Robot Learning for Autonomous Navigation in a Dynamic Environment

by

Yunfei Zhang

B.Sc., Qingdao University of Science and Technology, 2006

M.Sc., Shanghai Jiao Tong University, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Mechanical Engineering)

The University of British Columbia

(Vancouver)

September 2015

Abstract

This dissertation addresses autonomous navigation of robots in a dynamic environment where the existence of moving and/or unknown objects leads to more serious challenges for the robots than those when operating in a traditional stationary environment. Therefore, the use of learning capabilities to facilitate proper robotic operation in a dynamic environment has become an important research area in the past decade. This dissertation proposes several novel learning-based methods to overcome the shortcomings in the existing approaches of autonomous navigation. Three aspects are addressed in the present work.

First, a real-time path planning method is designed for autonomous navigation that can generate a path that avoids stationary and moving obstacles. To this end, learning ability is imparted to the robot. The present framework incorporates the statistical planning approach called probabilistic roadmap (PRM), Q-learning together with regime-switching Markov decision process (RSMDP) due to its beneficial characteristics, to form a robust Q-learning. Consequently, the initial path can be improved through robust Q-learning during interaction with a dynamic environment.

Second, motion planning under constraints is investigated. Specifically, a closed-form piecewise affine control law, called piecewise affine-extended linear quadratic regulator (PWA-ELQR), for nonlinear-nonquadratic control problems with constraints is proposed. Through linearization and quadratization in the vicinity of the nominal trajectories, nonlinear-nonquadratic control problems can be approximated to linear-quadratic problems where the closed-form results can be derived relatively easily.

Third, people detection is integrated into the autonomous navigation task. A

classifier trained by a multiple kernel learning-support vector machine (MKL-SVM) is proposed to detect people in sequential images of a video stream. The classifier uses multiple features to describe a person, and learn its parameter values rapidly with the assistance of multiple kernels.

In addition to the methodology development, the present research involves computer simulation and physical experimentation. Computer simulation is used to study the feasibility and effectiveness of the developed methodologies of path planning, motion planning and people detection. The experimentation involves autonomous navigation of a homecare robot system. The performance of the developed system is rigorously evaluated through physical experimentation and is improved by refining the developed methodologies.

Preface

The entire work presented in this dissertation was conducted at the Industrial Automation Laboratory of the University of British Columbia, Vancouver campus under the supervision of Dr. Clarence W. de Silva. A collection of manuscripts, resulting from the collaboration of several researchers, contribute the content of this work. I was responsible for majority of the research in this work, including literature survey, algorithm development and implementation, open-source software implement, numerical simulation and physical experimentation, with the guidance and advice of my supervisor, Dr. Clarence W. de Silva. I wrote the manuscripts, which were edited and refined by Dr. Clarence W. de Silva.

Parts of Chapter 2 and Chapter 3 are based on the work published on:

- Yunfei Zhang, Weilin Li and Clarence W. de Silva, "RSMDP-based Robust Q-learning for Optimal Path Planning in a Dynamic Environment," *International Journal of Robotics and Automation*, 2015 (accepted).

A version of combined Chapter 4 and Chapter 6 has been submitted to a journal and is under second round review:

- Yunfei Zhang, Xun Chen and Clarence W. de Silva, "PWA-Extended LQR for Optimal Motion Planning of Nonholonomic Mobile Robot with Constraints".

Chapter 5 is based on the work published on:

- Yunfei Zhang, Rajen Bhatt, and Clarence W. de Silva, "MKL-SVM-based human detection for autonomous navigation of a robot," *IEEE International Conference In Computer Science & Education (ICCSE)*, Vancouver, August 22-24, 2014.

Table of Contents

List of Tables

List of Figures

Nomenclature

$A = \{a_1, a_2, ..., a_N\}$	a set of actions of an MDP
$\mathbf{A}_t$	state matrix of a discrete-time formulation
$\bar{\mathbf{A}}_t$	state matrix of a reverse discrete-time formulation
α	learning rate
b	optimal parameter
$\mathbf{B}_t$	control matrix of a discrete-time formulation
$\bar{\mathbf{B}}_t$	control matrix of a reverse discrete-time formulation
c_t	immediate cost function at time-step t
c_l	immediate cost function at final time-step l
$\begin{bmatrix} \mathbf{D}_t & \mathbf{C}_t^T \\ \mathbf{C}_t & \mathbf{E}_t \end{bmatrix}$	cost weight matrix in cost-to-go function
$\begin{bmatrix} \bar{\mathbf{D}}_t & \bar{\mathbf{C}}_t^T \\ \bar{\mathbf{C}}_t & \bar{\mathbf{E}}_t \end{bmatrix}$	cost weight matrix in cost-to-come function
$\begin{bmatrix} \mathbf{d}_t \\ \mathbf{e}_t \end{bmatrix}$	cost weight vector in cost-to-go function
$\begin{bmatrix} \bar{\mathbf{d}}_t \\ \bar{\mathbf{e}}_t \end{bmatrix}$	cost weight vector in cost-to-come function
δ	iteration termination threshold
d_x	vertical stride size
d_y	horizontal stride size
η	geometric margin
E	mean value (expectation)
γ	discount factor
I	image
I_m	gradient magnitude image
I_o	gradient orientation image

I_x	vertical difference image
I_y	horizontal difference image
$\mathbf{k}$	kernel function
L	distance between the two wheels
P	transition function of an MDP
P_{ψ_k}	transition function of an RSMDP
π	policy of an MDP
π_{ψ_k}	policy of an RSMDP
π^*	optimal policy of an MDP
$\pi^*_{\psi_k}$	optimal policy of an RSMDP
ψ	regime
Ψ	collection of regime
Φ_i	feature space
Q	state-action function
Q^*	optimal state-action function
Q_{ψ_k}	state-action function in each regime
$Q^*{}_{\psi_k}$	optimal state-action function in regime ψ_k
$\mathbf{Q}$	Q value matrix
r	cost function
r_{ψ_k}	cost function in regime ψ_k
R	reward function of an MDP
R_{ψ_k}	reward function of an RSMDP
$\mathbf{R}$	reward matrix
σ	iteration threshold
$S = \{s_1, s_2, ..., s_N\}$	a set of states of an MDP
TC	terminal cost function of an MDP
TC_{ψ_k}	terminal cost function of an RSMDP
t	time
$\mathbf{u} = [v_l, v_r]^T$	robot control input, v_l left wheel velocity, v_r right wheel velocity
ε	greedy threshold
v_t	cost-to-go function
$\bar{v}_t$	cost-to-come function
V	value function
V^*	optimal value function
V_{ψ_k}	value function in each regime
$V^*{}_{\psi_k}$	optimal value function in regime ψ_k
w_m	normal vector to the hyperplane for the feature space Φ_m
$\mathbf{w}$	optimal parameter

List of Acronyms

ANN artificial neural networks

C-space configuration space

CNN convolutional neural networks

DP dynamic programming

EL evolutionary learning

ELQR extended linear-quadratic-regulator

HOG histograms of oriented gradients

HOG-HOF histograms of oriented gradients-histograms of optical flow

HOF histograms of optical flow

KKT Karush-Kuhn-Tucker

LQR linear-quadratic-regulator

LSVM linear support vector machine

MC Monte Carlo

MDP Markov decision process

MKL multiple kernel learning

MKL-SVM multiple kernel learning-support vector machine

MPC model predict control

IAL Industrial Automation Laboratory

IL imitation learning

PCL point cloud library

POMDP partially observable Markov decision process

PRM probabilistic roadmap

PWA-LQR piecewise affine-LQR

PWA-ELQR piecewise affine-ELQR

QP quadratic programming

RBF radial basis function

RGB red-green-blue

RGB-D red-green-blue-depth

RL reinforcement learning

RHC receding horizon control

ROS robot operating system

RSMDP regime-switching Markov decision process

SimpleMKL simple multiple kernel learning

SVM support vector machine

TDL temporal difference learning

Acknowledgments

Five years of pursing PhD degree is an extremely fantastic journey in life, bringing me much joy and cheers, at the price of many challenges, hardship, pressure and tears. Only after experiencing such feelings, can I thoroughly understand what gratitude is.

First, I wish to express my sincere gratitude to my supervisor, Dr. Clarence W. de Silva. It is he who offered me this precious opportunity for pursing a PhD degree, enabling me to freely swim in the ocean of knowledge and solidly build a foundation for my future, which without doubt has completely changed my path of life. I greatly appreciate all of his advice, mentorship, understanding and unwavering support during the past five years. In my eyes, Dr. de Silva is not just my academic supervisor, but also my life mentor who always encouraged me with his generosity and patience.

I would like to thank a few of senior labmates, Dr. Ying Wang for his recommendation, Dr. Tahir Khan for his humour, Dr. Roland Haoxiang Lang for the happy hours I spent at his home during festivals, and Dr. Yanjun Wang for enjoyable conversations in our lab. Without their help and guidance, I could not go through the difficulties in the beginning period of my PhD program.

Also, I wish to thank all my other colleagues in the Industrial Automation Laboratory (IAL), Ms. Yu Du, Mr. Muhammad Tufail Khan , Mr. Shan Xiao, Ms. Lili Meng, Mr. Hani Balkhair, Ms. Pegah Maghsoud, Mr. Min Xia, Mr. Shujun Gao, Mr. Zhuo Chen, Mr. Teng Li., and Mr. Tongxin Shu for their friendship and help.

A big thank you goes to my close friendship groups, THREEPLUSONE in China and WOCHANG in Vancouver. Their support and comfort have made me

believe that a bright future will come my way eventually. Special thanks go to my dear friend, Prof. Xun Chen, for his advice on research that inspires me to purse a higher academic goal.

This work has been supported by research grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada, the Canada Foundation for Innovation (CFI), the British Columbia Knowledge Development Fund (BCKDF), and the Tier 1 Canada Research Chair in Mechatronics and Industrial Automation held by C.W. de Silva. Thank all sponsors.

In conclusion, I would like to express my deepest gratitude to my family. Thank you my parents and parents-in-law, for your unconditional love and confidence in me that encouraged me to always go forward. Thank you my lovely twins, Lunlun and Yaya. You are god-sent gifts that can recharge me whenever I am out of power. Pingping Liu, my wife, a simple thank you cannot express my gratitude for your sharing and contributing to this journey. Only with her that I am able to know how my world rotates.

Chapter 1

Introduction

1.1 Motivation

During the past decade, researchers in robotics have increasingly redirected their attention from traditional industrial robots operating in structured or stationary environments to the more challenging area of mobile robotics in unstructured and dynamic environments with applications in military, space exploration, underwater operations, and service robots. In the area of service robotics, homecare robotics for the elderly and the disabled has a special significance for its contribution to improving the quality of life and reducing the caregiver burdens and costs. According to UN statistics shown in Fig. 1.1, the percentage of adults who are over 65 (with respect to those over 16) will be more than double in many countries (including Canada) in the next 50 years. As a result there will be a severe shortage of nursing assistants for the elderly and the disabled. This will generate a major opportunity for the development of robotic technologies to assist or even substitute the nursing assistants in a home environment. Also, this has important quality-of-life implications since it is known that people regardless of their age or the ability prefer to live independently in their own homes. The overall goal of the homecare project in our laboratory (Industrial Automation Lab) is to develop technologies for autonomous mobile robots that can assist the elderly and the disabled in their daily activities in a home environment. Many such activities involve autonomous navigation in the presence of static and dynamic obstacles under uncertain and unknown conditions.

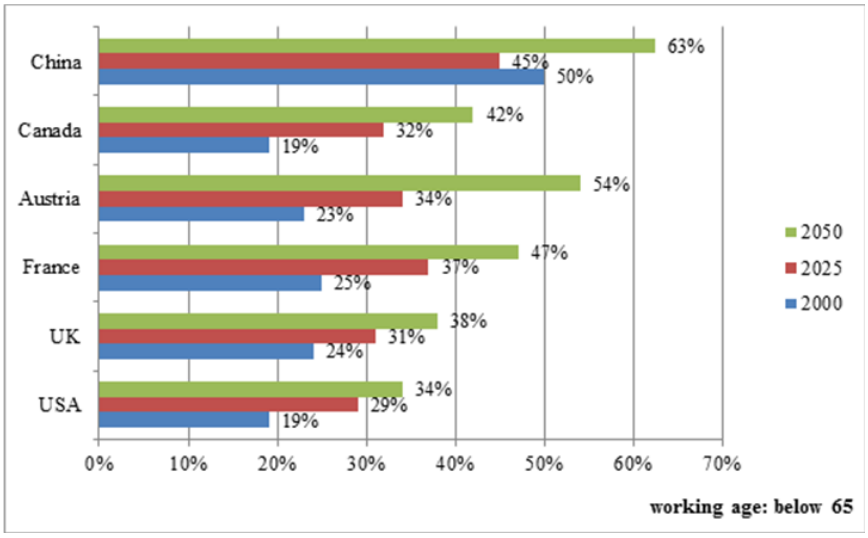


Figure 1.1: Percentage of elderly (>65) with respect to working-age (16-65) population.

Autonomous navigation means that the robot can navigate by itself from the start location to reach the goal location, in the presence of obstacles, which is an essential ability for a variety of application of intelligent robots such as military tasks, space exploration, and people rescue. In static and structured environments, maps and models of the world (environment in which the robot moves) can be generated to support autonomous navigation by a prior sensory/perception process together with available information. However, in homecare robotics, as shown in Fig. 1.2, autonomous navigation becomes more complex since the home environment is dynamic (and somewhat arbitrary), and changes take place due to moving objects such as humans and pets and their actions. For instance, giving first aid to a person who suddenly suffers heart attack or guiding an elderly person to a washroom involves robotic movement in a dynamic environment[1]. Therefore, it is important that such robots learn how to deal with dynamic environments so that they can repeat those or similar tasks more effectively, by taking advantage of their prior experience. The main focus of the present dissertation is the imparting of the learning capability to robots so that they can adapt to dynamic environments during

autonomous navigation.

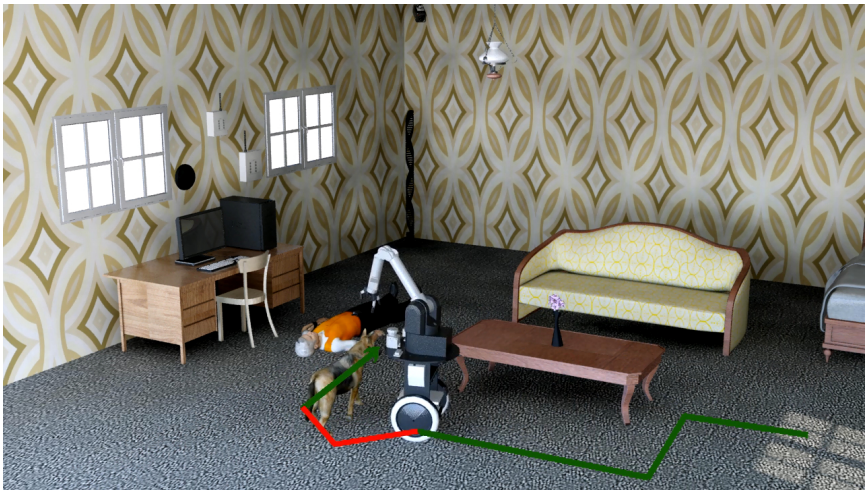


Figure 1.2: A scenario of autonomous navigation in home environment.

The problem of population aging is a global one due to the rising life expectancy and declining birth rate. Initially, this has been a problem primarily in countries that are economically more developed. But recently, developing countries such as China and India are also affected by this problem. According to the prediction of Statistics Canada, the percentage of the senior population will reach 25% of the total population by 2050. The social and economic effects of population aging are enormous. The elderly need some care and physical assistance when necessary, in their daily life. The total cost of providing care to people of this group is tremendous, considering the large size of the aging population.

Another group of people that will rapidly increase its size are those with physical and cognitive impairments. The Canadian government spends about $9 billion on disability related matters. The cost of basic care for a disabled person at home is about $10,000/month, and this is a huge burden on the Government.

The burden described above may be reduced through the use of a homecare robotic system, which consists of a group of robotic devices to work together in a coordinated and efficient manner and carry out a common task of providing assistance to elderly and/or disabled people in the home setting. The needed technologies of robotics, network communication, and control are sufficiently mature and

are available at reasonable cost. Due to the lack of necessary assistive technologies and specialized end-effector devices, and also due to insufficient efforts in bringing the pertinent technology teams together, the development of affordable and reliable systems has eluded the application area in the past. The research in homecare robotics is a new but rapidly developing field of service robotics, especially in Japan.

Through our established facilities in the Industrial Automation Laboratory we are in the process of developing an affordable and effective system of service robots for the elderly or the disabled people with physical and cognitive impairments in a home setting. The system comprises several sets of robot manipulators having mobile bases, and can provide assistance in daily activities, medical assistance, surveillance, and so on. They can work independently or collaboratively in a team, based on the complexity of the task. The designed homecare robotic system allows the care-receiver to stay in their familiar home environment where the assistance is provided, which is a merit of the associated technologies.

1.2 Research Scope and Problem Specification

1.2.1 Research Scope

In carrying out complex robotic tasks in a dynamic environment, autonomous navigation involves multi-domain technologies such as sensor fusion, data processing, computer vision, artificial intelligence, optimal control and so on. Typically it can be divided into three functional parts: perception part, computation part and execution part, as shown in Fig. 1.3. The perception part provides many kinds of sensor data to recognize the environment around the robot; the computation part analyses the data from the perception part and gives corresponding commands to the robot according to the task goals; and the execution part accomplishes all the mechanical action by following the commands from the computation part. In view of the needed capabilities of autonomous navigation for a homecare application, the scope of the present research focuses on three main areas: 1. global path planning that provides an optimal path to direct the robot to the global goal; 2. local motion planning that gives the robot specific controls to follow the global path; 3. people

detection in a home environment. Path planning and local motion planning which belong to the computation part are critical and indispensable for autonomous navigation. People detection also falls into the computation part. But more importantly, humans play an important role in the environment of many robotic applications, especially in a homecare environment. Therefore, taking the existence of humans into account during autonomous navigation has a vitally practical significance.

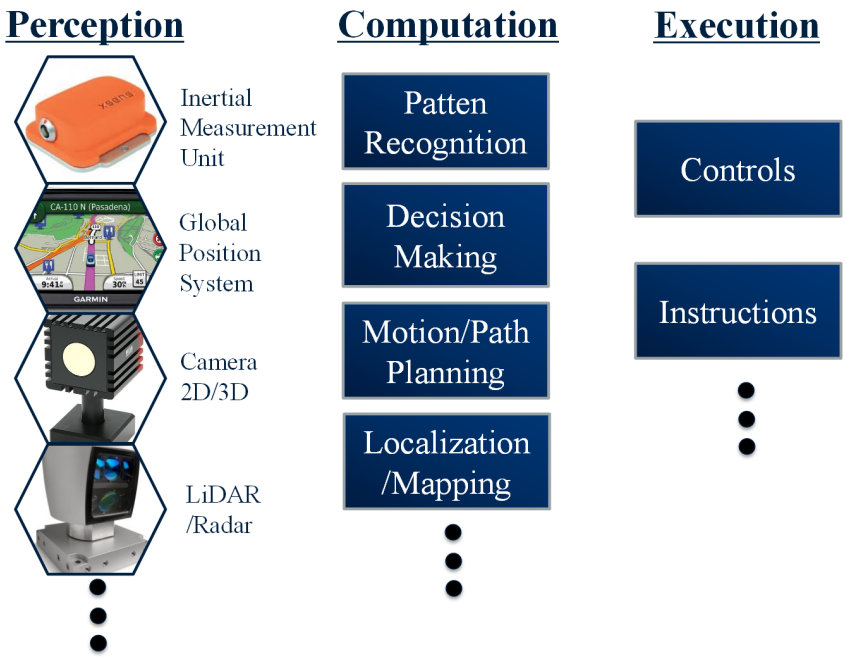


Figure 1.3: A typical function structure of autonomous navigation.

1.2.2 Problem Specification

This dissertation combines the aforementioned three research scopes into one autonomous navigation scenario, as shown in Fig. 1.4. The autonomous navigation system needs to complete the following tasks: detect people around the robot and the people's location, set the people location as the goal location, calculate an optimal path starting from the current location to the goal location, compute a local motion plan to follow the global path, and avoid obstacles (both moving and static)

during all of the procedures. The mobile robot that is used in this research work is a nonholonomic two-wheel differential-drive mobile robot. A zoom-out view of the mobile robot and its kinematic scheme are shown in Fig. 1.5.

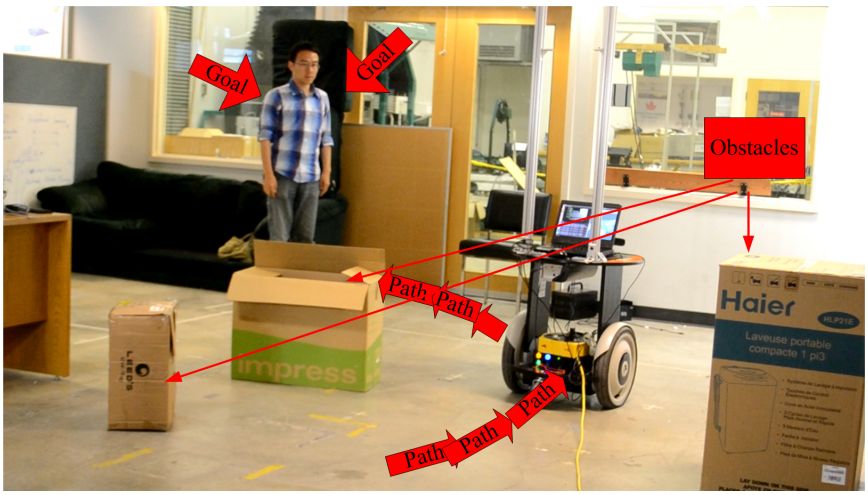


Figure 1.4: The designed scenario of autonomous navigation in a real-world home environment.

The robot states are represented by its position and orientation (angle of rotation) $\mathbf{x} = [x, y, \theta]^T$. Considering the left wheel velocity and the right wheel velocity as the robot control inputs, $\mathbf{u} = [v_l, v_r]^T$, the robot kinematic model can be expressed as

$$\begin{aligned} \dot{x} &= \frac{1}{2}(v_r + v_l)\cos\theta \\ \dot{y} &= \frac{1}{2}(v_r + v_l)\sin\theta \\ \dot{\theta} &= \frac{1}{L}(v_r - v_l) \end{aligned} \tag{1.1}$$

where L represents the distance between the two wheels.

Generally for formula derivation, it is common to express Equation (1.1) in a vector form of a deterministic continuous-time formulation which is the most typical way to describe the kinematics of a nonlinear robotic system, as given by Equation (1.2).

$$\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{u}(t)) \tag{1.2}$$

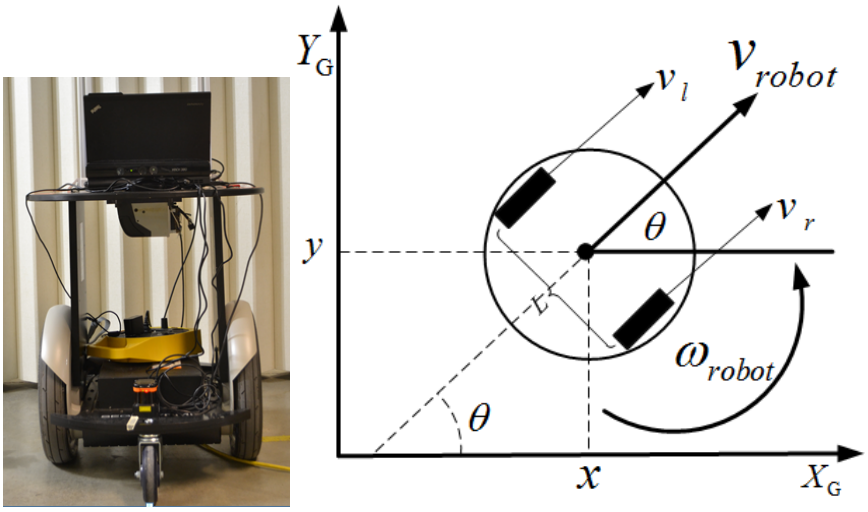
For computer representation of a continuous-time system, the discrete-time formulation of Equation(1.2) for any given time-step is expressed as:

$$\mathbf{x}_{t+1} = \mathbf{g}_t(\mathbf{x}_t, \mathbf{u}_t), \tag{1.3}$$

with $\mathbf{g}_t \in \mathbb{X} \times \mathbb{U} \to \mathbb{X}$, $\mathbf{x}_t \in \mathbb{X}$ and $\mathbf{u}_t \in \mathbb{U}$, where $\mathbb{X} \subset \mathbb{R}^n$ and $\mathbb{U} \subset \mathbb{R}^m$. Accordingly, for the purpose of cost-to-come (as discussed in Section 4.3.3) the inverse discrete-time kinematic is denoted as:

$$\mathbf{x}_t = \bar{\mathbf{g}}_t(\mathbf{x}_{t+1}, \mathbf{u}_t), \tag{1.4}$$

where $\bar{\mathbf{g}}_t \in \mathbb{X} \times \mathbb{U} \to \mathbb{X}$ such that $\mathbf{g}_t(\bar{\mathbf{g}}_\mathbf{t}(\mathbf{x}_t + 1, \mathbf{u}_t), \mathbf{u}_t) = x_{t+1}$.



(a) **(b)**

Figure 1.5: (a) The Segway two-wheel differential-drive mobile robot; (b) Kinematic scheme of the two-wheel differential-drive mobile robot.

1.3 Related Works

1.3.1 Path Planning

Path planning is critical for a homecare robot, as it moves in an environment of unknown and dynamic factors. In path planning, the path of the navigating robot from the start location to the goal location is planned according to some criteria (e.g., shortest path, quickest path and/or path of minimum energy) while avoiding collisions with static and moving obstacles, and subject to some constraints (e.g., robot capabilities with respect to its possible movements). However, path planning becomes more complex in the present application since the home environment is dynamic and unstructured due to moving objects such as humans and pets and their actions (e.g., rearrangement of furniture)[2–4].

Sampling-based methods such as probabilistic roadmap (PRM) [5–7] and Rapidly-exploring Random Trees (RRT) [5, 8] became very popular since the nineties. These algorithms have proved to be very effective for path planning in high-dimension since they rely on a collision-checking module instead of explicit representation of the environment; however, convergence to optimal solutions cannot be guaranteed with probability one . Recently in [8], a series of variants to the sampling-based methods (PRM*, RRT*) have been proposed to provide probabilistic completeness. This means, if a solution exists for a particular plan, the probability of discovering the optimal solution converges to one as the sampled number of states increases to infinity. All these methods do not explicitly consider moving obstacles when asymptotically converging to the optimal path.

In practical applications, however, most challenges in path planning come from factors of uncertainty in dynamic environments such as varying environment and unknown and moving obstacles. Inspired by the consideration of static obstacle and moving obstacle separately and the configuration-time state space, Van den Berg proposed a hybrid approach, which first constructs a path using PRM in the configuration-time state space based on the stationary obstacles in the environment, and subsequently plans a collision-free path from the original path by taking into account the moving obstacles, which are modeled as time discs [9, 10]. A particular advantage of this approach is that it does not need to exactly know the specific

movements of the obstacles, such as speed and direction, and the planner is able to generate a path on-line while taking into account changes in the environment during the period of deliberation. However, the approach has some disadvantages as well. One is the assumption that the maximum speed of obstacle motion must be less than that of the robot. Another is that the safety buffer of the time disc module sacrifices part of the collision-free space. Furthermore, the approach does not make use of the previous planning experience, which can help to reduce the computational burden.

1.3.2 Motion Planning

The objective of motion planning under constraints is to plan optimal motions for a robot such that a user-defined cost function is minimized, without disrupting any required constraints on states and inputs. It is basically treated as an optimal control problem. The linear-quadratic-regulator (LQR)[11] is a transitional and popular optimal feedback controller,which can determine a proper optimal robotic control action, given the current state. It has been studied for several decades and been successfully applied in many practical optimal control problems [12][13][14]. However, it only provides a closed-form optimal solution and assumes that the robot system is linear and the cost function is quadratic. Iterative LQR (iLQR)[15] and its variants [16][17] have been proposed in order to deal with nonlinear systems with nonquadratic cost functions. But these approaches do not consider constraints explicitly during the iteration process. They just add an extra effort (e.g., linear search) to guarantee iteration convergence. Consequently, the executed trajectories easily deviate the robot from the planned trajectories beyond a tolerant range, and the robot may even fail to complete the task (e.g., avoiding obstacles).

Much of the work related to nonlinear-nonquadratic control problems has focused on deterministic robot systems. Basically most of the numerical methods fall into one of the following three categories.

One category consists of sample-based methods [18][19].These methods utilize search algorithms to go through the configuration space that is defined by the user. Even though these methods can give global results and implicitly handle constraints by sampling, the need of gridding or other ad doc discretization can cause difficul-

ties. Learning-based method is another category. Based on the structure of Markov Decision Processing (MDP), approximate reinforcement learning (ARL)[20]uses polity iteration or value iteration to get an approximate cost function by learning from samples and then obtain the optimal control. Optimistic planning (OP)[21] approximates the cost function by using a new search method called simultaneous optimistic optimization for planning. However, these learning-based methods also have to first discretize the continuous space, which makes them unsuitable for large-scale problems.

The third category contains optimization-based methods. Differential dynamic programming (DDP)[22][23] belongs to this category. DDP first maintains a representation of a single trajectory, and then improves it locally relying on dynamic programming. It shows second-order convergence and is numerically more efficient than Newton's method[24]. iLQR is closely related to DDP but proves to be more efficient for complex control problems [15]. It iteratively linearizes the nonlinear system and quadratizes the cost function at the current nominal trajectory given by the previous iteration and gets a new trajectory via modified Riccati equations. In order to guarantee convergence, iterative LQR (iLQR) [15]additionally needs convergence measure (e.g. line search) to drive the new trajectory not far from where linearization and quadratization are valid. Extended LQR (ELQR)[17] is proposed to avoid the specific convergence measure through a concept called LQR-smoothing. LQR-smoothing utilizes cost-to-go and cost-to-come cost functions to establish a total-cost function for each iteration. It implements linearization and quadratization only about a sequence of states and controls that are dynamically feasible. A particular advantage of the methods DDP, iLQR and ELQR is that they yield feedback control laws that are suitable for real-time control. However, their common shortcoming is that they are unable to handle constraints explicitly.

Model Predictive control (MPC) [25][26][27], also called receding horizon control (RHC)[28], has become an acceptably systematic way to handle constraints explicitly for motion planning[29]. MPC iteratively solves the convex-optimization problem starting from the current state over a finite horizon, and then obtains a sequence of open-loop optimal controls. The main drawbacks of the MPC are the intrinsic open-loop characteristic and relatively formidable on-line computational effort[30]. These drawbacks limit its applicability to relatively slow or small prob-

lems, though the partially close-loop RHC (PCLRHC)[28] method can improve the speed of performance. Sequential quadratic programing (SQP) is also suitable for solving constraints. It can accommodate convex constraints on the state and the control input. However, SQP typically does not generate a feedback control policy, but gives a sequence of optimal open-loop control inputs. The piecewise affine-ELQR (PWA-ELQR) method proposed in the present work, inherits advantages from ELQR, and incorporates piecewise affine close-loop optimal control into the process of minimizing the total cost function, so that the input control constraints and differential constraints can be explicitly solved in each iteration. Additionally, the feedback control is locally optimal, which further facilities fast convergence under constraints to generate executable controls. The present work assumes that obstacle occupation is a unique form of state constraint, since the entire unreachable state space is defined as the obstacle state and is considered in the cost function.

