**Designing Robot Behavior in Human-Robot Interactions**

by

Changliu Liu

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Masayoshi Tomizuka, Chair
Professor Francesco Borrelli
Professor Michael Christ

Fall 2017

**Designing Robot Behavior in Human-Robot Interactions**

## Abstract

Designing Robot Behavior in Human-Robot Interactions

by

Changliu Liu

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Masayoshi Tomizuka, Chair

Human-robot interactions (HRI) have been recognized to be a key element of future robots in many application domains such as manufacturing, transportation, service and entertainment. These applications entail huge social and economical impacts. Future robots are envisioned to function as human's counterparts, which are independent entities that make decisions for themselves; intelligent actuators that interact with the physical world; and involved observers that have rich senses and critical judgements. Most importantly, they are entitled social attributions to build relationships with humans. We call these robots **co-robots**.

Technically, it is challenging to design the behavior of co-robots. Unlike traditional robots that work in structured and deterministic environments, co-robots need to operate in highly unstructured and stochastic environments. The fundamental research question to address in this dissertation is *how to ensure that co-robots operate efficiently and safely in dynamic uncertain environments.*

The focus of this dissertation is 1) to set up a unified analytical framework for various human-robot systems; 2) to establish a methodology to design the robot behavior to address the fundamental problem.

A multi-agent framework to model human-robot systems is introduced in Chapter 2. In order to address the uncertainties during human-robot interactions, a unique parallel planning and control architecture is introduced in Chapter 2, which has a cognition module for human behavior estimation and human motion prediction, a long term global planner to ensure efficiency of robot behavior, and a short term local planner to ensure real time safety under uncertainties. The functionalities of these components are discussed in Chapter 3 to Chapter 5. Chapter 3 discusses the cognition module, which includes offline classification and online adaptation of various human behaviors. Chapter 4 and Chapter 5 discuss the optimal control or optimization problems for short term and long term robot motion planning. In a cluttered environment, the optimization problems are highly nonlinear and non-convex, hence hard to solve in real time, which may delay the robot's response in emergency situations. Fast online algorithms are developed to handle the issue: the convex feasible set

algorithm (CFS) for the long term optimization, and the safe set algorithm (SSA) for the short term optimization. In particular, the CFS algorithm transforms the non-convex optimization problem into a sequence of convex optimization problems that can be solved efficiently online, which converges in fewer iterations and runs faster than conventional non-convex optimization solvers as shown in Chapter 6.

A method for theoretical evaluation of the designed behaviors is discussed in Chapter 7. The experimental platforms to evaluate the design are discussed in Appendix A. Applications of the proposed method on different co-robots are discussed in Chapter 8 and Chapter 9. Chapter 8 illustrates the application on automated vehicles in the framework of the robustly safe automated driving (ROAD) system. Chapter 9 discusses the application on industrial collaborative robots in the framework of the robot safe interaction system (RSIS).

To my family

# Contents

# List of Figures

# List of Tables

# Acknowledgments

The past five years has been an incredible journey in my life. I would not have come this far without the tremendous help from many people.

My deep and sincere gratitude goes to my Ph.D. advisor Professor Masayoshi Tomizuka, who has been a great mentor to me during the past five years for his profound knowledge, insightful visions, enthusiasm on work, and humor in life. Moreover, Professor Tomizuka respects all my research ideas, inspires me to explore the unknowns, and offers the strongest support in my career development. I sincerely wish I could be a great person, an extraordinary mentor as he is in the future.

To the late Professor J. Karl Hedrick, I am deeply thankful for his generous help and support during my Ph.D study. Professor Hedrick was on my dissertation committee. He also served as the chair of my qualifying exam committee and as a member in my Master of Science committee. His kindness and generosity live forever in my heart.

I am very grateful to Professor Francesco Borrelli and Professor Michael Christ for serving in my dissertation committee. Professor Christ, who also served as the chair in my Master of Art committee and as a member in my qualifying exam committee, is the person that leads me to the beautiful world of Mathematics, which provides me a powerful tool to see through complications. Meanwhile, I am thankful to Professor Pieter Abbeel and Professor Kameshwar Poolla for serving in my qualifying exam committee, as well as to Professor Jon Wilkening for serving in my Master of Art committee.

Special thanks go to Berkeley Fellowship, Denso International America and FANUC Corporation, Japan, the sponsors for the work in this dissertation. The valuable discussions with employees in those companies enhanced this dissertation from the industrial perspective.

Being a member in the Mechanical System Control (MSC) laboratory has been wonderful in the past five years. I received tremendous help from former MSC members: Dr. Wenjie Chen, Professor Xu Chen, Professor Wenlong Zhang, and Dr. Yizhou Wang, who are like big brothers and have provided so many precious suggestions to me in life and career. In addition, I sincerely thank my collaborators for all inspiring discussions: Wei Zhan, Jianyu Chen, Chen Tang, and Long Xin in autonomous driving, and Dr. Chung-Yen Lin, Hsien-Chung Lin, Te Tang, Yu Zhao, Yujiao Cheng, and Yongxiang Fan in robotics. I was also inspired by many current and past colleagues in the MSC lab and thank you all: Chi-Shen Tsai, Kan Kanjanapas, Mike Chan, Cong Wang, Raechel Tan, Junkai Lu, Robot Matthew, Minghui Zheng, Chen-Yu Chan, Liting Sun, Kevin Haniger, Xiaowen Yu, Yaoqiong Du, Shiying Zhou, Shuyang Li, Cheng Peng, Daisuke Kaneishi, Zining Wang, Kiwoo Shin, Yu-Chu Huang, Jiachen Li, Zhuo Xu, and Yeping Hu. In addition, I would like to thank all my friends in Berkeley for the wonderful time that we spent together, in particular, Yubei Chen, Chang Liu, Haoyu Wu, Shiman Ding, and Chen Chen.

Last but not least, my deepest love goes to my parents, my uncle and aunt, my cousins Julia and Angela, and my fiancé Xingxing, for your unconditioned love, support and encouragement.

# Chapter 1

# Introduction

## 1.1 Human-Robot Interactions: An Overview

Human-robot interactions (HRI) have been recognized to be a key element of future robots in many application domains such as manufacturing, transportation, service and entertainment. In factories, robots are leaving their cages and starting to work cooperatively with human workers [69]. Manufacturers are interested in combining human's flexibility and robot's productivity in flexible production lines [62, 64]. Meanwhile, as automated driving is widely viewed as a promising technology to revolutionize today's transportation system [21], substantial research efforts are directed into the field from research groups and companies [33]. As a consequence, autonomous vehicles interact with vehicles driven by human drivers on public roads, which poses new challenges in road safety [106]. Another example is in the field of rehabilitation. In order to rebuild the sensory connection of a patient after a stroke, robots or exoskeletons are needed to guide and assist the patient in walking. There are close physical interactions between the patient and the robot [63]. Others like nursing robots [109] or robot guide dogs [131] are also in great demand and involve HRI.

These applications entail huge social and economical impacts [20, 29]. Future robots are envisioned to function as human's counterparts, which are independent entities that make decisions for themselves; intelligent actuators that interact with the physical world; and involved observers that have rich senses and critical judgements. Most importantly, they are entitled social attributions to build relationships with humans [38]. We call these robots **co-robots**.

Technically, it is challenging to design the behavior of co-robots. Unlike traditional robots that work in structured and deterministic environments, co-robots need to operate in highly unstructured and stochastic environments. The fundamental research question to address in this dissertation is *how to ensure that co-robots operate efficiently and safely in dynamic uncertain environments.*

Due to the broadness and complexity of HRI [29], the following aspects need to be considered when addressing the fundamental question.

- Diverse modes of interactions

  The potential applications of co-robots lie in various domains, with different modes of interactions [30, 70, 107]. A unified model for various HRI applications is indispensable in order to provide a comprehensive understanding of HRI, guide the design of the robot behavior, and serve as an analytical framework for performance evaluation of the human-robot systems.

- Design of the behaviors

  Behavior is the way in which one acts or conducts oneself, especially toward others. We study the methodology of behavior design, i.e. how to realize the design goal (to ensure that co-robots operate efficiently and safely in dynamic uncertain environments) within the design scope (the inputs and outputs of the robotic system).

- Software embodiment of the behavior and its computation efficiency

  The designed behavior is recorded as lines of codes. The complexity of the software will increase dramatically when the environment or the task becomes more complicated. To ensure timely responses to environmental changes and guarantee safety during operation, real time computation and actuation is crucial. Efficient algorithms are highly demanded.

- Analysis, synthesis and evaluation of complex human-robot systems

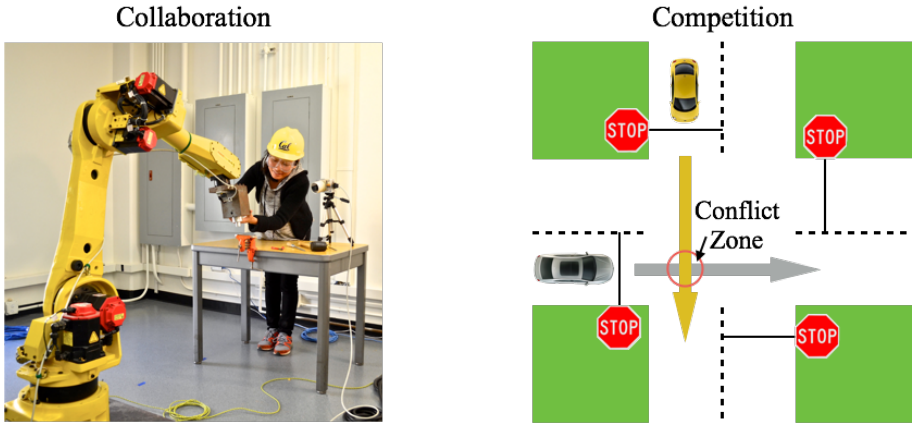
  The effectiveness of the designed robot behavior needs to be evaluated in the human-robot systems. The evaluation can be performed theoretically as well as experimentally. The difficulty in theoretical analysis is the complication of interweaving software modules. The difficulty in conducting experiments is that the tolerance of failure is extremely low when human subjects are in the loop, i.e. human safety is critical. Thus it is important to develop effective evaluation platforms for human-robot systems.

The focus of this dissertation is to 1) set up a unified analytical framework for various human-robot systems; 2) establish a methodology to design the robot behavior to address the fundamental problem. The considerations in those aspects will be elaborated in the following sections.

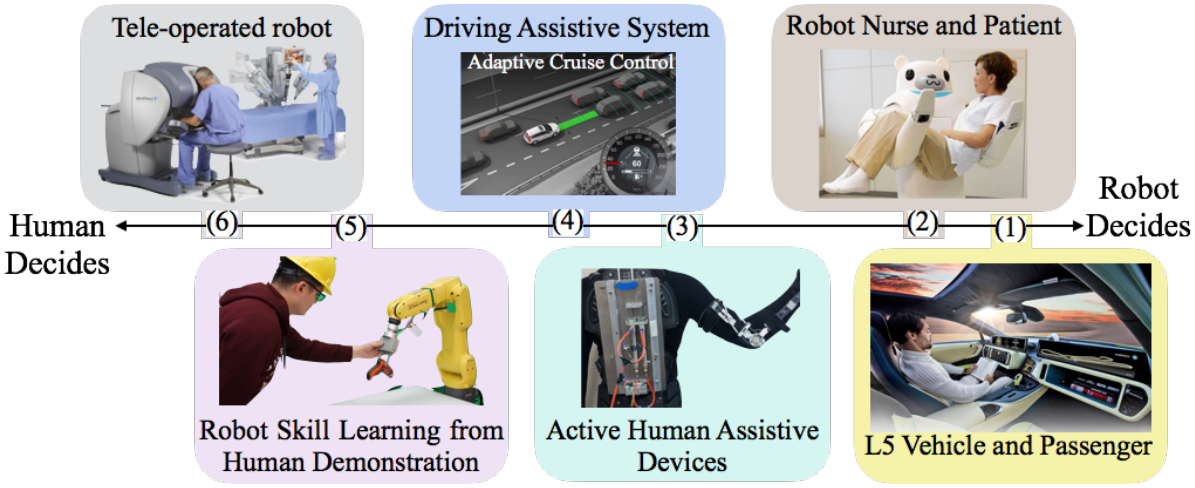
## 1.2 Modes of Interactions

The interaction between a human and a robot may have various modes. We divide it into two kinds of relationships: *parallel relationships* and *hierarchical relationships*.

**Parallel HRI**. In a parallel relationship, a human and a robot are two independent entities that make their own decisions, which is also called peer-peer interaction [30] in literature. Typical examples for parallel relationships are 1) the interaction between an automated vehicle and a human-driven vehicle and 2) the interaction between an industrial




(a) Parallel relationships in human-robot interactions.




(b) Hierarchical relationships in human-robot interactions.

Figure 1.1: Various modes of human-robot interactions.

co-robot and a human worker in production lines. In either case, the robot (the automated vehicle or the industrial co-robot) and the human (the human-driven vehicle or the human worker) are peers as opposed to hierarchical master-slave. In the parallel relationships shown in Fig.1.1a, the actions of the human and the robot either need to be synchronized (e.g. when the human and the robot are moving one workpiece cooperatively) or need to be asynchronized (e.g. two vehicles cannot occupy the conflict zone at the same time when they are crossing the intersection). We call the synchronized actions as *collaboration* and the asynchronized actions as *competition* (since there will always be a vehicle that pass the conflict zone first). Competition is the most common interaction mode. If the resource that human and robot are competing for is the space, competition can be understood as collision avoidance.

**Hierarchical HRI**. In hierarchical relationships, either the human or the robot transfers part of the responsibility of decision making to the other. Typical examples in hierarchical relationships are listed below, which is also shown in Fig.1.1b. (1) The interaction between an automated vehicle and the passenger inside the vehicle, where the human passenger transfers the right of driving to the vehicle[1]. (2) The interaction between a robot nurse and a patient, where the robot decides the trajectory for the patient[2]. (3) The interaction between a human and a human-assistive device such as an exoskeleton. The human can be guided by the robot, but can also "fight" against the robot. (4) The interaction between a human driver and a driver-assistance system[3]. The driver-assistance system can function as a guardian angel [56] that allows human to decide in safe situations and takes over in emergencies, or as a slave that takes charge in safe situations and asks human to take over in emergencies.[4] (5) The interaction between a human and a robot when the human teaches the robot a skill by lead-through. In this case, the robot is following the trajectory decided by the human. (6) The interaction between a tele-operator and a tele-operated robot, where the robot purely follows the command from the human[5]. As discussed in the above examples, the allocation of responsibilities varies for different hierarchical interactions. When human dominates the decision process, it is called supervisory interaction [99] in literature.

The modes of interactions among multiple humans and robots can be built from those basic interaction modes between one human and one robot. A multi-agent framework will be proposed in Chapter 2 to provide a unified framework to analyze various kinds of interactions, where every intelligent entity (human or robot) is viewed as one agent. In this dissertation, parallel relationships are studied with two major application domains: industrial collaborative robots and automated vehicles.

## 1.3 Robot Behavior Design

In this dissertation, we focus on behavior design from the perspective of physical movement, e.g. how to generate safe and efficient motions during interactions.

### 1.3.1 The Three Components in A Behavior System

To generate desired robot behavior, we need to 1) provide correct knowledge to the robot in the form of *internal cost* regarding the task requirements and *internal model*[6] that

[1]Online figure: http://www.autonews.com/article/20160111/OEM06/301119965/autonomous-vehicle-architects-begin-to-contemplate-the-human-inside.

[2]Online figure: http://newatlas.com/riba-robot-nurse/12693/.

[3]Online figure: http://editorial.autoweb.com/autowebs-guide-to-adaptive-cruise-control/.

[4]Note that the level of autonomy of an intelligent vehicle is divided into six categories in the SAE International's new standard J3016 [134], which ranges between the two extremes: either the human driver makes all the decisions or the automated vehicle makes all the decisions.

[5]Online figure: http://www.telepresenceoptions.com/2014/08/r/.

[6]Note that "internal" means that the cost and model are specific to the designated robot.

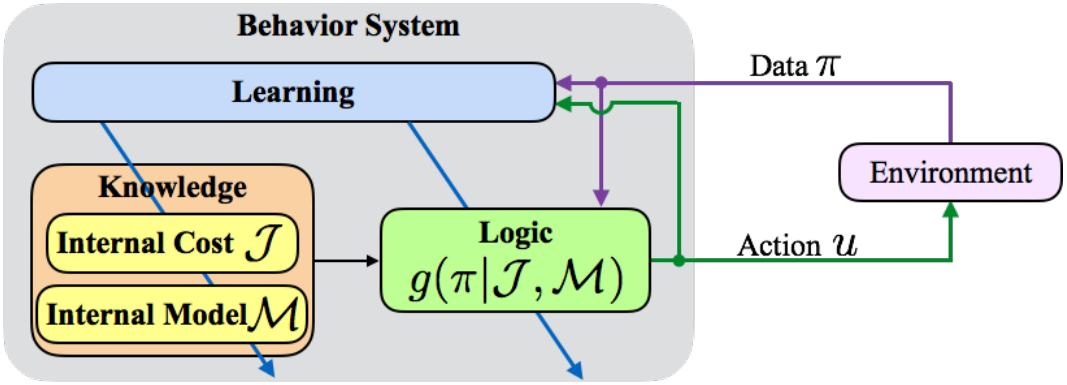



Figure 1.2: Components in a behavior system.

describes the dynamics of the environment, 2) design a correct logic to let the robot turn the knowledge into desired actions, and 3) design a learning process to update the knowledge and the logic in order to make the robot adaptable to unforeseen environments. *Knowledge*, *logic* and *learning* are the major components of a behavior system as shown in Fig.1.2. In the block diagram, the robot obtains data $\pi$ from the human-involved environment and generates its action $u$ according to the logic $g(\pi|\mathcal{J},\mathcal{M})$[7], which is a mapping from information to action that depends on its knowledge: the internal cost $\mathcal{J}$ and the internal model $\mathcal{M}$. The learning process updates the knowledge and the logic based on the data $\pi$, which is necessary since 1) the designed knowledge may not cover all possible scenarios and 2) the environment may be time varying. The mathematical formulation will be further explained in Chapter 2.

The lifetime of a robot is divided into three phases: the design phase, the training phase and the execution phase as shown in Fig.1.3. We call the first two phases offline and the third online. In the design phase, the three components in Fig.1.2 are built into the robot. In the training phase, the robot can learn the knowledge from experience or from human demonstration. The difference between the knowledge learned from human demonstration and the knowledge designed by human is that the former does not require the human to have a mathematical or quantitative representation of the knowledge. In many cases, such mathematical representations is hard to obtain and unintuitive for human. For example, it is easier for a human to gesture a trajectory than to come up with a mathematical function of the trajectory. In the execution phase, the robot performs its task and interacts with its human counterparts. While performing the tasks, the robot can update the knowledge or the logic through online learning. However, due to limitations in computation power, the online learning is restricted to small-scale parametric adaptation. Structural changes such as learning a new skill from scratch can only be performed by offline learning in the training phase. The training phase and the execution phase can be performed iteratively in an everlasting learning system. It is also possible that the robot goes directly from the

[7]The function $g$ is also called a control law in classic control theory or a control policy in decision theory.

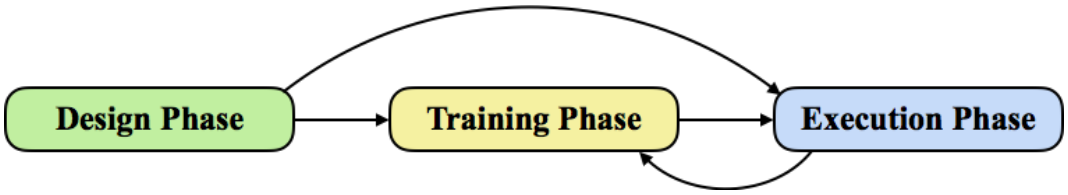
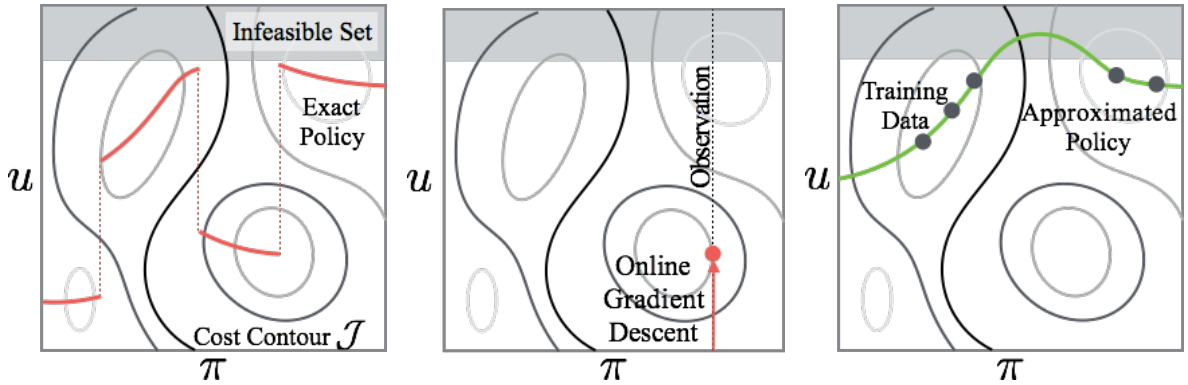



Figure 1.3: The life cycle of a behavior system.




(a) Model-based exact policy. (b) Model-based implicit policy. (c) Model-free policy.

Figure 1.4: Different ways to design the logic.

design phase to the online execution phase without going through the training phase.

Knowledge is the key content of a behavioral system. How much it should be designed (nature) and how much it should be learned (nurture) still remains arguable [28]. Although knowledge can be learned, the two other components, logic and learning, are essentially algorithms, hence should be designed. There are three ways to obtain the logic $g$ as shown in Fig.1.4. The contours in the figures represent the internal cost $\mathcal{J}(u, \pi)$. The darker the color, the higher the cost. Recall that the logic $g$ is a mapping from $\pi$ to $u$. (1) We can solve the optimization $g(\pi) = \min_u \mathcal{J}(u, \pi)$ given $\mathcal{M}$ explicitly in the design phase and get an *exact policy* as shown in the red curve in Fig.1.4a. Since the internal cost is non-convex, the function $g$ may be discontinuous. (2) The optimization can also be solved online in the execution phase. An algorithm (e.g. gradient descent) needs to be designed such that given any observation $\pi$, a desired control input will be obtained. This provides an *implicit policy* as shown in Fig.1.4b. Due to the non-convexity of $\mathcal{J}$, the control input $u$ computed online may only be a local optimum. Note that these two methods assume explicit $\mathcal{J}$ and $\mathcal{M}$, hence are model-based logics. (3) We can also approximate the policy using parametric functions (e.g. neural networks) in the training phase. A set of training data in the form of $(\pi, u)$ pairs should be obtained. Then the function $g$ can be approximated from the training data as shown in Fig.1.4c. Since no explicit knowledge is required, this is a model-free logic.

The three-component behavior system represents a variety of existing methods. We summarize the methods into the following four categories ranging from nature-oriented to nurture-oriented as shown in Fig.1.5.

- Category 1: the designer specifies the internal cost and the internal model and designs logics to solve the optimization explicitly without any learning process. Representative methods are 1) classic control method and Markov decision process (MDP) where exact policies are obtained in the design phase (e.g. in the control of flexible robot joints [63, 137], or in safety critical situations [49, 152]), and 2) model predictive control (MPC) method where the optimization is computed in the execution phase [26, 90, 94].

- Category 2: the designer specifies the internal cost, designs the logic explicitly, and defers to the learning process to identify the internal models. Classic adaptive control and adaptive MPC belong to this category. Application of this approach in human-robot interactions can be found in [46, 81, 98, 121]. The advantage of this method is that it can account for time varying environmental changes, especially when human is in the loop, while the designer still have control over the task performance through explicit design of the knowledge and the logic.

- Category 3: the designer only designs the logic and the learning process explicitly. The knowledge is obtained in the training phase by either trial-and-error or expert demonstration. Representative methods are model-based reinforcement learning and inverse reinforcement learning such as apprentice learning [1, 7, 41]. Application of this approach in human-robot interactions can be found in [4, 103, 135]. The advantage of this method is that mathematical modeling of the task and the environment is no longer required in the design phase.

- Category 4: the designer designs the learning process explicitly and uses a function (e.g. neural network) to approximate the logic. The robot will obtain the knowledge (e.g. parameters in the network) in the training phase. Unlike in Category 3, the knowledge is not explicitly learned, but implicitly encoded in the network. Representative methods are model-free reinforcement learning such as deep reinforcement learning [100] and imitate learning [65]. In addition to human subjects, the imitated object may be a behavior system in Category 1 to 3 [130]. This method is good for problems where 1) the task and the environment are extremely hard to model, 2) the state space is too large and 3) real time computation is critical.

In this dissertation, we work on a method in Category 2 as it allows designers to have more control over the robot behavior in order to guarantee safety during human-robot interactions. The existing methods in designing the three components will be reviewed below.

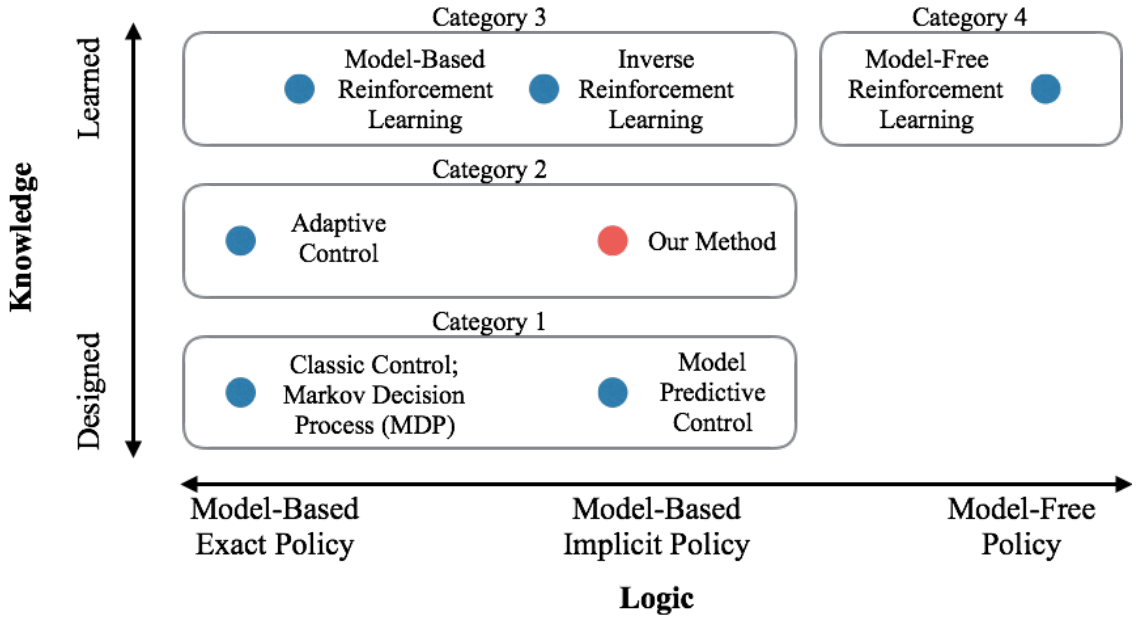



Figure 1.5: The classification of behavior systems.

### 1.3.2 Design of the Knowledge

The knowledge is a combination of the internal cost and the internal model. The internal cost provides incentives for the robot to finish the desired task efficiently as well as to interact safely with other agents in the system. The internal model should be designed in order to match the ground truth behavior of the environment.

**Internal Cost**. Internal cost is a function that depends on the robot's state and action, which can be divided into *static cost* and *dynamic cost*. Static cost is a function on current state and action. Dynamic cost is a function on a trajectory that goes into the future, which is a discounted sum of all static costs along the trajectory[8]. Dynamic cost can be summed either for fixed time horizon (finite time or infinite time) or for fixed task cycle. Fixed time horizon is adopted in linear quadratic regulator (LQR), preview control and receding horizon control. Fixed task cycle optimization is also called minimum time problem as time is treated as a decision variable, which is usually adopted in trajectory planning with a fixed target. Dynamic cost is considered in this dissertation. The internal cost $\mathcal{J}$ will refer to the dynamic cost in the following discussion. The design of $\mathcal{J}$ will be discussed with respect to specific applications as set forth in Chapter 8 and Chapter 9.

**Internal Model**: Since the robot model is usually known in advance, we focus on the models of other intelligent entities in the environment, especially humans. As pointed out in [24], the biggest challenge in human-robot collaboration comes from human factors. To make

[8]In the MDP context, the static cost is equivalent to the reward function and the dynamic cost is equivalent to the value function or the $Q$ function.

co-robots human-friendly, human behavior needs to be modeled, learned, and predicted [6, 18, 114, 153]. The following three kinds of models are frequently used to describe humans' behaviors: (1) the reactive model, in which a human's dynamics is described using a state space model, and the inputs are the identified features that affect his or her behavior [124]. Since his or her motion will be adjusted once the features change, it is considered to be reactive. (2) the rational model, in which a human is treated as a cost minimizer [102, 140], whose cost function depends on both his or her behavior and the robot's behavior. The human will reason his or her best action given his or her belief in the robot's behavior. (3) the Bayesian model, in which a human is treated as a stochastic agent [138], and a Bayesian network [110, 117] is associated to his decisions, i.e. the human's motion follows a probabilistic distribution conditioned on the robot's behavior. It should be noted that it is hard to obtain the cost function or probability distribution that precisely describes a human's behavior in sophisticated problems. Moreover, it is dangerous to assume that a human will always adapt his or her motion to a robot's motion, as it intrinsically implies that the robot can "control" the human [9]. Since no universal model exists that can precisely describe human behavior, learning is necessary.

### 1.3.3 Design of the Logic

The fundamental logic is to find the minimum in the internal cost $\mathcal{J}$ given the data $\pi$ and the internal model $\mathcal{M}$. As discussed in Section 1.3.1 and illustrated in Fig.1.4, we have three ways to design the logic. A model-based exact policy is suitable for well-defined tasks and environments, however, hard to cover a variety of complicated scenarios that co-robots may face. A model-free policy can cover many scenarios (if trained using enough data) and is computationally efficient. However, due to its strong dependency on data and lack of explainability, safety is hard to be guaranteed. In order to increase the flexibility of the logic to account for diverse scenarios and to provide safety guarantees, we adopt the model-based implicit policy. As the mapping $g$ is complicated, it is segmented into several sub functions in a "see-think-do" structure, where "see" deals with perception, "think" concerns with decision making and motion planning, and "do" refers to motor control, which will be further explained in Chapter 2. "Think" is the core function. Motion planning methods can be divided into two groups depending on their planning-horizons, namely the global planner (or the long term planner) and the local planner (or the short term planner).

**Global planners** evaluate the internal cost in a long time horizon. A comprehensive model of the environment is needed. The existing global motion planning methods go into two categories: planning by construction or planning by modification. Planning by construction refers to the method which extends a trajectory by attaching new points to it until the target point is reached. Search-based methods such as A* or D* search [127] and sampling-based methods such as rapidly-exploring random tree (RRT) [66, 68] are typical planning-by-construction methods. Planning by modification refers to the method that perturbs an existing trajectory such that the desired property is obtained. Optimization-based motion planning methods belong to this category, where the perturbation can be understood

as gradient descent in solving the optimization [34, 57, 123]. Each type of methods has pros and cons. The trajectories planned by construction are easier to be feasible. However, as the space is usually discretized during trajectory construction, the constructed trajectories are not as smooth as the trajectories planned by modification. In this dissertation, we focus on optimization-based methods, which will be discussed in Chapter 5. If the environment is previously known without any uncertainty, the global planner is capable of finding the global optimal trajectory. However, in practice, it is computationally expensive to build the model of the environment given limited sensing ability.

**Local planners** try to only plan a few steps in the near future based on the limited knowledge of the environment obtained by current sensory data. Such planners require less computation power, and are good choices for robots with limited sensing abilities. An extreme case of a local planner is the reactive controller where the planning horizon reduces to one. There are usually closed-form solutions for a reactive controller, e.g. a direct mapping from the sensory data to the control, which can greatly relieve the computation burden[9]. Existing reactive methods include virtual force field, potential field method [61], sliding mode method [45] and a family of biologically inspired methods [13]. However, as it only regulates the motion locally, a local planner is sensitive to local optima, e.g. it is possible for the robot to get stuck in some location and cannot reach the target. Thus the global convergence to the target needs to be addressed when designing a local planner. In general, it is very hard to guarantee global convergence for local planners, as the system under consideration is nonlinear (as the robot motion is nonholonomic), time-varying (as the obstacles in the system are moving and deforming), and stochastic (as the information about the environment is limited).

In this dissertation, we propose a parallel planning architecture that has a global planner and a local planner running in parallel to leverage the advantages of the two planners, which will be discussed in detail in Chapter 2.

### 1.3.4 Design of the Learning Process

The learning process updates the knowledge and the logic according to the observed data. In this dissertation, we focus the learning on the internal model, which is a *cognition skill*. In particular, interpretation and prediction of human motions are considered. Learning can be done both offline (in the training phase) and online (in the execution phase). Offline learning is usually used to classify human behaviors and to identify models to describe human behaviors as discussed in [8, 39, 116]. On the other hand, online learning is capable of adapting the offline learned models to humans' time-varying behaviors online as discussed in [43, 121]. The methods for learning will be discussed in Chapter 3.

[9] A reactive control policy is similar to an exact policy in Fig.1.4a in that they are both direct mapping from the sensory data to the control. However, only local information is considered in the reactive control policy, while the exact policy evaluates global information when it is computed in the design phase.

## 1.4 Real Time Computation

To ensure safe and efficient HRI, real time computation is critical, especially in optimization-based motion planning. The problem may be highly nonlinear due to the dynamic constraints, and highly non-convex due to the constraints for obstacle avoidance, making it hard to solve the optimization in real time.

Various methods have been developed to deal with the nonlinearity and non-convexity [104, 129]. One popular way is through convexification [133], e.g. transforming the non-convex problem into a convex one. Some authors tried to transform the non-convex problem to semidefinite programming (SDP) [36]. Some authors proposed to introduce lossless convexification by augmenting the space [2, 53]. And some authors proposed successive linear approximation to remove non-convex constraints [92, 93]. However, the first method requires the cost function to be quadratic. The second approach highly depends on the linearity of the system and may not be able to handle various obstacles. And the third approach may not generalize to non-differentiable problems. One of the most popular convexification method is the sequential quadratic programming (SQP) [128, 136], which approximates the non-convex problem as a sequence of quadratic programming (QP) problems and solves them iteratively. The method has been successfully applied to offline robot motion planning [58, 123]. However, as SQP is a generic algorithm, the unique structure of the motion planning problems is neglected, which usually results in failure to meet the real time requirement in engineering applications.

In practice, the cost function for motion planning is designed to be convex [115, 145], while the non-convexity mainly comes from the physical constraints, e.g. robot dynamics and collision avoidance. By exploiting the geometric structure of the problems, we propose the convex feasible set algorithm (CFS) [76] and the slack convex feasible set algorithm (SCFS) [82] to handle motion planning problems with convex objective functions and non-convex constraints. It will be shown in Chapter 5 and Chapter 6 that these methods perform better than generic methods in motion planning problems.

For short term planning, we exploit the idea of invariant set to transform the non-convex state space constraint into a convex control space constraint using the safe set algorithm (SSA) and the safe exploration algorithm (SEA). SSA ensures that the system state would always stay in a safe set given the predicted human behavior. SEA further constrains the robot motion by the perceived uncertainties in the predictions and seeks to reduce the uncertainty level by learning human behavior actively. These two algorithms will be discussed in Chapter 4.

## 1.5 System Evaluation

The evaluation of the human-robot system can be performed theoretically as well as experimentally.

**Theoretical Evaluation**. During theoretical analysis, the questions to answer are: (1) will the logic find the optimal action given the internal cost and internal model? (2) will the learning process generate converging sequence of the internal models? (3) will the designed internal cost lead to desired behavior of the multi-agent system? The first two questions are modular-wise. The third question is system-wise which concerns with the stability, robustness stability and optimality of the closed loop system, e.g. whether the closed loop multi-agent system is self-organized [91]. System level analysis is challenging due to 1) the complication of interactions among different agents, 2) the difficulty in justifying the assumptions on human behavior; 3) the insufficiency of existing tools in game theory to analyze suboptimal agents[10]. In Chapter 7, a new method will be explored to analyze the system performance with suboptimal agents.

**Experimental Evaluation**. For experimental evaluation of human-robot systems, in order to protect human subjects during the early phase of deployment, various evaluation platforms are developed which take the advantage of virtual reality and teleoperation to separate humans and robots. The platforms will be discussed in Appendix A.

## 1.6 Dissertation Contributions and Outline

This dissertation aims to establish a set of methodologies for designing the behavior of co-robots in order to maximize their performance in dynamic uncertain environments.

The remainder of the dissertation is organized as follows: Chapter 2 overviews the proposed method in a multi-agent framework, while Chapter 3-5 provides detailed discussions of the method. Chapter 3 discusses the identification of human behaviors. Chapter 4 discusses short term or local robot motion planning, and Chapter 5 long term or global robot motion planning. Chapter 6 discusses the optimization solvers for real time computation. Chapter 7 proposes a method to analyze the performance of the multi-agent system. Chapter 8 illustrates the application of the method on automated vehicles in the framework of the robustly safe automated driving (ROAD) system. Chapter 9 discusses the robot safe interaction system (RSIS) as an application of the methodology on industrial collaborative robots.

Each chapter is intended to be self-contained. The relationship among different chapters are shown in Fig.1.6. Some of the work has been published in [77–87, 89] or under review in [76].

The major contributions of the dissertation are summarized below:

- Proposed an unified framework to 1) model human-robot interactions under different modes of interactions and 2) analyze the performance of the multi-agent system with sub-optimal agents. (Chapter 2 and Chapter 7)

[10]The basic assumption in game theory is that all agents are rational, e.g. always behaving optimally, which, however, may not be true as the logic that we designed for the robot may not be perfect.

