a, 3.5b, 3.5d, and 3.5e);

b) A *star-shaped* synthetic polygon (Fig. 3.5c, which is also used in [22]);

c) A complex real building at our university (Fig. 3.5f).

[3]We use the academic version of CPLEX and Gurobi.

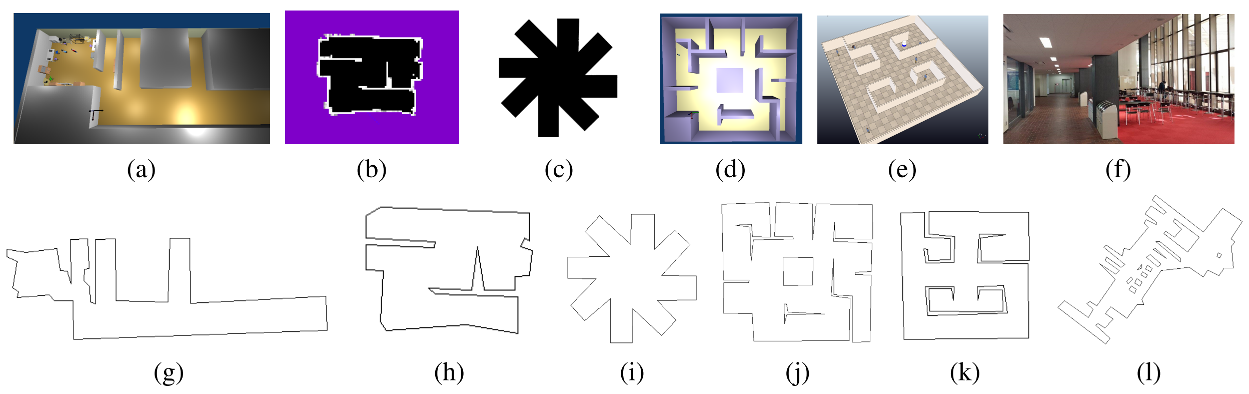


**Figure 3.5:** Environment map used for the coverage problem: (top sequences) the real environment map used as input, (bottom sequences) the simplified polygon of its respective map. The map-polygon pairs are a-g, b-h, c-i, d-j, e-k, and f-l.

The environments represent distinct types of the coverage problem. The environment in Fig. 3.5a, 3.5b, 3.5d, and 3.5e elucidate the ability of the proposed method to handle the coverage of various classes of polygon, as well as how to transform the environment itself into a polygon. To be more specific, Fig. 3.5a and 3.5b exemplify the coverage problem in the environment without holes, while Fig. 3.5d and 3.5e show the opposite one.

The star-shaped polygon (Fig. 3.5c) is the special case of the coverage, from which we want to exhibit how far the guard optimization can be carried out by our algorithm. Intuitively, a guard located exactly in the middle of the polygon should cover the entire environment.

Lastly, an attempt to solve the coverage of a real and complex building is demonstrated, showing the feasibility of our algorithm to be directly adopted in the real environment. Here we use a hall room inside our university (see Fig. 3.5f).

Table 3.1: Polygon complexity of the environment

| Environment | Map Name | # vertices | # holes |
|---|---|---|---|
| Fig. 3.5a | A | 32 | 0 |
| Fig. 3.5b | B | 26 | 0 |
| Fig. 3.5c | C | 24 | 0 |
| Fig. 3.5d | D | 51 | 2 |
| Fig. 3.5e | E | 38 | 1 |
| Fig. 3.5f | F | 102 | 11 |

Except for the star-shaped synthetic polygon, all environments are processed through the map simplification, explained in section 2.2.1, yielding the simplified polygon shown in the bottom part of Fig. 3.5. Table 3.1 shows the polygon complexity of each environment, where it varies from 24 to 102 vertices.

We first check out the initialization process of the GC-Solver. This stage is same for all classes/implementations of the coverage problem, before entering the iterative optimization. It consists of the map simplification process, determining the guard candidates, calculating the visibility polygon, and building the arrangement and *faces*. Table 3.2 indicates the numerical value of the initialization process of the GC-Solver for each environment.

Table 3.2: GC-Solver initialization for each environment

| Map Name | # Guard Candidates | # Arrangement Vertices | # Faces | Calcuation Time (seconds) |
|---|---|---|---|---|
| A | 62 | 1802 | 1773 | 1.383 |
| B | 50 | 972 | 947 | 0.762 |
| C | 46 | 992 | 969 | 0.843 |
| D | 104 | 5959 | 5906 | 3.058 |
| E | 76 | 913 | 876 | 1.133 |
| F | 224 | 63200 | 63093 | 29.760 |

Generality of the proposed algorithm is then confirmed by experiments in the following sections, which represent different kind of the coverage problems. We want to show that it can be achieved by simply changing the cost function $c(v)$.

## 3.4.2 Art Gallery Problem

In this experiment, we evaluate the capability of our proposed algorithm to solve the original issue of the Art Gallery Problem. The goal is obviously to minimize the number of guards without bothered by any guard preference and placement. It can be realized by making the cost function $c(v)$ in eq. (3.12) to be uniform.

Here we analyze the result of our approach applied into all environments mentioned in section 3.4.1. First, we examine the guards quality generated by our algorithm. Table 3.3 represents the optimal guards produced by the initial and iterative optimization (as have been explained in section

3.3.4.1 and 3.3.4.2), and its computational time respectively. The computational time shown in the table refers to the method using SCIP [47] as the *non-unicost* SCP solver, which performs the best among all solvers. Further analysis about the SCP solvers will be clarified later. Thus, the optimal guard positions can be qualitatively perceived in Fig. 3.6 and 3.8.

Table 3.3: Number of optimal guards for the Art Gallery Problem and its computational time.

| Map Name | # Guard Candidates | # Optimal Guards | | | |
|---|---|---|---|---|---|
| | | Initial Optim. | Time (s) | Iterative Optim. | Time (s) |
| A | 62 | 5 | 1.442 | 4 | 15.166 |
| B | 50 | 4 | 0.792 | 3 | 6.451 |
| C | 46 | 1 | 0.883 | 1 | 0.883 |
| D | 104 | 8 | 3.290 | 8 | 18.308 |
| E | 76 | 7 | 1.163 | 7 | 10.213 |
| F | 224 | 11 | 40.102 | 11 | 206.182 |

From table 3.3, we observe that the initial optimization reduces the number of guards significantly with a fast computational time. It can even be remarked as “near optimal” compared with the final results. It clarifies the merit of the guard candidate features which assists the GC-Solver to converge quickly.

The iterative optimization also plays a good role. In map A and B, the guards are further optimized to obtain the optimal solution. Figure 3.6a *vs* 3.8a and 3.6b *vs* 3.8b show the *head-to-head* comparison of the initial and iterative optimization. The other maps yields the same cardinality even after go through the iterative optimization process. An example of the evolution of the guard positions during this iterative optimization is shown by Fig. 3.7. Here we can make a suggestion to the user which concerns with the computational speed and willing to get considerably “good" cardinality, employing the result from the initial optimization of the GC-Solver has already served the purpose.

Another interesting result is the optimal guard of the star-shaped polygon (map C, Fig. 3.8c). This optimal solution is located at one of the skeleton vertices of the polygon. Even a slight deviation from this optimal position will not cover the entire polygon. Once again, it shows the benefit of using the skeleton guard candidates in our algorithm. Of course, here the iterative optimization is not needed since it is impossible to further improve the optimal solution (cardinality $= 1$).

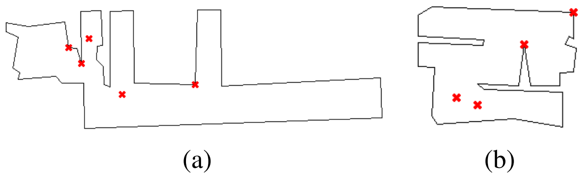


**Figure 3.6:** Initial optimization result of the Art Gallery Problem. The guard positions are marked by the red crosses.

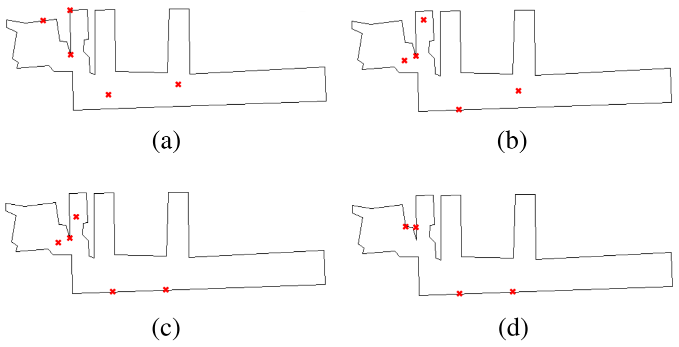


**Figure 3.7:** Evolution of the guards during the iterative optimization process for map A (From left to right).

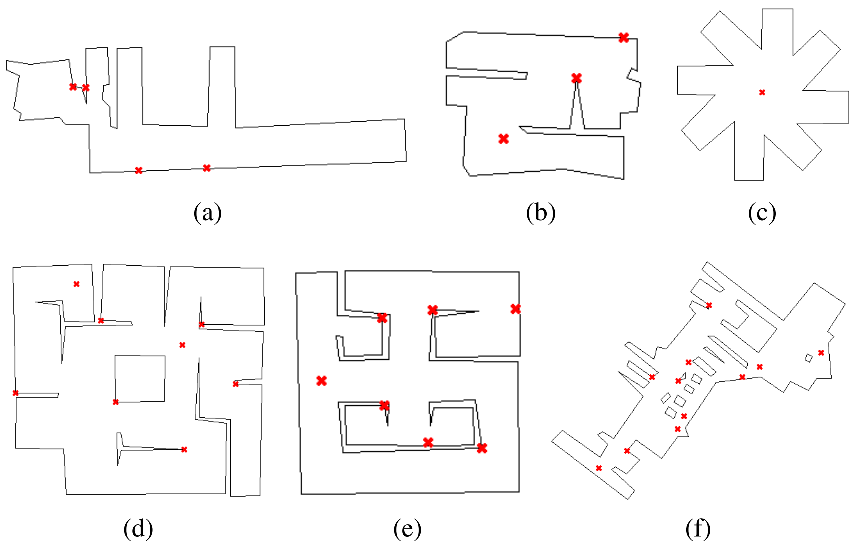


**Figure 3.8:** Optimal guards for the original Art Gallery Problem. The guard positions are marked by the red crosses.

### 3.4.3 Coverage problem with guard preferences

In this section, we investigate the performance of our algorithm when the guards are restricted based on the user preferences, for instance, it can only be placed at the vertices. It realizes the problem mentioned in the following example.

**Example – Vertex only guards :** This type of problem resembles most of the existing Art Gallery Problem, such as [33], [35], and [37]. Here we need to put the guard exactly on the vertex of a polygon. Following cost function is then used.

$$c(v) = \begin{cases} 1, & \text{for } v \in \text{ vertices,} \\ \infty, & \text{otherwise.} \end{cases} \tag{3.17}$$

Actually, this problem can be simply achieved by removing all guard candidates which are not located at the vertices of the polygon. In our case however, we want to exhibit how this problem can be solved by simply modifying the cost function, from which shows the generality of the GC-Solver. Figure 3.9 and table 3.4 demonstrate its optimization results.

Table 3.4: Optimal guards for the coverage problem with vertex only guards.

| Map Name | # optimal guards | | | |
|---|---|---|---|---|
| | Initial Optimization | Time (s) | Iterative Optimization | Time (s) |
| A | 5 | 1.520 | 5 | 11.600 |
| B | 4 | 0.703 | 4 | 5.139 |
| C | 3 | 0.770 | 3 | 6.770 |
| D | 8 | 3.590 | 8 | 11.891 |
| E | 7 | 1.025 | 7 | 7.780 |
| F | 13 | 35.400 | 13 | 100.632 |

Unlike the previous problem, according to table 3.4 there is no further optimization on the guard cardinality between the initial and iterative process. The reason is the infinity cost for the non-vertex guards gives the same effect as removing them from the candidacy. It causes the guard candidate choices become limited to a set of vertices, so that the iterative optimization cannot make

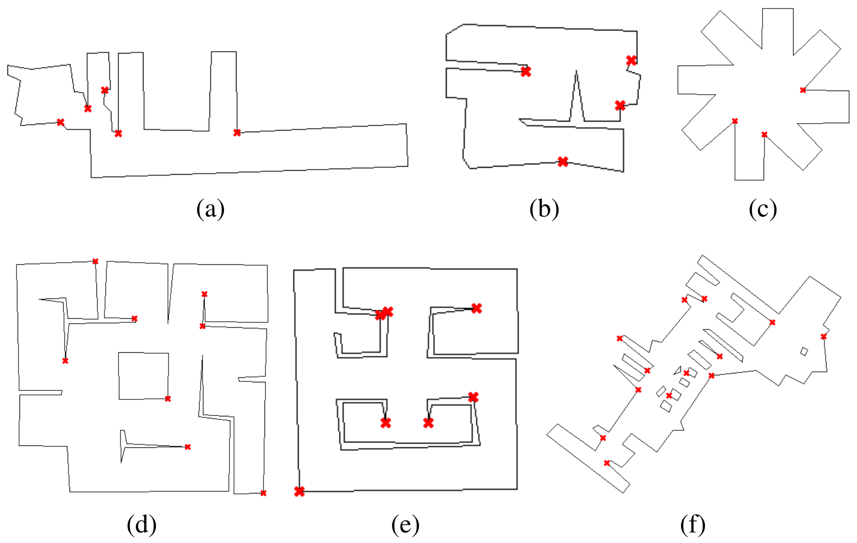


**Figure 3.9:** Optimal guards for the vertex coverage problem. The guard positions are marked by the red crosses.

any improvement.

We also notice that the guard cardinality is higher than the one in the Art Gallery Problem. It is because we lost some information from the non-vertex guards which is restricted by the cost function. It also signifies how the topological features play a great role to render an optimal solution for the coverage problem. An obvious example is the result of map C (Fig. 3.9c) where now it needs three vertex guards to cover the entire polygon, compared with one guard in the previous problem.

### 3.4.4 Coverage problem with arbitrary cost function

As the continuation of the above section, we now present the realization of the coverage problem with the arbitrary cost function, which becomes the main concern of the GC-Solver. We set up a case using a set of mobile robot formations to cover the environment, as follows.

**Example – Robot guards :** Here we need to consider the physical limitation of the mobile robot to do the coverage tasks. One example of this limitation is that the robot is supposed to be located not too close with the walls, to avoid a collision. We then accommodate this problem using a distance function as follows

$$c(v) = e^{-\|v-\delta\mathcal{P}\|}, \tag{3.18}$$

where $\delta\mathcal{P}$ represents the boundary of the polygon (in this case, the walls). Equation (3.18) tells us that we prefer to locate the robot far from the walls.

Table 3.5 and Fig. 3.10 show the optimization results of the coverage problem using the cost function in eq. (3.18). We can observe that there is no guard which lies at the vertices nor the boundary of the polygon, as suggested by the usage of the distance function. Most of the guards are located at the skeleton or the middle area of the polygon. Physically, it means the guard robots take place in the middle of room or the intersection of corridors, which is favorable according to the collision properties mentioned in above example.

One alluring result is that this coverage problem produces relatively higher cardinality of the optimal guards than the one in the Art Gallery Problem. This matter is plausible since the cost function in eq. (3.18) leads to a lower total cost as pointed by eq. (3.12), even though it has more number of guards.

Table 3.5: Optimal guards for the coverage problem with arbitrary cost function.

| Map Name | # optimal guards | | | |
|---|---|---|---|---|
| | Initial Optimization | Time (s) | Iterative Optimization | Time (s) |
| A | 7 | 2.881 | 6 | 21.859 |
| B | 6 | 1.471 | 5 | 10.747 |
| C | 1 | 1.024 | 1 | 12.488 |
| D | 10 | 10.849 | 9 | 29.538 |
| E | 7 | 1.820 | 7 | 13.417 |
| F | 14 | 73.668 | 13 | 212.976 |

## 3.4.5 Computation time

Now we evaluate the computation time needed for each section of the proposed algorithm, as well as the influence of using different solver for *the non-unicost* SCP. Most of the time are allocated for establishing the arrangement and *face*, including the visibility polygon calculation. In average, it takes around 74.25 % of the total time using the best performing SCP solver. The map simplification and generating the guard candidates spend around 17.45 %, while the non-unicost SCP only takes 8.3 % of the total time.

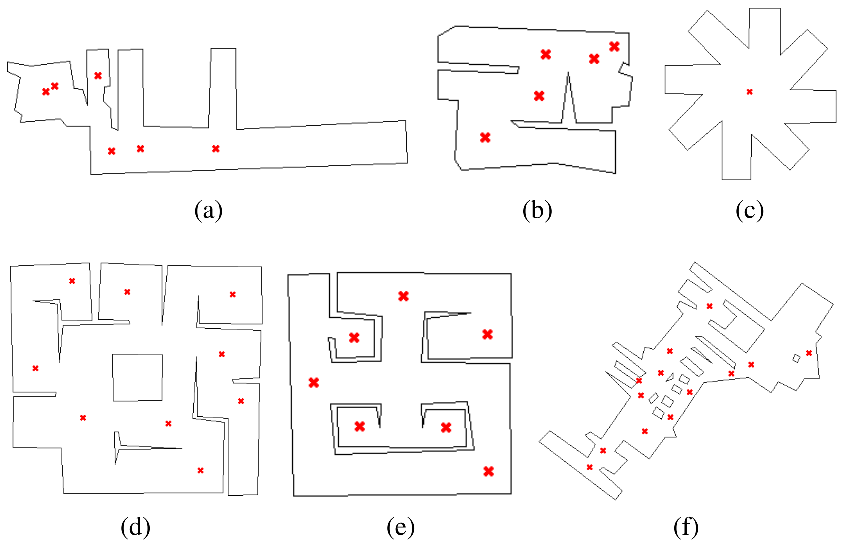


**Figure 3.10:** Optimal guards for the coverage problem using arbitrary cost map function. The guard positions are marked by the red crosses.

As a side result, we also report the effect of using different SCP solvers. Table 3.6 shows the calculation time needed for each SCP solver in the initial optimization stage. We select the initial optimization for comparing the performance of each solver because the huge number of guard candidates to be optimized are lied on it. As implied by table 3.6, the commercial solvers like CPLEX [48] and GUROBI [49] are superior to the open source solver such as GLPK [46]. For the map F, GPLK fails to retrieve the optimal solution. Surprisingly, SCIP [47] which is an open source solver has a comparable, or even better speed than the commercial one. Due to this reason, all results of the coverage problem mentioned in the previous section are accomplished using the SCIP.

Table 3.6: Calculation time of different SCP solver

| Map | Solving time (seconds) | | | |
|---|---|---|---|---|
| Name | GLPK | SCIP | GUROBI | CPLEX |
| A | 0.35 | **0.059** | **0.059** | 0.09 |
| B | 0.12 | **0.03** | **0.03** | 0.059 |
| C | 0.04 | 0.04 | **0.02** | 0.052 |
| D | 4.099 | **0.232** | 0.32 | 0.708 |
| E | 0.089 | **0.03** | **0.03** | 0.11 |
| F | N/A | **10.342** | 13.667 | 19.417 |

# Chapter 4

# Viewpoint Planning Algorithm

Now, we commence to the core of this thesis, i.e. the viewpoint planning algorithm for the guard robot. We initially introduce a variant of movement model called *geodesic motion model* which parameterizes the possible movement of both the person and the robot considering the environmental shape. We then provide two different approaches for the viewpoint planning algorithm, a greedy-based method and a stochastic optimization-based approach.

## 4.1 Geodesic Motion Model for The Robot and The Target Person Movement

Since an indoor environment is used, a simple linear motion model is not a favorable option. Under a long prediction time, the linear model most likely violates the obstacle constraints in a closed workspace. A natural way to predict the movement on an indoor setting is to follow the shape of the environment. Here we propose a geodesic motion model for achieving such requirement.

Consider a monotonic wavefront $\Phi(q)$ commenced from a source point $q_0$ moves across the configuration, the travel time $T$ of the wavefront in every point $q \in \mathcal{P}$ can be calculated by

$$\begin{aligned} \Phi(q) \simeq |\nabla T(q)| = \frac{1}{\zeta(q)}, \\ \Phi(q_0) = 0, \end{aligned} \tag{4.1}$$

where $\zeta(q)$ denotes a velocity function for specifying the speed of the wavefront. This problem is widely known as *Eikonal equation* problem. According to [50], eq. (4.1) can be approximated by the first order finite difference scheme

$$\left(\frac{T(q)-T_1}{\Delta x}\right)^2 + \left(\frac{T(q)-T_2}{\Delta y}\right)^2 = \frac{1}{\zeta(q)^2} \tag{4.2}$$

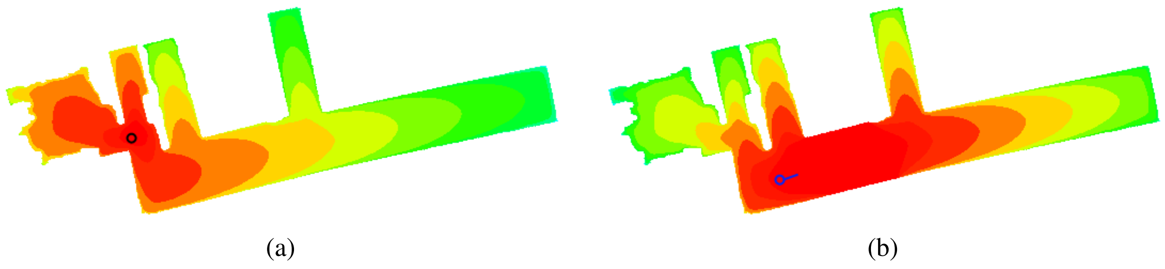


**Figure 4.1:** The travel time map of the robot (a) and the target (b). The black circle represents the robot's current position. The blue circle with line denotes the target position and its predicted movement.

where

$$\begin{aligned} T_1 &= \min\left(T(q_{x+1,y}), T(q_{x-1,y})\right), \\ T_2 &= \min\left(T(q_{x,y+1}), T(q_{x,y-1})\right), \end{aligned} \tag{4.3}$$

$\Delta x$ and $\Delta y$ respectively represent the difference of $q$ along *x-axis* and *y-axis*, while $q_{x+1,y}$ is the right side neighbor of $q$ along *x-axis* in a *cartesian coordinate*.

Numerical solution of eq. (4.2) is given by

$$\Phi(q) \simeq T(q) = \begin{cases} T_1 + \frac{1}{\zeta(q)} & \text{for } T_2 \geq T \geq T_1, \\ T_2 + \frac{1}{\zeta(q)} & \text{for } T_1 \geq T \geq T_2, \\ \Omega(T_1, T_2, \zeta(q)) & \text{for } T \geq \max(T_1, T_2), \end{cases} \tag{4.4}$$

where $\Omega(T_1, T_2, \zeta(q))$ denotes a quadratic solution[1] of (4.2).

We define the velocity model for both the robot and the target person, as follows:

1. The robot is expected to safely move within the environment. To achieve it, the distance map $\mathcal{D}$ in eq. (3.1) is utilized as the velocity function and normalized to the maximum speed of the robot $\varphi_r$,

$$\zeta_r(q) = \varphi_r \frac{\mathcal{D}(q)}{\|\mathcal{D}(q)\|}. \tag{4.5}$$

   It implies the robot is given a higher velocity at the location farther from obstacles (see Fig. 4.1a).

2. For the target, the current target velocity $\varphi_h$ is taken into account by using *velocity cone* model combined with the distance map $\mathcal{D}$. Let $\mathcal{C}$ be the cone area which consists of all point

[1]The quadratic solution in this equation means a quadratic equation $ax^2 + bx + c = 0$ has the solution $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

$q \in \mathcal{P}$ satisfying

$$\mathcal{C} = \left\{ \forall q | q \in \mathcal{P} \wedge \left( \angle q \leq \angle \vec{\varphi}_h \pm \frac{\pi}{3} \right) \right\}, \tag{4.6}$$

where $\angle \vec{\varphi}_h$ denotes the orientation of $\varphi_h$. Subsequently, the velocity function of the cone area $\zeta_C$ is given by

$$\zeta_C(q) = \begin{cases} \varphi_h & \text{for } q \in \mathcal{C} \\ \varepsilon & \text{for } q \notin \mathcal{C} \wedge q \in \mathcal{P} \\ 0 & \text{otherwise,} \end{cases} \tag{4.7}$$

where $\varepsilon$ is a small constant. $\zeta_C$ and $\mathcal{D}$ are then merged using *Hadamard product* to obtain the velocity model for the target person

$$\zeta_h(q) = \zeta_C(q) \circ \left( \frac{\mathcal{D}(q)}{\|\mathcal{D}(q)\|} \right). \tag{4.8}$$

Figure 4.1b shows that the travel time of the target person follows the shape of the environment.

By substituting eq. (4.5) and (4.8) into $\zeta(q)$ in eq. (4.2), the geodesic model of the travel time for the robot $\Phi_r(q)$ and the target person $\Phi_h(q)$ are then acquired.

## 4.2 Viewpoint Planning using Deterministic/Greedy Approach

This section is not intended to directly solve the guard robot problem mentioned in eq. (2.9)–(2.14). Instead, we establish a greedy method for solving the viewpoint planning problem, as a comparison. Back to the worst-case assumption, we try to find the most critical escaping gap $g_{critical} \in \mathcal{G}$ (i.e. the fastest one which can be reached by the target) by examining

$$g_{critical} = \arg\min_i T_{target}(g_i) \tag{4.9}$$

for $i = (1, 2 \ldots, n)$, $n$ is the number of escaping gaps, and $g_i \in \mathcal{G}$.

Basically, (4.9) also tells us that if the robot just stops at its position, it will lose the target at the predicted time $T_{target}(g_{critical})$. It gives us an important definition,

**Definition 4.2.1** (Condition of Losing Target)**.** ***1:*** *The stopping robot will lose the target at $T_{target}(g_{critical})$, except at a condition $T_{target}(g_{critical}) \rightarrow \infty$.*

It is imaginable that if the target moves away from all of escaping gaps (i.e. the target is always in the robot FOV), then based on the velocity model of the target, escaping gaps will have very small velocity which leads to a very large $T_{target}$ (see eq. (4.2) and (4.7)).

### 4.2.1 Planning using Cost Minimization

To get a minimum amount of the robot's movement while keeping the target inside the robot's FOV, we use cost minimization for the planning. We define several properties for the planning algorithm:

- $\mathcal{V}$ as set of viewpoints,
- $\mathcal{Q}$ as set of states,
- $\mathcal{A}$ as set of actions,
- $\mathcal{R}(\mathcal{Q},\mathcal{A})$ as applied rules based on $\mathcal{Q}$ and $\mathcal{A}$,
- $\beta(\mathcal{Q},\mathcal{A},\mathcal{R})$ as the cost caused by action $\mathcal{A}$, current state $\mathcal{Q}$, and rules $\mathcal{R}$.

We use two states, $\mathcal{Q} = \{q_0, q_1\}$, where $q_1$ is the state where the current position of the robot is at the one of possible viewpoints $\mathcal{V}$ and $q_0$ is the state where the robot is not at the viewpoint (i.e., the robot is moving from one viewpoint to another). We also define possible actions for the robot as $\mathcal{A} = \{a_0, a_{v_1}, \ldots, a_{v_k}\}$, where $k$ is number of viewpoints, $a_0$ represents "do nothing" action for the robot[2] (i.e. the robot just stops at the current position), and $\{a_{v_1}, \ldots, a_{v_k}\}$ are the action for the robot to go to one of viewpoint $v_i \in \mathcal{V}$.

We introduce the following rules for $\mathcal{R}(\mathcal{Q},\mathcal{A})$:

- **Rule 1**. The action $a_0$ is only applicable to the state $q_1$. This rule arises due to the definition of the viewpoint planning itself, where we want the robot to make a transition between viewpoints (i.e. the robot should not stop at non-viewpoint). It will cause a penalty $cr_1$, given by
$$cr_1(q,a) = \begin{cases} \infty & \text{for } q = q_0 \wedge a = a_0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

[2]The stopping action only describes that the robot stays at the same position, but actually the robot can make rotation or controlling the pan-tilt-zoom system to adjust its view toward the target, and it is also applicable for other actions.

- **Rule 2**. An action which leads to the viewpoint where the robot cannot see the critical escaping gaps is not applicable. It is understandable that such viewpoints will make our robot lose the target. This rule causes a penalty $cr_2$, given by

$$cr_2(a) = \begin{cases} 0 & \text{for } a \in \\ & \{a_i | \lambda_{critical} \in interior(\mathbb{V}(v_i))\} \\ \infty & \text{otherwise.} \end{cases} \tag{4.11}$$

- **Rule 3**. This rule is a consequence of “Definition 1”. Let $v_{critical}$ be a viewpoint inside the visibility polygon of the robot $\mathbb{V}(q^r)$ which is the nearest to the critical escaping gap. The rule is based on a simple intuition that if the time for the robot to reach the viewpoint $v_{critical}$ is longer than the time for the target to reach the critical escaping gap $g_{critical}$, then the robot will lose the target, or we can write it as

$$T_{robot}(v_{critical}) \leq T_{target}(g_{critical}) \tag{4.12}$$

Equation (4.12) happens on the stopping action, but it is also applicable for other actions. This rule causes a penalty $cr_3$, given by

$$cr_3(a) = \begin{cases} 0 & \text{for all } a \text{ when} \\ & T_{robot}(v_{critical}) \leq 0.8\mathbb{T} \\ 0 & \text{for } a = a_{v_{critical}} \text{ when} \\ & 0.8\mathbb{T} \leq T_{robot}(v_{critical}) \leq \mathbb{T} \\ \infty & \text{otherwise,} \end{cases} \tag{4.13}$$

where $\mathbb{T} = T_{target}(g_{critical})$. Basically, eq. (4.13) elaborates eq. (4.12) to see if the robot is at the critical time for losing the target. Before this critical time condition is violated, the robot can choose any action including the stopping action. When the critical time for losing the target is near, the robot has to do an action for preventing it (i.e., the robot should go to $P_{critical}$), because if the robot does not do anything then the condition will be violated, and any action cannot help the robot from losing the target (see the third row of eq. (4.13)). A constant 0.8 (experimentally obtained) is given to make sure the robot does not violate the critical time (i.e. ensuring the robot to move before violating the critical time).