1.3.3 People Detection

People detection for autonomous navigation of homecare robots has attracted considerable interest in the past decade, since humans are integral part of a homecare robots environment. Though remarkable progress has been made in human-free scenarios such as the DARPA Urban Challenge[1], autonomous navigation in dynamic environment where there are moving people is still a largely unsolved problem.

Digital cameras can deliver much richer appearance information than range sensors such as LIDAR, although cameras rarely reach the geometric accuracy as those of range sensors. This advantage of richer appearance makes cameras attractive for detecting humans in applications of autonomous navigation. One popular framework to solve appearance-based human detection usually insists on a space of image features and a learned classifier. In such a framework, there are three main steps: feature extraction, classifier training, and detection strategy. One type of features extracted from raw image data is called histograms of oriented gradients (HOG) [31] and is quite popular. Its variants include histograms of optical

[1] http://www.darpa.mil/GRANDCHALLENGE/

flow (HOF) [32], and combined versions with other features like integral channel features [33][34] or color features [35]. Classifiers usually use two categories of approaches: support vector machine (SVM) or some variants of boosting, which learn to map the features to the scores indicating the probability that the detected area represents a human. After training the classifier, a sliding window scheme [36] is commonly used to search the entire image and detect which part is believed to represent a human according to the detection score.

1.4 Challenges, Contributions and Organization of the Dissertation

The challenges in each considered component of autonomous navigation are summarized below:

1. The considered dynamic environment is unstructured and has uncertainty due to lack of knowledge of the behavior of moving obstacles. Hence it is rather difficult to model the surrounding environment and the unpredictable movements of the obstacles. Therefore, a good autonomous navigation system requires that the global path planning is able to deal with such dynamic environments.
2. Motion planning is more difficult as well in dynamic environments due to three main challenges:(1) continuous state and control space bring about the inevitable curse of dimensionality; (2) nonlinear robot system and non-quadratic cost function make the solving optimization problem extremely hard; and (3) inherent system constraints decrease the effectiveness of the optimal solution. Apparently, it is non-trivial to address the foregoing three points all-in-once for constrained motion planning.
3. Distinguishing moving people from other objects so as to locate a persons position is another big challenge. This challenge is caused due to several reasons [37]: (1) Human appearance possesses very high variability caused by changing pose, wearing different clothes with diverse colors, and having a considerable range of sizes. In addition, a cluttered background (home or urban environment) under a wide range of illumination and weather conditions varies the quality of the sensed information. The case of being partially

occluded by other objects definitely increases the analysis complexity; (2) Both humans and sensors that are in motion can complicate the movement analysis. Furthermore, humans appearing at different view angles and a sensor system working over a large scope of distances result in more complex situations. (3) The real-time requirement for detection demands fast reaction time and robust performance.

The goal of this research is to develop novel learning methods to address these challenges that arise when a homecare robot operates in a dynamic environment. In achieving this objective, the main three technical contributions are:

1. This present work proposes a novel framework called the regime-switching Markov decision process (RSMDP) scheme to represent a dynamic and unstructured global environment. Develop a novel optimal path planner by integrating an online reinforcement learning approach into the RSMDP scheme to avoid unknown or unpredictable moving obstacles
2. To address the constraint problem explicitly in the iteration process as well as the indicated four challenges together, a new iLQR-based feedback controller called PWA-ELQR is proposed in the present work. The idea of PWA-ELQR comes from the extended linear-quadratic-regulator (ELQR) [17] which is inspired by the extended Kalman filter. It incorporates a piecewise affine structure into ELQR to solve constraints explicitly, with the help of quadratic programming (QP). The entire iteration process begins with an arbitrary initial trajectory. Then it iteratively linearizes the robot system and quadratizes the cost function at the states of the previous nominal trajectory forwardly and backwardly. Next, it adds the cost functions of the forward pass and the backward pass to get approximate total-cost functions. Lastly, it minimizes the approximate total-cost functions to progressively obtain a better knowledge of the robot's future trajectory, and uses the new trajectory to repeat the entire process.
3. Design a classifier trained by an multiple kernel learning-support vector machine (MKL-SVM) method, that comes from an off-the-shelf open source, to detect moving people in sequential images from a video stream, where applying multiple features consisting of HOG + HOF features, which had been

proposed in literature, for detecting moving people. Then combine multiple features and the designed MKL-SVM classifier as the people detector for moving people.

1.5 Thesis Outline

The rest of this dissertation is organized as follows:

Chapter 3 presents a robust Q-learning method for global path planning in a dynamic environment. The method consists of three steps: first, a regime-switching Markov decision process,RSMDP, is formed to present the dynamic environment; second a probabilistic roadmap PRM is constructed, integrated with the RSMDP and stored as a graph whose nodes correspond to a collision-free world state for the robot; and third, an online Q-learning method with dynamic step size, which facilitates robust convergence of the Q-value iteration, is integrated with the PRM to determine an optimal path for reaching the goal. In this manner, the robot is able to use past experience for improving its performance in avoiding not only static obstacles but also moving obstacles, without knowing the nature of the obstacle motion. The use of regime switching in the avoidance of obstacles with unknown motion is particularly innovative. The developed approach is applied to a homecare robot in computer simulation. The results show that the online path planner with Q-learning is able to rapidly and successfully converge to the correct path.

Chapter 4 presents a closed-form piecewise affine control law for nonlinear-nonquadratic control problems with constraints. The existing ELQR is a valid iterative method for handling nonlinear-nonquadratic control problems. Yet it cannot explicitly deal with constraints during iteration. The present work proposes an effective method, PWA-ELQR, to take constraints into account explicitly during the iteration process, by combining QP and ELQR smoother. In this method, QP provides a linear piecewise affine feedback control law for linear-quadratic control problems, and the ELQR smoother provides tools to approximate nonlinear-nonquadractic control problems through linearization and quadratization in the vicinity of nominal trajectories. The developed approach is tested in a complex control problem: optimal motion planning for a nonholonomic mobile robot with constraints on the control input. The simulation and experimental results demonstrate good performance, and the proposed approach is a promising feedback con-

troller for optimal motion planning with constraints.

Chapter 5 presents a classifier trained by a multiple kernel-learning support vector machine,MKL-SVM to detect a human in sequential images from a video stream. The developed method consists of two aspects: multiple features consisting of HOG features and HOF features suitable for moving objects, and combined nonlinear kernels for SVM. For the purpose of real time application in autonomous navigation, the popular open-source algorithm called simple multiple kernel learning (SimpleMKL) is implemented into the proposed MKL-SVM classifier. It is able to converge rapidly with sufficient efficiency through a weighted 2-norm regularization formulation with an additional constraint on the weights. The classifier is compared with the state-of-the-art linear SVM using a dataset called *TUD*-Brussels, which is available on line. The results show that the proposed classifier outperforms the Linear SVM with respect to accuracy.

Chapter 6 presents a physical experimental platform, called SegPanda, to test the proposed methods in this dissertation, as well as the hardware and software used for SegPanda. The designed experiment combines into one autonomous navigation scenario all of the three key aspects: path planning, motion planning and people detection, which shows good performance based on the proposed methods.

The conclusions of the dissertation and suggestions for future research are summarized in Chapter 7.

Chapter 2

Robot Learning

2.1 Introduction

Robot Learning, an application of machine learning approaches to the physical world of robotics, is a research area that brings together the methods to endow robots with learning capabilities. Learning implies that every time the observation about the environment is obtained, the robot will be able to perform their activities better in view of the knowledge/experience gained through the learning process. Historically, industrial robots did not possess learning capabilities. They were used in controlled contexts such as factories to perform predefined specific tasks repeatedly with little uncertainty in interaction with humans or in a varying environment. Recently, the goal of developing autonomous robots that have capabilities of adapting to unstructured or unknown and dynamic environments has provided a good rationale for incorporating learning capabilities into robots.

There are many ways by which the robots may learn and exploit that capability. They may learn by adjusting parameters based on an outcome, build environmental models such as maps, exploit patterns, evolve rule sets for condition-action pairs, generate entire behaviors, devise new strategies for actions, predict environmental changes, recognize the strategies of the opponents or exchange knowledge with other robots. Such capability in robots would relieve humans from much of the drudgery of intervention to correct robotic actions and would potentially allow operation in environments that are unstructured, changeable, unpredictable or only

partially known. In view of (but not limited to) the mentioned advantages of learning, autonomous robots with the capability has been an important goal of robotics, artificial intelligence, and the cognitive sciences.

Although boundaries between different learning methods are blurred, basically robot learning is divided into the following three classifications in terms of what type of feedback can be received [38]:

Supervised Learning: In supervised learning, the robot observes some input-output pairs in the form of a given training set, and discovers a function that maps from input to output. Methods such as support vector machine(SVM) [39, 40], artificial neural networks (ANN) [41] and imitation learning (IL) [42] belong to this category.

Unsupervised Learning: In unsupervised learning, the robot needs to deduce various patterns by observing the input without explicit feedback from the output. evolutionary learning (EL) [43, 44] falls into this category.

Reinforcement Learning: In reinforcement learning, unlike the forgoing two types of learning, there is no available input in the beginning for the robot. It requires the robot to interact with its environment to acquire the input knowledge. Such a trial-and-error mechanism prompts the robot to find the optimal policy, which establishes the mapping between states and actions. Policy iteration [45] and value iteration [46] are common methods that belong to this category.

In addition, increasingly, researchers have adopted hybrid learning strategies to adapt to a dynamic environment. For example, the work reported in [1, 47, 48] employs inverse reinforcement learning methods to recover the intent of the teacher by modeling his cost function, and subsequently, derives the policy that is optimal with respect to the cost-to-go. The work in [49] combines ANN with imitation learning to teach goal finding and obstacle avoidance to a Nomad 200 mobile robot.

In the context of autonomous navigation in a dynamic home environment, the research presented in this dissertation mainly concentrates on suitable methods that belong to supervised learning and reinforcement learning (RL). The associated background and the algorithms (Algorithm 1-6) which had been proposed in the past by other researchers are introduced in the following sections.

2.2 Reinforcement Learning

2.2.1 Markov Decision Process

Much attention has been given recently [50–52] concerning the formulation of the use of robot learning as either Markov decision process (MDP) or a partially observable Markov decision process (POMDP) framework. Both MDP and POMDP perform very well in a dynamic environment, but their disadvantages are also obvious. MDP requires complete observation of states, and POMDP has a heavy computational burden of exact planning since it needs to estimate various possible uncertainties, thereby making the method unfeasible in just about any practical problem of robotics, without further extension [53].

In the present research, the approximation process in path/motion planning inevitably leads to the introduction of noise and uncertainty into the autonomous navigation task. Hence, explicit consideration of stochastic elements and uncertainty cannot result in adequate benefits for an autonomous navigation task. Therefore, unless explicitly stated, the entire system is treated as a deterministic one and MDP is chosen as the framework for the reinforcement learning methods. MDP

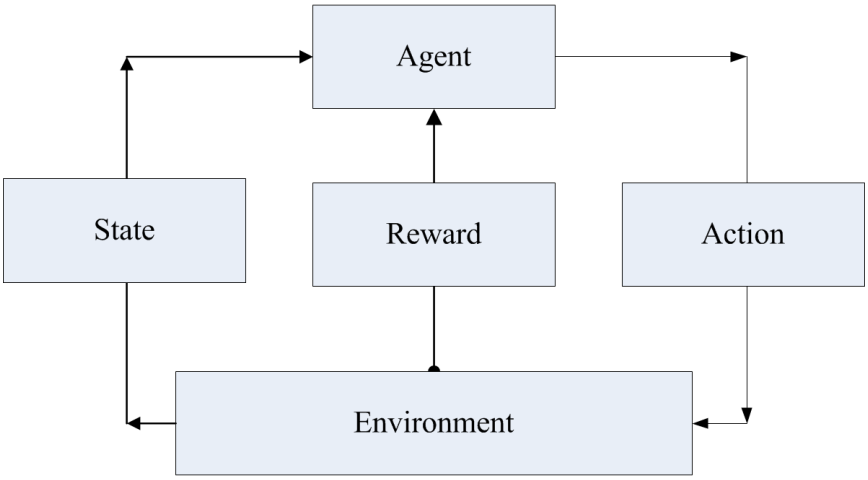


Figure 2.1: Basic MDP scheme for modeling the environment when robot learning is applied.

is a tuple (S,A,P,R) where $S = \{s_1, s_2, ..., s_N\}$ is a set of states, being infinite or

finite, discrete or continuous; $A = \{a_1, a_2, ..., a_N\}$ is a set of actions, being infinite or finite, discrete or continuous; $P(\cdot \mid \cdot, \cdot) : S \times S \times A \to \mathbb{R}_{\geqslant 0}$ is a transition function that satisfies $\sum_{s' \in S} P(s' \mid s, a) = 1$ for all $s \in S$ and $a \in A$; $R(\cdot, \cdot, \cdot) : S \times A \times S \to \mathbb{R}$ is an immediate reward (cost) function for all $s \in S$ and $a \in A$. Some researchers also use the form of (S, A, P, R, TC) where $TC : S \to \mathbb{R}$ is a terminal cost function denoting an end of an MDP, to especially express the impact of the terminal state. The transition function P and the reward function R together define the model of the MDP. The basic MDP scheme is shown in Fig. 2.1. Based on the status of P and R, the MDP problems can be further divided into subtypes. It is termed a a deterministic MDP if P or R is deterministic, otherwise it is called a stochastic MDP (e.g. P or R is a probabilistic function). In addition, it is termed a model-free MDP if P or R is unknown in advance. Conversely, it is called model-based MDP if both P and R are known in advance. The fundamental property of the MDP assumes that the current state s possesses all the information of the historical states, which implies that all of the future states are determined only by the current state, and not using the historical states. This assumption plays a significant role when applying the MDP. The following introduces the core components of an MDP.

Policies A policy π in an MDP acts as an mapping between a state $s \in S$ and an action $a \in A$. The mathematical form of deterministic policy π can be defined as $\pi : S \to A$. It also has the stochastic form defined as $\pi : S \times A \to [0, 1]$ such that $\pi(s, a) \geq 0$ and $\sum_{a \in A} \pi(s, a) = 1$ for each $s \in S$. In the present research, policies are assumed to be deterministic. Given a policy π, the MDP works in the following way: 1. Get an initial state s_0 using either a random mechanism or a predefined function (e.g., initial state distribution); 2. Obtain the responded action $a_0 = \pi(s_0)$ according to π; 3. With the obtained s_0 and a_0, predict the next state s_1 and the associated reward r_0 relying on the transition function $T(s_0, a_0, s_1)$ and the reward function $R(s_0, a_0, s_1)$; 4. Previous steps continue to get a sequence of state-action-reward process in the form $s_0, a_0, r_0, s_1, a_1, r_1 ....$ If the state-action-reward process ends in a terminal state s_{goal} and is restarted in a new random state s_0, the process from s_0 to s_{goal} is called one episode associated to one π. The objective of learning in an MDP is to find an optimal policy π^* which is able to gather the largest reward in an episode.

Optimality The optimality in an MDP is expressed by optimizing the gathered rewards. There are three models of gathering rewards, as given by:

$$E[\sum_{t=0}^{h} r_t] \tag{2.1}$$

$$E[\sum_{t=0}^{h} \gamma^t r_t] \tag{2.2}$$

$$\lim_{h\to\infty} E[\frac{1}{h}\sum_{t=0}^{h} r_t] \tag{2.3}$$

Equation 2.1 is the finite horizon model which takes rewards of a finite horizon of length h. Here π^* should optimize its expected rewards over this horizon and yield $h-$ steps of optimal action. The robot can either take all of the h actions or just choose the first certain amount of actions. The latter case is known as receding horizon control (RHC) or model predict control (MPC) in the context of optimal control problems. The problem that exits in this model is that the choice of the horizon length depends on the specific problem. Equation 2.2 represents the infinite horizon model, which takes long-run rewards into account. The parameter $\gamma \in [0,1)$ denotes the discount factor, which determines how many future rewards will be received. With the exponential form of γ, the further away a future reward is received, the lower the weight this future reward occupies in the entire set of rewards. Such design accords with real-world experience, since farther future rewards come with more uncertainty and hence their weights should be lower than the near future rewards. When $\gamma = 0$, only the immediate reward would be taken into account at the current state. In addition, even with infinite horizon, the sum of the gathered rewards is still finite since $\gamma \in [0,1)$ mathematically guarantees this property mathematically. Hence, many algorithms prefer to using this model. Equation 2.3 represents average-reward model, which is used when only the entire reward is the concern. Obviously this model cannot distinguish between two policies that have different rewards in different steps, because all the differences are averaged into the result. Therefore, which model should be chosen depends on the specific learning problem. If the length is known, the finite-horizon model is the best. Otherwise if the task continues infinitely and the associated episode has

no clear terminal goal, the infinite-horizon model would be more suitable. More details about choosing an optimal model are found in [54].

Value Functions Value functions in an MDP play the role of connecting the optimality models to the policies. A value function estimates how good the policy is in terms of the number of the expected rewards, which is termed as the return, the policy gathers at state s. The value function of a state s under policy π, represented as $V^{\pi}(s)$, denotes the expected return when starting in state s and following π thereafter. Using the infinite-horizon model of optimality in the present dissertation, the full form of the value function is expressed as:

$$V^{\pi}(s_t) = E_{\pi}\{\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s\} \tag{2.4}$$

Without of loss generality, the expectation symbol E is used in the above expression and in later sections to handle the stochastic MDP. However, E can be disregarded if only deterministic MDP is considered.

Sometimes an analogous state-action value function termed as Q-function may be defined as:

$$Q^{\pi}(s_t, a_t) = E_{\pi}\{\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s, a_t = a\} \tag{2.5}$$

where $Q : S \times A \rightarrow \mathbb{R}$. With Q-function, the state $s \in S$ and action $a \in A$ can be chosen without knowing the transition function T, so that it is suitable for model-free approaches.

Value function 2.4 possesses a fundamental recursive property, and hence the so-called Bellman equation can be deduced as follows:

$$\begin{aligned} V^{\pi}(s_t) &= E_{\pi}\{\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s\} \\ &= E_{\pi}\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \mid s_t = s\} \\ &= E_{\pi}\{r_t + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\} \\ &= \sum_{s_{t+1}} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma V^{\pi}(s_{t+1})). \end{aligned} \tag{2.6}$$

It denotes that an value function consists of the immediate reward, and the value of

the possible next state weighted by a discount factor. In addition, the reward-next-state set may be multiple due to transition probabilities. Each of the reward-next-state sets is weighted by the associated transition probabilities. The optimal value function $V^*(s_t)$ can be obtained by solving the following optimization problem:

$$V^*(s_t) = \max_{a \in A} \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma V^*(s_{t+1})). \tag{2.7}$$

The associated optimal policy can be obtained by:

$$\pi^*(s_t) = \arg\max_{a \in A} \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1}) \left(R(s_t, \pi(s_t), s_{t+1}) + \gamma V^*(s_{t+1})\right). \tag{2.8}$$

As in Equation 2.5, an analogous Q-function version of the Bellman equation and the related optimal policy are defined as:

$$Q^*(s_t, a_t) = \max_{a \in A} \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma Q^*(s_{t+1}), a_{t+1}), \tag{2.9}$$

$$\pi^*(s_t) = \arg\max_{a \in A} Q^*(s_t, a_t) \tag{2.10}$$

Policy Q^* and V^* is called a greedy policy because it selects the best action according to the maximum mechanism. The relation between Q^* and V^* is given by:

$$V^*(s_t) = \max_{a \in A} Q^*(s_t, a_t) \tag{2.11}$$

2.2.2 Dynamic Programming: Model-based Method for Solving MDP

Once all of the components of the MDP have been defined, explicitly solving the optimal policy from the value functions becomes the next step. One method of solution uses dynamic programming (DP). It represents a class of algorithms that is capable of computing optimal policies given a perfect model (P and R) of the MDP problem. Classical DP algorithms have limited application in reinforcement learning due to the requirement of a perfect model and its heavy computation. However, DP serves as an essential foundation since such assumption leads to good properties in terms of mathematical deduction. Hence, most other methods can be viewed

as attempts to achieve similar results as DP by transferring the original conditions to the perfect-model assumption, with some added noise. Also, because of such mathematical benefits, DP has been widely used in the optimal control domain, where the state vector is denoted by the set of $x = \{x_1, x_2, ..., x_N\}$ and the input vector by the action set $u = \{u_1, u_2, ..., u_N\}$, in order to differentiate from the planning problems. This situation will be encountered in Section 4.3 where an optimal motion controller for the motion planning function of autonomous navigation is proposed.

Two typical DP algorithms are policy iteration [55] and value iteration[56], which provide two fundamental structures for various variants and combinations. They are discussed now.

Policy Iteration

Policy iteration includes two phases: policy evaluation and policy improvement. These two phases iterate to a convergent result (optimal policy) that satisfies certain threshold conditions, which are used for terminating the iteration process. Fig. 2.2 shows the basic scheme of the entire iteration process.

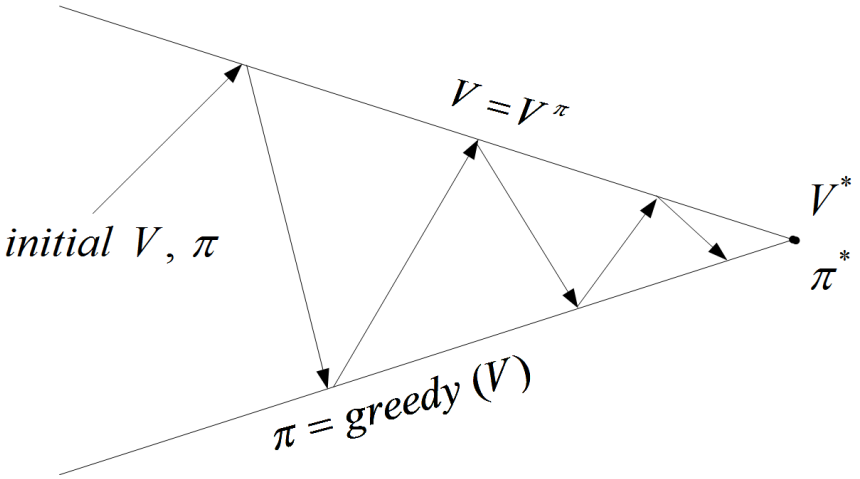


Figure 2.2: Basic MDP scheme for modeling the environment where robot learning are applied into.

Policy Evaluation This phase computers the value function of the k-th iteration $V_k^{\pi}(s_t)$ for each state $s_t \in S$ in the current policy π, based on Equation (2.6). In each iteration as k goes to infinity, the old value for the state s_t is replaced until convergence, based on the expected value of possible successor states. This sequence of updating $V_k^{\pi}(s_t)$ is termed *backup* operation. By defining a backup operator B^{π} over the value functions, the policy evaluation phase can be expressed as:

$$(B^{\pi}V)(s_t) = \sum_{s_{t+1}} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma V^{\pi}(s_{t+1})). \tag{2.12}$$

Policy Improvement This phase improves the current policy with respect to the newly available value functions obtained from the policy evaluation phase. Once the value functions for the states are known, the next step will determine if there is an action $a_t \in A$ that can improve the value function of the associated state s_t by using:

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= E_{\pi}\{r_t + \gamma V^{\pi}(s_{t+1}) \mid s_t, a_t\} \\ &= \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma V^{\pi}(s_{t+1}), a_{t+1}) \end{aligned} \tag{2.13}$$

If now $V^{\pi}(s_t)$ is less than $Q^{\pi}(s_t)$ for $a_t \in A$, then it means there is a better policy for this particular state. Similarly, the entire original policy can be improved by greedily selecting the best action in each state as follows:

$$\begin{aligned} \pi'(s_t) &= \arg\max_{a \in A} Q^{\pi}(s_t, a_t) \\ &= \arg\max_{a \in A} E_{\pi}\{r_t + \gamma V^{\pi}(s_{t+1}) \mid s_t, a_t\} \\ &= \arg\max_{a \in A} \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1}) \left(R(s_t, \pi(s_t), s_{t+1}) + \gamma V^{*}(s_{t+1})\right). \end{aligned} \tag{2.14}$$

The iterative procedure of these two phases generates a sequence of alternating policies and value functions. The complete algorithm of policy iteration is given in Algorithm 1.

Algorithm 1 The Policy Iteration Algorithm

Input: initial policy $\pi(s_t)$ and value function $V^{\pi}(s_t)$ for all $s_t \in S$; iteration threshold σ

Output: optimal policy $\pi^*(s_t)$

1: Policy Evaluation
2: **repeat**
3: $\delta := 0$
4: **for** each step of episode t **do**
5: **for** each $s_t \in S$ **do**
6: $v := V^{\pi}(s_t)$
7: $V(s_t) := \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma V(s_{t+1}))$
8: $\delta = \max(\delta, \mid v - V(s) \mid)$
9: **end for**
10: **end for**
11: **until** $\delta < \sigma$
12: Policy Improvement
13: policy-state:=true
14: **for** each step of episode t **do**
15: **for** each $s_t \in S$ **do**
16: $b := \pi(s_t)$
17: $\pi(s_t) := \arg\max_{a \in A} V(s_t)$
18: **if** $b \neq \pi(s_t)$ **then**
19: policy-state:=false
20: **end if**
21: **end for**
22: **end for**
23: **if** policy-stable **then**
24: $\pi^* = \pi$
25: Stop Iteration
26: **else**
27: go to Policy Evaluation
28: **end if**

Value Iteration

One disadvantage of the policy iteration algorithm 1 is that the policy evaluation phase can provide a useful value function only when it satisfies the convergence

Algorithm 2 The Value Iteration Algorithm

Input: initial policy $\pi(s_t)$ and value function $V^{\pi}(s_t)$ for all $s_t \in S$; iteration threshold σ

Output: optimal policy $\pi^*(s_t)$

repeat
  $\delta := 0$
  for each step of episode t **do**
    for each $s_t \in S$ **do**
      $v := V^{\pi}(s_t)$
      $V(s_t) := \max_{a \in A} \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma V(s_{t+1}))$
      $\delta = \max(\delta, \mid v - V(s) \mid)$
    end for
  end for
until $\delta < \sigma$
$\pi^*(s_t) = arg\max_{a \in A} \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1}) (R(s_t, \pi(s_t), s_{t+1}) + \gamma V^*(s_{t+1}))$

condition $\delta < \sigma$ in each iteration, which may cost significant computation. However, in some cases, the policy evaluation phase does not need to exactly converge in each iteration. Then, truncating it to save computational time will not hurt the convergence guarantee of the policy iteration[57].One traditional approach is to let the policy evaluation stop after only one step backup operation, and then execute the policy improvement. This algorithm is called value iteration, which may be expressed as follows:

$$\begin{aligned} V_{k+1}(s_t) &= \max_{a_t \in A} E\{r_{t+1} + \gamma V_k^{\pi}(s_{t+1}) \mid s_t, a_t\} \\ &= \max_{a_t \in A} \sum_{s_{t+1} \in S} P(s_t, \pi(s_t), s_{t+1})(R(s_t, \pi(s_t), s_{t+1}) + \gamma V_k(s_{t+1}), a_{t+1}). \end{aligned} \tag{2.15}$$

The complete value iteration algorithm is given in Algorithm 2.

2.2.3 Reinforcement Learning: Model-free Method for Solving MDP

RL is a class of algorithms that obtain an optimal policy for problems where a model of MDP is not available in advance. The underlining objective in RL con-

centrates on finding the estimated model for MDP. Due to the lack of a model, RL needs to sample and explore the MDP to acquire experience (statistical knowledge) about the unknown model, in order to establish the estimated value functions, as in DP methods. Basically the RL algorithms for MDP can be divided into two categories in terms of how to deal with the employed policy during learning: off-policy methods and on-policy methods. In off-policy methods, the policy used for generating behaviors (actions) is independent of the policy that is evaluated and improved. On the contrary, on-policy methods attempt to evaluate and improve the same policy that is used for generating behaviors. As a result, exploration must be built into the policy to find a possible improvement, and the speed of the policy improvements should be fast. These two different aspects usually oppose each other; specifically, strengthening one aspect will weaken the other aspect. This is called exploration-exploitation trade-off problem. The most basic exploration strategy is the ε-greedy method in which the best action is taken at probability $(1-\varepsilon)$ and the other action is randomly selected at probability ε. Many other exploration methods can be found in[58, 59]. In the following sections, two types of RL methods, Monte Carlo (MC) methods and temporal difference learning (TDL) methods, are introduced.

Monte Carlo Methods

MC comes from the law of large numbers in random theory; namely, in the limit, the expected value of a set of samples is equal to the average value (mean). That's the reason why MC is a class of algorithms solving RL that rely on averaging sample returns. It samples sequences of states, actions, and rewards from interaction with either actual or simulated environment to acquire an estimated model instead of a real model. Then the estimated model provides approximate transition and reward functions that are required by model-based methods. In addition, to guarantee satisfying principal conditions, the experience of MC is learned episode by episode rather than step by step. This means the value functions and the policies get updated only after one episode is finished, with the assumption that all episodes eventually reach the terminal state in the limit no matter what actions are selected.

Another property of MC is that it prefers to estimate the state-action value

functions $Q^{\pi}(s,a)$ under one policy π other than state value functions $V(s)$, since $V(s)$ alone is not sufficient to provide sample information for estimating the transition and reward functions when a model is not available. Therefore, all of the following MC methods use action value functions $Q^{\pi}(s,a)$. Suppose that a given set of episodes is sampled by following π and passing through state s. Each occurrence of state s in an episode is called a visit to s. If the estimated $Q^{\pi}(s,a)$ comes from the average of the returns following all the visits to s in a set of episodes, it is termed the every-visit MC method. Similarly, first-visit method uses the average of returns following just the first visit to s in a set of episodes. The first-visit method has been widely researched and hence the following section uses it to express MC methods.

On-Policy MC The overall scheme of on-policy MC is similar to DP methods. It also includes policy evaluation and policy improvement. As introduced before, on-policy MC improve the policy that is currently used for generating actions. An $\varepsilon - greedy$ policy exploration is built into the process of learning the optimal policy. An example of on-policy MC algorithm is given in Algorithm 3.

Off-Policy MC Off-policy MC distinguishes from on-policy MC in that the policy used to generate actions is not the one that is used for evaluation, as given in Algorithm 4.