- Developed a unique robot behavior architecture, in particular a unique parallel planning and control architecture which includes a long term efficiency-oriented planner and a short term safety-oriented planner to leverage the benefits of the two planning schemes. (Chapter 2 to Chapter 5)
- Developed the safe set algorithm (SSA) and the safe exploration algorithm (SEA) for safety-oriented short term planning and control in human-robot systems. (Chapter 4)
- Developed the convex feasible set algorithms (CFS) and the slack convex feasible set algorithms (SCFS) for real time non-convex optimization, which outperforms existing algorithms in optimization-based motion planning problems. (Chapter 6)
- Demonstrated the effectiveness of the proposed method in real world applications on automated vehicles and industrial co-robots. (Chapter 8 and Chapter 9)

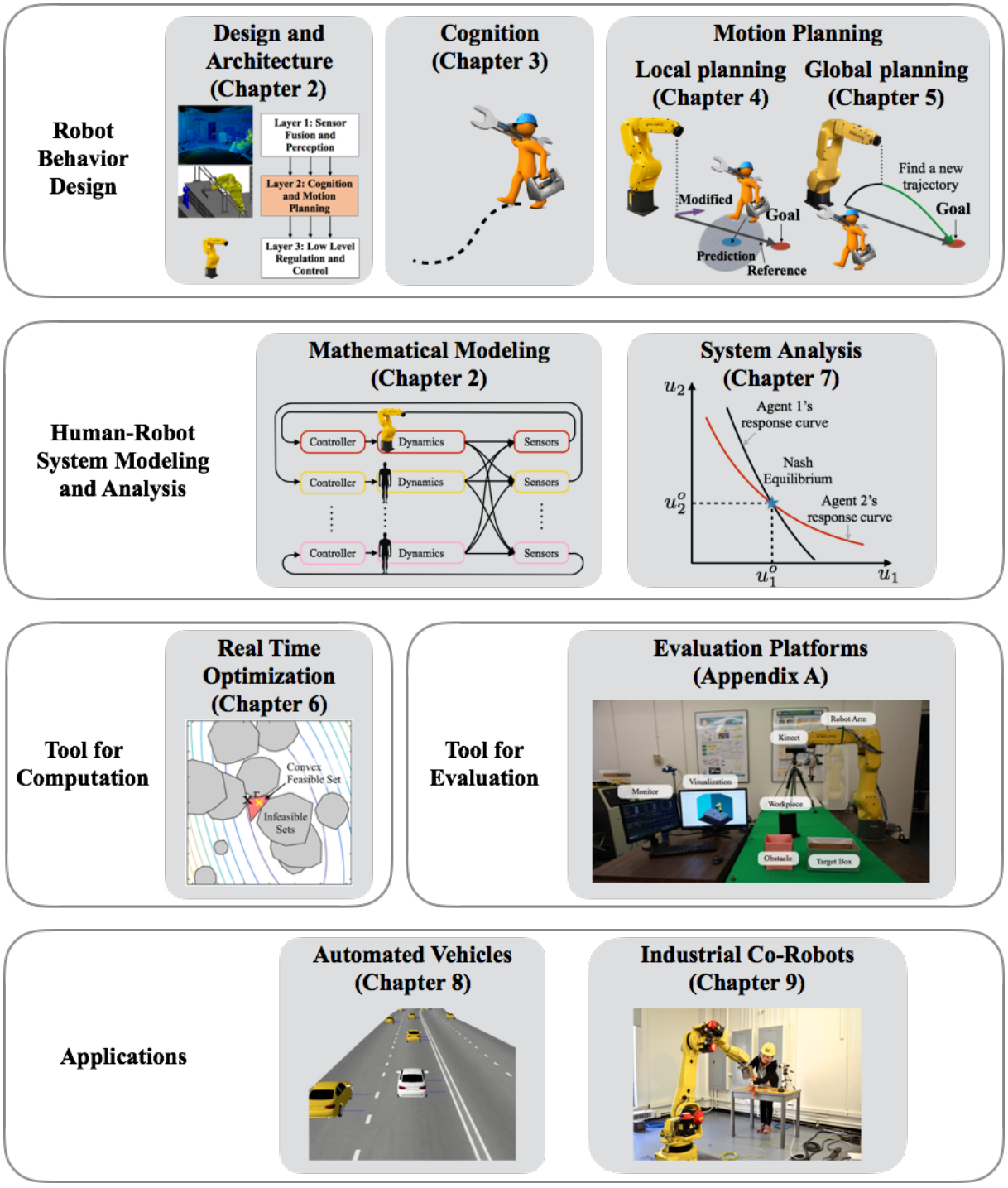



Figure 1.6: Dissertation outline.

# Part I

# Theory

# Chapter 2

# Framework and Architecture

Human-robot interactions can be modeled in a multi-agent framework, where robots and humans are regarded as agents. In this chapter, a multi-agent model will be introduced first, followed by the discussion of agent behavior design and architecture.

## 2.1 The Multi-Agent Model for Human-Robot Interactions

### 2.1.1 The General Multi-Agent Model

An agent is defined to be an independent autonomous entity which has the "see, think and do" ability, i.e. an agent observes through sensors and acts upon the environment using actuators and directs its activity towards achieving goals. The identification of individual agents depends on scenarios. For interactions among human workers and industrial robots, a human worker can be regarded as an agent. For interactions among automated vehicles and human-driven vehicles, a human driver together with the vehicle is viewed as one agent. Moreover, if a group of robots are coordinated by one central decision maker, they are regarded as one agent.

Suppose there are $N$ agents in the environment and are indexed from 1 to $N$. Denote agent $i$'s state as $x_i$, its control input as $u_i$, its data set as $\pi_i$ for $i = 1, ..., N$. The physical interpretations of the state, input and data set for different plants in different scenarios vary. For robot arm, the state can be joint position and joint velocity, and the input can be joint torque. For automated vehicle, the state can be vehicle position and heading, and the input can be throttle and pedal angle. When communication is considered, the state may also include the transmitted information and the input can be the action to send information. Let $x_e$ be the state of the environment.

Denote the system state as $x = [x_1^T, \ldots, x_N^T, x_e^T]^T \in X$ where $X$ is the state space of the system. The open loop system dynamics can be written as

$$\dot{x} = f(x, u_1, u_2, \ldots, u_N, w), \tag{2.1}$$

where $w$ is a noise term.

According to the discussion in Chapter 1.3 and Fig.1.2, agent $i$'s behavior system generates the control input $u_i$ based on the data set $\pi_i$, e.g.

$$u_i = \mathcal{B}_i(\pi_i). \tag{2.2}$$

Note that the function $\mathcal{B}_i$ contains the three components (knowledge, logic and learning). When there is no learning process, then $\mathcal{B}_i(\pi_i) = g_i(\pi_i|\mathcal{J}_i, \mathcal{M}_i)$ where $g_i$ is the logic function of agent $i$ and $\mathcal{J}_i$ is its internal cost. Agent $i$'s internal model $\mathcal{M}_i$ includes the estimates of the system dynamics (2.1) and other agents' behaviors, i.e. the function $\mathcal{B}_j$ for $j \neq i$. The data set contains observations on the system state $x$, which is a combination of the measured data and the communicated information. Agent $i$'s data set at time $T$ contains all the observations $y_i(t)$ from the start time $t_0$ up to time $T$, i.e. $\pi_i(T) = \{y_i(t)\}_{t\in[t_0,T]}$ where

$$y_i = h_i(x, v_i), \tag{2.3}$$

and $v_i$ is the measurement noise.

Applying (2.2) and (2.3) in the open loop dynamics (2.1), the closed loop dynamic equation becomes

$$\dot{x} = \mathcal{F}(x, v_1, ..., v_N, w|\mathcal{B}_1, \ldots, \mathcal{B}_N). \tag{2.4}$$

The system block diagram for the general multi-agent system is shown in Fig.2.1 according to (2.1), (2.2) and (2.3).

### 2.1.2 Models for Different Modes of Interactions

As discussed in Chapter 1.2, the relationships between human and robot are complicated. Nonetheless, the general model can cover the diversity of HRI.

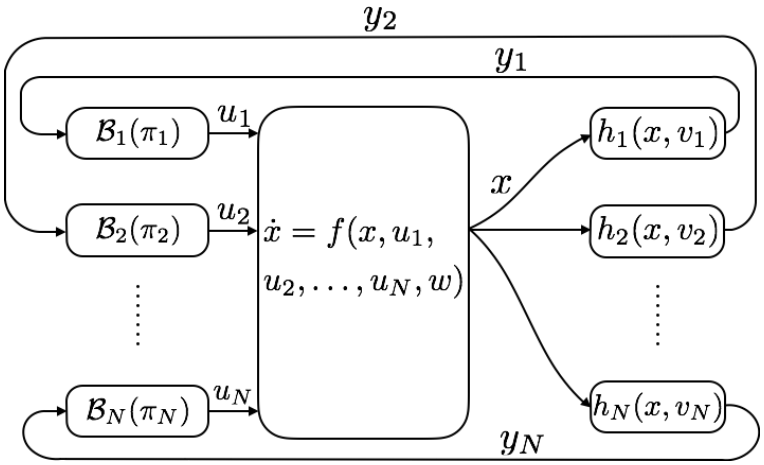



Figure 2.1: The block diagram of the multi-agent system.

### Space-Sharing Interactions and Time-Sharing Interactions

Usually, agent $i$ has its own dynamics, e.g.

$$\dot{x}_i = f_i(x_i, u_i, w_i, e_i), \tag{2.5}$$

where $e_i$ represents external inputs induced by the environment or other agents. When there is no direct contact among agents, $e_i = \emptyset$. Otherwise, $e_i \neq \emptyset$ and it comes in pairs, e.g. if agent $i$ receives an external input from agent $j$, then agent $j$ also receives an external input from agent $i$. The external input pair can be viewed as a constraint between two agents, such as the force and reaction between a human patient and an exoskeleton. When there is no constraint among agents, the system does not need to be synchronized in time. Such interaction is called space-sharing interaction [64]. When there are constraints among agents, the system needs to be synchronized and it is called time-sharing interaction. The block-diagram for the space-sharing system is shown in Fig.2.2.

### Complete Information and Incomplete Information

When there is no learning process, agent $i$'s control input $u_i$ is chosen by minimizing its internal cost function $\mathcal{J}_i(x, u_1, \cdots, u_N)$ given $\pi_i$ and $\mathcal{M}_i$, e.g.

$$u_i = g_i(\pi_i | \mathcal{J}_i, \mathcal{M}_i) = \arg\min_{u_i} E[\mathcal{J}_i(x, u_1, \cdots, u_N) | \pi_i, \mathcal{M}_i], \tag{2.6}$$

where the cost function depends on the system state $x$ and other agents' inputs as variables, and on the data set $\pi_i$ and agent $i$'s internal model $\mathcal{M}_i$ as parameters.

If the system dynamics and agents' internal costs are globally known, i.e. all agents' internal models of the environment are correct, the system is with *complete information*. If

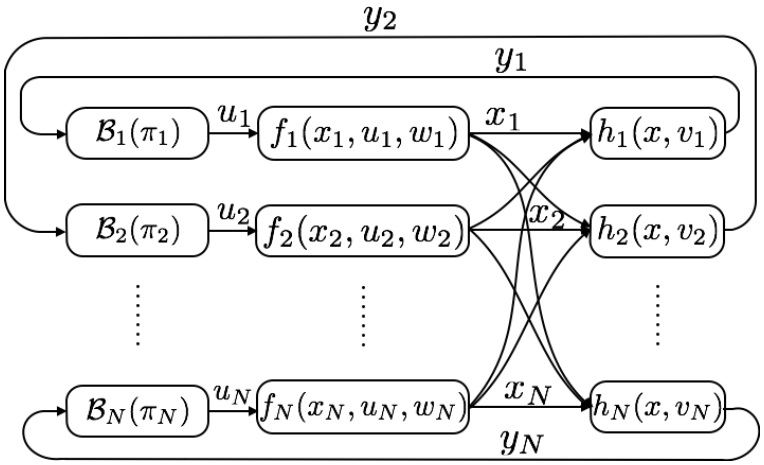



Figure 2.2: The block diagram of the decomposable multi-agent system.

any of such information is unknown to some agents, e.g. some agent's internal model does not match the true model of the environment, the system is with *incomplete information*. If some agent knows information that others don't know, e.g. an agent knows its own internal cost while others don't, it is called *information asymmetry*. Information asymmetry is common in human-robot systems as it is difficult to obtain human's internal cost in advance.

#### Sequential System and Simultaneous System

As discussed in Chapter 1.2, human-robot relationship can either be *parallel* or *hierarchical*, depending on how the agents' decisions are made. The decision hierarchy depends on the information structure of the system [11], e.g. what data is available to which agent when the agents are making decisions in (2.6).

For a parallel relationship, no agent can obtain more data before they make the next move. This is called a *simultaneous game*. For a hierarchical relationship, some agent obtains more data about others before they take the next move. The agents with fewer data are considered as leaders and the agents with more data are considered as followers. For example, in robot skill learning from human demonstration, the human teacher is a leader and the robot learner is a follower. The robot learner only moves after collecting enough human demonstration data, while the human teacher teaches the robot from the very beginning without any robot motion data. A leader is supposed to comprehensively evaluate the consequences of its action by taking into consideration of the follower's reactions. On the other hand, a follower only needs to follow the leader's plan. This is called a *sequential game*.

### 2.1.3 Features of Human-Robot Systems

Although HRI is modeled in a multi-agent system, it is different from conventional multi-agent models [148] both in the design phase and in the execution phase. The distinct features are summarized below:

1. Partial knowledge in the design phase

   The system is only partially known even to the designer in the design phase. It is common for a robot in the execution phase to have partial knowledge since it is beneficial to scale down the problem. However, the successful implementation of those systems depends on the designer's understanding of the overall system in the design phase. As there is no universal model to describe human behavior in the presence of fully automated robots, the system contains many structural or hyper-parametric uncertainties in the design phase. Methods to deal with those uncertainties need to be studied.

2. Time-varying topology in the execution phase

   When a robot is brought to life (e.g. in the execution phase), it may encounter various agents in diverse scenarios. For example, automated vehicles need to interact with different road participants in various traffic conditions, which implies that different

agents will be added or removed in the system model in Fig.2.1. The topology of the multi-agent system is time-varying. In comparison, conventional multi-agent networks such as power networks usually have relatively static topology. The time-varying topology brings large structural uncertainty to the design and analysis of the multi-agent systems.

Regarding the features mentioned above, we are trying to solve the following problems given the model in Fig.2.1.

- How to describe physical systems using the general model?
- How to design the behavior system $\mathcal{B}_i$ for a robot agent $i$ in such a system?
- How to analyze the performance of the designed behavior from the multi-agent system perspective?

The questions will be answered in the following chapters. In the remainder of this chapter, we discuss the architecture of the proposed behavior system for robot agents. Note that human behavior can also be shaped by providing different incentives to shape their internal costs and internal models. Readers are referred to [3, 10] for detailed discussions on how to provide desired incentives.

## 2.2 Agent Behavior Design and Architecture

Behavior design for robots concerns with the three components discussed in Chapter 1.3 and shown in Fig.1.2. In our approach, we concern with the design of $\mathcal{J}$ and $\mathcal{M}$, the design of the logic $g$, and the design of the learning process to update $\mathcal{M}$.

### 2.2.1 Knowledge: The Optimization Problem

For a robot agent $i$ in the multi-agent system, its internal cost is designed with the following structure,

$$\begin{aligned} \mathcal{J}_i(x, u_1, u_2, \cdots, u_N) = \; & J_i(x, u_1, u_2, \cdots, u_N), && \text{(2.7a)} \\ s.t. \quad & u_i \in \Omega, x_i \in \Gamma, \dot{x}_i = f_i(x_i, u_i, w_i, e_i), && \text{(2.7b)} \\ & y_i = h_i(x, v_i), && \text{(2.7c)} \\ & x \in \mathcal{I}, && \text{(2.7d)} \end{aligned}$$

where the right hand side of (2.7a) is the dynamic cost for task performance. It is evaluated over the planning horizon or the look-ahead horizon $T$, which can either be chosen as a fixed number or as a decision variable that should be optimized up to the accomplishment of the task. Equation (2.7b) is the constraint for the agent itself, e.g. constraint on the control input (such as control saturations), constraint on the state (such as joint limits for

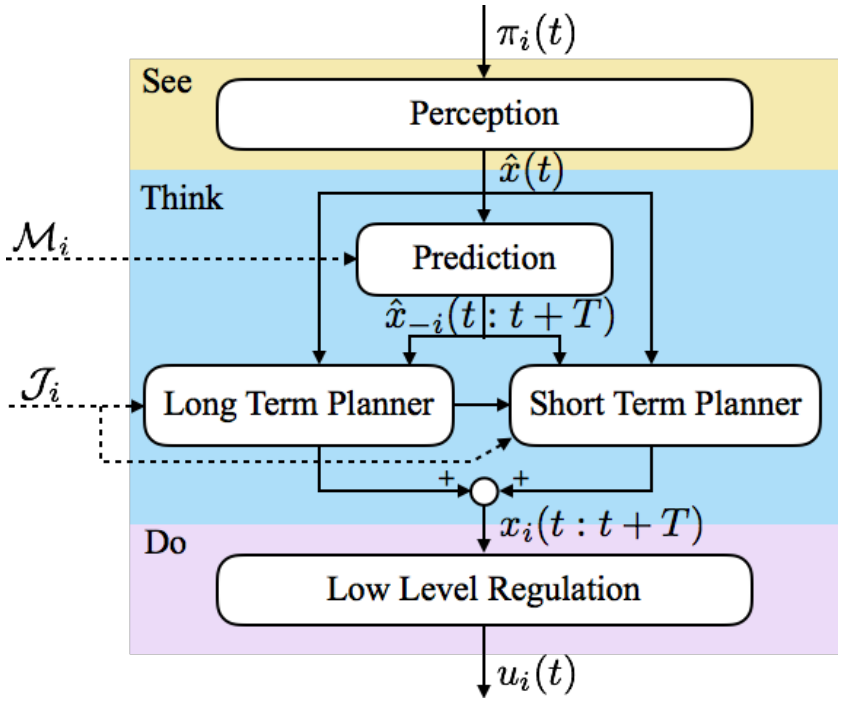



Figure 2.3: The designed architecture of the logic in this dissertation.

robot arms) and the dynamic constraint. Equation (2.7c) is the measurement constraint, which builds the relationship between the state $x$ and the data set $\pi_i$. Equation (2.7d) is the constraint induced by interactions in the multi-agent system, e.g. safety constraint for collision avoidance, where $\mathcal{I} \subset X$ is a subset of the system's state space. Equation (2.7a) can also be called the soft constraint and (2.7b-2.7d) hard constraints. The detailed design of the internal cost will be discussed with respect to specific applications in Chapter 8 and Chapter 9.

The internal model of the environment for agent $i$ is an estimation of the system dynamics and other agents' behaviors, i.e. all blocks that are unknown to agent $i$ in Fig.2.1. However, both the dynamics of the system and other agents' behavior are hard to obtain in the design phase in human-robot systems. They will be identified in the learning process. Moreover, since only the closed loop dynamics of other agents matter, the following function is identified,

$$\dot{x}_{-i} = F_i(x_{-i}, x_i, v_{-i}, w_{-i}), \tag{2.8}$$

where the subscript $-i$ refers to the combination of variables except for agent $i$, e.g. $x_{-i} = [x_1^T, \ldots, x_{i-1}^T, x_{i+1}^T, \cdots, x_N^T, x_e^T]^T$. Hence $\mathcal{M}_i$ is an estimate of $F_i$, e.g. $\mathcal{M}_i = \hat{F}_i$.

### 2.2.2 Logic: The Motion Planning Skill

A "see-think-do" structure is adopted to tackle the complicated optimization problem (2.7). The mission of the "see" step is to process the high dimensional data $\pi_i(t)$ to obtain an estimate of the system states $\hat{x}(t)$. The "think" step is to solve an approximated optimization problem using the estimated states and construct a realizable plan $x_i(t : t + T)$. The "do" step is to realize the plan by generating a control input $u_i(t)$.

In the "think" step, two important modules are *prediction* and *planning*. In the prediction module, agent $i$ makes predictions of other agents $\hat{x}_{-i}(t : t + T)$ based on the internal model $\mathcal{M}_i$. The prediction can be a fixed trajectory or a function depending on agent $i$'s future state $x_i(t : t + T)$. In the planning module, the future movement $x_i(t : t + T)$ is computed given the current state $\hat{x}(t)$, predictions of others' motion $\hat{x}_{-i}(t : t + T)$ as well as the task requirements and constraints encoded in $\mathcal{J}_i$.

As it is computationally expensive to obtain the optimal solution of (2.7) for all scenarios in the design phase, the optimization problem is computed in the execution phase given the obtained information. However, there are two major challenges in real time motion planning. The first challenge is the difficulty in planning a safe and efficient trajectory when there are large uncertainties in other agents' behaviors. As the uncertainty accumulates, solving the problem (2.7) in the long term might make the robot's motion very conservative. The second challenge is real-time computation with limited computation power since problem (2.7) is highly non-convex. We design a unique parallel planning and control architecture to address the first challenge as will be discussed below and develop fast online optimization solvers to address the second challenge as will be discussed in Chapter 4 to Chapter 6.

A toy example is presented in Fig.2.4 to illustrate the first challenge. There is a closed environment. The robot (shown as the purple dot) is required to approach the target (shown as the green dot) while avoiding the human (represented by the pink dot). The time axis is introduced to illustrate the spatiotemporal trajectories. At the first planning step, the robot makes a prediction of the human trajectory and plans a trajectory for itself avoiding the uncertainty cone. As time propagates, the planned trajectory is executed and the human trajectory is observed. In the next planning step, the robot repeats the process, predicts the human trajectory, plans its own trajectory and then executes the planned trajectory. The process is repeated in the following steps. This is the conventional model predictive control method, which is safe, however, too conservative, as the robot hesitates to get close to the human due to the perceived uncertainty.

On the other hand, the uncertainty will not accumulate too much for a short term planner. However, using a short term planner alone will also be problematic. The robot can easily get stuck in local optima and not be able to finish the task as illustrated in Fig.2.5, since the robot does not have a global perspective on how to bypass the obstacle. Although it is possible to construct a globally-converging local policy for a Dubins car[1] when the environment satisfies

[1]A Dubins car is a wheeled robot that runs at a constant speed $V$. It has a simple kinematic model, $\dot{x} = V\cos(\theta)$, $\dot{y} = V\sin\theta$ and $\dot{\theta} = u$ where $x$ and $y$ denote the planar position of the vehicle, $\theta$ the heading, and $u$ the control input.

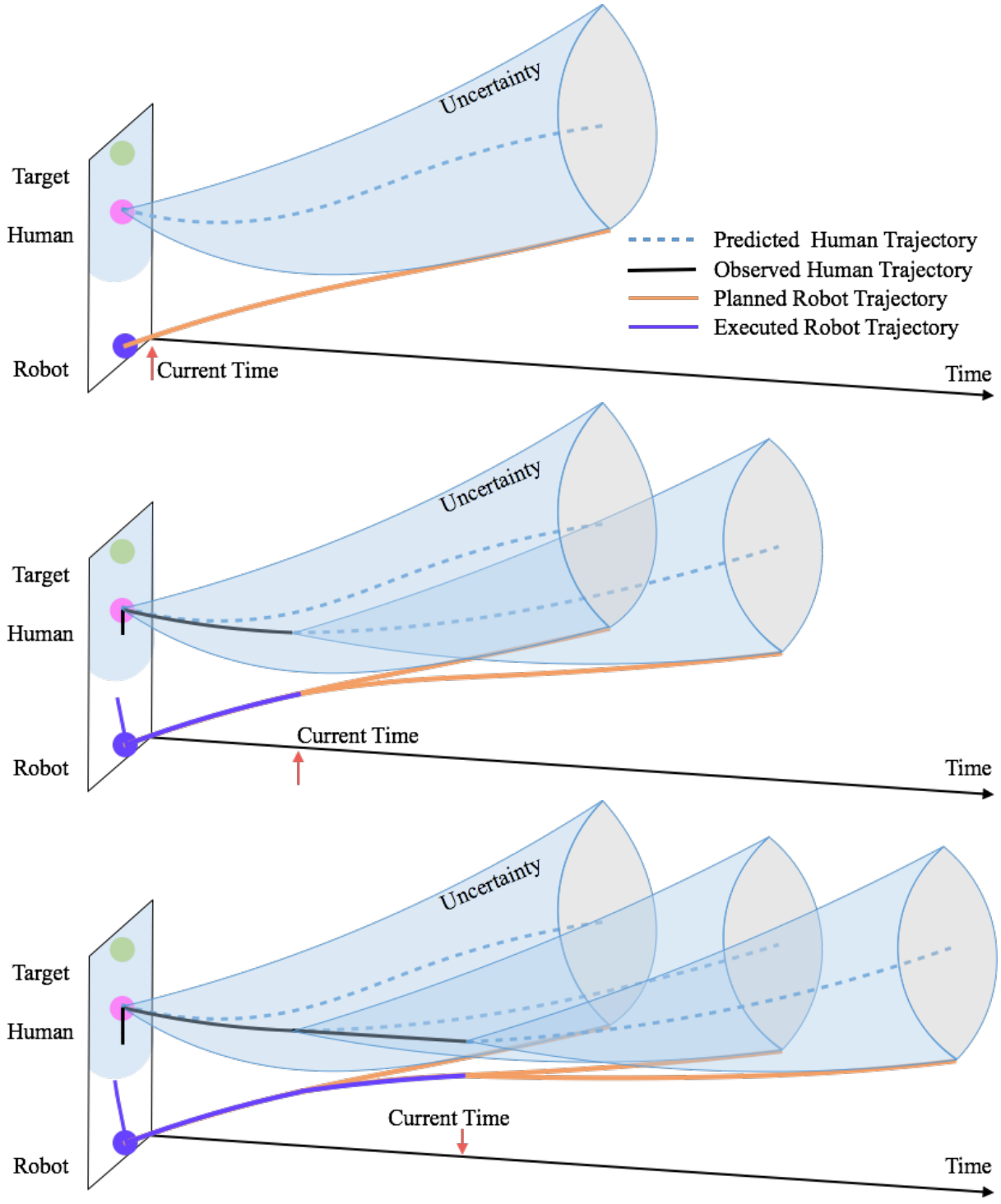



Figure 2.4: Illustration of the accumulation of uncertainty in the long term planning.

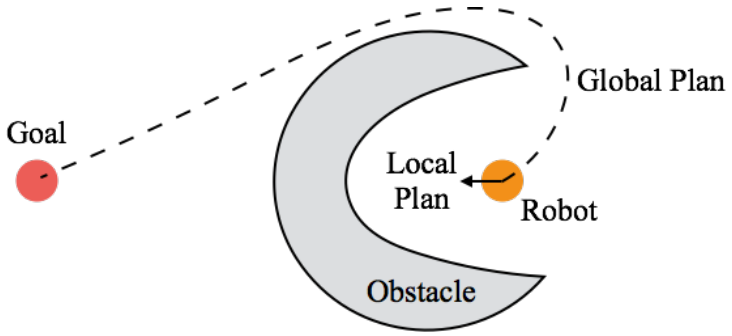



Figure 2.5: Illustration of the local optima problem with the local planner.

certain geometric properties [122], it is in general hard to obtain a globally-converging local policy for robots with complicated dynamics in complicated environments.

A parallel planner which consists of a global (long term) planner as well as a local (short term) planner is adopted in this dissertation to leverage the benefits of the two planners. The idea is to have a long term planner solving (2.7) while considering only rough estimation of human's future trajectory, and have a short term planner addressing uncertainties and enforcing the safety constraint (2.7d). The long term planning is efficiency-oriented and can be understood as deliberate thinking, while the short term planning is safety-oriented and can be understood as a reflex behavior.

The idea is illustrated using the previous example in Fig.2.6. For the long term planning, the robot has a rough estimation of the human's trajectory and it plans a trajectory without considering the uncertainties in the prediction. Then the trajectory is used as a reference in the short term planning. In the first time step, the robot predicts the human motion (with uncertainty estimation) and checks whether it is safe to execute the long term trajectory. As the trajectory does dot intersect with the uncertainty cone, it is executed. In the next time step, since the trajectory is no longer safe, the short term planner modifies the trajectory by detouring. Meanwhile, the long term planner comes with another long term plan and the previous plan is overwrote. The short term planner then monitors the new reference trajectory. The robot follows the trajectory and finally approaches the goal. This approach addresses the uncertainty and is non conservative. As a long term planning module is included, the local optima problem in Fig.2.5 can be avoided.

The block diagram of the parallel controller architecture is shown in Fig.2.3. In the following discussions, we also call the long term module as the efficiency controller and the short term module as the safety controller. The computation time flow is shown in Fig.2.7 together with the planning horizon and the execution horizon. Three long term plans are shown, each with one distinct color. First, plan 1 is computed in the long term planner. The upper part of the time axis shows the planning horizon. The middle layer is the execution horizon. Only the first half of the planned trajectory is executed. The bottom layer shows the computation time. The computation is done before the execution of the plan. Once

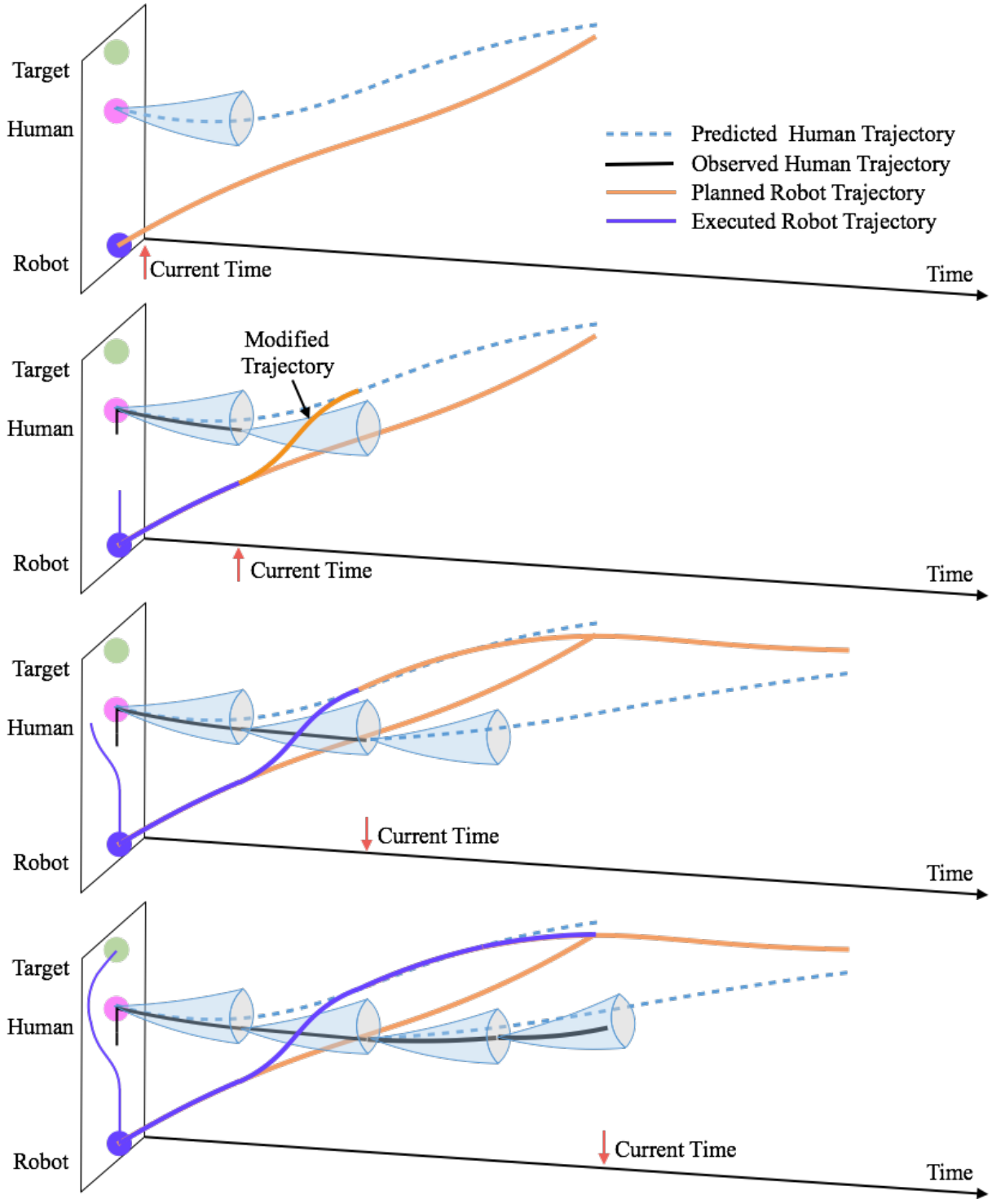



Figure 2.6: Illustration of the performance of the parallel planning.

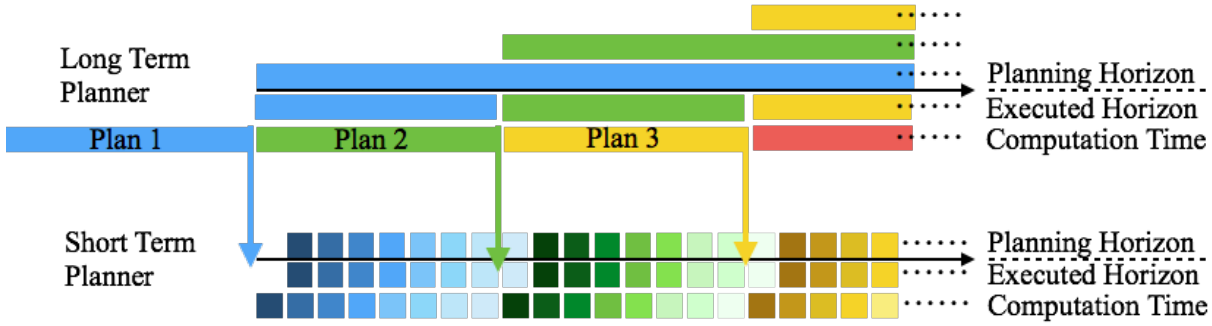



Figure 2.7: The time flow in the parallel planners.

computed, plan 1 is sent to the short term planner for monitoring. The sampling rate in the short term planner is much higher than that in the long term planner, since the computation time in the short term planner is much smaller than that in the long term planner. The mechanism in the short term planner is similar to that in the long term planner. Note that the planning horizon in the short term planner is not necessarily one time step, though the execution horizon is one time step. While the short term planner is monitoring the trajectory, the long term planner is computing a new long term trajectory. Once the new trajectory is computed, it will be sent to the short term planner for monitoring. The long term planner then computes another trajectory and so on.

This approach can be regarded as a two-layer MPC approach. Coordination between the two layers is important. To avoid instability, a margin is needed in the safety constraint in the long term planner so that the long term plan will not be revoked by the short term planner if the long term prediction of the human motion is correct. Theoretical analysis of the stability of the two-layer MPC method is out of the scope of this dissertation, and is left as a topic for future study.

Nonetheless, the successful implementation of the parallel control architecture highly depends on computation. It is important that the optimization algorithm finds a feasible and safe trajectory within the sampling time. The algorithms will be discussed in Chapter 4 to Chapter 6.

### 2.2.3 Learning Process: The Cognition Skill

The internal model $\mathcal{M}_i$ needs to be updated online, which is a cognition skill. As discussed in Chapter 1.3, there are many ways to describe agent behaviors. Among those methods, a reactive model (2.8) is adopted in this dissertation. The advantage of this model is that it directly approximates the optimal response of other agents. Hence no optimization is needed for motion prediction. The reactive model depends on other agents's internal costs and internal models, hence may be multi-modal. The learning is divided into two parts: goal inference and model adaptation. For example, in a factory environment, goal inference tells

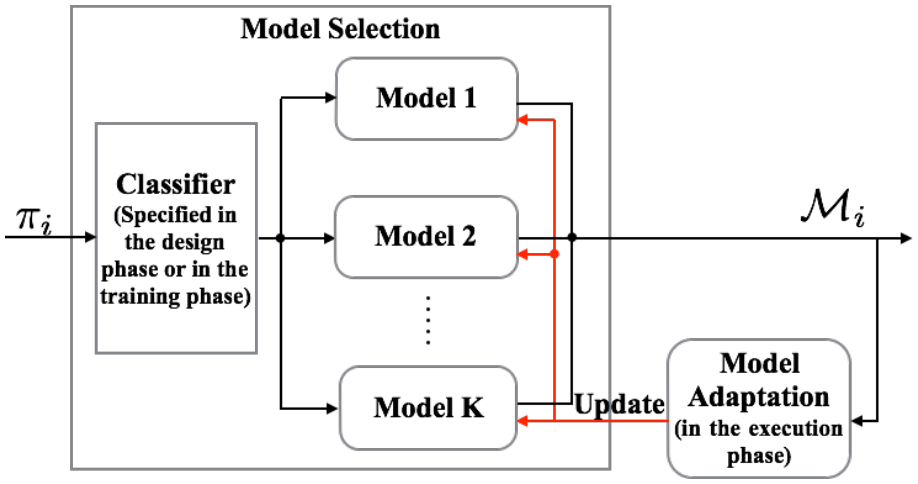



Figure 2.8: Cognition: the designed learning process in this dissertation.

whether a human worker is going to sit down, stand up, walk to a bench, or pick up a work piece. Indeed, goal inference identifies the context behind the observed motion. Mathematically, it can be understood as model selection. Given the goal (selected model), different agents may have different ways to achieve the goal. For example, a cautious pedestrian may wait for all the vehicles to pass before he crosses the street, while another pedestrian may cross the street assuming that all the vehicles will yield. Hence model adaptation is needed to account for individual differences. Another reason for model adaptation is to account for time-varying behaviors, since other agents are also adapting their behaviors to the environment. We can either build the structure of the models offline during the training phase or identify the models from sketch online. The advantage of such structure is that the prior knowledge in the design phase or in the training phase can be easily incorporated. The architecture of the cognition system is shown in Fig.2.8. The methods will be further explained in Chapter 3.