Putting them together, we define the sum of the penalties caused by the rules as

$$\mathcal{R}(q,a) = cr_1(q,a) + cr_2(a) + cr_3(a), \text{ for } a \in \mathcal{A}, q \in \mathcal{Q} \tag{4.14}$$

We then exclude all actions which give $\mathcal{R}(q,a) \neq 0$ from $\mathcal{A}$.

Finally, we select the action based on the minimization problem of the cost $\beta(\mathcal{Q},\mathcal{A},\mathcal{R})$, given by

$$\arg\min_a \beta(q,a,\mathcal{R}), \text{ for } a \in \mathcal{A}, q \in \mathcal{Q} \tag{4.15}$$

where

$$\beta(q,a,\mathcal{R}) = T_{robot}(v_i), \tag{4.16}$$

$v_i$ is the viewpoint selected by action $a_{v_i}$. To summarize, the behavior of the guard robot will be as follows:

- When the robot is at a viewpoint, it will stay there within the viewpoint until the condition in rule 3 is violated.
- When the robot is not at a viewpoint (i.e. the robot is moving from one viewpoint to another), the robot will not stop until it reaches one viewpoint.

After an action was selected, the path which leads to the goal of the action is extracted as the planning result. The path is calculated by backtracking the geodesic motion model of the travel time for the robot $T_{robot}(q^r)$ from the goal (chosen viewpoint $v_i$) to the robot position $q^r$. We then send the planning result as a set of waypoints to a local path planner to be executed. We use path planner algorithm in [15] as the local path planner.

## 4.3 Viewpoint Planning using Stochastic Optimization

In this section, we present the planning method for solving the minimization problem in eq. (2.9)–(2.14). By definition 2.2.3, the planning action can also be interpreted as the problem of finding the optimal time and location for the robot to move, while intercepting the possibility of losing the target visibility. Either way, the robot is preferred to be idle.

One may wonder whether using the travel time fields $\Phi_r(q)$ and $\Phi_h(q)$ is sufficient to directly discover the solution of the above optimal time problem, as has been done by the above greedy approach. Let $\tau_i^h$ be the estimated time for the target person to reach the escaping gap $g_i \in \mathcal{G}$, and $\tau_j^r$ be the time for the robot to reach a viewpoint $v_j \in \mathcal{V}$ (precisely, $\mathcal{V}_{opt}$) covering $g_i$. As long as $\tau_i^h > \tau_j^r$, the target person does not seem to leave the robot visibility while the robot stays idle. Yet, the time values offered by $\Phi(q)$ are deterministic and does not consider the uncertainty of both the robot and the target person future states. Here a particle-based motion model over the travel time field $\Phi(q)$ is proposed to solve the problem.

### 4.3.1 Representing future states as particles

As previously described, instead of using a linear motion model for predicting the future states, we employ a geodesic motion model which is more representative for an indoor problem setting. Let $\psi_i^h : [q_0^h, g_i] \mapsto \mathbb{R}^m$ be a continuous curve describing the path of the target person towards each escaping gap $g_i \in \mathcal{G}$. These paths are obtained by performing gradient descent search over $\Phi_h(q)$. The same definition is applied to $\psi_j^r : [q_0^r, v_j] \mapsto \mathbb{R}^m$, which represents the path of the robot towards each viewpoint $v_j \in \mathcal{V}$, using $\Phi_r(q)$.

Distribution of the future state trajectories of both the robot and the target person is then approximated using particles, respectively correspond to each path $\psi_i^h$ and $\psi_j^r$. Consider $\mathbf{q}_t^{h,1:N} = \{q_t^{h,1}, q_t^{h,2}, \dots, q_t^{h,N}\}$ or simply written as $\mathbf{q}_t^h$ as particles, probabilistically describe the target person state at time $t$, where $N$ is the number of particles. In addition, let $\mathbf{q}_t^r$ be the particles of the robot. The path $\psi_i^h$ is subsequently parameterized by a set of particles $\{\mathbf{q}_k^h, \mathbf{q}_{k+1}^h, \dots\}$, where $k \in \{0, \Delta t, 2\Delta t, \dots, \tau_i^h\}$ and $\Delta t$ is time step. Accordingly, the path for the robot $\psi_j^r$ is described by $\{\mathbf{q}_k^r, \mathbf{q}_{k+1}^r, \dots\}$ with $k \in \{0, \Delta t, \dots, \tau_j^r\}$.

For each path, we then propagate the particles as follows

$$\begin{aligned}
\mathbf{q}_{k+1}^h &= \mathbf{q}_k^h + diag((\hat{q}_t^h - \overline{\mathbf{q}}_k^h)\mathbf{I}) + \mathbf{w}_h, \\
\mathbf{q}_{k+1}^r &= \mathbf{q}_k^r + diag((\hat{q}_t^r - \overline{\mathbf{q}}_k^r)\mathbf{I}) + \mathbf{w}_r, \\
\hat{q}_t^h &\sim q_{(k+1)\Delta t}^h \in \psi_i^h, \\
\hat{q}_t^r &\sim q_{(k+1)\Delta t}^r \in \psi_j^r,
\end{aligned} \tag{4.17}$$

where $\overline{\mathbf{q}}_k^h$ represents the mean of each particle $\mathbf{q}_k^h$, $\hat{q}_t^h$ represents the point in the path $\psi_i^h$ which has $\Delta t$ difference with the $\overline{\mathbf{q}}_k^h$, $\mathbf{w}_h$ describes the uncertainty, function $diag(\cdot)$ returns a vector of matrix's diagonal, and $\mathbf{I}$ is an identity matrix. Therefore, $\{\overline{\mathbf{q}}_k^r, \hat{q}_t^r, \mathbf{w}_r\}$ notations follow the similar descriptions, except it is now for the robot.

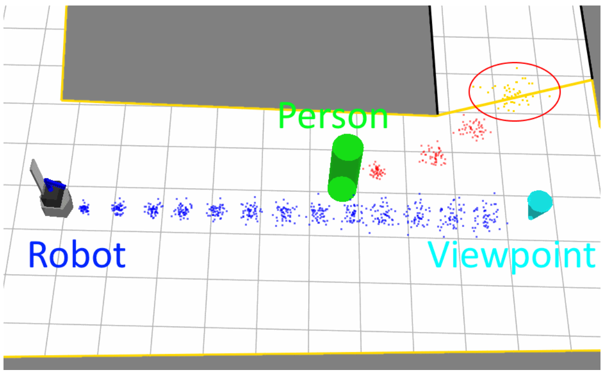


**Figure 4.2:** Particle illustration for the future states of the robot (blue) and the target person (red). Te yellow lines is the robot visibility polygon. The red ellipse denotes the predicted states of the target which violates the robot visibility.

The physical meaning of eq. (4.17) is that the particles for the target person try to follow the shape of the environment as pointed out by eq. (4.8). While for the robot, the particle sequences mean the robot is controlled towards the viewpoint using the speed suggested by the velocity function in eq. (4.5). It is basically the approximation of the robot control $u$ described in eq. (2.4). Figure 4.2 shows the particle representations of the future states.

### 4.3.2 The action plan based on chance constraint bound

In a greedy fashion, estimating the optimal time for the robot to start moving can be done by evaluating all possible combinations of the target and robot particles, the escaping gaps, and the viewpoints over the time. It should be done while ensuring the visibility towards the target person during the movement for each state of the robot and target person.

Let $\lambda$ be time delay for the robot to start moving from the current state. We expect to maximize $\lambda$ for reducing the robot movement,

$$\begin{aligned}
\max \quad & \lambda, \qquad \lambda = \{0, \dots, \tau_i^h - \tau_j^r\} && (4.18)\\
\text{s.t.} \quad & \mathbf{q}_k^h \in \mathbb{V}(\mathbf{q}_k^r) && (4.19)\\
& \forall \mathbf{q}_{k+\lambda}^h \in \psi_i^h, \quad \forall \mathbf{q}_k^r \in \psi_j^r && (4.20)\\
& \psi_i^h : [q_0^h, g_i] \mapsto \mathbb{R}^m, \quad \psi_j^r : [q_0^r, v_j] \mapsto \mathbb{R}^m, && (4.21)\\
& \forall g_i \in \mathcal{G}, \quad \forall v_j \in \mathcal{V}. && (4.22)
\end{aligned}$$

Equation (4.20) means the robot movement is delayed until time $k+\lambda$ (i.e. the target person has already moved for $\lambda$ unit time while the robot just starts to move, $k = 0$) for all possible $\lambda$. The maximum value of $\lambda$ is $\tau_i^h - \tau_j^r$ since for $\tau_i^h < \tau_j^r$ the target person will obviously leave the robot visibility. Calculating the visibility of the target person for each possible robot state in eq. (4.19) is costly and becomes the burden for running the optimization in real-time.

Instead of relying on the above greedy search to determine the optimal time to move, here we propose a technique to reduce the computational efforts, based on the following theorem,

**Theorem 4.3.1** (Trajectory Visibility Guarantee)**.** *The visibility of all states of the target person in a trajectory $\mathbf{q}^h_{1:\tau}$ is guaranteed, as long as $\mathbf{q}^h_\tau \in \mathbb{V}(\mathbf{q}^r_0)$.*

**Proof** (Trajectory Visibility Guarantee)**.** *The proof is straightforward. By definition 2.2.3, $\mathbf{q}^h_{1:\tau}$ is bounded by $\mathcal{P}$. Since the trajectory is monotonically increasing by means of the geodesic model, visibility of the last state $\mathbf{q}^h_\tau$ also implies the visibility of the rest states.*

By the fact that our proposed algorithm runs iteratively, instead of finding the optimal time for the robot to move in the future, we are more interested in examining whether it grants the visibility guarantee by stopping at the current state. By the above theorem, such guarantee can be acquired by evaluating the visibility of $\mathbf{q}^h_{\tau i,j}$ (the target person state when the robot reaches the viewpoint $v_j$, while the target is predicted to go to $g_i$). It implies the target should be still visible by the time the robot arrives at the viewpoint.

To correspond with the nature of the particle usage (i.e. uncertainty), we accordingly propose utilization of *chance constraint*, which is widely used in optimal control and obstacle collision assessment (e.g. [51] and [52]), for verifying bound of the visibility.

**Theorem 4.3.2** (Chance Constraint Bound)**.** *Probability of keeping the visibility of the target person is designated by chance constraint bound*

$$Pr(\mathbf{q}^h_{\tau_{i,j}} \notin \mathbb{V}(\mathbf{q}^r_0)) \leq \gamma, \tag{4.23}$$

*and it can be approximated using particles,*

$$\left(\frac{1}{N}\sum_{n=1}^{N} q^{h,n}_{\tau_{i,j}} \notin \mathbb{V}(q^r_0)\right) \leq \gamma, \tag{4.24}$$

**Proof** (Chance Constraint Bound). *We can write the expectation of event* $h(\mathbf{q}^h_{\tau_{i,j}}, \mathbf{q}^r_0) = \left(\mathbf{q}^h_{\tau_{i,j}} \notin \mathbb{V}(\mathbf{q}^r_0)\right)$ *as*

$$\mathbf{E}[h(\mathbf{q}^h_{\tau_{i,j}}, \mathbf{q}^r_0)] = \iint h(q^h_{\tau_{i,j}}, q^r_0) f(q^h_{\tau_{i,j}}) f(q^r_0) dq^h_{\tau_{i,j}} dq^r_0,$$

*where* $f(q^h_{\tau_{i,j}})$ *and* $f(q^r_0)$ *respectively represent the probability density function of* $\mathbf{q}^h_{\tau_{i,j}}$ *and* $\mathbf{q}^r_0$*. Since this integral is difficult to be evaluated in a closed form, it is approximated as*

$$\mathbf{E}[h(\mathbf{q}^h_{\tau_{i,j}}, \mathbf{q}^r_0)] \approx \frac{1}{N^2} \sum_{n=1}^{N} \sum_{n=1}^{N} h(q^{h,n}_{\tau_{i,j}}, q^{r,n}_0).$$

*By assuming a small uncertainty for the current robot state* $\mathbf{q}^r_0$*, we can approximate its mean,*

$$\mathbf{q}^r_0 \approx \bar{q}^r_0 = \frac{1}{N} \sum_{n=1}^{N} q^{r,n}_0.$$

*Hence, the event* $h(\mathbf{q}^h_{\tau_{i,j}}, \mathbf{q}^r_0) \approx h(\mathbf{q}^h_{\tau_{i,j}}, \bar{q}^r_0)$*. Subsequently, the left-hand side of eq.* (4.24) *is proved.*

Equation (4.23) means the target person state at time $\tau_{i,j}$ is permissible to be outside the robot visibility with probability at most $\gamma$, to be called “visible target". Using theorem 4.3.2, the optimal action $u^*_t$ for the robot is then selected as follows

$$u^*_t = \begin{cases} 0 & \text{for } \chi = 1, \\ \arg\min_u \phi(u, v, g) & \text{otherwise,} \end{cases} \tag{4.25}$$

where

$$\chi = \bigwedge_{\forall g_i \in \mathcal{G}, \forall v_j \in \mathcal{V}} Pr(\mathbf{q}^h_{\tau_{i,j}} \notin \mathbb{V}(\mathbf{q}^r_0)) \leq \gamma, \tag{4.26}$$

and

$$\begin{aligned} \phi(u, v, g) \simeq f(u, \varepsilon) &: [\mathbf{q}^r_0, v] \mapsto \mathbb{R}^m, \\ &\{\forall v \in \mathcal{V} | g \in \mathbb{V}(v)\}, \\ &\{\forall g \in \mathcal{G} | Pr(\mathbf{q}^h_{\tau_{i,j}} \notin \mathbb{V}(\mathbf{q}^r_0)) > \gamma\}. \end{aligned} \tag{4.27}$$

Equation (4.26) means the robot is “safe” to stop at the current position when all visibility bounds towards the escaping gaps are not violated by the predicted movement of the target person.

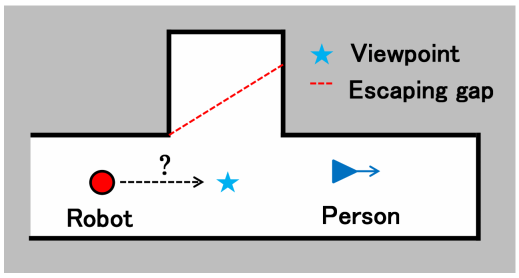


**Figure 4.3:** A case when moving the robot may be better than idle.

Either way, the robot should move to the viewpoint which covers the violated escaping gap. In case the robot faces several violated escaping gaps, the argument minimum over eq. (4.27) suggests the robot to choose the viewpoint which covers the most critical escaping gap and gives the minimum effort $f(u,\varepsilon)$ to execute the trajectory towards it.

### 4.3.3 Further consideration for the chosen action

Up to now, we have determined the bound for the robot to safely keep the target person visibility by staying at the current state. Let us consider another case, illustrated by Fig. 4.3. Figure 4.3 depicts that according to the current calculation (remember that our algorithm was done iteratively), the robot is suggested "safe" to stay at the current position. However, moving to the viewpoint may have a benefit to make the robot holds the visibility in a longer time, since the future escaping gap will disappear (or, shifted to a farther place). Here, an additional rule is incorporated into the optimization to let the robot executes the optimal action.

The rule basically examines distance of the nearest escaping gaps $||g_{ftr}||$ created by the future states of the robot $\mathbf{q}^r_{\tau^r_j}$ (when the robot reach the viewpoint $v_j$) towards the predicted target state at the same time step (i.e. $\mathbf{q}^h_{\tau^r_j}$). Equation (4.25) is then slightly modified, as follows

$$u_t^* = \begin{cases} 0 & \text{for } \chi = 1 \quad \&\& \quad ||g_{ftr}|| - ||g_{now}|| \leq \eta, \\ f(u,\varepsilon) : [\mathbf{q}^r_0, v^*] \mapsto \mathbb{R}^m & \text{for } \chi = 1 \quad \&\& \quad ||g_{ftr}|| - ||g_{now}|| > \eta, \\ \arg\min\limits_u \phi(u,v,g) & \text{otherwise,} \end{cases} \tag{4.28}$$

where $||g_{now}||$ denotes the distance between the target person and the nearest escaping gap for the current time, $v^*$ represents the viewpoint leading to the condition described in Fig. 4.3, and $\eta$ is a distance threshold.

The selected action drawn from eq. (4.28) is deemed as the planning result. It is then sent as a set of waypoints to a local motion planner [15] to be executed.

# Chapter 5

# Supporting Building Blocks: Path Planning Algorithm

In chapter 4, the viewpoint planning algorithm produces a set of waypoints for the robot. Using solely these waypoints will not make the robot moves as expected. Here, we suggest the use of path planning algorithm for executing a sequence of controls for the robot based on the given waypoints.

## 5.1 Introduction

In many robot applications, a path planner plays an important role for making the robot fulfill the given tasks, such as approaching a destination and avoiding collision with obstacles. The usual sequence of path planning algorithm is as follows; get the environment data using sensors, generate the path, and control the robot according to the generated path. There are several things which have to be considered to develop a path planning algorithm for real implementation of the robot: path optimality, path safety, and applicability to the real robot and environments.

Several parameters can be used to measure the path optimality, e.g. distance metrics, time cost, and other cost functions. For example, if we use a distance metrics as the measurement, it means that the path which gives the shortest distance toward a destination will be considered as the optimal path. The path safety means the algorithm must ensure that the robot has a “good" interaction with its surrounding environment, such as not colliding the other objects. The safety of the path also means that the generated path does not harm the robot itself, e.g. the algorithm does not generate a path with a very drastic change of velocity, which may harm the motor of the robot.

In the real application, the robot has kinematic and dynamic restrictions, such as speed, acceleration, and possible motion of the robot. These problems are often addressed as kinodynamic constraints of motion planning. Calculation time is also very critical for a real implementation of path planning algorithms.

### 5.1.1 Related Work

There are a lot of works which have been presented and discussed to address the problem of path planning. Randomized technique is one among many approaches which is used by many researchers ([53], [54]). A randomized path planner such as RRT (Rapidly-exploring Random Tree) [55] is widely accepted because of its ability to explore the tree in the vast area. A problem in RRT is that it produces path with many branches over the space due to its natural behavior of using randomized technique. Some studies have been conducted to overcome this problem. Urmson and Simmons [56] proposed a heuristic technique based on a probabilistic cost function to optimize generated trajectories. Another approach is presented by Bruce and Veloso [57] by introducing additional waypoint caches to improve the performance of the random tree algorithm.

Rodriguez *et al.* [54] presented a variant of RRT algorithm which can explore narrow passages. Vonasek *et al.* [58] propose an iterative scaling approach for RRT search, based on an iterative refinement of the guiding path using a scaled model of the robot. There are also researches on RRT using kinodynamic constraints by La Valle *et al.* [59] and Plaku *et al.* [60].

Several researchers ([53], [61], and [62]) also worked on sampling-based path planners. Karaman and Frazzoli [61] designed an incremental sampling-based path planner and proved its optimality on a static environment. Jaillet *et al.* [53] proposed a sampling-based method on configuration-space cost map. Zucker *et al.* [62] introduced a workspace-biased sampling to be applied on a bidirectional RRT.

Another work proposed by Hassouna *et al.* [63] does not use randomized technique, but instead uses a potential function generated by the Level Set Method over the free space. The path is extracted using sequences of the best value of the field between the initial position of the robot and the goal.

### 5.1.2 Our Approach

Most algorithms are either only consider static environment (e.g. [53], [54], [55], [56], [59], [61], [62], and [63]) or need a long calculation time thereby making it hard to be applied to the real robot (e.g. [60]). Some other algorithms do not consider kinodynamic restrictions of the robot in their simulations (e.g. [61] and [63]). That means it will need much efforts and modification to apply those algorithms to the real robot.

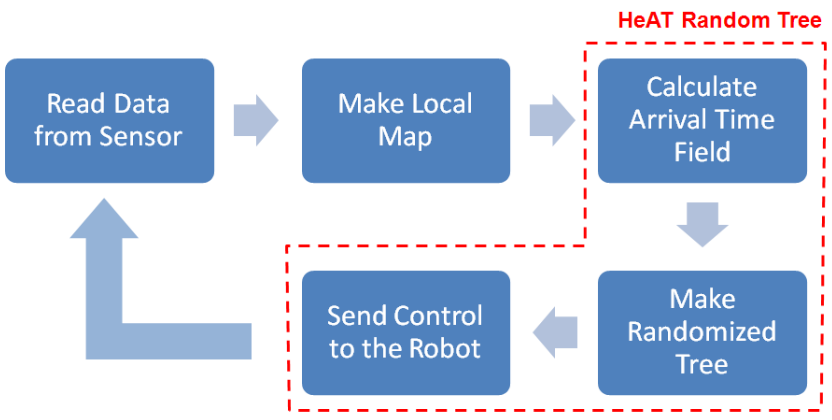


**Figure 5.1:** Flowchart of the path planning algorithm.

We introduce a new path planning algorithm using *the arrival time field* and randomized tree approaches. Basically, it is a real-time path planning algorithm that takes advantage on the high-exploration ability of randomized tree combined with an arrival time field and heuristics to achieve the path optimality, safety, and applicability to the real robot. We use the arrival time field to give a bias and guide the randomized tree expansion in a favorable way. Together with heuristics, the arrival time field effectively ensures the robot to choose the path in the tree expansion with a considerable clearance with any obstacle (safety) and has an optimum cost to reach the destination. These costs include the length, the time to go, and the smoothness of the path. Figure 5.1 shows the flowchart of our path planning algorithm.

## 5.2 Arrival Time Field

### 5.2.1 Definition

We first describe the basic idea of the *arrival time field*. Let us consider the environment $\mathbb{C}$ as a two dimensional space that holds information as follows:

- Non-passable area, which holds information about walls and static obstacles, denoted by $\boldsymbol{O} \subseteq \mathbb{C}$;
- Free space area, which defines observed areas where the robot will not collide with walls as well as static obstacles, denoted by $\boldsymbol{F} \subseteq \mathbb{C}$;

- Unknown area, which defines areas that have never been observed by the robot, e.g. area behind obstacle which cannot be observed by any sensor, denoted by $\boldsymbol{U} \subseteq \mathbb{C}$.

We define the *arrival time field* as a space containing the information about the time needed by each point in the space for approaching a determined goal point. Let us first consider a basic kinematic equation in one dimensional case which correlates speed $V$, position $x$, and time $T$ as

$$\Delta x = V \Delta T, \tag{5.1}$$

or we can rewrite it as

$$\frac{\Delta T}{\Delta x} = \frac{1}{V}. \tag{5.2}$$

In higher dimension, (5.2) can be expressed as

$$|\nabla T| = \frac{1}{V}. \tag{5.3}$$

### 5.2.2 Solving Eikonal Equation

Let a monotonic wave front originated from a determined source point moves across a space, then the arrival time of wave front in every point can be calculated using (5.3). The arrival time of a point depends on the distance from the source point and the speed used for traveling the wave front toward that point. This problem is known as *Eikonal equation* problem which can be solved by Godunov approximation [50]. In 2D space, for example, the equation is given by

$$\sqrt{\max\left(D_{i,j}^{-x}T, -D_{i,j}^{+x}T, 0\right)^2 + \max\left(D_{i,j}^{-y}T, -D_{i,j}^{+y}T, 0\right)^2} = \frac{1}{V_{i,j}}; (i,j) \in \boldsymbol{F} \tag{5.4}$$

where

$$\begin{aligned} D_{i,j}^{+x} &= T_{i+1,j} - T_{i,j}, \\ D_{i,j}^{-x} &= T_{i,j} - T_{i-1,j}, \\ D_{i,j}^{+y} &= T_{i,j+1} - T_{i,j}, and \\ D_{i,j}^{-y} &= T_{i,j} - T_{i,j-1}. \end{aligned} \tag{5.5}$$

$T_{i,j}$ is the arrival time value of cell (*i,j*), and $V_{i,j}$ denotes speed function of cell (*i,j*). Solution of (5.4) can be retrieved using a solver such as Fast Marching Method ([50], [63]), Fast Sweeping Method [64], or Fast Iterative Method [65].

We use Fast Marching Method (FMM) to solve (5.4). According to [63], (5.4) can be approximated by first order finite difference scheme

$$\max\left(\frac{T_{i,j}-T_1}{\Delta x},0\right)^2+\max\left(\frac{T_{i,j}-T_2}{\Delta y},0\right)^2=\frac{1}{V_{i,j}^2} \tag{5.6}$$

where

$$\begin{aligned} T_1 &= \min\left(T_{i+1,j},T_{i-1,j}\right)\\ T_2 &= \min\left(T_{i,j+1},T_{i,j-1}\right) \end{aligned} \tag{5.7}$$

The solution[1] of (5.6) is given by

$$T_{i,j}=\begin{cases} T_1+\frac{1}{V_{i,j}} & \text{for } T_2 \geq T \geq T_1\\ T_2+\frac{1}{V_{i,j}} & \text{for } T_1 \geq T \geq T_2\\ \text{quadratic solution of (5.6)} & \text{for } T \geq \max\left(T_1,T_2\right) \end{cases} \tag{5.8}$$

Equation (5.4) indicates that the arrival time of each point depends on its speed function $V_{i,j}$. That means we can set the influence of walls and static obstacles by adjusting the speed function, so that areas near obstacles have larger arrival time. For that purpose, we implement a monotonic function denoted by

$$V_{i_1,j_1}=\begin{cases} n^{\left\|x_{i_1,j_1}-x_{i_2,j_2}\right\|} & \text{for } x_{i_1,j_1}\in \boldsymbol{F}, x_{i_2,j_2}\in \boldsymbol{O}\\ 1 & \text{otherwise} \end{cases} \tag{5.9}$$

where $V_{i_1,j_1}$ is the speed on the point $x_{i_1,j_1}$, $x_{i_2,j_2}$ is the nearest point of obstacle to $x_{i_1,j_1}$, and $n$ is a constant for adjusting the monotonic function's value, to give more differences on each cell. We use the monotonic function's result for the speed function for calculating the arrival time field, which will make the speed near obstacle become smaller. These definitions make the robot keep some distances from walls and obstacles.

The emphasis point of the arrival time field is that it provides the minimum predicted arrival time for each point rather than the shortest distance to the goal. This predicted arrival time depends on its speed function. We can exploit the speed function by inserting information about several

[1]The quadratic solution in this equation means a quadratic equation $ax^2+bx+c=0$ has the solution $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$.

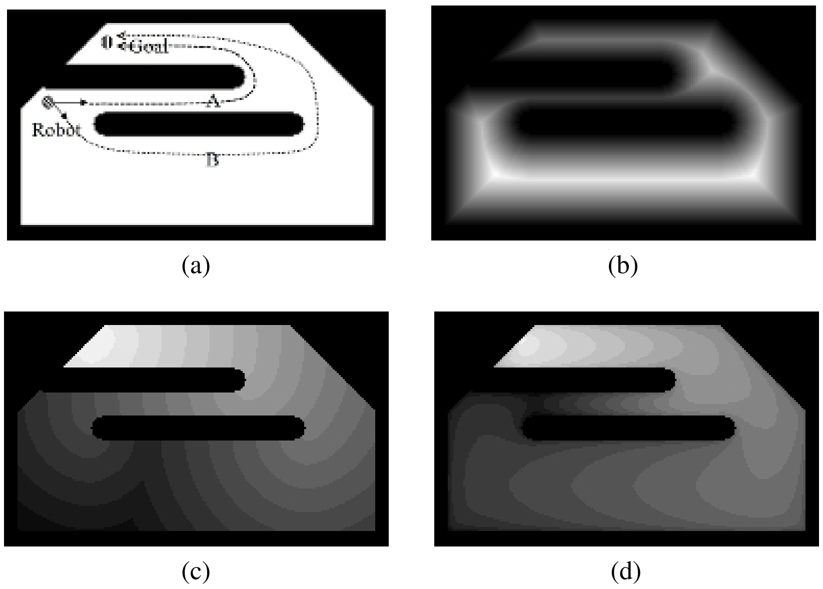


**Figure 5.2:** Advantages of using monotonic velocity function: (a) An environment with two possible paths toward the goal. (b) Monotonic velocity field of (a). (c) Arrival time field with uniform velocity function. (d) Arrival time field with monotonic velocity function. Darker color means longer arrival time. By occupying monotonic velocity function, taking path B is better according to (d)

possibilities that the robot may face in the environment. For example on a plain environment with obstacle, we can determine that the robot is better to move slower at a narrow space, a corridor, or an area next to the obstacles. We set a smaller speed on the area near the obstacle, then a shorter travel time will be through an area far from the obstacle (see Fig. 5.2). This example shows that the safety factor is also taken into account in the arrival time field calculation.

An *arrival time field* is used to provide a bias to drive expansion of the tree in a favorable direction toward the destination. We want the tree expands through a safe place within the fastest time. To fulfill such requirements, we apply the distance transform over free space ***F*** to give more weights on the cells near static obstacles and walls. We then perform calculation of the arrival time field originated from a goal point to give less weight on the closer position towards the goal.