Temporal Difference Learning

TDL is a class of algorithms combining the advantages of both DP and MC. It can learn the experience directly from interaction with the environment without a model of the environment like MC does, as well as update estimates by using part of other learned estimates without waiting for an entire episode to end like DP does. With all of these advantages, TDL has become an important breakthrough in RL because it can be applied to a wide situations such as the case where the episodes are very long or where there is no episodes at all. TDL update the estimates in an incremental manner based on other estimates, so that each step of update will generate a useful experience. The simplest TDL method TD(0) is given as an

Algorithm 3 The On-Policy Monte Carlo Algorithm

Input: for all $s_t \in S, a_t \in A(s)$: initial policy $\pi(s)$ with $\varepsilon - greedy$:= arbitrary; initial state-action value function $Q(s,a)$:=arbitrary; $Retures(s,a)$:=empty list; iteration threshold σ

Output: optimal policy $\pi^*(s)$

1: **repeat**
2: Generate an episode using π
3: **for** each step $t = 0, 1, 2, ..., T$ of the generated episode **do**
4: **for** each state-action pair s_t, a_t appearing in each episode **do**
5: $R :=$ the return following the first-visit MC
6: Append R to $Retures(s,a)$ (not (s_t, a_t), since the (s_t, a_t) may be same in different t)
7: $q := Q(s,a)$
8: $Q(s,a) := average(Retures(s,a))$
9: $\delta = \max(\delta, \mid q - Q(s,a) \mid)$
10: **end for**
11: **for** each state s appearing in each episode **do**
12: $a^* = \arg\max_a Q(s,a)$
13: for all $a \in A(s)$:$\varepsilon - greedy := \begin{cases} 1 - \varepsilon + \varepsilon / \mid A(s) \mid & if\ a = a^* \\ \varepsilon / \mid A(s) \mid & if\ a \neq a^* \end{cases}$
14: **end for**
15: **end for**
16: **until** $\delta < \sigma$
17: $\pi^*(s) = \pi(s)$

example:

$$V_{s_t} = V_{s_t} + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V_{s_t}], \tag{2.16}$$

where α is a constant step-size parameter denoting the learning speed and γ denotes the discount factor of the future estimate. From Equation (2.16), it is noted that TD(0) updates the estimated value function $V_(s_t)$ only until the next time step $t+1$ and by using the observed reward r_{t+1} and the estimated value function of next state $V_(s_{t+1})$. In MC, the return R_t, which can be obtained only when one episode ends, is used for the value function, whereas the observed reward r_{t+1} is used in TD(0). Hence the target for the TD update is $r_{t+1} + \gamma V_t(s_{t+1})$ instead of R_t in MC.

Algorithm 4 The Off-Policy Monte Carlo Algorithm

Input: for all $s_t \in S, a_t \in A(s)$:
initial policy $\pi(s)$:= arbitrary;
initial state-action value function $Q(s,a)$:=arbitrary;
$Nu(s,a)$:=0; Numerator of $Q(s,a)$
$De(s,a)$:=0; Denominator of $Q(s,a)$
iteration threshold σ
Output: optimal policy $\pi^*(s)$

1: **repeat**
2: **for** each step $t = 0, 1, ..., T$ of the generated episode **do**
3: Generate an episode $\{s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T, s_t\}$ by selecting a policy π'
4: τ=last time at which $a_\tau \neq \pi(s_\tau)$
5: **for** each state-action pair s_t, a_t appearing in each episode **do**
6: t=the time of first visit (after τ) of (s,a)
7: $\omega := \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k,a_k)}$
8: $Nu(s,a) := Nu(s,a) + \omega R_t$
9: $De(s,a) := De(s,a) + \omega$
10: $Q(s,a) := \frac{Nu(s,a)}{De(s,a)}$
11: $q := Q(s,a)$
12: $\delta = \max(\delta, \mid q - Q(s,a) \mid)$
13: **end for**
14: **for** each state s appearing in each episode **do**
15: $\pi(s) = \arg\max_a Q(s,a)$
16: **end for**
17: **end for**
18: **until** $\delta < \sigma$
19: $\pi^*(s) = \pi(s)$

As discussed under MC, TDL also has the requirement to build approximate transition and reward functions, and have to trade off exploration and exploitation by some policies like the $\varepsilon - greedy$ exploration policy. Accordingly, the state-action value function Q^π of TDL also has similar challenges. All of these methods can also be divided into two categories: on-policy methods and off-policy methods. In the following, an on-policy TDL, SARSA, and an off-policy TDL, Q-learning, are introduced.

Algorithm 5 The SARSA Algorithm

Input: for all $s_t \in S, a_t \in A(s)$:
initial state-action value function $Q(s,a)$:=arbitrary;
iteration threshold σ
Output: optimal policy $\pi^*(s)$

```
repeat
  for each episode do
    Randomly choose a state s as starting state
    Choose a for s using policy derided from Q(s,a) based on ε − greedy exploration
    repeat
      for each step t = 0,1,...,T of the generated episode do
        Choose a_t for s_t, observe reward r_t and the next state s_{t+1}
        Choose a_{t+1} for s_{t+1} using policy derived from Q(s,a) based on ε − greedy exploration
        Q_{s_t,a_t} = Q_{s_t,a_t} + α[r_{t+1} + γQ(s_{t+1},a_{t+1}) − Q_{s_t,a_t}]
      end for
    until s_t reaches the terminal state
  end for
  q := Q(s,a)
  δ = max(δ, | q − Q(s,a) |)
until δ < σ
π*(s) = arg max_a Q(s,a))
```

SARSA: On-Policy TDL In SARSA methods, the policy for generating actions is the same as the one to update evaluation. Each evaluation update occurs after every transition from a state-action pair (s_t, a_t) to next state-action pair (s_{t+1}, a_{t+1}), and receives a feedback reward r_t, as expressed in the following equation:

$$Q_{s_t,a_t} = Q_{s_t,a_t} + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q_{s_t,a_t}], \tag{2.17}$$

Here s_t is not a terminal state. If state s_{t+1} is terminal, then define $Q(s_{t+1}, a_{t+1})$ as zero. All of the elements of Equation 2.17 forms a quintuple of $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ that makes up a transition. This is why this algorithm is called SARSA. The complete algorithm is given in Algorithm 5.

Q-learning: An Off-Policy TDL method Q-learning is an effective method and a significant development of off-policy TDL. It is different from SARSA in that it uses a build-in max operator over the function values of the next state-action pair when updating every transition. The core algorithm is given by:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t(s_t,a_t) + \gamma \max_a Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \qquad (2.18)$$

where the value of $Q(s_t,a_t)$ is termed Q value. Q-learning builds its experience base through establishing a Q table. An example is presented here to demonstrate how to establish a Q table.

Suppose that a static collision-free roadmap has been obtained as shown in Fig. 2.3. According to the definitions given in Section 3.3, each node represents one state s, and action a is represented by the arrows around each edge. The reward r after executing an action will be 10 if it arrives at the goal state s_{11}; otherwise, it will be 0 after executing one action. Suppose that the mobile robot is in state s_1, and it has four action options: go to state s_1,s_2,s_4 and s_5. If it chooses state s_5, the value of r is set equal to 0. It cannot go to state s_7 because there is no edge between them, which means that the space between them is blocked either by moving obstacles or by static obstacles. Let us express such a state diagram and the instant reward values by the reward table, matrix **R**, given in Fig. 2.4.

After building the reward matrix **R**, the next step is to build the Q table according to matrix **R** and set the Q-learning algorithm given by Equation (2.18). In this dissertation, Q table is also expressed in the form of a matrix **Q**. We will start by setting all the Q values to zero as shown in Fig. 2.5, and the initial state as state s_1. According to the second row of matrix **R** there are four possibilities for the current action. By the strategy of uniform random selection among these four actions, we select to go to state s_2 without loss of generality. Suppose now that the robot is in state 2. Then according to the third row of matrix **R**, there are four possible actions: go to state s_1, s_3,s_5 or s_6. The corresponded transition is:

$$Q(1,2) = Q(1,2) + \alpha\{R(1,2) + \gamma \max[Q(2,1),Q(2,3),Q(2,5),Q(2,6) - Q(1,2)]$$

Now the next state s_2 becomes the current state. We repeat the above steps until the

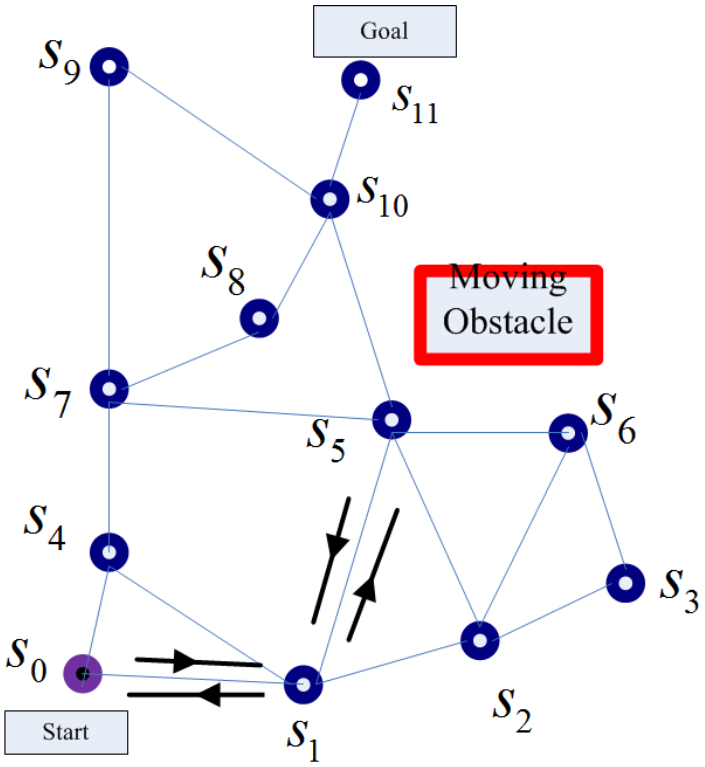


Figure 2.3: Static collision-free roadmap for establishing Q table.

robot arrives at the goal state s_{11}. In this manner we have completed one episode. We begin the next episode with a randomly chosen initial state, say state s_7, and update the matrix **Q** through similar calculation using Equation (2.18). Episodes are repeated until each Q value converges to a certain optimal value. Then we can choose the optimal policy for starting state s_0 to goal state s_{11} according to the matrix **Q**. Fig. 2.6 and Fig. 2.7 show the results of the Q value after 5 and 50 iterations, respectively. The value in the red square shows the updating process of the Q value of the state-action pair (s_1, a_2). Note that the converged values of matrix **Q** will not remain unchanged forever since it is initially obtained using a static environment. The robot will update its Q value when interacting with the real-time dynamic environment. The complete Q-learning algorithm is given in Algorithm 6[60].

States \ Actions	0	1	2	3	4	5	6	7	8	9	10	11
0	-1	0	-1	-1	0	-1	-1	-1	-1	-1	-1	-1
1	0	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1
2	-1	0	-1	-1	-1	0	0	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1
4	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
5	-1	0	0	-1	-1	-1	-1	0	-1	-1	0	-1
6	-1	-1	0	0	-1	-1	-1	-1	-1	-1	-1	-1
7	-1	-1	-1	-1	0	-1	-1	-1	0	0	-1	-1
8	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	0	-1
9	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	0	-1
10	-1	-1	-1	-1	-1	-1	-1	-1	0	0	-1	10
11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	10

Figure 2.4: Reward matrix **R** (-1 means there is no edge between nodes, 0 means an arrived node is not a goal node, 10 means an arrived node is a goal node).

2.3 Support Vector Machine: A Popular Method of Supervised Learning

Many applications of robotic technologies require the solution of classification problems. Detecting people, for instance, is a binary classification problem, or say yes-or-no classification to answer if the tested area contains people. SVM [61, 62] is among the best "out-of-box" supervised learning algorithms to solve classification problems. Given a training set of sample-label pairs $S = \{(\mathbf{x}_i, y_i), i = 1,2,...,m\}$ where $\mathbf{x}_i \in \mathbb{R}^n$ denotes the samples and $y \in \{1,-1\}^l$ denotes the class labels, the SVM works out the optimal parameters w,b with respect to the training set S for the linear classifier defined as:

$$h_{\mathbf{w},b}(x) = g(\mathbf{w}^T\mathbf{x} + b), \tag{2.19}$$

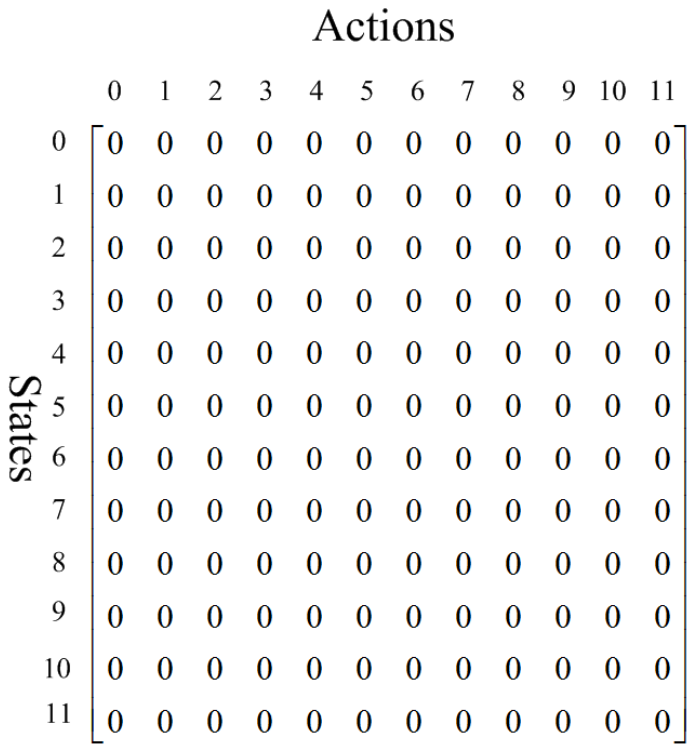


Figure 2.5: Q table in the form of matrix **Q** at the initial iteration.

where $g(\mathbf{w}^T\mathbf{x}+b)=1$ if $\mathbf{w}^T\mathbf{x}+b\geq 0$ and $g(\mathbf{w}^T\mathbf{x}+b)=-1$ if $\mathbf{w}^T\mathbf{x}+b<0$. Therefore, $\mathbf{w}^T\mathbf{x}+b=0$ is called the decision boundary, or the separating hyperplane if the dimension of the feature $\mathbf{x}$ is higher than 2. Fig. 2.8 shows the basic scheme of linear classifier.

2.3.1 Geometric Margins

With the definition of $(\mathbf{w},b)$, the geometric margin of $(\mathbf{w},b)$ with respect to each training sample $(\mathbf{x}_i,y_i)$ is defined as the shortest distance from the sample $(\mathbf{x}_i,y_i)$ to the decision boundary $\mathbf{w}^T\mathbf{x}+b=0$. It is denoted by the symbol η_i as shown in Fig. 2.9. In Fig. 2.9, point A represents a sample $\mathbf{x}_i$ and point B is the cross point generated by the orthogonal vector of the decision boundary which goes through point A. Segment AB is hence the distance from point A to the decision boundary,

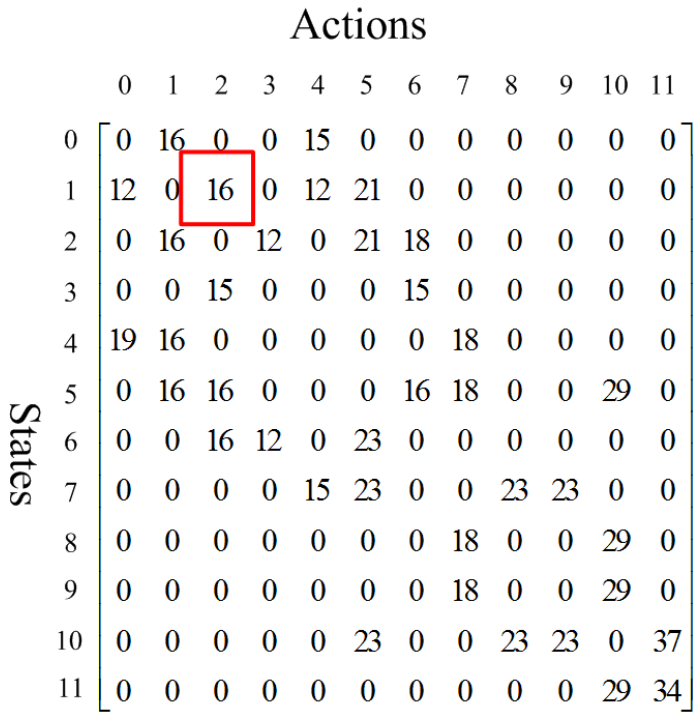


Figure 2.6: Q table in the form of matrix **Q** at the iteration 5.

denoted as the geometric margin η_A.

Since A is $\mathbf{x}_i$, point B can be given by $\mathbf{x}_i - \eta_i \times \frac{\mathbf{w}}{\|\mathbf{w}\|}$. Note that point B lies on the decision boundary. Hence $\mathbf{w}^T \left(\mathbf{x}_i - \eta_i \times \frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b = 0$, and the η_i can be solved as:

$$\eta_i = \left(\frac{\mathbf{w}}{\| \mathbf{w} \|}\right)^T \mathbf{x}_i + \frac{b}{\| \mathbf{w} \|}. \tag{2.20}$$

In Equation (2.20), the solution of η_i is obtained for the case of a positive sample A. Accordingly, the general solution of η_i is:

$$\eta_i = y_i \left(\left(\frac{\mathbf{w}}{\| \mathbf{w} \|}\right)^T \mathbf{x}_i + \frac{b}{\| \mathbf{w} \|} \right). \tag{2.21}$$

The forgoing geometric margin is associated with one training sample. Moreover, the geometric margin of $(\mathbf{w}, b)$ associated with the entire training set S is

State \ Action	0	1	2	3	4	5	6	7	8	9	10	11
0	0	22	0	0	17	0	0	0	0	0	0	0
1	28	0	22	0	17	28	0	0	0	0	0	0
2	0	22	0	17	0	28	22	0	0	0	0	0
3	0	0	22	0	0	0	22	0	0	0	0	0
4	17	22	0	0	0	0	0	22	0	0	0	0
5	0	22	22	0	0	0	22	22	0	0	36	0
6	0	0	22	17	0	28	0	0	0	0	0	0
7	0	0	0	0	17	28	0	0	28	28	0	0
8	0	0	0	0	0	0	0	22	0	0	36	0
9	0	0	0	0	0	0	0	22	0	0	36	0
10	0	0	0	0	0	28	0	0	28	28	0	46
11	0	0	0	0	0	0	0	0	0	0	36	46

Figure 2.7: Q table in the form of matrix **Q** at the iteration 50.

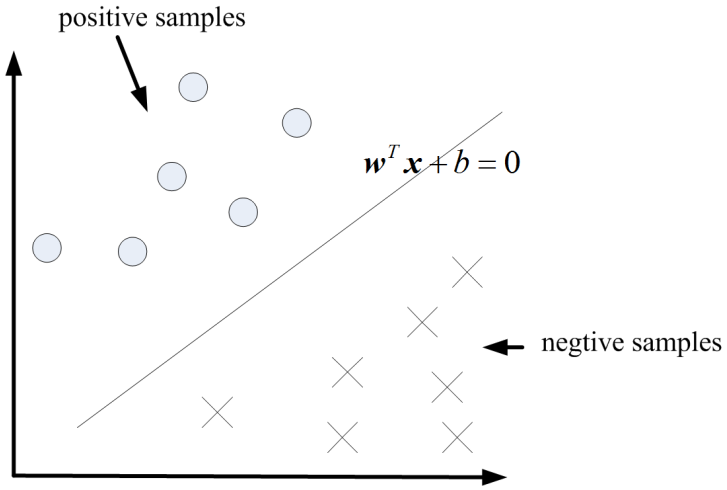


Figure 2.8: The basic scheme of linear classifier.

Algorithm 6 The Q-learning Algorithm

Input: discount factor γ, learning rate α
initial state-action value function $Q(s,a)$:=arbitrary, for all $s_t \in S, a_t \in A(s)$;
iteration threshold σ
Output: optimal policy $\pi^*(s)$

repeat
  for each episode **do**
    Randomly choose a state s as the starting state
    repeat
      for each step $t = 0, 1, ..., T$ of the generated episode **do**
        Choose a_t for s_t using policy derived from $Q(s,a)$ based on $\varepsilon -$ *greedy* exploration
        Observe reward r_t and the next state s_{t+1}, choose a_{t+1} for s_{t+1}
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t(s_t,a_t) + \gamma \max_a Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
      end for
    until s_t reaches the terminal state
  end for
  $q := Q(s,a)$
  $\delta = \max(\delta, \mid q - Q(s,a) \mid)$
until $\delta < \sigma$
$\pi^*(s) = \arg\max_a Q(s,a)$

defined as:

$$\eta = \min_{i=1,...,m} \eta_i. \tag{2.22}$$

Obviously, the bigger the η, the greater the gap between the positive training samples and the negative samples, which reflects a very confident decision boundary. Therefore, the objective of the classifier is to maximize the geometry margin η. The entire process of working out a linear classifier now becomes how to solve the following optimization problem:

$$\begin{aligned} &\max_{\eta,\mathbf{w},b} \quad \eta \\ \text{s.t.} \quad &y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq \eta, \;\; i = 1, ..., m \\ &\parallel \mathbf{w} \parallel = 1. \end{aligned} \tag{2.23}$$

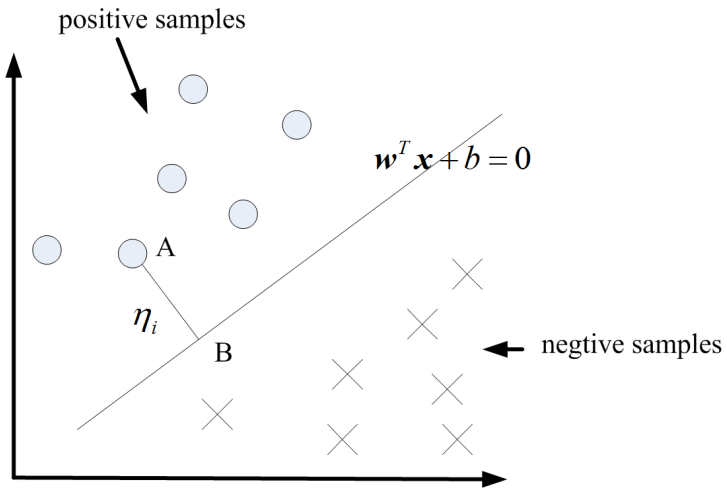


Figure 2.9: The basic scheme of geometric margins.

Apparently the optimization problem (2.23)cannot be solved by "off-the-shelf" methods, since the constraint $\| \mathbf{w} \| = 1$ is non-convex, which is an intractable optimization problem. Therefore, functional margins are imported into (2.23) in order to transfer (2.23) into a tractable optimization problem.

2.3.2 Optimal Margin Classifier

To aim at solving (2.23), the functional margin of $(\mathbf{w}, b)$ with respect to each training sample, $(\mathbf{x}_i, y_i)$ is defined as:

$$\hat{\eta}_i = y_i(\mathbf{w}^T \mathbf{x}_i + b) \tag{2.24}$$

The corresponded functional margin with respect to the training set S is defines as:

$$\hat{\eta} = \min_{i=1,\ldots,m} \hat{\eta}_i. \tag{2.25}$$

By comparing with the form of the geometry margin (2.21) with the functional margin (2.24), $\eta = \frac{\hat{\eta}}{\|\mathbf{w}\|}$ can be obtained. By integrating $\eta = \frac{\hat{\eta}}{\|\mathbf{w}\|}$ into (2.23), as well as noting that maximizing $\frac{\hat{\eta}}{\|\mathbf{w}\|} = \frac{\hat{\eta}}{\|\mathbf{w}\|}$ is the same as minimizing $\| \mathbf{w} \|^2$, a

more tractable optimization problem can be obtained:

$$\begin{aligned} \max_{\eta,\mathbf{w},b} \quad & \frac{1}{2} \parallel \mathbf{w} \parallel^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T)x_i + b) \geq 1, \;\; i = 1, ..., m \end{aligned} \tag{2.26}$$

The optimization problem (2.26) has a transitional quadratic objective with only linear constraints, which can be solved easily by many commercial methods such as the QP method. The solution gives the optimal classifier.

2.3.3 Dual Problem and Support Vectors

The previous optimization problem (2.26) is a so-called primal problem. An associated dual form of the problem will bring more benefits for solving the optimization problem (2.26). First the Lagrangian factor for (2.26) is constructed as follows:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \parallel \mathbf{w} \parallel - \sum_{i=1}^{m} \alpha_i [y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1], \tag{2.27}$$

where $\alpha = \{\alpha_i, i = 1, ..., m\}$ denotes the Lagrange multiplier. Then fix α, and minimize $L(\mathbf{w}, b, \alpha)$ with respect to $\mathbf{w}$ and b separately:

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b, \alpha) = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{2.28}$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \alpha) = \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.29}$$

Next combine Equations (2.27, 2.27, and 2.29), and consider the constraints $\alpha_i \geq 0$ together. The dual optimization problem can be given by:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, ..., m \\ & \sum_{i=1}^{m} \alpha_i y_i = 0, \end{aligned} \tag{2.30}$$

Obviously the form of the dual optimization problem (2.30) is convex so that it can be solved using optimization methods. It also shows two significant advantages. First, it transfers solving the optimal $(\mathbf{w}, b)$ into solving only one parameter α. Once the optimal α^* is solved, the optimal $\mathbf{w}^*$ and b^* can be obtained by substituting α^* back into Equations (2.28) and (2.29). More deeply, the experience on the solution of α discloses that most of α_i will be zero, with a few non-zero α which rightly associate with the training samples that have the smallest geometric margins (closest to the decision boundary). These samples are shown in Fig. 2.10 as the points that lie on the dash lines parallel to the decision boundary. These points are term **support vectors** . By using support vectors, the size of the training samples related to the solution of the optimal parameter will reduce significantly. The second advantage arises in the form of $< \mathbf{x}_i, \mathbf{x}_j >$ which is defined as **kernel**.The definition of kernel helps to map the feature space of the training samples to the feature space of higher dimension, where the nonlinear decision boundary in the low dimension space may become linear. These two advantages are the main reasons why SVM has been quite popular in the research community of classification.

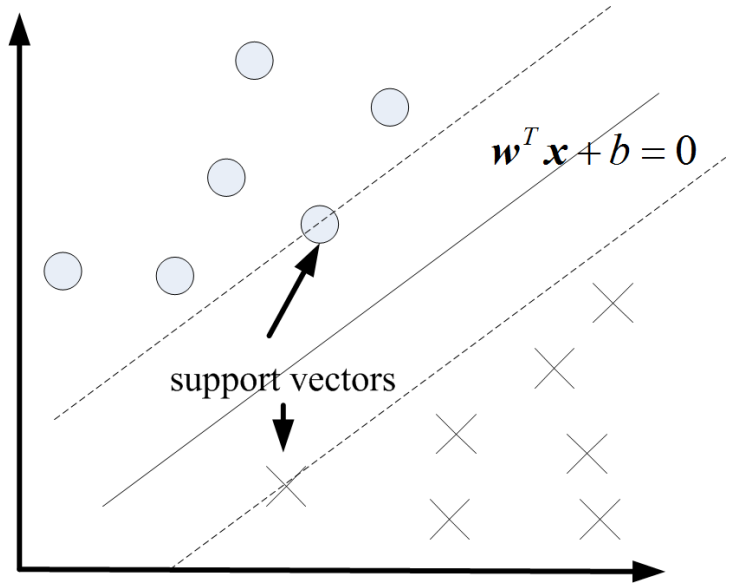


Figure 2.10: An example of the concept of support vectors for a linear classifier.

Chapter 3

Robust Q-learning with Regime-Switching Markov Decision Process for Optimal Path Planning

3.1 Introduction

This chapter presents a new path planning approach, which incorporates online reinforcement learning integrated with regime-switching Markov decision process (RSMDP) for a mobile robot moving in a dynamic environment. Modeling the surrounding dynamic environment is the first challenge. To address this challenge, a novel framework based on the RSMDP scheme is introduced to represent a dynamic environment. In addition, an online reinforcement learning approach is integrated into the RSMDP scheme to resolve the uncertainty in a model-free environment, and probabilistic roadmap(PRM), a sample-based method, is used to resolve the curse of dimensionality that arises with reinforcement learning when facing a continuous space of state and action.

The main contributions of the present chapter, in the context of the related previous work [63, 64], are as follows. First, unlike [63, 64], which consider only

static obstacles, the present path planner is able to return a globally optimal path in the presence of unknown moving obstacles, with regard to balancing the shortest path and obstacle avoidance. Second, through PRM, both state space and control space can be constrained to a low-dimensional finite space. Third, and most importantly, reinforcement learning is used in an online formation, using the concept of regime-switching [65, 66], to represent the changing environment caused by moving obstacles, where value iteration is robust to parameter changes. This appears to be the first application of regime switching to solve path planning problems in a dynamic and unstructured environment.

3.2 RSMDP

The scenario of path planning problem described in Section 1.2.2 can be formulated as an Markov decision process(MDP) because it has the Markov property that the future state depends only on the current state and has no dependence on the past states. An MDP is defined by a tuple with five elements: $M = (S, A, P, R, TC)$ where S is a set of states, A is a set of actions depending on S, $P(\cdot \mid \cdot,\cdot) : S \times S \times A \to \mathbb{R}_{\geqslant 0}$ is a transition probability function that satisfies $\sum_{s' \in S} P(s' \mid s, a) = 1$ for all $s \in S$ and $a \in A$, $R(\cdot,\cdot,\cdot) : S \times A \times S \to \mathbb{R}$ is an immediate cost function for all $s \in S$and $a \in A$, and $TC : S \to \mathbb{R}$ is a terminal cost function denoting the sign of an end of an MDP. An absorbing state with zero cost is usually used in a path planning problem . Whether or not a control policy $\pi : S \to A$ of one MDP is a good process is determined by its corresponding expected value function, which usually can be obtained by solving the Bellman equation given by:

$$V^{\pi}(s_t) = E_{\pi}(\sum_{i=0}^{\infty} \gamma^i r_{t+i} \,|s_t) = E_{\pi}(r_t + \gamma \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \,|s_t) = E_{\pi}(r_t + \gamma V^{\pi}(s_{t+1})) \quad (3.1)$$

where $\gamma \in [0,1)$ is the step size that corresponds to the iteration rate of the Bellman equation, and $s_{t+1} = \delta(s_t, a_t)$ is the transition function according to $P(\cdot \mid \cdot,\cdot) : S \times S \times A \to \mathbb{R}_{\geqslant 0}$. The goal is to find an optimal policy $\pi^*(s) = \arg\max_a V^{\pi}(s)$ that minimizes (or maximizes, depending on specific definition of the cost function R) the expected value (3.1) for every initial state S_0. In this section a sample-based method is applied to overcome the curse of dimensionality. Specifically, the finite

horizon discount version of MDP where $i < \infty$, is used for this purpose.

Next regime-switching is integrated into MDP to represent a dynamic environment. From experience it is known that a home environment can change between static and dynamic states. A regime ψ is defined as the time/step period between the last changes and the current changes. Therefore, the state, action and transition probability of MDP stay the same in one regime and vary from one regime to the next. Consider a countable collection Ψ of changing regimes of cost-minimizing MDP problems. Each regime $\psi_k \in \Psi$ is associated with one period of static MDP given by one RSMDP $M_{\psi_k} = (S_{\psi_k}, A_{\psi_k}, P_{\psi_k}, R_{\psi_k}, TC_{\psi_k})$, where $k \in N$ denotes the index of each discrete static time/step period of the changing environment. Consequently, the goal becomes finding the optimal policy $\pi_{\psi_k}{}^*(s) = \arg\max_a V^{\pi}{}_{\psi_k}(s)$ where $V^{\pi}{}_{\psi_k}(s)$is given by:

$$\begin{aligned} V^{\pi}{}_{\psi_k}(s_{\psi_k,t}) &= E_{\psi_k,\pi}(\sum_{i=0}^{\infty} {\gamma_{\psi_k}}^i r_{\psi_k,t+i} \,|\, s_{\psi_k,t}) \\ &= E_{\psi_k,\pi}(r_{\psi_k,t} + \gamma_{\psi_k} \sum_{i=0}^{\infty} {\gamma'}_{\psi_k}{}^i r_{\psi_k,t+i+1} \,|\, s_{\psi_k,t}) \\ &= E_{\psi_k,\pi}(r_{\psi_k,t} + \gamma_{\psi_k} V^{\pi}{}_{\psi_k}(s_{\psi_k,t+1})) \end{aligned} \tag{3.2}$$

It is seen that the optimal policy $\pi_{\psi_k} * (s)$varies from one regime to another. Hence in RSMDP, the problem of tracking the optimal policy$\pi_{\psi_k} * (s)$ corresponding to each regimeψ_k is considered. The only assumptions that is needed for $\pi_{\psi_k} * (s)$ is as follows:

Assumption 1: For each regime we have $\psi_k \in \Psi, E[T_\psi] \gg \sup E[t_i]$, where T_ψ represents the duration of regime $\pi_{\psi_k} * (s)$, and t_i represents the duration of each iteration in the Bellman equation.