## 2.3 Conclusion

In this chapter, we modeled the human-robot interactions in a multi-agent framework. The considerations in robot behavior design were discussed and will be further elaborated in the following chapters. The application of the method will be discussed in Chapter 8 and Chapter 9.

# Chapter 3

# Cognition: Understanding Others' Behaviors

## 3.1 Overview

Cognition is a skill to identify the dynamic model of the environment, especially the closed loop dynamics of human agents. It is a part of the learning process and can be performed either in the training phase (offline) or in the execution phase (online). Nonetheless, the learning algorithms should be specified in the design phase. This chapter discusses the algorithms according to the structure described in Chapter 2.2 and shown in Fig.2.8.

Reactive models are used to describe the behavior of other agents. Denote the state of the robot that we design for as $x_R$ and the state of the other agents as $x_H$. Ignoring the noise term, (2.8) can be rewritten in discrete time as

$$x_H(k+1) = F(x(k)). \tag{3.1}$$

There are many methods to identify (3.1). For example, Gaussian Mixture Model (GMM) [96] or Neural Networks (NN) [116]. The structure in Fig.2.8 is adopted, which is equivalent to the following equations,

$$\begin{aligned} x_H(k+1) &= F_{\delta(k)}(x(k)), & (3.2)\\ F_m(x(k)) &= \sum_j \theta_{m,j} f_{m,j}(x(k)), & (3.3) \end{aligned}$$

where $\delta(k) \in \{1, \ldots, M\}$ is a discrete classification function. $M$ is the number of models. $F_m$ is the closed loop dynamics in model $m$, which is a linear combination of several features $f_{m,j}(x)$. $\theta_{m,j} \in \mathbb{R}$ are coefficients that can be adapted online. Mathematically, $F_m$ can be understood as a group of basis for function $F$. The features should either be designed or identified in the training phase.

In the remainder of this chapter, the method for model selection will be discussed in Section 3.1, and the method for model adaptation in Section 3.2.

## 3.2 Classification of the Behaviors

### 3.2.1 Dynamic Classification using Hidden Markov Model

Hidden Markov Model (HMM) [23] is chosen for model selection where the classification function $\delta(k)$ is regarded as the hidden variable that needs to be inferred from the observation $\pi_R(k)$. There are two important relationships in a HMM: (1) how the current classification is affected by previous classification (or how the intentions of agents change dynamically) and (2) how the current classification is affected by current observation (or how the intentions of agents are revealed in data). The first is a dynamic model, which encodes the following relationship

$$P(\delta(k) = m|\delta(k-1) = n). \tag{3.4}$$

The second is a measurement model, which encodes the following relationship

$$P(y_R(k)|\delta(k) = m). \tag{3.5}$$

Then the estimate of $\delta(k)$ given $\pi_R(k)$ is

$$\begin{aligned} & P(\delta(k) = m|\pi_R(k)), \\ \propto\ & P(\delta(k) = m, y_R(1), \cdots, y_R(k)), \\ \propto\ & P(y_R(k)|\delta(k) = m)P(\delta(k) = m|\pi_R(k-1)), \\ \propto\ & P(y_R(k)|\delta(k) = m)\sum_n P(\delta(k) = m|\delta(k-1) = n)P(\delta(k-1) = n|\pi_R(k-1)), \end{aligned} \tag{3.6}$$

which depends on the estimate of $\delta(k-1)$ in the last time step, the dynamic model (3.4) and the measurement model (3.5). Then the value of $\delta(k)$ is chosen according to maximum likelihood.

The dynamic model (3.4) and the measurement model (3.5) can either be designed in the design phase or be learned in the training phase. The training data can be obtained through: 1) real world experiment; 2) virtual reality-based simulation as discussed in Appendix A.

### 3.2.2 Example: Behavior Classification of Surrounding Vehicles

Denote the intended behavior of a surrounding vehicle $i$ at time step $k$ as $b_i(k)$. In this example, three behaviors are considered:

- Behavior 1 ($B_1$): Lane following;
- Behavior 2 ($B_2$): Lane changing to the left;
- Behavior 3 ($B_3$): Lane changing to the right;

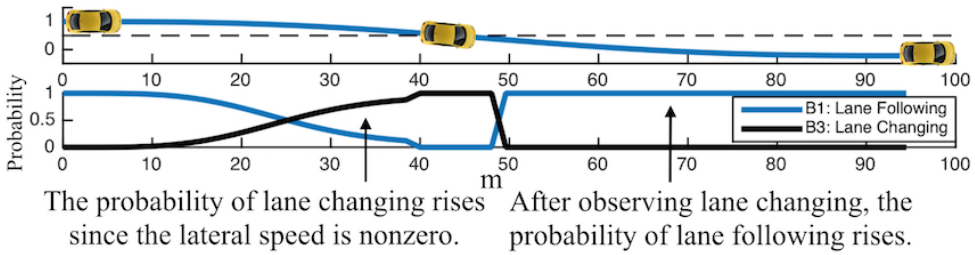



Figure 3.1: Behavior classification for a vehicle in a two-lane case.

where $B_1$ is the steady state behavior; $B_2$, $B_3$ are driving maneuvers.

The transition model is designed to be

$$A = \begin{bmatrix} 0.6 & 0.5 & 0.5 \\ 0.2 & 0.5 & 0 \\ 0.2 & 0 & 0.5 \end{bmatrix}, \tag{3.7}$$

where $A_{mn} := P(\delta(k) = m|\delta(k-1) = n)$.

The measurement model for a vehicle is designed according to the following the three features: (1) its lateral velocity $v_{lat}$, (2) its lateral deviation from the center of its current lane $d$ ($d > 0$ if the deviation is to the left), and (3) an indicator $c$ whether the vehicle is crossing the boundary of two lanes ($c = 1$ if true). Then the measurement model is

$$P(y_R(k)|\delta(k) = 1) = (1-c)e^{-(v_{lat}+d)^2}, \tag{3.8}$$
$$P(y_R(k)|\delta(k) = 2) = cI(d > 0)(1 - e^{-(v_{lat}+d)^2}), \tag{3.9}$$
$$P(y_R(k)|\delta(k) = 3) = cI(d < 0)(1 - e^{-(v_{lat}+d)^2}), \tag{3.10}$$

Given the transition model and measurement model, the distribution of $\delta(k)$ follows from (3.6). The simulation result is shown in Fig.3.1, which correctly predicts the lane change behavior before the yellow vehicle crossed the lane boundary. This classification model will be used in Chapter 8.

## 3.3 Adaptation of the Behavior Model

### 3.3.1 Online Learning Using Parameter Adaptation

For a selected model $m$, the learning objective is to approximate the function $F_m(x(k))$ such that the estimate $\hat{x}_H(k+1|k) = \hat{F}_m(x(k))$ minimizes the expected prediction error, e.g.

$$\hat{x}_H(k+1|k) = \arg\min_a E_{x_H(k+1)}(\|x_H(k+1) - a\|^2). \tag{3.11}$$

The notations for estimates of $x_H$ are listed in Table 3.1.

The robot's measurement of the human is denoted as $y_R^H$, which satisfies that

$$y_R^H(k) = x_H(k) + v_R^H(k). \tag{3.12}$$

where $v_R^H(k)$ is measurement noise.

Rewriting $F_m(x(k))$ in (3.3) and ignore $m$, we have the following dynamics

$$x_H(k+1) = \Phi(x(k))\vartheta^T + w(k). \tag{3.13}$$

where $\Phi(\cdot) = [f_1(\cdot), \cdots, f_j(\cdot), \cdots]$, $\vartheta = [\theta_1, \cdots, \theta_j, \cdots]$ and $w$ is a noise term assumed to be zero-mean, white and with covariance $W$.

Equations (3.13) and (3.12) form a nonlinear Gaussian system with unknown parameters. A recursive least square parameter adaptation algorithm (RLS-PAA) [44] is developed to identify the system online such that the prediction $\hat{x}_H(k+1|k)$ minimizes the expected prediction error (3.11). Define $\hat{\vartheta}(k)$ to be the estimates of the coefficients given the information up to the $k$-th time step.

- State Estimation

  At $k+1$-th time step, $\hat{x}_H$ is first updated according to the closed loop dynamics in (3.14). Then the measurement information is incorporated in the *a posteriori* estimate in (3.15). A constant update gain $\alpha \in (0,1)$ is chosen to ensure that the measurement information is always incorporated.

$$\hat{x}_H(k+1|k) = \Phi(\hat{x}(k|k))\hat{\vartheta}(k)^T, \tag{3.14}$$
$$\hat{x}_H(k+1|k+1) = (1-\alpha)\hat{x}_H(k+1|k) + \alpha y_R^H(k+1), \tag{3.15}$$

  where $\hat{x}(k|k) = [\hat{x}_H(k|k)^T, \hat{x}_R(k|k)^T]^T$, and $\hat{x}_R(k|k)$ is the estimate of the state of the robot itself, which can be obtained using Kalman Filter.

- Parameter Estimation

  The coefficients are estimated using RLS-PAA:

$$\hat{\vartheta}(k+1) = \hat{\vartheta}(k) + \left(\hat{x}_H(k+1|k+1) - \hat{x}_H(k+1|k)\right)^T \Phi(\hat{x}(k|k))F(k+1), \tag{3.16}$$

  where $F$ is the learning gain such that

$$F(k+1) = \frac{1}{\lambda}\left[F(k) - \frac{F(k)\,\Phi(\hat{x}(k|k))^T\Phi(\hat{x}(k|k))F(k)}{\lambda + \Phi(\hat{x}(k|k))F(k)\Phi(\hat{x}(k|k))}\right], \tag{3.17}$$

  where $\lambda \in (0,1)$ is a forgetting factor.

Table 3.1: The notations in state estimation and prediction.

| | State Estimate | Estimation Error | MSEE |
|---|---|---|---|
| *a posteriori* | $\hat{x}_H(k\|k)$ | $\tilde{x}_H(k\|k) = x_j(k) - \hat{x}_j(k\|k)$ | $X_H(k\|k)$ |
| *a priori* | $\hat{x}_H(k+1\|k)$ | $\tilde{x}_H(k+1\|k) = x_j(k+1) - \hat{x}_j(k+1\|k)$ | $X_H(k+1\|k)$ |

### 3.3.2 Quantifying the Uncertainty

The motions of other agents can be predicted using the estimate $\hat{\vartheta}$. However, we also need to quantify the uncertainty associated with the prediction, e.g. the mean square estimation error (MSEE),

$$X_H(k+h|k) = E_{x_H(k+h)}(\|x_H(k+h) - \hat{x}_H(k+h|k)\|^2). \tag{3.18}$$

Assume that the human state can be directly measured, i.e. $v_R^H(k) = 0$. Let $\tilde{\vartheta}(k) = \vartheta(k) - \hat{\vartheta}(k)$ be the estimation error. The *a priori* state estimation error is

$$\tilde{x}(k+1|k) = \Phi(x(k))\,\tilde{\vartheta}(k)^T + w(k). \tag{3.19}$$

Since $\hat{\vartheta}(k)$ only contains information up to the $(k-1)$-th time step, $\tilde{\vartheta}(k)$ is independent of $w(k)$. Thus the *a priori* MSEE is

$$X_{\tilde{x}\tilde{x}}(k+1|k) = E\left[\tilde{x}(k+1|k)\,\tilde{x}(k+1|k)^T\right] = \Phi(k)\,X_{\tilde{\vartheta}\tilde{\vartheta}}(k)\,\Phi^T(k) + W, \tag{3.20}$$

where $X_{\tilde{\vartheta}\tilde{\vartheta}}(k) = E\left[\tilde{\vartheta}(k)^T\,\tilde{\vartheta}(k)\right]$ is the mean squared error of the parameter estimation. The parameter estimation error is

$$\tilde{\vartheta}(k+1) = \tilde{\vartheta}(k) - F(k+1)\,\Phi^T(k)\,\tilde{x}(k+1|k) + \Delta\vartheta(k), \tag{3.21}$$

where $\Delta\vartheta(k) = \vartheta(k+1) - \vartheta(k)$. Since the system is time varying, the estimated parameter is biased and the expectation of the error can be expressed as

$$\begin{aligned} E\left(\tilde{\vartheta}(k+1)\right) &= \left[I - F(k+1)\,\Phi^T(k)\,\Phi(k)\right] E\left(\tilde{\vartheta}(k)\right) + \Delta\vartheta(k), \\ &= \sum_{n=0}^{k}\prod_{i=n+1}^{k}\left[I - F(i+1)\,\Phi^T(i)\,\Phi(i)\right]\Delta\vartheta(n). \end{aligned} \tag{3.22}$$

The mean squared error of parameter estimation follows from (3.21) and (3.22):

$$\begin{aligned} & X_{\tilde{\vartheta}\tilde{\vartheta}}(k+1) \\ = \;& F(k+1)\,\Phi^T(k)\,X_{\tilde{x}\tilde{x}}(k+1|k)\,\Phi(k)\,F(k+1) \\ & -X_{\tilde{\vartheta}\tilde{\vartheta}}(k)\,\Phi^T(k)\,\Phi(k)\,F(k+1) - F(k+1)\,\Phi^T(k)\,\Phi(k)\,X_{\tilde{\vartheta}\tilde{\vartheta}}(k) \\ & +E\left[\tilde{\vartheta}(k+1)\right]\Delta\vartheta^T(k) + \Delta\vartheta(k)\,E\left[\tilde{\vartheta}(k+1)\right]^T - \Delta\vartheta(k)\,\Delta\vartheta(k)^T + X_{\tilde{\vartheta}\tilde{\vartheta}}(k). \end{aligned} \tag{3.23}$$

Since $\Delta\vartheta(k)$ is unknown in (3.22) and (3.23), it is set to an average time varying rate $d\vartheta$ in the implementation.

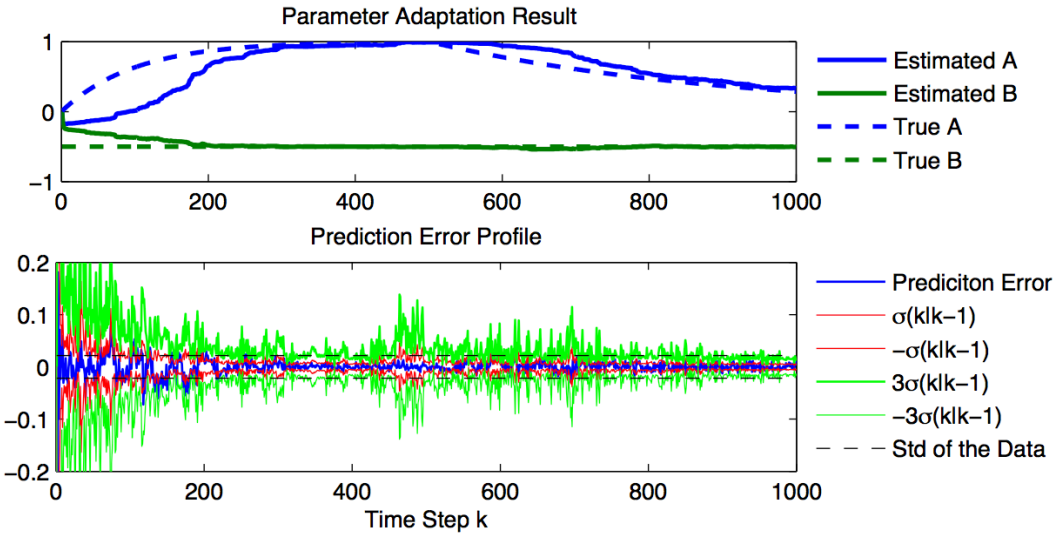



Figure 3.2: Identification of a linear time varying system.

### 3.3.3 Example: Identification of a Linear Time Varying System

In this example, we illustrate the performance of the algorithms in a linear scalar system

$$x(k+1) = A(k)x(k) + B(k)u(k) + w(k), \tag{3.24}$$

where $x$, $u$, $A$ and $B$ are all scalars. $A$ and $B$ are unknown coefficients that need to be identified. $B$ is constant, while $A$ is time varying. $x(k)$ and $u(k)$ are linear features that is directly measurable.

Figure 3.2 shows the simulation result of the proposed learning algorithm on a first order system with a noise covariance $W = 0.005^2$. A forgetting factor $\lambda = 0.98$ is used. The solid and dashed blue lines in the upper figure are $\hat{A}(k)$ and $A(k)$, while the solid and dashed green lines are $\hat{B}(k)$ and the constant parameter $B$ respectively. As shown in the figure, the time varying parameter $A(k)$ is well approximated by $\hat{A}(k)$, while $\hat{B}(k)$ converges to $B$. In the lower figure, the blue curve is the one step prediction error $\tilde{x}(k|k-1)$. The green curves are the $3\sigma$ bound ($\sigma = \sqrt{X_{\tilde{x}\tilde{x}}(k|k-1)}$). The black dashed line is the statistical standard deviation (Std) of the data $\tilde{x}(k|k-1)$ from $k = 1$ to $k = 1000$. As shown in the figure, the $3\sigma$ value offers a good bound for the prediction errors as all measured errors lie between the green curves. Moreover, the MSEE is larger when the parameter is changing faster, which captures the time varying property of the system. On the other hand, the statistical standard deviation does not give a good description of the data in real time.

Note that although the method works well in practice, it is possible that it goes unstable for complicated dynamics. Theoretical analysis will be performed in the future.

## 3.4 Conclusion

This chapter discussed a HMM-based classification algorithm for model selection and a RLS-PAA-based adaptation algorithm for model adaptation in order to identify the models to describe other agents' behaviors. The identified models will be used in predicting other agents' behaviors, which belongs to the "think" function in the system architecture in Fig.2.3.

# Chapter 4

# Safety-Oriented Local Motion Planning

## 4.1 Overview

Human safety is one of the biggest concerns in HRI [132]. Two different approaches can be used to address the safety issues. One way is to increase the intrinsic safety level of the robot through hardware design or low level control, so that even if collision happens, the impact on the human is minimized [50]. The other way is to let the robot behave safely, which is called "interactive safety" as opposed to the intrinsic safety [139]. In this chapter, interactive safety will be addressed in the context of local motion planning and control.

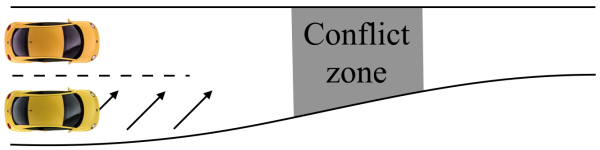
Conventional approach to address the interactive safety is conservative, which slows down the robot when human is nearby, hence sacrifices productivity for the sake of safety. However, to make the interaction desirable, both safety and efficiency need to be considered in designing the robot behavior. Less conservative methods in the context of obstacle avoidance in robot motion planning have also been used to address safety in HRI, such as potential field methods [61, 108] and sliding mode methods [45]. These two methods result in closed-form analytical control laws; but they do not emphasize optimality (or efficiency) and sometimes may not enforce hard constraints. Some authors formulate the problem as an optimization or optimal control problem with hard constraints to represent the safety requirements [34]. Unfortunately, these non-convex optimization problems are generally hard to solve analytically [54] and different approximations and numerical methods are used [123, 127]. As the algorithms must be designed such that it can be executed fast enough for timely responses in emergency situations, those methods may not be desirable for online applications. New approaches to deal with safety in HRI are needed, e.g. the robot should be designed intelligent enough to conduct social behavior [153] to interact with humans safely and efficiently even in emergency situations.

By designing the robot behavior, the safety issues need to be understood in the human-robot systems. This multi-agent situation complicates the interactions as the individual

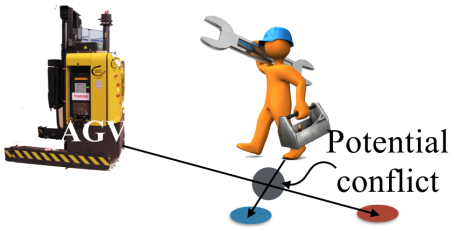
optima usually do not coincide with the system optima [91]. An agent's interest is to be efficient, i.e. finish the task as soon as possible, while staying safe. For example, an automated vehicle's interest is to go to the destination in minimum time without colliding with surrounding vehicles. The safety is the mutual interests for all agents in the system, while the efficiency goals may conflict with one another. Figure 4.1a shows the conflict between two vehicles during lane merging. The two vehicles cannot pass the conflict zone at the same time. One of the vehicle must yield (i.e. sacrifice efficiency) to ensure safety. Figure 4.1b shows the conflict between an autonomous guided vehicle (AGV) and a human worker as their future trajectories intersect. The intersection point is a potential conflict zone if neither the AGV or the human detours.

Take human's social behavior as an example, to solve conflicts of interests, a human's behavior is usually constrained. The constraint comes from at least two factors: the social norm and the uncertainties that he or she perceives for other people. The social norm guides human's behavior in a certain situation or environment as "mental representations of appropriate behavior". For example, in the AGV case, the social norm will suggest the AGV to keep a safe distance from human and detour if necessary. In practice, the perceived uncertainties also affect human's decision. Similarly, whether the AGV needs to detour and how much the AGV should detour highly depends on how certain the AGV is about its prediction of the human's trajectory. Moreover, the uncertainties can be attenuated through active learning (refer to uncertainty reduction theory [15]). A common experience is: a newcomer tends to behave conservatively in a new environment due to large uncertainties. But through observing and learning his peers, he will gradually behave freely due to the reduction of uncertainties.

Based on these observations, a safety oriented method is introduced to design the short




(a) Conflict between two vehicles during lane merging.




(b) Conflict between a human worker and an autonomous guided vehicle (AGV) in a factory floor.

Figure 4.1: Conflicts in multi-agent systems and safety issues in human-robot interactions.

term motion planner for the robot. As discussed in Chapter 2, local planning is mainly to address safety. Suppose a reference input $u_R^o$ is received from the long term planner, the short term planner needs to ensure that the interaction constraint $x \in \mathcal{I}$ will be satisfied after applying this input. Hence the problem can be formulated as the following optimization,

$$u_R^* = \min \ \|u_R - u_R^o\|_Q^2 \tag{4.1a}$$

$$s.t.\ u_R \in \Omega, x \in \mathcal{I}, \dot{x}_R = f_R(x_R) + h_R(x_R)u_R, \tag{4.1b}$$

where $\|u_R - u_R^o\|_Q = (u_R - u_R^o)^T Q(u_R - u_R^o)$ penalizes the deviation from the reference input. The last equation in (4.1b) is the robot dynamic function, which is assumed to be affine in the control input. The interaction constraint $x \in \mathcal{I}$ defines a safe set in the system state space. In this chapter, we define $X_S := \mathcal{I}$ and assume that all agents agree on this constraint, which can be understood as a social norm. The safe set and the robot dynamics impose nonlinear and non-convex constraints which make the problem hard to solve. We propose to transform the non-convex state space constraint into convex control space constraint using the idea of invariant set. The safe set algorithm (SSA) [78] is developed to enforce invariance in the safe set according to the predicted human behavior, and the safe exploration algorithm (SEA) further constrains the robot motion by uncertainties in the predictions [83]. By actively learning human behaviors, the robot will be more "confident" about its prediction of human motion, hence able to access a larger subset of the safe set when the uncertainty is smaller.

The remainder of the chapter is organized as follows: in Section 4.2, the design methodology of the safety-oriented local motion planning will be discussed. The safe set algorithm (SSA) and the safe exploration algorithm (SEA) will be discussed in Section 4.3 and Section 4.4 respectively. A method to combine SSA and SEA will be discussed in Section 4.5. Section 4.6 concludes the chapter.

## 4.2 The Safety-Oriented Behavior Design

**The Safety Principle**

According to the safe set $X_S$, define the state space constraint $R_S$ for the robot as $R_S(x_H) = \{x_R : [x_R^T, x_H^T]^T \in X_S\}$, which depends on humans' states. If human will take care of the safety, then the safe set for the robot is

$$R_S^1 = \{x_R : x_R \in R_S(x_H) \text{ for some } x_H\}. \tag{4.2}$$

However, to make the system reliable, the safety problem should be taken care of by the robot. In the case that the robot knows human's next move $\hat{x}_H$, the safety bound for the robot becomes

$$R_S^2 = \{x_R : x_R \in R_S(\hat{x}_H)\}. \tag{4.3}$$

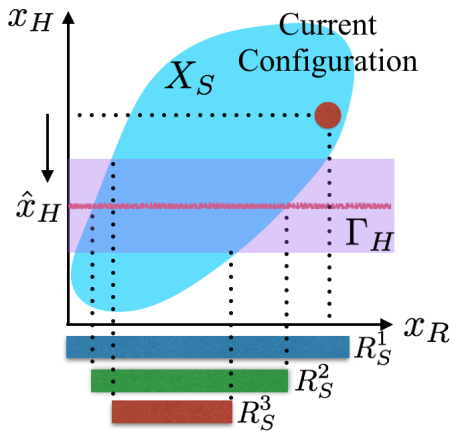



Figure 4.2: Illustration of the state space safety constraints $X_S$, $R_S^1$, $R_S^2$ and $R_S^3$.

Due to noises and uncertainties, the estimate $\hat{x}_H$ may not be accurate. The human state $x_H$ may lie in a set $\Gamma_H$ containing $\hat{x}_H$. Then the robot motion should be constrained in a smaller set

$$R_S^3 = \{x_R : x_R \in R_S(x_H), \forall x_H \in \Gamma_H\}. \tag{4.4}$$

Figure 4.2 illustrates the safe set $X_S$ and the state space constraints $R_S^1$, $R_S^2$ and $R_S^3$. It is clear that $R_S^3 \subset R_S^2 \subset R_S^1$.

**The Safety Principle**: the robot control input $u_R(t)$ should be chosen such that $X_S$ is invariant, i.e. $x(t) \in X_S$ for all $t$, or equivalently, $x_R(t) \in R_S^3(t)$ for $\Gamma_H(t)$ which accounts for almost all possible noises $v_1, ...v_N, w_1, ..., w_N$ and human behaviors $\mathcal{B}_i(\cdot), i \in H$ (those with negligible probabilities will be ignored).

Figure 4.3 illustrates the expected outcome of the robot behavior under the safety principle. In view of the potential conflict, the robot detours in the safe region $R_S^3$.

### The Safety Index

The safety principle requires the designed control input to make the safe set invariant with respect to time. In addition to constraining the motion in the safe region $R_S^3$, the robot should also be able to cope with any unsafe human movement. Given the current configuration in Fig.4.2, if the human is anticipated to move downwards, the robot should go left in order for the combined trajectory to stay in the safe set. To cope with the safety issue dynamically, a safety index is introduced as shown in Fig.4.4. The safety index $\phi : X \to \mathbb{R}$ is a function on the system state space such that

1. $\phi$ is differentiable with respect to $t$, i.e. $\dot{\phi} = (\partial\phi/\partial x)\dot{x}$ exists everywhere;
2. $\partial\dot{\phi}/\partial u_R \neq 0$;

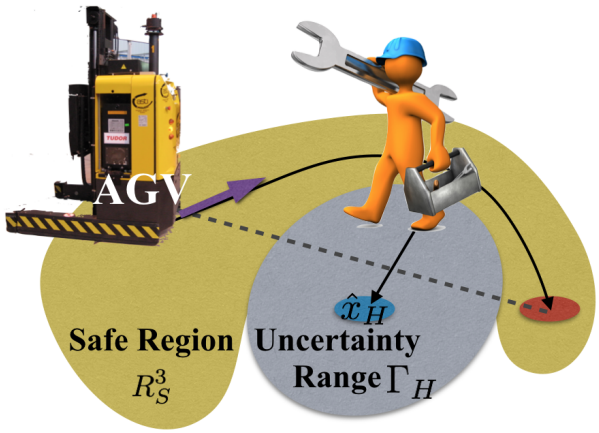



Figure 4.3: Solving the conflicts by re-planning in the safe region $R_S^3$.

3. The unsafe set $X \setminus X_S$ is not reachable given the control law $\dot{\phi} < 0$ *when* $\phi \geq 0$ and the initial condition $x(t_0) \in X_S$.

The first condition is to ensure that $\phi$ is smooth. The second condition is to ensure that the robot input can always affect the safety index. The third condition provides a criteria to determine whether a control input is safe or not, e.g. all the control inputs that drive the state below the level set 0 are safe and unsafe otherwise.

**Lemma: (Existence of the Safety Index)** A function $\phi$ satisfying all three conditions exists for any set $X_S$ that can be represented by a smooth function $\phi_0(x)$, i.e. $X_S = \{x : \phi_0(x) \leq 0\}$.[1]

To ensure safety, the robot's control must be chosen from the set of safe control $U_S(t) = \{u_R(t) : \dot{\phi} \leq -\eta_R \; when \; \phi \geq 0\}$ where $\eta_R \in \mathbb{R}^+$ is a safety margin. By the dynamic equation in (4.1b), the derivative of the safety index can be written as

$$\dot{\phi} = \frac{\partial \phi}{\partial x_R} h_R u_R + \frac{\partial \phi}{\partial x_R} f_R + \sum_{j \in H} \frac{\partial \phi}{\partial x_j} \dot{x}_j. \tag{4.5}$$

Then the set of safe control is

$$U_S(t) = \{u_R(t) : L(t)\, u_R(t) \leq S(t, \dot{x}_H)\}, \tag{4.6}$$

[1]The Lemma is proved in [78]. $\phi$ can be constructed in the following procedure: first, check the order from $\phi_0$ to $u_R$ in the Lie derivative sense, denote it by $n$; then define $\phi$ as $\phi_0 + k_1\dot{\phi}_0 + ... + k_{n-1}\phi_0^{(n-1)}$. The coefficients $k_1, ..., k_n$ are chosen such that the roots of $1 + k_1 s + ... + k_{n-1}s^{n-1} = 0$ all lie on the negative real line.

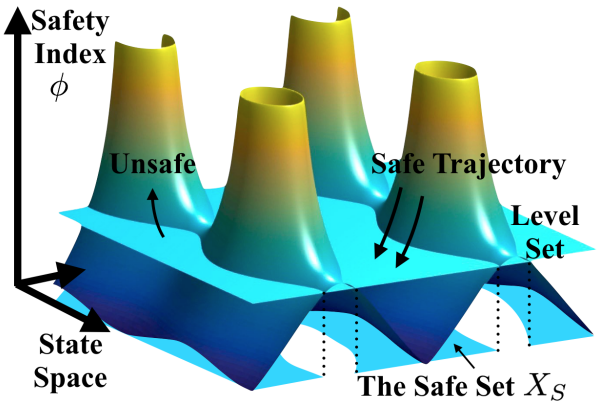



Figure 4.4: The safety index and the safe set.

where

$$L(t) = \frac{\partial \phi}{\partial x_R} h_R, \tag{4.7}$$

$$S(t, \dot{x}_H) = \begin{cases} -\eta_R - \sum_{j \in H} \frac{\partial \phi}{\partial x_j} \dot{x}_j - \frac{\partial \phi}{\partial x_R} f_R & \phi \geq 0 \\ \infty & \phi < 0 \end{cases}. \tag{4.8}$$

$L(t)$ is a vector at the "safe" direction, while $S(t, \dot{x}_H)$ is a scalar indicating how much control effort is needed to be safe, which can be broken down into three parts: a margin $-\eta_R$, a term to compensate human motion $-\sum_{j \in H} \frac{\partial \phi}{\partial x_j} \dot{x}_j$ and a term to compensate the inertia of the robot itself $-\frac{\partial \phi}{\partial x_R} f_R$. In the following arguments when there is no ambiguity, $S(t, \dot{x}_H)$ denotes the value in the case $\phi \geq 0$ only. Under different assumptions of the human behavior, $S(t)$ varies. The sets of safe control correspond to $R_S^1$, $R_S^2$ and $R_S^3$ are

$$U_S^1(t) = \{u_R(t) : L(t) u_R(t) \leq S(t, \dot{x}_H) \text{ for some } \dot{x}_H\}, \tag{4.9}$$

$$U_S^2(t) = \left\{u_R(t) : L(t) u_R(t) \leq S\left(t, \hat{\dot{x}}_H\right)\right\}, \tag{4.10}$$

$$U_S^3(t) = \left\{u_R(t) : L(t) u_R(t) \leq S(t, \dot{x}_H) \text{ for all } \dot{x}_H \in \dot{\Gamma}_H\right\}, \tag{4.11}$$

where $\hat{\dot{x}}_H$ is the velocity vector that moves the current configuration $x_H$ of human to $\hat{x}_H$ and $\dot{\Gamma}_H$ is the set of velocity vectors that move $x_H$ to $\Gamma_H$. Computationally, $\hat{\dot{x}}_H = \frac{\hat{x}_H - x_H}{t_s}$ where $t_s$ is the sampling time. Obviously $U_S^3 \subset U_S^2 \subset U_S^1$. When the uncertainties in the estimation of $\hat{\dot{x}}_H$ reduces, $U_S^3$ converges to $U_S^2$.

The difference between $R_S$ and $U_S$ is that $R_S$ is static as it is on the state space, while $U_S$ is dynamic as it concerns with the "movements". Due to introduction of the safety index, the non-convex state space constraint $R_S$ is transformed to a convex state space constraint $U_S$. For example, in Fig.4.3, the safe region $R_S^3$ for the AGV is the space outside the uncertainty range. But the set $U_S^3$ according to (4.11) is a half space.

According to the safety principle, the robot's control input should lie in the set $U_S^3$. There are two ways to choose the uncertainty bound $\dot{\Gamma}_H$. One way is to use a constant bound based on the mean prediction error. Another way is to dynamically adjust the uncertainty bound according to the level of uncertainties in real time. The first way corresponds to the safe set algorithm (SSA) [78] while the second one corresponds to the safe exploration algorithm (SEA) [83], which will be discussed in the following two sections.

## 4.3 The Safe Set Algorithm (SSA)

The safe set algorithm offers a fast online solution concerning the safety principle. In this section, the control algorithm and the learning algorithm in SSA will be discussed, followed by an application on a robot arm.

### 4.3.1 The Algorithm

In SSA, a constant $\lambda_R^{SSA} \in \mathbb{R}^+$ is introduced to bound the noises and uncertainties in the estimation, i.e.

$$S^{SSA}\left(t, \hat{\dot{x}}_H\right) = S(t, \hat{\dot{x}}_H) - \lambda_R^{SSA} = -\eta_R - \lambda_R^{SSA} - \sum_{j \in H} \frac{\partial \phi}{\partial x_j} \hat{\dot{x}}_j - \frac{\partial \phi}{\partial x_R} f_{Rx}^*, \tag{4.12}$$

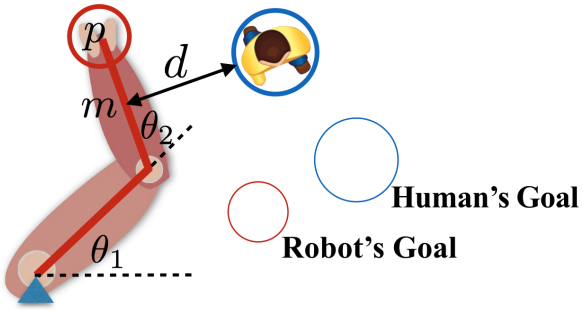
where $\hat{\dot{x}}_j = \frac{\hat{x}_j(k+1|k) - \hat{x}_j(k|k)}{t_s}$ for all $j \in H$. The index $k$ represents the time step of the last measurement before $t$, while $k+1$ is the time step for the next anticipated measurement. $\hat{x}_j(p|k)$ is the estimate of $x_j(p)$ given information up to time step $k$ as discussed in Chapter 3. The computation of $\hat{x}_j(k+1|k)$ and $\hat{x}_j(k|k)$ follow from (3.14-3.17). $L(t)$ and $S(t)$ will also be written as $L(k)$ and $S(k)$ to denote that the last measurement is taken at $k$-th time step. The control algorithm can also be designed in discrete time as discussed in [78].

With the set of safe control $U_S^{SSA}$, we can transform the non-convex optimization problem (4.1) into a convex optimization problem:

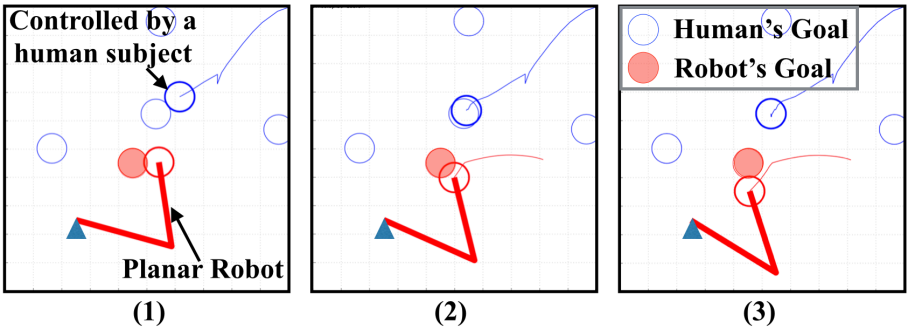
$$u_R^* = \min_{u_R \in \Omega \cap U_S^{SSA}} \quad \|u_R - u_R^o\|_Q^2, \tag{4.13}$$

where $U_S^{SSA}(t) = \{u_R : L(t)u_R \leq S^{SSA}(t, \hat{\dot{x}}_H)\}$. An analytic control law that solves the optimization approximately can be obtained. Let $c = \min_{u \in U_S^{SSA}(t)} |L(t)(u - u_R^o(t))|$. When $\Omega$ is not tight, the safe control input $u_R^*(t)$ is [78]:

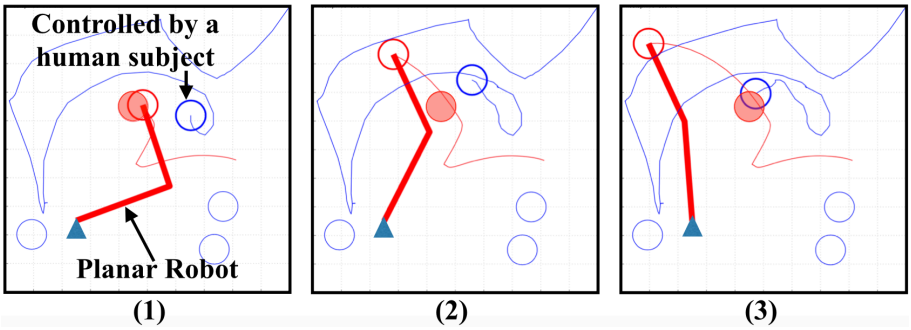
$$u_R^*(t) = u_R^o(t) - c\frac{Q^{-1}L(t)^T}{L(t)Q^{-1}L(t)^T}. \tag{4.14}$$




(a) The interaction between a robot arm and a human.