The result of the arrival time field calculation is normalized and inverted so that the goal point has the highest weight. We then use that result as a bias for guiding the tree expansion.

## 5.3 Heuristic Arrival Time Field-biased (HeAT) Random Tree

It is easy to extract an optimal path for a point robot, i.e. the robot can freely move to all directions, using the result of *arrival time field* by backtracking the path along the fastest field from the start to the goal [63]. In the case of considering kinematics and dynamics of robot as constraints, as well as dynamic environments, it is very difficult to apply such approaches. We therefore propose a randomized kinodynamic path planner algorithm utilizing the arrival time field bias as its guidance and heuristic search to optimize the path, called *Heuristic Arrival Time Field-biased (HeAT) Random Tree*.

### 5.3.1 Definition

Our randomized tree is constructed by collections of reachable states $\mathbb{S}$ called node. Every node is defined by the tuple $S = \{x, y, \theta, v, w, t\} \in \mathbb{S}$, representing robot position in $xy$-axis and its heading $\theta$, current translational velocity ($v$) and rotational velocity ($w$) of the robot in that node, and time $t$ for reaching that node from the current state.

We give a predefined set of possible motions to the path planner. Each motion in the set consists of a translational and a rotational velocity as robot control denoted by $u_i = \{v_i, w_i\}, (i = 1, 2, \ldots, K)$, where $K$ is the total number of the motions, which satisfies kinematic constraints of the robot. Based on experiments, we currently use $K =$ a set of 85 motions which combine translational velocity (in mm/s) $v \in \{-100, -50, 0, \ldots, 600\}$ and rotational velocity (rad/s) $w \in \{-\frac{\pi}{2}, \ldots, \frac{\pi}{2}\}$.

Let $S_t$ be the current state and $S_{t+1}$ be the next state reached from $S_t$ using a chosen motion $u = \{v, w\}$. We define this action of extending state $S_t$ as a function

$$S_{t+1} \leftarrow g(S_t, u), \tag{5.10}$$

and according to [66], the new robot pose of $S_{t+1}$ can be obtained by the following equation:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \frac{v_1}{w_1} \begin{pmatrix} -\sin\theta_t + \sin(\theta_t + w_1 \Delta t) \\ \cos\theta_t - \cos(\theta_t + w_1 \Delta t) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ w_1 \Delta t \end{pmatrix}, \tag{5.11}$$

where $\Delta t$ is the time difference between $S_t$ and $S_{t+1}$.

### 5.3.2 Short-time Dynamic Obstacle Motion Model

In a dynamic environment, we consider a state as collision-free state if the state does not hit any static and dynamic obstacles. We need to perform a collision checking of every new state. Collision with static obstacles is checked using the information of non-passable area $\boldsymbol{O}$. For dynamic obstacles, we make a short-time dynamic obstacle motion model using constant speed for motion prediction of dynamic objects. Let $D_x^{'}(t)$ and $D_y^{'}(t)$ be the predicted position of an obstacle at time $t$ in $x$ and $y$ coordinate. We predict the position of each dynamic obstacle by

$$D_x^{'}(t) = D_x(0) + v_{D_x}t \tag{5.12}$$

$$D_y^{'}(t) = D_y(0) + v_{D_y}t \tag{5.13}$$

where $D_x(0)$ and $D_y(0)$ are the current position of the obstacle, and $v_{D_x}$ and $v_{D_y}$ are the speed of the obstacle on the respective $x$ and $y$ coordinate.

We assume that motion prediction of moving obstacle is effective for a short range of time, due to its uncertain behavior. We currently use fixed 10 time slices with cycle time of 500 milliseconds, started from the time of the current state of the robot. For each time slice, we predict both the position of each moving obstacle and that of the robot in order to see whether the robot and dynamic obstacles will cause a collision in that time slice.

### 5.3.3 Random Tree Algorithm

We expand the tree from the current position of the robot, using a similar approach to basic RRT [55] to take the advantages of its random exploration ability. Unlike the basic RRT algorithm which chooses a random point from the entire space, however we select a random point using the bias from the *arrival time field* so that the tree grows in a favorable direction toward the goal. We iteratively choose a random point $P_{target}$ which has higher value of arrival time field than a threshold value (please note that we invert the result of the arrival time field calculation, so that the goal point has the highest value). The threshold $Th$ is determined by

$$Th = bias(S_{init}) + K_{th}(bias(S_{far}) - bias(S_{init})) \tag{5.14}$$

where $bias(S_{init})$ is the arrival time value at the current robot position, $bias(S_{far})$ is the arrival time value at the node which has the highest value of the arrival time in the current iteration, and $K_{th}$ is a

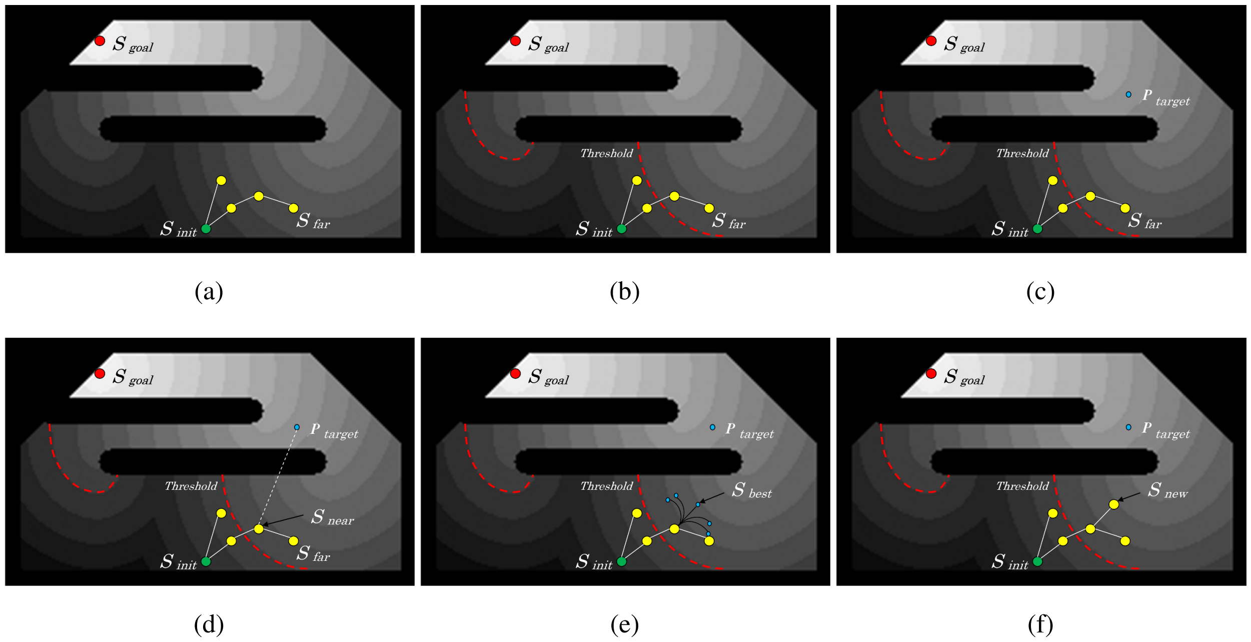


**Figure 5.3:** Random tree algorithm: (a) Initial condition before extending the tree. (b) Determine the threshold. (c) Pick a random point ($P_{target}$). (d) Choose the nearest node ($S_{near}$). (e) Evaluate all of possible motions. (f) Extend the tree ($S_{new}$).

constant between 0 and 1. The threshold value is started from the value of the arrival time field of the initial state (current robot position), and will grow when a new created node has a higher bias value than all of the current existing nodes. We then choose the nearest node $S_{near}$ to the $P_{target}$ among all of nodes in the tree.

Every time $S_{near}$ is chosen and eligible to be expanded, we will calculate a new state $S_{new}$ of that node ($S_{near}$) by evaluating all of possible motion controls

$$S_{u_i} \leftarrow g(S_{near}, u_i), \text{for } i \in \{1, 2, 3, \ldots, K\} \tag{5.15}$$

where $S_{u_i}$ is the extension of $S_{near}$ using motion control $u_i$, and $K$ is the total number of the motion set, which satisfies kinematic constraints of the robot, and free from any collision. Let $\{v_i, w_i\} \in u_i$ be $\{v_2, w_2\}$ and $\{v, w\} \in S_{near}$ be $\{v_1, w_1\}$, then possible motion controls which meet kinematic constraints can be chosen if the motion satisfies

$$\begin{aligned} v_2 &\leq v_{max}, \\ w_2 &\leq w_{max}, \\ \frac{(v_2 - v_1)}{\Delta t} &\leq a_{max}, \\ \frac{(w_2 - w_1)}{\Delta t} &\leq \alpha_{max}, \end{aligned} \tag{5.16}$$

where $v_{max}$, $w_{max}$, $a_{max}$, and $\alpha_{max}$ respectively denote maximum allowable translational velocity, angular velocity, translational acceleration, and angular acceleration, and $\Delta t$ is cycle time of calculation, currently 500 milliseconds.

We then pick the best motion control

$$u_{best} = \arg\min_i cost(S_{u_i}), \tag{5.17}$$

which gives the best cost function, to get the new node

$$S_{new} = S_{u_{best}} \leftarrow g(S_{near}, u_{best}), \tag{5.18}$$

$cost(S)$ is a cost function for evaluating a motion, defined by which is evaluated by

$$\alpha M_1 + \beta M_2 + \delta M_3, \tag{5.19}$$

$$M_1 = bias(S), \tag{5.20}$$

$$M_2 = dist(P_{target} - S), \tag{5.21}$$

$$M_3 = |\theta_S - \theta_{S_{near}},| \tag{5.22}$$

where $bias(S)$ is the arrival time value at predicted point$(x_S, y_S)$, $dist((x_{P_{target}}, y_{P_{target}}) - (x_S, y_S))$ is the distance between destination point $(x_{P_{target}}, y_{P_{target}})$ and the predicted point $(x_S, y_S)$. $|\theta_S - \theta_{S_{near}}|$ is the heading difference of the robot between the current state and the predicted state, and $\alpha$, $\beta$, and $\delta$ are constants.

The algorithm above is summarized as follows (see also Fig. 5.3 and algorithm 2):

1. Determine the threshold of region for picking a random point by (5.14);

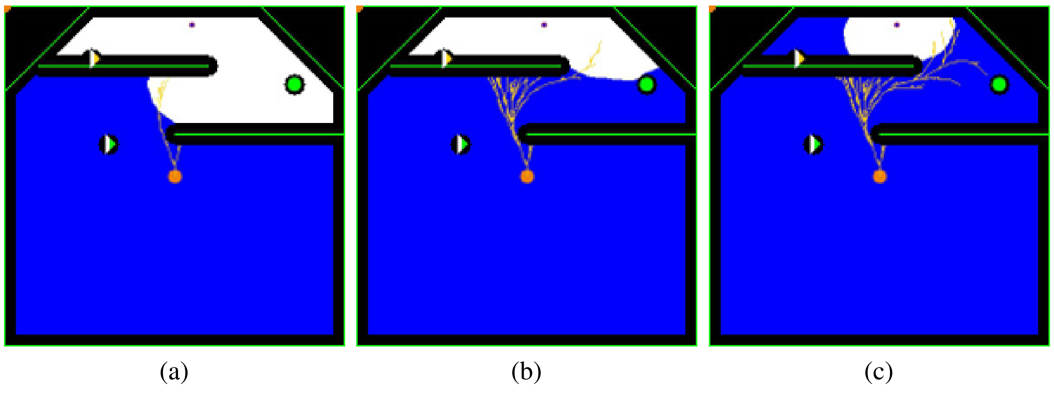


**Figure 5.4:** Growing the threshold from (a) to (c). The white area is the region for choosing a random point.

2. Pick a random point $P_{target}$ which has a better bias value than the threshold (see algorithm 3);
3. Choose the nearest node $S_{near}$ to the random point $P_{target}$;
4. Evaluate all of possible motions using (5.15); and
5. Extend the tree ($S_{new}$) by choosing the best motion (see algorithm 4).

**Algorithm 2** HeAT Random Tree Planner

```
 1: Require :
 2: $\mathbb{S}$ = collection of nodes
 3: $u$ = motion control
 4: $bias(S)$ = arrival time value of node $S$
 5: $threshold$ = bound the area for choosing random point
 6:
 7: procedure HEAT_RANDOM_TREE_PLANNER()
 8:     $\mathbb{S} \Leftarrow S_{init}$
 9:     $threshold \Leftarrow bias(S_{init})$
10:     while $time\_is\_available$ do
11:         $S_{near} \Leftarrow$ CHOOSE_STATE$(P_{target}, \mathbb{S})$
12:         $\mathbb{S} \Leftarrow \mathbb{S} \cup$ EXTEND_TREE$(S_{near}, u)$
13:         Update$(threshold)$                                  ▷ eq. (5.14)
14:     end while
15: end procedure
```

The constant $K_{th}$ in (5.14) gives us the control how fast the tree will grow and how disperse the tree will be. High value of $K_{th}$ means the threshold will increase rapidly, then make the tree grows fast and focus toward the goal. Low value of $K_{th}$ means the threshold will increase slowly,

**Algorithm 3** Choose State

```
1: **Require :**
2: $\mathbb{S}$ = collection of nodes
3:
4: **procedure** CHOOSE_STATE($P_{target}, \mathbb{S}$)
5:     **while** $time\_is\_available$ **do**
6:         $P_{target}$ = random point from $F$
7:         **if** $bias(P_{target}) \geq threshold$ **then**
8:             **return** nearest_node($P_{target}, \mathbb{S}$)        ▷ the nearest node in $\mathbb{S}$ to $P_{target}$.
9:         **end if**
10:    **end while**
11: **end procedure**
```

**Algorithm 4** Extending Tree

```
1: **Require :**
2: $u$ = set of motion controls
3: $temp\_cost$ = temporary variable for storing cost value
4: $S'$ = temporary variable for storing the information of a node
5:
6: **procedure** EXTEND_TREE($S_{near}, u$)
7:     **for all** $u$ **do**
8:         $S' \Leftarrow (S_{near}, u)$
9:         **if** $cost(S') \leq temp\_cost$ **then**        ▷ cost value is calculated according to (5.19).
10:            $S_{new} \Leftarrow S'$
11:            $temp\_cost = cost(S')$
12:        **end if**
13:    **end for**
14:    **return** $S_{new}$
15: **end procedure**
```

then we will get a more disperse tree. In the current implementation, we use $K_{th} = 0.25$. Fig. 5.4 shows the growth of the threshold.

The basic randomized planner always tends to make a disperse path due to its natural behavior. We need to determine a proper criterion for extending every node chosen by the randomized planner to reduce inefficient and disperse motions. In this case, we want to reduce unstable movements that are often found in the path created by randomized planner. We use the previous heading criterion as defined in (5.22) to ensure that we will not choose a very large difference of heading on each pair node causing unstable movements.

### 5.3.4 Restarting Tree Algorithm

We can expect that growing the tree from the initial point to the goal using the arrival time field bias takes a small amount of time. We will take the advantage of this fact to construct more possible paths. Once a node in the tree reach the goal area, the threshold for choosing random point is set back to the value of arrival time field of the robot's current state, and we repeat the process of expanding the tree. We call this processes as "restarting tree algorithm". We run the tree expansion algorithm for a fix amount of time, currently 200 ms. These restrictions are implemented in order to keep the computation time as fast as possible to be recognized as a real-time path planner.

### 5.3.5 Directing Initial Robot Heading

The utilization of kinodynamic constraints to the robot, i.e. the robot cannot freely move to all directions, may lead the randomized planner to make a large curve of path when the target position is in the opposite direction of the robot. In this case, it is better to direct the robot in a certain heading; pointing the robot directly to the goal position is not best choice, because in appearance of obstacles, it may lead the robot to wrong trajectories. We overcome this problem by adding a heuristic that is to direct the initial robot heading to the most promising area using a small frame of arrival time field.

A small frame of the *arrival time field*, centered on the robot initial position, is divided into four regions (Fig. 5.5). We calculate the total weight of each region and choose the region which has the largest weight. When the region behind the initial position of the robot has a larger weight, we will rotate the robot to that region and make that rotation as the initial expansion of the tree. We can see a comparison of the algorithms with and without this initial heading in Fig. 5.6.

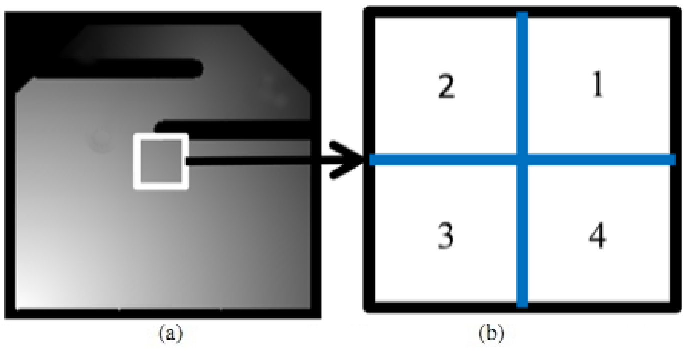


**Figure 5.5:** Making a small frame of the arrival time field: (a) the arrival time field, (b) a part of (a) centered at the robot position, divided into four region

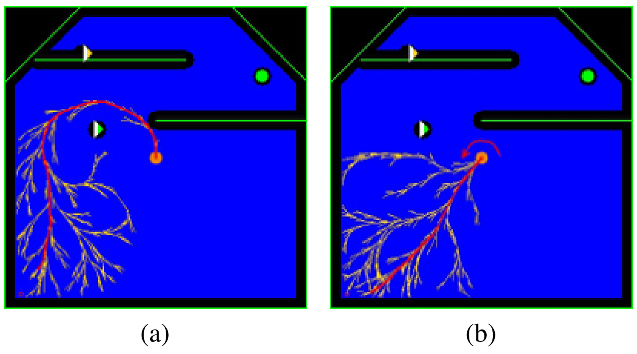


**Figure 5.6:** Effect of directing the initial heading of the robot: (a) without and (b) with the initial heading. The goal is at the lower left corner.

Fig. 5.6a shows a long arch of path when the robot does not use the initial heading. Contrary, we can reduce the cost of the path on the robot when we apply the initial heading at the beginning of the tree, as shown in Fig. 5.6b.

### 5.3.6 Path Extraction

HeAT Random Tree will provide several feasible paths of the robot from the initial state to the goal due to our *restarting tree algorithm*, satisfying the kinodynamic constraints and is free of collision with any static and dynamic obstacles. We examine all of the feasible paths by backtracking from the nodes which reach the goal area to the root of the tree and select the path which is the fastest one and has the most *Minimum Work (MW)* cost (see eq. (5.23)). The term *fastest* here means the path which has minimum time to reach the goal, and there are possibilities that several paths will have the same minimum time because of the restarting tree algorithm.

MW cost is computed by

$$MW = \sum_{t=start}^{goal} |w(t+1) - w(t)| \tag{5.23}$$

where $w(t)$ is angular velocity/steering radius (in radian/s) on the node $S_t$. The path, steering radius of which often change, i.e. non-smooth path, will have higher value of MW.

MW cost ensures that the chosen path is the smoothest one among all of the fastest paths. The first motion of the path is sent to the robot controller.

### 5.3.7 Reusability of Path

We use a pretty fast time cycle (currently, 500 milliseconds per cycle) for calculating the entire algorithm i.e., updating map information, static and dynamic obstacles, and performing calculation of HeAT Random Tree. We assume that the environment is not so much changed during that cycle. The path generated by the previous calculation is still expected to be feasible for the current cycle. The previous path is examined from the root to the longest collision-free state of the path and used it as initial tree for the current calculation.

## 5.4 Experiment Results

We test HeAT Random Tree path planner both in simulation and experiment with the real robot. All of implementations were done using a laptop PC (Core2Duo, 2.1 GHz, 2GB memory, Windows XP). We implement our path planner algorithm as an RT-component (see Figs. 5.7 and 5.8) which is a software module running on RT-middleware[2] environment [67] for reusability (cooperates with other modules in the real implementation, e.g. sensor modules, mapping modules, etc.).

### 5.4.1 Simulation: Local Planner

We use an Environment Simulator [68] to perform simulation of our path planner as a local planner. The simulator generates a 200x200 cells of map consisting of free space and static obstacles

[2]RT-middleware is a specification on a component model and infrastructure services applicable to the domain of robotics software development, authorized by OMG (Object Management Group).

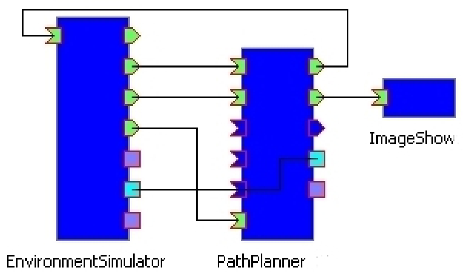


**Figure 5.7:** RT-Component connection for the simulations

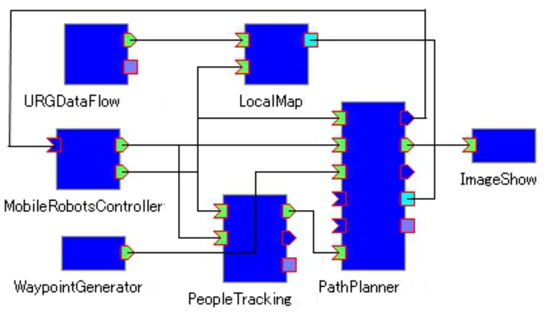


**Figure 5.8:** RT-Component connection for the experiments

as the local map for the robot, mimicking the canteen of our university. This simulator also provides people movement information to the path planner. Flow of the simulated people behavior is as follows: people enter the canteen, queue the ticket at ticket machine, take the meals using a tray, go to a free seat, stay on the seat for eating, bring the tray to the washing place, and go to the exit (see Fig. 5.9).

We apply HeAT Random Tree path planner to people tracking and waypoint following problems. Figure 5.10 shows simulation result on both problems, where the blue area denotes free space, the green line is wall and static obstacles, the orange circle denotes robot position, the red circle is the goal, the triangles represent people movement, and the black area is extending space for obstacles. Here, we want to show that our path planner is fast enough to run as a local planner, and can deal with both static and dynamic obstacles. Especially for people tracking problem, we want to show that our path planner can deal with a dynamic goal.

In the people tracking simulation, the robot has to follow one of people while avoiding static obstacles, walls, and other people. We will say that the robot has succeeded in solving people tracking problem when the robot can follow the tracked person from the entrance until that people stay at the table for eating (see Fig. 5.10a and 5.10c).

In the waypoint following simulation, the robot is given sequence of waypoints to follow

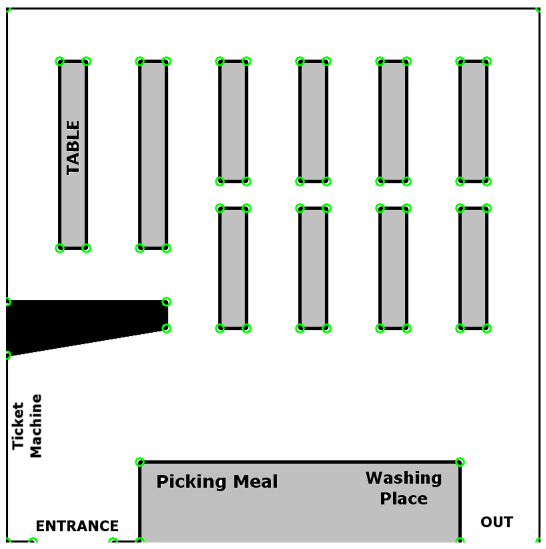


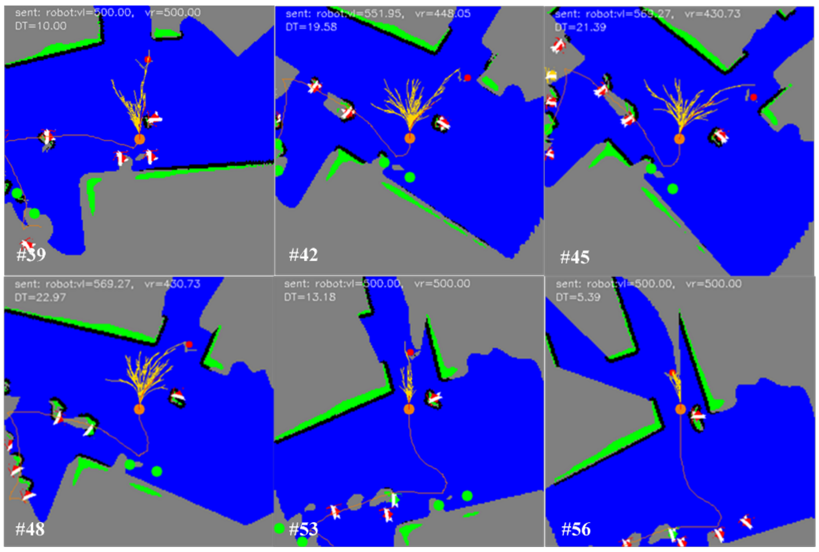
**Figure 5.9:** Modeled environment for simulation

**Table 5.1:** Statistic of Simulation Result

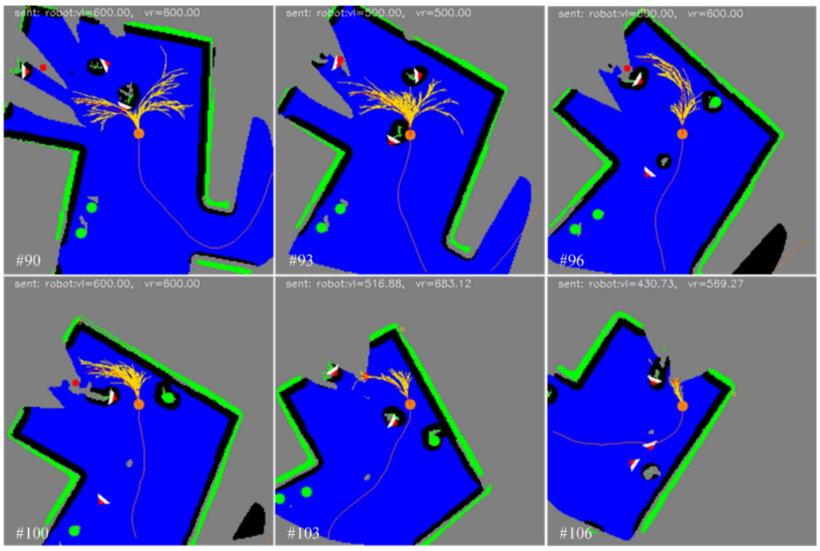
| Statistic | People Tracking | Waypoint Following |
|---|---|---|
| Arrival Time Field calculation (max) | 40 ms | 40 ms |
| Random Tree calculation (max) | 250 ms | 250 ms |
| Number of nodes (avg) | 1500 nodes | 3000 nodes |
| Maximum speed | 500 mm/second | 600 mm/second |
| Number of simulation | 10 times | 10 times |
| Successful runs | 100% | 100% |

where the goal lies on the very last waypoint of the sequence. The simulator gives several condition of environment like corridors, intersections, and open space with wandering dynamic obstacles. The task of waypoint following problem will be judged as succeed if the robot can arrive at the goal safely (see Fig. 5.10b and 5.10d).

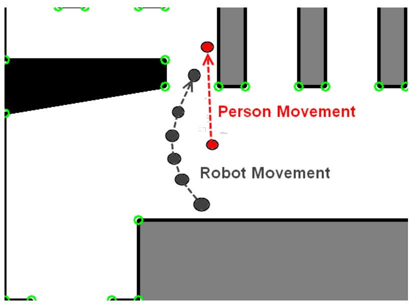
Table 5.1 shows the robustness of HeAT Random Tree algorithm. Each problem is done by 10 times simulations and has been done successfully. Overall calculations need less than 500 milliseconds. The path planner produces less node in the people tracking problem because the robot keeps close to the goal (i.e. followed person), so that the area for expanding the tree becomes narrower than the one on the waypoint following problem.


sent: robot:vl=500.00, vr=500.00
DT=10.00
#39
sent: robot:vl=551.95, vr=448.05
DT=19.58
#42
sent: robot:vl=569.27, vr=430.73
DT=21.39
#45
sent: robot:vl=569.27, vr=430.73
DT=22.97
#48
sent: robot:vl=500.00, vr=500.00
DT=13.18
#53
sent: robot:vl=500.00, vr=500.00
DT=5.39
#56

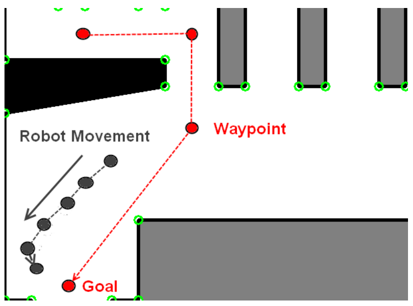
(a)


sent: robot:vl=600.00, vr=600.00
#90
sent: robot:vl=500.00, vr=500.00
#93
sent: robot:vl=600.00, vr=600.00
#96
sent: robot:vl=600.00, vr=600.00
#100
sent: robot:vl=516.88, vr=683.12
#103
sent: robot:vl=430.73, vr=569.27
#106

(b)



(c)



(d)

**Figure 5.10:** Screenshot sequences of simulation using Environment Simulator: (a) people tracking problem, (b) waypoint following problem, (c) global map view of (a), (d) global map view of (b). People is the goal in (a). The blue area denotes free space, the green line is wall and static obstacles, the orange circle denotes robot position, the red circle is the goal, the triangles represent people movement, and the black area is extending space for obstacles. Grey circles in (c) and (d) is global position of the robot respecting to local position in (a) and (b).