Assumption 1 implies that the requirement of successfully converging to $\pi_{\psi_k} * (s)$ means that the regime does not change too often when compared with the time used for each iteration step in (3.2). This is satisfied in the practical scenario of path planning that is considered in the present work. Next, the way to express the path planning problem by incorporating PRM into RSMDP is described.

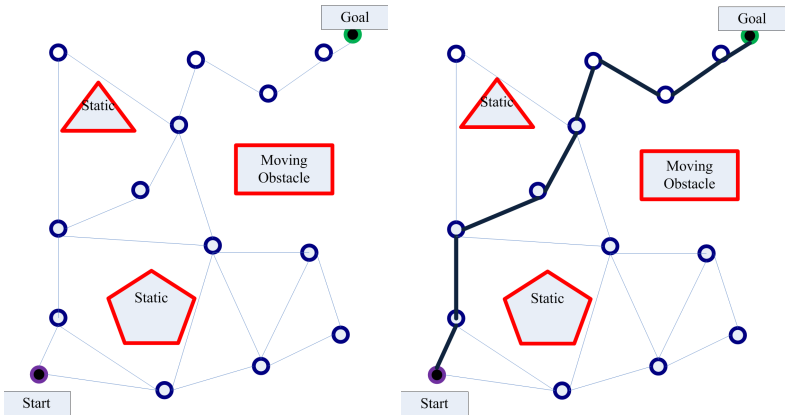
3.3 Probabilistic Roadmap for RSMDP

A home environment is arguably unstructured. For example, furniture may be cluttered and unorganized, and it is difficult to determine the structure of such furniture using sensors. Furthermore, this will impose a huge computational burden when building an accurate model to represent the environment. Sampling-based methods have adequately resolved the problem of computational burden, because these methods rely on a collision-checking module instead of using an explicit representation of the environment. PRM and its variants [6, 7], provide effective methods of path planning that are sampling-based.

PRM is a network of simple curve segments or arcs that meet at nodes. Each node corresponds to a configuration in the configuration space (C-space). Each arc between two nodes corresponds to a collision free path between two configurations. It comprises a preprocessing phase and a query phase. In the following, let C denote the robot's C-space, C_f the free C-space, N the node set, and E the edge set. First, initiate a graph $R = (N, E)$ that is empty. The preprocessing phase constructs the free C-space, giving the global picture, as shown in Fig. 3.1a. The query phase, shown in Fig. 3.1b, generates an optimal global collision-free path (bold line in Fig. 3.1b) by connecting the start and goal nodes to the roadmap, where heuristic methods are usually used (Q-learning is used in the present section). Details are found in [6, 7].

As defined, state set S_{ψ_k} in RSMDP corresponds to the node set N in PRM, and action set A_{ψ_k} corresponds to the edge set E. If the system is continuous, index t denotes the time interval; otherwise, index t denotes the step interval. In this section t is considered as the step interval without losing generality since the dynamic environment is formulated as a discrete RSMDP and each step is very short relative to the entire C-space. Therefore, the action subset $\bar{A}_{\psi_k} \in A_{\psi_k}$ associated with a state $s_{\psi_k,i} \in S_{\psi_k}$ at step i under regime ψ_k, is countable and corresponds to those edges connecting to the associated node in PRM. Hence which action should be chosen at a certain state in the learning process is governed by a stochastic behavior due to the unpredictable motion of the obstacles, although the available actions are countable. The final goal of the present work is to find the optimal policy ${\pi_{\psi_k}}^*(s)$ iteratively after the Q-learning process, as described in Section 3.4. Regime ψ_k

will not be changed unless the moving obstacles affect the current $\pi_{\psi_k}{}^*(s)$. Fig. 3.2 shows two common scenarios of regime change where one moving obstacle is detected using some distance threshold. As it blocks the current optimal path, the transition probability (although not known in advance) of the corresponding states and actions will be changed so that the current regime ψ_k will be changed into the next regime ψ_{k+1}. Then the Q-learning process will choose another available path according to the new regime. Sometimes, once the chosen path is blocked in the current regime, there may not exist an available path to choose from due to the lack of sampled nodes in PRM. For example in Fig. 3.2c, an obstacle moves to block all possible paths to the goal node and stays there for a long time. Here again, PRM is imported to build a local roadmap around the robot and the moving obstacle, in order to determine a feasible path. In particular, as shown in Fig. 3.2d,



(a) **(b)**

Figure 3.1: The PRM process in the C-space: (a)Preprocessing stage; (b) Query state. *Note*:Polygons represent static and moving obstacles, and the blank space represents the collision-free C-space. In order to represent a robot as a point as it moves on the ground, the standard practice is to expand the obstacles corresponding to the size reduction of the robot, as shown by the bold sideline of polygons. A uniformly random sample method is used to construct deterministic nodes in the free C-space. Then, such nodes are collision-free nodes. The roadmap is constructed using the collision-free nodes.

first a semicircular local region is built centered on the location of the previous state of the blocked edge. Its radius is calculated from the distance between the locations of two states connected by the same blocked edge. Then, a roadmap is generated within this semicircle by the same PRM method as before. Clearly the extra sampled nodes generated by the local roadmap will change the structure of the current PRM and consequently change the current regime ψ_k into the next regime ψ_{k+1}.

3.4 Path Planner with Online Q-learning

PRM works well in a static environment, but it cannot adapt to a dynamic environment where there are moving obstacles. Re-planning might be an intuitive alternative, in the presence of moving obstacles, but it would be impractical in general. For example, a moving obstacle might rapidly change its position after the path planner re-calculates a path based on the previous sensor information of the moving obstacle, and that new position of the moving obstacle might still block the new path. Considering such problems, the present section incorporates the reinforcement learning method, Q-learning, into the query phase of the PRM in the RSMDPformulation. In this manner, when the optimal path is blocked by a moving obstacle, the path planner is able to quickly choose another optimal path, using the previous experience about the map, as determined by the Q function value.

In the 1990s reinforcement learning (RL) was proposed to solve MDP problems [5]. In RL, an agent learns its behavior through trial-and-error interactions with a dynamic environment, while receiving rewards for good actions and penalties for bad actions. Specifically, the agent performs an action a_t in state s_t and receives a real-valued reward $r_t = r(s_t, a_t) \in R$ from the environment. Through this process, the agent learns an optimal policy $\pi^*(s) = \arg\max_a V^\pi(s)$ where $V^\pi(s)$ is equal to 1, which maps the state set S into the action set A, and arrives at its next state $s_{t+1} = \delta(s_t, a_t)$. The policy should be able to maximize the cumulative reward according to $V^\pi(s)$.

Q-learning is a popular version of off-policy reinforcement learning which, regardless of the policy being followed, always estimates the optimal Q-function that is defined as $Q(s_t, a_t) : S \times A \rightarrow \mathbb{R}$. Q-learning has two main advantages when

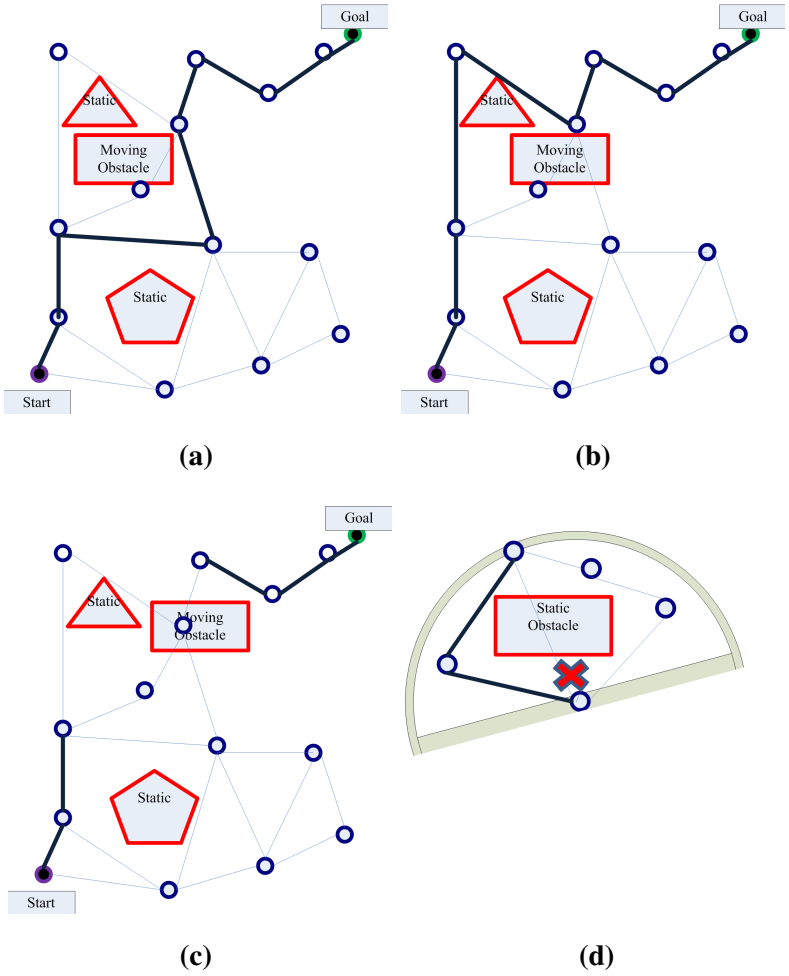


Figure 3.2: Local roadmap generation: (a) an original path generated by PRM; (b) an alternative path selected if the original path is blocked; (c)(d) if no alternative path is available, a semicircle is used to define the scope where some new points will be generated to attach to the PRM in order to find an alternative collision-free path .

compared with other approaches of reinforcement learning. First, it does not require a model of the environment, which is advantageous when dealing with an unknown environment. For example, in an unknown environment $r_t = r(s_t, a_t)$ and $s_{t+1} = \delta(s_t, a_t)$ are nondeterministic functions. Then, r and s are initiated arbi-

trarily and the algorithm will eventually converge to the optimal $Q^*(s,a)$ value in view of its mathematical basis. Second, it is able to update the estimates using partially learned estimates without waiting for completing the entire episode, which means it bootstraps. These aspects are discussed under MDP formulation. The core algorithm of Q-learning in RSMDP is given by:

$$Q_{\psi_k}(s_t,a_t) \leftarrow Q_{\psi_k}(s_t,a_t) + \gamma_{\psi_k,t}[r_{\psi_k,t}(s_t,a_t) + \alpha_{\psi_k}\max_a Q_{\psi_k}(s_{t+1},a_{t+1}) - Q_{\psi_k}(s_t,a_t)] \tag{3.3}$$

The optimal action policy $\pi^*_{\psi_k}$ is given by:

$$\pi^*_{\psi_k}(s) = \arg\max_a Q^*_{\psi_k}(s,a) \tag{3.4}$$

There are two conditions that should be satisfied to guarantee the convergence of Q-learning to optimal $Q^*_{\psi_k}(s,a)$ with probability one [11]. These are given now.

Condition 1: All the state-action pairs $Q(s_t,a_t)$ are visited infinitely often as the number of transitions approaches infinity.

Condition 2: The step size γ_t should satisfy $\gamma_{\psi_k,t} > 0, \forall k, \quad \sum_{t=0}^{\infty} \gamma_{\psi_k,t} = \infty,$ $\sum_{t=0}^{\infty} \gamma^2_{\psi_k,t} < \infty$.

Condition 1 is called exploration, which requires that Q-learning has nonzero probability of choosing any action when it also needs to exploit its current knowledge in order to perform well by selecting greedy actions in the current Q-function. A popular method to balance exploration with exploitation is the $\varepsilon - greedy$ approach:

$$a_t \leftarrow \begin{cases} an\ uniform\ random\ action\ in\ A_{\psi_k}, with\ probability\ \varepsilon_k \\ a \in \arg\max_a Q_{\psi_k}(s_t,a_t), with\ probability\ 1-\varepsilon_k \end{cases} \tag{3.5}$$

Condition 2 implies that the step size should meet the requirement of $\lim_{t\to 0} \gamma_{\psi_k,t} = 0$. There is also a tradeoff problem when choosing $\gamma_{\psi_k,t}$ in regime ψ_k. In order for Q-learning to converge to optimal $Q^*_{\psi_k}(s,a)$ quickly, the step size $\gamma_{\psi_k,t}$ has to be large; however, the step size $\gamma_{\psi_k,t}$ should be small in order to minimize the magnitude of the fluctuations of the Q-function within a given regime. In traditional Q-learning, this tradeoff does not considerably affect the system since the speed is

usually adequate to solve related problems in a static situation. However, in obstacle avoidance in the RSMDP framework, the way how the Q-function converges to the optimal value greatly affects the robot system. A large $\gamma_{\psi_k,t}$ is expected to produce a fast convergence speed, but the high-magnitude fluctuations caused by small $\gamma_{\psi_k,t}$ will lead to an incorrect optimal Q-function, possibly causing the robot to collide with obstacles. At high speeds, safety should be given more attention. The strategy to achieve these performance requirements is to make the Q-function iteration process robust to $\gamma_{\psi_k,t}$. Then, accurate optimal value can be achieved at a satisfactory speed. Therefore, in the current work, by setting and resetting $\gamma_{\psi_k,t} = \gamma'_{\max}$, the path planner always selects the largest possible step size for the current regime and makes it converge to zero within the same regime. But the step size is reset to the largest possible value again when the regime changes. In this way, $\gamma_{\psi_k,t}$ is able to eventually converge to zero. But it is set to a large value in the beginning of the iteration so that the Q-function iteration is robust to the changes in $\gamma_{\psi_k,t}$, while having sufficient speed of converging to the new optimal Q-function $Q^*_{\psi_k}(s,a)$ to adapt to the new regime ψ_{k+1}. When to reset $\gamma_{\psi_k,t}$ is critical in the present approach. In view of Assumption 1 in Section 3.4, the changing frequency of the dynamic environment should not be very high although the moving obstacle may always make the environment to change. To this end, it is assumed that the regime is changed only when the current path has been blocked by obstacles that enter the robot's dangerous area as defined by some threshold, rather than when moving obstacles change the PRM.

The online path planner with Q-learning, which is used in the present section, chooses the optimal path according to the maximum Q value with respect to each state-action pair. The Q value of each state-action pair is obtained by taking into account both the shortest path and obstacle avoidance in the cost function $r_{\psi_k,t}(s_t,a_t)$ defined by:

$$r_{\psi_k,t}(s_t,a_t) = \omega f(s_t,s_{t+1}) + (1-\omega)h(s_t) \tag{3.6}$$

Here $f(s_t,s_{t+1}) = \|s_t,s_{t+1}\|_\delta$ denotes a distance function defined by the δ norm, ω is the weighting parameter used to balance $f(s_t,s_{t+1})$, and $h(s_t)$ denotes the reward function according to moving obstacles.

Once an optimal path is obtained in the current regime, the robot begins to move. When a moving obstacle blocks the recurrent optimal path, the step size resetting will be made and the path planner will chooses another available optimal path by quickly converging to the new optimal policy. It is seen that although the step size $\gamma_{\psi_k,t}$ changes every time when the regime changes, the learning rate within the new regime will be faster than in the previous regimes since Q-value for each state-action pair is saved as the knowledge for the new regime. That is the reason why the present path planner is able to adapt to a dynamic environment.

3.5 Simulation Studies

This section presents simulation studies to validate the online path planner developed for mobile robot. The system is simulated using open-source software OpenCV[1].

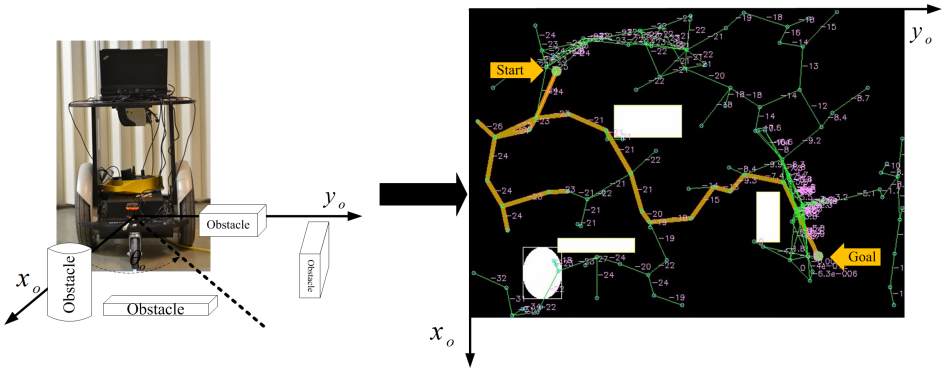


Figure 3.3: Simulated environment for the mobile robot in a 640×480 image plane, in a learning process.

Fig. 3.3 shows an image of a simulated environment obtained from a presumed global camera before the mobile robot is started. The white rectangles represent static obstacles and the white ellipse represents a moving obstacle. The world state is made up of the pixel coordinates of the nodes of the roadmap within a 640×480 image plane, which is represented as $s(x,y)$ where $x = 0,1,\ldots,640$; $y = 0,1,2\ldots,480$. After obtaining the collision-free states, the starting point and

[1] http://opencv.org/

the goal point of the robot are set at fixed positions indicated by arrows. These two points are added to the collision-free roadmap in the same way that the roadmap is generated, except that loop connection is used instead of incremental connection in order to provide more possibilities of path connection between the starting point and the goal point. The thin lines represent available paths and the thicker lines represent optimal paths obtained during the learning process. The rotation coordination θ_o can be ignored here to illustrate path planning for the purpose of simplicity of simulation, because the mobile robot can first rotate to the correct direction before it starts to move. The number close to each edge is the Q value for each state-action pair corresponding to the edge. Euclidian distance is used by setting $\delta = 2, \omega = 10/(640+480)$ in Equation (3.6) and setting the reward function as follows:

$$h(s_t) = \begin{cases} R= \text{ 0; when the robot reaches the gaol} \\ R= \text{ -10; when the robot touches the obstacle} \\ R= \text{ -5; in any other situation.} \end{cases} \tag{3.7}$$

The first simulation (shown in Fig. 3.4) tests how the Q value converges to its

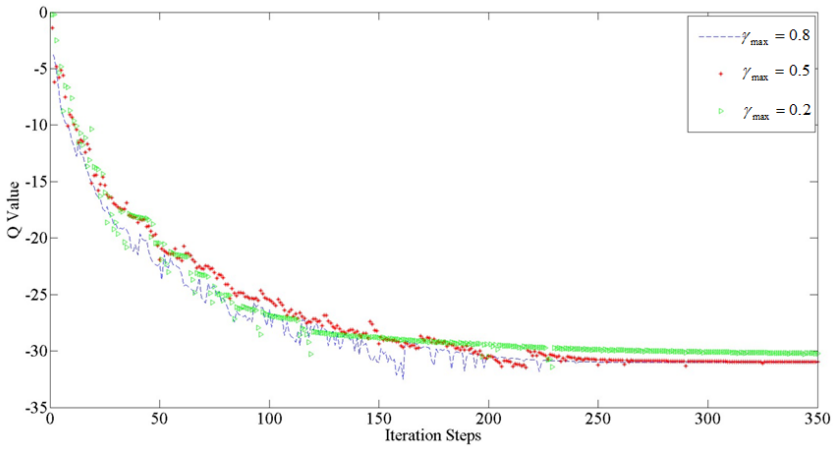


Figure 3.4: History of Q-value with $\gamma_{\max} = 0.8, 0.5, 0.2$ and 100 sampled points generated in PRM.

optimal value in terms of the possible maximum beginning value of the step size

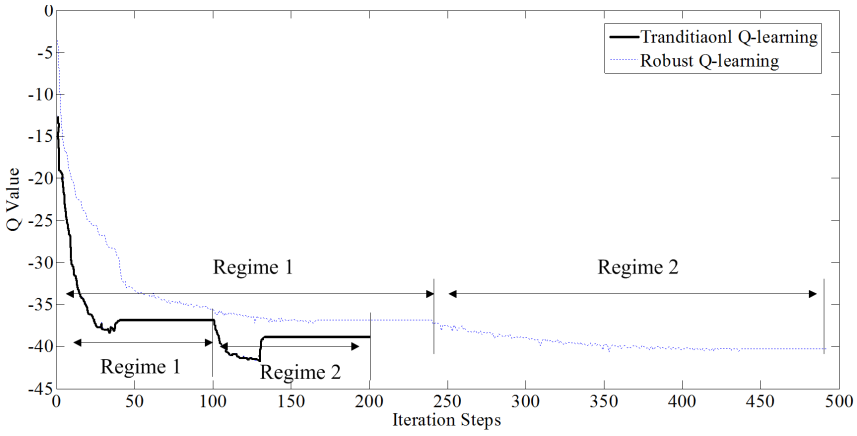


Figure 3.5: Performance comparison between traditional Q-learning and robust Q-learning with $\gamma_{\max} = 0.98$.

when using the strategy $\gamma_{\psi_k,t} = \gamma^t_{\max}$. The Q value of the edge close to the goal point is chosen under three beginning step-size values, $\gamma_{\max} = 0.8, 0.5, 0.2$(shown by the dotted line, star line and triangle line, respectively, in the figure). It is seen that the smallest value $\gamma_{\max} = 0.2$ can bring up less intense fluctuations, which is in accordance with Condition 2 in Section 3.4. Nevertheless, it is seen that the optimal Q-value and the learning rate do not change appreciably for different maximum beginning values of the step size $\gamma_{\max}$. In this sense, the Q-value iteration process is robust for choosing $\gamma_{\max}$. Specifically, $\gamma_{\max}$can be chosen as high as possible in order to obtain the fastest possible learning rate or the fastest convergence speed. Fig. 3.5 shows the performance of the iteration process when $\gamma_{\max}$ is set at 0.98 for robust Q-learning and traditional Q-learning. In Fig. 3.5, it is seen that both robust Q-learning (dotted line) and traditional Q-learning (solid line) are able to successfully converge to a new optimal Q-value when the regime changes from regime 1 to regime 2, caused by moving obstacles. Although traditional Q-learning is faster than robust Q-learning, robust Q-learning converges to the optimal Q-value more smoothly when compared with the traditional Q-learning. In addition, higher rewords (in Q-value) are obtained by robust Q-learning. These properties will help the robot to choose a more accurate optimal action if the optimal Q-values of the actions with respect to one state intersect with each other when the regime changes

quickly and there is not enough time for the Q-value iteration to converge to the optimal value. Therefore, the safety can be guaranteed as much as possible at the cost of low speed of convergence. This low speed is fast enough in such conditions and can be achieved using a modern computer.

The second simulation (shown in Fig. 3.6) verifies that the present algorithm is able to successfully avoid both static and moving obstacles under the RSMDP and robust Q-learning framework. Keyboard controller is used to control the moving obstacle and make it move to block the obtained optimal path. With the cost function as defined in Equation (3.6) and Equation (3.7), the simulated robot reaches the goal point by choosing the shortest available path and avoiding obstacles, which is considered as regime 1. When the robot detects that the moving obstacle is blocking its current optimal path, it quickly finds another optimal path by using the learning experience, which is considered as regime 2. It is noted that although increasing the number of PRM nodes will generate more available paths, the time spent to learn a new optimal path will also increase. Hence, there is a tradeoff between the number of nodes and the time taken to avoid obstacles. In the present case, having 400 sampled nodes can provide the fastest speed to adapt to a dynamic environment.

The online robust Q-learning method that was investigated in the present section is a behavior-based decision-making process. In this method, the robot continuously observes the world states and selects the action having the optimal Q value among the possible actions in the current state, as given by the Q function. This is different from a traditional behavior-based system where the rule base of behavior is designed entirely by a human expert in advance. The rule base of Q-learning is learned autonomously when the robot interacts with its environment during the training process. The curse of dimensionality is a serious challenge in this process because, in theory, an infinite number of iterations would be needed to guarantee convergence to the optimal value. The method proposed in the present work overcomes this problem by incorporating a PRM roadmap as the world state for the robot. The safety is another challenge, when dealing with rapidly moving obstacles in a dynamic environment. The robust Q-learning in the present work guarantees smooth convergence for Q-value iteration so that a relatively accurate action is chosen when the regime changes.

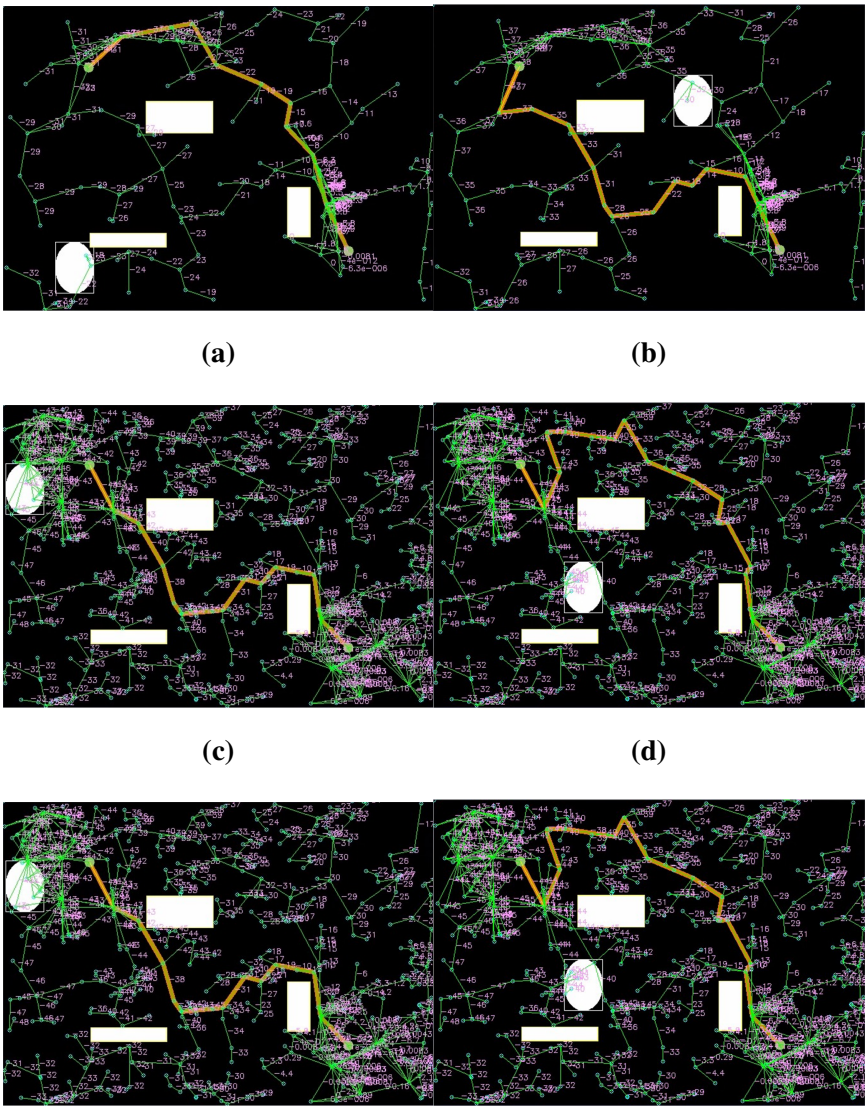


Figure 3.6: Obstacle avoidance in dynamic environment. (a), (c), (e) original optimal path in regime 1; (b), (d), (f) scenarios for regime 2 where moving obstacle is blocking the current path, hence choose another optimal path. (a) and (b) correspond to 100 sampled points in PRM, (c) and (d) correspond to 400 sampled points, and (e) and (f) correspond to 600 sampled points.

Chapter 4

Extended Linear Quadratic Regulator Enhanced with PWA for Motion Planning

4.1 Introduction

This chapter begins by reviewing linear-quadratic-regulator (LQR) with regard to cost-to-go function, and how to obtain the piecewise affine closed-form control law, piecewise affine-LQR (PWA-LQR), for constrained LQR. It is done by incorporating quadratic programming (QP) in the traditional LQR. PWA-LQR also brings in the concept of cost-to-come function which uses a similar procedure as the cost-to-go function, except that cost-to-come runs forward in time. PWA-LQR smoothing combines these two functions to form a smoother that provides a sequence of optimal states for the linear-quadratic control problem. Then PWA-LQR is extended (E) to PWA-ELQR which is used to deal with nonlinear-nonquadratic cases where an iteratively local approximation method is used.

4.2 Optimal Motion Planning with Constraints

In the present section, the objective of the optimal motion planning is to find an optimal control policy $\pi_t \in \mathbb{X} \rightarrow \mathbb{U}$ such that for the entire horizon $0 \leqslant t \leqslant l$ the

selected input control $\mathbf{u}_t = \pi_t(\mathbf{x}_t)$ can (approximately) minimize a cost function defined by:

$$c_l(\mathbf{x}_l) + \sum_{t=0}^{l-1} c_t(\mathbf{x}_t, \mathbf{u}_t), \tag{4.1}$$

where l denotes the horizon, $c_l \in \mathbb{X} \to \mathbb{R}$ denotes the final-step cost function and $c_t(\mathbf{x}_t, \mathbf{u}_t) \in \mathbb{X} \times \mathbb{U} \to \mathbb{R}$ represents the immediate cost function at time-step t.