(b) The simulation result: scenario 1.




(c) The simulation result: scenario 2.

Figure 4.5: Application of the SSA algorithm on a planar robot arm.

### 4.3.2 Example: Local Planning of a Planar Robot Arm

In this section, the SSA is applied to a planar robot arm to demonstrate the effectiveness of the algorithm.

The environment is shown in Fig.4.5a where both the robot arm and the human need to approach their respective goals in minimum time without colliding with each other. The robot arm has two links with joint positions $\theta_1$ and $\theta_2$ and the joint velocities $\dot{\theta}_1$ and $\dot{\theta}_2$. Let $\theta = [\theta_1, \theta_2]^T$. Considering the kinematics of the robot arm, the control input is defined to be the joint accelerations, i.e. $u_R = \left[\ddot{\theta}_1, \ddot{\theta}_2\right]^T$. The end point position of the robot is denoted as $p = (p_x, p_y)$. The optimal control law is designed to be

$$u_R^o = J_p^{-1}\left\{K_p \begin{bmatrix} p_x - g_x \\ p_y - g_y \end{bmatrix} + K_v \begin{bmatrix} \dot{p_x} \\ \dot{p_y} \end{bmatrix} - H_p\right\}, \tag{4.15}$$

where $J_p$ is the Jacobian matrix at $p$ and $H_p = \dot{J}_p\dot{\theta}$. The pair $(g_x, g_y)$ is the goal point in the work space. The matrices $K_p \in \mathbb{R}^{2\times 2}$ and $K_v \in \mathbb{R}^{2\times 2}$ are the control gain.

The closest point to the human on the robot arm is denoted as $m = (m_x, m_y)$. Define the robot state as $x_R = [m_x, m_y, \dot{m}_x, \dot{m}_y]^T$. The state space equation is

$$\dot{x}_R = A_R x_R + B_R J_m u_R + B_R H_m, \tag{4.16}$$

where

$$A_R = \begin{bmatrix} 0 & I_2 \\ 0 & 0 \end{bmatrix}, B_R = \begin{bmatrix} 0 \\ I_2 \end{bmatrix},$$

and $J_m$ is the Jacobian matrix at $m$ with $H_m = \dot{J}_m\dot{\theta}$.

The human is simplified as a circle, whose state is taken as $x_H = [h_x, h_y, \dot{h}_x, \dot{h}_y]$ where $h_x$ and $h_y$ is the position and $\dot{h}_x$ and $\dot{h}_y$ is the velocity. Define the safe set as $X_S = \{x : d \geq d_{min}\}$ where $d$ measures the smallest distance between the human and the robot arm and $d_{min}$ is a positive constant. Based on the discussion in Section 4.2, the safety index is designed as $\phi = D - d^2 - k_\phi \dot{d}$ where $D = d_{min}^2 + \eta_R t_s + \lambda_R^{SSA} t_s$ and $k_\phi > 0$ are constants [78]. Let the relative distance, velocity and acceleration vectors be $\mathbf{d} = \begin{bmatrix} I_2 & 0 \end{bmatrix}(x_R - x_H)$,$\mathbf{v} = \begin{bmatrix} 0 & I_2 \end{bmatrix}(x_R - x_H)$ and $\mathbf{a} = \begin{bmatrix} 0 & I_2 \end{bmatrix}(\dot{x}_R - \dot{x}_H)$. Then $d = |\mathbf{d}|$ and

$$\begin{aligned} \dot{\phi} &= -2d\dot{d} - k_\phi \ddot{d} = -2\mathbf{d}^T\mathbf{v} - k_\phi \frac{\mathbf{d}^T\mathbf{a} + \mathbf{v}^T\mathbf{v} - \dot{d}^2}{d}, \\ &= -2\mathbf{d}^T\mathbf{v} - k_\phi \frac{\mathbf{d}^T\left(J_m u_R + H_m\right) - \mathbf{d}^T \begin{bmatrix} 0 & I_2 \end{bmatrix}\dot{x}_H + \mathbf{v}^T\mathbf{v}}{d} + k_\phi \frac{(\mathbf{d}^T\mathbf{v})^2}{d^3}. \end{aligned} \tag{4.17}$$

Hence

$$L(t) = -k_\phi \frac{\mathbf{d}^T}{d} J_m, \tag{4.18}$$

$$S(t, \dot{x}_H) = -\eta_R + 2\mathbf{d}^T\mathbf{v} + k_\phi \frac{\mathbf{d}^T H_m - \mathbf{d}^T \begin{bmatrix} 0 & I_2 \end{bmatrix}\dot{x}_H + \mathbf{v}\cdot\mathbf{v}}{d} - k_\phi \frac{(\mathbf{d}^T\mathbf{v})^2}{d^3}. \tag{4.19}$$

In the simulation, several goals were assigned for the human. Before parameter adaptation, the robot inferred on the human's current goal first [81]. Sampling time $t_s = 0.1s$. Figure 4.5b shows the human avoidance behavior of the robot. In (1), the human and the robot were both near their respective goals. However, since the human was heading towards the robot in high speed, the robot went backward in (2) and (3). Figure 4.5c shows the robot behavior under unexpected human behavior. In (1), the human suddenly changed his course. Although all of his goal points were in the lower part of the graph, the human started to go up. By observing that, the robot went away from the human in (2) and (3). The simulation results confirms the effectiveness of the algorithm.

## 4.4 The Safe Exploration Algorithm (SEA)

In SSA, the bound for the uncertainties (i.e. $\lambda_R^{SSA}$) is a constant. However, the mean squared estimation error (MSEE) of the human's state is changing from time to time. A larger bound is needed if the MSEE is larger. To capture this property, the safe exploration algorithm (SEA) is introduced, where the control input changes under different levels of uncertainties.

### 4.4.1 The Algorithm

In a belief space [34], the state estimate of $x_j$ for $j \in H$ is no longer a point but a distribution, i.e. $\mathcal{N}(\hat{x}_j, X_j)$ where $X_j$ is the covariance representing the level of uncertainties in the estimation. All the distributions are assumed to be Gaussian. Since $x_j \sim \mathcal{N}(\hat{x}_j, X_j)$, the covariance can be written as $X_j = E\left[(x_j - \hat{x}_j)(x_j - \hat{x}_j)^T\right]$, which is the mean squared estimation error (MSEE) defined in Chapter 3. The *a priori* and *a posteriori* estimates, estimation errors and MSEEs are all defined in Chapter 3 in Table 3.1. At the $k$-th time step, the best prediction for $x_j(k+1)$ has the following distribution

$$\mathcal{N}(\hat{x}_j(k+1|k), X_j(k+1|k)). \tag{4.20}$$

In the belief space, since the distribution of $x_j(k+1)$ is unbounded, the inequality in (4.11) is ill-defined. Indeed, $u_R$ needs to satisfy a probability constraint

$$P(\{x_j(k+1) : L(k)u_R \leq S(k, x_H)\}) \geq 1 - \epsilon, \forall j \in H, \tag{4.21}$$

where $\epsilon > 0$ is a small number. A bounded set $\Gamma_j(k)$ can be defined for $j \in H$ such that the probability density of $x_j \notin \Gamma_j(k)$ is small and $P(x_j \in \Gamma_j(k)) \geq 1 - \epsilon$. For a Gaussian distribution, the probability mass lying within the $3\sigma$ deviation is 0.997. Set $\epsilon = 0.003$ and let $\Delta x_j = x_j - \hat{x}_j(k+1|k)$, then the set $\Gamma_j$ can be defined as

$$\Gamma_j(k) = \left\{x_j : \Delta x_j^T X_j(k+1|k)^{-1} \Delta x_j \leq 9\right\}. \tag{4.22}$$

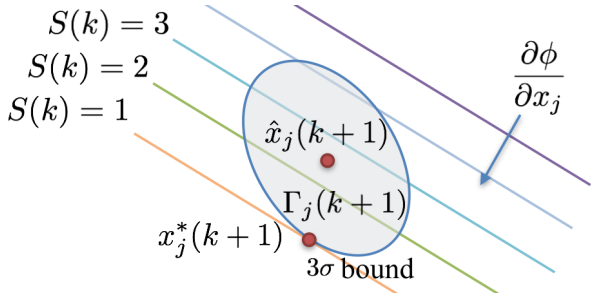



Figure 4.6: Illustration of the safety constraint in the belief space.

By (4.11) and (4.21), the constraint in $U_R^3$ is satisfied if the following inequality holds,

$$L(k)u_R \leq S^{SEA}(k) = \min_{x_j(k+1)\in\Gamma_j(k),\forall j\in H} \{S(k)\}. \tag{4.23}$$

By (4.8), the RHS of (4.23) can be decoupled as a sequence of optimization problems, i.e. for all $j \in H$,

$$\min_{x_j(k+1)\in\Gamma_j(k)} \frac{\partial \phi}{\partial x_j} x_j(k+1). \tag{4.24}$$

By Lagrangian method[2], the optimal solution $x_j^*(k+1)$ for all $j \in H$ is

$$x_j^*(k+1) = \hat{x}_j(k+1|k) + \frac{3X_j(k+1|k)\left(\frac{\partial \phi}{\partial x_j}\right)^T}{\left[\left(\frac{\partial \phi}{\partial x_j}\right) X_j(k+1|k)\left(\frac{\partial \phi}{\partial x_j}\right)^T\right]^{\frac{1}{2}}}. \tag{4.26}$$

According to (4.26), $S^{SEA}(k)$ can be expressed as

$$S^{SEA}(k) = S(k, \hat{x}_H) - \lambda_R^{SEA}(k) = -\eta_R - \lambda_R^{SEA}(k) - \sum_{j\in H} \frac{\partial \phi}{\partial x_j}\hat{\dot{x}}_j - \frac{\partial \phi}{\partial x_R} f_{Rx}^*, \tag{4.27}$$

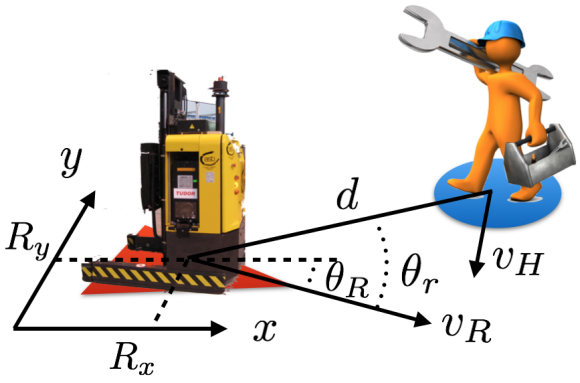
where

$$\lambda_R^{SEA}(k) = \frac{3}{t_s}\sum_{j\in H}\left[\left(\frac{\partial \phi}{\partial x_j}\right) X_j(k+1|k)\left(\frac{\partial \phi}{\partial x_j}\right)^T\right]^{\frac{1}{2}} + \lambda_R^o, \tag{4.28}$$

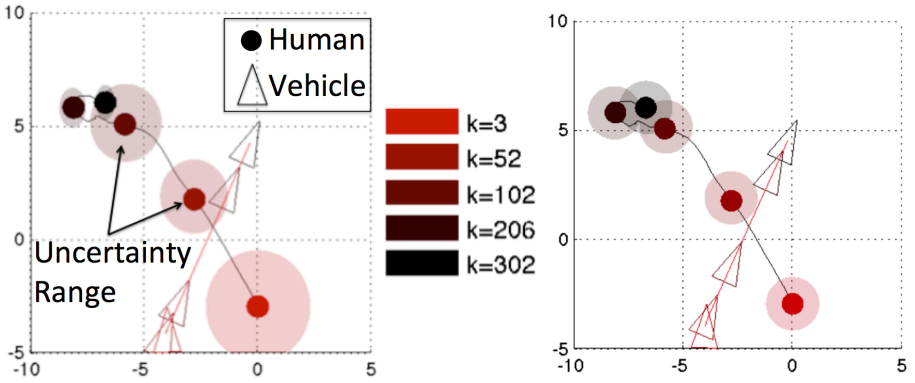
[2]The objective function is linear while the constraint function defines an ellipsoid as shown in Fig.4.6. The optimal solution must lie on the boundary of the ellipsoid. Let $\gamma$ be a Lagrange multiplier. Define the new cost function as:

$$J_j^* = \frac{\partial \phi}{\partial x_j} x_j(k+1) + \gamma\left[9 - \Delta x_j^T X_j(k+1|k)^{-1}\Delta x_j\right]. \tag{4.25}$$

The optimal solution satisfies $\frac{\partial J_j^*}{\partial x_j(k+1)} = \frac{\partial J_j^*}{\partial \gamma} = 0$, i.e. $(\frac{\partial \phi}{\partial x_j})^T - 2\gamma X_j(k+1|k)^{-1}\Delta x_j = 0$ and $9 - \Delta x_j^T X_j(k+1|k)^{-1}\Delta x_j = 0$. Then (4.26) follows.




(a) The interaction between an AGV and a human.




(b) Simulation result with SEA. (c) Simulation result with SSA.

Figure 4.7: Application of the SEA algorithm and the SSA algorithms on an AGV.

and $\lambda_R^o \in \mathbb{R}^+$ is the bound for other uncertainties. All other equations follow from the safe set algorithm except for the learning and prediction part. In SSA, we only estimate the expected state of the humans, e.g. $\hat{x}_j(k+1|k)$. In SEA, the MSEE of $\hat{x}_j(k+1|k)$ for all $j \in H$ also needs to be estimated using the algorithm discussed in Chapter 3 in (3.20) and (3.23). In the implementation, the covariance of the noise $W$, the time varying rate $d\vartheta$ and the initial values are hand-tuned.

### 4.4.2 Example: Local Planning of a Vehicle

In this section, a comparative study between SSA and SEA is performed on a vehicle shown in Fig.4.7a. The vehicle's state is denoted by $x_R = [R_x, R_y, v_R, \theta_R]^T$ where $R_x$ is the x-position of the vehicle, $R_y$ the y-position, $v_R$ the speed and $\theta_R$ the direction. The control input of the vehicle is $u_R = [\dot{v}_R, \dot{\theta}_R]^T$ (saturations apply: $|\dot{v}_R| \leq a_{max}$ and $|\dot{\theta}_R| \leq \omega_{max}$, where

$a_{max}, \omega_{max}$ are positive constants). The state equation is

$$\dot{x}_R = f^*_{Rx}(x_R) + Bu_R, \tag{4.29}$$

where $f^*_{Rx}(x_R) = \begin{bmatrix} v_R\cos\theta_R & v_R\sin\theta_R & 0 & 0 \end{bmatrix}^T$, $B = \begin{bmatrix} 0 & I_2 \end{bmatrix}^T$.

The vehicle can measure its own state directly. It can also measure the relative distance $d$ and the relative direction $\theta_r$ towards the nearby human as illustrated in Fig.4.7a. The human's state is $x_H = [h_x, h_y, \dot{h}_x, \dot{h}_y]^T$, which is calculated based on the measurements and the robot state. Suppose that the goal point of the robot is $[G_x, G_y]$. The baseline control law is designed as [83]

$$\dot{v}_R = -\left[(R_x - G_x)\cos\theta_R + (R_y - G_y)\sin\theta_R\right] - k_v v_R, \tag{4.30}$$

$$\dot{\theta}_R = k_\theta \left[\arctan\frac{R_y - G_y}{R_x - G_x} - \theta_R\right], \tag{4.31}$$

where $k_v, k_\theta \in \mathbb{R}^+$ are constants.

The safety index $\phi = D - d^2 - k_\phi \dot{d}$ is chosen to be the same as in section 4.3.2. In SSA, $D$ is set to be $d^2_{min} + \eta_R t_s + \lambda^{SSA}_R t_s$. In SEA, $D$ is set to be $d^2_{min} + \eta_R t_s + \lambda^{SEA}_R(k) t_s$. The relative distance, velocity and acceleration vectors are

$$\begin{aligned} \mathbf{d} &= [d\cos(\theta_r + \theta_R), d\sin(\theta_r + \theta_R)]^T, \\ \mathbf{v} &= [v\cos(\theta_R) - \dot{h}_x, v\sin(\theta_R) - \dot{h}_y]^T, \\ \mathbf{a} &= \begin{bmatrix} \cos\theta_R & -v_R\sin\theta_R \\ \sin\theta_R & v_R\cos\theta_R \end{bmatrix} u_R - \begin{bmatrix} 0 & I_2 \end{bmatrix} \dot{x}_H. \end{aligned}$$

Similar to (4.17), the time derivative of the safety index is

$$\begin{aligned} \dot{\phi} &= -2\mathbf{d}^T\mathbf{v} - k_\phi \frac{\mathbf{d}^T\mathbf{a} + \mathbf{v}^T\mathbf{v} - \dot{d}^2}{d}, \\ &= -2\mathbf{d}^T\mathbf{v} - k_\phi \frac{[d\cos\theta_r, -dv_R\sin\theta_r]\, u_R - \mathbf{d}^T \begin{bmatrix} 0 & I_2 \end{bmatrix} \dot{x}_H + \mathbf{v}^T\mathbf{v}}{d} + k_\phi \frac{(\mathbf{d}^T\mathbf{v})^2}{d^3} \end{aligned} \tag{4.32}$$

which implies

$$L(t) = k_\phi \left[\cos\theta_r, -v_R\sin\theta_r\right], \tag{4.33}$$

$$S(t, \dot{x}_H) = -\eta_R + 2\mathbf{d}^T\mathbf{v} + k_\phi \frac{\mathbf{v}^T\mathbf{v} - \mathbf{d}^T \begin{bmatrix} 0 & I_2 \end{bmatrix} \dot{x}_H}{d} - k_\phi \frac{(\mathbf{d}^T\mathbf{v})^2}{d^3}. \tag{4.34}$$

Then $S^{SSA}$ and $S^{SEA}$ follow from (4.12) and (4.27) respectively, and the final control input follows from (4.14).

Figure 4.7b and Fig.4.7c show the vehicle trajectories under SSA and SEA. The vehicle needed to approach (0,5) from (-5,-5) while the human went from (0,-3) to (-5,5). Five time steps are shown in the plots: $k = 3, 52, 102, 206, 302$ from the lightest to the darkest. The

solid circles represent the human, which was controlled by a human subject through a multi-touch trackpad in real time (notice there was overshoot as the control was not perfect). The triangles represent the vehicle. The transparent circles in Fig.4.7b represent the set $\Gamma_H(k)$ in (4.22) mapped into 2D, which is shrinking gradually due to the reduction of uncertainties as an effect of learning. In Fig.4.7c, the transparent circles represent the equivalent uncertainty levels introduced by $\lambda_R^{SSA}$, thus the radius remain constant throughout the time.

Figure 4.8 shows the distance profiles and the vehicle velocity profiles under SSA and SEA. Due to large initial uncertainties, the vehicle only started to accelerate after $k = 50$ (when the relative distance was large) in SEA. However, in SSA, the vehicle tried to accelerate in the very beginning, then decelerated when the relative distance to the human decreased. The velocity profile in SSA was serrated, while the one in SEA was much smoother. Meanwhile, in both algorithms, the relative distance was always greater than $d_{min} = 3$. However, before $k = 150$, the relative distance was kept larger in SEA than in SSA, since the vehicle was more conservative in SEA due to large uncertainty. Figure 4.8c shows that the *a priori* MSEE provides a perfect bound for the prediction error, while the prediction error reduces gradually, hence validates the learning algorithms.

In conclusion, the behavior in SSA is: move and modify; while in SEA, it is: move only if confident. The behavior under SEA is better for a new comer, while the behavior under SSA is better if the robot is already very familiar with the environment, i.e. with low uncertainty levels.

## 4.5 An Integrated Method for Time Varying Topology

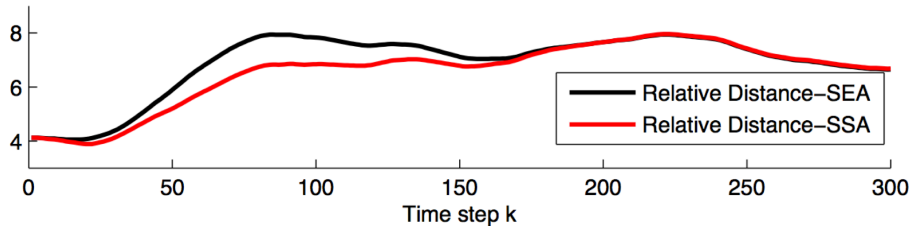
In real world applications, the system topology is usually time varying, e.g. the robot will encounter different agents at different time in different locations [40]. Mathematically, that means some agents will be decoupled from the system block diagram in Fig.2.2 and others will join from time to time. The robot is not faced with the "same" system throughout the time. This scenario is common for mobile robots and automated vehicles [80].

As the robot needs to deal with new agents, SEA is more appropriate than SSA. However, when the number of agents increase, the computation complexity in SEA increases dramatically. In this section, a method to combine SSA and SEA is discussed in order to balance the performance and the computation complexity.

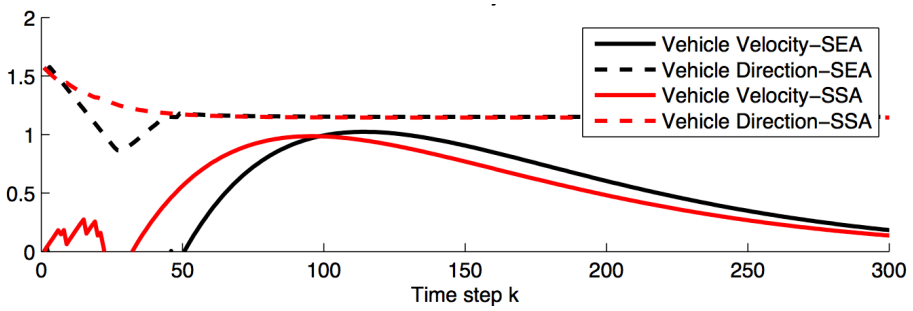
### 4.5.1 The Algorithm

#### The Control Algorithm

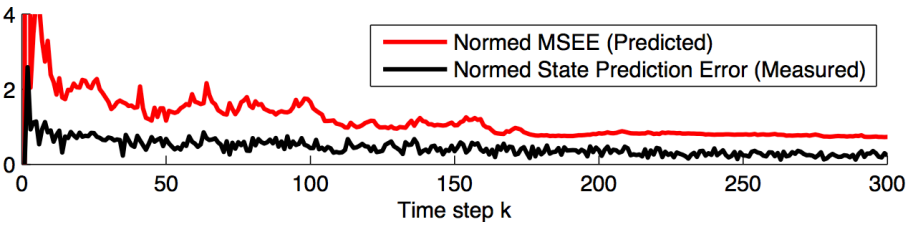
Due to limited sensing capability, the robot can only track humans that are within certain distance. Every agent within this range will be assigned a special identification number. Let $H(k)$ denotes the collection of those identification numbers at time step $k$. A safety index $\phi_j$




(a) The relative distance profiles in SEA and SSA.




(b) The velocity profiles of the vehicle in SEA and SSA.




(c) The *a priori* MSEE as a bound of the state prediction error.

Figure 4.8: Comparison between the SEA algorithm and the SSA algorithm.

is designed for each agent $j \in H(k)$, as it is hard to design one analytical safety index that satisfies all the requirements for time varying $H(k)$. In this way, $u_R$ needs to be constrained by

$$U_R^3 = \bigcap_{j \in H(k)} U_{R,j}^3 = \{u_R : L_j(k) u_R(k) \leq S_j(t, \dot{x}_j) \text{ for all } \dot{x}_j \in \dot{\Gamma}_j\}, \tag{4.35}$$

where $L_j$ and $S_j$ are calculated with respect to $\phi_j$. The uncertainty bound $\dot{\Gamma}_j$ is chosen according to SEA only for new agents. Once the MSEE converges, the algorithm is switched to SSA for that agent. The idea is illustrated in Algorithm 4.1, where $\Pi$ is a set that records the identification numbers of the agents who are no longer considered as new agents.

Initialize $\Pi = \emptyset$, $k = 0$;
**while** *Controller is Active* **do**
  $k = k + 1$;
  Read current $H(k)$ and $y_R^j(k)$ for $j \in H(k)$;
  **for** $j \in H(k)$ **do**
    calculate the estimate $\hat{x}_j(k+1|k)$ based on measurements $y_R^j(k)$;
    **if** $j \notin \Pi$ **then**
      calculate the MSEE $X_j(k+1|k)$;
      **if** $X_j$ *converges* **then**
        $\Pi = \Pi \cup \{j\}$;
      **end**
    **end**
  **end**
  **for** $j \in H(k)$ **do**
    **if** $j \in \Pi$ **then**
      $U_{R,j}^3 = U_{R,j}^{SSA} = \{u_R : L_j(k) \leq S_j^{SSA}(k)\}$ (Apply SSA to $\phi_j$);
    **else**
      $U_{R,j}^3 = U_{R,j}^{SEA} = \{u_R : L_j(k) \leq S_j^{SEA}(k)\}$ (Apply SEA to $\phi_j$);
    **end**
  **end**
  $U_R^3 = \bigcap_{j \in H(k)} U_{R,j}^3$;
  Choose control $u_R^*$ by optimizing over $U_R^3$;
**end**

**Algorithm 4.1:** The Algorithm Combining SSA and SEA

### The Learning Algorithm

When the number of agents increases, the computation complexity regarding (3.13) increases exponentially. However, the correlation among agents are over estimated, e.g. an agent's motion may only be affect by several surrounding agents instead of by all agents. When the system topology is time varying, it is better to learn agents' dynamics separately and use low dimension features to represent the correlations among agents. It is assumed that an agent $j$'s motion will be affected by several features $f_j^p$ for $p = 1, 2, \ldots$, e.g. the distance to the nearest agent and distance to the goal point. Then the linearized closed loop dynamics of agent $j$ can be written as

$$x_j(k+1) = A_j(k)\, x_j(k) + \sum_p B_j^p(k)\, f_j^p(k) + w_j^*(k), \tag{4.36}$$

where $w_j^*(k)$ is a noise term. Then the parameters $A_j(k)$ and $B_j^p(k)$ can be identified using the methods discussed in Chapter 3.3.

### 4.5.2 Example: Robot Navigation in a Crowded Environment

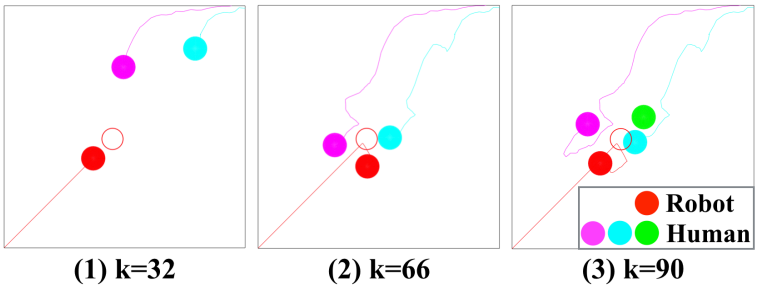
The strategy is tested on robot navigation in a crowded environment with multiple humans as shown in Fig.4.9a. The robot is modeled as a double integrator whose input is the acceleration in $x$ and $y$ directions. The position of the humans were controlled in real time by several human subjects who observed the virtual environment though a screen. The humans did not have specific tasks and were just "wandering" in the environment. The robot is required to approach its goal while avoiding humans. The safety index was the same as in section 4.4.2. The features in (4.36) were chosen to be the distance to the closest human and the distance to the robot.

The simulation result is shown in Fig.4.9b, Fig.4.9c and Fig.4.9d. Before the 50-th time step, the robot was trying to approach its goal. It detoured when the blue agent came close. At the same time, the green agent which was previously hidden by the blue agent showed up in the robot's view. The new agent surprised the robot, and resulted in a large peak in the robot velocity profile in Fig.4.9c. Algorithm-wise, it was the large uncertainty of the green agent that "pushed" the robot away. The constraint $U_R^3$ was effective after the 50-th time step as evidenced in Fig.4.9c. The relative distance between the robot and every human agent was always maintained greater than $d_{min}$ as shown in Fig.4.9d.

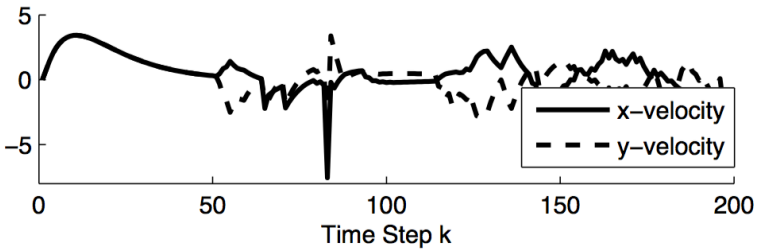
## 4.6 Conclusion

This chapter discussed a general methodology in designing the short term robot controller for safe HRI. The safety issues were understood as conflicts in the multi-agent system. To solve the conflicts, the robot's behavior was constrained according to the "social norm" and the uncertainties it perceived for the other agents (including humans and other robots). Two algorithms were discussed under the framework: the safe set algorithm (SSA) and the safe exploration algorithm (SEA). In both algorithms, the robot calculated the optimal action to finish the task while staying safe with respect to the predicted human motion. The difference was that SEA actively tracked the uncertainty levels in the prediction and incorporated that information in robot control, while SSA did not. As shown in the human-involved simulations, SEA was better when the uncertainty levels change from time to time, especially in the early stages of human-robot interactions. On the other hand, SSA was better when the predictions were more accurate, e.g. when the robot was "familiar" with the human, as SSA was more computationally efficient than SEA. Finally, a method to combine both algorithms was proposed to take the advantage of both algorithms. Several case studies were presented and demonstrated the effectiveness of the method.

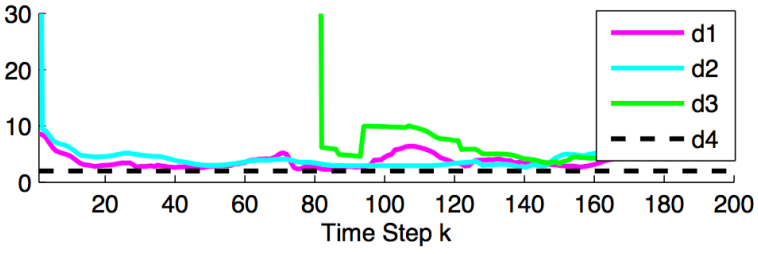

(a) Robot navigation in a crowded environment.




(b) The snapshots of the simulation result.




(c) The velocity profile of the robot.




(d) The relative distance profile among the robot and the human agents.

Figure 4.9: Application of the integrated method on robot navigation.

# Chapter 5

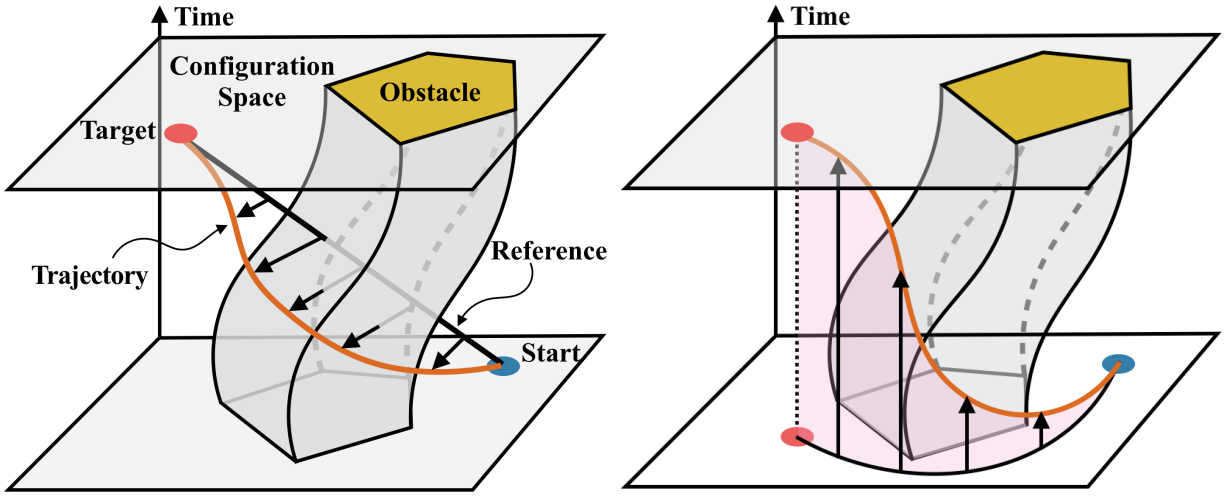
# Efficiency-Oriented Global Motion Planning

Global or long term trajectory or motion planning is one of the key challenges in robotics. Robots need to find motion trajectories to accomplish certain tasks in constrained environments in real time. The scenarios include but are not limited to navigation of unmanned areal or ground vehicles in civil tasks such as search and rescue, surveillance and inspection; navigation of autonomous or driverless vehicles in future transportation systems; or motion planning for industrial robots in factory floor. In the context of human-robot interactions, the motion planning should be accomplished in real time in order to for the robot to response to environmental changes. In this chapter, we focus on optimization-based global motion planning.

## 5.1 Overview

Global motion planning is to find a trajectory that solves (2.7) in a long time horizon. A trajectory has two attributions, spatial attribution and temporal attribution. To avoid dynamic obstacles such as human, the robot can resort to either spatial maneuvers such as detours or temporal maneuvers such as slowing down or speeding up. Regarding these two kinds of maneuvers, there are two planning frameworks in literature, e.g. the integrated framework and the layered framework.

The integrated framework relies on spatiotemporal planning, which considers the spatial and temporal maneuvers simultaneously. On the other hand, the layered framework separates the considerations on the spatial and temporal maneuvers by planning a path first and then generating a speed profile along the path [42]. In the cases when there is not much freedom for the robot to choose alternative paths, nudging the speed profile is the only choice to respond to moving obstacles. Figure 5.1 illustrates the differences between the two frameworks under optimization-based methods. The horizontal plane represents the 2D state space. The vertical axis represents time. The shaded volume is the time-augmented obstacle. In the




(a) Optimization-based spatial-temporal planning. (b) Optimization-based temporal planning.

Figure 5.1: The global motion planning problem and typical methods.

integrated framework, a trajectory needs to be computed directly in the spatiotemporal space starting from a simple reference trajectory connecting the start point and the target point as shown in Fig.5.1a. In the layered framework, the trajectory is planned by assigning temporal information to the points on a given path as shown in Fig.5.1b.

In the remainder of this chapter, the global motion planning problem will be re-formulated first, followed by the discussion of optimization-based trajectory planning in an integrated framework in Section 5.3 and optimization-based speed profile planning in a layered framework in Section 5.4.

## 5.2 Problem Formulation

The problem (2.7) will be reformulated in the context of trajectory planning in this section.

**Representing the Trajectory** Denote the configuration of the robot as $p \in C \subset \mathbb{R}^n$ where $C$ is the configuration space and $n$ is its dimension. Note that the configuration of a robot may not be identical to the state $x_R$ of the robot since configuration does not include velocity information. For a mobile robot, $p$ is the position of the robot in the plane; for an aerial robot, $p$ is the position of the robot in the space; for a robot arm, $p$ is the joint position of the robot.

Let $\mathbf{p} : [0, s^*] \to C$ denotes a path which is parameterized by length $s$. The total length of the path is $s^*$. $\mathbf{p}(s) \in \mathbb{R}^n$ is a point on the path $\mathbf{p}$ which has distance $s$ to the starting

point $\mathbf{p}(0)$. $\dot{\mathbf{p}} := \partial\mathbf{p}/\partial s$ and $\ddot{\mathbf{p}} := \partial^2\mathbf{p}/\partial s^2$ denote the tangent vector and the normal vector of the path respectively. Since the path $\mathbf{p}$ is parameterized by length, its tangent vector has unit length, e.g. $\|\dot{\mathbf{p}}\| = 1$, and the length of its normal vector represents the curvature of the path. Hence we define the curvature $\kappa : [0, s^*] \to \mathbb{R}$ along the path $\mathbf{p}$ as $\kappa(s) := \|\ddot{\mathbf{p}}(s)\|$.

Let $\mathbf{s} : [0, t^*] \to [0, s^*]$ be a mapping from the time axis to the station on the path. $\mathbf{s}$ encodes the speed profile, e.g. $\dot{\mathbf{s}}(t)$ is the speed of the object at time $t$. This mapping is non decreasing, but may not be bijective as the speed can be zero.

With the speed profile, a trajectory $\mathbf{T} : [0, t^*] \to \mathbb{R}^n$ is defined to be $\mathbf{T} := \mathbf{p} \circ \mathbf{s}$ where $\circ$ denotes function composition. The velocity along the path is

$$\dot{\mathbf{T}}(t) = \dot{\mathbf{p}}(\mathbf{s}(t)) \cdot \dot{\mathbf{s}}(t), \tag{5.1}$$

which equals to the speed $\dot{\mathbf{s}}$ times the tangent vector $\dot{\mathbf{p}}$ at point $\mathbf{s}(t)$. The acceleration along the path is

$$\ddot{\mathbf{T}}(t) = \ddot{\mathbf{p}}(\mathbf{s}(t)) \cdot \dot{\mathbf{s}}(t)^2 + \dot{\mathbf{p}}(\mathbf{s}(t)) \cdot \ddot{\mathbf{s}}(t), \tag{5.2}$$

which have two components, e.g. the longitudinal acceleration $\ddot{\mathbf{s}}$ and the lateral acceleration $\kappa(\mathbf{s})\dot{\mathbf{s}}^2$. For simplicity, we sometimes omit the parameter $t$ or $s$ if there is no ambiguity.

Let $\mathbf{p}_{a\to b}$ be a path connecting points $a$ and $b$ in $\mathbb{R}^n$, e.g. $\mathbf{p}(0) = a$ and $\mathbf{p}(s^*) = b$. Similarly, let $\mathbf{s}_{c\to d}$ be a speed profile such that $\dot{\mathbf{s}}(0) = c$ and $\dot{\mathbf{s}}(t^*) = d$. Let $\mathbf{T}_{[a,c]\to[b,d]}$ be a trajectory $\mathbf{T}$ composed of $\mathbf{p}_{a\to b}$ and $\mathbf{s}_{c\to d}$.