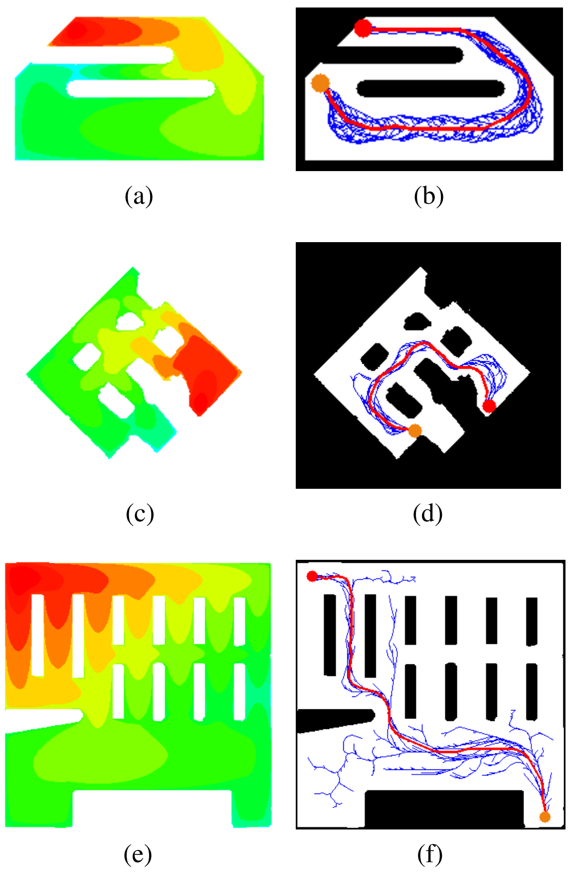


**Figure 5.11:** Simulation of HeAT Random Tree as global planner using various maps, its *arrival time field* (left), and respecting tree expansions (right). The blue lines denote trees, the red line is robot path, the orange circle denotes robot position, the red circle is the goal, the white and black area are respectively free spaces and obstacles.

### 5.4.2 Simulation: Global Planner

We then use our algorithm as a global planner (see Fig. 5.11). Fig. 5.11a and 5.11b show the benefit of *the arrival time field* with safety profile as we had explained in section 5.2, where the safest path will be chosen if applicable for the robot. Fig. 5.11d and 5.11f show how our path planner will act in the environment with several possible paths. By using *arrival time field*, our algorithm will try to find the fastest and smoothest path for the robot while maintains the safety. For example in Fig. 5.11f, there are many possible paths and *the arrival time field* nicely guides the tree expansion.

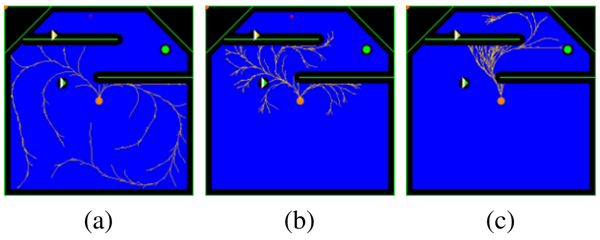


**Figure 5.12:** Comparison of tree expansion on the dynamic environment: (a) Original RRT, (b) hRRT, and (c) HeAT Random Tree

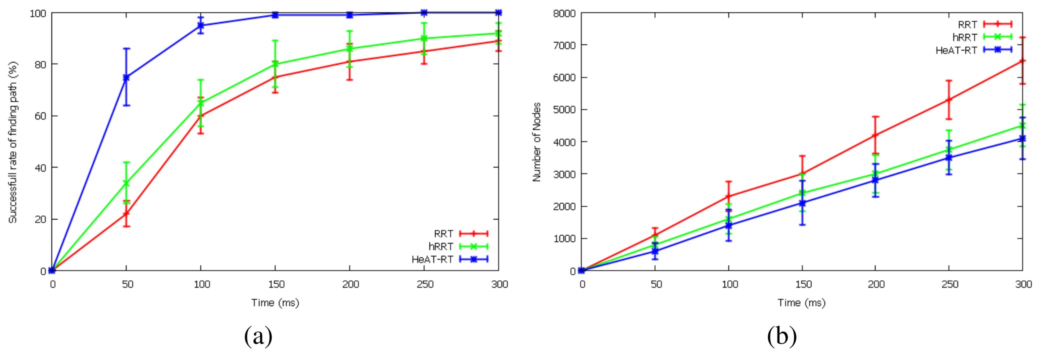


**Figure 5.13:** Comparison of RRT, hRRT, and HeAT-RT algorithms: (a) Successful rate of finding the path, (b) Number of nodes.

### 5.4.3 Comparison with Other Algorithms by Simulation

We provide a brief comparison between HeAT Random Tree algorithm and other existing RRT-based path planning algorithms. We use the original RRT algorithm by [55] and hRRT algorithm by [56] for comparison. Original RRT algorithm chooses a random point uniformly from the entire space, then extends the tree from the nearest node to that random point. hRRT algorithm uses a heuristic method based on a probabilistic cost function. This algorithm selects a random point for extending the tree using a *distance cost function* to reduce the dispersion of the tree produced by original RRT algorithm.

For fairness of comparison, we use the same environment. We also apply the same kinodynamic constraints to all of algorithms, even if each original algorithm did not concern about it. We set the maximum time to 300 milliseconds for performing the calculation of each algorithm, to be considered as real-time algorithm.

Fig. 5.12 can give us a good illustration about how the tree will expand in each algorithm on a dynamic environment. Original RRT algorithm expands the tree in a disperse way (Fig. 5.12a). hRRT gives a better result than Original RRT by occupying heuristic method of cost function, but

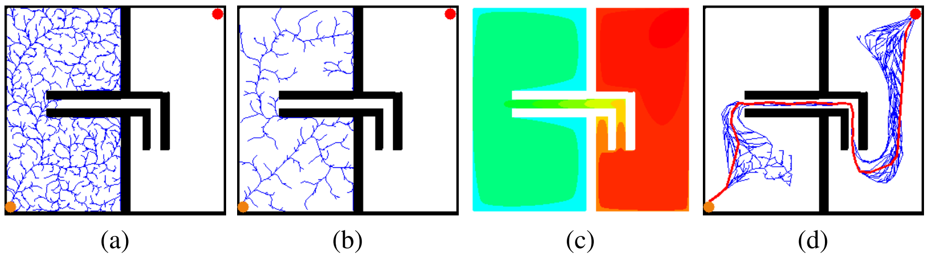


**Figure 5.14:** Comparison of tree expansion on the environment with narrow passage: (a) Original RRT, (b) hRRT, (c) Arrival Time Field, and (d) HeAT Random Tree

this heuristic method based on distance cost only considers the free spaces and the static obstacles, therefore it cannot handle dynamic environment and difficult to get out of obstacles in front of the robot as shown in Fig. 5.12b. On the contrary, HeAT Random Tree algorithm is nicely guided by the arrival time field and heuristics, and expands in a favorable way as shown in Fig. 5.12c.

Fig. 5.13 shows the effectiveness of HeAT Random Tree against original RRT and hRRT to find the path, based on the experiment on the Fig. 5.12. HeAT Random Tree can rapidly find and generate the path, even though RRT and hRRT can produce more nodes at the same time. High-rate of successful path generation is needed to ensure the robustness of the algorithm. HeAT Random Tree is expected to have longer time of calculation because of *arrival time field* generation, but surprisingly it is just a slight difference of node numbers among the three algorithms. This slight difference happens because original RRT and hRRT face many collision states that will slow down the computation time due to kinodynamic constraints, whereas HeAT Random Tree is well-guided by the arrival time field so that the tree will be expanded to the safe area.

Fig. 5.14 and 5.15 show comparison of each algorithm on the environment with narrow passages and bug-traps. Our algorithm effectively expands the tree, produces a nice path, and maintains the safety, while RRT and hRRT fail to find the path. Both figures show the benefit of *the arrival time field* with safety profile which guide the tree expansion in favorable way. Please notice in Fig. 5.14d, the path lies on the middle of narrow passage, which means our path planner can still maintain the safety of the path even in a difficult environment.

We then compare our HeAT Random Tree algorithm with RRT and hRRT algorithms using Environment Simulator for solving waypoint following problem. The robot is given the same initial position and goal to reach using each algorithm. Along the way in the simulator, the robot runs through several situations like narrow corridors, intersections, and open spaces with several moving obstacles surrounding the robot (see Fig. 5.16). We aim to test each algorithm using different kinds of environments and situations, to get better illustrations of performance of each algorithm. We run

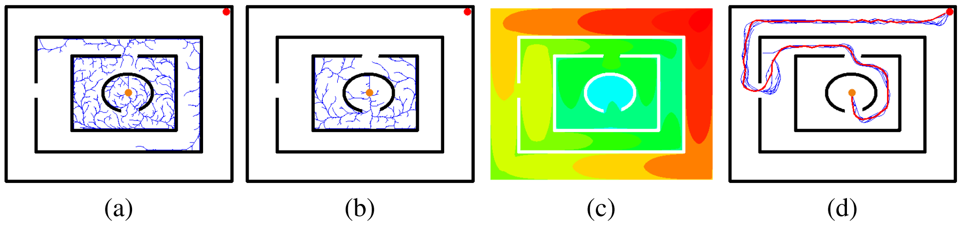


**Figure 5.15:** Comparison of tree expansion on the environment with multiple bug-traps: (a) Original RRT, (b) hRRT, (c) Arrival Time Field, and (d) HeAT Random Tree

**Table 5.2:** Comparison of Three Algorithms using Environment Simulator

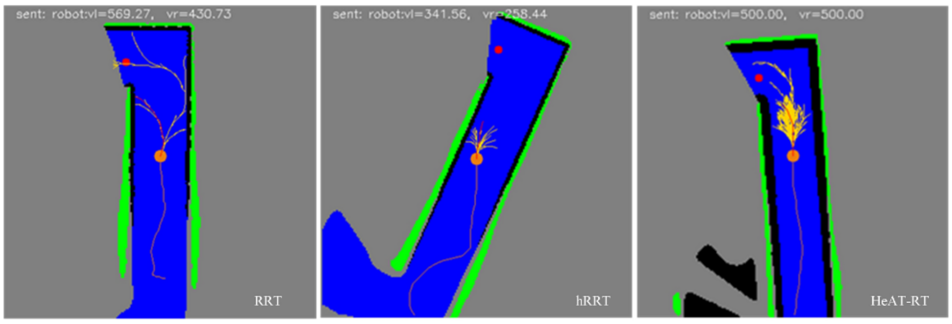
| | RRT | hRRT | HeAT RT |
|---|---|---|---|
| Time to goal (avg) | 157 sec. | 113 sec. | 58 sec. |
| Number of collision (avg) | 10 | 9 | 2 |
| MW-cost (avg) | 0.57 | 0.33 | 0.15 |
| Successful runs | 50% | 70% | 100% |

the simulation 10 times for each algorithm. Figure 5.16 visually shows that our algorithm can keep the path as smooth as possible on the narrow corridor and intersection and effectively grows the tree toward the goal on all of situations, comparing to other algorithms.

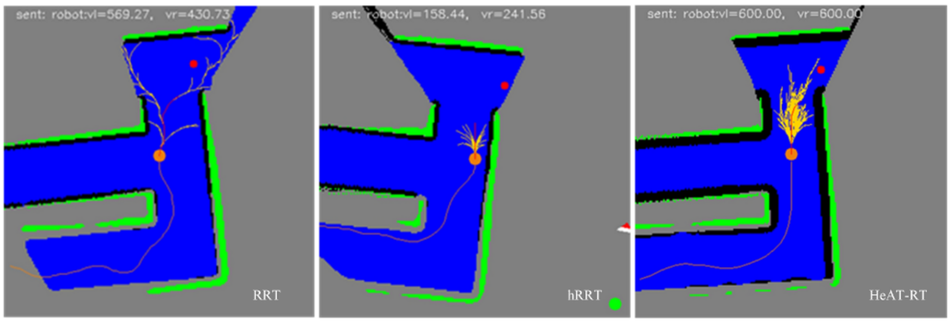
Table 5.2 shows the simulation results of each algorithm for waypoint following problem. The results are taken and averaged from successful run (i.e. the robot reaches the goal). Most of collisions occur because of moving obstacles. Averaged MW-cost results show the smoothness of the path generated by HeAT Random Tree, computed by

$$AveragedMW = \frac{\sum_{step=start}^{goal} |w(step+1) - w(step)|}{\sum step} \tag{5.24}$$

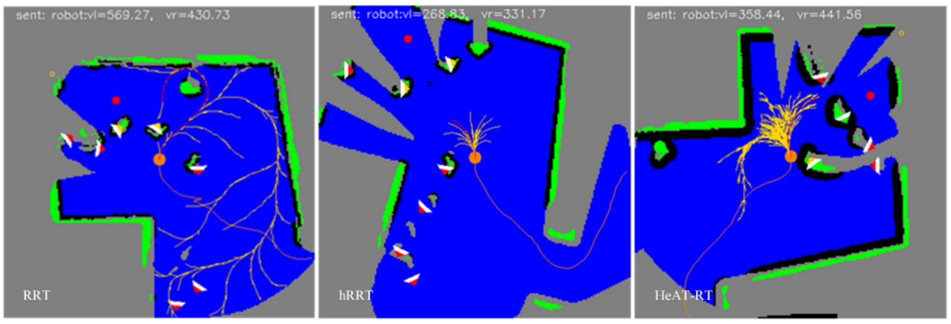
where $w(step)$ is angular velocity/steering radius control (in radian/s) on each step, and $\sum step$ is the total number of steps from the start to the goal, and collision states are not included in computation. Lower value of Averaged MW means more rare the robot changes the direction. Overall, this results show the superiority of our algorithm.


sent: robot:vl=569.27, vr=430.73
RRT
sent: robot:vl=341.56, vr=258.44
hRRT
sent: robot:vl=500.00, vr=500.00
HeAT-RT

(a)


sent: robot:vl=569.27, vr=430.73
RRT
sent: robot:vl=158.44, vr=241.56
hRRT
sent: robot:vl=600.00, vr=600.00
HeAT-RT

(b)


sent: robot:vl=569.27, vr=430.73
RRT
sent: robot:vl=268.83, vr=331.17
hRRT
sent: robot:vl=358.44, vr=441.56
HeAT-RT

(c)

**Figure 5.16:** Comparison of three algorithms at (a) narrow corridor, (b) intersection, (c) open space with several moving obstacles.

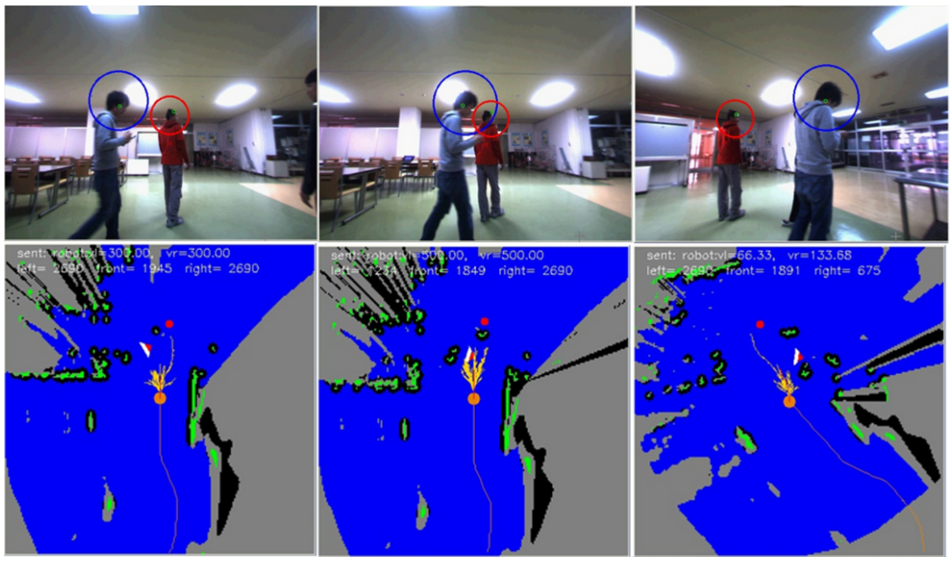


**Figure 5.17:** Experiment of people tracking in complex environment (canteen of our university): (from left to right) sequences of the real world (top) and local map scene (bottom). The robot has to follow the person in the red circle.

### 5.4.4 Experiment on Real Robots

We use a Patrafour robot by Kanto Auto Works Ltd. and ENON by Fujitsu Ltd. for experiments, equipped with a stereo camera (Bumblebee2 by Point Grey), a laser range finder (UTM-30LX by Hokuyo), and laptop PC. We test HeAT Random Tree path planner using two problems: people tracking and waypoint following problems.

The path planner utilizes a 200x200 grid of probabilistic local map with actual size of 10x10 meter. In the people tracking problem, the robot has to follow a person using a people tracking algorithm [69] while examining its environment. We use the canteen of our university representing a more complex environment (see Fig. 5.17). The robot follows one person while avoiding any collision with surrounding people.

In the waypoint following problem, we determine several waypoints which have to be reached by the robot on the real world. We use in-room environment for doing waypoint following experiment. The robot has to reach the goal which is located in front of the door while avoiding any collision with obstacles and wandering peoples (see Fig. 5.18).

Both following waypoints and people tracking tasks are successfully done within 500 milliseconds of computation time per cycle and using maximum speed of 500 millimeters per second. Those experiments show the ability of HeAT algorithm to be directly implemented on the real robot using a real environment.

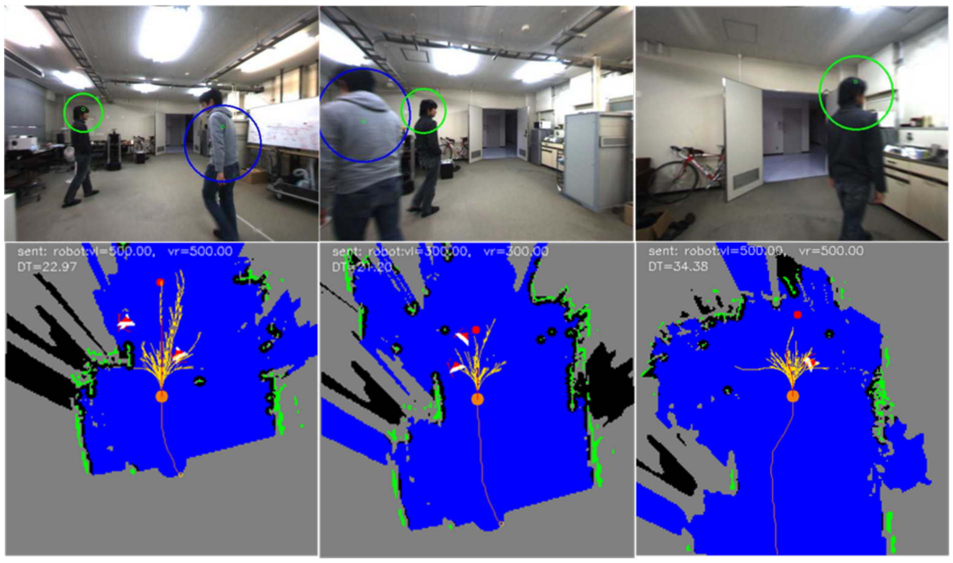

sent: robot:vl=500.00, vr=500.00
DT=22.97
sent: robot:vl=300.00, vr=300.00
DT=21.20
sent: robot:vl=500.00, vr=500.00
DT=34.38

**Figure 5.18:** In-room experiment of following waypoints: (from left to right) sequences of the real world (top) and local map scene (bottom). Circles in the top images represent detected person.

# Chapter 6

# Supporting Building Blocks: Person Detection and Tracking

By mean of the guard robot problem and the viewpoint planning, it suggests the need of "watching" action continuously to the target person. Here we propose a framework for doing such task, by employing a camera-based person detection and tracking algorithm. In addition, we also consider supplemental information of the person state, i.e. the person orientation. For the viewpoint planning purpose, we mainly utilize the positional information of the person, yet the orientation information serves a possibility of vast applications.

## 6.1 Introduction

Human body detection is one of challenging yet useful tasks for the mobile robot and surveillance applications. The human body detection with an additional orientation information can tell us how people interact with each other in the surveillance scenes. For example, we may predict that a group of persons facing each other for a long time are having conversation, or other social semantic predictions such as waiting for the bus together.

From the mobile robots point of view, the human body detection with an orientation estimation can assist the robot to get a better prediction for avoiding a person when doing navigation tasks. It also helps the robot to make a social interaction with the human in outdoor navigation, such as approaching a person and asking the way. Here the robot certainly needs the estimation of human orientation for facing the person.

Several works try to accomplish the human body detection and orientation estimation problem. Andriluka, *et al.* [70] uses banks of viewpoint specific part based detectors and linear Support Vector Machine (SVM) for estimating the whole body orientation. Another recent work is done by Weinrich, *et al.* [71] which performs the human upper body orientation estimation using Histogram of Oriented Gradients (HOG) features and SVM Tree. A work by Baltieri, *et al.* [72] employs

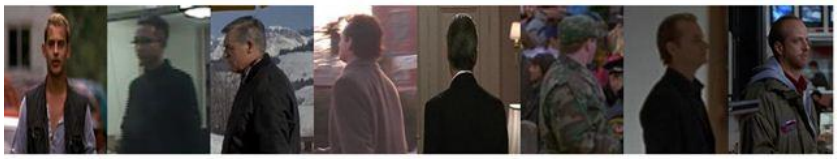


**Figure 6.1:** Eight classes of the human upper body orientation, representing (from left to right) front, front-left, left, back-left, back, back-right, right, and front-right directions.

a Mixture of Approximated Wrapped Gaussian (MoAWG) weighted by the detector outputs for increasing the correct estimation rates. The above researchers do not consider how the person movements will affect the overall estimation results.

One notable work by Chen, *et al.* [73] uses multi-level HOG features and sparse representation for classifying the human pose. They also employ a soft-coupling technique between the whole body orientation and its velocity using the particle filter framework.

To solve the problems we have mentioned above, here we propose a system for detecting and estimating the human upper body orientation, as well as its motion. We prefer to use the upper body part rather than the whole body for gaining the robustness under occlusion; the full body is often partially occluded by small objects such as chair, table, bicycle, and so on.

Our main contribution resides in the use of the Partial Least Squares (PLS)-based model of gradient and texture features for estimating the human upper body orientation. Here, our PLS model is a modification of the one used in [74] which has been successfully applied for human detection. We also provide an Unscented Kalman Filter (UKF) framework integrating the human movement prediction and the orientation estimation for improving the estimation results.

## 6.2 Model-based human upper body orientation estimation

### 6.2.1 Hierarchical system

Our system is built in a hierarchical manner. First we detect and create bounding boxes around the human upper body using a fast detector. These detection results are then fed to the orientation estimator part. We divide the orientation into eight quantization (see Fig. 6.1). Results from the detector and the orientation estimator are used as the observation inputs for the tracker. Figure 6.2 explains our framework on the human upper body orientation estimation as described above.

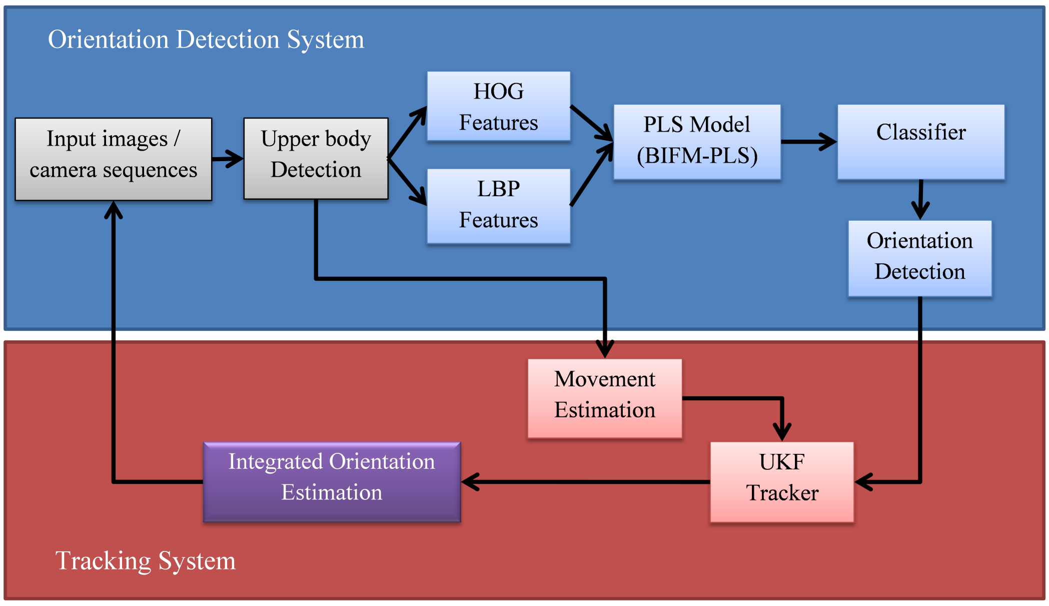


**Figure 6.2:** Diagram of the human upper body orientation estimation system.

Our system is composed of the image-based orientation estimation and the tracking system (see Fig. 6.2). The influence between the systems is one-directional. That is, the orientation estimation system runs independently, and its results are used by the tracking system for increasing the robustness of its pose and motion estimation. This also implies the orientation estimation system can be applied to still images.

### 6.2.2 Human upper body detection

Histogram of Oriented Gradients (HOG) [75] is one of state-of-the-art descriptor for the whole body person detection. However, the original HOG algorithm is slow and unsuitable for the real time applications. Here we exploit the extended work of HOG by [76], which employs *Adaboost* for selecting features and *cascade rejection* for speeding up the detection time. Instead of using the whole human body, we prefer to exploit the upper body part for gaining the robustness under occlusion, as we will show in subsection 6.4.2. The detection results (bounding boxes of the human upper body) are then used as the input for the orientation estimator part.

## 6.2.3 Extracting features for human upper body orientation estimation

The other works (*e.g.* [71] and [73]) solely depend on the gradient features, which capture the body shape. For the human upper body orientation case, we make an assumption that using only the body shape is not enough, for example, there is no big difference between the shape of the body facing front and back. Therefore, we propose a combination of the gradient-based and texture-based features which grab the shape and the texture cues. Here we expect the textured part of the body such as the face will be captured. We then apply the modified Partial Least Squares (PLS) models for enhancing the important cues of the human orientation.

#### 6.2.3.1 Shape cue

For capturing the human upper body shape, we use a multi-level HOG descriptor [75]. The gradient magnitude for each upper body image sample is first computed using 1-D mask $[-1\ \ 0\ \ 1]$ on each $x$ and $y$ direction. Every sample is divided into $3 \times 4$, $6 \times 8$, and $12 \times 16$ blocks, where each block consists of four cells. The gradient orientation is then quantized into nine bins. Overall we have 252 blocks and $9,072$ dimensional feature vectors of HOG descriptor.

#### 6.2.3.2 Texture cue

We use Local Binary Pattern (LBP), adopting the work of [77], to make a texture descriptor. We calculate image textures using $LBP_{8,1}$ operator for each pixel

$$I_{LBPc} = \sum_{p=0}^{7} 2^p \varsigma(I_p - I_c), \quad \varsigma(a) = \begin{cases} 1 & \text{for } a \geq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{6.1}$$

where $I_c$ is the center pixel from which we calculate the LBP value $I_{LBPc}$ and $p$ is eight-surrounding pixels of $I_c$. We then divide the LBP images into multiple blocks similar to the HOG features above. For each block, we make a histogram containing 59 labels based on *uniform patterns*[1]. This procedure gives us $14,868$ dimensional feature vectors in total.

[1]According to the original paper [77], the uniform patterns contain at most two bit transitions from 0 to 1 and *vice versa*. For a 8-bit data, there are 58 uniform patterns, and the other patterns which have more than two bit transitions are grouped into one label, so we have the total 59 labels.

### 6.2.4 Partial Least Squares for modeling features

#### 6.2.4.1 Partial Least Squares

Partial Least Squares (PLS) is a statistical method used for obtaining relations between sets of observed variables through the estimation of a low dimensional latent space which maximizes the separation between samples with different characteristics ([78], [74]). The PLS builds new predictor variables called *latent variables*, as combinations of a matrix $\boldsymbol{X}$ of the descriptor variables (features) and a vector $\boldsymbol{y}$ of the response variables (class labels).

Let us consider a problem with $\gamma$ samples, $\boldsymbol{X} \subset \mathbb{R}^{\delta}$ be an $\delta$-dimensional space representing the feature vectors and $\boldsymbol{y} \subset \mathbb{R}$ denote a 1-dimensional space of the class labels. The PLS then decomposes the $(\gamma \times \delta)$ matrix of zero mean variables $\boldsymbol{X}$ and the vector of zero mean variables $\boldsymbol{y}$ into

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{T}\boldsymbol{P}^{T} + \boldsymbol{E}, \\ \boldsymbol{y} &= \boldsymbol{U}\boldsymbol{q}^{T} + \boldsymbol{r}, \end{aligned} \tag{6.2}$$

where $\boldsymbol{T}$ and $\boldsymbol{U}$ represent $(\gamma \times s)$ matrices of $s$ extracted latent vectors, the $(\delta \times s)$ matrix $\boldsymbol{P}$ and the $(1 \times s)$ vector $\boldsymbol{q}$ are the loadings, and the $(\gamma \times \delta)$ matrix $\boldsymbol{E}$ and the $(\gamma \times 1)$ vector $\boldsymbol{r}$ are the residuals.