Based on the concept of cost function given by Equation(4.1), the cost-to-go function $v_t \in \mathbb{X} \to \mathbb{R}$ and the related optimal control policy π_t at the current state $\mathbf{x}_t$ can be calculated by the backward recursion procedure:

$$\begin{aligned}
& v_l(\mathbf{x}_l) = c_l(\mathbf{x}_l) \\
& v_t(\mathbf{x}_t) = \min_{\mathbf{u}_t}(c_t(\mathbf{x}_t, \mathbf{u}_t) + v_{t+1}(\mathbf{g}_t(\mathbf{x}_t, \mathbf{u}_t))), \\
& \pi_t(\mathbf{x}_t) = \operatorname{argmin}_{\mathbf{u}_t}(c_t(\mathbf{x}_t, \mathbf{u}_t) + v_{t+1}(\mathbf{g}_t(\mathbf{x}_t, \mathbf{u}_t))), \\
\text{s.t. } & \mathbf{x_{min}} \leqslant \mathbf{x}_t \leqslant \mathbf{x_{max}}, \\
& \mathbf{u_{min}} \leqslant \mathbf{u}_t \leqslant \mathbf{u_{max}}, \\
& \mathbf{x}_{t+1} = \mathbf{g}_t(\mathbf{x}_t, \mathbf{u}_t),
\end{aligned} \tag{4.2}$$

According to the foregoing definition, the kinematic system (1.1) is nonlinear and the cost function (4.1) generally is nonquadratic. Hence linear and quadratic approximation methods are needed for handling the problem. Basically, by deriving Riccati equations when solving a quadratic optimization problem at time step t, a formula for minimizing the cost-to-go function can be obtained at time step $t-1$. However, optimizing a quadratic problem is solvable only when the Hessian is positive-(semi)definite. This rule must be followed to assure that all of the immediate cost functions have positive-(semi)definite Hessians when designing cost functions. Specifically,

$$\begin{aligned}
\frac{\partial^2 c_l}{\partial \mathbf{x}_l \partial \mathbf{x}_l} &\geqslant 0, \\
\frac{\partial^2 c_t}{\partial \mathbf{u}_t \partial \mathbf{u}_t} &> 0, \\
\begin{bmatrix} \frac{\partial^2 c_t}{\partial \mathbf{x}_t \partial \mathbf{x}_t} & \frac{\partial^2 c_t}{\partial \mathbf{x}_t \partial \mathbf{u}_t} \\ \frac{\partial^2 c_t}{\partial \mathbf{u}_t \partial \mathbf{x}_t} & \frac{\partial^2 c_t}{\partial \mathbf{u}_t \partial \mathbf{u}_t} \end{bmatrix} &\geqslant 0.
\end{aligned} \tag{4.3}$$

4.3 Piecewise Affine-ELQR

4.3.1 Traditional Linear-Quadratic-Regulator

LQR is a prevalent controller that provides a closed-form optimal solution for the linear-quadratic control problem[11]. The linear kinematics for all horizon $0 \leqslant t \leqslant l$ are defined as:

$$\mathbf{x}_{t+1} = \mathbf{g}_t(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t + \mathbf{c}_t, \tag{4.4}$$

where $\mathbf{A}_t \in \mathbb{R}^{n\times n}$, $\mathbf{B}_t \in \mathbb{R}^{n\times m}$, $\mathbf{c}_t \in \mathbb{R}^n$, and both q_l and p_t are constant. The quadratic immediate cost functions for all horizons $0 \leqslant t \leqslant l$ are defined as:

$$\begin{aligned} c_l(\mathbf{x_l}) &= \frac{1}{2}\mathbf{x}_l^T\mathbf{Q}_l\mathbf{x}_l + \mathbf{x}_l^T\mathbf{q}_l + q_l, \\ c_t(\mathbf{x}_t, \mathbf{u}_t) &= \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T\begin{bmatrix}\mathbf{Q}_t & \mathbf{P}_t^T\\ \mathbf{P}_t & \mathbf{R}_t\end{bmatrix}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T\begin{bmatrix}\mathbf{q}_t\\ \mathbf{r}_t\end{bmatrix} + p_t, \end{aligned} \tag{4.5}$$

where$\mathbf{Q}_t \in \mathbb{R}^{n\times n}$,$\mathbf{R}_t \in \mathbb{R}^{m\times m}$,$\mathbf{P}_t \in \mathbb{R}^{m\times n}$,$\mathbf{q}_t \in \mathbb{R}^n$, and $\mathbf{r}_t \in \mathbb{R}^m$, such that $\mathbf{Q}_l > 0$ and $\mathbf{R}_t > 0$ are positive-definite, and $\begin{bmatrix}\mathbf{Q}_t & \mathbf{P}_t^T\\ \mathbf{P}_t & \mathbf{R}_t\end{bmatrix} \geqslant 0$ is positive-semidefinite, in accordance with Equation (4.3).

There are two attractive features in the linear-quadratic control problem. The first one is that the corresponding cost-to-go functions have explicit quadratic formulation as follows:

$$\begin{aligned} v_l(\mathbf{x_l}) &= \frac{1}{2}\mathbf{x}_l^T\mathbf{S}_l\mathbf{x}_l + \mathbf{x}_l^T\mathbf{s}_l + s_l, \\ v_t(\mathbf{x_t}) &= \frac{1}{2}\mathbf{x}_t^T\mathbf{S}_t\mathbf{x}_t + \mathbf{x}_t^T\mathbf{s}_t + s_t, \end{aligned} \tag{4.6}$$

where s_l,s_t are constant, $\mathbf{S}_t \in \mathbb{R}^{n\times n} > 0$, and $\mathbf{s}_t \in \mathbb{R}^n$. For final-step l, $\mathbf{S}_l = \mathbf{Q}_l$, $\mathbf{s}_l = \mathbf{q}_l$ and $s_l = q_l$. By combining Equations (4.2), (4.4), and (4.5), we can obtain:

$$v_t(\mathbf{x}_t) = \min_{\mathbf{u}_t}\left(\frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T\begin{bmatrix}\mathbf{D}_t & \mathbf{C}_t^T\\ \mathbf{C}_t & \mathbf{E}_t\end{bmatrix}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T\begin{bmatrix}\mathbf{d}_t\\ \mathbf{e}_t\end{bmatrix} + p_t\right), \tag{4.7}$$

where:

$$\begin{aligned}
\mathbf{C}_t &= \mathbf{P}_t + \mathbf{B}_t^T \mathbf{S}_{t+1} \mathbf{A}_t, \\
\mathbf{D}_t &= \mathbf{Q}_t + \mathbf{A}_t^T \mathbf{S}_{t+1} \mathbf{A}_t, \\
\mathbf{E}_t &= \mathbf{R}_t + \mathbf{B}_t^T \mathbf{S}_{t+1} \mathbf{B}_t, \\
\mathbf{d}_t &= \mathbf{q}_t + \mathbf{A}_t^T \mathbf{s}_{t+1} + \mathbf{A}_t^T \mathbf{S}_{t+1} \mathbf{c}_t, \\
\mathbf{e}_t &= \mathbf{r}_t + \mathbf{B}_t^T \mathbf{s}_{t+1} + \mathbf{B}_t^T \mathbf{S}_{t+1} \mathbf{c}_t.
\end{aligned} \tag{4.8}$$

and accordingly,

$$\mathbf{S}_t = \mathbf{D}_t - \mathbf{C}_t^T \mathbf{E}_t^{-1} \mathbf{C}_t, \tag{4.9}$$

$$\mathbf{s}_t = \mathbf{d}_t - \mathbf{C}_t^T \mathbf{E}_t^{-1} \mathbf{e}_t. \tag{4.10}$$

The second feature is that linear-quadratic control problems have a closed-form optimal controller which is an explicit linear formulation in the feedback form:

$$\mathbf{u}_t = \pi_t(\mathbf{x}_t) = \mathbf{L}_t \mathbf{x}_t + \mathbf{l}_t, \tag{4.11}$$

$$\mathbf{L}_t = -\mathbf{E}_t^{-1} \mathbf{C}_t, \mathbf{l}_t = -\mathbf{E}_t^{-1} \mathbf{e}_t. \tag{4.12}$$

4.3.2 Piecewise Affine Feedback Control for Constrained Control Problems

The closed-form feedback control law (4.11) comes from the minimization of the quadratic cost-to-go function (4.6) with the assumption that none of the variables have constraints. This can be considered as solving an unconstrained optimization problem. Even when there are natural constraints on variables, it may be reasonable to disregard them as their effects on the solution are negligible. However, in many practical problems, the constraints play a critical role (like in the example of the present work). Therefore, the unconstrained problem has to be transformed into a

constrained one as follows, in accordance with Equation (4.6):

$$\begin{aligned} v_l(\mathbf{x_l}) &= \frac{1}{2}\mathbf{x}_l^T\mathbf{S}_l\mathbf{x}_l + \mathbf{x}_l^T\mathbf{s}_l + s_l, \\ v_t(\mathbf{x_t}) &= \frac{1}{2}\mathbf{x}_t^T\mathbf{S}_t\mathbf{x}_t + \mathbf{x}_t^T\mathbf{s}_t + s_t, \\ \text{s.t.}&\mathbf{x_{min}} \leqslant \mathbf{x}_t \leqslant \mathbf{x_{max}}, \\ &\mathbf{u_{min}} \leqslant \mathbf{u}_t \leqslant \mathbf{u_{max}}, \\ &\mathbf{x}_{t+1} = \mathbf{g}_t(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t + \mathbf{c}_t, \end{aligned} \tag{4.13}$$

Equation (4.13) is a typical constrained optimization problem which can be solved by QP [67].

An optimization problem with a quadratic objective function and linear constraints is called a quadratic program. Clearly, Equation (4.13) belongs to inequality-constrained problems and satisfies first-order optimality conditions. Therefore, it can be solved by applying Karush-Kuhn-Tucker (KKT) conditions, also known as first-order necessary conditions as follows:

Step 1, Lagrangian function for Equation (4.13):

$$\begin{aligned} \pounds(\mathbf{x}_t, \lambda_t) = v_t(\mathbf{x}_t) &+ \lambda_{(1)t}(\mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t + \mathbf{c}_t) \\ &+ \lambda_{(2)t}(\mathbf{x}_t - \mathbf{x_{min}}) \\ &+ \lambda_{(3)t}(\mathbf{x_{max}} - \mathbf{x}_t) \\ &+ \lambda_{(4)t}(\mathbf{u}_t - \mathbf{u_{min}}) \\ &+ \lambda_{(5)t}(\mathbf{u_{max}} - \mathbf{u}_t), \end{aligned} \tag{4.14}$$

Step 2, specifying KKT conditions for the optimal solution $\mathbf{x}^*$:

$$\begin{aligned}
\nabla_x \pounds(\mathbf{x}_t^*, \lambda_t^*) &= 0, \\
\mathbf{x}_t^* - \mathbf{x_{min}} &\geq 0, \\
\mathbf{x_{max}} - \mathbf{x}_t^* &\geq 0, \\
\mathbf{u}_t - \mathbf{u_{min}} &\geq 0, \\
\mathbf{u_{max}} - \mathbf{u}_t &\geq 0, \\
\lambda_{(1)t} &\geq 0, \\
\lambda_{(2)t} &\geq 0, \\
\lambda_{(3)t} &\geq 0, \\
\lambda_{(4)t} &\geq 0, \\
\lambda_{(5)t} &\geq 0, \\
\lambda_{(1)t}(\mathbf{A}_t\mathbf{x}_t^* + \mathbf{B}_t\mathbf{u}_t + \mathbf{c}_t) &= 0, \\
\lambda_{(2)t}(\mathbf{x}_t^* - \mathbf{x_{min}}) &= 0, \\
\lambda_{(3)t}(\mathbf{x_{max}} - \mathbf{x}_t^*) &= 0, \\
\lambda_{(4)t}(\mathbf{u}_t - \mathbf{u_{min}}) &= 0, \\
\lambda_{(5)t}(\mathbf{u_{max}} - \mathbf{u}_t) &= 0.
\end{aligned} \tag{4.15}$$

Based on (4.14) and (4.15), the piecewise affine feedback control law for (4.13) can be obtained as follows:

$$\mathbf{u}_t = \pi_t(\mathbf{x}_t) = \mathbf{L}_t^i\mathbf{x}_t + \mathbf{l}_t^i, \mathbf{x}_t \in CR_i \tag{4.16}$$

where CR_i is the critical region to which x_t belongs, according to the constraints on the state and (or) the control input, and,

$$\mathbf{S}_t^i = \mathbf{D}_t - \mathbf{C}_t^T\mathbf{L}_t^i, \tag{4.17}$$

$$\mathbf{s}_t^i = \mathbf{d}_t - \mathbf{C}_t^T\mathbf{l}_t^i, \tag{4.18}$$

where $\mathbf{C}_t$, $\mathbf{D}_t$, $\mathbf{E}_t$, $\mathbf{d}_t$ and $\mathbf{e}_t$ are similar as (4.8).

There are a handful of effective methods to calculate CR_i.They include active-

set methods which are suitable for small- and medium-sized problems[68], and interior-point methods for large problems[69][70][71]. It is indicated in [69] that, in convex quadratic programming, interior-point methods are generally much faster on large problems than active-set methods.

QPs can always be solved (or shown to be infeasible) in a finite amount of computation. But the effort required to find a solution depends strongly on the characteristics of the objective function and the number of inequality constraints. If the Hessian matrix $\mathbf{G}$ is positive semi-definite, Equation (4.13) is said to be a convex QP, and in this case the problem is often similar in difficulty to a linear program (strictly convex QPs are those in which $\mathbf{G}$ is positive definite). Nonconvex QPs, in which $\mathbf{G}$ is an indefinite matrix, can be more challenging because they can have several stationary points and local minima."

4.3.3 PWA-LQR Smoothing

The cost-to-go functions (4.2) is called backward LQR or PWA-LQR if there are constraints. It means only the total future cost that will occur between stage t and stage l (inclusive of stage t and stage l) is considered. In this subsection the similar concepts, cost-to-come functions and forward LQR, are introduced. Then the forward LQR is extended to forward PWA-LQR. Finally the PWA-LQR smoother is obtained by combining backward PWA-LQR and forward PWA-LQR.

Cost-to-come functions and Forward PWA-LQR

Cost-to-come functions, denoted by $\bar{v}_t(\mathbf{x}_t)$, give the total past cost that occurred between stage 0 and stage t (excluded). Given the inverse kinematics (1.4), $\bar{v}_t(\mathbf{x}_t)$ and inverse control policy $\bar{\pi}_t$ are defined by the following forward value iteration[17]:

$$\begin{aligned}
\bar{v}_0(\mathbf{x}_0) &= 0 \\
\bar{v}_{t+1}(\mathbf{x}_{t+1}) &= \min_{\mathbf{u}_t}(c_t(\bar{\mathbf{g}}_t(\mathbf{x}_{t+1},\mathbf{u}_t)),\mathbf{u}_t) + \bar{v}_t(\bar{\mathbf{g}}_t(\mathbf{x}_{t+1},\mathbf{u}_t))), \\
\pi_t(\mathbf{x}_{t+1}) &= \operatorname{argmin}_{\mathbf{u}_t}(c_t(\bar{\mathbf{g}}_t(\mathbf{x}_t,\mathbf{u}_t)) + \bar{v}_t(\bar{\mathbf{g}}_t(\mathbf{x}_{t+1},\mathbf{u}_t))), \\
\text{s.t. } & \mathbf{x_{min}} \leqslant \mathbf{x}_t \leqslant \mathbf{x_{max}}, \\
& \mathbf{u_{min}} \leqslant \mathbf{u}_t \leqslant \mathbf{u_{max}}, \\
& \mathbf{x}_t = \bar{\mathbf{g}}_t(\mathbf{x}_{t+1},\mathbf{u}_t),
\end{aligned} \tag{4.19}$$

The control input $\mathbf{u}_t = \bar{\pi}_t(\mathbf{x}_{t+1})$ drives state $\mathbf{x}_t$ to state $\mathbf{x}_{t+1}$ with minimal cost-to-come cost. Assume that the inverse kinematics are linear, defined by:

$$\mathbf{x}_t = \bar{\mathbf{g}}_t(\mathbf{x}_{t+1}, \mathbf{u}_t) = \bar{\mathbf{A}}_t \mathbf{x}_{t+1} + \bar{\mathbf{B}}_t \mathbf{u}_t + \bar{\mathbf{c}}_t, \tag{4.20}$$

where $\bar{\mathbf{A}}_t = \mathbf{A}_t^{-1}$, $\bar{\mathbf{B}}_t = -\mathbf{A}_t^{-1}\mathbf{B}_t$ and $\bar{\mathbf{c}}_t = -\mathbf{A}_t^{-1}\mathbf{c}_t$. Combining (4.19), (4.20) the same local cost functions as (4.5), an explicit quadratic formulation of global cost-to-come functions can be obtained:

$$\begin{aligned} \bar{v}_0(\mathbf{x}_0) &= 0 \\ \bar{v}_t(\mathbf{x_t}) &= \frac{1}{2}\mathbf{x}_t^T \bar{\mathbf{S}}_t \mathbf{x}_t + \mathbf{x}_t^T \bar{\mathbf{s}}_t + \bar{s}_t, \end{aligned} \tag{4.21}$$

Similar to (4.11) and (4.12), the feedback control law for forward LQR is:

$$\mathbf{u}_t = \bar{\pi}_t(\mathbf{x}_{t+1}) = \bar{\mathbf{L}}_t \mathbf{x}_{t+1} + \bar{\mathbf{l}}_t \tag{4.22}$$

$$\bar{\mathbf{L}}_t = -\bar{\mathbf{E}}_t^{-1}\bar{\mathbf{C}}_t, \bar{\mathbf{l}}_t = \bar{\mathbf{E}}_t^{-1}\bar{\mathbf{e}}_t. \tag{4.23}$$

where

$$\bar{\mathbf{S}}_{t+1} = \bar{\mathbf{D}}_t - \bar{\mathbf{C}}_t^T \bar{\mathbf{L}}_t, \tag{4.24}$$

$$\bar{\mathbf{s}}_{t+1} = \bar{\mathbf{d}}_t - \bar{\mathbf{C}}_t^T \bar{\mathbf{l}}_t, \tag{4.25}$$

$$\begin{aligned} \bar{\mathbf{C}}_t &= \bar{\mathbf{B}}_t^T(\bar{\mathbf{S}}_t + \mathbf{Q}_t)\bar{\mathbf{A}}_t + \mathbf{P}_t\bar{\mathbf{A}}_t, \\ \bar{\mathbf{D}}_t &= \bar{\mathbf{A}}_t^T(\bar{\mathbf{S}}_t + \mathbf{Q}_t)\bar{\mathbf{A}}_t, \\ \bar{\mathbf{E}}_t &= \bar{\mathbf{B}}_t^T(\bar{\mathbf{S}}_t + \mathbf{Q}_t)\bar{\mathbf{B}}_t + \mathbf{R}_t + \mathbf{P}_t\bar{\mathbf{B}}_t + \bar{\mathbf{B}}_t^T\mathbf{P}_t^T, \\ \bar{\mathbf{d}}_t &= \bar{\mathbf{A}}_t^T(\bar{\mathbf{s}}_t + \mathbf{q}_t) + \bar{\mathbf{A}}_t^T(\bar{\mathbf{S}}_t + \mathbf{Q}_t)\bar{\mathbf{c}}_t, \\ \bar{\mathbf{e}}_t &= \mathbf{r}_t + \mathbf{P}_t\bar{\mathbf{c}}_t + \bar{\mathbf{B}}_t^T(\bar{\mathbf{s}}_t + \mathbf{q}_t) + \bar{\mathbf{B}}_t^T(\bar{\mathbf{S}}_t + \mathbf{Q}_t)\bar{\mathbf{c}}_t \end{aligned} \tag{4.26}$$

Suppose that there are constraints on the state and control input in the cost-to-

go functions, given by:

$$\begin{aligned}
&\bar{v}_0(\mathbf{x}_0) = 0 \\
&\bar{v}_t(\mathbf{x_t}) = \frac{1}{2}\mathbf{x}_t^T \bar{\mathbf{S}}_t^i \mathbf{x}_t + \mathbf{x}_t^T \bar{\mathbf{s}}_t^i + \bar{s}_t, \;\; \mathbf{x}_t \in CR_i \\
\text{s.t.} \;\; &\mathbf{x_{min}} \leqslant \mathbf{x}_t \leqslant \mathbf{x_{max}}, \\
&\mathbf{u_{min}} \leqslant \mathbf{u}_t \leqslant \mathbf{u_{max}}, \\
&\mathbf{x}_t = \bar{\mathbf{g}}_t(\mathbf{x}_{t+1}, \mathbf{u}_t),
\end{aligned} \tag{4.27}$$

The similar KKT conditions (4.14)(4.15) should be added, and the corresponded piecewise affine feedback control law can be obtained as:

$$\mathbf{u}_t = \bar{\pi}_t(\mathbf{x}_{t+1}) = \bar{\mathbf{L}}_t^i \mathbf{x}_{t+1} + \bar{\mathbf{l}}_t^i, \;\; \mathbf{x}_{t+1} \in CR_i \tag{4.28}$$

where

$$\bar{\mathbf{S}}_{t+1}^i = \bar{\mathbf{D}}_t - \bar{\mathbf{C}}_t^T \bar{\mathbf{L}}_t^i, \tag{4.29}$$

$$\bar{\mathbf{s}}_{t+1}^i = \bar{\mathbf{d}}_t - \bar{\mathbf{C}}_t^T \bar{\mathbf{l}}_t^i, \tag{4.30}$$

$$\begin{aligned}
\bar{\mathbf{C}}_t &= \bar{\mathbf{B}}_t^T (\bar{\mathbf{S}}_t^i + \mathbf{Q}_t)\bar{\mathbf{A}}_t + \mathbf{P}_t \bar{\mathbf{A}}_t, \\
\bar{\mathbf{D}}_t &= \bar{\mathbf{A}}_t^T (\bar{\mathbf{S}}_t^i + \mathbf{Q}_t)\bar{\mathbf{A}}_t, \\
\bar{\mathbf{E}}_t &= \bar{\mathbf{B}}_t^T (\bar{\mathbf{S}}_t^i + \mathbf{Q}_t)\bar{\mathbf{B}}_t + \mathbf{R}_t + \mathbf{P}_t \bar{\mathbf{B}}_t + \bar{\mathbf{B}}_t^T \mathbf{P}_t^T, \\
\bar{\mathbf{d}}_t &= \bar{\mathbf{A}}_t^T (\bar{\mathbf{s}}_t^i + \mathbf{q}_t) + \bar{\mathbf{A}}_t^T (\bar{\mathbf{S}}_t^i + \mathbf{Q}_t)\bar{\mathbf{c}}_t, \\
\bar{\mathbf{e}}_t &= \mathbf{r}_t + \mathbf{P}_t \bar{\mathbf{c}}_t + \bar{\mathbf{B}}_t^T (\bar{\mathbf{s}}_t^i + \mathbf{q}_t) + \bar{\mathbf{B}}_t^T (\bar{\mathbf{S}}_t^i + \mathbf{Q}_t)\bar{\mathbf{c}}_t
\end{aligned} \tag{4.31}$$

The recursive update Equations (4.25)-(4.27) run forward from stage 0 to stage t. The overall procedure is referred as forward PWA-LQR.

PWA-LQR Smoother

Executing both backward and forward PWA-LQR for a given constrained linear-quadratic control problem gives the cost-to-go functions v_t and cost-to-come functions $\bar{v}_t$. The sum of v_t and $\bar{v}_t$ results in the total cost, denoted by $\hat{v}_t$, which accumulates the entire cost between stage 0 and stage l by a minimal-cost sequence of

states and controls that visits $\mathbf{x}_t$ at stage t:

$$\hat{v}_t(\mathbf{x}_t) = v_t(\mathbf{x}_t) + \bar{v}_t(\mathbf{x}_t) = \frac{1}{2}\mathbf{x}_t^T(\mathbf{S}_t^i + \bar{\mathbf{S}}_t^i)\mathbf{x}_t + \mathbf{x}_t^T(\mathbf{s}_t^i + \bar{\mathbf{s}}_t^i) + s_t. \tag{4.32}$$

Using $\hat{\mathbf{x}}_t$ to denote the state at stage t that minimizes the total-cost function $\bar{v}_t$, we obtain:

$$\hat{x}_t = -(\mathbf{S}_t^i + \bar{\mathbf{S}}_t^i)^{-1}(\mathbf{s}_t^i + \bar{\mathbf{s}}_t^i). \tag{4.33}$$

The $\hat{x}_t$ is the smoothed state, which results in the sequence of minimum-cost states $\{\hat{\mathbf{x}}_0, ..., \hat{\mathbf{x}}_l\}$ for the given linear-quadratic control problems.

4.3.4 PWA-ELQR: Local Approximation for Nonliear-Nonquadratic Control Problems

The foregoing discussion concerns the linear quadratic control problem. However, most practical systems come under nonlinear, nonquadratic control problems, such as that considered in the present work. Therefore, it is necessary to extend PWA-LQR (both forward and backward) to the case of nonlinear kinematic and nonquadratic local cost function. This is similar to extending the Kalman filter to nonlinear systems. This subsection proposes the PWA-ELQR to deal with the linearization and quadralization. These two approximation procedures are performed in the vicinity of the state and control input candidates for both backward PWA-ELQR and forward PWA-ELQR. A critical challenge in this regard is how to choose candidates about which to linearize the kinematics and quadratize the local cost functions. In the present work, the smoother state $\hat{x}_t$ and the corresponding control $\hat{u}_t$ are selected as the candidates since they bring excellent convergence characteristic for the entire iterative structure of PWA-ELQR. The details of linearization and quadralization are introduced next.

For the backward PWA-ELQR with $0 \leqslant t \leqslant l$, the recursion sequence of updating cost-to-go functions proceeds from $t = l - 1$ until $t - 1$, with the current-best approximation of total-cost at stage $t + 1$:

$$\hat{x}_{t+1} = -(\mathbf{S}_{t+1}^i + \bar{\mathbf{S}}_{t+1}^i)^{-1}(\mathbf{s}_{t+1}^i + \bar{\mathbf{s}}_{t+1}^i). \tag{4.34}$$

Using the transitional kinematic and control policy from the backward procedure,

as introduced in section 4.3.1, the state and control input candidates for stage t can be obtained by:

$$\hat{u}_t = \bar{\pi}_t(\hat{\mathbf{x}}_{t+1}), \hat{x}_t = \bar{\mathbf{g}}_t(\hat{\mathbf{x}}_{t+1}, \hat{\mathbf{u}}_t) \tag{4.35}$$

Then linearize the kinematic $\mathbf{g}_t(\mathbf{x}_t, \mathbf{u}_t)$ about $\hat{u}_t$, $\hat{x}_t$ for (4.4), with:

$$\begin{aligned} &\mathbf{A}_t = \frac{\partial \mathbf{g}_t}{\partial \mathbf{x}_t}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t), \mathbf{B}_t = \frac{\partial \mathbf{g}_t}{\partial \mathbf{u}_t}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t), \\ &\mathbf{c}_t = \hat{\mathbf{x}}_{t+1} - \mathbf{A}_t \hat{\mathbf{x}}_t - \mathbf{B}_t \hat{\mathbf{u}}_t, \end{aligned} \tag{4.36}$$

and quadratize the local cost function $c_t(\mathbf{x}_t), \mathbf{u}_t$ about $\hat{u}_t$, $\hat{x}_t$ for (4.5), with:

$$\begin{aligned} &\begin{bmatrix} \mathbf{Q}_t & \mathbf{P}_t^T \\ \mathbf{P}_t & \mathbf{R}_t \end{bmatrix} = \frac{\partial^2 c_t}{\partial \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} \partial \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t), \\ &\begin{bmatrix} \mathbf{q}_t \\ \mathbf{r}_t \end{bmatrix} = \frac{\partial c_t}{\partial \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) - \begin{bmatrix} \mathbf{Q}_t & \mathbf{P}_t^T \\ \mathbf{P}_t & \mathbf{R}_t \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\mathbf{u}}_t \end{bmatrix} \end{aligned} \tag{4.37}$$

Combining (4.36) and (4.37) with (4.17) and (4.18), the corresponding $\mathbf{S}_t^i$, $\mathbf{s}_t^i$ and control policy π_t can be calculated repeatedly until $t = 0$.

For the forward PWA-ELQR with $0 \leqslant t \leqslant l$, the recursion sequence of updating cost-to-come functions proceeds from $t = 0$ until $t = t + 1$, with the current-best approximation of total-cost at stage t:

$$\hat{x}_t = -(\mathbf{S}_t^i + \bar{\mathbf{S}}_t^i)^{-1}(\mathbf{s}_t^i + \bar{\mathbf{s}}_t^i). \tag{4.38}$$

Using the inverse kinematics and the inverse control policy from the forward procedure (introduced in section 4.3.3), the state and the control input candidates for stage t can be obtained by:

$$\hat{u}_t = \bar{\pi}_t(\hat{\mathbf{x}}_t), \hat{x}_{t+1} = \bar{\mathbf{g}}_t(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \tag{4.39}$$

Take $\hat{u}_t$ and $\hat{x}_{t+1}$ as the candidates about which to linearize the inverse kinematic

$\bar{\mathbf{g}}_t(\mathbf{x}_t, \mathbf{u}_t)$ to get (4.20):

$$\begin{aligned} &\bar{\mathbf{A}}_t = \frac{\partial \bar{\mathbf{g}}_t}{\partial \mathbf{x}_{t+1}}(\hat{\mathbf{x}}_{t+1}, \hat{\mathbf{u}}_t), \bar{\mathbf{B}}_t = \frac{\partial \bar{\mathbf{g}}_t}{\partial \mathbf{u}_t}(\hat{\mathbf{x}}_{t+1}, \hat{\mathbf{u}}_t), \\ &\bar{\mathbf{c}}_t = \hat{\mathbf{x}}_t - \bar{\mathbf{A}}_t \hat{\mathbf{x}}_{t+1} - \bar{\mathbf{B}}_t \hat{\mathbf{u}}_t, \end{aligned} \tag{4.40}$$

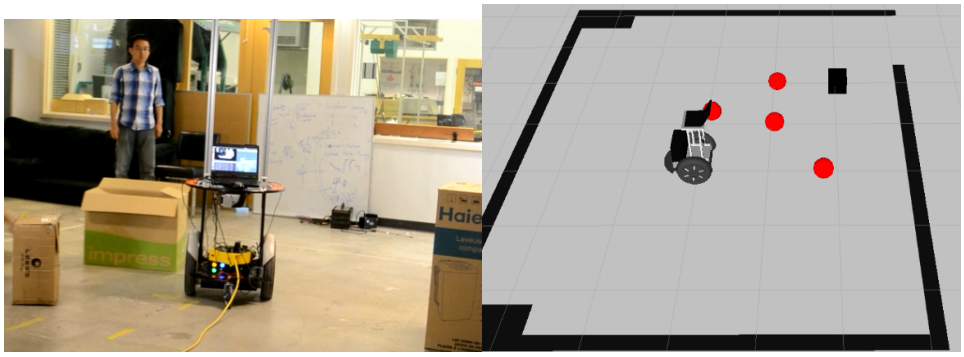
Given the above matrices and vectors, the parameters for quadratizing the local cost function $c_t(\mathbf{x}_t), \mathbf{u}_t$ can be calculated in the same way as (4.37). Then $\bar{\mathbf{S}}_t^i$, $\bar{\mathbf{s}}_t^i$ and the control policy $\bar{\pi}_t$ can be calculated repeatedly up to $t = t + 1$ based on (4.22)-(4.26).

The entire iterative algorithm of PWA-ELQR is summarized in Algorithm 7. From Algorithm 7, it is seen that PWA-ELQR is a locally optimal method due to its requirement for linearization and quadralization around state and input candidates. Therefore, explicitly considering constraints during the iterative process will facilitate restriction of the searching region to the vicinity of the state and input candidates. This advantage of local approximation for nonlinear-nonquadratic control problems helps to avoid divergence, which is caused by blindly following the optimal principle, and hence increases the speed of convergence. The simulation given later will show this characteristic.

4.4 Simulation Studies

In this section, the proposed PWA-ELQR method is applied to the constrained motion planning function of autonomous navigation for a two-wheeled differential-drive mobile robot, named SegPanda, as described in detail in chapter 6. SegPanda is equipped with a Segway nonholonomic mobile robot, a Kinect camera to detect the goal position and a Hokoyu laser ranger finder to detect obstacles (both static and moving) in the environment. All the physical devices (mobile robot, drivers, sensors etc.) are operated by Robot Operating System (ROS, http://www.ros.org/), which is a popular open-source software platform for controlling many types of robots. A main advantage of ROS is that, it provides powerful simulation tools for motion planning and result visualization, and the simulated results can be transferred to a physical experiment seamlessly.

The PWA-ELQR is implemented in both real world environment as shown



(a) **(b)**

Figure 4.1: (a) The real-world environment where SegPanda executes autonomous navigation; (b) The simplified simulation environment corresponding to (a).

in Fig. 4.1a, and simplified simulation environment created by ROS shown in Fig. 4.1b. With the help of ROS, the result from the simulated environment can be transferred conveniently to the corresponding real-world environment. In Fig. 4.1a, the circles represent obstacles (corresponding to boxes in Fig. 4.1b) with radius of 0.3 m.