**The Safety Constraint** The area in the Cartesian space that is occupied by the robot with configuration $p$ is denoted as $V(p) \in \mathbb{R}^k$ where $k = 2$ or 3 is the dimension of the Cartesian space. The area occupied by the obstacles or other agents in the environment at time $t$ is denoted as $\mathcal{O}_t^C \in \mathbb{R}^k$, which can be predicted using the methods described in Chapter 3. Let $d_E : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ be the Euclidean distance function in the Cartesian space. The distance from the robot to the obstacles in the Cartesian space is computed as

$$d(p, \mathcal{O}_t^C) := \min_{y\in V(p), z\in\mathcal{O}_t^C} d_E(y, z). \tag{5.3}$$

Suppose the minimum distance requirement is $d_{min}$. Then the obstacle in the configuration space at time $t$ is

$$\mathcal{O}_t := \{p : d(p, \mathcal{O}_t^C) <= d_{min}\}. \tag{5.4}$$

Note that $\mathcal{O}_t$ can be regarded as the projection of the Cartesian space obstacles to the configuration space.

**The Dynamic Constraint** The constraints come from kinematic limits and limitations on control effort. Kinematic limits include speed limits and acceleration limits. For example, the curvature of the path should be bounded for nonholonomic vehicles. Denote the constraint as

$$F(\mathbf{T}, \dot{\mathbf{T}}, \ddot{\mathbf{T}}, \cdots) \in \Omega. \tag{5.5}$$

**The Boundary Constraint** The initial condition is always fixed as it is determined by the robot's current position and velocity. The terminal condition varies, which can either be in fixed time horizon or for fixed target as discussed in Chapter 2. One example of fixed time horizon problems is vehicle lane following where we only need to specify the lateral position and desired longitudinal speed for the vehicle instead of the longitudinal position. One example of fixed target problems is robot grasping or vehicle parking which requires the robot to go to certain destination.

**The Problem** The global motion planning problem can be re-formulated as

$$\min_{\mathbf{T}_{\mathcal{A}\to\mathcal{Z}}} \quad J(\mathbf{T}_{\mathcal{A}\to\mathcal{Z}}) \tag{5.6a}$$

$$F(\mathbf{T}, \dot{\mathbf{T}}, \ddot{\mathbf{T}}, \cdots) \in \Omega \tag{5.6b}$$

$$\mathbf{T}(t) \notin \mathcal{O}_t, \forall t. \tag{5.6c}$$

where (5.6a) is the cost function (soft constraint), (5.6b) is the dynamic constraint and (5.6c) is the collision avoidance constraint. $\mathcal{A}$ and $\mathcal{Z}$ are the boundary conditions.

## 5.3 Optimization-Based Trajectory Planning

Optimization methods are widely adopted in robot trajectory planning due to its flexibility to address multiple objectives [115, 123]. The major challenges lie in real time computation. The optimization problem for trajectory smoothing in a clustered environment is usually highly non-convex, which is hard to solve in real time using conventional non-convex optimization solvers such as sequential quadratic programming (SQP) [17]. A convex feasible set (CFS) algorithm [76] is proposed to convexify the problems. In this section, the optimization-based trajectory planning problem will be discussed first, followed by a quadratic approximation of the non-convex problem using the CFS algorithm. Then its performance will be illustrated through several examples.

### 5.3.1 Problem Formulation

Discretize the trajectory $\mathbf{T}$ into $h$ points and define $x_q := \mathbf{T}(qt_s)$ where $t_s$ is the sampling time. The discrete trajectory is now denoted as $\mathbf{x} = [x_0^T, x_1^T, \cdots, x_h^T]^T$. The reference trajectory is denoted as $\mathbf{T}^r$ and also discretized as $\mathbf{x}^r$ which consists of a sequence of reference states $x_q^r := \mathbf{T}^r(qt_s)$ for different time step $q$. Define the finite difference operators $V \in \mathbb{R}^{nh\times n(h+1)}$ and $A \in \mathbb{R}^{n(h-1)\times n(h+1)}$ as

$$V = \frac{1}{t_s}\begin{bmatrix} I_n & -I_n & 0 & \cdots & 0 \\ 0 & I_n & -I_n & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & I_n & -I_n \end{bmatrix}, A = \frac{1}{t_s^2}\begin{bmatrix} I_n & -2I_n & I_n & 0 & \cdots & 0 \\ 0 & I_n & -2I_n & I_n & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & I_n & -2I_n & I_n \end{bmatrix}.$$

Note that $V\mathbf{x}$ is the velocity vector and $A\mathbf{x}$ is the acceleration vector of the trajectory $\mathbf{x}$. Rewriting (5.6) in the discrete time as

$$\min_{\mathbf{x}} \; J(\mathbf{x};\mathbf{x}^r) = w_1\|\mathbf{x}-\mathbf{x}^r\|_Q^2 + w_2\|\mathbf{x}\|_S^2 \tag{5.7a}$$

$$s.t.\; x_0 = \mathcal{A}, x_h = \mathcal{Z} \tag{5.7b}$$

$$v_{min} \le V\mathbf{x} \le v_{max}, a_{min} \le A\mathbf{x} \le a_{max} \tag{5.7c}$$

$$x_q \notin \mathcal{O}_q, \forall q = 1, \cdots, h-1. \tag{5.7d}$$

The cost function is designed to be quadratic where $w_1, w_2 \in \mathbb{R}^+$. $\|\mathbf{x}-\mathbf{x}^r\|_Q^2 := (\mathbf{x}-\mathbf{x}^r)^T Q(\mathbf{x}-\mathbf{x}^r)$ penalizes the distance from the target trajectory to the reference trajectory. $\|\mathbf{x}\|_S^2 := \mathbf{x}^T S\mathbf{x}$ penalizes the properties of the target trajectory itself, e.g. length of the trajectory and magnitude of acceleration. The positive definite matrices $Q, S \in \mathbb{R}^{n(h+1)\times n(h+1)}$ can be constructed from the following components: 1) matrix for position $Q_1 := I_{n(h+1)}$; 2) matrix for velocity $Q_2 := V^T V$ and 3) matrix for acceleration $Q_3 := A^T A$. Then $Q := \sum_{i=1}^3 c_i^q Q_i$ and $S := \sum_{i=1}^3 c_i^s Q_i$ where $c_i^q$ and $c_i^s$ are positive constants. Constraint (5.7b) is the boundary condition. Constraint (5.7c) is the linear constraint which represents velocity and acceleration limits where $v_{min}, v_{max} \in \mathbb{R}^h$ and $a_{min}, a_{max} \in \mathbb{R}^{h-1}$. Constraint (5.7d) is for collision avoidance where $d_{min} \in \mathbb{R}^+$ is constant.

### 5.3.2 Quadratic Approximation

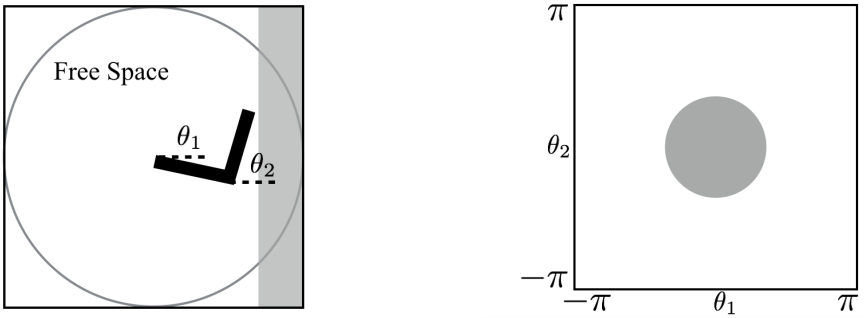
The non-convex optimization problem (5.7) is hard to solve in general using conventional non-convex optimization solvers. We will solve it using the convex feasible set algorithm (CFS). The details of the algorithm will be discussed in Chapter 6. The key idea is to transform problem (5.7) into a quadratic programming by obtaining a linear subset of the nonlinear constraint (5.7d).

For each time step $q$, the infeasible set is $\mathcal{O}_q$. Let $d_E^* : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be the Euclidean distance function in the Configuration space. Then constraint (5.7d) is equivalent to $d^*(x_q, \mathcal{O}_q) \ge 0$ where $d^*(x_q, \mathcal{O}_q)$ is the signed distance function to $\mathcal{O}_q$ such that

$$d^*(x_q, \mathcal{O}_q) := \begin{cases} \min_{z\in\partial\mathcal{O}_q} d_E^*(x_q, z) & x_q \notin \mathcal{O}_q \\ -\min_{z\in\partial\mathcal{O}_q} d_E^*(x_q, z) & x_q \in \mathcal{O}_q \end{cases}. \tag{5.8}$$

$\partial\mathcal{O}_q$ denotes the boundary of the obstacle $\mathcal{O}_q$. Figure 5.2 illustrates the obstacle in the Cartesian space and the corresponding $\mathcal{O}_q$ (shaded part) in the configuration space for a planar robot arm.

Note that if $\mathcal{O}_q$ is convex, then the function $d^*(\cdot, \mathcal{O}_q)$ is also convex. Hence $d^*(x_q^r, \mathcal{O}_q) + \nabla d^*(x_q^r, \mathcal{O}_q)(x_q - x_q^r) \ge 0$ implies $d^*(\cdot, \mathcal{O}_q) \ge 0$, which further implies that $x_q \notin \mathcal{O}_q$. If the obstacle $\mathcal{O}_q$ is not convex, we then break it into several simple convex objects $\mathcal{O}_q^i$ such as circles or spheres, polygons or polytopes. The $\mathcal{O}_q^i$'s need not be disjoint. Then $d^*(\cdot, \mathcal{O}_q^i)$ is the convex cone of the convex set $\mathcal{O}_q^i$ as shown in Fig.5.3a. Replacing (5.7d) with its first




(a) The constraint in the Cartesian space. (b) The constraint in the Configuration space.

Figure 5.2: The constraints for a planar robot arm in Cartesian and Configuration spaces.

order approximation, the new optimization problem becomes

$$\min_{\mathbf{x}} \quad J(\mathbf{x};\mathbf{x}^r) = w_1\|\mathbf{x}-\mathbf{x}^r\|_Q^2 + w_2\|\mathbf{x}\|_S^2 \tag{5.9a}$$

$$s.t. \quad x_0 = \mathcal{A}, x_h = \mathcal{Z} \tag{5.9b}$$

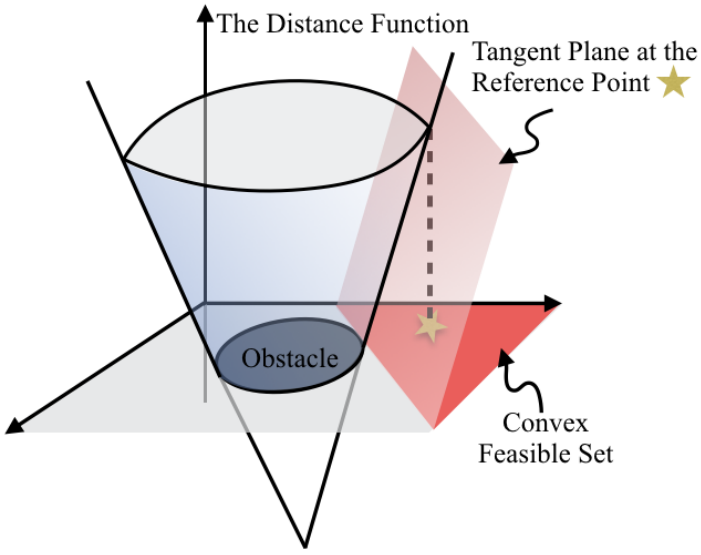
$$v_{min} \le V\mathbf{x} \le v_{max}, a_{min} \le A\mathbf{x} \le a_{max} \tag{5.9c}$$

$$d^*(x_q^r, \mathcal{O}_q^i) + \nabla d^*(x_q^r, \mathcal{O}_q^i)(x_q - x_q^r) \ge 0, \forall q, i. \tag{5.9d}$$

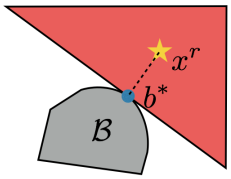
The constraint (5.9d) is a convex feasible set of the original non-convex constraint since $d^*(\cdot, \mathcal{O}_q^i)$ are convex for all $i$. The non-convex problem (5.7) reduces to a quadratic problem (5.9) which can be solved efficiently using quadratic programming.

For iterative implementation, we can set $\mathbf{x}^{(0)}$ to be $\mathbf{x}^r$ and substitute $x_q^r$ in (5.9d) with $x_q^{(k)}$ at each iteration $k$. It will be shown in Chapter 6 that the sequence $\mathbf{x}^{(k)}$ generated by the iterations will converge to a local optimum of the original problem (5.7) if the reference $\mathbf{x}^r$ is nearly feasible, i.e. there exists some $\mathbf{x}$ that satisfies the constraint (5.7d). Nonetheless, in the simulations, it will be verified that the trajectory after one iteration is good enough in the sense of feasibility and optimality. Thus we can safely work with the non-iterative version if strict optimality is not required and the computation time is limited.

**Visualization of the Convex Feasible Set** The convex feasible set in (5.9d) is illustrated in Fig.5.3a for certain $q$ and $i$ when $X \subset \mathbb{R}^2$. The left hand side of (5.9d) represents the tangent plane of the distance function $d^*$ at the reference point $x_q^r$. Due to the cone structure of $d^*$, the tangent plane touches the boundary of the obstacle. The convex feasible set is the projection of the positive portion of the tangent plane onto $\mathbb{R}^2$, which is a half space. The half space is maximal in the sense that the distance from the reference point to the boundary of the half space is maximized, which is equal to the distance from the reference point to the obstacle. With this observation, we can construct the convex feasible set using




(a) Illustration of the convex feasible set.




(b) The geometric interpretation.

Figure 5.3: The convex feasible set and its geometric interpretation.

purely geometric method without differentiation. For any reference point $x^r$ and any convex obstacle $\mathcal{O}$, denote the closest point on $\mathcal{O}$ to $x^r$ as $b^*$. The convex feasible set for $x^r$ with respect to $\mathcal{O}$ is just the half space which goes through $b^*$ and whose normal direction is along $b^* - x^r$ as shown in Fig.5.3b.

At each time step $q$, the convex feasible set with respect to all obstacles is a polygon that is the intersection of all feasible half spaces. The polygon is always nonempty since the reference point is nearly feasible. Thus the convex feasible set for the whole planning horizon is a "tube" around the reference trajectory whose cross sections are those polygons. Similar idea of using a "tube" constraint to simplify the trajectory smoothing problem can be founded in [155]. The advantages of our method over the existing methods are that 1) the "tube" is maximized and 2) the feasibility and convergence of the algorithm is guaranteed theoretically.

### 5.3.3 Examples: Trajectory Planning for Various Systems

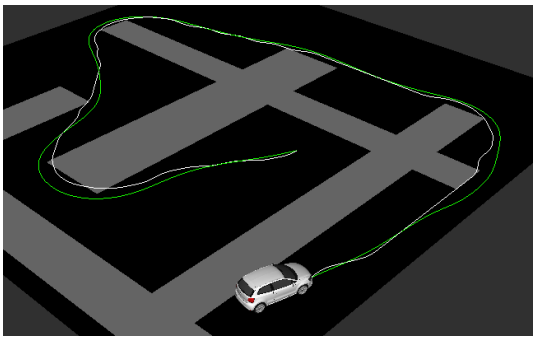
**Mobile Vehicle** In this section, trajectory planning problems for various robots are considered, including a mobile robot, a planar robot arm and an areal robot. The problem is solved using both the CFS algorithm and the sequential quadratic programming (SQP) algorithm. The algorithms are run in Matlab (using Matlab script) on a MacBook of 2.3 GHz using Intel Core i7. The SQP algorithm solves the problem (5.7) directly using Matlab `fmincon` function. The CFS algorithm transforms the problem to (5.9) and solves it using Matlab `quadprog` function iteratively till convergence. The termination conditions for the two algorithms are set to be the same. For the CFS algorithm, we record both the processing time for transforming the problem from (5.7) to (5.9) as well as the computation time for the resulting quadratic problem.

Consider navigation of mobile vehicles in indoor environments or autonomous driving in parking lots as shown in Fig.5.4a. In this case, $X = \mathbb{R}^2$ and $\mathcal{O}_q$ is a static maze for all $q$. We identify the configuration space with the Cartesian space and assume that the nonlinear constraint on vehicle dynamics is considered in the reference trajectory and will not be violated if the reference trajectory is slightly modified. The point cloud model of $\mathcal{O}_q$ is shown in Fig.5.4a. It is first partitioned into five polygons as shown in Fig.5.4b. The reference trajectory (shown as the solid line in Fig.5.4b) is computed using Lattice A* search [85]. Since the possible turning directions are discretized, there are undesirable oscillations in the reference trajectory. In the optimization problem, large penalty on acceleration is applied in order to git rid of the oscillatory waves. The convex feasible set in (5.9d) is illustrated in a time-augmented state space in Fig.5.4c where the z-axis is the time axis. At each time step, the convex feasible set is just a polygon around the reference point. All those polygons form a "tube" around the reference trajectory. The horizon of the problem is $h = 116$. Hence the dimension of the problem is 234. A safety margin is added as $d_{min} = 3$.

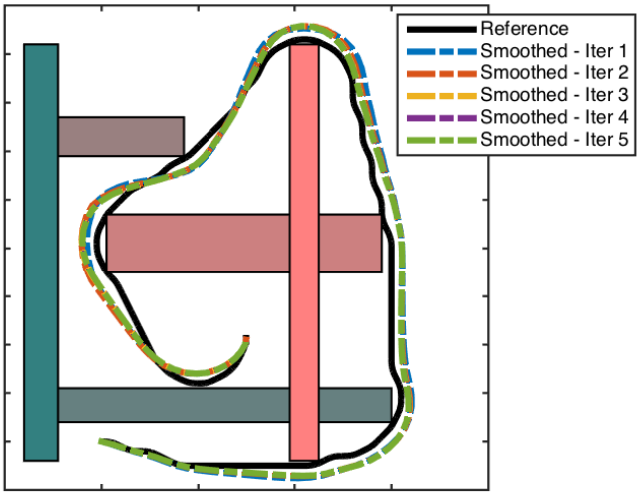
The CFS algorithm converges after 5 iterations, with total computation time 0.9935s. The smoothed trajectories for each iteration are shown in Fig.5.4b. The average time for transforming the problem from (5.7) to (5.9) during each iteration is 0.1632s. The average time for solving the optimization problem (5.7) is 0.0355s. The cost profile is $J(\mathbf{x}^r) = 423.7$, $J(\mathbf{x}^{(1)}) = 584.3$, $J(\mathbf{x}^{(2)}) = 455.0$, $J(\mathbf{x}^{(3)}) = 407.6$, $J(\mathbf{x}^{(4)}) = 403.7$, $J(\mathbf{x}^{(5)}) = 403.6$. However, it is worth noting that although the cost changes quite a lot from the first iteration, the resulting paths are similar to each other as shown in Fig.5.4b. The descent in cost is due to the adjustment of the velocity and acceleration profiles, e.g. redistributing sample points on the path.

The SQP algorithm does not converge or even find a feasible solution within the max function evaluation limit 5000 with computation time 387.2s. The SQP algorithm undergoes 20 iterations. When terminated, the cost is drops from 423.7 to 287.2, but the feasibility of the trajectory, i.e. $-\min_{q,i} d^*(x_q, \mathcal{O}_q^i)$, only goes from 3.1 to 2.3.

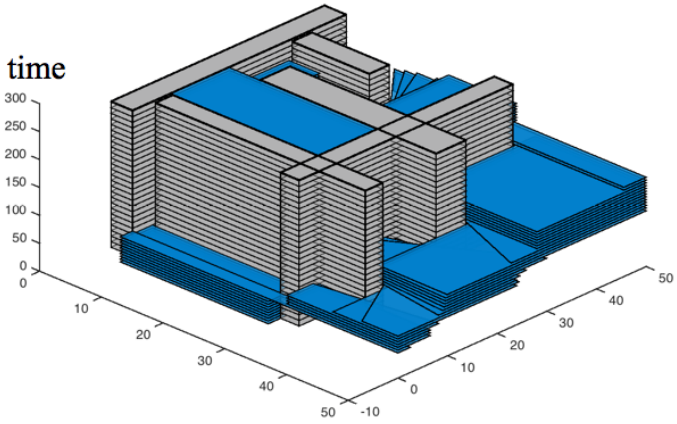
**Robot Arm** In this case, a three-link planar robot arm is considered as shown in Fig.5.5 with $X = [0, 2\pi)^3$. The infeasible area in the Cartesian space is wrapped by a capsule,



(a) The environment.

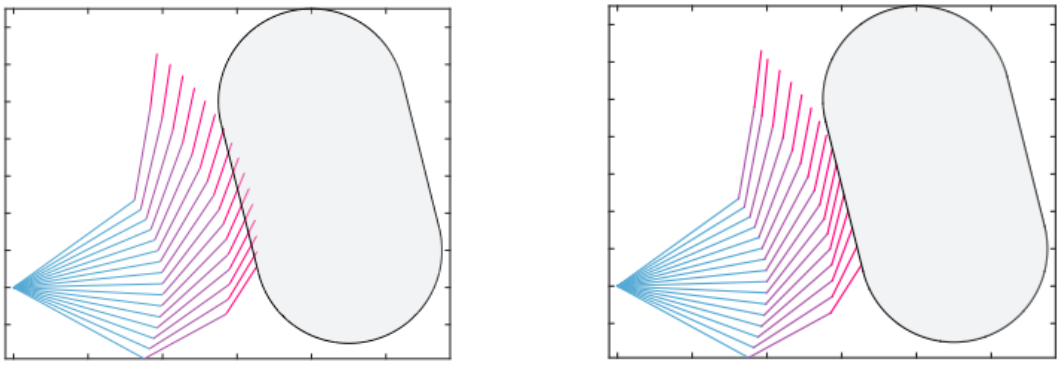



(b) The trajectories.




(c) The convex feasible set in time-augmented state space.

Figure 5.4: Trajectory smoothing for a mobile vehicle.



(a) Reference trajectory. (b) Smoothed trajectory.

Figure 5.5: Trajectory smoothing for a robot arm.

which characterizes $\mathcal{O}_q$ for all $q$. The reference trajectory is generated by linear interpolation between boundary conditions $\mathcal{A}$ and $\mathcal{Z}$, which violated the constraint. The horizon is $h = 15$.

The CFS algorithm converges after 2 iterations, with total computation time 2.4512s. The smoothed trajectory is shown in Fig.5.5b which is feasible. The average time for transforming the problem from (5.7) to (5.9) during each iteration is 1.2213s while the average time for solving the optimization problem is 0.0043s. The pre-processing time is much longer than that for the mobile vehicle due to the nonlinearity of the robot arm. Moreover, the solution in the first iteration is already good enough and the second iteration is just for verification that a local optimum has been found.

In comparison, the SQP algorithm converges after 41 iterations, with total computation time 5.34s. Both the CFS algorithm and the SQP algorithm converge to the local optima at $J = 616.7$.

**Areal Vehicle** In this case, an areal vehicle needs to fly over an uneven terrain. Thus $X = \mathbb{R}^3$ and $\mathcal{O}_q$ is the uneven terrain for all $q$, which is segmented into three cones. The reference trajectory is demonstrated by human, which is very coarse. The planning horizon is $h = 30$. The margin is $d_{min} = 1$.

CFS algorithm converges after two iterations with total time 0.1931s. The average time for pre-processing is 0.0730s and the average time for solving the optimization is 0.0235s. On the other hand, SQP converges to the same trajectory after 10 iterations with total time 1.5329s.

## 5.4 Optimization-Based Speed Profile Planning

Decoupling temporal planning from spatiotemporal planning is beneficial to reduce the computation complexity since the computation complexity grows exponentially when the

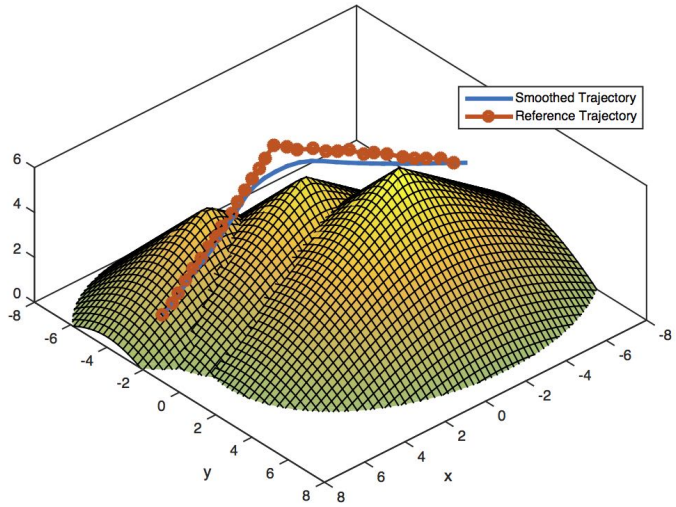



Figure 5.6: Trajectory smoothing for an areal vehicle.

dimension of the problem grows.

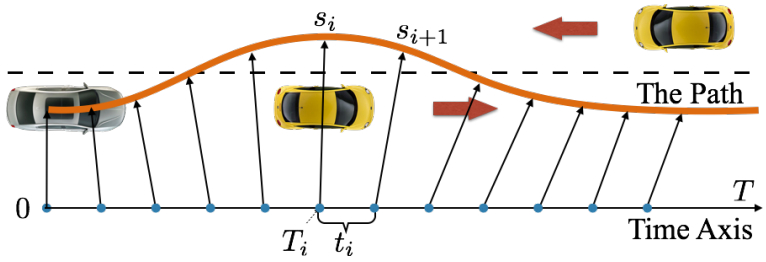
A speed profile is a one-to-one mapping between the time domain and the distance domain of the path. Optimization can be performed either over station as shown in Fig.5.7a or over time as shown in Fig.5.7b. Optimization over station requires an analytical parameterization of the path as discussed in [47, 48, 71, 112, 150]. However, globally continuous analytical parameterization of a complicated path is difficult, which usually requires approximation. For a curvy path, the approximation may introduce infeasibility, e.g. curvature exceeding the vehicle's kinematic limits. Moreover, the complexity of the optimization problem may increase when complicated expressions of the parameterization enter the objective function. On the other hand, if we optimize over time as discussed in [14], only a sequence of way points are needed instead of a continuous parameterization of the path. This section focuses on speed profile planning via temporal optimization which optimizes the time stamps for all waypoints along a given path.

In this section, the mathematical problem underlying speed profile planning will be discussed first by introducing the $s - T$ graph. Then the temporal optimization problem for speed profile generation will be formulated.

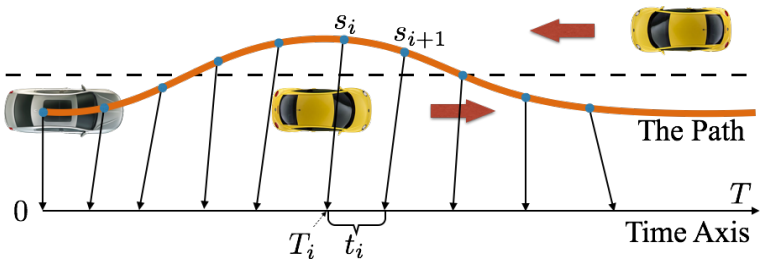
### 5.4.1 Problem Formulation

#### Representing Speed Profile Using the $s - T$ graph

Suppose a path $\mathbf{p}$ is given. There is no speed information in the path $\mathbf{p}$. In order to determine the speed at each station, the time axis need to be introduced. The $s - T$ graph is shown in Fig.5.8a where the vertical axis is the one dimensional parameterization of the path and the horizontal axis is the time axis. A speed profile $\mathcal{V}$ for the path $\mathbf{p}$ is denoted




(a) Optimization over station. For each sampled time step $T_i$, we need to find a corresponding station $s_i$ on the path.




(b) Optimization over time. For each sampled station $s_i$, we need to find a corresponding time stamp $T_i$ on the time axis.

Figure 5.7: Two optimization schemes to obtain the speed profile.

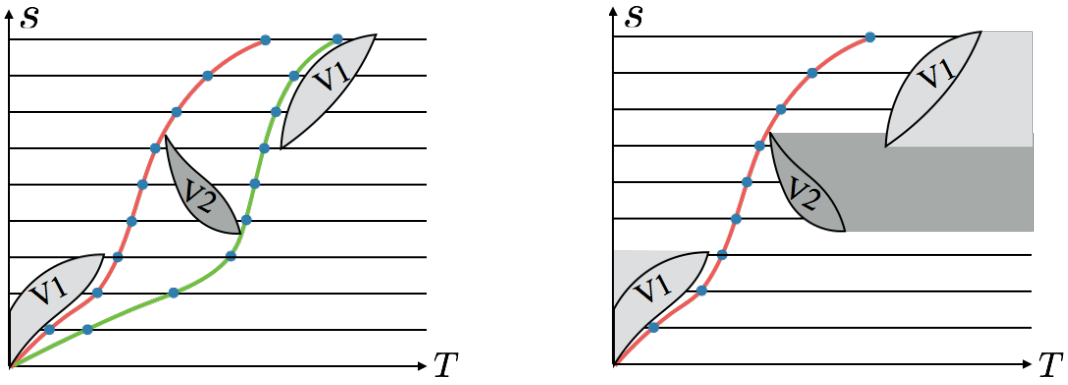
as a monotone curve on the $s-T$ graph as shown in Fig.5.8a. Since the curve is monotone, there are two ways to obtain the speed profile. The first way is to find a mapping from the $T$ axis to the $s$ axis, e.g. fix a sequence of time steps $\{T_i\}$ and find the desired $s_i$ for each $T_i$ as shown in Fig.5.7a. The second way is to find a mapping from the $s$ axis to the $T$ axis, e.g. fix a sequence of sampled stations $\{s_i\}$ and find the desired $T_i$ for each $s_i$ as shown in Fig.5.7b. The first way is the optimization over station as $s_i$'s are the decision variables, while the second way is the optimization over time as $T_i$'s are the decision variables. In either case, the speed profile is represented by the curve $\mathcal{V} = \{(T_i, s_i)\}_i$ in the $s-T$ graph.

### Computing Speed, Acceleration and Jerk

Consider the speed profile $\mathcal{V} = \{(T_i, s_i)\}_i$. Denote $p_i := \mathbf{p}(s_i)$ and the time interval between $p_i$ and $p_{i+1}$ as $t_i = T_{i+1} - T_i$. When the time interval $t_i$ and the distance between $p_i$ and $p_{i+1}$ are not too large, the velocity $v_i$, acceleration $a_i$ and jerk $j_i$ at $p_i$ can be approximated by

$$v_i = \frac{p_{i+1} - p_i}{t_i}, \quad a_i = \frac{2(v_i - v_{i-1})}{t_i + t_{i-1}}, \quad j_i = \frac{3(a_i - a_{i-1})}{t_i + t_{i-1} + t_{i-2}}. \tag{5.10}$$

Denote the heading of the vehicle at point $p_i$ as $\theta_i := \arctan(\dot{\mathbf{p}}(s_i))$. Hence the longitudinal and lateral directions at $p_i$ are denoted as $\tau(\theta_i) := [\cos\theta_i, \sin\theta_i]$ and $\eta(\theta_i) := [\sin\theta_i, -\cos\theta_i]$ respectively. Then the longitudinal and lateral velocity, acceleration and jerk




(a) Multiple topological trajectories. (b) One topological trajectory.

Figure 5.8: Illustration of the constraints and the topological trajectories on the $s-T$ graph.

are defined as

$$v_i^\tau = v_i \cdot \tau(\theta_i), v_i^\eta = v_i \cdot \eta(\theta_i), a_i^\tau = a_i \cdot \tau(\theta_i), a_i^\eta = a_i \cdot \eta(\theta_i), j_i^\tau = j_i \cdot \tau(\theta_i), j_i^\eta = j_i \cdot \eta(\theta_i). \quad (5.11)$$

Note that both $p_i$ and $\theta_i$ depend on station $s_i$. If we optimize over station, an analytical expression of $\mathbf{p}(s)$ is indispensable, which will also increase the complexity of the problem. On the other hand, if we optimize over time, $s_i$ is predefined and $p_i$ and $\theta_i$ are fixed for all $i$. The analytical expression of $\mathbf{p}(s)$ is not necessary. Moreover, in a lot of cases, the paths are represented by a sequence of points instead of an analytical expression. Hence temporal optimization is desired.

### Speed Profile Planning via Temporal Optimization

As discussed earlier, given a path that is represented by a list of points $\{p_i\}_1^{h+1}$, we need optimize the time stamps $T_i$'s for each point along the path. The safety constraints for the optimization problem is illustrated in the $s-T$ graph in Fig.5.8a. The shaded area represents the moments that another road participant is occupying some parts of the path (e.g. the conflict zone) so that the ego vehicle cannot enter. The constraints in Fig.5.8a are generated regarding the case shown in Fig.5.9a where the slow front vehicle is denoted as V1 and the vehicle in the opposite lane is denoted as V2. As the ego vehicle can choose to enter the conflict zone before or after another road participant, there are more than one topological trajectory (or homotopy class), hence more than one local optimum as shown in Fig.5.8a. The red trajectory corresponds to the case that the ego vehicle overtook V1 before V2 passed by, while the green trajectory corresponds to the case that the ego vehicle overtook V1 after V2 went away. It is assumed that which homotopy class to choose is determined by a high level planner. In temporal optimization, the speed profile planner only tries to find the local optimum inside the chosen homotopy class as shown in Fig.5.8b. This implies that for each

point $p_i$, the high level planner would specify an interval constraint $[T_i^{min}, T_i^{max}]$ for the time stamp $T_i$. Hence the optimization problem is formulated as

$$\min_{\mathbf{t}} \; J_1(\mathbf{t}) + J_2(\mathbf{u}), \tag{5.12a}$$

$$s.t.\; T_i \in [T_i^{min}, T_i^{max}], \tag{5.12b}$$

$$|u_i| \leq \bar{u}, \tag{5.12c}$$

$$\mathcal{F}(\mathbf{t}, \mathbf{u}) = 0. \tag{5.12d}$$

where $\mathbf{t} := [t_1, \cdots, t_h]$, $\mathbf{u} := [u_1, \cdots, u_h]$ and $u_i := [a_i^\tau, a_i^\eta, j_i^\tau, j_i^\eta, v^r - v_i^\tau]$[1]. For simplicity, let $u_i^j \in \mathbb{R}$ be the $j$-th entry in $u_i$. Equation (5.12a) is the cost function, which penalizes cycle time for efficiency in $J_1$ and speed, acceleration and jerk for smoothness of the speed profile in $J_2$. $J_1$ and $J_2$ are convex. Moreover, $J_2$ is symmetric with respect to 0. Equation (5.12b) is the safety constraint to avoid dynamic obstacles. Equation (5.12c) is the dynamic constraint on the speed, acceleration and jerk, which is assumed to be a box constraint. Equation (5.12d) encodes the relationship between the time stamps and the speed profile in (5.10-5.11). Indeed, (5.12d) can be written as a sequence of affine equations, $f_i^j(\mathbf{t}) + h_i^j(\mathbf{t})u_i^j = 0$ for all $i, j$ where

$$\begin{aligned}
f_i^1(\mathbf{t}) &= 2\left[t_i dp_{i-1}^{\tau_i} - t_{i-1} dp_i^{\tau_i}\right], \\
h_i^1(\mathbf{t}) &= t_i t_{i-1}(t_i + t_{i-1}), \\
f_i^2(\mathbf{t}) &= 2\left[t_i dp_{i-1}^{\eta_i} - t_{i-1} dp_i^{\eta_i}\right], \\
h_i^2(\mathbf{t}) &= t_i t_{i-1}(t_i + t_{i-1}), \\
f_i^3(\mathbf{t}) &= 6[(t_{i-1} + t_{i-2})t_{i-1}t_{i-2}dp_i^{\tau_i} - (t_i + 2t_{i-1} + t_{i-2})t_i t_{i-2} dp_{i-1}^{\tau_i} + (t_i + t_{i-1})t_i t_{i-1} dp_{i-2}^{\tau_i}], \\
h_i^3(\mathbf{t}) &= t_i t_{i-1} t_{i-2}(t_i + t_{i-1})(t_i + t_{i-1} + t_{i-2}), \\
f_i^4(\mathbf{t}) &= 6[(t_{i-1} + t_{i-2})t_{i-1}t_{i-2}dp_i^{\eta_i} - (t_i + 2t_{i-1} + t_{i-2})t_i t_{i-2} dp_{i-1}^{\eta_i} + (t_i + t_{i-1})t_i t_{i-1} dp_{i-2}^{\eta_i}], \\
h_i^4(\mathbf{t}) &= t_i t_{i-1} t_{i-2}(t_i + t_{i-1})(t_i + t_{i-1} + t_{i-2}), \\
f_i^5(\mathbf{t}) &= v^r t_i - dp_i^{\tau_i}, \\
h_i^5(\mathbf{t}) &= t_i,
\end{aligned}$$

where $dp_i = p_{i+1} - p_i$, $dp_i^{\tau_j} = dp_i \cdot \tau(\theta_j)$, and $dp_i^{\eta_j} = dp_i \cdot \eta(\theta_j)$. By definition, $h_i^j \geq 0$ for all $i$ and $j$. This property will be exploited to relax the problem. In order to compute the acceleration and jerk at $i = 1, 2$, constants $p_0$, $p_{-1}$, $t_0$ and $t_{-1}$ are defined according to the initial velocity $v_0$ and the initial acceleration $a_0 = 0$.

### 5.4.2 Quadratic Approximation

To solve (5.12) efficiently, we exploit its quadratic approximation using the slack convex feasible set (SCFS) algorithm. As will be discussed in Chapter 6, the SCFS algorithm is

[1] $u_i$ may contain other parameters depending on the objective of the problem.

designed for problems with convex costs and nonlinear equality constraints. The idea is to 1) relax the nonlinear equality constraints to a set of non degenerating nonlinear inequality constraints using slack variables by exploiting symmetry of the problem, and 2) approximate the relaxed problem using quadratic programs. A set of nonlinear inequality constraints are non degenerating if they do not imply any nonlinear equality constraint.