**Algorithm 5** PLS/NIPALS Algorithm

```
1: procedure PLS()
2:     Init $\boldsymbol{X}_0, y_0$
3:     for $i = 1 \to s$ do
4:         $\boldsymbol{w}_{i+1} \leftarrow X_i^T y_i$
5:         $\boldsymbol{w}_{i+1} \leftarrow \frac{\boldsymbol{w}_{i+1}}{\|\boldsymbol{w}_{i+1}\|}$
6:         $\boldsymbol{t}_{i+1} \leftarrow \boldsymbol{X}_i \boldsymbol{w}_{i+1}$
7:         $\boldsymbol{P}_{i+1} \leftarrow \frac{\boldsymbol{X}_i^T \boldsymbol{t}_{i+1}}{\boldsymbol{t}_{i+1}^T \boldsymbol{t}_{i+1}}$
8:         $q_{i+1} \leftarrow \frac{y_i^T \boldsymbol{t}_{i+1}}{\boldsymbol{t}_{i+1}^T \boldsymbol{t}_{i+1}}$
9:         $\boldsymbol{X}_{i+1} \leftarrow \boldsymbol{X}_i - \boldsymbol{t}_{i+1} \boldsymbol{P}_{i+1}^T$
10:        $y_{i+1} \leftarrow y_i - \boldsymbol{t}_{i+1} q_{i+1}$
11:    end for
12: end procedure
```

Here we implement the nonlinear iterative partial least squares (NIPALS) algorithm [79] to

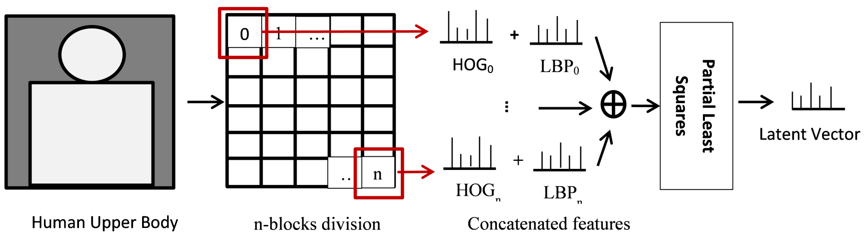


**Figure 6.3:** Extracting features using CFM-PLS

find a set of projection vectors (weight vectors) $\boldsymbol{W} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_s\}$ such that

$$[cov(\boldsymbol{t}_i, \boldsymbol{u}_i)]^2 = \max_{|\boldsymbol{w}_i|=1} [cov(\boldsymbol{X}\boldsymbol{w}_i, \boldsymbol{y})]^2, \tag{6.3}$$

where $\boldsymbol{t}_i$ is the $i$-th column of matrix $\boldsymbol{T}$ , $\boldsymbol{u}_i$ is the $i$-th column of matrix $\boldsymbol{U}$, and $cov(\boldsymbol{t}_i, \boldsymbol{u}_i)$ is the sample covariance between latent vectors $\boldsymbol{t}_i$ and $\boldsymbol{u}_i$. The process of constructing projection vectors $\boldsymbol{W}$ is shown in Algorithm 5.

#### 6.2.4.2 Block Importance Feature Model

Here we introduce our PLS-based feature models, *Block Importance Feature Model* of PLS (BIFM-PLS). We also provide another simple PLS model called *Combined Feature Model* of PLS (CFM-PLS) for comparison purpose.

**The Combined Feature Model of PLS (CFM-PLS)** is created by concatenating all of HOG-LBP features into one vector, and simply run the PLS algorithm on it with specified number of the latent vectors. As the result, the CFM-PLS produces a reduced set of features by projecting the feature vectors $\boldsymbol{f} \subset \boldsymbol{X}$ onto the weight vectors,

$$\check{\boldsymbol{x}} = \boldsymbol{W}\boldsymbol{f}. \tag{6.4}$$

This result is then used as the input for the classifier. Figure 6.3 explains the procedure of building CFM-PLS.

The CFM-PLS method, which resembles an ordinary PLS technique, is only used for comparison purposes, and from now we focus on the BIFM-PLS method.

**The Block Importance Feature Model of PLS (BIFM-PLS)** is built by an idea that not all of blocks or features have a high contribution to the classification. Here we want to examine the

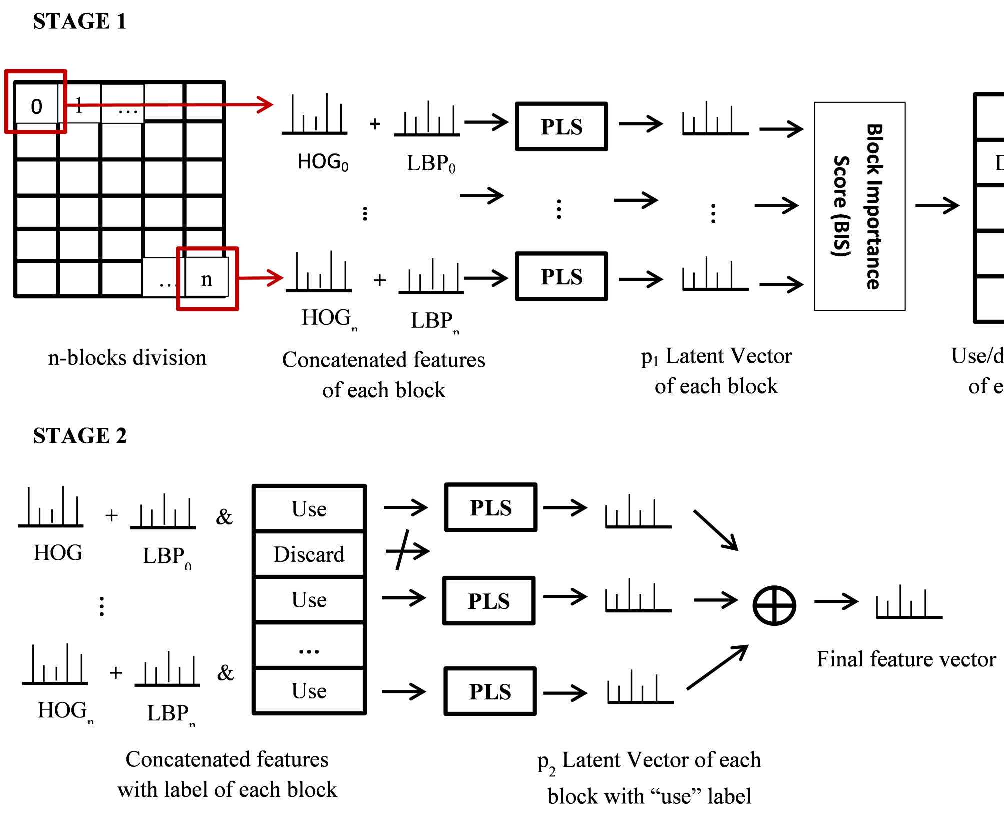

STAGE 1
0
1
...
n
HOG0
LBP0
HOGn
LBPn
PLS
Block Importance Score (BIS)
Use
Discard
Use
...
Use
n-blocks division
Concatenated features of each block
p1 Latent Vector of each block
Use/discard label of each block
STAGE 2
HOG
LBPn
&
Final feature vector
Concatenated features with label of each block
p2 Latent Vector of each block with "use" label

**Figure 6.4:** Extracting features using BIFM-PLS

contribution of each block and discard the one which has low importance. Unlike the CFM-PLS which highly reduce the feature space, the BIFM-PLS is intended to retain some details about the features to be fed up to the classifier.

We adopt and extend the method of [74] for picking out the representative blocks. For creating BIFM-PLS, we employ the feature selection called Variable Importance on Projection (VIP) (see [74] and [80]). The VIP gives a score for each feature representing its predictive power in the PLS model. The VIP of the $i$-th feature $\boldsymbol{f}$ is given by

$$VIP_i(\boldsymbol{f}) = \sqrt{\frac{\kappa \sum_{j-1}^{p} b_j^2 \boldsymbol{w}_{ij}^2}{\sum_{j=1}^{p} b_j^2}}, \tag{6.5}$$

where $\kappa$ is the number of features, $\boldsymbol{w}_{ij}$ denotes the $i$-th element of vector $\boldsymbol{w}_j$, and $b_j$ represents the regression weight for the $j$-th latent variable, $b_j = \boldsymbol{u}_j^T \boldsymbol{t}_j$.

We then extend the definition of VIP by introducing *block importance score* (BIS). The BIS ranks the predictive power of each block in the PLS model. The BIS on a multi-level blocks exhibits a “hierarchical" modeling, which first tries to find the important blocks from the multi-level/multi-size blocks and retrieve more detail information from the blocks which have better importance score. Algorithm 6 and Fig. 6.4 show the procedure of creating the BIFM-PLS model.

**Algorithm 6** BIFM-PLS Algorithm

```
Require:
 1: $n$: number of blocks
 2: $p_1$: number of latent vectors at stage 1
 3: $p_2$: number of latent vectors at stage 2
 4:
Ensure:
 5: $f_{BIFM}$: the reduced set of features vectors
 6: ————————————————————
 7: procedure CREATEBIFM()
 8:     BIFM_First($p_1$)
 9:     $f_{BIFM} \leftarrow$ BIFM_Reproject($p_2$)
10: end procedure
11: ————————————————————
12: procedure BIFM_FIRST($p_1$)
13:     for $i = 1 \rightarrow n$ do
14:         $f_i^{first} \leftarrow$ PLS($f_i, p_1$)
15:     end for
16: end procedure
17: ————————————————————
18: function BIFM_REPROJECT($p_2$)
19:     for $i = 1 \rightarrow n$ do
20:         if BIS($f_i^{first}$)$<$ threshold then
21:             $f_{BIFM} \leftarrow \bigcup$ PLS($f_i, p_2$)
22:         end if
23:     end for
24:     return $f_{BIFM}$
25: end function
```

The BIFM-PLS algorithm consists of two important stages: building the block importance and projecting the features on the important block. Let $n$ be the number of blocks for each HOG and LBP features, $m$ be the total number of concatenated HOG-LBP features in one block, $f_i$ denotes the feature sets at the $i$-th block, and $f_{i,j}$ represents the $j$-th feature at the $i$-th block with $i = \{1, \dots, n\}$ and $j = \{1, \dots, m\}$.

In the first stage, we extract a PLS model, take first $p_1$ latent variables for each block, and concatenate them to build a model $\boldsymbol{f}^{first}$, as follows

$$\boldsymbol{f}_i^{first} = PLS(\boldsymbol{f}_i, p_1), \tag{6.6}$$

where $PLS(\boldsymbol{f}_i, p_1)$ is a function for extracting $p_1$ elements from $\boldsymbol{f}_i$. Here $\boldsymbol{f}_i^{first}$ has element $\boldsymbol{f}_{i,j_1}^{first}$ where $j_1 = \{1, \ldots, p_1\}$.

We assume that a small number of $p_1$ is enough to see the contribution of each block to the orientation estimation, since the PLS considers the response variables (*i.e.* class labels). To see the importance of each block, we compute the VIP scores using eq. 6.5 and get the block importance score (*BIS*) as

$$BIS_i = \frac{1}{p_1} \sum_{j_1=1}^{p_1} VIP(\boldsymbol{f}_{i,j_1}^{first}), \quad i = 1, \ldots, n. \tag{6.7}$$

The top part of Fig. 6.4 shows the processes explained by eq. 6.6 and 6.7 above.

In the second stage, we build the BIFM feature vectors by calculating and concatenating the first $p_2$ latent variables on the important blocks, similar to eq. 6.6

$$\begin{aligned} \boldsymbol{f}_i^{second} &= PLS(\boldsymbol{f}_i, p_2), \\ \boldsymbol{f}^{BIFM} &= \bigcup_i \boldsymbol{f}_i^{second}, \quad \{\forall i | BIS_i > \eth\}. \end{aligned} \tag{6.8}$$

The blocks which have *BIS* under the threshold $\eth$, which mean less important blocks, are then discarded. These procedures are illustrated by the bottom part of Fig. 6.4.

Here we use $p_2 > p_1$, to get more detail information on each important block. We will examine the proper value of $p_1$ and $p_2$ in the experimental section (subsection 6.4.4).

## 6.2.5 Random Forest

One of the notable classifier which works well on the multi-class data is *Random Forest*, introduced by Breiman [81]. It is an ensemble learning method which combines the prediction of many decision trees using a majority vote mechanism. The Random Forest is devoted for its accuracy on the large dataset and multi-class learning. These advantages make us choose the Random Forest for training our eight-orientation classification problem with a large set of features.

Our Random Forest is constructed by multiple trees $T = \{T_1, T_2, \ldots, T_N\}$, where $N$ is number of trees. Let $\{d_i \in \mathcal{D}\}_{i=1\ldots K}$ denote $K$ training sets and $\{c_i \in \mathcal{C}\}_{i=1\ldots K}$ be its corresponding labels or classes (in our case, we have eight classes, $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_8\}$), where $\mathcal{D} \subset \mathbb{R}^M$ is the feature space. In the training phase, the Random Forest learns the classification function $T : \mathcal{D} \rightarrow \mathcal{C}$. Details of the Random Forest algorithm can be discovered at the original paper [81].

We use a linear split function [82]

$$\Phi(\boldsymbol{f}) = \boldsymbol{q}^T \boldsymbol{f} + \boldsymbol{z}, \tag{6.9}$$

where $\boldsymbol{q}$ is a vector which has the same dimensions as the feature vector $\boldsymbol{f}$ and $\boldsymbol{z}$ is a random number. The recursive training is run until the stopping criteria is reached, *i.e.* the maximum depth is met or no further information gain can be drawn.

In the testing phase, for a test case $d$, the Random Forest provides the posterior probability of each class as

$$p(c|d) = \frac{1}{N} \sum_{i=1}^{N} p_i(c|d), \tag{6.10}$$

where

$$p_i(c|d) = \frac{\Lambda_{i,c}}{\sum\limits_{j=c_1}^{c_8} \Lambda_{i,j}}, \tag{6.11}$$

$p_i(c|d)$ is the probability estimation of class $c \in \mathcal{C}$ given by the $i^{th}$ tree, and $\Lambda_{i,c}$ is the number of leaves in the $i^{th}$ tree which votes for class $c$. The overall multi-class decision function of the forest is then defined as

$$C(d) = \arg\max_{c \in \mathcal{C}} p(c|d). \tag{6.12}$$

## 6.3 Integration of orientation estimation and tracking

In the normal situation, the possibility that the person body orientation will be the same with its movements increases along with its speed. Based on this observation, our orientation estimation system is built using an assumption that the human body orientation is aligned with the direction of the human movements. In this case, we utilize both the result of the orientation detections and that of human movements estimation. On the opposite, we only rely on the orientation information from detections when the human movements are slow or even in a static condition.

To handle those matters, we predict the movement of persons in the world coordinate. We then use the UKF framework to combine the orientation estimation from the detection results and the movement predictions. The idea of coupling the detection result and the person movement is also used by [73], but at least three (3) things distinct our works from the [73]; the baseline (upper body vs whole body), features (HOG-LBP-PLS vs Sparse-HOG), and the framework (UKF vs particle filter).

### 6.3.1 Estimating human movement through its position in the real world

We derive movement of the persons from the change of their positions. We follow the work of [83] and [84] for projecting the position of each detected person from the 2D image coordinate to the 3D world coordinate. Let us consider a pinhole camera model, with the following parameters: focal length $f_c$, camera height $y_c$, horizontal center point $\mu_c$, and horizon position $v_0$. According to [84], the projection to the world coordinate is given by

$$W_d = \begin{bmatrix} \dfrac{y_c(\mu_d - \mu_c)}{v_d - v_0} \\ \dfrac{f_c y_c}{v_d - v_0} \\ \dfrac{h_d y_c}{v_d - v_0} \end{bmatrix}, \tag{6.13}$$

where $(\mu_d, v_d)$ is the bottom center point of the extended bounding box[2], and $h_d$ is the height of each detected person $d$ in the 2D image. Vector $W_d = [x_d^{world}, y_d^{world}, h_d^{world}]^T$ denotes the position $(x_d^{world}, y_d^{world})$ in the real world relative to the camera position and the height $h_d^{world}$ of the detected person $d$ in the image.

The horizon position $v_0$ is obtained by collecting line segments in the image using hough line detector and running RANSAC to evaluate all segments and get the intersection. This horizon estimation is done off-line[3] and we use it as a pre-calibrated value for the on-line tracking. For each frame, we send the position $(x_d^{world}, y_d^{world})$ to the tracker for getting the movement estimation in the real world.

[2]The extended bounding box is calculated by scaling the height of the bounding box to that of the human body, so that the bottom center is on the ground plane.

[3]To keep the speed of algorithm, we compute and use the horizon estimate as a constant. We assume the camera tilt is small (or in other words, the ground plane where the robot runs is flat).

## 6.3.2 Tracking strategies

### 6.3.2.1 State and observation models

Our state model is decomposed from the person position $(x_k, y_k)$, its derivative $(\dot{x}_k, \dot{y}_k)$, and the orientation components $(\varphi_k^M, \varphi_k^D, \theta_k)$ where $\varphi_k^M$ and $\varphi_k^D$ respectively denote the person orientation due to the movement and the detection estimation and $\theta_k$ denotes the final orientation of the person after considering the movement and the detection result, as stated by following tuple, $\mathcal{F}(\mathcal{X}) = \{x_k, y_k, \dot{x}_k, \dot{y}_k, \varphi_k^M, \varphi_k^D, \theta_k\}$. A constant velocity model is used for representing the person position and its derivative, as follows

$$\begin{cases} x_k &= x_{k-1} + \dot{x}_{k-1}\delta t + \varepsilon_x, \\ y_k &= y_{k-1} + \dot{y}_{k-1}\delta t + \varepsilon_y, \\ \dot{x}_k &= \dot{x}_{k-1} + \varepsilon_{\dot{x}}, \\ \dot{y}_k &= \dot{y}_{k-1} + \varepsilon_{\dot{y}}, \end{cases} \tag{6.14}$$

where $\{\varepsilon_x, \varepsilon_y, \varepsilon_{\dot{x}}, \varepsilon_{\dot{y}}\}$ are the gaussian noises for each component, and $\delta_t$ is the time sampling.

The orientation components of $\mathcal{F}(\mathcal{X})$ are then described as

$$\begin{cases} \varphi_k^M &= \arctan(\frac{\dot{y}_{k-1}}{\dot{x}_{k-1}}) + \varepsilon_{\varphi_M}(\upsilon_{k-1}), \\ \varphi_k^D &= \varphi_{k-1}^D + \varepsilon_{\varphi_D}, \\ \theta_k &= \varphi_{k-1}^D + \frac{\omega(1-e^{-\upsilon_{k-1}})}{1+e^{-\upsilon_{k-1}}}(\varphi_{k-1}^M - \varphi_{k-1}^D) + \varepsilon_\theta, \end{cases} \tag{6.15}$$

where $\{\varepsilon_{\varphi_D}, \varepsilon_\theta\}$ are the gaussian noises and $\omega$ is a constant.

The noise function $\varepsilon_{\varphi_M}(\upsilon_{k-1})$ in the first line of eq. 6.15 is defined as

$$\begin{aligned} \varepsilon_{\varphi_M}(\upsilon_{k-1}) &= \mathcal{N}(0, g(\upsilon_{k-1})), \\ g(\upsilon_{k-1}) &= \sigma_\upsilon \exp^{-\upsilon_{k-1}}, \\ \upsilon_{k-1} &= \sqrt{\dot{x}_{k-1}^2 + \dot{y}_{k-1}^2}, \end{aligned} \tag{6.16}$$

where $\upsilon_{k-1}$ is the magnitude of the person movement, and $\sigma_\upsilon$ is a constant. Using eq. 6.16, the noise function $\varepsilon_{\varphi_M}(\upsilon_{k-1})$ can be read as a zero-mean gaussian of which the variance varies *w.r.t* the person movement $\upsilon$.

Equation 6.15 suggests the influence of the person movement to the orientation estimation.

This equation, together with eq. 6.16, tells us that when the velocity of a person is small enough ($\upsilon_{k-1} \approx 0$), the uncertainty of the movement orientation becomes large (see $\varepsilon_{\varphi_M}(\upsilon_{k-1})$), and the orientation estimation will mainly depend on $\varphi_k^D$ (which is updated using the orientation detection result in the observation model (see eq. 6.17)). On the contrary, when the velocity is large then the movement orientation becomes more reliable and the orientation estimation depends on both detection and the person movement.

Here the constant $\omega$ has a duty for controlling the influence of the detection and the person movement to the overall orientation. The value of $\omega$ is later investigated in the experimental section (subsection 6.4.5).

We then use the observation model for the person position in the world coordinate (derived from eq. 6.13) and for the person orientation from the result of the section 6.2, as follows

$$\mathcal{H}(\mathcal{X}) = \begin{cases} \mu_k &= \frac{x_k(\frac{f_c y_c}{y_k})}{y_c} + \mu_c + \varepsilon_\mu \\ v_k &= \frac{f_c y_c}{y_k} + v_0 + \varepsilon_v \\ h_k &= \frac{\frac{f_c y_c}{y_k} h_d}{y_c} + \varepsilon_h \\ C_k &= f(C(d), \varphi_k^D) + \varepsilon_c \end{cases} \tag{6.17}$$

where $(x_k, y_k)$ denote the person position, $(f_c, y_c, \mu_c, v_0)$ are the camera parameters (focal length $f_c$, camera height $y_c$, horizontal center point $\mu_c$, and horizon position $v_0$), and $\{\varepsilon_\mu, \varepsilon_v, \varepsilon_h, \varepsilon_c\}$ are the gaussian noises for each observation component. $\mu_k, v_k$, and $h_k$ have the same definition with $\mu_d, v_d$, and $h_d$ in the subsection 6.3.1. The function $f(C(d), \varphi_k^D)$ maps the result of the multi-class decision function $C(d)$ (eq. 6.12) into their equivalent angle of the orientation $\varphi_k^D$.

#### 6.3.2.2 Unscented Kalman Filter tracker

For choosing the tracker, we consider our hardware limitation and the system nonlinearities. Several well-known filters can be adopted for handling the nonlinearities, such as Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and particle filter. Here we choose UKF because we want to avoid the resource costs of the particle filter (used by [73]) and the calculus of the Jacobian matrices used in the EKF.

The UKF [85] employs the *unscented transform*, a deterministic sampling technique, to take a minimal set of sample points called sigma points around the mean. Consider the current state with the mean $\hat{x}$ and its covariance $\boldsymbol{P}_x$. We build a set of 2L+1 sigma points matrix $\mathcal{X}$, as follows

$$\begin{aligned}
\mathcal{X}_0 &= \hat{x} \\
\mathcal{X}_{i,k-1} &= \hat{x} + (\sqrt{(L+\lambda)\boldsymbol{P}_x})_i, i = \{1,\dots,L\} \\
\mathcal{X}_{i,k-1} &= \hat{x} - (\sqrt{(L+\lambda)\boldsymbol{P}_x})_{i-L}, i = \{L+1,\dots,2L\} \\
\lambda &= \alpha^2(L+\kappa) - L,
\end{aligned} \tag{6.18}$$

where $L$ is the dimension of the augmented state, $\alpha$ and $\kappa$ are constants for controlling the spread of the sigma points.

We then define $\boldsymbol{W}^{(m)}$ and $\boldsymbol{W}^{(c)}$ as the weight for the mean and covariance respectively, given by

$$\begin{aligned}
\boldsymbol{W}_0^{(m)} &= \frac{\lambda}{(L+\lambda)} \\
\boldsymbol{W}_0^{(c)} &= \frac{\lambda}{(L+\lambda)} + (1-\alpha^2+\beta) \\
\boldsymbol{W}_i^{(m)} &= \boldsymbol{W}_i^{(c)} = \frac{1}{2(L+\lambda)}, i = \{1,\dots,2L\}
\end{aligned} \tag{6.19}$$

where $\beta$ integrates prior knowledge of the distribution of $\hat{x}$. We use the default value, $\alpha = 10^{-3}, \beta = 2$, and $\kappa = 0$.

We compute mean and covariance of the prior estimation ($\mathcal{X}_{k|k-1}$ and $\boldsymbol{P}_k^-$) using the state model $\mathcal{F}(\mathcal{X})$ in eq. 6.14 and 6.15, and the sigma points above as follows

$$\begin{aligned}
\mathcal{X}_{i,k|k-1} &= \mathcal{F}(\mathcal{X}_{i,k-1}), i = \{1,\dots,2L\} \\
\hat{x}_k^- &= \sum_{i=0}^{2L} \boldsymbol{W}_i^{(m)} \mathcal{X}_{i,k|k-1} \\
\boldsymbol{P}_k^- &= \sum_{i=0}^{2L} \boldsymbol{W}_i^{(c)} [\mathcal{X}_{i,k|k-1} - \hat{x}_k^-][\mathcal{X}_{i,k|k-1} - \hat{x}_k^-]^T + \boldsymbol{Q}
\end{aligned} \tag{6.20}$$

where $\boldsymbol{Q}$ is the covariance of the process noises.

In the measurement update phase, we project the sigma points through the observation function

$\mathcal{H}(\mathcal{X})$

$$\begin{aligned}\mathcal{Z}_{i,k|k-1} &= \mathcal{H}(\mathcal{X}_{i,k|k-1}), i=\{1,\ldots,2L\}\\ \hat{z}_k^- &= \sum_{i=0}^{2L} \boldsymbol{W}_i^{(m)} \mathcal{Z}_{i,k|k-1},\\ \boldsymbol{P}_{z_k z_k} &= \sum_{i=0}^{2L} \boldsymbol{W}_i^{(c)} [\mathcal{Z}_{i,k|k-1} - \hat{z}_k^-][\mathcal{Z}_{i,k|k-1} - \hat{z}_k^-]^T + \boldsymbol{R},\end{aligned} \tag{6.21}$$

where $\hat{z}_k^-$ denotes the predicted measurement, $\boldsymbol{P}_{z_k z_k}$ is its covariance, and $\boldsymbol{R}$ is the covariance of the observation noises. The state measurement cross-covariance $\boldsymbol{P}_{x_k z_k}$ and the kalman gain $\boldsymbol{K}$ are computed as

$$\begin{aligned}\boldsymbol{P}_{x_k z_k} &= \sum_{i=0}^{2L} \boldsymbol{W}_i^{(c)} [\mathcal{X}_{i,k|k-1} - \hat{x}_k^-][\mathcal{Z}_{i,k|k-1} - \hat{z}_k^-]^T,\\ \boldsymbol{K} &= \boldsymbol{P}_{x_k z_k} \boldsymbol{P}_{z_k z_k}^{-1}.\end{aligned} \tag{6.22}$$

The mean and covariance of the posterior estimation are then calculated as

$$\begin{aligned}\hat{x}_k &= \hat{x}_k^- + \boldsymbol{K}(z_k - \hat{z}_k^-),\\ \boldsymbol{P}_k &= \boldsymbol{P}_k^- - \boldsymbol{K}\boldsymbol{P}_{z_k z_k}\boldsymbol{K}^T.\end{aligned} \tag{6.23}$$

We also give to the tracker, the color histogram information $H_{tr}$ retrieved from the lower third of the bounding box, from which we expect to get clothing features[4]. In the implementation using a moving robot (see subsection 6.4.7), the camera movement is currently compensated by using the odometry of the robot. The use of *Kanade-Lucas-Tomasi* (KLT) features tracker [84] or the *visual odometry* [86] for compensating the camera movement are further investigated in the future.

#### 6.3.2.3 Association

We treat each detection of the human upper body as an observation, to be associated with the trackers. For every observation, we use the position and histogram information for calculating the

[4]In the upper body bounding box, the lower third part usually contains the human shoulder and body. We can expect the cloth color can be retrieved from this region and does not change much during the tracking process.

relative distance to each tracker $\Delta_{ob,tr}$, given by

$$\Delta^P_{ob,tr} = \frac{e^{-\frac{1}{2}(\bar{\mu}_{ob}-\bar{\mu}_{tr})^T(\Sigma_{ob}+\Sigma_{tr})^{-1}(\bar{\mu}_{ob}-\bar{\mu}_{tr})}}{\sqrt{2\phi|(\Sigma_{ob}+\Sigma_{tr})|}} \tag{6.24}$$

$$\Delta^H_{ob,tr} = \frac{\sum_I(H_{ob}(I)-\bar{H}_{ob})(H_{tr}(I)-\bar{H}_{tr})}{\sqrt{\sum_I(H_{ob}(I)-\bar{H}_{ob})^2\sum_I(H_{tr}(I)-\bar{H}_{tr})^2}} \tag{6.25}$$

$$\Delta_{ob,tr} = \eta\Delta^P_{ob,tr} + \varsigma\Delta^H_{ob,tr} \tag{6.26}$$

where $\Delta^P_{ob,tr}$ is the gaussian correlation between mean and covariance of the observed position $(\bar{\mu}_{ob}, \Sigma_{ob})$ and the tracker position $(\bar{\mu}_{tr}, \Sigma_{tr})$. $\Delta^H_{ob,tr}$ is color histogram correlation distance of the observation $H_{ob}$ and the tracker $H_{tr}$, and $\eta$ and $\varsigma$ are constants.

Under the nearest neighbor assumption, each observation is assigned to the tracker when the distance $\Delta_{ob,tr}$ is below the threshold. A new tracker is born from the observation which has no association, and any track which is not associated with any observation and has a large uncertainty is then deleted.

#### 6.3.2.4 ROI-based tracking

For reducing the calculation time, we do a hierarchical *Region of Interest* (ROI)-based tracking (including the orientation estimation) along the frame sequences. First, we search the whole space of the image to obtain initial observations and trackers. The next sequences utilize the position information from the trackers as the ROI. We do the detection around the ROI and in the area where persons may come up to the scene (see Fig. 6.5). This technique[5] considerably reduces the time for detection. Lastly, every 15 frames, we do the whole space search to anticipate the missing detection.