The simulated SegPanda, which is built to the same scale as the physical Segway mobile robot, has a radius of 0.3 m. The initial state is set to $\mathbf{x}_0^* = (0,0,0)$ and the goal state (the people position in Fig. 4.1a and the black cube position in Fig. 4.1b) to $\mathbf{x}_l^* = (2,2,2)$, where the number of steps is fixed at $l = 60$. PWA-ELQR need not be seeded with an initial trajectory. The algorithm runs until the relative improvement satisfies:

$$\frac{oldcost - newcost}{newcost} < 10^{-4}$$

Three types of constraints are considered in the present control problem:

Differential/kinematic constraints as (1.1);

Control input constraints $v_l, v_r \in [-0.75, 0.75] m/s$;

State constraints $[x, y, \theta] \in [(-10m, 10m), (-10m, 10m), (-3.14rad, 3.14rad)]$.

Actually here the only state constraints is the maximum scope of the entire

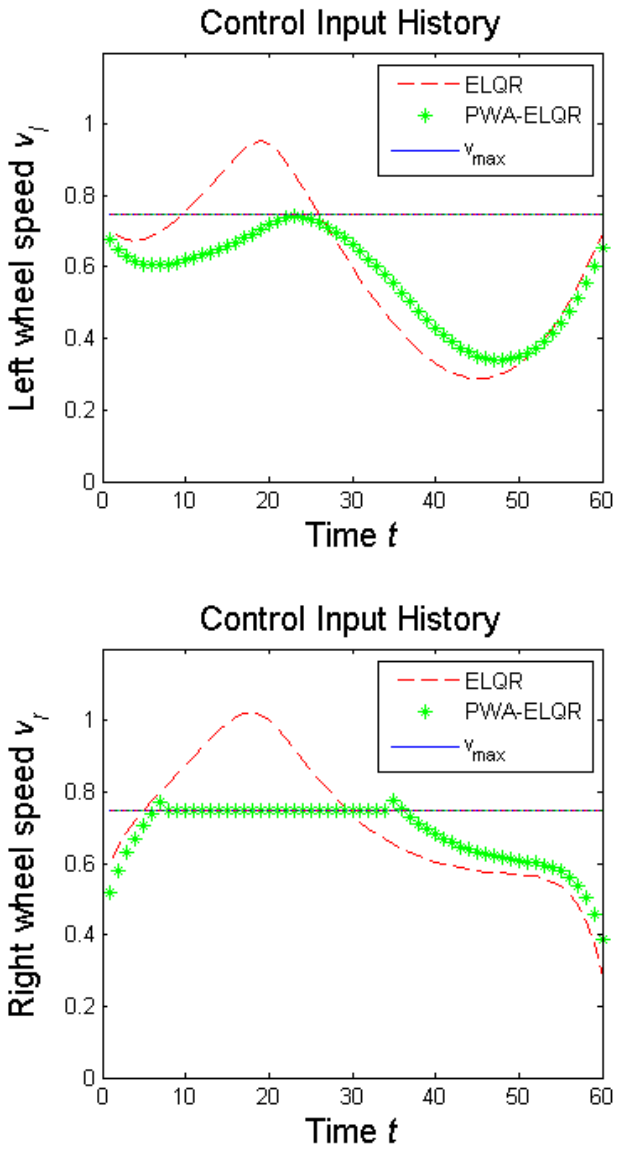


Figure 4.2: Comparison of the optimal trajectories of the input control between ELQR (does not consider constraints on control during motion planing) and PWA-ELQR.

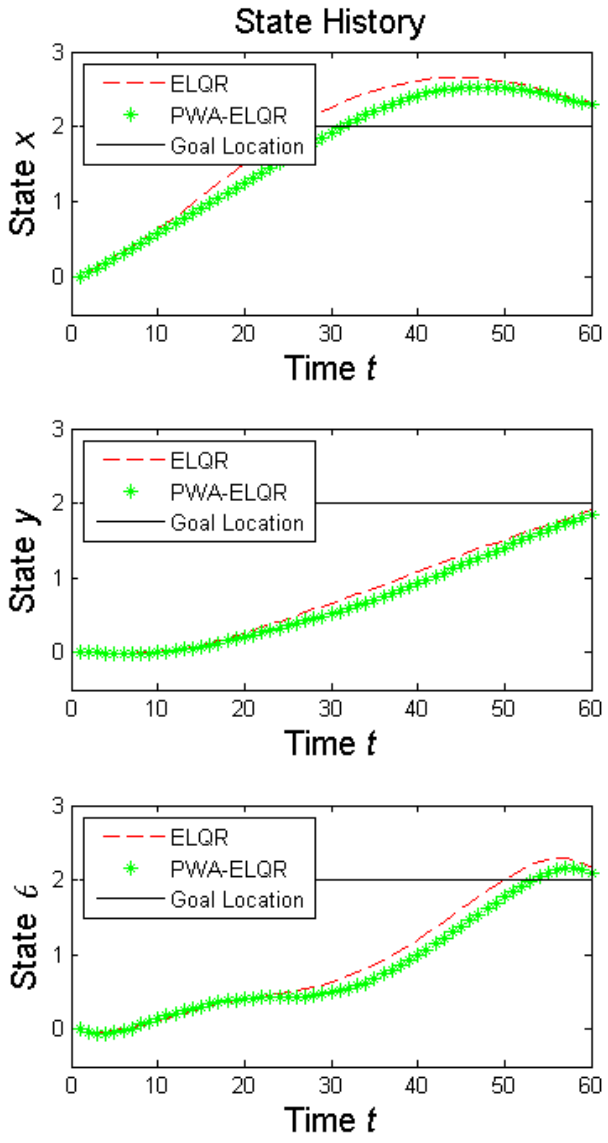


Figure 4.3: The corresponding comparison of the optimal trajectories of the state between ELQR, which does not consider constraints on control during motion planing, and PWA-ELQR(Hence the state trajectory of ELQR can theoretically reach to the goal location).

environment. The obstacles can also bring constraints on states. However, in order to improve convergence performance, a more efficient way is to consider the effect of the obstacles in cost functions defined as:

$$
\begin{aligned}
c_l &= \frac{1}{2}(\mathbf{x}-\mathbf{x}_l^*)^T\mathbf{Q}(\mathbf{x}-\mathbf{x}_l^*),\\
c_0 &= \frac{1}{2}(\mathbf{x}-\mathbf{x}_0^*)^T\mathbf{Q}(\mathbf{x}-\mathbf{x}_0^*)+\frac{1}{2}(\mathbf{u}-\mathbf{u}^*)^T\mathbf{R}(\mathbf{u}-\mathbf{u}^*),\\
c_t &= \frac{1}{2}(\mathbf{u}-\mathbf{u}^*)^T\mathbf{R}(\mathbf{u}-\mathbf{u}^*)+q\sum_i \exp(-d_i(\mathbf{x})),
\end{aligned}
$$

where $q \in \mathbb{R}^+$ gives the weight the obstacle cost occupies in the entire local cost function; $\mathbf{u}^* = [0.25, 0.25]$m/s represents the nominal control input which is the moving speed we expect the SegPanda has during navigation; and the function $-d_i(\mathbf{x})$ represents the signed distance between the SegPanda and the i'th obstacle in the environment. It is seen that the Hessian of the term $q\sum_i \exp(-d_i(\mathbf{x}))$ cannot be guaranteed to be positive-semidefinite, hence its Hessian is regularized by computing the eigen decomposition and setting the negative eigenvalues to zero[72].

With this setting, both ELQR and PWA-ELQR are tested in the environment of Fig. 4.1. Fig. 4.2 shows the comparison of the optimal trajectories the input control between ELQR and PWA-ELQR. It is seen that the result of PWA-ELQR follows the constrains of $v_l, v_r \in [-0.75, 0.75]$m/s rather accurately. On the other hand, ELQR just blindly follows the optimal principle without taking $v_l, v_r \in [-0.75, 0.75]$m/s into account. Common sense tells us that SegPanda cannot realize the control input during execution. The corresponding comparison of the optimal trajectories of the state between ELQR and PWA-ELQR is shown in Fig. 4.3 and in Fig. 4.4. In Fig. 4.4, it is noted that although ELQR results in a somewhat shorter path, it costs 142.97 and can not be executed because of control constraints. PWA-ELQR gives a competitive path that costs 140.77 and can be executed successfully, since PWA-ELQR follows the control input constraints when planning motions.

Another advantage of PWA-ELQR is that, in the present implementation, it takes 8 iterations to achieve convergence, which is 3 fewer than that for ELQR. It is noted that constraints introduce extra calculation for each iteration of PWA-

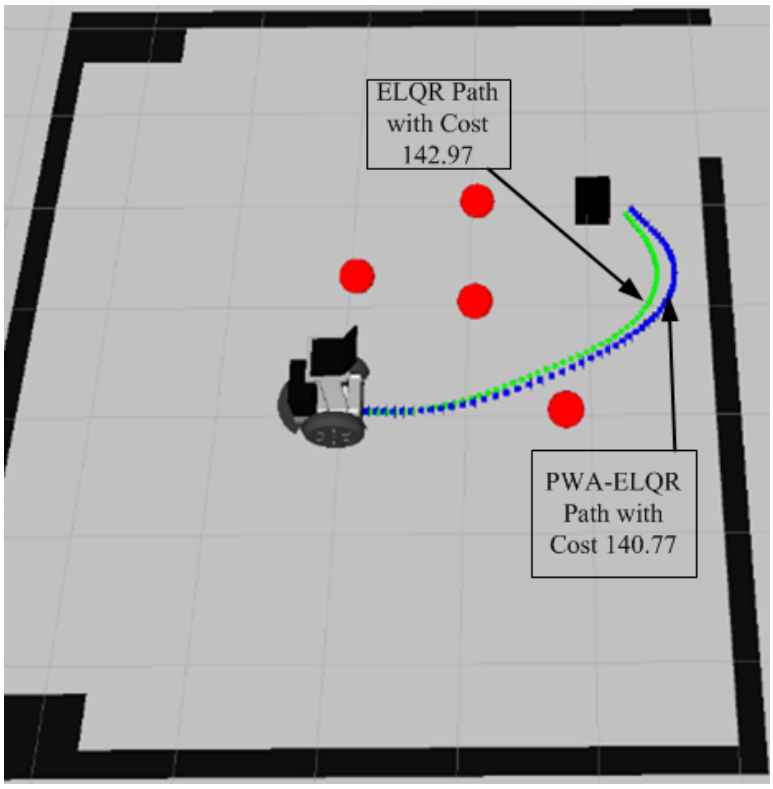


Figure 4.4: Comparison of the optimal trajectories of the state between ELQR and PWA-ELQR in a simplified simulation environment. Although ELQR results in a somewhat shorter path, it costs 142.97 and can not be executed because of control constraints. PWA-ELQR costs 140.77 and is executed successfully since it considers constraints during motion planning.

ELQR. However, with modern computers (e.g., in our case Intel i5 3320 2.60GHz, 4GB RAM), such differences can be ignored with regard to real-time control.

Algorithm 7 The PWA-ELQR Algorithm (proposed in the present work)

Input: local cost functions:$c_t, 0 \leqslant t \leqslant l$; standard kinematic equation: (1.3) and inverse kinematic equation: (1.4); l: number of time steps

Variables: $\hat{\mathbf{x}}_t$:smoothed states; π_t: control policy; $\bar{\pi}_t$:inverse control policy v_t:cost-to-go function;$\bar{v}_t$:cost-to-come function

Output: π_t for all t

$\pi_t = 0, \mathbf{S}_t = 0, \mathbf{s}_t = 0, s_t = 0$
repeat
  $\bar{\mathbf{S}}_t := 0, \bar{\mathbf{s}}_t := 0, s_t := 0$
  for $t = 0; t < l; t = t + 1$ **do**
    $\hat{x}_t = -(\mathbf{S}_t^i + \bar{\mathbf{S}}_t^i)^{-1}(\mathbf{s}_t^i + \bar{\mathbf{s}}_t^i)$ (smoothed states following constraints)
    $\hat{\mathbf{u}}_t = \pi_t(\hat{\mathbf{x}}_t), \hat{\mathbf{x}}_{t+1} = \mathbf{g}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$
    Linearize Eq. (4.40) and Quadratize Eq. (4.37) about $(\hat{\mathbf{x}}_{t+1}, \hat{\mathbf{u}}_t)$
    Compute $\bar{\mathbf{S}}_t, \bar{\mathbf{s}}_t, \bar{s}_t$ and $\bar{\pi}_t$ of forward PWA-LQR based on Eqs. (4.22)-(4.25)
    if $\mathbf{u}_t = \bar{\pi}_t(\hat{\mathbf{x}}_{t+1}) \in [\mathbf{u}_{min}, \mathbf{u}_{max}]$ **then**
      $\bar{\mathbf{S}}_t^i = \bar{\mathbf{S}}_t, \bar{\mathbf{s}}_t^i = \bar{\mathbf{s}}_t, \bar{s}_t^i = \bar{s}_t, \bar{\pi}_t$
    else
      Re-compute $\bar{\mathbf{S}}_t^i, \bar{\mathbf{s}}_t^i, \bar{s}_t^i$ and $\bar{\pi}_t$ of constrained forward PWA-LQR based on Eqs. (4.28)-(4.30)
    end if
  end for
  Quadratize c_l about $\hat{\mathbf{x}}_l$ in the form of Eq. (4.37)
  $\mathbf{S}_l = \mathbf{Q}_l$, $\mathbf{s}_l = \mathbf{q}_l$ and $s_l = q_l$
  for $t = l - 1; t \geq 0; t = t - 1$ **do**
    $\hat{x}_{t+1} = -(\mathbf{S}_{t+1}^i + \bar{\mathbf{S}}_{t+1}^i)^{-1}(\mathbf{s}_{t+1}^i + \bar{\mathbf{s}}_{t+}^i)$ (smoothed states following constraints)
    $\hat{\mathbf{u}}_t = \bar{\pi}_t(\hat{\mathbf{x}}_{t+1}), \hat{\mathbf{x}}_t = \bar{\mathbf{g}}(\hat{\mathbf{x}}_{t+1}, \hat{\mathbf{u}}_t)$
    Linearize Eq. (4.4) and Quadratize Eq. (4.37) about $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$
    Compute $\mathbf{S}_t, \mathbf{s}_t, s_t$ and π_t of backward PWA-LQR based on Eqs. (4.16)-(4.18)
    if $\mathbf{u}_t = \pi_t(\hat{\mathbf{x}}_t) \in [\mathbf{u}_{min}, \mathbf{u}_{max}]$ **then**
      $\mathbf{S}_t^i = \bar{\mathbf{S}}_t, \mathbf{s}_t^i = \mathbf{s}_t, s_t^i = s_t$
    else
      Re-compute $\mathbf{S}_t^i, \mathbf{s}_t^i, s_t^i$ and π_t of constrained backward PWA-LQR based on Eqs. (4.16)-(4.18)
    end if
  end for
until Converged

Chapter 5

People Detection-using MKL-SVM

5.1 Introduction

The present chapter mainly focuses on finding a suitable people detector for application in autonomous navigation. Specifically, a learning scheme that integrates multiple kernel learning (MKL) with support vector machine (SVM), leading to MKL-SVM, is designed in order to increase the precision of detection. Comparing with a state-of-the-art classifier linear support vector machine (LSVM) [73], the designed MKL-SVM can map the features into a higher dimension space so that the linear assumption for data set can be released, while controlling the training time within an acceptable range. An experiment based on open-source dataset, *TUD-Brussels* [74] is used to show that the proposed classifier achieves good performance.

The performance of a sliding window-based detector is significantly affected by the choice of both feature and classifier. For autonomous navigation, the moving system and the dynamic environment require that the people detector is able to consider features caused by movements. Therefore, the present work chooses to describe the people feature in an image by combining static feature HOG(histograms of orientated gradients) and motion feature HOF(histograms of optical flow) as in [35], but does not include color feature due to the specific application for real-time

detection that is treated in the dissertation. Then MKL-SVM classifier based on the combined features is proposed and how to apply the learned classifier to detect a person in an image is explained.

5.2 Combined Features: HOG-HOF

5.2.1 Histogram of Oriented Gradients

HOG, first proposed in [31], characterizes local object appearance with the distribution of local intensity gradients. HOG feature is a local description. The gradient information is quite suitable for depicting the edge features of the body of a person. Apart from preprocessing steps like illumination normalization and image graying, the process of extracting HOG feature has three main steps:

Figure 5.1: The definition of cell and block for a sample image

1. Define some area parameters by dividing the entire sample image $I = height \times width$ into cells, each of which consists of $c \times c$ pixels, and combining the neighbored $e \times e$ cells into one block, as shown in Fig. 5.1. Then calculate the gradients at each pixel in the sample image I, as shown in Fig. 5.2. Obtain one gradient for a pixel by first calculating the vertical difference by the vertical filter $[-1,0,1]^T$ and calculating the horizontal different by the horizontal filter $[-1,0,1]$; then calculate the gradient magnitude image I_m and the gradient orientation image I_o based on the vertical difference image I_x and the horizontal difference image I_y using the following relationships:

$$\begin{aligned} I_m &= \sqrt{I_x^2 + I_y^2} \\ I_o &= \arctan \frac{I_y}{I_x} \end{aligned} \tag{5.1}$$

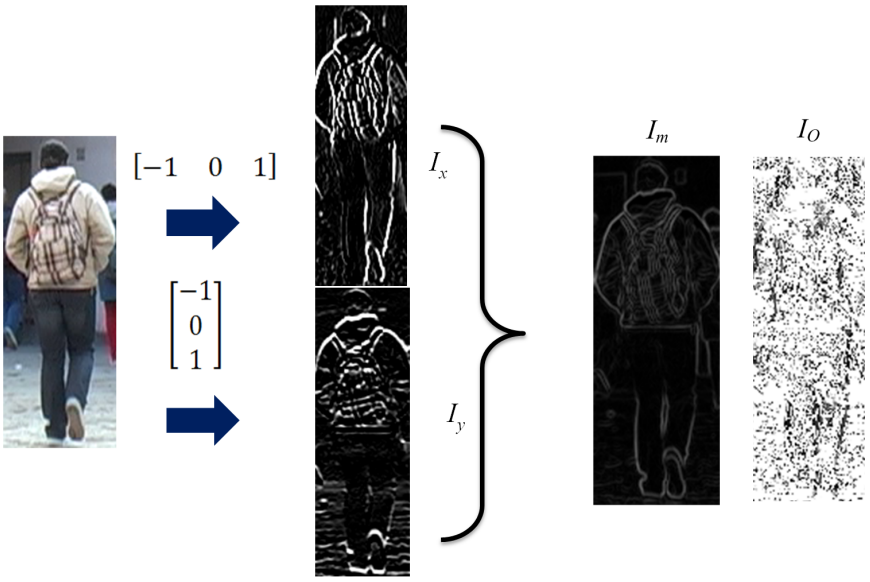


Figure 5.2: Calculation of the gradients of the sample image.

2. Compute the histogram of oriented gradients for each cell by voting orientations I_o in each cell through distributing the gradient orientation into b bins of the histogram. The bins are defined as the gradient orientation interval

with equal space. The present research uses an 8-bins histogram orientation based on a 360-degree setting, as shown in Fig. 5.3. The magnitude of each is obtained by the following equation:

$$\begin{aligned} h(c_i) &= h(c_i) + m \times (1 - \frac{o - c_i}{b}) \\ h(c_j) &= h(c_j) + m \times (\frac{o - c_i}{b}) \end{aligned} \tag{5.2}$$

where c_i, c_j are the two closest bin centers to the orientation of one pixel for the orientation histogram of the cell which covers the pixel; b is the bin size of the orientation histogram; m is the magnitude of each orientation of a pixel; $h(c_i)$ is the magnitude of each bin with center c_i in the orientation histogram, and the same $h(c_j)$ for c_j. Note that c_i and c_i can be obtained through interpolation (e.g., trilinear interpolation). After this, concatenate the histograms of each cell within this block and normalize the value of each histogram to form the HOG feature for each block.

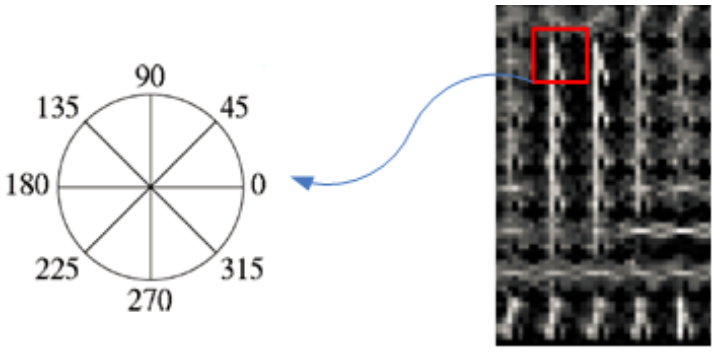


Figure 5.3: 8-bin histogram orientation for one cell

3. Slide the block along the sample image vertically and horizontally with stride sizes $d_x$and d_y. The sliding block is referred to as the HOG descriptor. Then send each HOG descriptor into a trained SVM classifier which generates the people detection result for this block. If this block is positive, it is considered as people area and is marked as people position.

Interpolation and normalization make the HOG representation robust to changes in lighting conditions and small variations in pose. The entire process is summarized in Fig. 5.4.

Figure 5.4: Overview of extracting HOG feature

5.2.2 Histograms of Optical Flow

HOF has been proposed to analyze video content or camera images on moving equipment where the background moves as much as the people in the scene [32]. The basic idea of HOF came from HOG, and their principles are almost same except that HOF uses optical flow gradients instead of pixel intensity gradients. Further improvements can be done for making the HOF scheme more accurate. In the present work, an IMHdiff (Internal Motion Histograms Difference) scheme as in [32] is used for real-time application. The process of calculating HOF is summarized in Fig. 5.5.

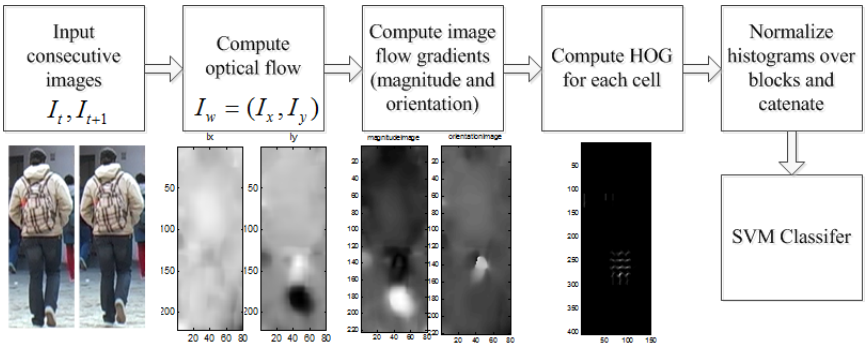


Figure 5.5: Overview of extracting HOF feature

5.3 MKL-SVM Classifier

The classification algorithm is a key element of the detection procedure in the MKL-SVM classifier. Its reliability, computational complexity and recognition efficiency have a significant effect on the overall performance of the system. Two families of classifiers, SVMs and AdaBoosts, provide the best performance, according to a recent evaluation [75]. Since AdaBoosts cannot achieve more competitive results [76], it is not considered here. Linear SVMs often perform well, but the assumption that the dataset is linearly separable inevitably brings errors into the detection. Therefore, in the present work, a multiple kernel learning SVM, that is termed MKL-SVM, is proposed to increase the precision of the people detector.

SVM uses an optimal separating hyperplane between classes by focusing on the support vectors, which are training samples lying at the edge of the class distributions. The remaining training samples can be effectively discarded by this way. The principle of the SVM classifier is that only the training samples lying on the class boundaries are considered for discrimination. A more detailed discussion of SVM is found in [77]. In SVM, a feature map $\mathbf{k}(\mathbf{x_i},\mathbf{x_j}) = \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle$, named kernel function $\boldsymbol{k}$, implicitly maps samples $\boldsymbol{x}$ into a high-dimension feature space. For kernel algorithms, the solution of the learning problem is of the form:

$$f(x) = \sum_{i=1}^{\ell} w_m \mathbf{k}(\mathbf{x_i},\mathbf{x}) + b \tag{5.3}$$

Since there are different kernel types such as radial basis function (RBF) kernel, polynomial kernel, and Hyperbolic tangent kernel, it is often unknown what the most suitable kernel is for the presented task. Hence the designer may combine several possible kernels. One problem with simply adding kernels is that the use of uniform weights may not result in an optimal solution. For example, one kernel unrelated with the labels will just add noise and degrade the performance. Hence, using an optimized kernel function that could fit the data structure well is a desirable way. An intuitive thought is to use a group of optimally weighted kernel functions to fit the data structure. Therefore, combination of multiple features as well as multiple kernels in SVM has been proposed in literature. The present work proposes the use of the following multiple kernel learning approach for people

detection.

A useful way of optimizing kernel weights is MKL. In the MKL approach, a convex formation of M kernels is used as follows:

$$\mathbf{k}(\mathbf{x_i}, \mathbf{x_j}) = \sum_{m=1}^{M} d_m \mathbf{k}(\mathbf{x_i}, \mathbf{x_j}) \tag{5.4}$$

where $d_m \geq 0$, $\sum_{m=1}^{M} d_m = 1$, and M is the total number of kernels used. In the multiple kernel learning problem, for binary classification, assume that we have a training set of N samples $\{\mathbf{x_i}, y_i\}$, $i = 1,\ldots,N$, $y_i \in \{1,-1\}$, where $\mathbf{x_i}$ is translated via M mappings $\Phi_m(\mathbf{x_i}) \mapsto \mathrm{R}^{D_m}$, $m = 1,...,M$, from the input into M feature spaces $(\Phi_1(\mathbf{x_i}),...,\Phi_M(\mathbf{x_i}))$, where D_m denotes the dimensionality of the m-th feature space. We need to solve the following MKL primal problem [78], which is equivalent to the linear SVM when M =1:

$$\begin{aligned}
\text{min} \qquad & \frac{1}{2}\left(\sum_{m=1}^{M} d_m \|\mathbf{w_m}\|_2\right)^2 + C\sum_{i=1}^{N}\xi_i \\
\text{w.r.t} \qquad & \mathbf{w_m} \in \mathbb{R}^{D_m}, \boldsymbol{\xi} \in \mathbb{R}_+^N, b \in \mathbb{R} \\
\text{subject to} \qquad & y_i\left(\sum_{m=1}^{M} d_m \langle \mathbf{w_m}, \Phi_m(\mathbf{x}_i)\rangle + b\right) \geqslant 1 - \xi_i \\
& \forall i = 1,...N
\end{aligned} \tag{5.5}$$

where $\mathbf{w_m}$ is the normal vector to the hyperplane for the feature space Φ_m. In [78]

the MKL dual is derived for the problem (5.5) as given below:

$$\begin{aligned}
&\text{min} && \gamma \\
&\text{w.r.t} && \gamma \in \mathbb{R}, \alpha \in \mathbb{R}^N \\
&\text{subject to} && \mathbf{0} \leqslant \alpha \leqslant \mathbf{1}C, \sum_{i=1}^{N} \alpha_i y_i = 0 \\
& && \underbrace{\frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{k}_m(\mathbf{x_i}, \mathbf{x_j}) - \sum_{i=1}^{N} \alpha_i}_{=:S_m(\alpha)} \leqslant \gamma \\
& && \forall m = 1, ...M
\end{aligned} \tag{5.6}$$

where $\alpha = (\alpha_1, \alpha_2, ..., \alpha_N)$, α_i are Lagrange multipliers, and $\mathbf{w_m} = \sum_{i=1}^{N} \alpha_i y_i \Phi_m(\mathbf{x}_i)$. In order to solve (5.6), a weighted 2-norm regularization formulation has been proposed in [79], in order to deal with the MKL problem. The work in [79] uses an additional constraint on the weights so as to encourage sparse kernel combinations. This algorithm is called SimpleMKL, and it can converge rapidly at comparable efficiency. Therefore, SimpleMKL will be adopted as the MKL approach in the present work.

5.4 Performance Evaluation

In this section, *TUD-Brussels* database is chosen as the evaluation data due to its arguably realistic and challenging dataset, which provides motion pair images for the HOF feature. The motion-pair training set contains 1776 annotated pedestrians recorded from multiple viewpoints taken from a handheld camera in a pedestrian zone. These 1776 annotated pedestrians are positive samples cropped from the 1092 positive image pairs. In the present example 300 positive samples are chosen among the 1776 samples. *TUD-Brussels* also contains 192 negative image pairs, from which 300 negative samples with fixed [height width] size of [128 64] are chopped. Therefore, a total of 600 motion-pair samples are obtained from *TUD-Brussels*.

The sizes of the sliding windows for both extracting histograms of oriented

gradients-histograms of optical flow (HOG-HOF) features and detecting on the targeted image are set as [128 64]. To fit such sliding windows, positive samples need to be resampled at the same size of [128 64]. Within each sliding window, every block contains 2×2 cells each of which having 8×8 pixels. The stride size for both sliding window and sliding block is 8. The bin size for histogram orientation is set at 9 within the scope of [0,2π]. With these settings, each vector of HOG-HOF feature contains 4608×2 components and the entire training dataset has a matrix size of 9261×600.

For training the MKL-SVM and LSVM classifier, 60% of the 600 motion-pair samples are selected randomly as the training subset. Specifically, all the left 40% samples are considered as the testing subset. With the duality gap of 0.01 for the convergence threshold of both MKL-SVM and LSVM, Table 5.1 shows the test results when using different features combined with different classifier.

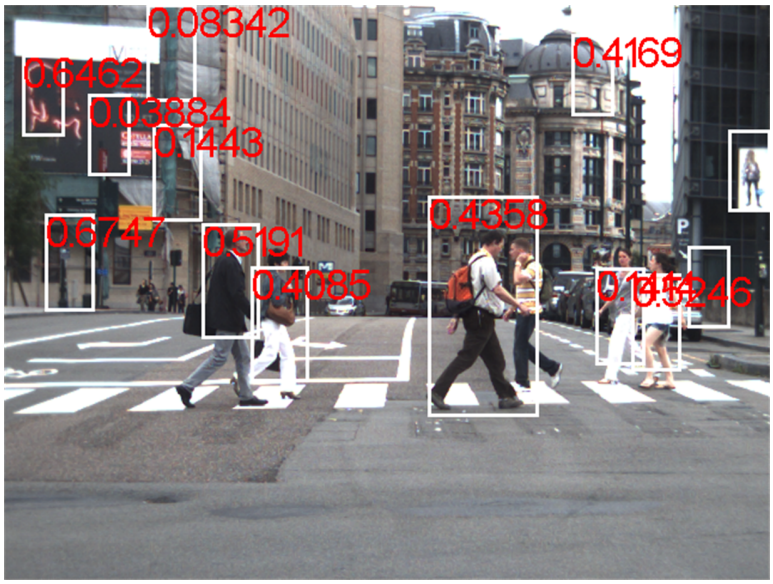
The first row gives the LSVM result, and the remaining rows concern MKL-SVM. It is seen that MKL-SVM has higher precision than LSVM. For MKL-SVM, two common nonlinear kernels, RBF kernel and polynomial kernel (Poly), are used. The tuned parameter for the RBF kernels is sigma, and for the Poly kernels it is degree. Clearly RBF is able to describe the HOG-HOF better than other options. Three combined RBF kernels are enough to reach the highest precision in this evaluation. Further increasing the kernel number cannot produce any improvement.