Assuming $J_1$ and $J_2$ are quadratic. Since (5.12) is symmetric with respect to $\mathbf{u}$ in the cost function (5.12a) and in the dynamic constraints (5.12c), we define $\mathbf{y} \geq |\mathbf{u}|$ to be the slack variable. Then the relaxed problem is as

$$\min_{\mathbf{t},\mathbf{y}} \; J_1(\mathbf{t}) + J_2(\mathbf{y}), \tag{5.13a}$$

$$s.t. \; A\mathbf{t} \leq b, y_i \leq \bar{u}, \forall i, \forall j = 1, 2, \tag{5.13b}$$

$$f_i^j(\mathbf{t}) + h_i^j(\mathbf{t}) y_i^j \geq 0, f_i^j(\mathbf{t}) - h_i^j(\mathbf{t}) y_i^j \leq 0. \tag{5.13c}$$

where

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 1 & \cdots & 1 & 1 \\ -1 & 0 & \cdots & 0 \\ -1 & -1 & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ -1 & \cdots & -1 & -1 \end{bmatrix}, b = \begin{bmatrix} T_1^{max} \\ \vdots \\ T_h^{max} \\ -T_1^{min} \\ \vdots \\ -T_h^{min} \end{bmatrix}.$$

According to Proposition 6.10 in Chapter 6, (5.13) is equivalent to (5.12) in the sense that: if $(\mathbf{t}^o, \mathbf{y}^o)$ is a local optimum of (5.13), then $\mathbf{t}^o$ is a local optimum of (5.12); and if $\mathbf{t}^o$ is a local optimum of (5.12), then $(\mathbf{t}^o, \mathbf{y}^o)$ is a local optimum of (5.13) with $(y_i^j)^o := |[h_i^j(\mathbf{t}^o)]^{-1} f_i^j(\mathbf{t}^o)|$.

The relaxed problem is solved iteratively. Suppose at the $k$-th iteration, we have the solution $\mathbf{t}^{(k)}$ and $\mathbf{y}^{(k)}$. Then at the $(k+1)$-th iteration, (5.13c) is linearized with respect to $\mathbf{t}^{(k)}$ and $\mathbf{y}^{(k)}$, and the following approximated quadratic program is formulated,

$$\min_{\mathbf{t},\mathbf{y}} \; J_1(\mathbf{t}) + J_2(\mathbf{y}), \tag{5.14a}$$

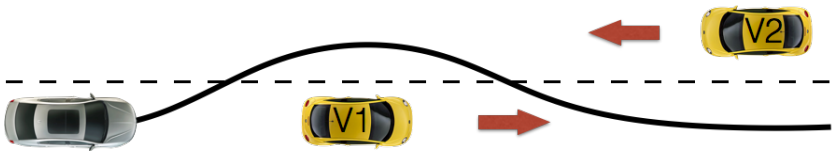
$$s.t. \; A\mathbf{t} \leq b, y_i \leq \bar{u}, \forall i, \forall j = 1, 2, \tag{5.14b}$$

$$\left[\nabla_{\mathbf{t}} f_i^j(\mathbf{t}^{(k)}) + \nabla_{\mathbf{t}} h_i^j(\mathbf{t}^{(k)})(y_i^j)^{(k)}\right](\mathbf{t} - \mathbf{t}^{(k)}) + h_i^j(\mathbf{t}^{(k)}) y_i^j + f_i^j(\mathbf{t}^{(k)}) \geq 0, \tag{5.14c}$$

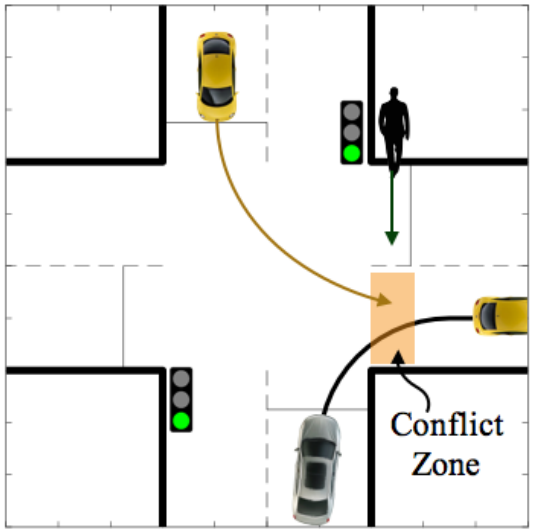
$$\left[\nabla_{\mathbf{t}} f_i^j(\mathbf{t}^{(k)}) - \nabla_{\mathbf{t}} h_i^j(\mathbf{t}^{(k)})(y_i^j)^{(k)}\right](\mathbf{t} - \mathbf{t}^{(k)}) - h_i^j(\mathbf{t}^{(k)}) y_i^j + f_i^j(\mathbf{t}^{(k)}) \leq 0. \tag{5.14d}$$

Solving the above quadratic program, we obtain $\mathbf{t}^o$ and $\mathbf{y}^o$. Define $\mathbf{t}^{(k+1)} := \mathbf{t}^o$ and $(y_i^j)^{(k+1)} := |[h_i^j(\mathbf{t}^o)]^{-1} f_i^j(\mathbf{t}^o)|$. Then we iterate until the solution converges, e.g. $\|\mathbf{t}^{(k+1)} - \mathbf{t}^{(k)}\| \leq \epsilon$ for $\epsilon$ small. $\mathbf{t}^{(0)}$ and $\mathbf{y}^{(0)}$ is initialized as

$$t_i^{(0)} := \frac{dp_i^{\tau_i}}{v^*}, (y_i^j)^{(0)} := |[h_i^j(\mathbf{t}^{(0)})]^{-1} f_i^j(\mathbf{t}^{(0)})|, \tag{5.15}$$




(a) Case 1: Overtake a slow front vehicle using the opposite lane.




(b) Case 2: Right turn at an intersection.

Figure 5.9: Urban driving scenarios.

where $v^*$ can be the reference $v^r$ or the initial speed $v_0^\tau$.

### 5.4.3 Example: Speed Profile Planning for Autonomous Vehicles

Autonomous driving is widely viewed as a promising technology to revolutionize today's transportation system. However, it is still challenging to plan collision-free, time-efficient and comfortable trajectories for an automated vehicle in dynamic environments such as urban roads, since the vehicle needs to interact with other road participants. For example, how to overtake a slow front vehicle safely using the opposite lane as shown in Fig.5.9a and how to turn safely at an intersection when pedestrians are crossing and vehicles in the opposite lane are turning left as shown in Fig.5.9b.

For vehicles, the path $\mathbf{p} \subset \mathbb{R}^2$ concerns with the position of the center of the rear axle. Regarding the driving quality, $a^\tau$ and $j^\tau$ should be minimized for longitudinal comfort, while $a^\eta$ and $j^\eta$ should be minimized for lateral comfort. Moreover, a reference speed $v^r$ should be tracked for time efficiency. Hence, the cost function in (5.12a) is designed to be

$$J_1(\mathbf{t}) = 0, J_2(\mathbf{u}) = w_1 \sum \|a_i^\tau\|^2 + w_2 \sum \|a_i^\eta\|^2 + w_3 \sum \|j_i^\tau\|^2, \tag{5.16}$$

where $w_1 = 1$ and $w_2 = w_3 = w_4 = w_5 = 10$ are weights. The dynamic constraint in (5.12c) is designed to be

$$|a_i^\tau| \leq \bar{a}, |a_i^\eta| \leq \bar{a}, \tag{5.17}$$

for $\bar{a} := 2.5m/s^2$ for passenger comfort.

The performance of the proposed method will be illustrated in several scenarios including the two cases in Fig.5.9. The simulations were run in Matlab on a MacBook of 2.3 GHz using Intel Core i7. In the SCFS algorithm, (5.14) for each iteration was solved using the `quadprog` function. For comparison, (5.12) was also solved using the SQP in the `fmincon` function. The SCFS algorithm and the SQP algorithm terminated if the step size (difference between the two consecutive solutions) was less than $10^{-6}$. For better illustration, the speed profiles after every iteration are shown in grayscale (the later in the iteration, the darker) together with the optimal speed profile in all three cases.

**Case 0: Speed Profile for a Curvy Road** In this scenario, the automated vehicle needs to pass a curvy road shown in Fig.5.10. The reference speed is $v_r = 11m/s$. The vehicle's initial speed is $v_0^\tau = 11m/s$ and $v_0^\eta = 0m/s$. The path is sampled every 2m and 35 points are chosen. The SCFS algorithm (iteratively solving (5.14)) converges after 5 steps with total computation time 0.185s. The optimal speed, acceleration and jerk profiles are shown in Fig.5.10. The horizontal axes in the plots represents the traveling distance along the path. The vehicle decelerated first in order to meet the acceleration constraint in the lateral direction. It then accelerated after the maximum curvature was reached. The SQP algorithm converges to the same speed profile after 80 iterations with total computation time 54.479s.

**Case 1: Speed Profile for Overtake** The scenario is shown in Fig.5.9a where the automated vehicle wants to overtake the slow front vehicle using the opposite lane. The reference speed is $v_r = 11m/s$. The vehicle's initial speed is $v_0^\tau = 10m/s$ and $v_0^\eta = 0m/s$. $\mathbf{t}^{(0)}$ is initialized using $v_0^\tau$. The path is sampled every 2m and 34 points are chosen. The SCFS algorithm converges at iteration 5 with computation time 0.308s. The optimal speed, acceleration and jerk profiles are shown in Fig.5.11. The horizontal axis in the plots represents the traveling distance along the lane. The corresponding time stamps for all stations are shown in Fig.5.12. Snapshots are also shown in Fig.5.12 where the gray rectangle represents the ego vehicle and the yellow rectangles are the surrounding vehicles. The ego vehicle slowed down first to keep a safe headway from the front vehicle V1. When it changed to the adjacent lane, it speeded up to overtake V1. Before the vehicle V2 in the opposite direction came, the ego vehicle went back to its lane. The optimal speed profile is on the boundary of the safety constraint as shown in Fig.5.12 and on the boundary of the feasibility constraint as shown in the acceleration profile in Fig.5.11. For comparison, SQP method converges to the same optimum after 48 iterations with computation time 28.513s.

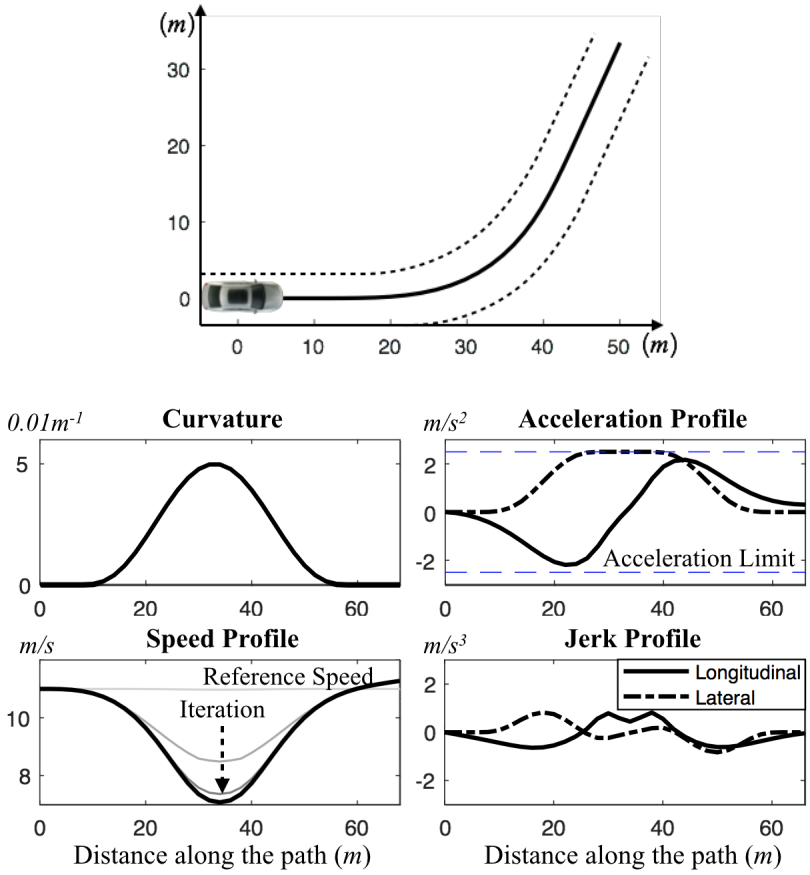



Figure 5.10: Speed profile planning for driving on a curvy road.

**Case 2: Speed Profile for Right Turn** The scenario is illustrated in Fig.5.9b where the automated vehicle tries to turn right in green light when a vehicle in the opposite direction turns left and a pedestrian is crossing the street. The reference speed is $v_r = 5m/s$. The vehicle's initial speed is $v_0^\tau = 2.5m/s$ and $v_0^\eta = 0m/s$. $\mathbf{t}^{(0)}$ is initialized using $v_0^\tau$. The path is sampled every 0.5m and 36 points are chosen. The strategy determined by the high level planner for the ego vehicle is to pass the conflict zone after the pedestrian at $T = 5s$ and before the left-turn vehicle at $T = 6s$. To meet the safety constraint, the ego vehicle slowed down to yield the pedestrian and speeded up to pass the conflict zone before the left-turn vehicle, as shown in the speed profile in Fig.5.13a. For comparison, the optimal speed profile without the safety constraint is shown in Fig.5.13b where the ego vehicle turned smoothly. The SCFS algorithm converges after 8 iterations with computation time 0.522s. The SQP algorithm does not converge before the maximum number of iterations 100 is reached.

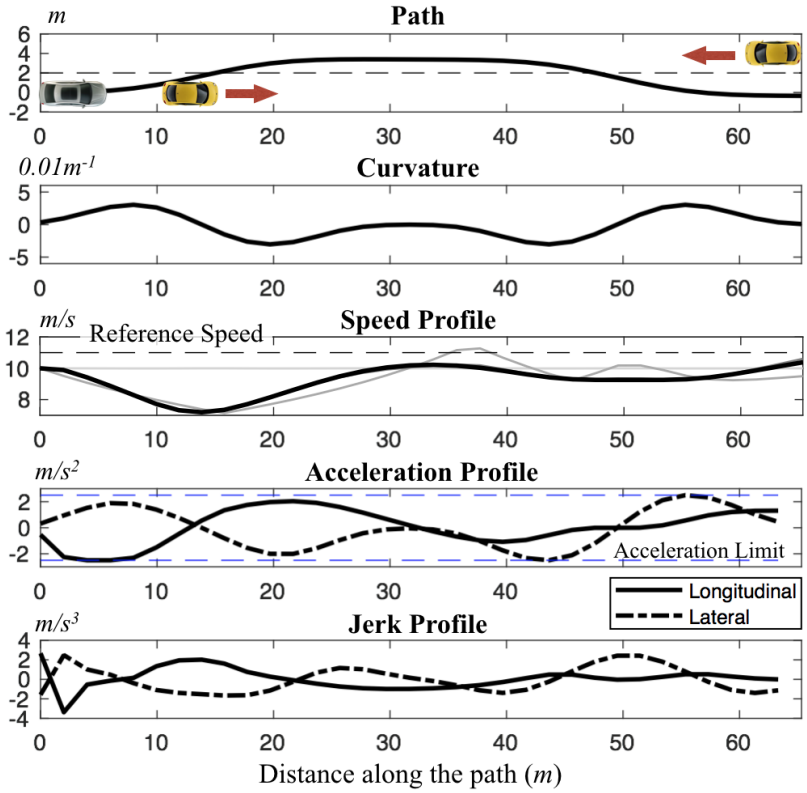



Figure 5.11: Speed profile planning for overtake.

## 5.5 Conclusion

In this chapter, we discussed the optimization-based methods for global motion planning, e.g. optimization-based trajectory planning in an integrated framework and optimization-based speed profile planning in a layered framework. The CFS algorithm and SCFS algorithm are adopted to transform the non-convex optimization problems into quadratic programs. The properties of the two algorithms will be discussed in Chapter 6.

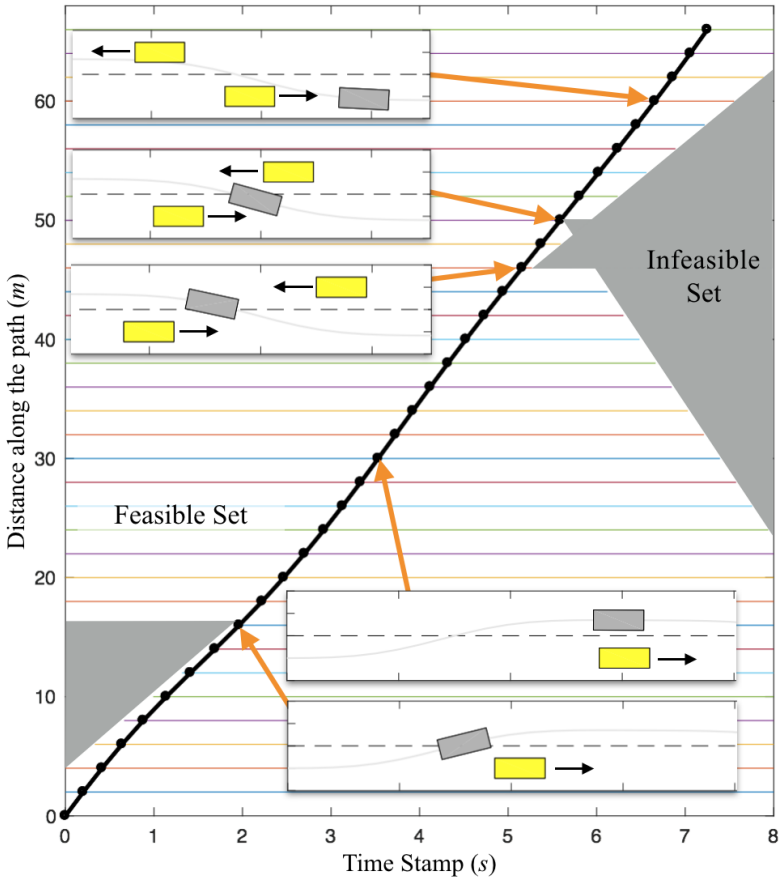
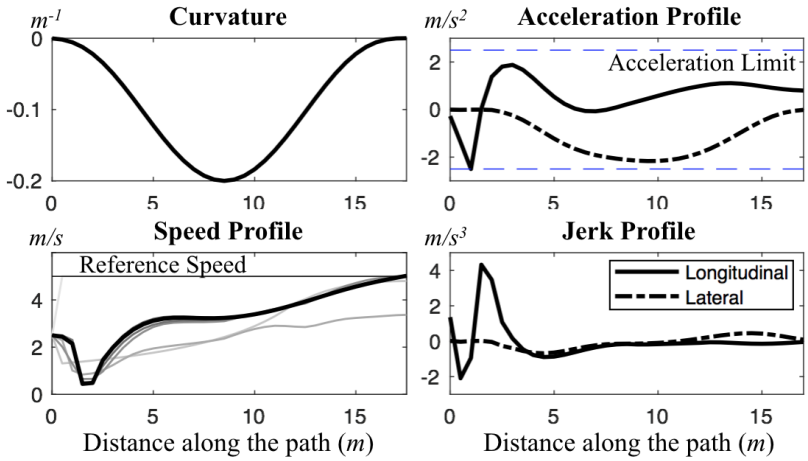
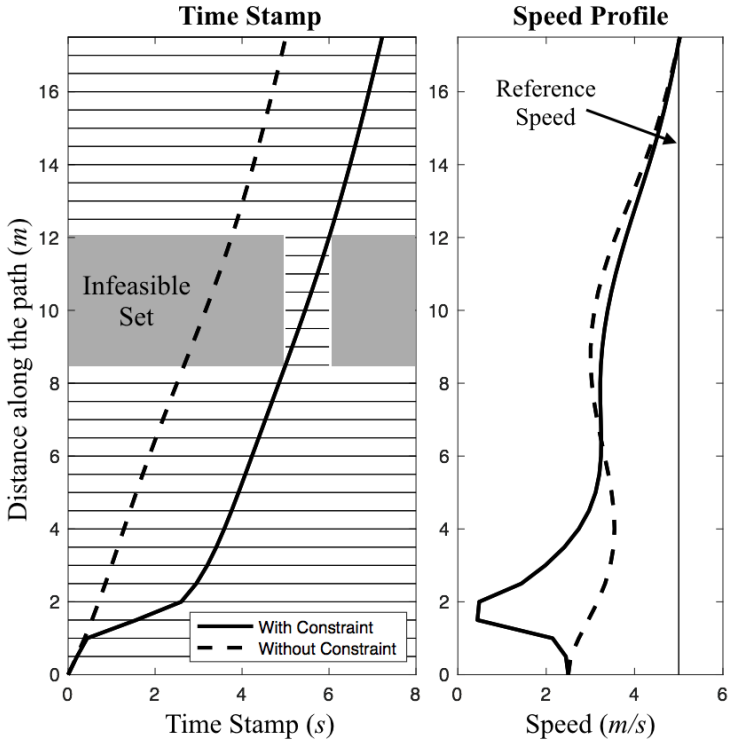



Figure 5.12: The optimal time stamps for overtake.




(a) The speed profile.




(b) Comparison between the speed profiles with and without the temporal constraint.

Figure 5.13: Speed profile planning for turning at intersection.

# Chapter 6

# Real-Time Numerical Optimization

Real time trajectory optimization is critical for robotic systems to achieve full autonomy. Due to nonlinear system dynamics and obstacles in the environment, the optimization problems for motion planning, such as (5.7) and (5.12), are highly nonlinear and non-convex, hence hard to be computed in real time. In this chapter, we discuss the convex feasible set algorithm (CFS) and the slack convex feasible set algorithm (SCFS) for efficient computation of the motion planning problems by exploiting the geometric structure of the problem. The CFS algorithm is for non-convex problems on linear space. The SCFS algorithm is for non-convex problems with on nonlinear space. The benchmark problem, the algorithm, the theoretical results and the applications will be discussed for each method.

## 6.1 Non-Convex Optimization on Linear Space

### 6.1.1 The Benchmark Problem

Consider an optimization problem with a convex cost function but non-convex constraints, i.e.

$$\min_{\mathbf{x}\in\Gamma} J(\mathbf{x}), \tag{6.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the decision variable and the problem follows two assumptions.

**Assumption 6.1** (Cost)**.** *$J : \mathbb{R}^n \to \mathbb{R}^+$ is smooth, strictly convex.*

**Assumption 6.2** (Constraint)**.** *The set $\Gamma \subset \mathbb{R}^n$ is connected and complete, while its complements $\Gamma^c := \mathbb{R}^n \setminus \Gamma$ is a disjoint collection of simply-connected open sets with piecewise smooth boundaries and disjoint closures. For every point $x$ on the boundary $\partial\Gamma$, there exists an $n$-dimensional convex polytope $P \subset \Gamma$ such that $x \in P$.*

Assumption 6.1 implies that $J$ is radially unbounded, i.e. $J(\mathbf{x}) \to \infty$ when $\|\mathbf{x}\| \to \infty$. Assumption 6.2 specifies the geometric features of the feasible set $\Gamma$, where the first part deals with the topological features of $\Gamma$ and $\Gamma^c$ and the second part ensures that the constraint

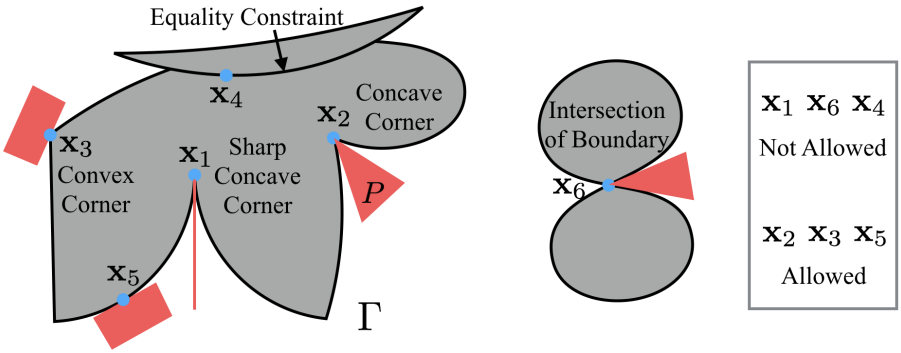


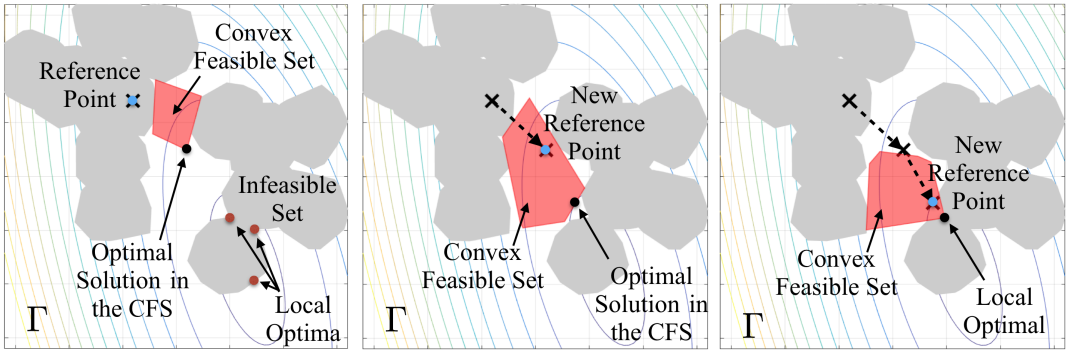

Figure 6.1: Illustration of Assumption 6.2, the geometric features of the constraint $\Gamma$.

can be locally convexified. Intuitively, Assumption 6.2 excludes (1) local and global equality constraints[1], (2) sharp concave corners[2] in $\Gamma^c$ and (3) intersecting boundaries, which are illustrated in Fig.6.1. In the figure, $\Gamma$ includes the white area and the black boundaries. When there is an equality constraint at $\mathbf{x} \in \partial\Gamma$, e.g. $\mathbf{x}_4$ in Fig.6.1, we cannot find an $n$-dimensional convex polytope $P \subset \Gamma$ with $\mathbf{x} \in P$ since the dimension of $\Gamma$ is less than $n$ locally. When there are sharp concave corners in the infeasible set, e.g. $\mathbf{x}_1$ in Fig.6.1, the dimension of the convex feasible polytope is strictly less than $n$. On the other hand, the desired polytopes (shown in red) can be defined at non-sharp concave corners such as $\mathbf{x}_2$, convex corners such as $\mathbf{x}_3$ and smooth boundary points such as $\mathbf{x}_5$. When the boundary intersects with itself such as in $\mathbf{x}_6$, it is possible to find such polytopes. But the 8-shaped set consists of two disjoint simply-connected open sets whose closures are no longer disjoint, which violates the first part of Assumption 6.2. Hence intersecting boundaries are not considered in this chapter. We will relax this assumption in our future work.

The geometric structure of problem (6.1) is illustrated in Fig.6.2a. The contour represents the cost function $J$, while the gray parts represent $\Gamma^c$. There are two disjoint components in $\Gamma^c$. The goal is to find a local optimum (hopefully global optimum) starting from the initial reference point (blue dot). As shown in Fig.6.2a, the problem is highly non-convex and the non-convexity mainly comes from the constraint. To make the computation more efficient, we propose the convex feasible set algorithm in this section, which transforms the problem into a sequence of convex optimizations by obtaining a sequence of convex feasible sets inside the non-convex domain $\Gamma$. As shown in Fig.6.2, the idea is implemented iteratively. At current iteration, a convex feasible set for the current reference point (blue dot) is obtained. The optimal solution in the convex feasible set (black dot) is set as the reference point for the next iteration. The formal mathematical description of this algorithm will be discussed in

[1]We say a point $\mathbf{x} \in \Gamma$ in on a local equality constraint when the dimension of the neighborhood of $\mathbf{x}$ is less than $n$. If $\Gamma$ is on a $m$-dimensional manifold with $m < n$, we then call it a global equality constraint.

[2]A sharp concave corner is a concave corner such that some pieces of boundary are tangent to each other locally.




(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

Figure 6.2: Geometry of problem 6.1 and the idea of the convex feasible set algorithm.

Section 6.1.2. The feasibility of this method, i.e. the existence of an $n$-dimensional convex feasible set, is implied by Assumption 6.2. Nonetheless, in order to compute the convex feasible set efficiently, we still need an analytical description of the constraint.

Analytically, $\Gamma$ can be represented by $N$ continuous and piecewise smooth functions $\phi_i : \mathbb{R}^n \to \mathbb{R}$, e.g.

$$\Gamma = \bigcap_i \{\mathbf{x} : \phi_i(\mathbf{x}) \geq 0\} = \bigcap_i \Gamma_i, \tag{6.2}$$

where $\Gamma_i := \{\mathbf{x} : \phi_i(\mathbf{x}) \geq 0\}$. $\phi_i$ can be regarded as a potential function of $\Gamma_i$ where the boundary and the interior of $\Gamma_i$ satisfy that $\partial\Gamma_i = \{\mathbf{x} : \phi_i(\mathbf{x}) = 0\}$ and $\Gamma_i^o = \{\mathbf{x} : \phi_i(\mathbf{x}) > 0\}$. Note that $\Gamma_i^c$'s are not required to be disjoint and $N$ should be greater than or equal to the number of disjoint components in $\Gamma^c$. The decomposition from $\Gamma$ to $\Gamma_i$'s is not unique. Neither is the function $\phi_i$ that represent $\Gamma_i$. In many cases, $\phi_i$ can be chosen as a signed distance function to $\partial\Gamma_i$. Since $\phi_i$ is only piecewise smooth, it may not be differentiable at all points. For any $v \in \mathbb{R}^n$, define the directional derivative $\partial_v$[3] as

$$\partial_v \phi_i(\mathbf{x}) := \lim_{a \to 0^+} \frac{\phi_i(\mathbf{x} + av) - \phi_i(\mathbf{x})}{a}. \tag{6.3}$$

Let $\mathcal{S}(\phi_i, \mathbf{x}) := \{v \in \mathbb{R}^n : \partial_v \phi_i(\mathbf{x}) + \partial_{-v} \phi_i(\mathbf{x}) = 0\}$ denote all the smooth directions of function $\phi$ at point $\mathbf{x}$. Define the sub-differential of $\phi_i$ at $\mathbf{x}$ as

$$D\phi_i(\mathbf{x}) := \{d \in \mathbb{R}^n : d \cdot v \leq \partial_v \phi_i(\mathbf{x}), \forall v \in \mathbb{R}^n\}. \tag{6.4}$$

This definition is valid, i.e. the right hand side of (6.4) is non empty under Assumption 6.3 below and will be justified later. The elements in $D\phi_i(\mathbf{x})$ are called sub-gradients. When $\phi_i$ is smooth at $\mathbf{x}$, $D\phi_i(\mathbf{x})$ reduces to a singleton set which contains only the gradient $\nabla\phi_i(\mathbf{x})$.

[3]Note that $\partial$ means boundary when followed by a set, e.g. $\partial\Gamma$. It means derivative when followed by a function, e.g. $\partial_v \phi_i$.

The definition (6.4) follows from Clarke (generalized) sub-gradients for non-convex functions [27]. Nonetheless, we make the following assumption on $\phi_i$'s.

**Assumption 6.3** (Regularity)**.** *(1) $\phi_i$ is semi-convex for all $i$, i.e. there exists a positive semi-definite $H_i^* \in \mathbb{R}^{n\times n}$ such that for any $\mathbf{x}, v \in \mathbb{R}^n$,*

$$\phi_i(\mathbf{x}+v) - 2\phi_i(\mathbf{x}) + \phi_i(\mathbf{x}-v) \geq -v^T H_i^* v. \tag{6.5}$$

*(2) $0 \notin D\phi_i(\mathbf{x})$ if $\mathbf{x} \in \partial\Gamma_i$ or if $D\phi_i(\mathbf{x})$ is a singleton set. (3) for any $\mathbf{x}$ such that $I := \{i : \phi_i(\mathbf{x}) = 0\} \neq \emptyset$, there exists $v \in \mathbb{R}^n$ such that $\partial_v\phi_i(\mathbf{x}) < 0$ for all $i \in I$.*

**Lemma 6.1** (Properties of Semi-Convex Functions)**.** *If $\phi_i$ is semi-convex, for any $\mathbf{x}, v, v_1, v_2 \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that $v = v_1 + v_2$, the following inequalities hold,*

$$0 \leq \partial_v\phi_i(\mathbf{x}) + \partial_{-v}\phi_i(\mathbf{x}), \tag{6.6}$$

$$b\partial_v\phi_i(\mathbf{x}) \leq \partial_{bv}\phi_i(\mathbf{x}), \tag{6.7}$$

$$\partial_v\phi_i(\mathbf{x}) \leq \partial_{v_1}\phi_i(\mathbf{x}) + \partial_{v_2}\phi_i(\mathbf{x}). \tag{6.8}$$

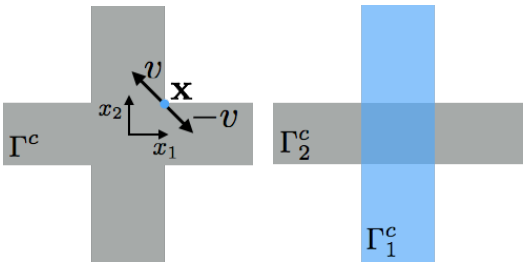
*The equalities in (6.6) and (6.7) are achieved when $\phi_i(\mathbf{x})$ is smooth at direction $v$. The equality in (6.8) is achieved when $\phi_i(\mathbf{x})$ is smooth at directions $v_1$ and $v_2$. Moreover, for any unit vector $w \in \mathbb{R}^n$, $\partial_w\phi_i(\mathbf{x})$ is locally bounded.*

Lemma 6.1 is proved in [76], which justifies the definition in (6.4). By (6.7) and (6.8), for any $v_1, v_2 \in \mathcal{S}(\phi_i, \mathbf{x})$, $a, b \in \mathbb{R}$ and $v = av_1 + bv_2$, $\partial_v\phi_i(\mathbf{x}) = a\partial_{v_1}\phi_i(\mathbf{x}) + b\partial_{v_2}\phi_i(\mathbf{x})$ and $\partial_v\phi_i(\mathbf{x}) + \partial_{-v}\phi_i(\mathbf{x}) = 0$. Hence $v \in \mathcal{S}(\phi_i, \mathbf{x})$. We can conclude that 1) $\mathcal{S}(\phi_i, \mathbf{x})$ is a linear subspace of $\mathbb{R}^n$ and 2) the function induced by the directional derivative $v \mapsto \partial_v\phi_i(\mathbf{x})$ is a sub-linear function[4] on $\mathbb{R}^n$ and a linear function on $\mathcal{S}(\phi_i, \mathbf{x})$. By Hahn-Banach Theorem [37], there exists a vector $d \in \mathbb{R}^n$ such that $d \cdot v = \partial_v\phi_i(\mathbf{x})$ for $v \in \mathcal{S}(\phi_i, \mathbf{x})$ and $d \cdot v \leq \partial_v\phi_i(\mathbf{x})$ for $v \in \mathbb{R}^n$. Moreover, as the directional derivative is bounded, the sub-gradients are also bounded. Hence the definition in (6.4) is justified.

Geometrically, Assumption 6.3 implies that there cannot be any concave corners in $\Gamma_i^c$ or convex corners in $\Gamma_i$. Suppose $\Gamma_i^c$ has a concave corner at $\mathbf{x} \in \partial\Gamma_i$. Since $0 \notin D\phi_i(\mathbf{x})$, we can choose a unit vector $v$ such that $\partial_v\phi_i < 0$ and $\partial_{-v}\phi_i < 0$ as shown in Fig.6.3a. Then (6.6) is violated, which contradicts with the assumption on semi-convexity. Nonetheless, concave corners are allowed in $\Gamma^c$, but should only be formulated by a combination of several intersecting $\Gamma_i^c$'s as shown in Fig.6.3. Those $\Gamma_i^c$'s should be intersecting due to the third statement of Assumption 6.3. In the example, the set $\Gamma = \{\mathbf{x} = (x_1, x_2) : \min(|x_1| - 1, |x_2| - 1) \geq 0\}$ is partitioned into two sets $\Gamma_1 = \{\mathbf{x} : |x_1| - 1 \geq 0\}$ and $\Gamma_2 = \{\mathbf{x} : |x_2| - 1 \geq 0\}$. Both $\phi_1 = |x_1| - 1$ and $\phi_2 = |x_1| - 1$ satisfies Assumption 6.3. Without the partition, $\phi = \min(|x_1| - 1, |x_2| - 1)$ violates the condition on semi-convexity[5]. A method to partition

[4] A function $f$ is called sub-linear if it satisfies positive homogeneity $f(ax) = af(x)$ for $a > 0$, and sub-additivity $f(x + y) \leq f(x) + f(y)$.

[5] Let $\mathbf{x} = (1, 1)$ and $v = (\cos\frac{\pi}{4}, \sin\frac{\pi}{4})$. Then $\phi_i(\mathbf{x}+av) - 2\phi_i(\mathbf{x}) + \phi_i(\mathbf{x}-av) = -a\cos\frac{\pi}{4} - a\sin\frac{\pi}{4} = -\sqrt{2}a$, which can not be greater than any $-a^2 v^T H_i^* v$ when $a$ is small.




(a) The constraint $\Gamma$ (b) One partition of $\Gamma$

Figure 6.3: Representing $\Gamma$ using $\Gamma_i$ and $\phi_i$.

the obstacles and construct the desired $\phi_i$'s is discussed in [87]. It is our hypothesis that any set $\Gamma$ that satisfies Assumption 6.2 can be partitioned into $\Gamma_i$'s and represented by $\phi_i$'s that satisfy Assumption 6.3, which will be verified in our future work.

Assumption 6.3 will be exploited in computing the convex feasible set in Section 6.1.2. Lemma 6.1 will be used in the proofs in Section 6.1.3.