## 6.4 Experiments

Size of all images and camera sequences used in our experiments is $640 \times 480$. All of experiments were done using C++ implementation on a laptop PC (Core2Duo 2.1 GHz, 2GB memory, and Windows 7 OS).

[5]Basically, this heuristic method is enough for our cases. For more general scenarios, we open this problem to the interested reader.

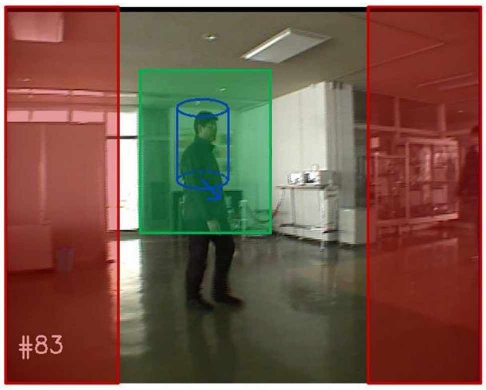


**Figure 6.5:** Region of Interest (ROI) for tracking. The red transparent shadows are places which has a high probability that person may appear to the scene from the side. The green transparent shadow is the ROI of the tracker. The blue cylinder denotes the current tracker bounding box.

The evaluation of our method starts from explaining the dataset used by our system. We then build an analysis to support the advantages of using our method for the human orientation estimation, by comparing it with several methods. We also evaluate performance of the orientation estimation and tracking integration. Lastly, we implement our system to the real environment using a camera attached on a mobile robot.

### 6.4.1 Dataset

First, we create our upper body dataset by cropping the INRIA [75] and Fudan-Penn [87] data into $48 \times 64$ pixel images containing the upper-half body of persons. We also add the CALVIN upper body dataset [88] into our dataset, so that we have 4250 positive samples of the human upper body. Around 3000 positive samples are used for training the upper body detector, and the rest is for testing purpose. 2500 negative samples are created from images which do not contain the human upper body, including the bottom part of the human body. From now, we refer it as *dataset A*[6].

To do a comprehensive test of the human upper body orientation estimation, we use several datasets for both static and dynamic scenes. For the static scenes, we use TUD-Multiview dataset [70] which is also used by the other state-of-the-art papers ([73], [72]). This dataset consists of 1486 images for training, 248 images for validation, and 248 images for testing. The TUD-Multiview dataset is annotated with bounding boxes of full body and eight orientation classes. For our purpose, we change the bounding boxes into the half upper part of the body.

We also use the dataset created for upper body training (*dataset A*), for the orientation clas-

[6]This generic naming is for simplicity only, not intended to give a new name to the existing databases.

sification purpose. We then separate the training samples of the above dataset into eight classes representing the eight orientation of the human body (see Fig. 6.1). The testing samples are treated in the same manner with the training samples. This dataset is then called as *dataset B*.

For the dynamic scenes, we use TUD-Stadmitte dataset [70] which contains 200 frames of the street scenes with several pedestrians crossing the street with a complex environment and many occlusions, and the camera calibration data[7]. Here we use the raw video for the TUD-Stadmitte dataset without annotations.

We then take an indoor video of our laboratory (from here, it is referred as *InLab dataset*). This video contains 487 frames with the number of persons varies from zero to three persons on each frame, and also the camera calibration data.

We summarize the usage of each dataset, as follows:

- *Dataset A* is used for training and testing the human upper body detection;
- *Dataset B* is used for evaluating the orientation estimation under various features and classifiers (see subsection 6.4.3 and 6.4.4);
- TUD-Multiview dataset is used for comparison with the state-of-the-arts (see subsection 6.4.6);
- Combination of *subset B* and TUD-Multiview datasets are then used for training the orientation model to be used in the dynamic scenes, i.e. TUD-Stadmitte and InLab datasets (see subsection 6.4.5), and the real world application (see subsection 6.4.7).

### 6.4.2 Human upper body detection performance

At the beginning, we want to show the reason for choosing the human upper body over the full body detection in our system. We build the human full body dataset from a subset of the INRIA dataset [75], and for the upper body dataset, we use the *dataset A* mentioned in subsection 6.4.1. We then create two testing samples; the first samples contain a subset of testing samples of the INRIA which do not contain occluded person, and in the second ones we add several images with persons occluded by the chairs, tables, and furnitures in additional to the testing samples of the

[7]Camera calibration data of the TUD-Stadtmitte dataset is provided at *http://www.gris.informatik.tu-darmstadt.de/∼aandriye/data.html*

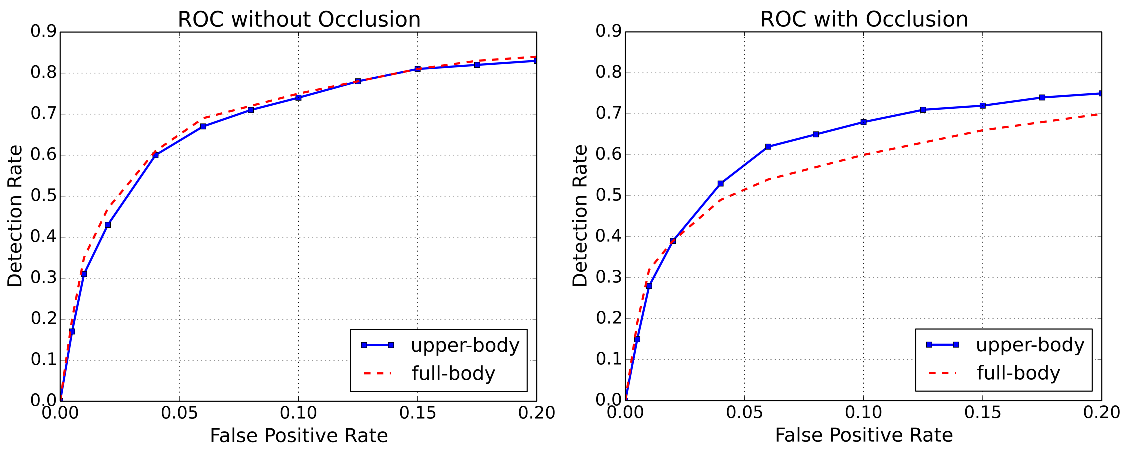


**Figure 6.6:** Performance of the human full body and upper body detection. The left graph shows the result on the dataset with no occlusion. the right graph shows the result on the occluded dataset.

INRIA. The training of both cases is then performed using *Boost-Cascade* method, as mentioned in the subsection 6.2.2. The ROC of both occluded and non-occluded cases are shown in Fig. 6.6.

The results show that the human upper body detector works better on the occluded dataset. This result suggests the use of the human upper body detection on the real cases such as an indoor environment with many tables, chairs, and furnitures, rather than using the full body model.

### 6.4.3 Evaluation of the orientation estimation using various features and classifiers

First, we conduct experiments to see the influence of the features, the models, and the classifiers to the orientation estimation results by using the *dataset B*. Following parameters are used for the experiments; we use the multi-level HOG and LBP features explained in the beginning of this chapter; for the multi-class SVM classifier [89], we use a standard RBF kernel with gamma set to 4e-4 , and regularization parameter is set to 1.0; for the MultiBoost [90], we use “FilterBoost" for the strong learner and “SingleStump" for the base learner; for the Random Forest, the number of trees is set to 100 and the maximum depth of trees is set to 25; the latent vector for CFM-PLS is set to 15, and for the BIFM-PLS, we set $p_1$ to 3 and $p_2$ to 15 (we will discuss these values of the PLS models in the next subsection).

Table 6.1 shows the experimental result using the combination of features and classifiers. Based on this table, the combination of multi-level HOG-LBP features, BIFM-PLS model, and Random Forest classifier, is superior against the others. In general, combination of HOG and LBP

Table 6.1: Evaluation of the orientation estimation using various features and classifiers

| Feature | Classifier | Accuracy | |
|---|---|---|---|
| | | TUD-Multiview | Dataset B |
| HOG | MultiSVM | 0.35 | 0.34 |
| HOG | MultiBoost | 0.37 | 0.38 |
| HOG | Random Forest | 0.44 | 0.46 |
| HOG+LBP | MultiSVM | 0.45 | 0.43 |
| HOG+LBP | MultiBoost | 0.48 | 0.45 |
| HOG+LBP | Random Forest | 0.53 | 0.52 |
| HOG+LBP+CFM-PLS | MultiSVM | 0.45 | 0.42 |
| HOG+LBP+CFM-PLS | MultiBoost | 0.50 | 0.47 |
| HOG+LBP+CFM-PLS | Random Forest | 0.54 | 0.54 |
| HOG+LBP+BIFM-PLS | MultiSVM | 0.60 | 0.57 |
| HOG+LBP+BIFM-PLS | MultiBoost | 0.60 | 0.58 |
| **HOG+LBP+BIFM-PLS** | **Random Forest** | **0.64** | **0.60** |

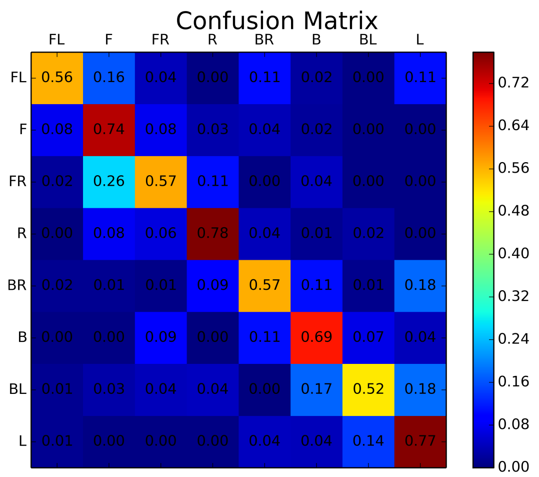


**Figure 6.7:** Confusion matrix on TUD-Multiview dataset using BIFM-PLS and random forest.

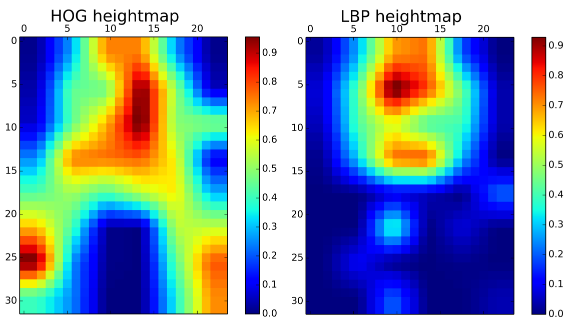


**Figure 6.8:** The block importance of each feature using the weight of the first projection vector. The left figure show the importance of one level block due to the HOG features. The right figure show the importance of one level block due to the LBP features. Red color represents the high importance.

features is better than using only HOG features. The RF classifier also gives better impact than using MultiSVM and MultiBoost. The interesting part is that the usage of PLS models yields better result of the orientation estimation than the concatenated features. We will discuss these matters in the next subsection.

Figure 6.7 shows the distribution of the orientation estimation. From this figure, we can conclude that estimating an oblique direction is more difficult than detecting the perpendicular one.

### 6.4.4 Analysis of the PLS models

We now discuss about how our PLS models (BIFM-PLS) give a significant contribution to the orientation estimation. We extract the block importance score (BIS) of each LBP and HOG features using the BIFM-PLS algorithm, and draw the weight of the first projection vector.

Figure 6.8 shows the contribution of each features to the orientation estimation. The left figure, which utilize the HOG features, shows that the areas around the edges of the body have a high importance, and the background tends to have a low importance. It means the HOG extracts the shape of the body for the orientation estimation. In the right figure, we can see the high importance area is around the head area and the body, but not for the background and the clothing. Here we can understand that the LBP captures the head features for the orientation estimation, while the clothing and background which are varies from one sample to the others are discarded.

Based on the Fig. 6.8, and supported by the results in the table 6.1, we show the effect of the BIFM-PLS model to the classification. Even the classifier such as the Random Forest is noted to be able to extract the importance of the features (for example, in [91] and [92]), here the BIFM-PLS removes the “noisy" areas (such as the various background and clothing) as shown in the Fig. 6.8,

Table 6.2: Performance of PLS and PCA for the orientation estimation

| Method | Accuracy | |
|---|---|---|
| | TUD Multiview | Dataset B |
| HOG+LBP+CFM-PLS+RF | 0.54 | 0.54 |
| HOG+LBP+BIFM-PLS+RF | 0.64 | 0.60 |
| HOG+LBP+PCA+RF | 0.52 | 0.51 |

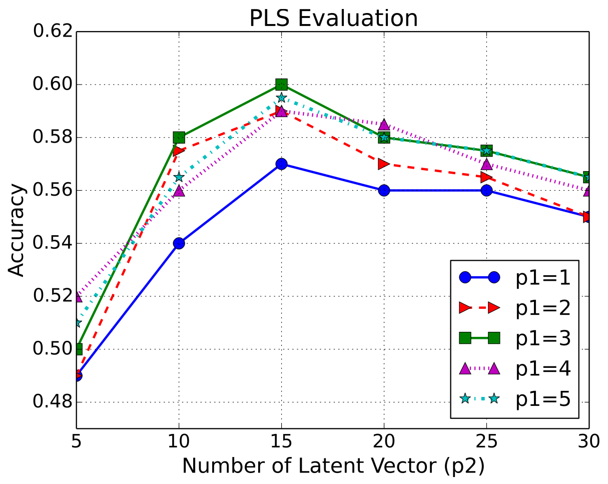


**Figure 6.9:** The effect of varying $p_1$ and $p_2$ value of the BIFM-PLS to the orientation estimation results.

and helps the classifier focus to do the classification on the high importance features.

We can also see the advantages of the PLS models as a dimensional reduction algorithm. We use another popular dimensional reduction algorithm, Principal Component Analysis (PCA), as the baseline. Table 6.2 shows the superiority of the PLS models against the PCA. It is understandable because unlike the PCA, the PLS also considers the class labels besides the variance of the samples.

We then discuss about the effect of the constants $p_1$ and $p_2$ used in the BIFM-PLS algorithm, shown by Fig. 6.9. $p_1$ represents how well a block contributes to the orientation estimation, and $p_2$ examines the contribution of a feature inside a block. By examining the Fig. 6.9, we choose the optimal value for both constants, $p_1 = 3$ and $p_2 = 15$. Over that values, we can consider it as the data overfit.

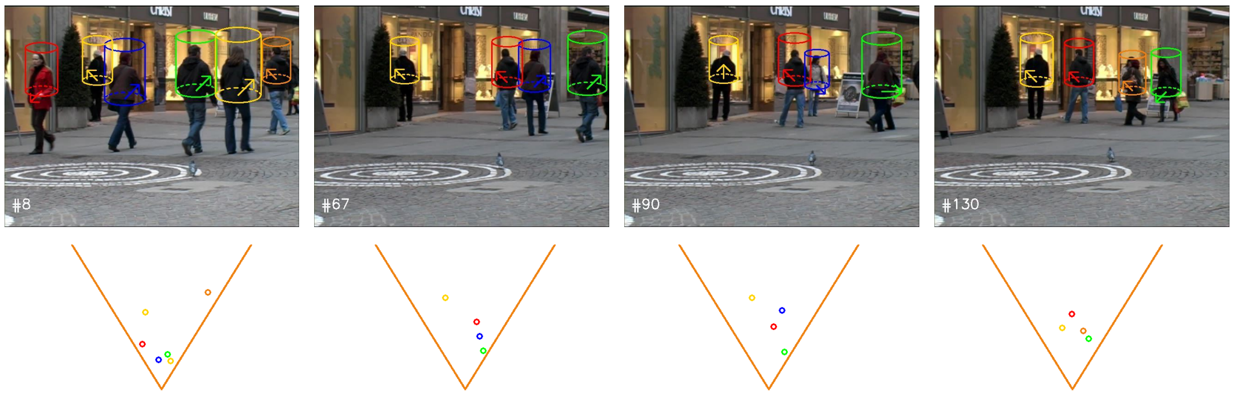


**Figure 6.10:** Human upper body orientation estimation results on TUD-Stadtmitte datasets using our proposed framework. Top sequence shows the detection (bounding boxes) and orientation estimation (arrows) results. Bottom sequence shows the estimated position of each detected person (circles) in the 3D world coordinate, with respect to the person position in the top images. Two orange line segments in the bottom sequences denote the camera FOV.

### 6.4.5 Evaluation on integrated orientation estimation and tracking performances

The next experiments are intended for evaluating the performance of the integrated orientation estimation and tracking. Here we use the TUD-Stadmitte and InLab datasets for the evaluations. We also investigate the influence of choosing various values of $\omega$ as mentioned in eq. 6.15. The higher value of $\omega$ means the object movement will give a higher influence to the orientation estimation result.

Figure 6.10 shows the performance of our proposed system using the TUD-Stadtmitte dataset. The top sequence shows the orientation detections and tracking results. The bottom sequence shows the person locations and movement predictions in the 3D world coordinate, relative to the position and FOV of the camera.

We then test our framework on InLab dataset, as shown in Fig. 6.11 (please refer to Fig. 6.10 for the figure properties explanation). Figure 6.11 exhibits the robustness of the multi target tracking and orientation estimation under many occlusions.

Figure 6.12 shows the effect of changing $\omega$ value to the overall orientation estimation results. We change the value of $\omega$ from 0, which means the orientation estimation depends only on the detection, to $\omega = 1$ which represents a heavy dependency to the object movements.

In the TUD-Stadmitte dataset, the orientation estimation based on the object movements ($\omega = 1$) is relatively high compared to the one based on the detection ($\omega = 0$), due to the consistent

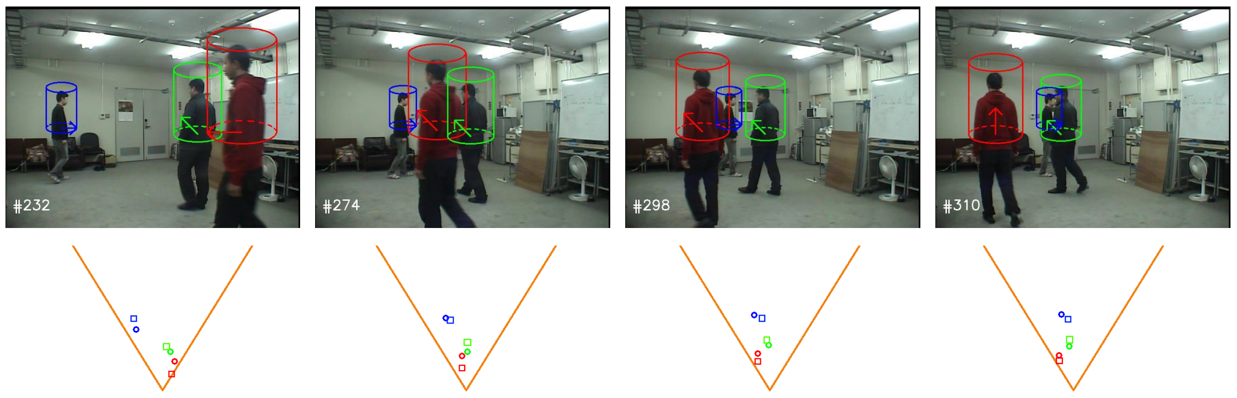


**Figure 6.11:** Human upper body orientation estimation results in an indoor environment. Top sequence shows the detection (bounding boxes) and orientation estimation (arrows) results. Bottom sequence shows the position of estimated each detected person (circles) and the ground truth (rectangles) in the 3D world coordinate, with respect to the person position in the top images. Two orange line segments in the bottom sequences denote the camera FOV.

movement of each person inside the video sequences, which gives a high confidence to the movement estimation. Contrary, the orientation estimation based on the object movement in the InLab dataset is lower because the persons frequently change their direction. Overall, combining both detection and movement estimation tends to make a higher orientation estimation results rather than solely depends on the detection or the movement estimation, and in our cases we choose $\omega = 0.5$.

### 6.4.6 Comparison with the state-of-the-art

We compare our algorithm to the state-of-the-art papers such as [70], [73], and [72] using the TUD-Multiview dataset. Since the state-of-the-art papers use the whole body information for estimating the orientation, we can not directly use the result of their papers as the comparison. Instead, we need to test their methods using the upper body information. Unfortunately, the author of those papers do not provide any implementation code.

To overcome the problem above, we re-implement their algorithms based on our understanding to their papers. We then evaluate it using the full body information. We expect the result will be similar with the one mentioned in their respective paper. The first and second columns of table 6.3 show the comparison of the original and the re-implemented version of the state-of-the-art papers. We can see that our re-implementation gives a close result to the one mentioned in each paper. Until this stage, our algorithm beats all of Andriluka's and Chen's works and comparable to Baltieri's work, even though they use the full body information. Now we can assume our re-implementation of the state-of-the-art can be used for further comparison.

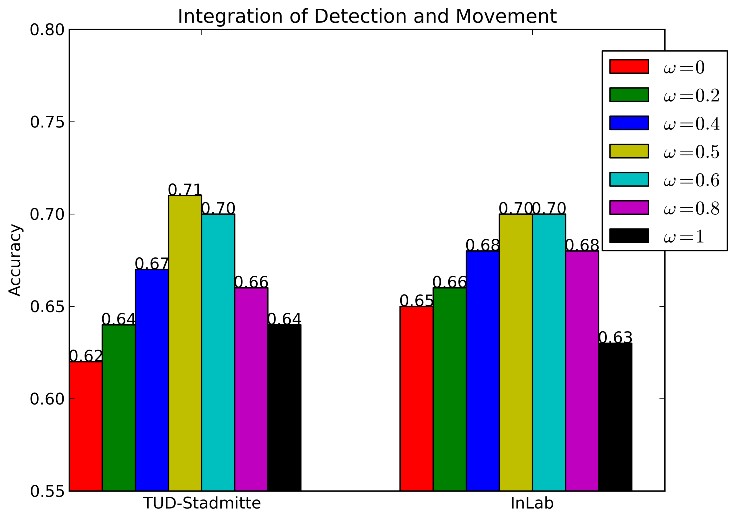


**Figure 6.12:** The effect of varying $\omega$ value to the integration of orientation results.

Table 6.3: Comparison of the state-of-the-art algorithms

| Method | Accuracy Full Body | | Accuracy |
|---|---|---|---|
| | Paper | Our Implementation | Upper Body |
| Andriluka [70] - Max | 0.31 | 0.29 | 0.20 |
| Andriluka [70] - SVM | 0.42 | 0.39 | 0.35 |
| Andriluka [70] - SVM_adj | 0.35 | 0.33 | 0.27 |
| Chen [73] - Sparse | 0.55 | 0.52 | 0.40 |
| Baltieri [72] - AWG | 0.65 | 0.59 | 0.51 |
| Ours | N/A | N/A | 0.64 |

We then use the re-implementation of the state-of-the-art for making a fair comparison with our algorithm, *i.e.* by applying the same upper body dataset which contains less information. Each algorithm is then trained and tested using the upper body version of the TUD-Multiview dataset. The last column of table 6.3 shows the result of each algorithm using only the upper body information. The result of each state-of-the-art degrades significantly, as less information is available for obtaining the body orientation. It also shows the importance of using various cues. Other works use only the shape features which decrease the performance when the information becomes less. On the other hand, our algorithm uses the shape and texture features simultaneously to overcome those problems and gains the best performance on the upper body orientation estimation.

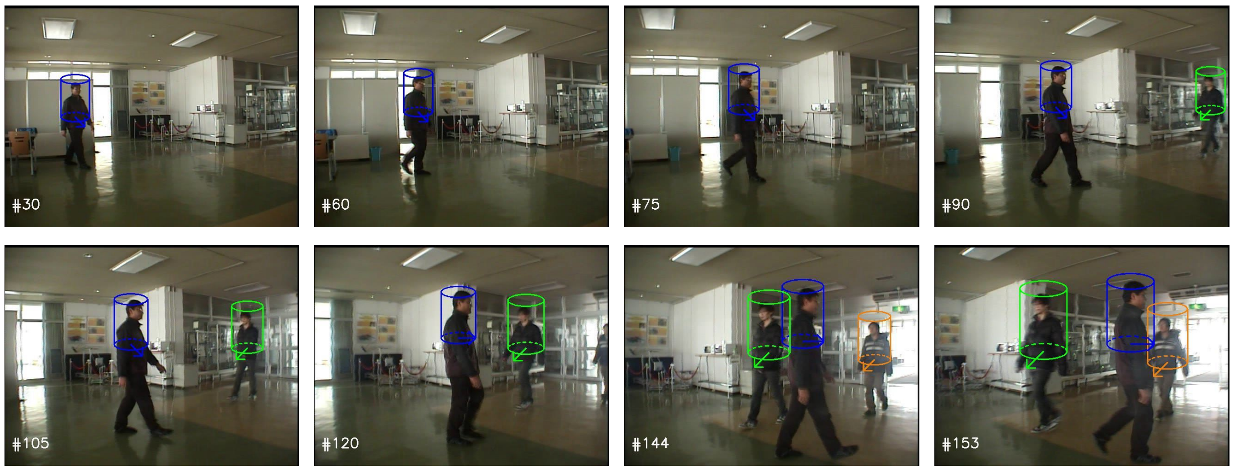


**Figure 6.13:** Human upper body orientation estimation results in the cafetaria with a moving camera.

### 6.4.7 Orientation estimation on moving camera

In this experiment, we attach a monocular camera on the mobile robot base. The robot is then controlled to move while performing the human upper body orientation estimation and tracking in real-time. The experiment was performed at the university cafeteria, with the total of 190 frames. Figure 6.13 shows the experimental results using the camera with a moving base. We can achieve the accuracy rate of 0.70 and the frame rate of 5-12 Hz, fast enough to be used for an on-line purpose. Once again, it shows the consistency and robustness of our orientation estimation and tracking system.

# Chapter 7

# Implementation of Viewpoint Algorithm for The Guard Robot

The implementation of our viewpoint planning algorithm for the guard robot is done on a Windows PC (Core i7 2.4 GHz, 16 GB RAM) using C++ programming language. The viewpoint planner as well as the supporting algorithms (e.g. robot controller, localization, local planner, etc.) are implemented in a distributed manner as RT-Components, which are software modules running on RT-Middleware [67] environment. MRPT library [93] is also used mainly for visualization purposes. The proposed algorithm's feasibility is then evaluated by both simulations and the real experiments. We also provide a comparison with the person following algorithm to clearly certify the benefit of our viewpoint planner.

For both simulations and the real experiments, our viewpoint planner runs in two stages: off-line and on-line stages. In the off-line stage, the map data is acquired from a SLAM algorithm [66]. We then retrieve viewpoints from the map using the algorithm mentioned in chapter 3. These viewpoints are subsequently utilized to make a global plan for the robot in the on-line stage. The action yielded by the global plan are accordingly executed by a local motion planner [15].

## 7.1 Simulations using a realistic 3D simulator

A Linux PC (Core i5, 2.1 GHz) is utilized for running the 3D simulator [94] and interconnected to the viewpoint planner through a socket communication [95]. All sensor data such as laser range measurement and the robot odometry are simulated and taken from the simulator, resembling the real condition. For the target person, we simulate and administer a predefined path for a human object, such that it will move continuously and independently around the simulated environment.

We arrange three different environments for conducting simulations. The first map imitates the real first floor of ICT building at our university (see Fig. 7.1a) with a slight modification. The second map represents a more challenging environment where it has several rooms and holes (see

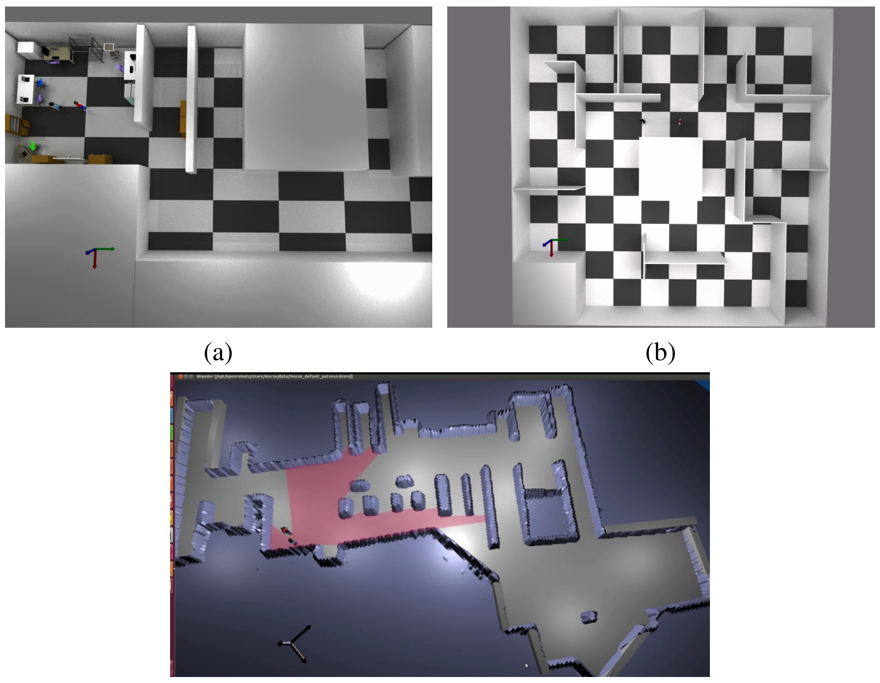


**Figure 7.1:** Environment used for simulations: (a) the 1st floor of ICT building (environment A), (b) generic complex map (environment B), (c) the A building at our university (Complex Map).