Table 5.1: Comparisons of different kernels and corresponding results

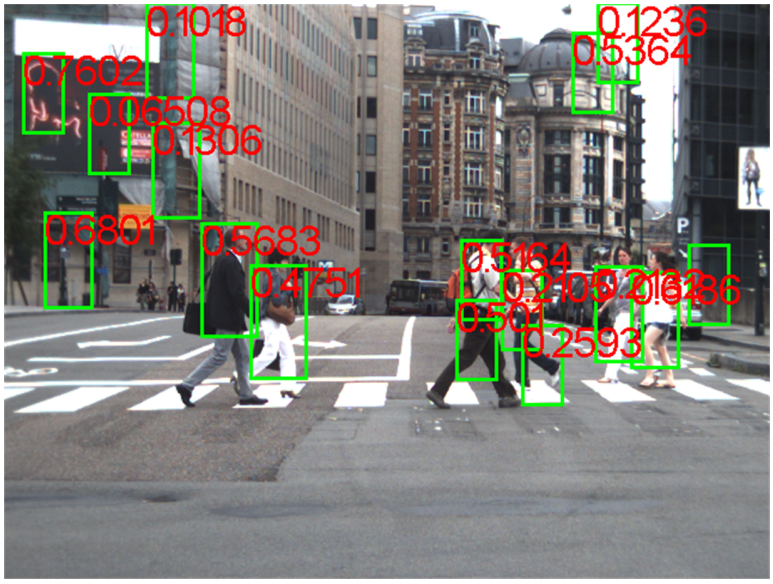
Kernel Type	***Kernel Number***	***Kernel Parameters***	***Precision***
Linear SVM			94.24%
RBF	10	[0.5 1 2 5 8 10 12 15 17 20]	96.68%
RBF	5	[0.5 2 5 10 15 20]	96.68%
RBF	3	[0.5 2 5]	96.68%
RBF	2	[0.5 2]	74.27%
RBF, Poly	8	[0.5 2 5 10 15 20],[1 2]	96.68%
RBF, Poly	6	[0.5 2 5],[1 2 3]	96.27%
RBF, Poly	4	[0.5 2],[1 2]	92.95%

Choosing the best kernel combination according to Table 5.1, Fig. 5.6 shows

an example of the detection results given by the trained classifiers based on HOG features and HOG-HOF features. It is seen that HOG-HOF features result in better detecting performance than with HOG features. The scores on the pictures indicate the possibility that the bounding boxes contain humans. A score larger than 0 gives a positive result while a score less than 0 means a negative answer. The closer it is to 1, the higher the likelihood of people presence. Conversely, closer to 0 means a lower likelihood. The chosen images in Fig. 5.6 show for a classifier a typical complex scenario where there exist some advertising boards with people models or some parts of the construction are similar to a people body profile. This can cause some false positive bounding boxes. Even the motion feature failed to distinguish them from the presence of a real person.



(a)



(b)

Figure 5.6: (a) Detection results (light color bounding boxes) based on HOG-HOF feature using MKL-SVM; (b) Detection results (dark color bounding boxes) based on HOG feature using MKL-SVM.

Chapter 6

Implementation and Experimentation

6.1 Overview

This chapter presents the physical experimentation carried out in the Industrial Automation Laboratory (IAL) of the University of British Columbia. The laboratory mimics a home environment, to implement and evaluate the performance of the methodologies developed in the present research. A prototype autonomous navigation system, named "SegPanda", has been assembled for the experimentation as shown in Fig. 6.1. It consists of three hardware components: a mobile base, a on-board 2D laser ranger finder and a 3D camera. The laser ranger finder is used to build maps and detect obstacles around the robot. The 3D camera is used for detecting people. All of the hardware operates under an open-source system called robot operating system (ROS)[1] which provides rich libraries for a wide variety of hardware. Fig. 6.2 shows an example of a simulated environment built by ROS, which also builds the modeled SegPanda. One big advantage of ROS is that the results of simulations can be transferred to the physical SegPanda very easily.

SegPanda's task in this experiment is to realize the function of human-aware autonomous navigation in a home environment, as shown in Fig. 1.4. Specifically,

[1] http://www.ros.org/

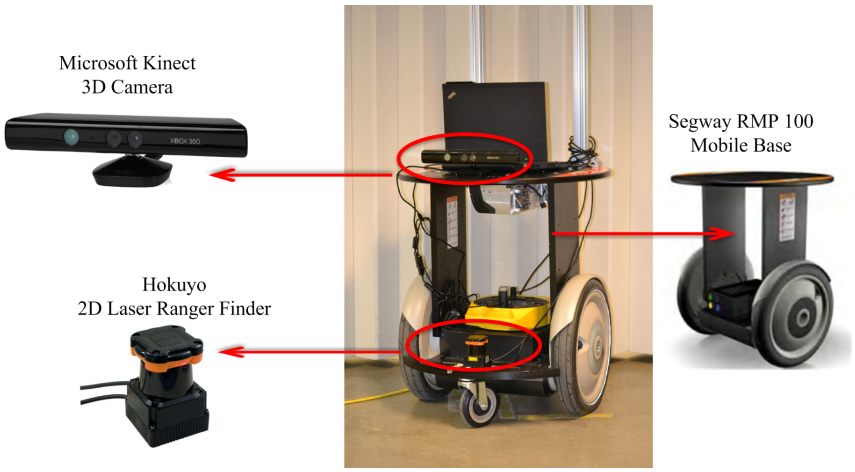


Figure 6.1: The prototype of the autonomous navigation system-SegPanda.

Figure 6.2: An example of a simulated environment built by ROS.

SegPanda needs to first recognize if there are people in front of it and to determine the location of a detected person. Then taking the location of the detected people as the goal location, a planned path is followed to reach that location, while avoiding obstacles simultaneously. The designed experiment does not limit itself to complete a common-sense human following function. In addition, it evaluates whether the proposed methodologies of path planning, motion planning and human detection, each of which can be applied to many other robotic tasks, can perform well. That is more meaningful than just completing an autonomous navigation function.

6.2 The Hardware System

6.2.1 Segway Mobile Base

The mobile base used in this experiment is a Segway RMP100. It is a typical non-holonomic two-wheel differential-drive mobile robot, which can be represented by a unicycle model as in Fig. 1.5b. The Segway mobile base has a build-in odometer which can provide wheel speed information, so that the speed and the position of the robot can be obtained accordingly by using ROS libraries. The related specifications are found in Table 6.1[2].

Table 6.1: Segway RMP100 Specifications

Segway RMP100	***Specifications***
Top Speed	10km/h
Track Width	46cm
Tire Diameter	41cm
Turning Radius	zero degrees
Turning Envelope	64cm
Data Updated Rate	100Hz
Maximum Payload	68Kg

6.2.2 3D Camera: Microsoft Kinect

Kinect was originally developed as a motion sensing device for Microsoft Xbox 360. Because of its relatively low price, it has become a popular experimental sensor in the robotics community. Kinect consists of four key components: infrered optics, red-green-blue (RGB) camera, motorized tilt and multi-array microphone, as shown in Fig. 6.3[3]. The two components related to this experiment are the RGB camera and the infrared optics. The RGB camera provides a 2D image with rich color information, and the infrared optics provides depth information. The com-

[2]http://rmp.segway.com/

[3]http://www.winbeta.org/news/company-behind-microsofts-kinect-sensor-sold-apple-345-million

bination of the RGB camera and the infrared optics can generate red-green-blue-depth (RGB-D) 3D data, which have been used in many robotics applications[80–82].

Figure 6.3: The structure of Kinect components.

The specifications of Kinect are listed in Table 6.2[4]. It is seen that the depth space range of 0.8m-4m is its main disadvantage. However, this range is good enough to realize the experiment in the present work where the people are located in this range.

Table 6.2: Microsoft Kinect Specifications

Kinect	***Specifications***
Viewing Angle	43° vertical by 57° horizontal
Vertical Tilt Range	27°
Frame Rate (depth and color stream)	30 frames per second (FPS)
Depth Space Range	0.8m-4m

[4]http://msdn.microsoft.com/en-us/library/jj131033.aspx

6.2.3 2D Laser Ranger Finder: Hokuyo UTM-30LX

SegPanda uses a single UTM-30LX Hokuyo sensor to map the environment and detect obstacles in a 2D map. The UTM30LX has a 30m and 270 degree scanning range, which is ideal for mapping lab environments (usually fairly large rooms). Another advantage of this sensor is its low weights of only 370g. As a result it is a suitable sensor for small robots. The sensor is located about 20cm above the ground, and hence only works for obstacles that can be seen at that height, such as boxes and walls. But this setting usually can detect most obstacles in a home environment. The green lines pointed by red arrows in Fig. 6.4 show an example of the scanning result of Hokuyo in the IAL lab environment. More details of its specifications are found in Table 6.3[5].

Table 6.3: Hokuyo UTM-30LX Specifications

Segway RMP100	***Specifications***
Detection Range	0.1m to 30m
Accuracy	0.1 to 10m:30mm; 10 to 30m:50mm
Angular Resolution	0.25°
Scan Time	25msec/scan
Weight	Approx. 370g(with cable attachment)

6.3 Software System

6.3.1 People Detection

SegPanda detects people in front of it using a Microsoft Kinect 3D camera. A ROS-wrapped driver, called OpenNI[6], publishes information from the Kinect sensor into ROS, which is then analyzed using point cloud library (PCL) 1.7 [7] functions to provide image and depth information. The people namespace in PCL is used to process the published RGB-D data. This algorithm is focused on speed

[5] https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html
[6] http://wiki.ros.org/openni_launch/
[7] http://pointclouds.org

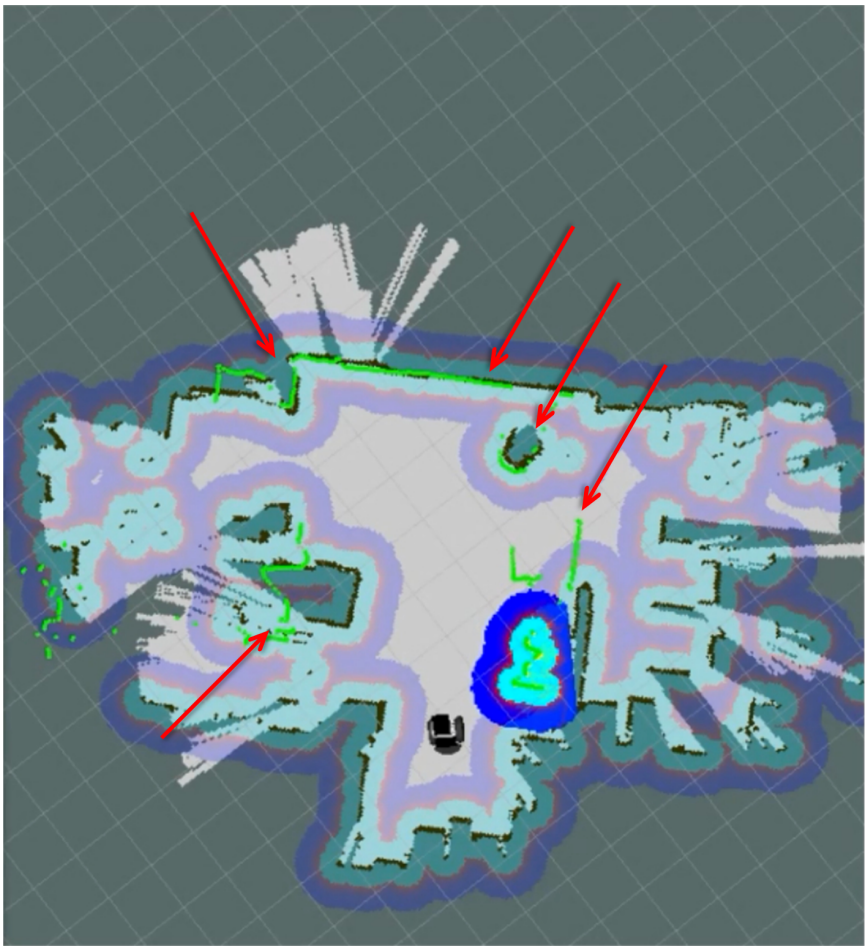


Figure 6.4: An example of the scanning result of Hokuyo in the IAL laboratory environment. The green lines pointed using red arrows show the scanning result of Hokuyo.

and real time processing without the use of a dedicated GPU. The people detection is based on Euclidean clustering using multiple kernel learning-support vector machine(MKL-SVM) parameters trained by the method in Section 5.3 for detection of human bodies. If there are several people in an image, the clustering may group two people together into a single person. To resolve this problem, the Head-Based subclustering in ROS is borrowed to perform after Euclidean Clustering to detect the number of heads and use that information to split groups of people

that were clustered together into separate people. The confidence of each person is calculated using histograms of oriented gradients-histograms of optical flow(HOG-HOF). This algorithm has satisfactory performance in slow moving and stationary people who stand upright, which is executed as a ROS node. When the people detection ROS node is first run, it waits for point cloud data indefinitely until the first RGB-D data is acquired. When the first RGB-D data are acquired, they are displayed in the visualizer and the user is requested to select three floor points manually to calculate where the floor in the 2D RGB image is. After processing, the people with HOG-HOF confidence above a threshold are published onto a visualizer and also published as a custom ROS message. The ROS message that is eventually published is used to send as a navigation goal to make the robot follow around the person. Fig. 6.5 and Fig. 6.6 show the results of the people detection.

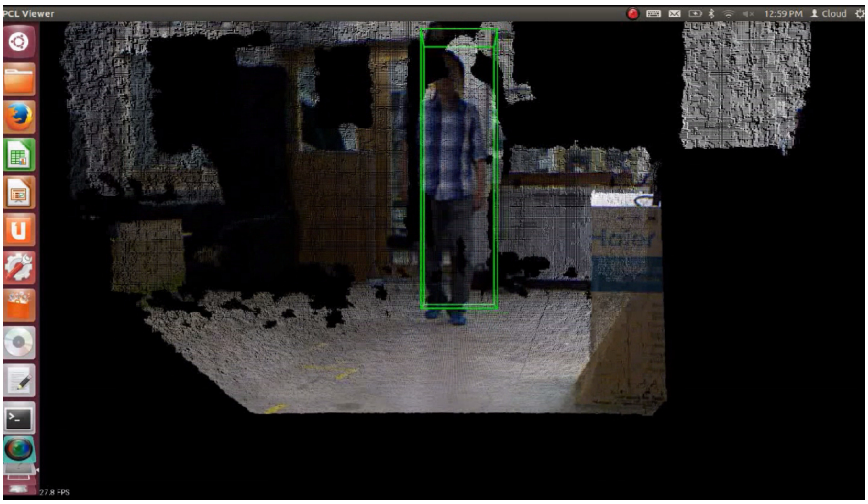


Figure 6.5: The result of people detection in a one-person case.

The Microsoft Kinect has a relatively narrow viewing angle and can only generate relatively high-accuracy RGB-D data from a distance of 0.8 to 4 meters. This limited viewing range plus the fact that the algorithm can only process upright people limits the possible places the person can stand to be seen by the people detection algorithm. The algorithm does not reliably work in all environments. In environments where random object orientations can appear like a person when transformed into a point cloud, the algorithm will have trouble separating people

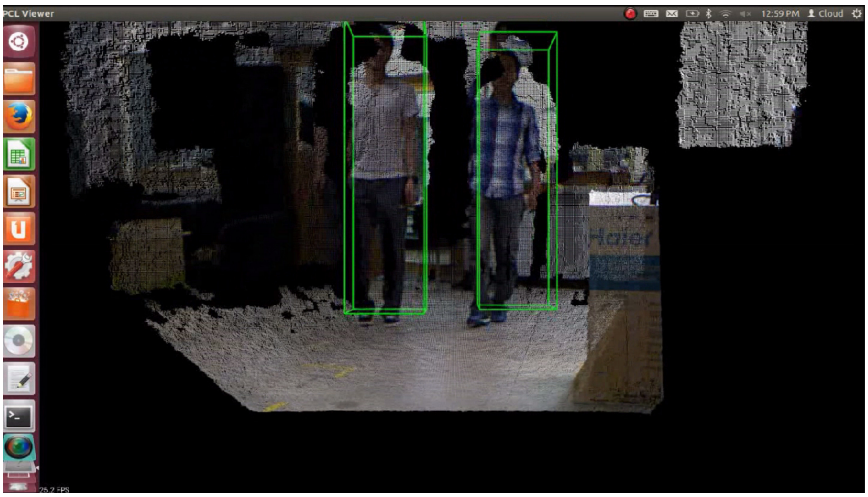


Figure 6.6: The result of people detection in a two-person case.

from inanimate objects. Due to the user entered ground plane coordinate implementation, the ground plane must always stay the same in the Kinect view frame. This means the Kinect cannot be translated along the z axis or rotated along the x and y axis without the people detection algorithm failing.

6.3.2 Autonomous Navigation

SegPanda relies on the ROS Navigation stack to combine all of the elements of the autonomous navigation system and make it to move. The overview structure of the ROS Navigation stack is shown in Fig. 6.7[8].

A goal is given to the navigation stack by either user input or another ROS node configured to interact with the navigation stack. The navigation stack uses the odometry of the mobile base and at least one sensor to localize itself on the global map created by the Gmapping package. It also uses the sensor to create a local map that updates quickly for real time object avoidance. The navigation algorithm operates by using a global costmap and local costmap. Each costmap is modeled as an occupancy grid with the local costmap having a higher resolution. Each grid square is assigned a probability for it to be occupied. The global costmap

[8]http://wiki.ros.org/navigation/Tutorials/RobotSetup

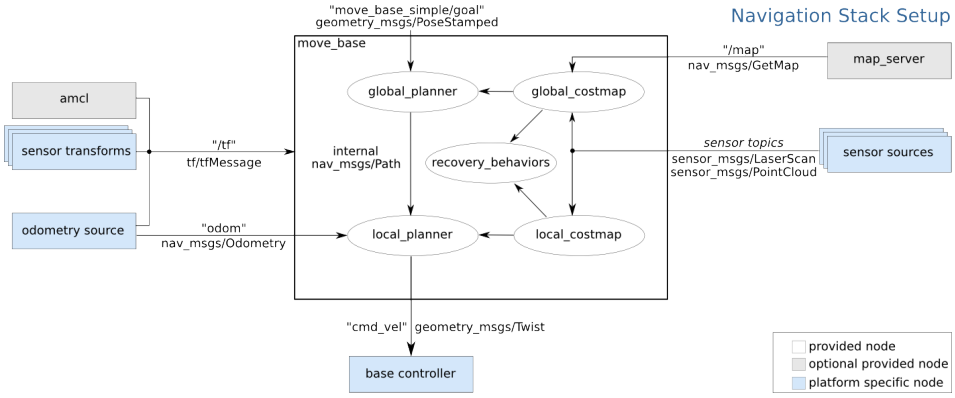


Figure 6.7: The overview structure of the ROS navigation stack.

is taken directly from the map created using Gmapping when the navigation stack first boots.

Amcl package is used to localize the robot inside the global costmap, according to data from the Hokuyo sensor. The localization only works when the odometry is somewhat closely matched with the global map frame. The user is responsible for providing the navigation stack with a pose estimate of where the robot is when the navigation stack first starts. Once the navigation starts, Amcl localizes the robot automatically. This removes the reliance on perfect odometry and prevents SegPanda from drifting out of position on the global costmap as it navigates around the environment. Odometry must still be relatively accurate for Amcl to work. The tests specified in the mapping section for odometry must be satisfied for Amcl navigation as well.

The navigation stack takes a goal input and calculates a global path based on the most current global costmap, by using the path planner presented in Chapter 3. The goal can be input into the navigation stack using RVIZ or a goal sending program that conforms to the navigation API. Once a goal is received, the navigation stack calculates a global path to the goal using the global costmap. The path is usually the shortest way to the goal that the robot considers to be safe. Then, based on the global path, the navigation stack creates a constantly updating local path that goes to the edge of the local costmap (5m x 5m). The local path is created through the motion planner proposed in Chapter 4 based on the local costmap, which attempts

to adhere generally to the global path but has a more accurate and smooth path. The local path is what the navigation stack uses to send the robot velocity commands. Due to the fast update rate of the local path and local costmap, the robot is able to navigate in dynamic environments, and able to avoid randomly slow moving objects.

The navigation package outputs to the mobile base driver by sending velocity commands under the unicycle model. It sends a velocity command that consists of a linear velocity with positive forward and an angular velocity with positive counter-clockwise. The navigation stack is plugged into SegPanda's tf transforms. Using, the odometry message and the base_link message to various robot body link transforms, the navigation stack is able to create an accurate 2D model of the robot inside the global map. The navigation stack takes into account the radius of the robot when navigating, instead of navigating with only a center point. This allows it to solve navigation problems such as how far should the calculated path be away from any obstacles and whether or not the robot can fit in between two obstacles.

The local motion planner (local planner) and global planner can be configured as plugins to ROS. SegPanda uses the previously mentioned global planner and the local planner for its navigation. The Eband local planner is the planner included in the segbot ROS package that we took and modified for our SegPanda. As long as the navigation algorithm conforms to the ROS Navcore API, it can be used as a plugin for either global planners or local planners.

The driver package for the mobile base of SegPanda is taken from the Segbot package available on the ROS Wiki[9], and modified according to SegPanda's hardware. The odometry information is published by the driver and read by the navigation stack for use in navigation and localization. The robot driver creates odometry information by keeping track of its motor rotations and how it would affect the robot's real world position. Since SegPanda is much heavier than the original Segway50rmp, which the driver was written for, the odometry scale and min-max speeds have been modified to suit SegPanda's specifications.

[9]http://wiki.ros.org/segbot

6.4 Experimental Results

The overall scheme of the present experiment is shown in Fig. 6.8.

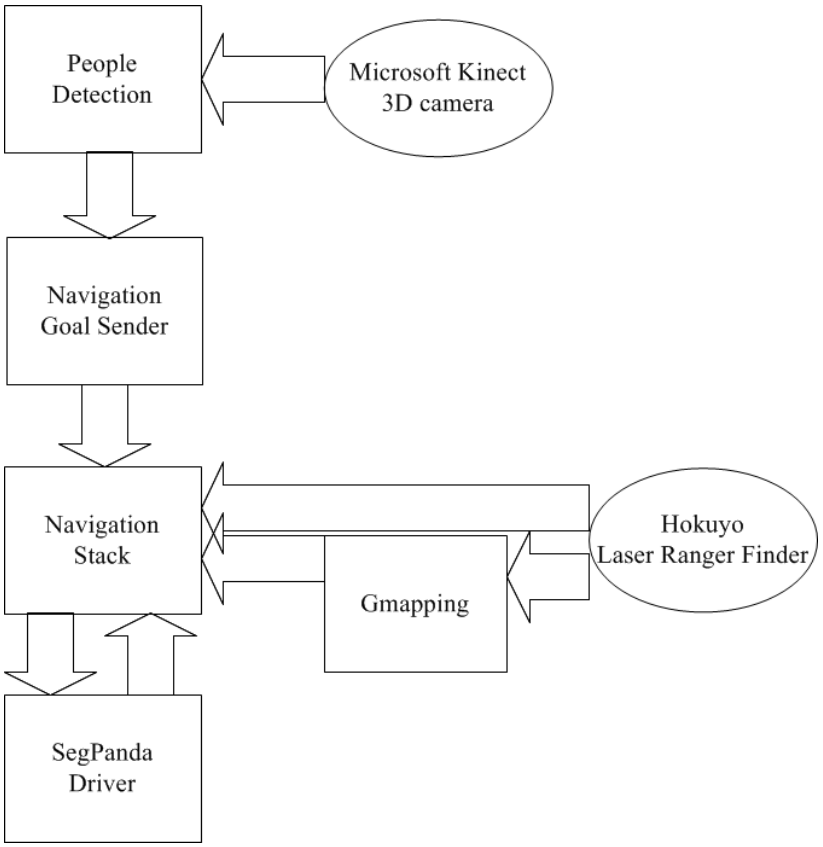


Figure 6.8: The overall structure of the autonomous navigation system

SegPanda uses the dual navigation goal node to implement the goal sending capability. The navigation goal program connects the people detection program with the ROS navigation stack. The people detection node publishes the people location information to a ROS topic. The navigation goal node sender subscribes to that ROS topic and uses the data to determine the goal that is to be sent to the navigation stack. The dual navigation goal node in fact has two nodes each of which sends navigation goals to the navigation stack. Each node works by receiving the position of a person from the people detection node, transforming the data into a position on the map and then sending it to the navigation stack as a goal. The pro-

gram waits until the robot has reached the goal before a new goal can be received and sent. Thus, an individual navigation node is not capable of changing the goal during navigation. A property of the navigation stack is that it can be interrupted in the middle of navigating if a new goal is received from an input source. The implementation on SegPanda takes advantage of this property and has two navigation goal nodes alternatively sending the location of the person as a goal at a frequency of about 1 Hz. Since these two nodes interrupt each other continuously, the navigation goal will always be reset to the current position of the person, allowing the robot to follow a person in real time.

In the event that more than one person is in view or no people are in view, the robot will idle and wait until it sees a person before moving. While idling, it can still be controlled by manual navigation goals or teleoperation key controls. Since the people detection algorithm may interpret noise data as people while the robot is moving, a noise filtering algorithm is implemented to ensure that the robot only starts moving once it is confident that what it sees is a person. The Kinect must have five consecutive frames where a person is in the RGB-D data before it sends as a navigation command. This solves the problem when random noise makes one or two frames see a fake person. However, this does not deal with the problem that occurs when the people detection node legitamately detects a non-person object as a person, which happens fairly often in uncontrolled environments.

In addition, the navigation system that is used on SegPanda requires a map file of the robot's environment constructed beforehand. This map file is used as the global costmap for navigation, allowing the robot to self-localize using Amcl. The used map is created using the ROS wrapper for OpenSlam's Gmapping[10], which is a package available on ROS that is supported up to ROS Indigo. Odometry information published by SegPanda and at least one horizontally mounted sensor (Hokuyo Laser Ranger Finder in our case) are required for Gmapping to make a map of the environment.

Once the global costmap is obtained, using the UTM-30LX hokuyo sensor and most currently global costmap, SegPanda creates a fast updating local costmap in a 5×5m square with itself located in the center (assuming rolling window is set

[10]http://wiki.ros.org/gmapping

to true), as shown in Fig. 6.9. In Fig. 6.9, the blue area denotes the expansion of the obstacles according to the radius of SegPanda. In this way, the robot can be considered as a single point to facilitate the calculation during planning. The obstacles seen in the local costmap are saved to update to global costmap constantly as the robot navigates around. The primary use of the local costmap is to recognize non static objects (such as boxes or chairs) and navigate around them. The updates to the global costmap are not saved in the map file, so each time the navigation stack boots up, a new instance of the global costmap is created.

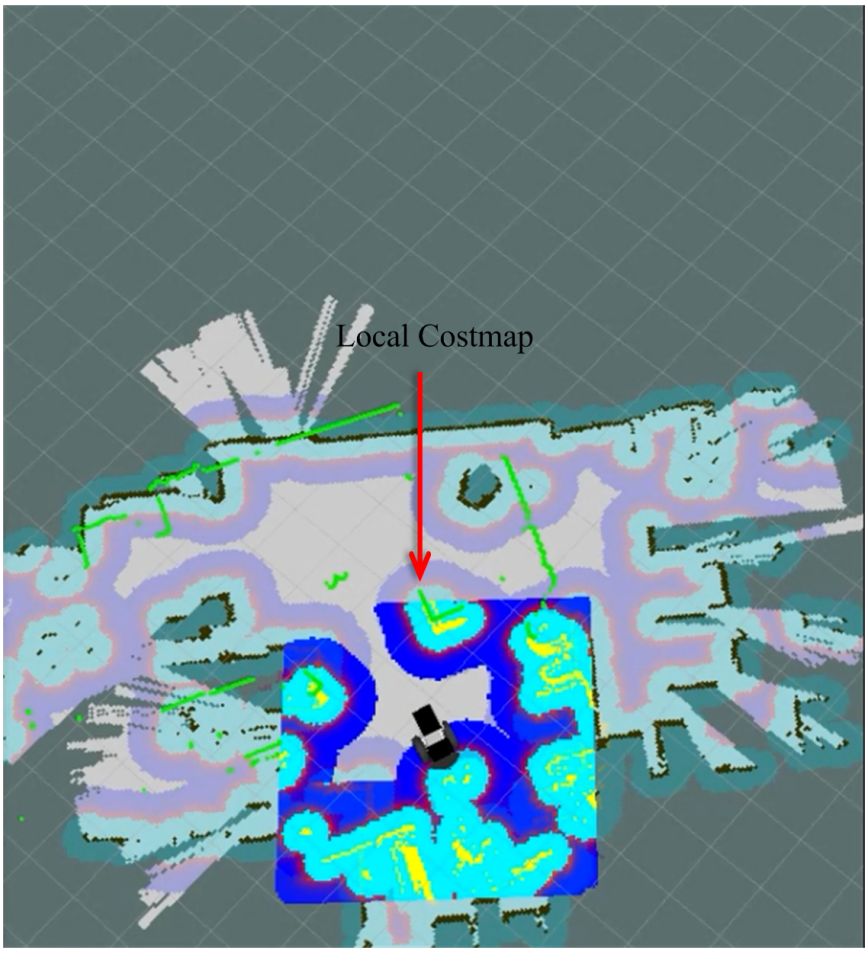


Figure 6.9: An example of the local costmap used for autonomous navigation.

Now all the preliminaries for the core navigation algorithm have been developed and presented. First the global planner provides, based on the current global costmap, a global path which is able to avoid static obstacles. But the global path does not possess specific control commands to move SegPanda. It provides a rough idea on how and where the SegPanda should move. Therefore, the crossing point between the global path and the frame of the local costmap, which is in front of the robot, is sent to the local planner as its temperate goal. The local planner creates a local motion plan and control actions based on the local costmap which contains all the information on the real-time environment. The local costmap always has the robot in the middle and is constantly updated so that the created motion plan can have capability to avoid obstacles (either static or moving) which are unknown when determining the global path. Fig. 6.10 shows the calculated global path (green line) and the local motion path (green circle). It is noted that the local motion path does not completely follow the global path, and it has a new trajectory within the scope of the associated local costmap.