### 6.1.2 The Convex Feasible Set Algorithm

To solve the problem (6.1) efficiently, we propose the convex feasible set algorithm. A convex feasible set $\mathcal{F}$ for the set $\Gamma$ is a convex set such that $\mathcal{F} \subset \Gamma$. $\mathcal{F}$ is not unique. We define the desired $\mathcal{F}$ in Section 6.1.2. As $\Gamma$ can be covered by several (may be infinitely many) convex feasible sets, we can efficiently search the non-convex space $\Gamma$ for solutions by solving a sequence of convex optimizations constrained in a sequence of convex feasible sets. The idea is implemented iteratively as shown in Fig.6.2a. At iteration $k$, given a reference point $\mathbf{x}^{(k)}$, a convex feasible set $\mathcal{F}^{(k)} := \mathcal{F}(\mathbf{x}^{(k)}) \subset \Gamma$ is computed around $\mathbf{x}^{(k)}$. Then a new reference point $\mathbf{x}^{(k+1)}$ will be obtained by solving the resulting convex optimization problem

$$\mathbf{x}^{(k+1)} = \arg \min_{\mathbf{x} \in \mathcal{F}^{(k)}} J(\mathbf{x}). \tag{6.9}$$

The optimal solution will be used as the reference point for the next step. The iteration will terminate if either the change in solution is small, e.g.

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq \epsilon_1, \tag{6.10}$$

for some small $\epsilon_1 > 0$, or the descent in cost is small, e.g.

$$J(\mathbf{x}^{(k)}) - J(\mathbf{x}^{(k+1)}) \leq \epsilon_2, \tag{6.11}$$

for some small $\epsilon_2 > 0$. We will show in Section 6.1.3 that these two conditions are equivalent and both of them imply convergence. The process is summarized in Algorithm 6.1.

Initialize initial guess $\mathbf{x}^{(0)}$, $k := 0$;
**while** *True* **do**
  Find a convex feasible set $\mathcal{F}^{(k)} \subset \Gamma$ for $\mathbf{x}^{(k)}$;
  Solve the convex optimization problem (6.9) for $\mathbf{x}^{(k+1)}$;
  **if** *Solution converges* **then**
    Break the while loop;
  **end**
  $k := k + 1$;
**end**
**return** $\mathbf{x}^{(k+1)}$;

**Algorithm 6.1:** The Convex Feasible Set Algorithm

### Finding the Convex Feasible Set

Since the constraint $\Gamma$ can be represented by several piecewise smooth functions (6.2). We try to find a convex feasible set $\mathcal{F}_i$ for each constraint $\Gamma_i = \{\mathbf{x} : \phi_i(\mathbf{x}) \geq 0\}$.

**Case 1: $\phi_i$ is concave** Then $\Gamma_i$ is convex. The convex feasible set is chosen to be itself,

$$\mathcal{F}_i = \Gamma_i. \tag{6.12}$$

**Case 2: $\phi_i$ is convex** Then $\Gamma_i^c$ is convex. The convex feasible set $\mathcal{F}_i$ with respect to a reference point $\mathbf{x}^r \in \mathbb{R}^n$ is defined as

$$\mathcal{F}_i(\mathbf{x}^r) := \{\mathbf{x} : \phi_i(\mathbf{x}^r) + \hat{\nabla}\phi_i(\mathbf{x}^r)(\mathbf{x} - \mathbf{x}^r) \geq 0\}, \tag{6.13}$$

where $\hat{\nabla}\phi_i(\mathbf{x}^r) \in D\phi_i(\mathbf{x}^r)$ is a sub-gradient. When $\phi_i$ is smooth at $\mathbf{x}^r$, $\hat{\nabla}\phi_i(\mathbf{x}^r)$ equals to the gradient $\nabla\phi_i(\mathbf{x}^r)$. Otherwise, the sub-gradient is chosen according to the method discussed in Section 6.1.2. Since $\phi_i$ is convex, $\phi_i(\mathbf{x}) \geq \phi_i(\mathbf{x}^r) + \partial_{\mathbf{x}-\mathbf{x}^r}\phi_i(\mathbf{x}) \geq \phi_i(\mathbf{x}^r) + d \cdot (\mathbf{x} - \mathbf{x}^r)$ for all $d \in D\phi_i(\mathbf{x}^r)$. Hence $\mathcal{F}_i(\mathbf{x}^r) \subset \{\mathbf{x} : \phi_i(\mathbf{x}) \geq 0\} = \Gamma_i$ for all $\mathbf{x}^r \in \mathbb{R}^n$.

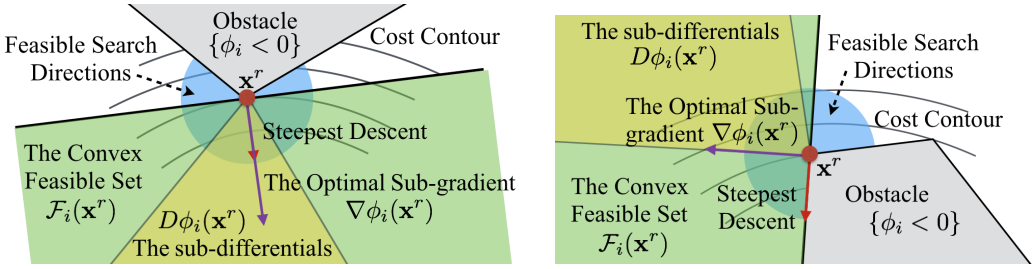
**Case 3: $\phi_i$ is neither concave nor convex** Considering Assumption 6.3, the convex feasible set with respect to the reference point $\mathbf{x}^r$ is defined as

$$\mathcal{F}_i(\mathbf{x}^r) := \{\mathbf{x} : \phi_i(\mathbf{x}^r) + \hat{\nabla}\phi_i(\mathbf{x}^r)(\mathbf{x} - \mathbf{x}^r) \geq \frac{1}{2}(\mathbf{x} - \mathbf{x}^r)^T H_i^*(\mathbf{x} - \mathbf{x}^r)\}, \tag{6.14}$$

where $\hat{\nabla}\phi_i(\mathbf{x}^r) \in D\phi_i(\mathbf{x}^r)$, which is chosen according to the method discussed in Section 6.1.2. Since $\phi_i$ is semi-convex, $\phi_i(\mathbf{x}) \geq \phi_i(\mathbf{x}^r) + \partial_{\mathbf{x}-\mathbf{x}^r}\phi_i(\mathbf{x}) - \frac{1}{2}(\mathbf{x} - \mathbf{x}^r)^T H_i^*(\mathbf{x} - \mathbf{x}^r) \geq \phi_i(\mathbf{x}^r) + d \cdot (\mathbf{x} - \mathbf{x}^r) - \frac{1}{2}(\mathbf{x} - \mathbf{x}^r)^T H_i^*(\mathbf{x} - \mathbf{x}^r)$ for all $d \in D\phi_i(\mathbf{x}^r)$. Hence $\mathcal{F}_i(\mathbf{x}^r) \subset \{\mathbf{x} : \phi_i(\mathbf{x}) \geq 0\} = \Gamma_i$ for all $\mathbf{x}^r \in \mathbb{R}^n$.

Considering (6.12), (6.13) and (6.14), the convex feasible set for $\Gamma$ at $\mathbf{x}^r$ is defined as

$$\mathcal{F}(\mathbf{x}^r) := \bigcap_i \mathcal{F}_i(\mathbf{x}^r). \tag{6.15}$$




(a) Case one (b) Case two

Figure 6.4: The choice of sub-gradient $\hat{\nabla}\phi_i(\mathbf{x}^r)$ on non-smooth point $\mathbf{x}^r$.

### Choosing the Optimal Sub-Gradients

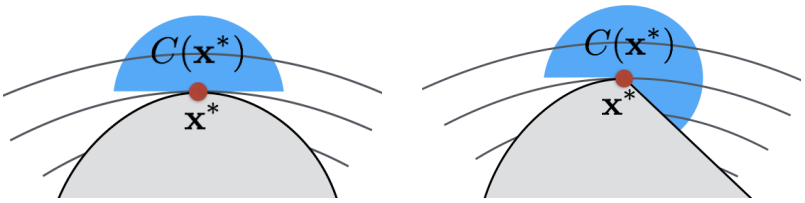
The sub-gradients in (6.13) and (6.14) should be chosen such that the steepest descent of $J$ in the set $\Gamma$ is always included in the convex feasible set $\mathcal{F}$.

Let $B(x, r)$ denote the unit ball centered at $x \in \mathbb{R}^n$ with radius $r$. At point $\mathbf{x}^r$, a search direction $v \in \partial B(0, 1)$ is feasible if for all $i$, one of the three conditions holds:

- $\phi_i(\mathbf{x}^r) > 0$;
- $\phi_i(\mathbf{x}^r) = 0$ and there exists $d \in D\phi_i$ such that $v \cdot d \geq 0$;
- $\phi_i(\mathbf{x}^r) < 0$ and there exists $d \in D\phi_i$ such that $v \cdot d > 0$.

Define the set of feasible search directions as $C(\mathbf{x}^r)$, which is non empty since we can choose $v$ to be $d/\|d\|$ for any nonzero $d \in D\phi_i$. $D\phi_i$ always contain a nonzero element by the second statement in Assumption 6.2. Then the direction of the steepest descent is $v^* := \arg\min_{v \in C(\mathbf{x}^r)} \nabla J \cdot v$. If $v^*$ is not unique, the tie breaking mechanism is chosen as: choosing the one with the smallest first entry, the smallest second entry, and so on[6]. Then the optimal sub-gradient is chosen to be $\hat{\nabla}\phi_i := \arg\min_{d \in DF_i} \nabla J \cdot d/\|d\|$, where $DF_i$ is the feasible set of sub-gradients for $\phi_i$ such that $DF_i := D\phi_i$ when $\phi_i > 0$; $DF_i := \{d \in D\phi_i | d \cdot v^* \geq 0\}$ when $\phi_i = 0$; and $DF_i := \{d \in D\phi_i | d \cdot v^* > 0\}$ when $\phi_i < 0$. The set $DF_i$ is non empty by definition of $C(\mathbf{x}^r)$. To avoid singularity, let $\|d\| = 1$ when $d = 0$. Figure 6.4 illustrates the above procedure in choosing the optimal sub-gradient, where the short arrow shows the direction of the steepest descent of $J$, the shaded sector shows the range of sub-differentials, the long arrow denotes the optimal sub-gradient and the shaded half-space is the convex feasible set $\mathcal{F}_i$. In case one, $v^*$ is in the same direction of $\hat{\nabla}\phi_i$, while the two are perpendicular to each other in case two.

[6]Note that the tie braking mechanism can be any as long as it makes $v^*$ unique. The uniqueness is exploited in Lemma 6.4.




(a) A strong local optimum. (b) A weak local optimum.

Figure 6.5: Definition of local optima.

### 6.1.3 Theoretical Results

In this section, the feasibility and convergence of Algorithm 6.1 will be shown. The main result is summarized in the following theorem.

**Theorem 6.2** (Convergence of Algorithm 6.1)**.** *Under Algorithm 6.1, the sequence* $\{\mathbf{x}^{(k)}\}$ *will converge to some* $\mathbf{x}^* \in \Gamma$ *for any initial guess* $\mathbf{x}^{(0)}$ *such that* $\mathcal{F}^{(0)} \neq \emptyset$*. When the limit is attained, e.g. there exists a constant* $K \in \mathbb{N}$ *such that* $\mathbf{x}^{(k)} = \mathbf{x}^*$ *for all* $k > K$*,* $\mathbf{x}^*$ *is a strong local optimum of (6.1). Otherwise, the sequence* $\{J(\mathbf{x}^{(k)})\}$ *is strictly decreasing and* $\mathbf{x}^*$ *is at least a weak local optimum of (6.1).*

The point $\mathbf{x}$ that satisfies $\mathcal{F}(\mathbf{x}) \neq \emptyset$ is called nearly feasible. We say that $\mathbf{x}^*$ is a strong local optimum of (6.1) if $J$ is nondecreasing along any feasible search direction, e.g. $\nabla J(\mathbf{x}^*)v \geq 0$ for all $v \in C(\mathbf{x}^*)$ as shown in Fig.6.5a. We say that $\mathbf{x}^*$ is a weak local optimum of (6.1) if the KKT condition is satisfied, i.e. $\nabla J(\mathbf{x}^*) + \sum_{i=1}^{N} \lambda_i d_i = 0$ for some $d_i \in D\phi_i(\mathbf{x}^*)$ as shown in Fig.6.5b. $\lambda_i$ is a Lagrange multiplier such that $\lambda_i \leq 0$ and $\lambda_i \phi_i(\mathbf{x}) = 0$ (complementary slackness) for all $i = 1, \cdots, N$. A strong local optimum is always a weak local optimum. The two are equivalent when all $\phi_i$'s are smooth at $\mathbf{x}^*$.

Before proving Theorem 6.2, we present some preliminary results that are useful towards proving the theorem. The proofs of the preliminary results can be found in [76]. We say that a reference point $\mathbf{x}^r \in \mathbb{R}^n$ is feasible if $\mathbf{x}^r \in \Gamma$; and $\mathbf{x}^* \in \Gamma$ is a fixed point of Algorithm 6.1 if

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{F}(\mathbf{x}^*)} J(\mathbf{x}). \tag{6.16}$$

**Lemma 6.3** (Feasibility)**.** *If* $\mathbf{x}^r \in \Gamma$*, then* $\mathbf{x}^r \in \mathcal{F}(\mathbf{x}^r)$ *and* $\mathcal{F}^o(\mathbf{x}^r) \neq \emptyset$*.*

**Proposition 6.4** (Fixed point)**.** *If* $\mathbf{x}^*$ *is a fixed point of Algorithm 6.1, then* $\mathbf{x}^*$ *is a strong local optimum of (6.1).*

Lemma 6.3 and Lemma 6.4 imply that a feasible $\mathbf{x}^r$ can always be improved by optimizing over the convex feasible set $\mathcal{F}(\mathbf{x}^r)$ if $\mathbf{x}^r$ itself is not a local optimum. However, the existence of nonempty convex feasible set for an infeasible reference point is more intricate, which is deeply related to the choice of the function $\phi_i$'s.

**Lemma 6.5** (Strong descent)**.** *For any feasible* $\mathbf{x}^{(k)}$*, the descent of the objective function satisfies that* $\nabla J(\mathbf{x}^{(k+1)})(\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}) \geq 0$*. Moreover, if* $J(\mathbf{x}^{(k+1)}) = J(\mathbf{x}^{(k)})$*, then* $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$*.*

**Proposition 6.6** (Convergence of strictly descending sequence)**.** *Consider the sequence* $\{\mathbf{x}^{(k)}\}$ *generated by Algorithm 6.1. If* $J(\mathbf{x}^{(1)}) > J(\mathbf{x}^{(2)}) > \cdots$*, then the sequence* $\{\mathbf{x}^{(k)}\}$ *converges to a weak local optimum* $\mathbf{x}^*$ *of (6.1).*

Lemma 6.5 and Lemma 6.6 justify the adoption of the terminate condition (6.11), which is indeed equivalent to the standard terminate condition (6.10), e.g. convergence in the objective function implies convergence in the solution.

*Proof of Theorem 6.2.* If $\mathcal{F}^{(0)}$ is nonempty, then $\mathbf{x}^{(1)} \in \Gamma$ can be obtained by solving the convex optimization (6.9). By Lemma 6.3, $\mathcal{F}^{(1)}$ has nonempty interior, then $\mathbf{x}^{(2)} \in \Gamma$ can be obtained. By induction, we can conclude that $\mathbf{x}^{(i)} \in \mathcal{F}^{(i-1)} \subset \Gamma$ for $i = 1, 2, 3, \cdots$. Moreover, as a better solution is found at each iteration, then $J(\mathbf{x}^{(1)}) \geq J(\mathbf{x}^{(2)}) \geq \cdots$. This leads to two cases. The first case is that $J(\mathbf{x}^{(K)}) = J(\mathbf{x}^{(K+1)})$ for some $K$, while the second case is that the cost keeps decreasing strictly, e.g. $J(\mathbf{x}^{(1)}) > J(\mathbf{x}^{(2)}) > \cdots$. In the first case, the condition $J(\mathbf{x}^{(K)}) = J(\mathbf{x}^{(K+1)})$ is equivalent to $\mathbf{x}^{(K)} = \mathbf{x}^{(K+1)}$ by Lemma 6.5. By induction, the algorithm converges, e.g. $\mathbf{x}^{(k)} = \mathbf{x}^{(k+1)}$ and $J(\mathbf{x}^{(k)}) = J(\mathbf{x}^{(k+1)})$ for all $k \geq K$. Moreover, as $\mathbf{x}^* := \mathbf{x}^{(K)}$ is a fixed point, it is a strong local optima by Lemma 6.4. If the cost keeps decreasing, e.g. $J(\mathbf{x}^{(1)}) > J(\mathbf{x}^{(2)}) > \cdots$, then the sequence $\{\mathbf{x}^{(k)}\}$ converges to a weak local optima $\mathbf{x}^*$ by Lemma 6.6. □

### 6.1.4 Applications

Many motion planning problems can be formulated into (6.1) when $\mathbf{x}$ is regarded as the trajectory as discussed in Chapter 5. The dimension of the problem $n$ is proportional to the number of sampling points on the trajectory. If continuous trajectories are considered, then $n \to \infty$ and $\mathbb{R}^n$ approaches the space of continuous functions $\mathcal{C}(\mathbb{R})$ in the limit.

The performance of Algorithm 6.1 will be illustrated through one example, which will also be compared to the performance of existing non-convex optimization methods such as interior point (ITP) and sequential quadratic programming (SQP). For simplicity, only convex obstacles and convex boundaries are considered[7]. Algorithm 6.1 is implemented in both Matlab and C++. The convex optimization problem (6.9) is solved using the interior-point-convex method in `quadprog` in Matlab and the interior point method in Knitro [22] in C++. For comparison, (6.1) is also solved directly using ITP and SQP methods in `fmincon` [105] in Matlab and in Knitro in C++. To create fair comparison, the gradient and the

[7]The non-convex obstacles or boundaries can either be partitioned into several convex components or be replaced with their convex envelops. Moreover, in practice, obstacles are measured by point clouds. The geometric information is extracted by taking convex hull of the points. Hence it automatically partitions the obstacles into several convex polytopes.

Hessian of the objective function $J$ and the optimal sub-gradients $\hat{\nabla}\phi_i$ of the constraint function $\phi_i$'s are also provided to the ITP and SQP solvers.

In the examples, the problem formulation in (5.7) is adopted where $x_0 = (0, 0)$ and $x_h = (9, 0)$. The planning horizon $h$ goes from 30 to 100. $t_s = (h + 1)^{-1}$. The cost function (5.7a) penalizes the average acceleration along the trajectory, e.g. $Q = 0$ and $S = h^{-1}A^T A$. The initial reference $\mathbf{x}^{(0)}$ is chosen to be a straight line connecting $x_0$ and $x_h$ with equally sampled waypoints. In the first scenario, there are three disjoint convex obstacles as shown in Fig.6.6. In the constraint, a distance margin of 0.25 to the obstacles is required.

The computation time under different solvers is listed in Table 6.1. The first column shows the horizon. In the second column, "-M" means the algorithm is run in Matlab and "-C" means the algorithm is run in C++. Under each scenario, the first column shows the final cost. The second column is the total number of iterations. The third and fourth columns are the total computation time and the average computation time per iteration respectively (only the entries that are less than $100ms$ are shown). It does happen that the algorithms find different local optima, though CFS-M and CFS-C always find the same solution. In terms of computation time, Algorithm 6.1 always outperforms ITP and SQP, since it requires less time per iteration and fewer iterations to converge. This is due to the fact CFS does not require additional line search after solving (6.9) as is needed in ITP and SQP, hence saving time during each iteration. CFS requires fewer iterations to converge since it can take unconstrained step length $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|$ in the convex feasible set as will be shown later. Moreover, Algorithm 6.1 scales much better than ITP and SQP, as the computation time and time per iteration in CFS-C go up almost linearly with respect to $h$ (or the number of variables).

The computation time of CFS consists of two parts: 1) the processing time, i.e. the time to compute $\mathcal{F}$ and 2) the optimization time, i.e. the time to solve (6.9). As shown in Fig.6.7, the two parts grow with $h$. In Matlab, the processing time dominates, while the optimization time dominates in C++.

To better illustrate the advantage of Algorithm 6.1, the runtime statistics across all methods when $h = 100$ is shown in Fig.6.8. The first log-log figure shows the cost $J(\mathbf{x}^{(k)})$ versus iteration $k$, while the second semi-log figure shows the feasibility error $\max\{0, -\min_i \phi_i(\mathbf{x}^{(k)})\}$ versus $k$. At the beginning, the cost $J = 0$ and the feasibility error is 0.75. In Algorithm 6.1, $\mathbf{x}^{(k)}$ becomes feasible at the first iteration while the cost $J(\mathbf{x}^{(k)})$ jumps up. In the following iterations, the cost goes down and converges to the optimum value. In ITP-C, the problem becomes feasible at the third iteration. In SQP-C, it is the fifth iteration. In order to make the problem feasible, the cost jumps much higher in ITP-C than in CFS-C. Once the problem is feasible, it also takes more iterations for ITP-C and SQP-C to converge than CFS-C. On the other hand, ITP-M and SQP-M have very small step length in the beginning. The problem only becomes feasible after 100 iterations. But once the problem is feasible, the performance of ITP-M and SQP-M is similar to that of ITP-C and SQP-C. Note that the cost below 1 is not shown in the figure.

The optimal trajectories computed by Algorithm 6.1 for different $h$ is shown in Fig.6.6. Those trajectories converge to a continuous trajectory when $h$ goes up.

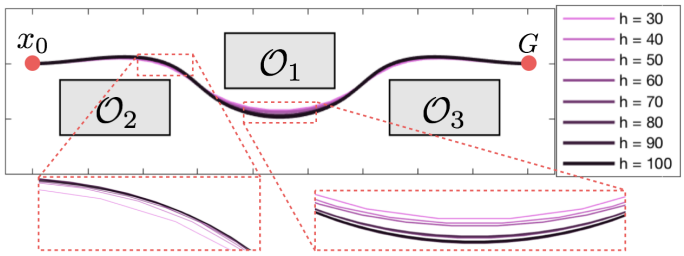



Figure 6.6: Simulation environment and the optimal trajectories for different horizon $h$.

Table 6.1: Comparison among CFS, ITP and SQP.

| $h$ | Method | Cost | Iter | Time | dT |
|---|---|---|---|---|---|
| 100 or 60 | SQP-M | 1358.9 | 239 | 140.5s | - |
| | ITP-M | 1358.9 | 470 | 50.6s | - |
| | CFS-M | 1358.9 | 18 | 1.8s | 98.8ms |
| | SQP-C | 1347.6 | 123 | 47.3s | - |
| | ITP-C | 1341.7 | 306 | 2.9s | 9.5ms |
| | CFS-C | 1358.9 | 18 | **74.4ms** | **4.1ms** |
| 50 | SQP-M | 2299.5 | 110 | 25.8s | - |
| | ITP-M | 1308.3 | 187 | 8.8s | 47.1ms |
| | CFS-M | 1458.2 | 8 | 212.1ms | 26.5ms |
| | SQP-C | 1308.3 | 52 | 3.4s | 65.4ms |
| | ITP-C | 1275.1 | 131 | 390ms | 3.0ms |
| | CFS-C | 1458.2 | 8 | **23.7ms** | **3.0ms** |
| 40 | SQP-M | 3391.6 | 97 | 15.6s | - |
| | ITP-M | 1317.0 | 150 | 5.2s | 34.7ms |
| | CFS-M | 1317.0 | 8 | 172.6ms | 21.6ms |
| | SQP-C | 1317.0 | 40 | 1.5s | 37.5ms |
| | ITP-C | 1170.5 | 102 | 240.5ms | 2.4ms |
| | CFS-C | 1317.0 | 8 | **16.5ms** | **2.1ms** |
| 30 | SQP-M | 1039.2 | 106 | 8.4s | 79.2ms |
| | ITP-M | 1039.2 | 109 | 2.8s | 25.7ms |
| | CFS-M | 1039.2 | 12 | 208.2ms | 17.3ms |
| | SQP-C | 1453.3 | 27 | 379.1ms | 14.0ms |
| | ITP-C | 1039.2 | 59 | 118.5ms | 2.0ms |
| | CFS-C | 1039.2 | 12 | **19.0ms** | **1.6ms** |

With respect to the results, we conclude that Algorithm 6.1 is time-efficient, local-optimal and scalable.

In addition to motion planning problems, the proposed method deals with any problem with similar geometric properties as specified in Assumption 6.1 and Assumption 6.2. Moreover, problems with global linear equality constraints also fit into the framework if we solve the problem in the low-dimensional linear manifold defined by the linear equality constraints.

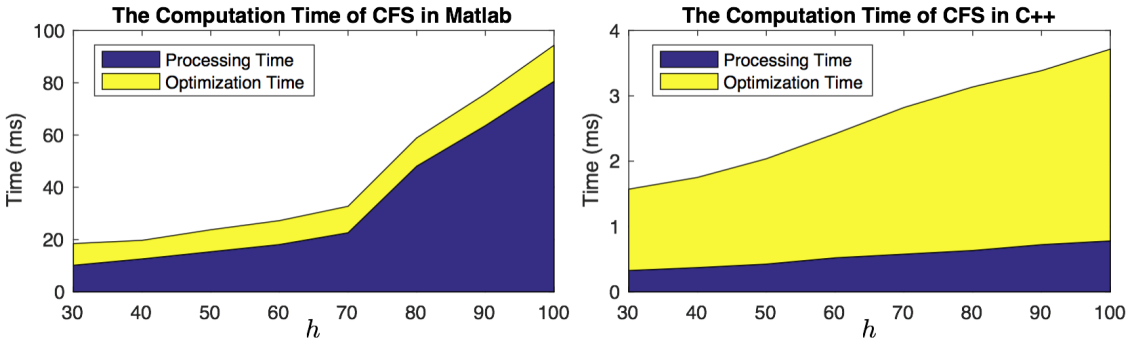



Figure 6.7: The decomposed time per iteration using Algorithm 6.1.

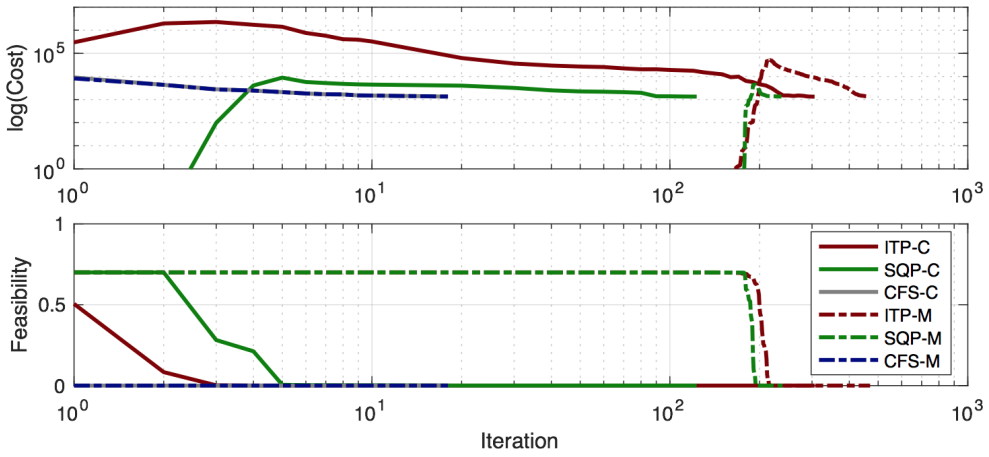



Figure 6.8: The run time statistics.

The case for nonlinear equality constraints will be discussed below.

## 6.2 Non-Convex Optimization on Nonlinear Space

One limitation of the CFS algorithm is that it may not converge to local optima under nonlinear equality constraints (such as nonlinear dynamic constraints) since the convex feasible set for a nonlinear equality constraint may reduce to a singleton point.

In this section, the slack convex feasible set algorithm (SCFS) is introduced to handle optimization problems with convex cost functions and non-convex equality and inequality constraints. The idea is to relax the nonlinear equality constraints to several nonlinear inequality constraints using slack variables and then solve the relaxed problem using CFS. The feasibility, convergence and optimality of the algorithm will be shown in this section. The performance of SCFS will be compared to that of conventional non-convex solvers.

The remainder of the section will be organized as follows: in Section 6.2.1, the benchmark problem is proposed; Section 6.2.2 introduces the slack convex feasible set algorithm; Section 6.2.3 proves the convergence of the SCFS algorithm; Section 6.2.4 discusses the application of the algorithm.

### 6.2.1 The Benchmark Problem

Denote the state of the robot as $x \in X \subset \mathbb{R}^n$ where $X$ represents $n$ dimensional state space. Denote the control input of the robot as $u \in U \subset \mathbb{R}^m$ where $U$ represents $m$ dimensional control space. The robot needs to travel from $\mathcal{A}$ to $\mathcal{Z}$. Its trajectory is denoted as $\mathbf{x} = [x_0^T, x_1^T, \cdots, x_h^T]^T \in X^{h+1}$ where $x_q$ is the robot state at time step $q$ and $h$ is the planning horizon. Without loss of generality, the sampling time $t_s$ is assumed to be 1. Similarly, the input trajectory is denoted as $\mathbf{u} = [u_0^T, u_1^T, \cdots, u_{h-1}^T]^T \in U^h$ where $u_q$ is the robot input at time step $q$. Let $u_q^j$ denote the $j$-th entry in $u_q$ for $j = 1, \cdots, m$.

Consider the following optimization problem

$$\min_{\mathbf{x},\mathbf{u}} \; J(\mathbf{x}, \mathbf{u}) \tag{6.17a}$$

$$s.t. \; \mathbf{x} \in \Gamma, \mathbf{u} \in \Omega, \mathcal{G}(\mathbf{x}, \mathbf{u}) = 0, \tag{6.17b}$$

where $J : X^{h+1} \times U^h \to \mathbb{R}$ is the cost function; $\Gamma$ is the constraint on the augmented state space $X^{h+1}$; $\Omega$ is the constraint on the augmented control space $U^h$; and $\mathcal{G} : X^{h+1} \times U^h \to \mathbb{R}^{mh}$ represents the dynamic relationship between states and inputs. Assumptions 6.4 to 6.8 are required.

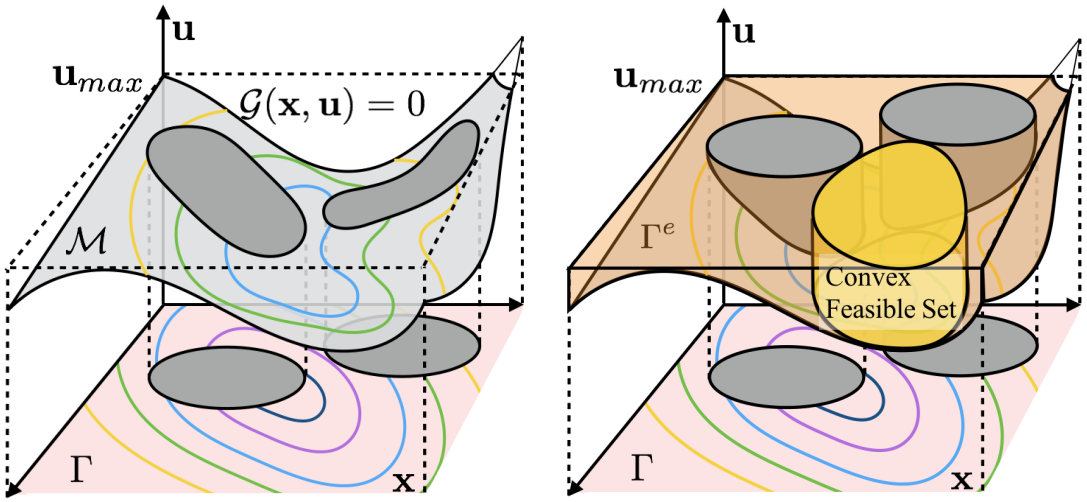
**Assumption 6.4** (Cost Function)**.** *The cost function $J(\mathbf{x}, \mathbf{u}) = J_1(\mathbf{x}) + J_2(\mathbf{u})$ is smooth and bounded below by 0. $J_1$ is strictly convex. $J_2$ is strictly convex and symmetric, and it achieves minimum at $\mathbf{u} = 0$.*

**Assumption 6.5** (State Constraint)**.** *The constraint $\Gamma$ is a collection of linear equality constraints, linear inequality constraints and $N$ nonlinear inequality constraints, i.e. $\Gamma = \cap_i \Gamma_i$ where*

$$\Gamma_i = \begin{cases} \{\mathbf{x} : \phi_i(\mathbf{x}) \geq 0\} & i = 1, \cdots, N \\ \{\mathbf{x} : A_{eq}\mathbf{x} = b_{eq}\} & i = N+1 \\ \{\mathbf{x} : A\mathbf{x} \leq b\} & i = N+2, \end{cases} \tag{6.18}$$

*$A_{eq} \in \mathbb{R}^{k_{eq} \times n(h+1)}$, $b_{eq} \in \mathbb{R}^{k_{eq}}$, $A \in \mathbb{R}^{k \times n(h+1)}$, and $b \in \mathbb{R}^k$ where $k_{eq} < n(h+1)$ and $k$ are the dimensions of the constraints such that $rank(A_{eq}) = k_{eq}$. $\phi_i : \mathbb{R}^{n(h+1)} \to \mathbb{R}$ is continuous, piece-wise smooth and semi-convex, e.g. there exists a positive semi-definite matrix $H_i^* \in \mathbb{R}^{n(h+1) \times n(h+1)}$ such that for any $\mathbf{x}, v \in \mathbb{R}^{n(h+1)}$, $\phi_i(\mathbf{x} + v) - 2\phi_i(\mathbf{x}) + \phi_i(\mathbf{x} - v) \geq -v^T H_i^* v$. Moreover, the interior of the inequality constraints is nontrivial, i.e. $\cap_i \{\mathbf{x} : \phi_i(\mathbf{x}) > 0\} \neq \emptyset$[8].*

[8]This is to exclude the case that some combination of nonlinear inequality constraints indeed forms a nonlinear equality constraint, such as $\Gamma = \{\mathbf{x} : \phi_i(\mathbf{x}) \geq 0, -\phi_i(\mathbf{x}) \geq 0\}$.




(a) The geometry of problem (6.17). (b) The convex feasible set for problem (6.23).

Figure 6.9: Geometric illustration of the non-convex optimization on a nonlinear space.

The linear equality constraints are for boundary conditions at the start point and the goal point. The linear inequality constraints are for state limits. The nonlinear equality constraints are for collision avoidance where $\phi_i$ can usually be identified as a signed distance function to an obstacle. The semi-convexity assumption on $\phi_i$ is satisfied if there is no concave corner in the obstacle.

**Assumption 6.6** (Control Constraint)**.** *The constraint $\Omega$ is a box constraint such that $-\bar{\mathbf{u}} \leq \mathbf{u} \leq \bar{\mathbf{u}}$ for some constant vector $\bar{\mathbf{u}} = [\bar{u}_0^1, \cdots, \bar{u}_{h-1}^m]^T > 0$.*

**Assumption 6.7** (Dynamic Constraint)**.** *The dynamic equation $\mathcal{G}(\mathbf{x},\mathbf{u}) = 0$ is affine in $\mathbf{u}$, i.e. there exist functions $F : X^{h+1} \to \mathbb{R}^{mh}$ and $H : X^{h+1} \to \mathbb{R}^{mh\times mh}$ such that*

$$\mathcal{G}(\mathbf{x},\mathbf{u}) = F(\mathbf{x}) + H(\mathbf{x})\mathbf{u} = 0. \tag{6.19}$$

*$H$ is assumed to be diagonal, non-singular and positive definite. Equation (6.19) is equivalent to*

$$f_q^j(\mathbf{x}) + h_q^j(\mathbf{x})u_q^j = 0, \forall q = 0, \cdots, h-1, j = 1, \cdots, m, \tag{6.20}$$

*where $f_q^j : X^{h+1} \to \mathbb{R}$ and $h_q^j : X^{h+1} \to \mathbb{R}^+$ are entries in $F$ and $H$, which are smooth with bounded derivatives and Hessians.*

Equation (6.19) and (6.20) cover a wide range of typical nonlinear dynamic systems. For robot arms, let $x$ be the robot joint position and $u$ be the torque input, then the relationship between $x$ and $u$ is

$$M(x_q)(x_{q+1} - 2x_q + x_{q-1}) + N(x_q, x_q - x_{q-1}) = u_q, \tag{6.21}$$

where $M(\cdot)$ represents the generalized inertia matrix and $N(\cdot,\cdot)$ represents the Coriolis and centrifugal forces. Finite differences are used to compute joint velocity and joint acceleration. For vehicles, let $x$ be the position of the rear axle of the vehicle and $u$ be the yaw rate, then

$$(x_q - x_{q-1}) \times (x_{q+1} - x_q) = \|x_q - x_{q-1}\|^2 u_q. \tag{6.22}$$

where $\times$ denotes the cross product. Both cases satisfy (6.20). Moreover, the relationship (5.12d) in the temporal optimization also satisfies (6.20).

By Assumptions 6.5 to 6.7, the constraints in (6.17b) form a $K$ dimensional manifold $\mathcal{M}$ where $K = n(h+1) - k_{eq}$[9]. In order for the optimization to be nontrivial, the manifold $\mathcal{M}$ should have non empty interior, which leads us to the following assumption.

**Assumption 6.8** (Connected Nontrivial Domain)**.** *The domain that satisfies (6.17b) is connected. There exist* $\mathbf{x}^*$ *and* $\mathbf{u}^*$ *that satisfy all the constraints in (6.17b) such that* $A\mathbf{x}^* < b$, $\phi_i(\mathbf{x}^*) > 0$ *for all* $i$ *and* $-\bar{\mathbf{u}} < \mathbf{u}^* < \bar{\mathbf{u}}$.