Fig. 7.1b). The third map is a 3D elevated map taken from the SLAM (Simultaneous Localization and Mapping) data of the A building at our university, which is the most challenging among all maps (see Fig. 7.1c). Table 7.1 displays parameter settings used in the simulations.

Figure 7.2 shows the viewpoints obtained by our viewpoint extractor. The viewpoints mainly reside at the middle of a room, or at the intersections connecting several corridors. It justifies our assumptions in chapter 3, declaring such places are suitable for the robot to have a wide visibility in a longer time.

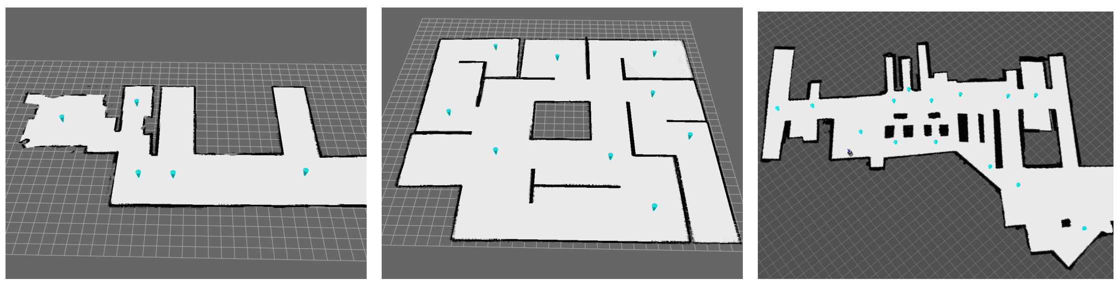


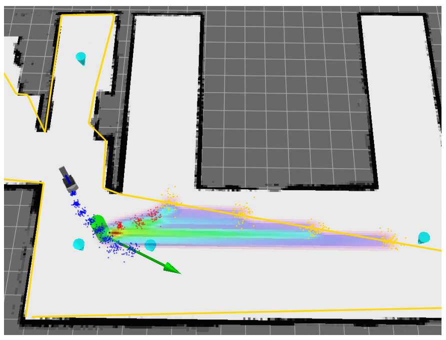
**Figure 7.2:** Viewpoints obtained by the viewpoint extractor, shown by cyan cones.

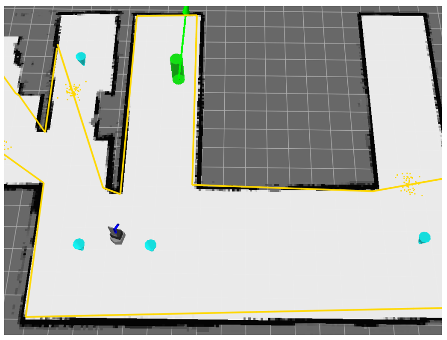
Table 7.1: Parameter settings

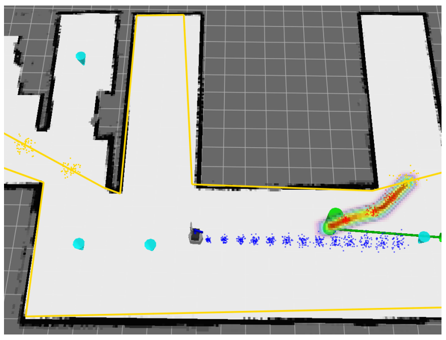
| Parameters | Value |
|---|---|
| Person velocity | 0.8 m/s |
| Robot max velocity | 0.7 m/s |
| Simulation length<br>Env A<br>Env B<br>Complex Map | <br>1504 steps<br>473 steps<br>848 steps |
| Time step | 0.25 s |
| $N$ (number of particles) | 50 |
| $\gamma$ | 0.05 |
| $\eta$ | 5.0 |

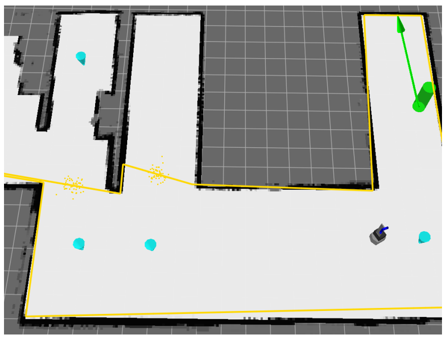
The simulation results are then exhibited by Fig. 7.3, 7.4, and 7.5, respectively for the environment A and B, and the Complex Map. For each figure, the top row represents the viewpoint planning results and the bottom row shows its associated states in the 3D simulator. When the target person is predicted to go through the escaping gaps (e.g. Fig. 7.3a, 7.3c, 7.4a–7.4c, 7.5a, and 7.5c), our algorithm makes a plan for the robot to move towards the viewpoint which covers those escaping gaps. On Fig. 7.3b, 7.3d, 7.4d, 7.5b, and 7.5d when the target person enters a “dead end" area, instead of following closely behind the target person, the robot is staying near the viewpoint to observe it. It implies our viewpoint planner is able to predict the situation that the target person will unlikely escape from the robot visibility. It also clearly distinguishes our method from the ordinary person tracking algorithm.

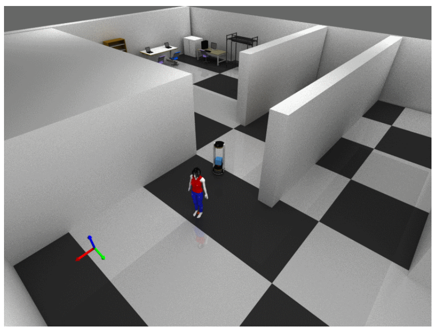
One may wonder that the robot is not staying at the viewpoint when it is in an idle condition (see Fig. 7.3b, 7.3d, 7.4d, 7.5b, and 7.5d). It actually justifies the main purpose of our viewpoint planner, i.e. to reduce the movement. The robot does not have to exactly “reach" the designated viewpoint as long as it is safe enough to keep the target person under its visibility. Thanks to the iterative visibility bound checking given in eq. (4.25) and (4.28), our proposed method can evaluate and predict the target person states and the eligibility of the robot to move or to be idle time-by-time, ensuring the optimality of the planner.

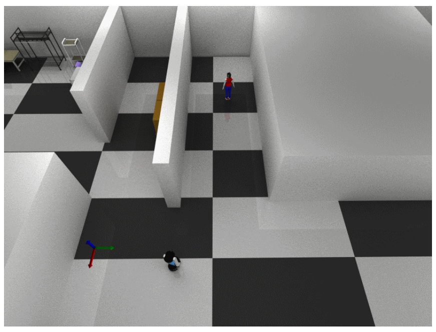

(a)

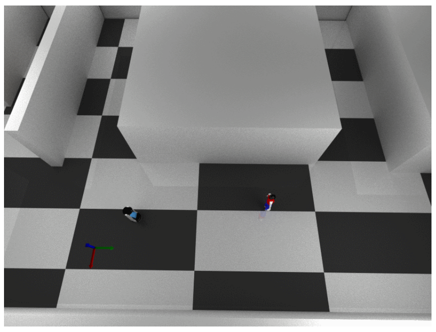

(b)


(c)


(d)


(e)


(f)


(g)

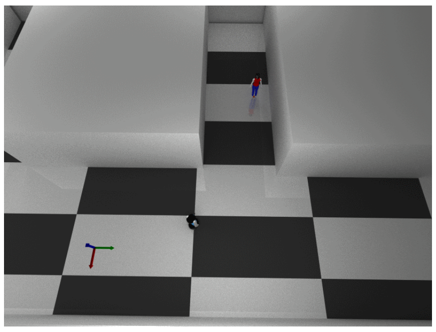

(h)

**Figure 7.3:** Executing viewpoint planner on environment A. Top row shows the viewpoint planning results. Bottom row shows its corresponding states on the 3D simulator. For the top row figures, the green cylinder with arrow represents the target person and its moving direction. The jetmap (colored cost map) represents the predicted future movement of the person. Cyan cones denote the viewpoints. The black object with a sequence of blue particles represents the robot and its movement controls. Yellow lines show the visibility polygon of the robot.

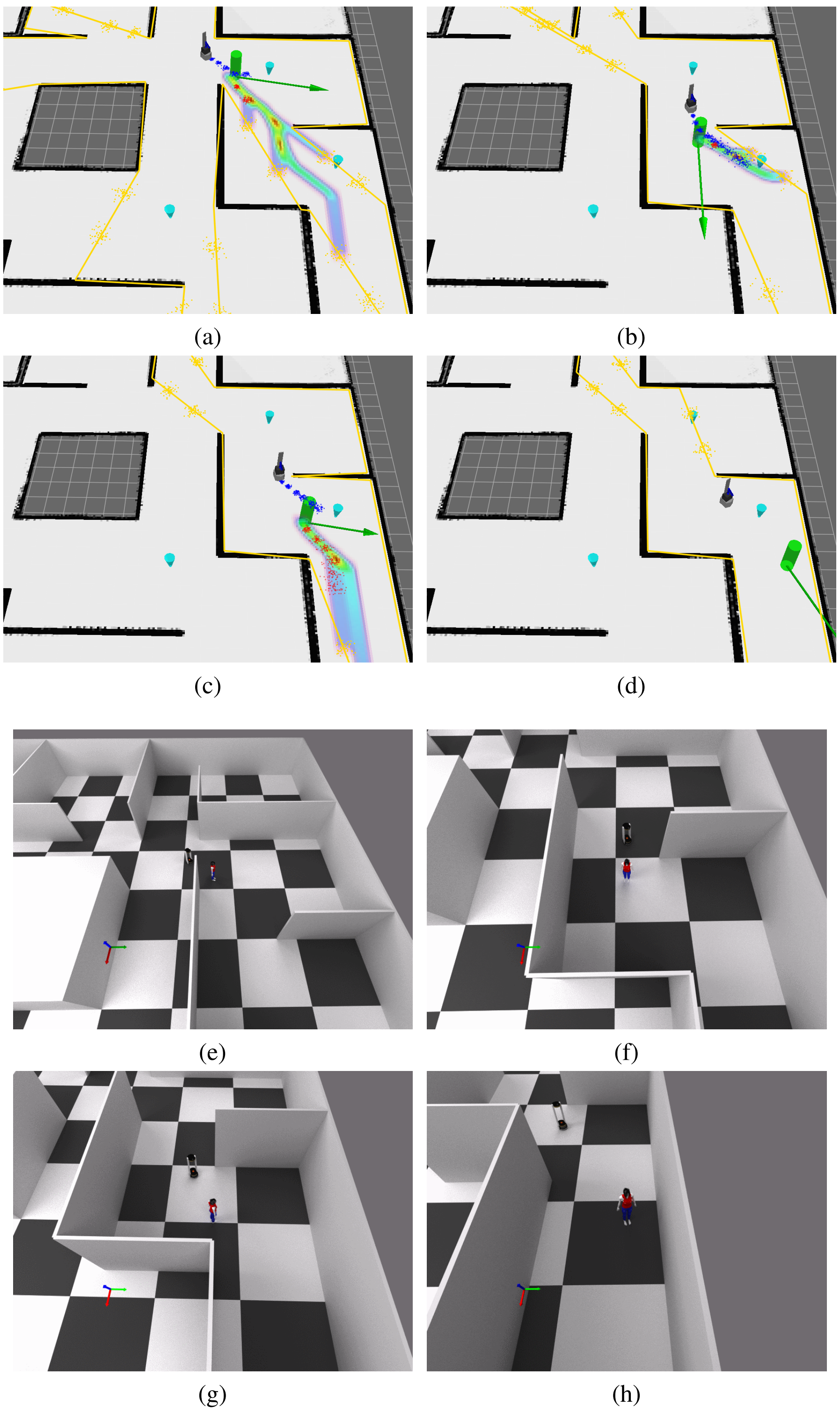


**Figure 7.4:** Executing viewpoint planner on environment B. Here, the same figure information on Fig. 7.3 is applied too.

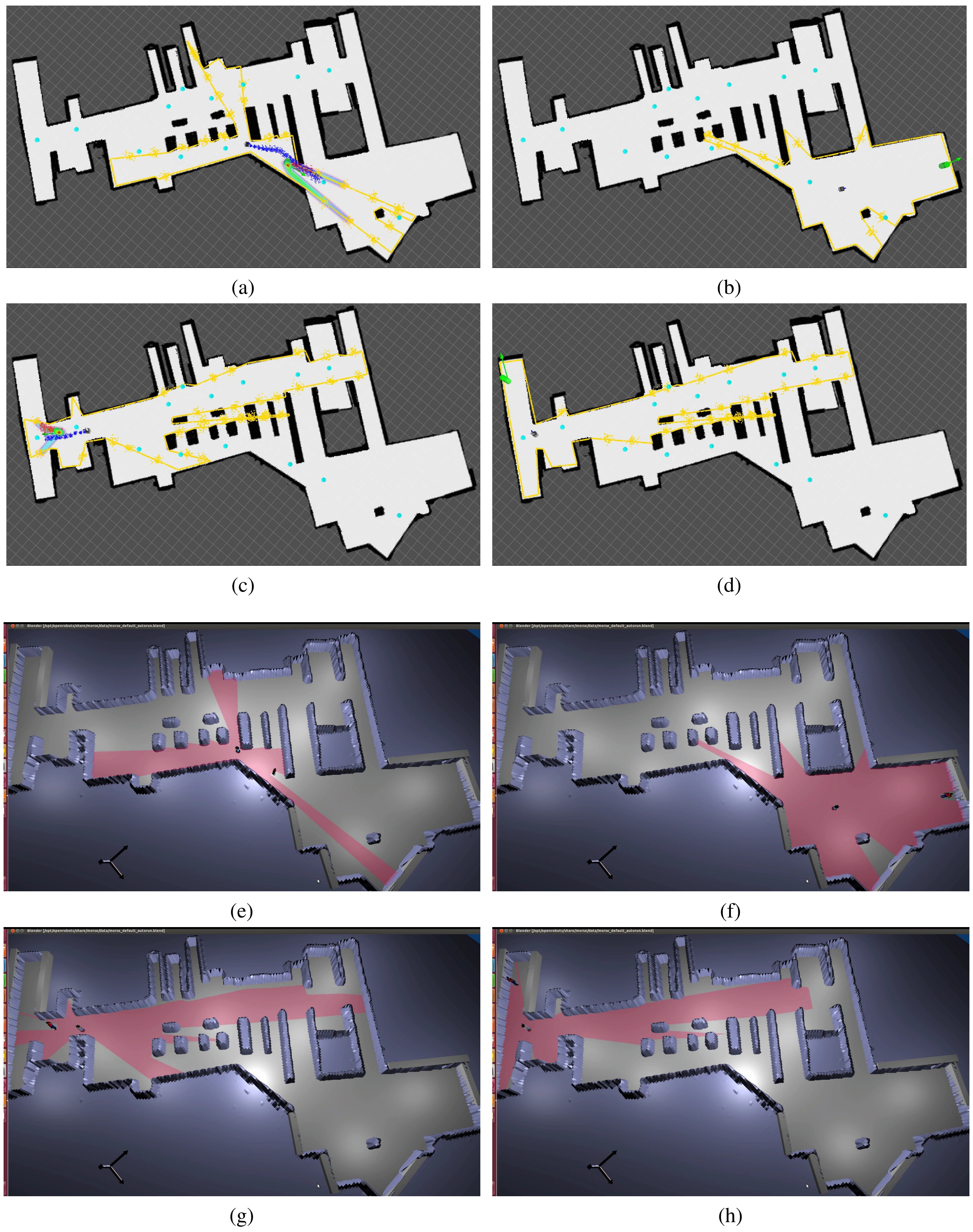


**Figure 7.5:** Executing viewpoint planner on Complex Map. Here, the same figure information on Fig. 7.3 is applied too.

## 7.2 Comparison with the person following algorithm

To get a picture about the effectiveness of the robot movement, we compare performance of our viewpoint planner with the ordinary person tracking algorithm as described in [15], under condition that the exactly same path and behavior of the target person are used. Figure 7.6 exhibits the recorded velocity profiles of the robot during the runtime for both algorithms. It can be noted that our viewpoint planner produces longer idle condition compared with the person tracking algorithm, indicated by longer zero velocity. It means the robot has an effective movement for observing the target person, without always closely follows him. In the environment B and Complex Map, the idle condition is not as much as the one in the environment A, since the target person mainly moves along corridors rather than any dead end or rooms.

We then quantitatively compare the used energy for both viewpoint planning and the ordinary person tracking algorithms, as well as our previous method [17], using following metric,

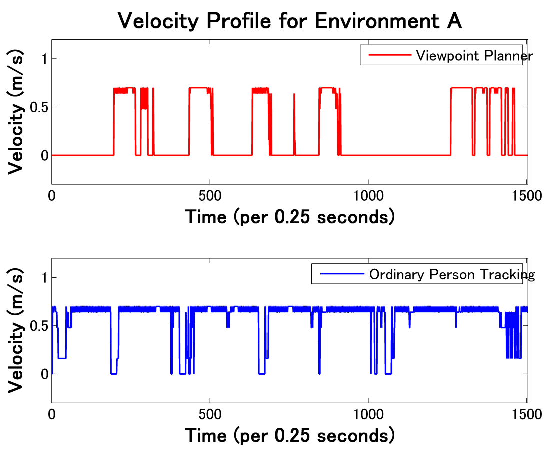
$$Energy = \sum_{t=0}^{T} (u_t)^2, \tag{7.1}$$

where $u_t$ denotes the control applied to the robot, and $T$ is total time for one simulation.

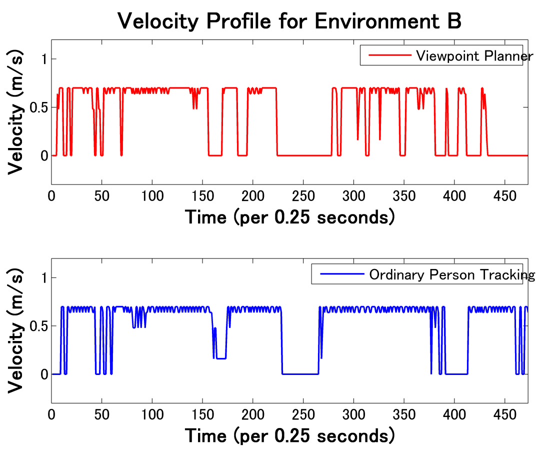
From the above metric, we expect to capture the total movement of the robot. Table 7.2 displays the average energy for each algorithm after five simulations. It clearly shows the benefit of our proposed algorithm which is able to reduce the movement and leads to the energy saving. Our proposed algorithm also has a slightly better performance compared with our previous approach.

Table 7.2: Comparison of energy used by the robot (lower is better)

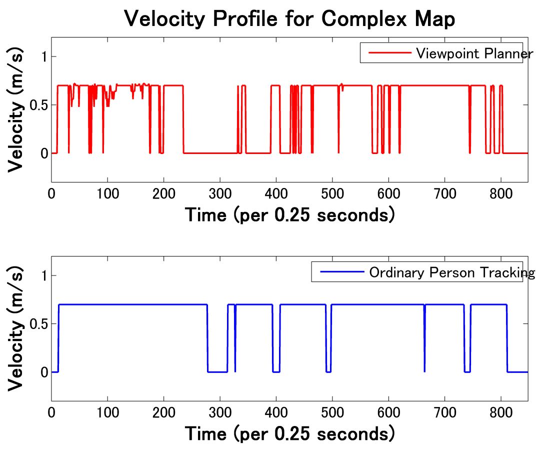
| | Proposed Viewpoint Planner | Person Tracking Algorithm | Previous Approach [17] |
|---|---|---|---|
| Env A | 205.53 | 592.91 | 220.73 |
| Env B | 130.06 | 166.96 | 135.15 |
| Complex Map | 268.86 | 356.23 | 278.41 |



(a)



(b)



(c)

**Figure 7.6:** Comparison of controls yielded by viewpoint planner and ordinary person tracking algorithm for : (a) environment A, (b) environment B, (c) Complex Map.

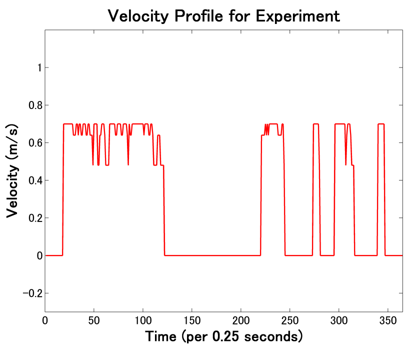


**Figure 7.7:** Velocity profile of the real experiment.

## 7.3 Experiments on a real environment

Finally, we evaluate performance of the proposed guard robot algorithm using the real robot on a real environment. A Pioneer 3DX robot equipped with laser range sensors, a camera, and a laptop PC is used. The laptop PC is basically utilized for acquiring the sensor data, sending commands to the robot, and distributing the workload. It is then connected to the main PC running the viewpoint planning algorithm through the same communication manner as the one in simulations. The parameter setting is also the same as the simulation, except for the person velocity which depends on the tracker. For obtaining the target person data, an image-based [96] and laser-based person tracker were used.

We carry out the experiment at the ICT building of our university. Figure 7.8a exhibits the obtained map of the building. As shown in Fig. 7.8d, the robot only needs to “stay” while observing the target person, since it is predicted not to leave the robot visibility. This behavior of the robot is also confirmed by the velocity profile given in Fig. 7.7, where the robot does not always move to closely follow the target. The ability to keep the view towards the target person is then proved by the bottom row of Fig. 7.8. This experiment result also demonstrates the feasibility of our algorithm to be used in real-time.

## 7.4 Experiments with Partial Occlusion

Up to now, we always assume a homogeneous visibility, i.e. every detected obstacle is treated as a “wall" regardless of its types and characteristics. It happens since we use a laser-based SLAM for building the map. As the consequence, the robot cannot distinguish the type of obstacles, especially its height, encountered on the generated map.

In a realistic setting, the robot may deal with the following circumstances; Tables or other low objects block the robot’s path. In this case, the target person may be actually visible from the robot, but the generated map assumes that it is occluded. If this happens, the robot which actually can stay to see the target person is then forced to move due to the occlusion’s misinterpretation by the viewpoint planner. We then offer a solution to deal with such problem.

We take advantage the fact that we use the geodesic motion model on the on-line stage of the viewpoint planning framework, which does not affect the viewpoint extraction on the off-line stage. It means if a non-obscuring obstacle appears, it will not give any effect to the viewpoint extraction

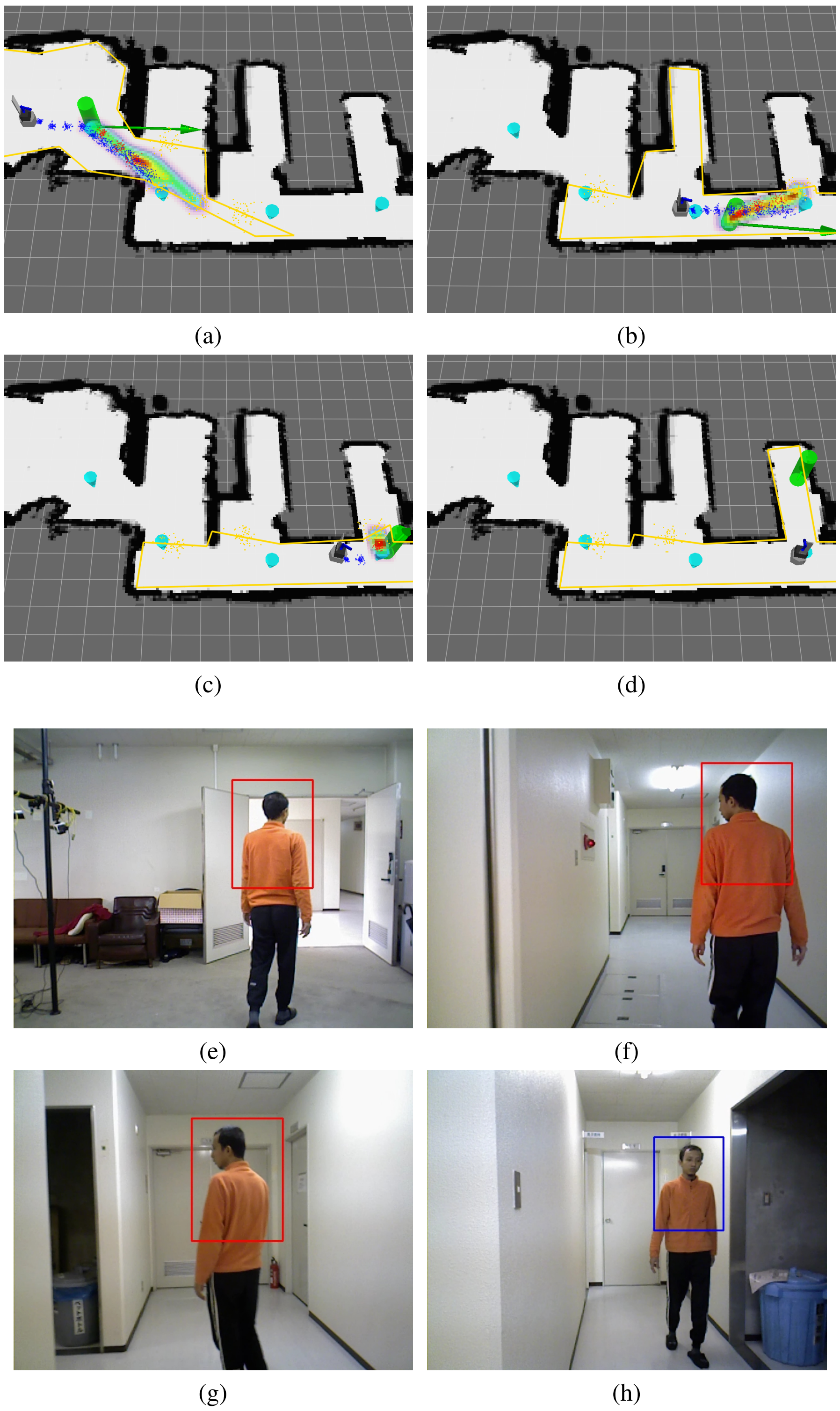
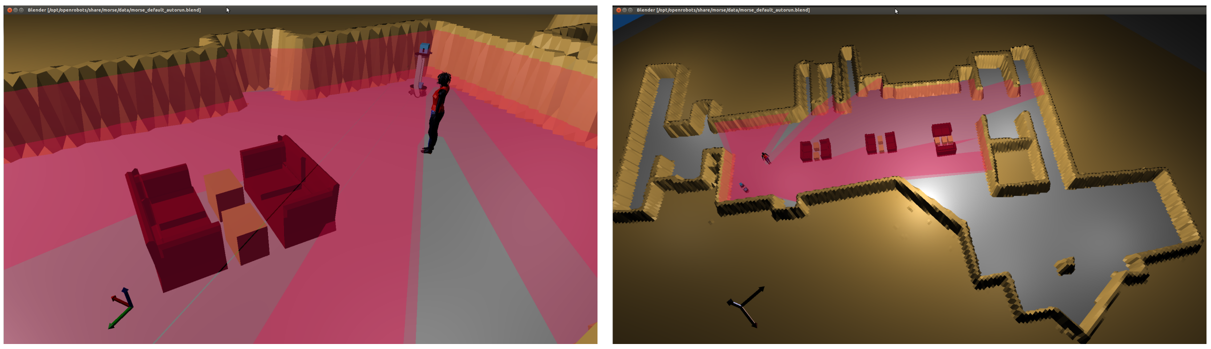

(a) (b) (c) (d) (e) (f) (g) (h)

**Figure 7.8:** Executing viewpoint planner on the real environment. Top row shows the planning results on the map. Bottom row shows the target person view from the robot, associated with the plan on the top row.



(a) (b)

**Figure 7.9:** Two-level laser range sensors for solving the occlusion problems: (a) Orthographic view of the laser placement, (b) Bird-view of the two-level laser system, where the bold magenta area is the visibility of the bottom laser and the mild magenta is the visibility of the top laser.

results and the visibility constraints (viewing task). Rather, it will only change the predicted time of losing the target on the on-line stage, as well as the path for the robot to move (navigation task). It suggests both viewing and navigation tasks can be separated. Under such characteristics, our idea is to adopt two separated maps, “viewing" and “driving" maps, respectively parameterize what the robot sees (i.e. the visibility map from which we extract the viewpoints and watch the target) and what the robot actually faces (i.e. the navigational map containing non-obscuring obstacles on which we calculate the predicted time of losing the target).

We first build separated “driving" and “viewing" maps, by putting two-level laser range sensors onto the robot. The driving map is made by employing the bottom laser sensor, while the viewing map is created using the top laser sensor placed in parallel with the camera. Figure 7.9 shows the placement of both lasers.

We then conduct a simulation using a complex map which is modified from Fig. 7.1c, by replacing some obstacles in the middle of the map with tables and chairs (see Fig. 7.10). The robot uses the laser configurations as explained above, and creates both driving and viewing maps as shown in Fig. 7.11. To make sure that the viewpoints extracted from the viewing map do not lie inside the obstacles on the driving map, we set the cost function in eq. 3.12 such that the obstacles on the driving map will gives a high cost.