The last step is to control the SegPanda for it to move to the goal location . The driver plugs into ROS, sends the robot odometry and receives unicycle model control commands from the local planner. The unicycle model is a convenient abstraction of linear and angular velocity that the driver converts into left wheel speed and right wheel speed. The navigation stack is responsible for sending these velocity commands that the driver then executes. The complete flowchart of the autonomous navigation system is shown in Fig. 6.11. The people detection module launches the Microsoft Kinect and analyzes the RGB-D data to find any people in the Kinect viewing range. If a person is found, the position of the person relative to the Kinect is published for another module to read. The people navigation goal module waits for the people detection module to send it the position of the person to follow, then sends the position of the person as a goal to the navigation stack. The navigation goal module constantly updates the position of the person and sends new goals to interrupt previous ones so as to follow the person in real time. The navigation stack receives the goal sent by the people navigation goal module and uses a combination of global and local planners to try to navigate to the point. It must receive the transform functions and odometry information from the robot driver and description to work effectively. The navigation stack outputs

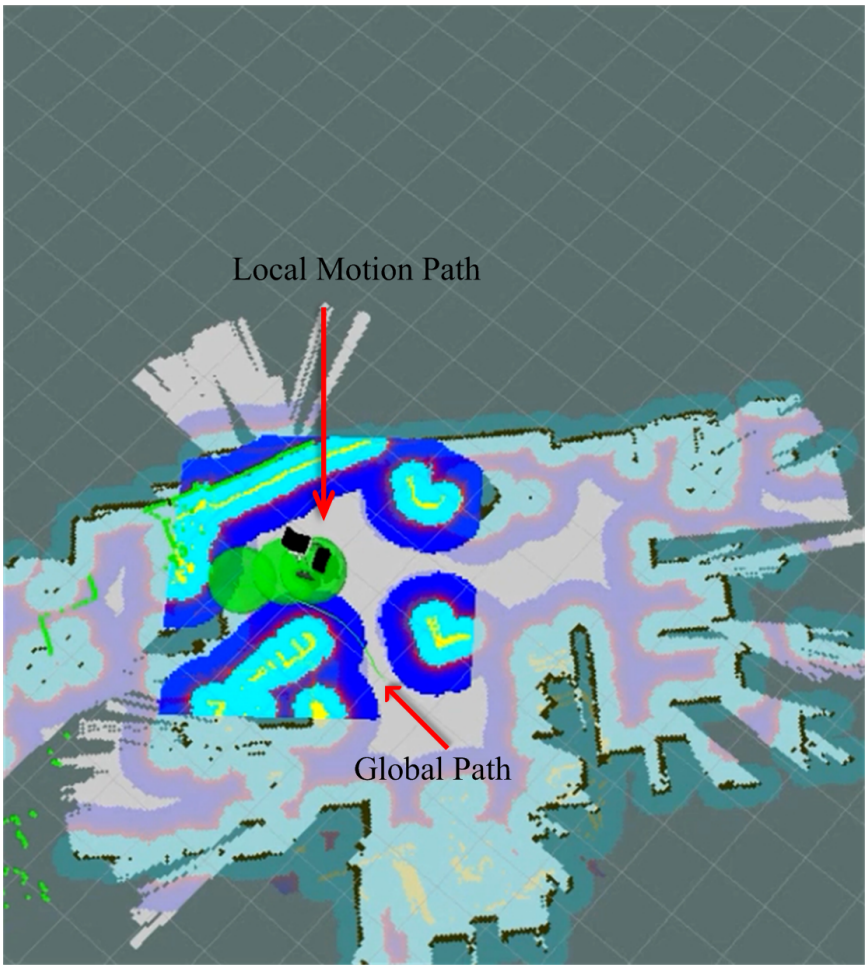


Figure 6.10: An example of the computed global path used for autonomous navigation.

only a linear velocity command and an angular velocity command in each cycle, and will output a stop command once the goal is reached. The robot driver moves the motors based on the commands received from the navigation stack and constantly publishes the odometry information at a rapid rate. Based on the forgoing algorithm, the physical experiment has been executed in IAL to evaluate its validity for navigation. The screen of the experiment demonstration is divided into three different views as shown in Fig. 6.12: the top-left view represents the Kinect view

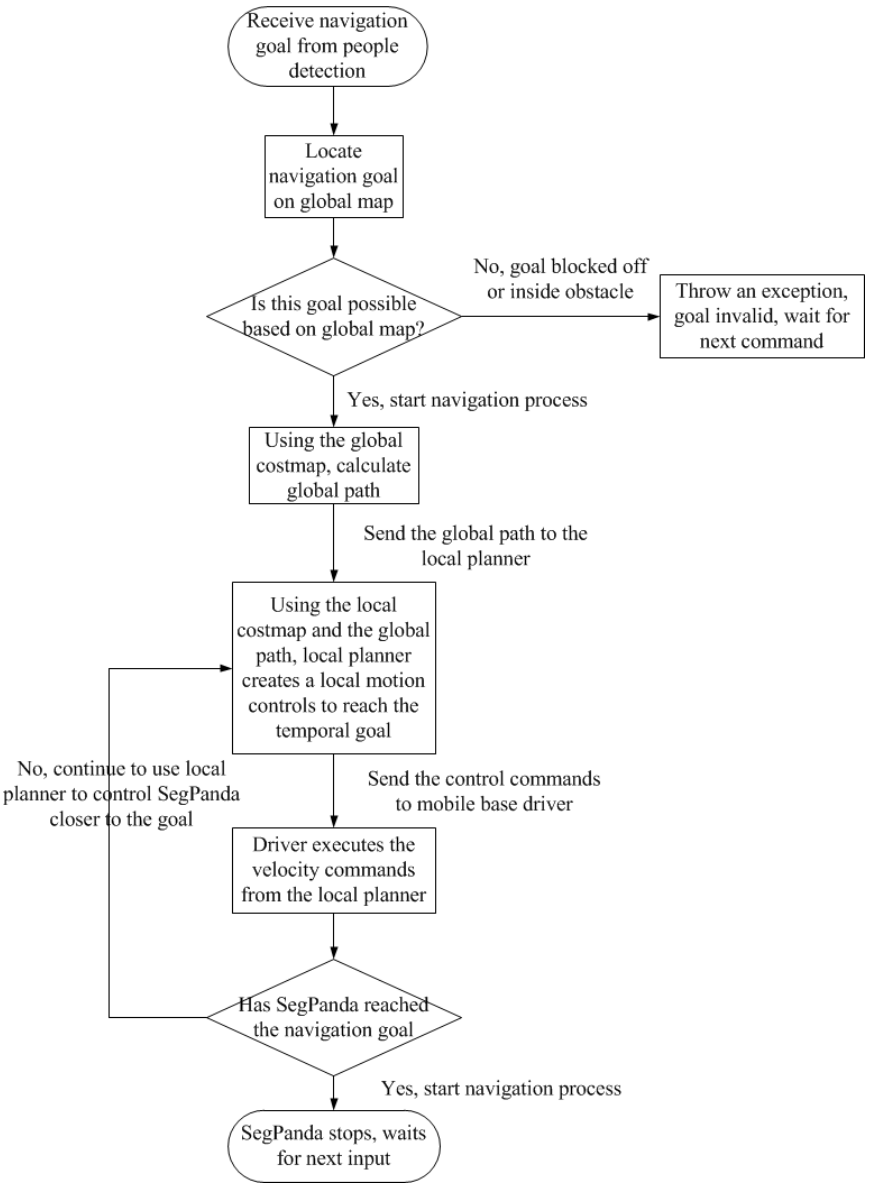


Figure 6.11: The flowchart of the overall autonomous navigation system.

which shows the result of people detection; the down-left view represents the real world as seen by the audience; the entire right view represents the navigation function, or say, the view of SegPanda, operated by ROS. The overall process of the

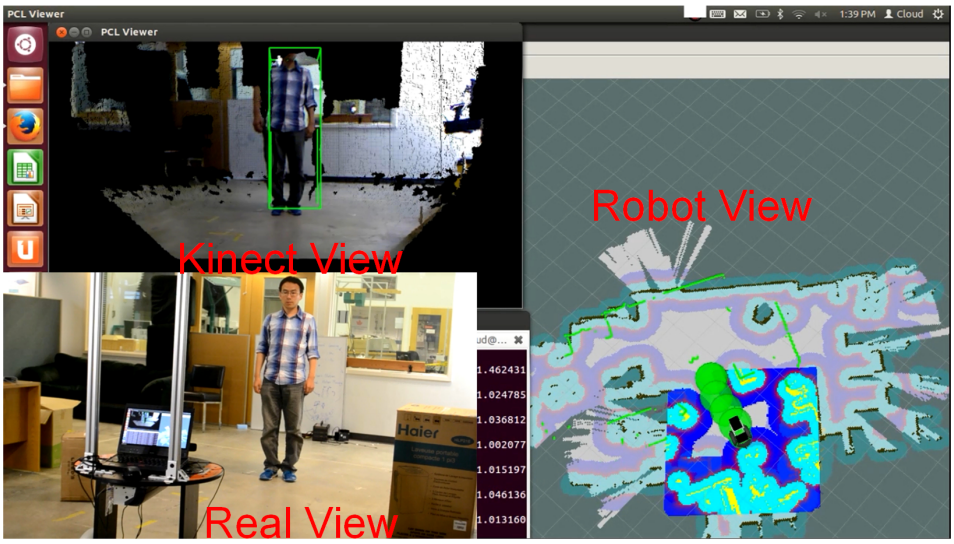


Figure 6.12: The views of the experimental demonstration.

present experiment is shown in the sequence of still images in the following figures. Fig. 6.13 shows the case of detecting people and making the SegPanda successfully move to the goal location, where there are only static obstacles (the boxes in the real world view) in the environment. The generated paths illustrate that SegPanda is able to reach the global goal even when the global goal is changing during the navigation process. Fig. 6.14 is the sequential image set that follows Fig. 6.13. It shows SegPanda's ability of avoiding a moving obstacle during navigation. The red-circled box denotes the moving obstacle, which is unknown in advance, and is pushed out in the middle of the original path, allowing SegPanda to move forward (sequence 1-3 in Fig. 6.14). The sensor of SegPanda detects this change, and SegPanda updates the costmap accordingly, generates alternative paths, and goes forward to the detected person successfully (sequence 4-6 in Fig. 6.14).

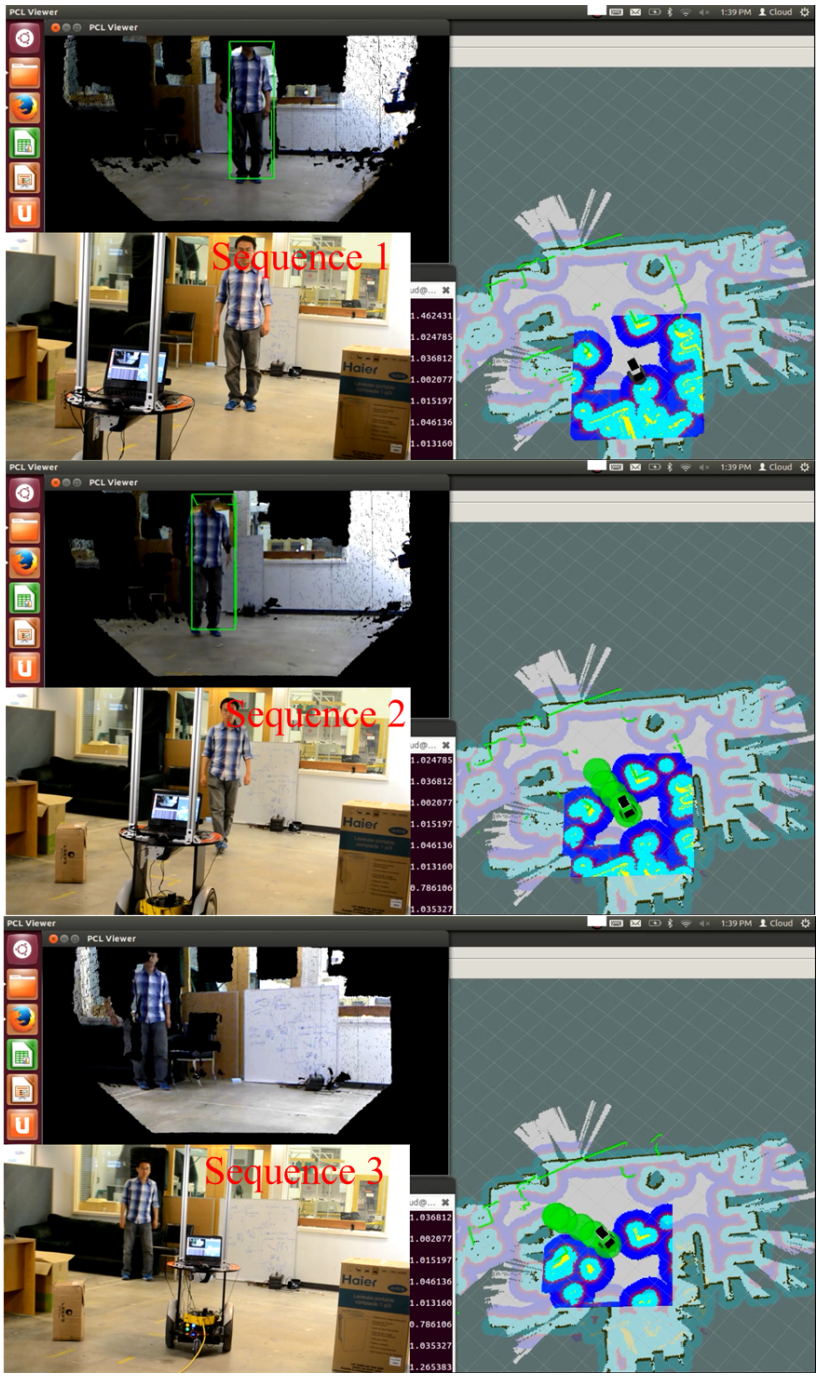


Figure 6.13: The views of the experimental demonstration: avoiding a static obstacle.

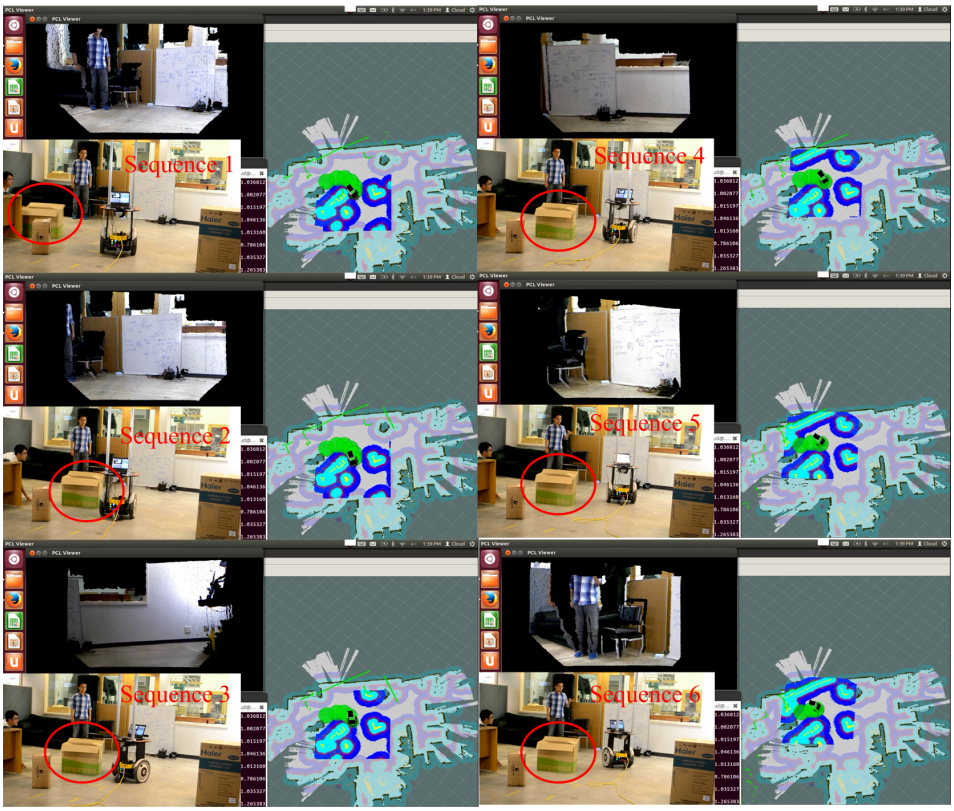


Figure 6.14: The views of the experimental demonstration: avoiding a moving obstacle.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this dissertation, several main challenges related to three aspects of autonomous navigation in a dynamic environment were investigated, primarily with a focus on robot learning methods. These three aspects are: path planning, motion planning and people detection.

Path planning:

An online, robust Q-learning path planning method for an autonomous robot in a dynamic environment with unpredictable moving obstacles was proposed and developed. The strategy of dynamic step size in online robust Q-learning is central when using regime-switching Markov decision process (RSMDP) to represent a dynamic environment. This strategy makes the Q-value iteration robust to the choice of maximum initial step size. Probabilistic roadmap (PRM) contributes to overcoming the curse of dimensionality, which is a common problem in Markov decision process (MDP) and in reinforcement learning. Simulation studies presented in this dissertation have shown that the developed path planner can rapidly and safely find an optimal path in a dynamic environment in the presence of both static and moving obstacles.

Motion planning:

A novel locally optimal feedback control, piecewise affine-extended linear-quadratic regulator (PWA-ELQR), was proposed and developed to solve nonlinear-

nonquadratic constrained motion planning for a nonholonomic mobile robot. In optimization, PWA-ELQR gives an optimal control policy with the objective of minimizing a user-defined cost function. First, piecewise affine-linear quadratic regulator (PWA-LQR) was introduced to handle constrained, linear-quadratic, control problems, by combining quadratic programming (QP) and PWA-LQR smoother. QP specifically handles constraints by providing a piecewise affine control law. PWA-LQR smoother facilitates the convergence performance by combining cost-to-go functions and cost-to-come functions to form total-cost functions. Next PWA-ELQR was proposed and developed to extend PWA-LQR to nonlinear nonquadratic control problems by iteratively performing linearization and quadratization in the vicinity of nominal trajectories. The characteristic of local optimization of PWA-ELQR is inherently suitable to handle constraints since it seeks to keep the changes of trajectories small in scope. Conversely, solving constraints using QP benefits the convergence performance of PWA-ELQR by producing results within the constraint scope, which results in fewer iteration steps. The proposed approach was implemented in both simulation and real-world experimentation. The results demonstrated that PWA-ELQR could provide competitive results compared with ELQR, but with a better executable control policy that takes into account constraints.

People detection:

Under this topic, an multiple kernel learning-support vector machine (MKL-SVM) classifier was designed, combined with multiple features, static feature HOG (histograms of oriented gradients) and motion feature HOF (histograms of optical flows) which have been proposed by other researchers, to detect people in a video sequence for the present task of autonomous navigation. MKL-SVM usually only focuses on one type of feature to increase the detection precision. This is because calculating a multiple kernel matrix for large quantities of data introduces a significant computational burden. In order to increase the detection accuracy while decreasing the computational load, the SimpleMKL algorithm, which is available as an open-source library, was incorporated into the conventional support vector machine (SVM) scheme. SimpleMKL effectively relieves the computational load through a weighted 2-norm regularization formulation with an additional constraint on the weights. Tests carried out on the open-source *TUD-Brussels* database

showed that the detection precision was increased as a result, when compared to LIBLINER SVM.

Finally, an autonomous navigation experiment was designed to combine all the proposed methods into one framework and test if these methods together can work well in a real environment. To this end, an experimental platform, called SegPanda, was assembled. All of the hardware of SegPanda are controlled by robot operating system (ROS) system that also provides rich simulation tools and open-source functions. With the help of ROS, the simulated results can be transferred to physical SegPanda without much difficulties. The experimental results showed that SegPanda could successfully detect and localize people who appear in its Kinect view, set the people location as the goal location, and reach the goal location by successfully avoiding static and moving obstacles during navigation.

7.2 Future Work

The performance comparison between the proposed piecewise affine-extended linear quadratic regulator (PWA-ELQR) and the existing ELQR was presented in this dissertation. The PWA-ELQR originated from ELQR, and the latter approach does not incorporate constraints in the motion planning problem. Hence a more reasonable comparison would be with other methods that can directly resolve the constraint problem in optimal motion planning. For instance, model prediction control (MPC) [25][26][27] has also been proposed to solve constraint problems for motion planning. To this end, comparing the performance of PWA-ELQR with MPC may lead to stronger evidence on the performance of PWA-ELQR in situations with constraints associated with optimal motion planning. A suitable approach for this would be: compare the paths generated by PWA-ELQR and MPC individually in both simulation and experimentation, and then analyse the results by using numerical data. This plan will be undertaken as important future work. In addition, autonomous navigation that was investigated and developed in this dissertation assumes that the robot knows the state of the environment with certainty, so that MDP model can be used successfully for learning how to act in a dynamic environment. Obviously, this is a strong assumption which is typically not satisfied in a real situation. In a real dynamic environment, there always exist uncertainties caused by the

robot itself or the environment. For example, robot's on-board sensors suffer from noise due to mechanical reasons. Another source of uncertainty may come from some hidden states. For instance, if a part of the body of a person is occluded by a table, then the camera fails to detect the person from the surrounding environment. To better solve the problem of autonomous navigation in a dynamic environment full of uncertainty, future work may utilize the partially observable Markov decision process (POMDP), which provides a solid mathematical framework to solve the problem of optimal decision making in environments that are only partially observable to a robot. In addition to applying POMDP in robot navigation in a small 2D grid like that in the work of [83], new problems with hundreds of thousands of state or even with a continuous state space can be solved in just a few of minutes [84–86]. Therefore, exploring POMDP for autonomous navigation with uncertainty and continuous state will be a promising future direction of research.

Another possible future direction is to apply deep learning, specifically convolutional neural networks (CNN), to people detection, since the developments in cloud computation can greatly enhance the practical utility of CNN [87]. In the work of the present dissertation, only one location was needed for autonomous navigation. Hence, a method of detecting a single person was integrated into the autonomous navigation framework. The big advantage of CNN is that there is no need, or little need, to specify design specific features for specific objects. The algorithm itself is able to learn proper features to distinguish from different objects [88]. Therefore, with one learning framework, multiple types of objects can be detected and classified by using CNN, as some recent work has introduced [89, 90].

The Segway robot has only been tested in the IAL and may not work well in other environments, especially those with many complex objects. It also requires to be connected to a power source, as the Kinect and Hokuyo need outlet power to function. A desirable direction of future work can be the expansion of the autonomous navigation application to a larger and more complicated environment like an urban area. If the method proposed in the present dissertation can work well in such dynamic environments, a variety of meaningful applications, such as self-driving vehicles, can arise from it.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21th international conference on Machine learning*, page 1. ACM, 2004.

[2] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[3] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[4] Joelle Pineau, Michael Montemerlo, M. Pollack, Nicholas Roy, and Sebastian Thrun. Probabilistic control of human robot interaction: Experiments with a robotic assistant for nursing homes. In *The second IARP/IEEE/RAS Joint Workshop on Technical Challenges for Robots in Human Environments (DRHE)*, October 2002.

[5] Ryan Luna, Ioan A Sucan, Mark Moll, and Lydia E Kavraki. Anytime solution optimization for sampling-based motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5068–5074, 2013.

[6] EE Kavraki, Mihail N Kolountzakis, and J-C Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, 1998.

[7] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[8] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt*. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1478–1483, 2011.

[9] Jur Van Den Berg, Dave Ferguson, and James Kuffner. Anytime path planning and replanning in dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2366–2371, 2006.

[10] Jur Van Den Berg and Mark Overmars. Planning time-minimal safe paths amidst unpredictably moving obstacles. *The International Journal of Robotics Research*, 27(11-12):1274–1294, 2008.

[11] Dimitri P Bertsekas. *Dynamic programming and optimal control, Third Edition*, volume 1. Athena Scientific Belmont, MA, 2005.

[12] Huzefa Shakir and Won-Jong Kim. Nanoscale path planning and motion control with maglev positioners. *IEEE/ASME Transactions on Mechatronics*, 11(5):625–633, 2006.

[13] Yantao Shen, Eric Winder, Ning Xi, Craig A Pomeroy, and Uchechukwu C Wejinya. Closed-loop optimal control-enabled piezoelectric microforce sensors. *IEEE/ASME Transactions on Mechatronics*, 11(4):420–427, 2006.

[14] Thomas Baumgartner and Johann Walter Kolar. Multivariable state feedback control of a 500 000-r/min self-bearing permanent-magnet motor. *IEEE/ASME Transactions on Mechatronics*, 2015 (To be published).

[15] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics*, August 25-28 2004.

[16] Jan van den Berg. Iterated lqr smoothing for locally-optimal feedback control of systems with non-linear dynamics and non-quadratic cost. In *American Control Conference (ACC), 2014*, pages 1912–1918, 2014.

[17] Jur van den Berg. Extended lqr: Locally-optimal feedback control for systems with non-linear dynamics and non-quadratic cost. In *International Symposium of Robotics Research (ISRR)*, December 2013.

[18] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2696, 2010.

[19] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[20] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2010.

[21] Lucian Busoniu, Alexander Daniels, Rémi Munos, and Robert Babuska. Optimistic planning for continuous-action deterministic systems. In *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, pages 69–76, 2013.

[22] David H Jacobson and David Q Mayne. Differential dynamic programming. 1970.

[23] Evangelos Theodorou, Yuval Tassa, and Emo Todorov. Stochastic differential dynamic programming. In *American Control Conference (ACC)*, pages 1125–1132, 2010.

[24] Li-zhi Liao and Christine A Shoemaker. Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems. Technical report, Cornell University, 1992.

[25] Morteza Farrokhsiar and Homayoun Najjaran. An unscented model predictive control approach to the formation control of nonholonomic mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1576–1582, 2012.

[26] M Farrokhsiar and H Najjaran. A robust probing motion planning scheme: A tube-based mpc approach. In *American Control Conference (ACC)*, pages 6492–6498, 2013.

[27] Morteza Farrokhsiar and Homayoun Najjaran. Unscented model predictive control of chance constrained nonlinear systems. *Advanced Robotics*, 28(4):257–267, 2014.

[28] Noel E Du Toit and Joel W Burdick. Robot motion planning in dynamic, uncertain environments. *IEEE Transactions On Robotics*, 28(1):101–115, 2012.

[29] C Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.

[30] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[31] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR)*, pages 886–893, 2005.

[32] Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. In *European Conference on Computer Vision(ECCV)*, pages 428–441. Springer, 2006.

[33] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. In *The British Machine Vision Conference (BMVC)*, 2009.

[34] Rodolphe Sepulchre, Derek A Paley, and Naomi Ehrich Leonard. Stabilization of planar collective motion with limited communication. *IEEE Transactions on Automatic Control*, 53(3):706–719, 2008.

[35] Stefan Walk, Nikodem Majer, Konrad Schindler, and Bernt Schiele. New features and insights for pedestrian detection. In *IEEE conference on Computer vision and pattern recognition (CVPR)*, pages 1030–1037, 2010.

[36] Paul Viola and Michael J Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[37] David Geronimo, Antonio M Lopez, Angel Domingo Sappa, and Thorsten Graf. Survey of pedestrian detection for advanced driver assistance systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(7):1239–1258, 2010.

[38] Stuart Russell and Peter and Norvig. *Artificial Intelligence: A Modern Approach*. 3 edition, 2011.

[39] Dylan Campbell and Mark Whitty. Metric-based detection of robot kidnapping with an svm classifier. *Robotics and Autonomous Systems*, 2014.

[40] Kun Li and Max Meng. Robotic object manipulation with multilevel part-based model in rgb-d data. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3151–3156, 2014.

[41] S Hamid Dezfoulian, Dan Wu, and Imran Shafiq Ahmad. A generalized neural network approach to mobile robot navigation and obstacle avoidance. In *Intelligent Autonomous Systems 12*, pages 25–42. Springer, 2013.

[42] David Silver, James Bagnell, and Anthony Stentz. High performance outdoor navigation from overhead data using imitation learning. *Robotics: Science and Systems IV*, pages 262–269, 2009.

[43] Chia-Feng Juang and Yu-Cheng Chang. Evolutionary-group-based particle-swarm-optimized fuzzy controller with application to mobile-robot navigation in unknown environments. *IEEE Transactions on Fuzzy Systems*, 19(2):379–392, 2011.

[44] Matt Knudson and Kagan Tumer. Adaptive navigation for autonomous robots. *Robotics and Autonomous Systems*, 59(6):410–420, 2011.

[45] Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *The International Journal of Robotics Research*, 27(2):175–196, 2008.

[46] Daoyi Dong, Chunlin Chen, Jian Chu, and T Tarn. Robust quantum-inspired reinforcement learning for robot navigation. *IEEE/ASME Transactions on Mechatronics*, 17(1):86–97, 2012.

[47] Adam Coates, Pieter Abbeel, and Andrew Y Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th international conference on Machine learning*, pages 144–151. ACM, 2008.

[48] Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

[49] Noel E Sharkey. Learning from innate behaviors: A quantitative evaluation of neural network controllers. *Machine Learning*, 31(1-3):115–139, 1998.

[50] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 797–803, 2010.

[51] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

[52] Tiago Veiga, Matthijs Spaan, and Pedro Lima. Point-based pomdp solving with factored value function approximation. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[53] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[54] Sven Koenig and Yaxin Liu. The interaction of representations and planning objectives for decision-theoretic planning tasks. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(4):303–326, 2002.

[55] Ronald A Howard. Dynamic programming and markov processes. 1960.

[56] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

[57] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[58] Stuart Ian Reynolds. *Reinforcement learning with exploration*. PhD thesis, The University of Birmingham, 2002.

[59] Bohdana Ratitch. *On characteristics of Markov decision processes and reinforcement learning in large domains*. PhD thesis, McGill University, 2005.

[60] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[61] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[62] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[63] Ron Alterovitz, Thierry Siméon, and Kenneth Y Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and Systems (RSS)*, volume 3, pages 233–241, 2007.

[64] Vu Anh Huynh, Sertac Karaman, and Emilio Frazzoli. An incremental sampling-based algorithm for stochastic optimal control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2865–2872, 2012.

[65] G Yin, Vikram Krishnamurthy, and Cristina Ion. Regime switching stochastic approximation algorithms with application to adaptive discrete stochastic optimization. *Society for Industrial and Applied Mathiematics Journal on Optimization*, 14(4):1187–1215, 2004.

[66] Andre Costa and Felisa J Vázquez-Abad. Adaptive stepsize selection for tracking in a regime-switching environment. *Automatica*, 43(11):1896–1908, 2007.

[67] Nocedal Jorge and J Wright Stephen. Numerical optimization,second edition. *Springerverlang, USA*, 2006.

[68] Elizabeth Lai Sum Wong. Active-set methods for quadratic programming. 2011.

[69] Nicholas IM Gould and Philippe L Toint. *Numerical methods for large-scale non-convex quadratic programming*. Springer, 2002.

[70] Nick Gould, Dominique Orban, and Philippe Toint. Numerical methods for large-scale nonlinear optimization. *Acta Numerica*, 14:299–361, 2005.

[71] E Michael Gertz and Stephen J Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1):58–81, 2003.

[72] Nicholas J Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear algebra and its applications*, 103:103–118, 1988.

[73] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[74] Christian Wojek, Stefan Walk, and Bernt Schiele. Multi-cue onboard pedestrian detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 794–801, 2009.

[75] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743–761, 2012.

[76] Dennis Park, C Lawrence Zitnick, Deva Ramanan, and Piotr Dollár. Exploring weak stabilization for motion feature extraction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2882–2889, 2013.

[77] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM, 2008.

[78] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the 21th international conference on Machine learning*, page 6. ACM, 2004.

[79] Alain Rakotomamonjy, Francis Bach, Stéphane Canu, Yves Grandvalet, et al. Simplemkl. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[80] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1691–1696, 2012.

[81] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3748–3754, 2013.

[82] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *Experimental Robotics*, pages 477–491. Springer, 2014.

[83] Amalia Foka and Panos Trahanias. Real-time hierarchical pomdps for autonomous robot navigation. *Robotics and Autonomous Systems*, 55(7):561–571, 2007.

[84] Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A Ngo. Monte carlo value iteration for continuous-state pomdps. In *Algorithmic foundations of robotics IX*, pages 175–191. Springer, 2011.

[85] Hanna Kurniawati and Vinay Yadav. An online pomdp solver for uncertainty planning in dynamic environment. ISRR, 2013.

[86] Konstantin M Seiler, Hanna Kurniawati, and Surya PN Singh. An online and approximate solver for pomdps with continuous action space. In *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)*, 2015.

[87] Rodrigo Benenson, Mohamed Omran, Jan Hosang, and Bernt Schiele. Ten years of pedestrian detection, what have we learned? In *European Conference on Computer Vision (ECCV)*, pages 613–627. Springer, 2014.

[88] Pulkit Agrawal, Ross Girshick, and Jitendra Malik. Analyzing the performance of multilayer neural networks for object recognition. In *European Conference on Computer Vision(ECCV)*, pages 329–344. Springer, 2014.

[89] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *Computing Research Ropository*, abs/1310.1531, 2013.

[90] Ning Zhang, Jeff Donahue, Ross Girshick, and Trevor Darrell. Part-based r-cnns for fine-grained category detection. In *European Conference on Computer Vision(ECCV)*, pages 834–849. Springer, 2014.