Figure 7.11 clearly shows the advantage of separating the viewing map from the driving map, where the first creates more compact viewpoints than the later. The smaller number of viewpoints and the low obstacles negligence on the viewing map give the robot a wider visibility. It subsequently allows the robot to move efficiently over the smaller set of points and stay watching the target in a longer time, as shown in Fig. 7.10a – 7.10d. Table 7.3 justifies the efficiency of using

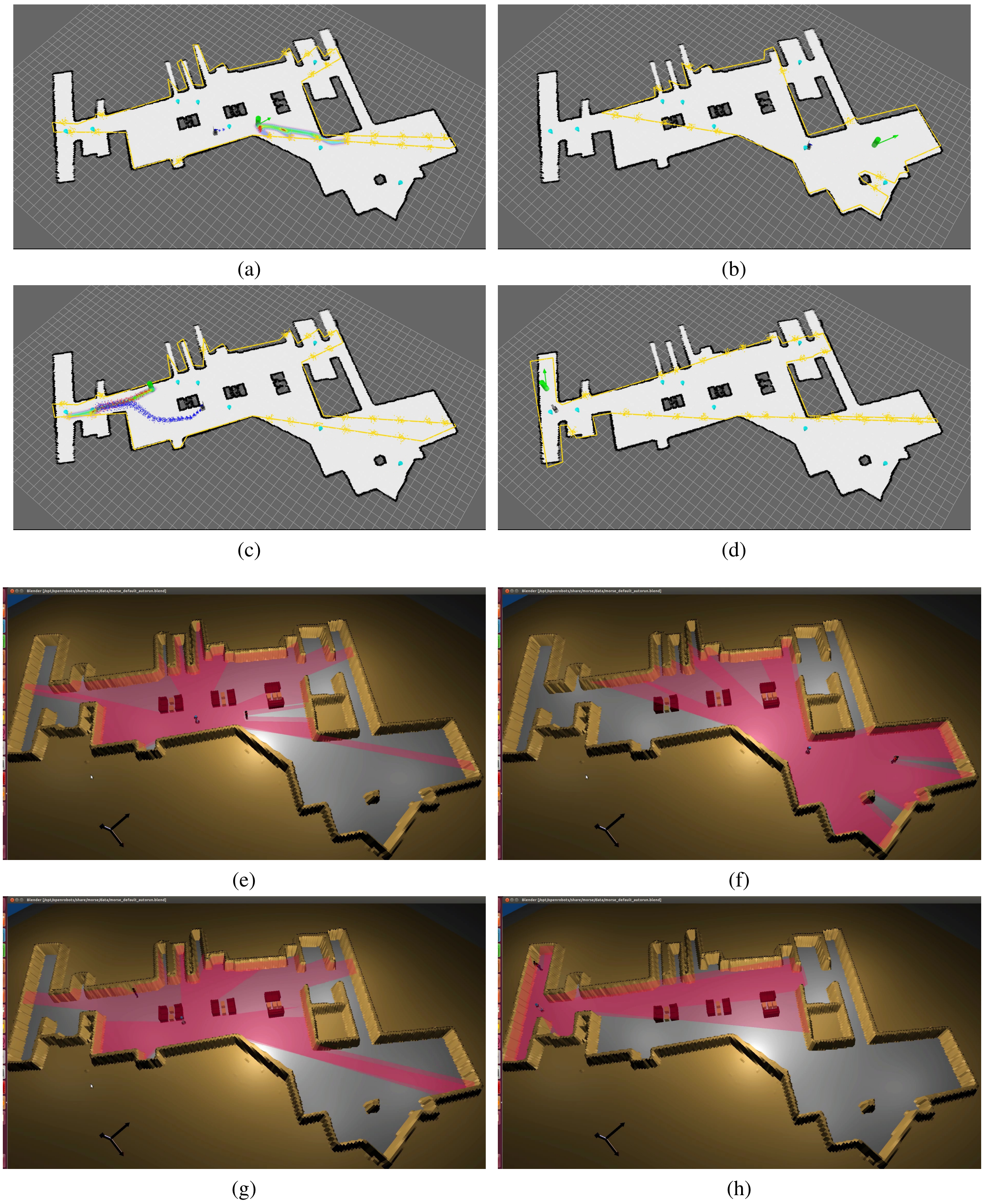


**Figure 7.10:** Executing viewpoint planner with partial occlusion. Here, the same figure information on Fig. 7.3 is applied too.

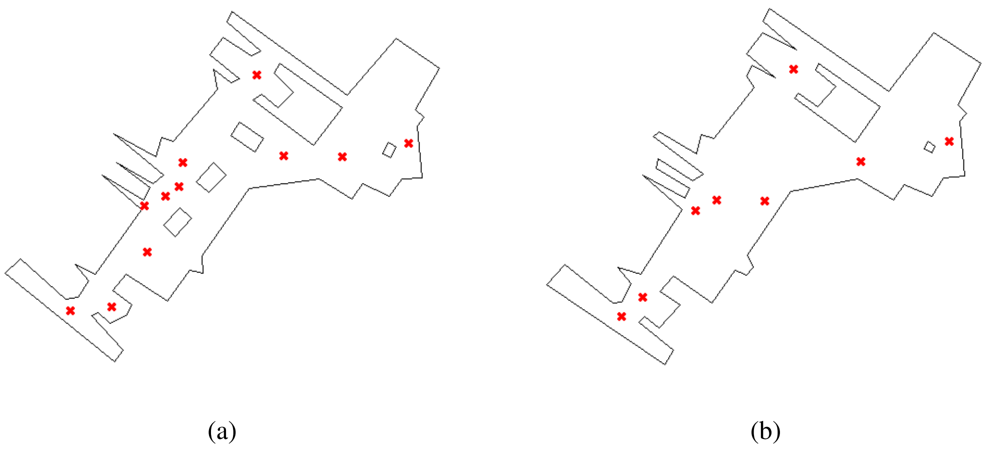


**Figure 7.11:** Comparison of viewpoints generated for: (a) Driving map (using bottom laser), (b) Viewing map (using top laser). Notice that tables and chairs are detected on the driving map, but not on the viewing map.

separated maps in the terms of the energy usage by the robot. As an additional credit point, we even do not need to change the viewpoint planning framework to cope with the partial occlusions, except for employing a new configuration of the laser range sensors and maps.

Table 7.3: Comparison of energy usage by the robot

| | Energy Cost (according to eq. (7.1)) |
|---|---|
| Separated Maps | 294.65 |
| Non-separated Map | 310.78 |

# Chapter 8

# Conclusions and Future Work

## 8.1 Conclusions

We have described a novel viewpoint planning algorithm for the guard robot problem. We utilize the topology of the environment to make an effective movement of the robot, by extracting a set of topology-based viewpoints from the environment. A geodesic motion model is also used for predicting both the robot and the target person movement. Two approaches of the viewpoint planning, a greedy and stochastic methods, are then exhibited. The greedy approach employs a deterministic planning using cost minimization, while the stochastic method utilizes a chance constraint-based bound for verifying the optimal planning and its safety to keep the target person under surveillance. Supported by the motion planning and person tracking algorithms, simulation and experiment results show that the proposed algorithm can reduce the movement of the robot, thereby saving the energy used by the robot.

As a supporting block in the viewpoint planning framework, we have presented a novel path planning algorithm which uses the arrival time field as a bias for a randomized tree search. Heuristic approaches of our algorithm are proved to be effective for handling a dynamic environment and kinematic constraints of the robot. We have shown that our algorithm is superior to other existing path planner algorithms. Simulation and experimental results also show that our algorithm is applicable to the real robot, and can be used in real-time.

We have also described an orientation estimation and tracking method to be used as another supporting block in the framework. Our human upper body orientation classification system, utilizing a partial least squares model-based shape-texture features combined with the random forest, is proved to work better than any existing methods. Its integration with the tracking system boosts up the performance of the orientation estimation even further. Another notable result is that our system works real-time, giving a possibility to be used in the real robot application such as the person tracking.

To summarize, we contribute several novelties related to each portion of the viewpoint planning framework. We are raising a distinctively new variant of robotic problem, which includes the problem of keeping a target under the robot visibility while reducing its movement. We then successfully establish a generalized algorithm for solving the coverage area under arbitrary cost function, which raises a unified solver for tackling several applications (e.g. *Art Gallery Problem*, *Sensor Placement*, and *Robot Coverage*). We also introduce a potential based randomized tree algorithm for robotic motion planning, which combines a high-exploration ability of the randomized tree and an arrival time field and heuristics to achieve the path optimality, safety, and applicability to the real robot. Lastly, we make an incremental improvement to the person tracking algorithm by applying an upper body-based detection and tracking for ensuring its robustness.

## 8.2 Future Work

Since our guard robot problem is theoretically new, there are still many rooms for the research improvement in the future. First, there is a possibility to use the viewpoint planner implementation for building a map on an unknown environment (map exploration) effectively. By updating the viewpoints over the growing map, it is expected to guide the robot to visit unexplored area efficiently, which also considers the energy usage.

The second possibility is to properly model the person motion, considering the context of the environment. By learning how usually the person moves in a certain place, we can capture the human intention and its motion tendency on each environment structure. It may help to improve the motion prediction quality of the viewpoint planner.

The current implementation of our viewpoint planner strictly considers only one person. Dealing with multiple persons and their interaction may become an attractive case for the future direction of the guard robot framework. For example, knowing the interaction between the target person and other persons in a building can help the viewpoint planner to determine a proper action for the robot, such as changing the viewpoint angle toward the target person so that we can record his interaction with the others.

Other possible interesting future researches will be the implementation for outdoor problem cases. Since our proposed method considers the environmental topology, it is possible to do the same approach by carefully drawing out the information of an outdoor scene, such as road structures, park shapes, etc, and subsequently extracting its interesting viewpoints. The proposed viewpoint planning algorithm can be then applied in the same manner.

The use of Unmanned Aerial Vehicle (UAV) which has more degree of freedom to freely observe the target person is also a plausible direction of this research. By utilization of an UAV, it may break any constraint posed by the mobile robot, such as movement restriction, uneven ground plane, or the scattered small objects.

From the coverage problem perspective, there are some open problems which can direct the future of the research. One of them is to include the field-of-view (FOV) limitation of the guards into the system. This constraint is very interesting since it will broaden the type of guards which can be used in the framework, e.g. we can use any sensor or camera widely sold in the market for setting up the surveillance system. Another possible direction is to find an exact formulation and solution for the Generalized Coverage Problem.

For the person tracking problem, the possible future research is to combine it with other sensors such as laser range finders. By adopting multi-sensory fusion, we expect to build a more robust system for person localization. There are also possibilities for choosing and adding better heuristics in the integration of the orientation estimation and tracking, such as a better way for selecting the region-of-interest, so that the system will be applicable for more general scenarios.

# Bibliography

[1] J. O'Rourke, *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

[2] V. Pinciu, "A coloring algorithm for finding connected guards in art galleries," in *Discrete Mathematics and Theoretical Computer Science*, vol. 2731 of *Lecture Notes in Computer Science*, pp. 257–264, Springer Berlin Heidelberg, 2003.

[3] L. Erickson and S. LaValle, "An art gallery approach to ensuring that landmarks are distinguishable," in *Proc. of Robotics: Science and Systems (RSS)*, (Los Angeles, USA), 2011.

[4] A. Bottino and A. Laurentini, "A nearly optimal algorithm for covering the interior of an art gallery," *Pattern Recognition*, vol. 44, no. 5, pp. 1048–1056, 2011.

[5] J. Durham, A. Franchi, and F. Bullo, "Distributed pursuit-evasion with limited-visibility sensors via frontier-based exploration," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 3562–3568, 2010.

[6] V. Isler, S. Kannan, and S. Khanna, "Randomized pursuit-evasion in a polygonal environment," *IEEE Trans. on Robotics*, vol. 21, no. 5, pp. 875–884, 2005.

[7] K. Klein and S. Suri, "Capture bounds for visibility-based pursuit evasion," *Computational Geometry*, vol. 48, no. 3, pp. 205–220, 2015.

[8] L. Guilamo, B. Tovar, and S. LaValle, "Pursuit-evasion in an unknown environment using gap navigation trees," in *Proc. of IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 3456–3462, 2004.

[9] B. Gerkey, S. Thrun, and G. Gordon, "Visibility-based pursuit-evasion with limited field of view," *International Journal of Robotic Research*, vol. 25, no. 4, pp. 299–315, 2006.

[10] S. Carlsson, H. Jonsson, and B. J. Nilsson, "Finding the shortest watchman route in a simple polygon," in *Algorithms and Computation*, vol. 762 of *Lecture Notes in Computer Science*, pp. 58–67, Springer Berlin Heidelberg, 1993.

[11] S. Ntafos, "Watchman routes under limited visibility," *Computational Geometry*, vol. 1, no. 3, pp. 149–170, 1992.

[12] X. Tan, "Approximation algorithms for the watchman route and zookeeper's problems," *Discrete Applied Mathematics*, vol. 136, no. 2–3, pp. 363–376, 2004.

[13] N. Bellotto and H. Hu, “People tracking with a mobile robot: a comparison of kalman and particle filters,” in *Proc. of the 13th IASTED Int. Conf. on Robotics and Applications*, 2007.

[14] D. Schulz, W. Burgard, D. Fox, and A. Cremers, “People tracking with mobile robots using sample-based joint probabilistic data association filters,” *Int. Journal of Robotic Research*, vol. 22, no. 2, pp. 99–116, 2003.

[15] I. Ardiyanto and J. Miura, “Real-time navigation using randomized kinodynamic planning with arrival time field,” *Robotics and Autonomous Systems*, vol. 60, no. 12, pp. 1579–1591, 2012.

[16] R. Liu, G. Huskic, and A. Zell, “Dynamic objects tracking with a mobile robot using passive uhf rfid tags,” in *Proc. of IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4247–4252, 2014.

[17] I. Ardiyanto and J. Miura, “Visibility-based viewpoint planning for guard robot using skeletonization and geodesic motion model,” in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, (Karlsruhe, Germany), pp. 652–658, 2013.

[18] S. Suzuki and K. Abe, “Topological structural analysis of digitized binary images by border following,” *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[19] D. H. Douglas and T. K. Peucker, “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,” *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[20] S. L. Devadoss and J. O’Rourke, *Discrete and Computational Geometry.* Princeton University Press, 2011.

[21] J. Urrutia, “Art gallery and illumination problems,” in *Handbook of Computational Geometry* (J. Sack and J. Urrutia, eds.), pp. 973–1027, Elsevier, 2000.

[22] H. González-Banos and J. C. Latombe, “A randomized art-gallery algorithm for sensor placement,” in *Proc. of the 17th Annual Symposium on Computational Geometry*, pp. 232–240, 2001.

[23] P. K. Agarwal, E. Ezra, and S. K. Ganjugunte, “Efficient sensor placement for surveillance problems,” in *Proc. of the 5th IEEE Int. Conf. on Distributed Computing in Sensor Systems*, pp. 301–314, 2009.

[24] B. Wang, “Coverage problems in sensor networks: A survey,” *ACM Comput. Surv.*, vol. 43, no. 4, pp. 32:1–32:53, 2011.

[25] K. J. Obermeyer, A. Ganguli, and F. Bullo, “Multi-agent deployment for visibility coverage in polygonal environments with holes,” *International Journal on Robust and Nonlinear Control*, vol. 21, no. 12, pp. 1467–1492, 2011.

[26] J. Leitner, “Robot formations for area coverage,” in *Intelligent Robotics and Applications*, vol. 5928 of *Lecture Notes in Computer Science*, pp. 100–111, Springer Berlin Heidelberg, 2009.

[27] D. Borrmann, P. J. de Rezende, C. C. de Souza, S. P. Fekete, S. Friedrichs, A. Kröller, A. Nüchter, C. Schmidt, and D. C. Tozoni, “Point guards and point clouds: Solving general art gallery problems,” in *Proc. of The 29th Annual Symposium on Computational Geometry*, pp. 347–348, 2013.

[28] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, and J. Hertzberg, “Globally consistent 3d mapping with scan matching,” *Robotics and Autonomous Systems*, vol. 56, no. 2, pp. 130 – 142, 2008.

[29] A. Efrat and S. Har-Peled, “Guarding galleries and terrains,” *Information Processing Letters*, vol. 100, no. 6, pp. 238 – 245, 2006.

[30] S. Eidenbenz, C. Stamm, and P. Widmayer, “Inapproximability results for guarding polygons and terrains,” *Algorithmica*, vol. 31, no. 1, pp. 79–113, 2001.

[31] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell, “A constant-factor approximation algorithm for optimal terrain guarding,” in *Proc. of The 16th Annual Symposium on Discrete Algorithms*, pp. 515–524, 2005.

[32] M. C. Couto, P. J. de Rezende, and C. C. de Souza, “An exact algorithm for minimizing vertex guards on art galleries,” *International Transactions in Operational Research*, vol. 18, no. 4, pp. 425–448, 2011.

[33] V. Chvátal, “A combinatorial theorem in plane geometry,” *Journal of Combinatorial Theory, Series B*, vol. 18, no. 1, pp. 39 – 41, 1975.

[34] S. Fisk, “A short proof of chvátal’s watchman theorem,” *Journal of Combinatorial Theory, Series B*, vol. 24, no. 3, pp. 374 – 375, 1978.

[35] D. Avis and G. Toussaint, "An efficient algorithm for decomposing a polygon into star-shaped polygons," *Pattern Recognition*, vol. 13, no. 6, pp. 395 – 398, 1981.

[36] D. Lee and A. Lin, "Computational complexity of art gallery problems," *IEEE Transactions on Information Theory*, vol. 32, no. 2, pp. 276–282, 1986.

[37] S. K. Ghosh, "Approximation algorithms for art gallery problems in polygons," *Discrete Applied Mathematics*, vol. 158, no. 6, pp. 718 – 722, 2010.

[38] A. Deshpande, T. Kim, E. Demaine, and S. Sarma, "A pseudopolynomial time o(logn)-approximation algorithm for art gallery problems," in *Algorithms and Data Structures*, vol. 4619 of *Lecture Notes in Computer Science*, pp. 163–174, Springer Berlin Heidelberg, 2007.

[39] Y. Amit, J. S. B. Mitchell, and E. Packer, "Locating guards for visibility coverage of polygons," in *Proc. of The 9th Workshop on Algorithm Engineering and Experiments*, pp. 120–134, 2007.

[40] A. Kröller, T. Baumgartner, S. P. Fekete, and C. Schmidt, "Exact solutions and bounds for general art gallery problems," *J. Exp. Algorithmics*, vol. 17, pp. 2.3:2.1–2.3:2.23, 2012.

[41] A. Kröller, M. Moeini, and C. Schmidt, "A novel efficient approach for solving the art gallery problem," in *WALCOM: Algorithms and Computation*, vol. 7748 of *Lecture Notes in Computer Science*, pp. 5–16, Springer Berlin Heidelberg, 2013.

[42] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner, "A novel type of skeleton for polygons," *The Journal of Universal Computer Science*, vol. 1, no. 12, pp. 752–761, 1995.

[43] E. Fogel, D. Halperin, and R. Wein, *CGAL Arrangements and Their Applications - A Step-by-Step Guide*, vol. 7 of *Geometry and Computing*. Springer, 2012.

[44] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. New York: Cambridge University Press, 2011.

[45] CGAL, *Computational Geometry Algorithms Library, version 4.3*. `http://www.cgal.org`.

[46] GLPK, *GNU Linear Programming Kit (GLPK), version 4.34*. `http://www.gnu.org/software/glpk/glpk.html`.

[47] T. Achterberg, "Scip: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[48] CPLEX, *CPLEX Optimizer, version 12.6.* `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`.

[49] GUROBI, *GUROBI Optimizer, version 5.6.2.* `http://www.gurobi.com/`.

[50] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *Natl. Academy of Sciences*, vol. 93, pp. 1591–1595, 1996.

[51] L. Blackmore, M. Ono, and B. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, 2011.

[52] J.-P. Calliess, D. Lyons, and U. D. Hanebeck, "Lazy auctions for multi-robot collision avoidance and motion control under uncertainty," in *Advanced Agent Technology*, vol. 7068 of *Lecture Notes in Computer Science*, pp. 295–312, Springer Berlin Heidelberg, 2012.

[53] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps.," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, 2010.

[54] S. Rodriguez, X. Tang, J.-M. Lien, and N. Amato, "An obstacle-based rapidly-exploring random tree," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 895–900, 2006.

[55] S. M. Lavalle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000.

[56] C. Urmson and R. Simmons, "Approaches for heuristically biasing rrt growth," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 2, pp. 1178–1183, 2003.

[57] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2383–2388, 2002.

[58] V. Vonasek, J. Faigl, T. Krajnik, and L. Preucil, "A sampling scheme for rapidly exploring random trees using a guiding path," in *Proc. of The 5th European Conf. on Mobile Robots*, p. 201–206, 2011.

[59] S. LaValle and J. Kuffner, J.J., "Randomized kinodynamic planning," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 1, pp. 473–479, 1999.

[60] E. Plaku, L. Kavraki, and M. Vardi, "A motion planner for a hybrid robotic system with kinodynamic constraints," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 692–697, 2007.

[61] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, June 2010.

[62] M. Zucker, J. Kuffner, and J. Bagnell, "Adaptive workspace biasing for sampling-based planners," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 3757–3762, 2008.

[63] M. Hassouna, A. Abdel-Hakim, and A. Farag, "Pde-based robust robotic navigation," *Image and Vision Computing*, vol. 27, no. 1-2, pp. 10–18, 2009.

[64] H. Zhao, "A fast sweeping method for eikonal equations.," *Math. Comput.*, vol. 74, no. 250, pp. 603–627, 2005.

[65] W.-K. Jeong and R. T. Whitaker, "A fast iterative method for eikonal equations," *SIAM J. Sci. Comput.*, vol. 30, no. 5, pp. 2512–2534, 2008.

[66] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[67] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "Rt-middleware: distributed component middleware for rt (robot technology)," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 3933–3938, 2005.

[68] A. Shigemura, Y. Ishikawa, J. Miura, and J. Satake, "An rt component for simulating people movement in public space and its application to robot motion planner development," *Journal of Robotics and Mechatronics*, vol. 24, no. 1, pp. 165–173, 2012.

[69] J. Satake and J. Miura, "Robust stereo-based person detection and tracking for a person following robot," in *Proc. of ICRA Workshop on Person Detection and Tracking*, 2009.

[70] M. Andriluka, S. Roth, and B. Schiele, "Monocular 3d pose estimation and tracking by detection," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 623–630, 2010.

[71] C. Weinrich, C. Vollmer, and H.-M. Gross, "Estimation of human upper body orientation for mobile robotics using an svm decision tree on monocular images," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 2147–2152, 2012.

[72] D. Baltieri, R. Vezzani, and R. Cucchiara, "People orientation recognition by mixtures of wrapped distributions on random trees," in *ECCV 2012*, vol. 7576 of *Lecture Notes in Computer Science*, pp. 270–283, Springer Berlin Heidelberg, 2012.

[73] C. Chen, A. Heili, and J. Odobez, “Combined estimation of location and body pose in surveillance video,” in *Proc. of The 8th IEEE Int. Conf. on Advanced Video and Signal-Based Surveillance (AVSS)*, pp. 5–10, 2011.

[74] W. Schwartz, A. Kembhavi, D. Harwood, and L. Davis, “Human detection using partial least squares analysis,” in *Proc. of IEEE 12th Int. Conf. on Computer Vision*, pp. 24–31, 2009.

[75] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proc. of Int. Conf. on Computer Vision & Pattern Recognition*, vol. 2, pp. 886–893, 2005.

[76] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan, “Fast human detection using a cascade of histograms of oriented gradients,” in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, pp. 1491–1498, 2006.

[77] T. Ojala, M. Pietikäinen, and T. Mäenpää, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 971–987, 2002.

[78] R. Rosipal and N. Krämer, “Overview and recent advances in partial least squares,” in *Subspace, Latent Structure and Feature Selection*, vol. 3940 of *Lecture Notes in Computer Science*, pp. 34–51, Springer Berlin Heidelberg, 2006.

[79] H. Wold, “Soft modeling by latent variables; the nonlinear iterative partial least squares approach,” *Perspectives in Probability and Statistics*, pp. 520–540, 1975.

[80] S. Wold, W. Johansson, and M. Cocchi, “Pls-partial least squares projections to latent structures,” in *3D QSAR in Drug Design*, vol. 1, pp. 523–550, Springer Science & Business Media, 1993.

[81] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[82] A. Bosch, A. Zisserman, and X. Muoz, “Image classification using random forests and ferns,” in *Proc. of IEEE 11th Int. Conf. on Computer Vision*, pp. 1–8, 2007.

[83] D. Hoiem, A. Efros, and M. Hebert, “Putting objects in perspective,” in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, pp. 2137–2144, 2006.

[84] W. Choi and S. Savarese, “Multiple target tracking in world coordinate with single, minimally calibrated camera,” in *ECCV 2010*, vol. 6314 of *Lecture Notes in Computer Science*, pp. 553–567, Springer Berlin Heidelberg, 2010.

[85] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[86] A. Ess, B. Leibe, K. Schindler, and L. Van Gool, "Robust multiperson tracking from a mobile platform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 10, pp. 1831–1846, 2009.

[87] L. Wang, J. Shi, G. Song, and I.-f. Shen, "Object detection combining recognition and segmentation," in *ACCV 2007*, vol. 4843 of *Lecture Notes in Computer Science*, pp. 189–199, Springer Berlin Heidelberg, 2007.

[88] V. Ferrari, M. Marin-Jimenez, and A. Zisserman, "Progressive search space reduction for human pose estimation," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[89] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *The Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2002.

[90] D. Benbouzid, R. Busa-Fekete, N. Casagrande, F.-D. Collin, and B. Kégl, "Multiboost: A multi-purpose boosting package," *Journal of Machine Learning Research*, vol. 13, pp. 549–553, 2012.

[91] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable selection using random forests," *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2225–2236, 2010.

[92] K. J. Archer and R. V. Kimes, "Empirical characterization of random forest variable importance measures," *Computational Statistics & Data Analysis*, vol. 52, no. 4, pp. 2249–2260, 2008.

[93] J. L. B. Claraco, "Development of scientific applications with the mobile robot programming toolkit," *The MRPT reference book. Machine Perception and Intelligent Robotics Laboratory, University of Málaga, Málaga, Spain*, 2008.

[94] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 46–51, 2011.

[95] I. Ardiyanto and J. Miura, "Rt components for using morse realistic simulator for robotics," in *Proc. of The 13th SICE System Integration Division Annual Conference*, pp. 535–538, 2012.

[96] I. Ardiyanto and J. Miura, "Partial least squares-based human upper body orientation estimation with combined detection and tracking," *Image and Vision Computing*, vol. 32, no. 11, pp. 904–915, 2014.

# Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Jun Miura, whose expertise, understanding, and patience, gives a great influence to my research experience. I appreciate his vast knowledge and skill in the robotic and vision areas, and his assistance in writing papers, reports, and thesis, also for giving me chances to attend many prestigious conferences. I gain a lot of knowledge during my study here.

I also want to express my gratitude to Prof. Kuriyama, Prof. Terashima, and Prof. Tsubouchi as the examiner members who kindly read my thesis and gave many useful suggestions.

I am deeply grateful to the former Assistant Professor Dr. Junji Satake for supporting my research, especially in the people tracking part. Special thanks given to Assistant Professor Dr. Shuji Oishi for the stimulating discussions. My sincere thank also goes to Ms. Mikiko Kobayashi, who has helped me in many ways, like preparing a lot of documents for conferences and Research Assistant.

I thank my lab-mates at Advance Intelligent System Lab: D3 Mr. Bima Sena, D1 Kenji Koide, M2 Yohei Inoue, Masanobu Shimizu, Tomoyoshi Hara, Kenta Yamada, Yuta Takaba, Albadr Lutan Nasution, and Yuki Namihira, M1 Taku Kudou, Takahiro Sakai, Kaichiro Nishi, and Wataru Miyazaki, B4 Yoshiki Kohari, Seiichiro Une, Genki Nagai, Mitsuhiro Demura, and Yuutaro Chikada, for all the fun we have had in the lab. Also I thank my former senior Junichi Sugiyama, who always helps me when I had troubles during experiments. Special thanks to Koide-kun and Shimizu-kun as the close partners for doing researches.

At the last, I want to express my gratitude to JICA which has supported my study through AUN SEED-Net Scholarship.

# List of Publications

## Journals

[1] I. Ardiyanto and J. Miura, "Partial Least Squares-based Human Upper Body Orientation Estimation with Combined Detection and Tracking," *Image and Vision Computing*, vol. 32(11), pp. 904-915, 2014. (Chapter 6)

[2] I. Ardiyanto and J. Miura, "Real-time Navigation using Randomized Kinodynamic Planning with Arrival Time Field," *Robotics and Autonomous Systems*, vol. 60(12), pp. 1579-1591, 2012. (Chapter 5)

## Conferences

[1] I. Ardiyanto and J. Miura, "Human Motion Prediction Considering Environmental Context," *The 14th IAPR Conference on Machine Vision and Application (MVA)*, pp. 390-393, May 18-22, 2015, Tokyo, Japan. (Chapter 4)

[2] I. Ardiyanto and J. Miura, "Cameraman Robot: Dynamic Trajectory Tracking with Final Time Constraint using State-time Space Stochastic Approach," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3108-3115, September 14-18, 2014, Chicago, USA. (Chapter 5)

[3] I. Ardiyanto and J. Miura. "Visibility-based Viewpoint Planning for Guard Robot using Skeletonization and Geodesic Motion Model," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 652-658, May 6-10, 2013, Karlsruhe, Germany. (Chapter 2, 3, and 4)

[4] I. Ardiyanto and J. Miura, "3D Time-space Path Planning Algorithm in Dynamic Environment Utilizing Arrival Time Field and Heuristically Randomized Tree," *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 187-192, May 14-18, 2012, St. Paul, Minneapolis, USA. (Chapter 5)

[5] I. Ardiyanto and J. Miura, "Heuristically Arrival Time Field-Biased (HeAT) Random Tree: An Online Path Planning Algorithm for Mobile Robot Considering Kinodynamic Constraints," *2011 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 360-365, December 7-11, 2011, Phuket, Thailand. (Chapter 5)