The geometry of problem (6.17) is illustrated in Fig.6.9a where the horizontal plane represents the augmented state space $X^{h+1}$ and the vertical axis represents the augmented control space $U^h$. The dark holes represent the infeasible set $X^{h+1} \setminus \Gamma$. The manifold $\mathcal{M}$ is represented by the shaded curvy surface under the nonlinear equality constraints. The cost $J$ on the manifold $\mathcal{M}$ is shown in the contours. Although the cost $J$ is convex in $\mathbf{x}$ and $\mathbf{u}$ (as shown in the contours on the horizontal plane), the cost $J|_{\mathcal{M}}$ is non-convex as $\mathcal{M}$ is nonlinear and non-convex.

### 6.2.2 The Slack Convex Feasible Set Algorithm

The slack convex feasible set algorithm (SCFS) will be introduced in this section to solve problem (6.17). The key idea is to introduce slack variables and transform the nonlinear equality constraint (6.19) to a set of non degenerating nonlinear inequality constraints such that CFS algorithm can be applied.

#### The Relaxed Problem

Let $\mathbf{y}$ be the slack variable for $\mathbf{u}$ such that $-\mathbf{y} \le \mathbf{u} \le \mathbf{y}$. Symmetry of problem ( 6.17) in $\mathbf{u}$ is exploited in the relaxation. By Assumption 6.4, $J_2(\mathbf{u}) \le J_2(\mathbf{y})$. By Assumption 6.6, $\mathbf{y} \le \bar{\mathbf{u}}$ implies $-\bar{\mathbf{u}} \le \mathbf{u} \le \bar{\mathbf{u}}$. Using the slack variable $\mathbf{y}$, constraint (6.19) can be relaxed to two inequality constraints with respect to $\mathbf{y}$, e.g. $F(\mathbf{x}) - H(\mathbf{x})\mathbf{y} \le F(\mathbf{x}) + H(\mathbf{x})\mathbf{u} = 0 \le F(\mathbf{x}) + H(\mathbf{x})\mathbf{y}$. Then we have the following relaxed problem.

$$\min_{\mathbf{x},\mathbf{y}} \; J(\mathbf{x},\mathbf{y}) \tag{6.23a}$$

$$s.t.\ \mathbf{x} \in \Gamma, \mathbf{y} \le \bar{\mathbf{u}} \tag{6.23b}$$

$$F(\mathbf{x}) + H(\mathbf{x})\mathbf{y} \ge 0, F(\mathbf{x}) - H(\mathbf{x})\mathbf{y} \le 0. \tag{6.23c}$$

[9]The dimension of the decision variables $\mathbf{x}$ and $\mathbf{u}$ is $n(h+1)+mh$. As there are $k_{eq}+mh$ independent equality constraints, the dimension of the manifold is reduced to $n(h+1)-k_{eq}$.

Consider the extended state $\mathbf{z} = [\mathbf{x}^T, \mathbf{y}^T]^T$. The relaxed problem can be transformed to

$$\min_{\mathbf{z}\in\Gamma^e} J(\mathbf{z}), \tag{6.24}$$

where $\Gamma^e := (\Gamma \oplus \{\mathbf{y} \le \bar{\mathbf{u}}\}) \cap \{F(\mathbf{x}) + H(\mathbf{x})\mathbf{y} \ge 0\} \cap \{-F(\mathbf{x}) + H(\mathbf{x})\mathbf{y} \ge 0\}$. The difference between problem (6.23) and problem (6.17) is illustrated in Fig.6.10, where the curve in Fig.6.10a represents the nonlinear equality constraint (6.19) in problem (6.17) and the shaded area in Fig.6.10b represents the nonlinear inequality constraints (6.23c) in problem (6.23). Due to the introduction of the slack variable $\mathbf{y}$, the nonlinear equality constraint is successfully removed. $\Gamma^e$ is illustrated in Fig.6.9b, whose dimension is $n(h+1) + mh - k_{eq}$ and $\mathcal{M}$ belongs to the boundary of $\Gamma^e$.

The intuition behind the relaxation is that: as $J_2$ achieves minimum at $\mathbf{u} = 0$, the optimization algorithm will pull the optimal solution down to $\mathcal{M}$ which is on the "bottom" of $\Gamma^e$, so that we may still get the same optimal solution as in problem (6.17). This property will be proved in Theorem 6.8, Lemma 6.9 and Proposition 6.10. We first verify that the relaxed problem satisfies Assumption 6.5 (which will imply that Assumption 6.1 to Assumption 6.3 are satisfied).

**Lemma 6.7.** *$\Gamma^e$ satisfies Assumption 6.5 if we take $\mathbf{z}$ as the state variables.*

*Proof.* It is easy to verify that $\Gamma^e$ is a collection of linear equality constraints, linear inequality constraints and nonlinear inequality constraints, e.g. $\Gamma^e = \cap_i \Gamma_i^e$ where

$$\Gamma_i^e = \begin{cases} \{\mathbf{z} : \phi_i(\mathbf{x}) \ge 0\} & i = 1, \cdots, N \\ \{\mathbf{z} : \varphi_q^j(\mathbf{z}) \ge 0\} & i = N + qm + j \\ \{\mathbf{z} : \psi_q^j(\mathbf{z}) \ge 0\} & i = N + (q+h)m + j \\ \{\mathbf{z} : A_{eq}\mathbf{x} = b_{eq}\} & i = N + 2hm + 1 \\ \{\mathbf{z} : A\mathbf{x} \le b\} & i = N + 2hm + 2 \\ \{\mathbf{z} : \mathbf{y} \le \mathbf{u}_{max}\} & i = N + 2hm + 3 \end{cases}, \tag{6.25}$$

$\varphi_q^j(\mathbf{z}) := f_q^j(\mathbf{x}) + h_q^j(\mathbf{x})y_q^j$, and $\psi_q^j(\mathbf{z}) := -f_q^j(\mathbf{x}) + h_q^j(\mathbf{x})y_q^j$. $\varphi_q^j$ and $\psi_q^j$ are smooth as $f_q^j$ and $h_q^j$ are smooth. We just need to check whether they are semi-convex. Since $f_q^j$ and $h_q^j$ have bounded derivatives and Hessians and $y_q^j$ is bounded, the Hessian of $\varphi_q^j$ is bounded below, e.g.

$$\begin{aligned} \nabla^2_{[\mathbf{x},y_q^j]}\varphi_q^j &= \begin{bmatrix} \nabla^2_{\mathbf{x}} f_q^j(\mathbf{x}) + \nabla^2_{\mathbf{x}} h_q^j(\mathbf{x}) y_q^j & \nabla_{\mathbf{x}} h_q^j(\mathbf{x}) \\ \nabla_{\mathbf{x}} h_q^j(\mathbf{x}) & 0 \end{bmatrix} \\ &\succeq \begin{bmatrix} \nabla^2_{\mathbf{x}} f_q^j + \nabla^2_{\mathbf{x}} h_q^j y_q^j - \nabla_{\mathbf{x}} h_q^j \nabla^T_{\mathbf{x}} h_q^j & 0 \\ 0 & -1 \end{bmatrix} \\ &\succeq -\begin{bmatrix} H_q^j & 0 \\ 0 & 1 \end{bmatrix}, \end{aligned} \tag{6.26}$$

where $H_q^j$ depends on the bounds on $\nabla^2_{\mathbf{x}} f_q^j$, $\nabla^2_{\mathbf{x}} h_q^j$, $\nabla_{\mathbf{x}} h_q^j$ and $y_q^j$. Similar condition holds for $\nabla^2_{[\mathbf{x},y_q^j]}\psi_q^j$.


(a) The nonlinear equality constraint. (b) The relaxed nonlinear inequality constraint.

Figure 6.10: The relaxation of the nonlinear equality constraint.

Then we need to verify that the interior of the nonlinear inequality constraints is non empty. Consider $\mathbf{x}^*$ and $\mathbf{u}^*$ in Assumption 6.8. Let $\mathbf{z}^* = [(\mathbf{x}^*)^T, \bar{\mathbf{u}}^T]^T$. Then $F(\mathbf{x}^*) - H(\mathbf{x}^*)\bar{\mathbf{u}} < 0 = F(\mathbf{x}^*) + H(\mathbf{x}^*)\mathbf{u}^* < F(\mathbf{x}^*) + H(\mathbf{x}^*)\bar{\mathbf{u}}$. Hence $\mathbf{z}^* \in \cap_i\{\mathbf{z} : \phi_i(\mathbf{x}) > 0\} \cap \{\mathbf{z} : F(\mathbf{x}) + H(\mathbf{x})\mathbf{y} > 0\} \cap \{\mathbf{z} : -F(\mathbf{x}) + H(\mathbf{x})\mathbf{y} > 0\}$. Thus $\Gamma^e$ satisfies Assumption 6.5. □

**The Algorithm**

The steps in the SCFS algorithm are summarized below.

**Algorithm 1** (SCFS)**.** *Problem (6.17) is solved using the following steps:*

1. *Transform problem (6.17) to problem (6.23);*
2. *Initialize* $\mathbf{z}^{(0)}$*;*
3. *Apply Algorithm 6.1 on problem (6.23) by iteratively solving the following problem*

$$\mathbf{z}^{(k+1)} = \arg \min_{\mathbf{z} \in \mathcal{F}^e(\mathbf{z}^{(k)})} J(\mathbf{z}), \tag{6.27}$$

   *where* $\mathcal{F}^e(\mathbf{z}^{(k)})$ *(illustrated in Fig.6.9b) is the convex feasible set of* $\Gamma^e$ *constructed according to the three cases in Section 6.1.2. Denote the solution as* $\hat{\mathbf{z}} = [\hat{\mathbf{x}}^T, \hat{\mathbf{y}}^T]^T$*;*
4. *Construct the solution* $\hat{\mathbf{w}} = [\hat{\mathbf{x}}^T, \hat{\mathbf{u}}^T]^T$ *for problem (6.17) from* $\hat{\mathbf{z}}$ *by setting* $\hat{\mathbf{u}}$ *to be* $-H^{-1}(\hat{\mathbf{x}})F(\hat{\mathbf{x}})$*.*

### 6.2.3 Theoretical Results

**Theorem 6.8** (Convergence of SCFS)**.** *Algorithm 1 will converge to a local optimum of problem (6.17) for any nearly feasible initial value* $\mathbf{z}^{(0)}$*.*

This is the main result in this section. Before proving the theorem, we first show that the local optima of problem (6.23) is equivalent to the local optima of problem (6.17).

**Lemma 6.9.** *Any local optimum* $\hat{\mathbf{z}} = [\hat{\mathbf{x}}^T, \hat{\mathbf{y}}^T]^T$ *of problem (6.23) satisfies* $|H^{-1}(\hat{\mathbf{x}})F(\hat{\mathbf{x}})| = \hat{\mathbf{y}}$.

*Proof.* Let $\hat{\mathbf{u}} = -H^{-1}(\hat{\mathbf{x}})F(\hat{\mathbf{x}})$. For each entry $\hat{u}_q^j$ in $\hat{\mathbf{u}}$, constraint (6.23c) implies that $\hat{u}_q^j = -(h_q^j)^{-1}(\hat{\mathbf{x}})f_q^j(\hat{\mathbf{x}}) \in [-\hat{y}_q^j, \hat{y}_q^j]$. Suppose there exist some $q$ and $j$ such that $|\hat{u}_q^j| < \hat{y}_q^j$. Define $\tilde{\mathbf{z}} := [\hat{\mathbf{x}}^T, \hat{y}_0^1, \cdots, \hat{y}_q^j - \epsilon, \cdots, \hat{y}_{h-1}^m]$ for small $\epsilon \in (0, \hat{y}_q^j - |\hat{u}_q^j|)$. Since $|\hat{u}_q^j| < \hat{y}_q^j$, then $\varphi_q^j(\hat{\mathbf{z}}) > 0$ and $\psi_q^j(\hat{\mathbf{z}}) > 0$. Due to the continuity of $\varphi_q^j$ and $\psi_q^j$, $\varphi_q^j(\tilde{\mathbf{z}}) > 0$ and $\psi_q^j(\tilde{\mathbf{z}}) > 0$ for $\epsilon$ sufficiently small. Hence $\tilde{\mathbf{z}} \in \Gamma^e$. However $J(\tilde{\mathbf{z}}) < J(\hat{\mathbf{z}})$ by Assumption 6.4, which contradicts the fact that $\hat{\mathbf{z}}$ is a local optimum. Hence $|\hat{u}_q^j| = \hat{y}_q^j$ for all $q$ and $j$. □

**Proposition 6.10** (Equivalence)**.** *If* $\hat{\mathbf{z}} = [\hat{\mathbf{x}}^T, \hat{\mathbf{y}}^T]^T$ *with* $\hat{\mathbf{y}} = |H^{-1}(\hat{\mathbf{x}})F(\hat{\mathbf{x}})|$ *is a local optimum of problem (6.23), then* $\hat{\mathbf{w}} = [\hat{\mathbf{x}}^T, \hat{\mathbf{u}}^T]^T$ *with* $\hat{\mathbf{u}} = -H^{-1}(\hat{\mathbf{x}})F(\hat{\mathbf{x}})$ *is a local optimum of problem (6.17), and vice versa.*

*Proof.* Since $\hat{\mathbf{z}}$ is a local optimum in problem (6.23), there exist Lagrangian multipliers $\eta_q^j, \xi_q^j, \lambda_i \le 0$, $\alpha_q^j \ge 0$, $\Lambda_1 \ge 0$ and $\Lambda_2 \in \mathbb{R}^{k_{eq}}$ for all $i$, $j$ and $q$ such that $\nabla L^r|_{\hat{\mathbf{z}}} = 0$, where the Lagrangian $L^r$ is defined as

$$\begin{aligned} L^r &= J + \sum_{q,j}(\eta_q^j\varphi_q^j + \xi_q^j\psi_q^j + \alpha_q^j(y_q^j - \bar{u}_q^j)) + \sum_i \lambda_i\phi_i \\ &\quad + \Lambda_1^T(A\mathbf{x} - b) + \Lambda_2^T(A_{eq}\mathbf{x} - b_{eq}). \end{aligned} \tag{6.28}$$

Taking the partial derivatives at $\hat{\mathbf{z}}$, we have

$$\begin{aligned} 0 &= \nabla_{\mathbf{x}} J_1 + \sum \eta_q^j \nabla_{\mathbf{x}}\varphi_q^j + \sum \xi_q^j \nabla_{\mathbf{x}}\psi_q^j + \sum \lambda_i \nabla_{\mathbf{x}}\phi_i \\ &\quad + A^T\Lambda_1 + A_{eq}^T\Lambda_2, \end{aligned} \tag{6.29}$$

$$0 = \nabla_{y_q^j} J_2 + \eta_q^j \nabla_{y_q^j}\varphi_q^j + \xi_q^j \nabla_{y_q^j}\psi_q^j + \alpha_q^j. \tag{6.30}$$

The complementary slackness condition at $\hat{\mathbf{z}}$ is

$$\eta_q^j\psi_q^j = \xi_q^j\varphi_q^j = \lambda_i\phi_i = \alpha_q^j(\hat{y}_q^j - \bar{u}_q^j) = 0, \tag{6.31}$$

$$\Lambda_1^T(A\hat{\mathbf{x}} - b) = 0. \tag{6.32}$$

We need to show that (6.29-6.32) imply the first order optimality condition for problem (6.17) at $\hat{\mathbf{w}}$. Define $\beta_q^j, \gamma_q^j \ge 0$ such that

$$\beta_q^j = \begin{cases} \alpha_q^j & u_q^j > 0 \\ 0 & u_q^j \le 0 \end{cases}, \gamma_q^j = \begin{cases} \alpha_q^j & u_q^j < 0 \\ 0 & u_q^j \le 0 \end{cases}.$$

By Lemma 6.9, $|\hat{\mathbf{u}}| = \hat{\mathbf{y}}$. When $u_q^j = y_q^j > 0$, $\varphi_q^j = 0$ and $\psi_q^j \ne 0$. When $u_q^j = -y_q^j < 0$, $\psi_q^j = 0$ and $\varphi_q^j \ne 0$. By complementary slackness (6.31) and symmetry of $J_2$, (6.30) implies that

$$0 = \begin{cases} \nabla_{u_q^j} J_2 + \eta_q^j h_q^j + \beta_q^j & u_q^j > 0 \\ -\nabla_{u_q^j} J_2 + \xi_q^j h_q^j - \gamma_q^j & u_q^j < 0 \\ \eta_q^i h_q^j + \xi_q^j h_q^j & u_q^j = 0 \end{cases}. \tag{6.33}$$

For the third equation in (6.33), since $h_q^j > 0$ and $\eta_q^i, \xi_q^j \leq 0$, then $\eta_q^i = \xi_q^j = 0$ when $u_q^j = 0$. Define

$$\delta_q^j = \begin{cases} \eta_q^j & u_q^j > 0 \\ -\xi_q^j & u_q^j < 0 \\ 0 & u_q^j = 0 \end{cases}.$$

Then the complementary slackness (6.31) implies that $\eta_q^j \nabla_{\mathbf{x}} \varphi_q^j + \xi_q^j \nabla_{\mathbf{x}} \psi_q^j = \delta_q^j (\nabla_{\mathbf{x}} f_q^j + \nabla_{\mathbf{x}} h_q^j u_q^j)$. Hence (6.29) and (6.33) imply that at $\hat{\mathbf{w}}$,

$$\begin{aligned} 0 &= \nabla_{\mathbf{x}} J_1 + \sum \delta_q^j \nabla_{\mathbf{x}} (f_q^j + h_q^j u_q^j) + \sum \lambda_i \nabla_{\mathbf{x}} \phi_i \\ &\quad + A^T \Lambda_1 + A_{eq}^T \Lambda_2, \qquad (6.34) \\ 0 &= \nabla_{u_q^j} J_2 + \delta_q^j \nabla_{u_q^j} (f_q^j + h_q^j u_q^j) + \beta_q^j + \gamma_q^j. \qquad (6.35) \end{aligned}$$

The above equation is equivalent to $\nabla L|_{\hat{\mathbf{w}}} = 0$ where the Lagrangian $L$ is defined as

$$\begin{aligned} L &= J + \sum_i \lambda_i \phi_i + \Lambda_1^T (A\mathbf{x} - b) + \Lambda_2^T (A_{eq}\mathbf{x} - b_{eq}) \qquad (6.36) \\ &\quad + \sum_{q,j} \left( \delta_q^j (f_q^j + h_q^j u_q^j) + \beta_q^j (u_q^j - \bar{u}_q^j) + \gamma_q^j (-u_q^j - \bar{u}_q^j) \right). \end{aligned}$$

It is easy to check that primal feasibility (6.37), dual feasibility (6.38) and complementary slackness (6.39) are all satisfied.

$$\hat{\mathbf{x}} \in \Gamma, -\bar{\mathbf{u}} \leq \hat{\mathbf{u}} \leq \bar{\mathbf{u}}, F(\hat{\mathbf{x}}) + H(\hat{\mathbf{x}})\hat{\mathbf{u}} = 0, \qquad (6.37)$$

$$\beta_i \geq 0, \gamma_i \geq 0, \qquad (6.38)$$

$$\delta_q^j (f_q^j + h_q^j \hat{u}_q^j) = \beta_q^j (\hat{u}_q^j - \bar{u}_q^j) = \gamma_q^j (-\hat{u}_q^j - \bar{u}_q^j) = 0. \qquad (6.39)$$

Hence $\hat{\mathbf{w}}$ is a local optimum in problem (6.17). To prove the other direction, we just need to reverse the above process, which will not be elaborated. □

*Proof of Theorem 6.8.* For any nearly feasible initial value $\mathbf{z}^{(0)}$, the sequence $\{\mathbf{z}^{(k)}\}$ in Algorithm 1 converges to a local optimum of problem (6.23) by Theorem 6.2 and Lemma 6.7. Hence $\hat{\mathbf{z}} = \lim_{k\to\infty} \mathbf{z}^{(k)}$ is a local optimum of problem (6.23). Then $\hat{\mathbf{w}}$ is a local optimum of problem (6.17) by Proposition 6.10. □

### 6.2.4 Applications

The performance of SCFS is illustrated in a motion planning problem for a vehicle in a crowded environment as shown in Fig.6.11. State $x \in \mathbb{R}^2$ denotes the planar position of the vehicle. Control $u \in \mathbb{R}$ is the yaw rate. Then $\mathbf{x} = [x_0^T, x_1^T, \cdots, x_h^T]^T \in \mathbb{R}^{2(h+1)}$ is the planar trajectory. $\mathbf{u} = [u_0^T, u_1^T, \cdots, u_{h-1}^T]^T \in \mathbb{R}^h$ is the trajectory of yaw rate. $\mathbf{x}$ and $\mathbf{u}$ satisfy the dynamic equation in (6.22) for any $q$. The area occupied by the $j$-th obstacle at time step $q$

is denoted as $\mathcal{O}_{j,q} \in \mathbb{R}^2$. Define $\phi_{j,q}(\mathbf{x}) = d(x_q, \mathcal{O}_{j,q}) - d_{min}$ where $d(\cdot,\cdot)$ computes the signed distance between the vehicle and the $j$-th obstacle and $d_{min} > 0$ is the minimum distance requirement. The vehicle needs to travel from $\mathcal{A}$ to $\mathcal{Z}$ with small yaw rate without colliding with any obstacle. The optimization problem is formulated as

$$\begin{aligned} \min_{\mathbf{x}} \quad & J(\mathbf{x}) = w_1\|\mathbf{x}-\mathbf{x}^r\|_Q^2 + w_2\|\mathbf{x}\|_S^2 + w_3\|\mathbf{u}\|_R^2 \\ s.t. \quad & x_0 = \mathcal{A}, x_h = \mathcal{Z}, -\bar{\mathbf{u}} \le \mathbf{u} \le \bar{\mathbf{u}} \\ & (x_q - x_{q-1}) \times (x_{q+1} - x_q) = \|x_q - x_{q-1}\|^2 u_q \\ & \phi_{j,q}(\mathbf{x}) \ge 0, \forall j, \forall q = 1, \cdots, h-1. \end{aligned}$$

The cost function is designed to be quadratic where $w_1, w_2, w_3 \in \mathbb{R}^+$. $\|\mathbf{x}-\mathbf{x}^r\|_Q^2 := (\mathbf{x}-\mathbf{x}^r)^T Q(\mathbf{x}-\mathbf{x}^r)$ penalizes the distance from the target trajectory to a reference trajectory $\mathbf{x}^r$. $\|\mathbf{x}\|_S^2 := \mathbf{x}^T S\mathbf{x}$ concerns with the smoothness of the trajectory. $\|\mathbf{u}\|_R^2 := \mathbf{u}^T R\mathbf{u}$ penalizes the magnitude of yaw rate. In a bounded environment $X$, the second order derivatives of the nonlinear equality are also bounded. Hence the lower bound of Hessian in (6.14) can be chosen manually offline. Admittedly, the requirement of the lower bound of Hessian introduces inconvenience in implementing Algorithm 1. Relaxation of the lower bound will be considered in the future work.

The simulation was run in C++ on a MacBook of 2.3 GHz using Intel Core i7. Planning horizon is $h = 40$. The reference trajectory $\mathbf{x}^r$ is a straight line connecting the start point and the goal point (which is not shown in the figure). The green sections correspond to the static obstacles $\mathcal{O}_{j,q}$. The initial value is chosen as $\mathbf{x}^{(0)} = \mathbf{x}^r$ and $\mathbf{u}^{(0)} = 0$. The termination condition is that the step size $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|$ is smaller than 0.001.

Algorithm 1 (SCFS) converges after 11 iterations with total computation time 119.9ms and final cost 1408.25. Equation (6.27) is solved using KNITRO [22] interior-point method. The trajectories under different iterations are shown in Fig.6.11. The dark curve is the optimal path. The gray curves are the solutions before convergence, the lighter the earlier in iterations. The decomposed computation time in SCFS is shown in Fig.6.12. The computation time during each iteration is composed of two parts, the processing time, e.g. the time spent to transform the problem to a convex problem, and the optimization time, e.g. the time spent to solve the convex optimization. The processing time for each iteration is around 0.80ms (7.3% of the total time), which is very consistent through iterations. The optimization time varies from iteration to iteration.

For comparison, the planning problem is also solved using built-in non-convex optimization solvers in KNITRO directly, where the same termination condition is applied. In Table 6.2, the computation time in interior point (ITP), sequential quadratic programming (SQP), and active set (ACT) methods is compared. ITP-J, ACT-J and SQP-J refer to the case that we provide Jacobian to the solver. Otherwise, the gradient is computed numerically by central finite differences. The Hessian is computed numerically using quasi-Newton symmetric rank 1 method for all cases, which has desired accuracy and relatively small computation load. For ITP, the computation time reduced 96.7% when Jacobian is provided[10]. As shown

[10]Note that the number of iterations and the final cost are not identical in ITP-J and ITP. That is due

Table 6.2: Comparison among SCFS, ITP, ACT and SQP.

| Method | Computation Time (ms) | Iterations | Cost |
|---|---|---|---|
| SCFS | 119.9 | 11 | 1408.25 |
| ITP-J | 419.4 | 94 | 1489.72 |
| ITP | 14031.4 | 90 | 1489.58 |
| ACT-J | 5970.2 | 718 | 1489.71 |
| SQP-J | 20632.7 | 848 | 3487.26 |

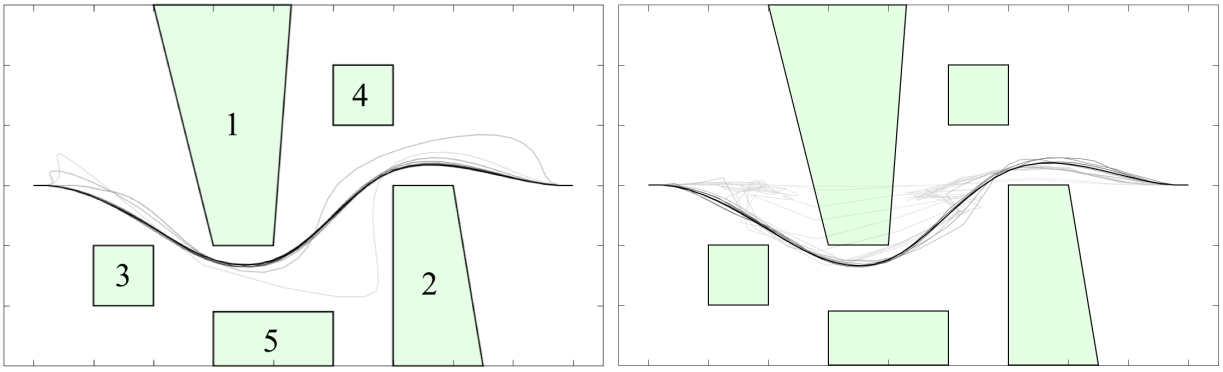
in the table, the other two methods (ACT and SQP) are much slower than ITP and require more iterations to converge. As ITP-J is most efficient, we only provide detailed comparison between SCFS and ITP-J. Figure 6.13 shows the run time statistics of the two methods, e.g cost $J(\mathbf{x})$ and feasibility (absolute value of the maximum violation of the constraints) during each iteration. The cost profiles have similar characteristics under the two methods, i.e. the cost first jumps up to make the problem feasible, then drops down to the optimal value. The difference is that the cost drops down much faster in SCFS than in ITP-J, as shown in the zoomed-in figure. Moreover, ITP-J finds a local optimum with higher cost than SCFS. The trajectories in ITP-J are shown in Fig.6.11b, where the color of the trajectories corresponds to the iteration number, the lighter the earlier. Different from the trajectories in SCFS, the trajectories in the early iterations of ITP-J are not feasible and the step size in ITP-J is smaller.

The performance of SCFS under different conditions is shown in Fig.6.14, where the number of sampling points $h$ changes from 20 to 40, and the number of obstacles changes from 1 to 5. The start point and the goal point are fixed. We use "$l$ obstacles" to denote the case that we only consider polygons 1 to $l$ in Fig.6.11a. The average processing time in SCFS scales linearly with respect to $h$ and $l$. In general, the total computation time in SCFS also grows with respect to $h$ and $l$. However, the trend in the total computation time is not as clear as in the average processing time, since the optimization time for each iteration and the number of iterations up to convergence are highly problem-specific and sensitive to initial conditions. Nonetheless, SCFS always outperforms ITP-J in terms of computation time as shown in Fig.6.11a. In those cases, SCFS either finds a better optima than ITP-J or converges to the same local optima as ITP-J.

## 6.3 Conclusion

This chapter introduced two fast algorithms for real time motion planning based on the convex feasible set. The CFS algorithm can handle problems that have convex cost function and non-convex constraints which are usually encountered in robot motion planning.

to the fact that the constraint is non differentiable at some points, e.g. at a corner point of an obstacle. At those points, the sub-gradients computed by the Jacobian and by the central finite differences are different.




(a) The trajectories in SCFS. (b) The trajectories in ITP-J.

Figure 6.11: The motion planning problem in 2D.

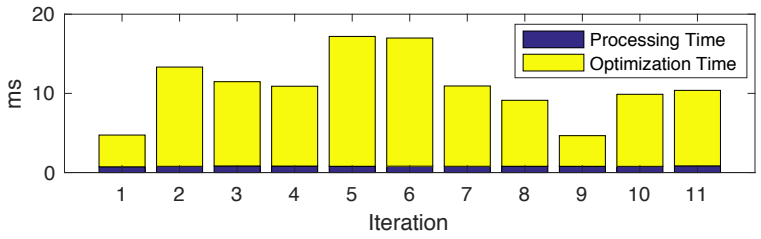



Figure 6.12: Decomposition of the computation time in SCFS.

By computing a convex feasible set within the non-convex constraints, the non-convex optimization problem was transformed into a convex optimization. Then by iteration, we can efficiently eliminate the error introduced by the convexification. It was proved that the proposed algorithm was feasible and stable. Moreover, it can converge to a local optimum if the initial reference was nearly feasible. The performance of CFS was compared to that of ITP and SQP. It was shown that CFS outperformed ITP and SQP in computation time, hence better suited for real time applications. In order to handle nonlinear equality constraints, the SCFS algorithm was proposed to solve the constrained trajectory optimization problem by relaxation and convexification. The feasibility, convergence and optimality of the algorithm were proved. Simulation results also validated the efficiency of the algorithm and illustrated its advantage over existing algorithms.

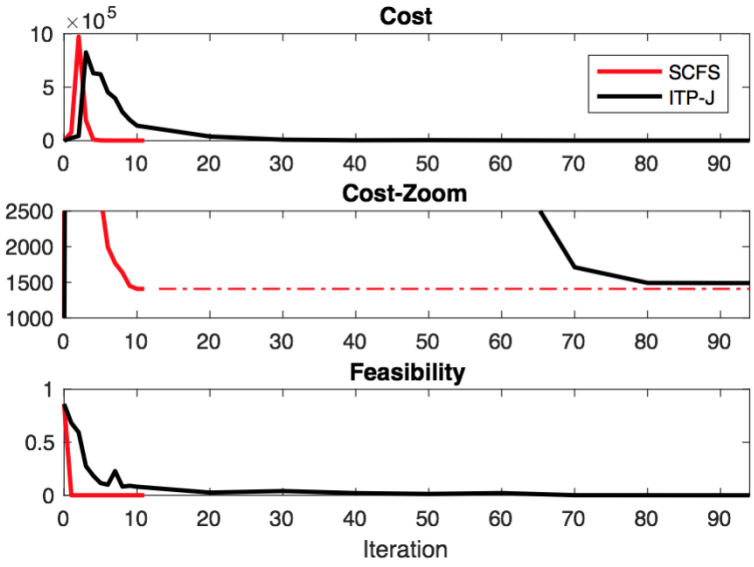



Figure 6.13: Profiles of cost and feasibility in SCFS and ITP-J.

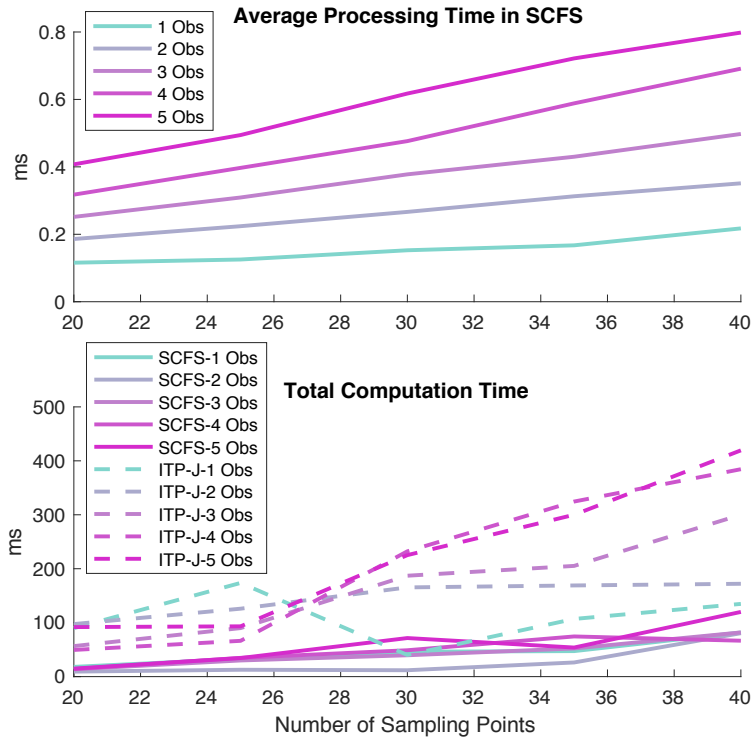



Figure 6.14: Performance of SCFS under different conditions.

# Chapter 7

# Evaluation of Systems under Information Asymmetry

In the previous chapters, we discussed the methods to design the behavior of a robot. The performance of the design needs to be evaluated in the context of multi-agent system. The key feature of such system is that the information is asymmetric in the sense that each agent only knows its own model. In this chapter, we discuss methods to analyze the performance of MAS under information asymmetry.

## 7.1 Overview

In the robotics community, the idea of multi-robot teams has received much attention due to its wide applications in search and rescue, intelligent manufacturing, and unknown environment exploration. Conventionally, distributed control laws are designed for each robot agent [118]. Following the control laws specified by the intelligent designer, who has an omniscient perspective, the agents do not need much intelligence. Although it is not explicitly stated that the control laws are globally known, the effectiveness of the controllers is based on the confidence that all agents will exactly follow the designed control laws, i.e. an agent just needs to follow his own control law, while other agents will take care of everything else by following their designed control laws. This strategy usually works well on problems with predefined environment and interaction patterns, but has limited extendability, as it is hard to design a universal control law that leads to desired results in many cases, especially when the system topology is time-varying (interactions are happening among different agents at different time).

Another approach is to design a cost function and a corresponding reasoning strategy (how to minimize cost under uncertainties) following the behavior system discussed in Chapter 1.3 (in Fig.1.2) for each agent and let the behavior evolve during interactions [81], so that the multi-agent system can be self-organized [91]. The cost function determines the behavior of each agent, which can be understood as the agent's character. In certain cases, the cost

function of an agent may not be clear to other agents, which is natural during HRI. This scenario is called information asymmetry [113]. The problem of interest is how to design the reasoning strategy so that the desired behavior will evolve even in the case of information asymmetry. This is not only important from the control point of view, but also essential in understanding how cooperation evolves among humans and how robots may cooperate with humans.

The reasoning strategy for an agent consists of the learning strategy (estimating the behavior of other agents) and the control strategy (minimizing its own cost based on the estimates). The learning strategy corresponds to the learning module and the control strategy corresponds to the logic in the behavior system in Fig.1.2. From the agent perspective, most learning and control strategies follow a centralized design philosophy, such as reinforcement learning [72] and adaptive control [67], which adjust agent's estimation if its observation deviates from its prediction. However, in MAS, where each agent acts on its estimation of the unknowns, the gap between an agent's observation and prediction can be caused by the following two factors:

1. the agent's wrong estimation (wrong prediction);
2. other agents' wrong estimations and subsequent improper actions (deviated observations).

Since the centralized design strategies (called the "Blame-Me" strategy) do not address the second factor, they are likely to produce instability in MAS as will be shown in this chapter. To deal with this problem, a new "Blame-All" strategy that considers both factors will be proposed.

Given the reasoning strategies, it is important to determine **whether the agents can cooperate**. To answer the question, the closed loop response of the MAS needs to be analyzed, especially the response in the equilibrium where single agent cannot change its control law for lower cost. For example, in the Bayesian game theory, a Bayesian Nash Equilibrium (BNE) [146] is defined as a profile of control laws that minimizes the expected cost for each agent given its belief about others' costs. However, since the learning process is not considered, BNE is not suitable to analyze dynamic systems with information asymmetry. To deal with this problem, the Trapped Equilibrium (TE) is introduced in this chapter to characterize the control laws (as well as the resulting time-invariant closed loop system) when all learning processes have converged (but may or may not converge to the true values). The aforementioned question will be answered regarding the TE, e.g. whether the cooperative goal can be achieved in the TE.

This chapter discusses the closed loop performance of MAS in the TE under different learning and control strategies in a dynamic simultaneous game, and proposes the Blame-All strategy to improve the closed loop performance. The remainder of this chapter is organized as follows: the mathematical problem is formulated in Section 7.2; Section 7.3 sets up an analytical framework for quadratic games; Section 7.3.3 shows the drawbacks of the

Blame-Me strategy; the Blame-All strategy is discussed in Section 7.3.4; Section 7.4 gives an illustrating example; and Section 7.5 concludes the chapter.

The notations used in this chapter are listed below.

- $\underline{x}, \underline{u}_i, \underline{U}$

  The observed state and control inputs.

- $x^o(k)$

  The optimal state at $k$ under complete information given $\underline{x}(k-1)$.

- $u_i^o, U^o$

  The optimal control inputs under complete information.

- $\delta x, \delta u_i, \delta U$

  The deviation of the state and control inputs due to information asymmetry: $\delta x = \underline{x} - x^o$, $\delta u_i = \underline{u}_i - u_i^o$, $\delta U = \underline{U} - U^o$.

- $\hat{\cdot}^{(i)}, \tilde{\cdot}^{(i)}$

  The estimate and estimation error of a parameter by agent $i$ such that $\tilde{\cdot}^{(i)} = \hat{\cdot}^{(i)} - \cdot$, e.g. $\hat{\theta}_j^{(i)}(k)$ is the estimate of $\theta_j$ by agent $i$ at time $k$ and $\tilde{\theta}_j^{(i)}(k) = \hat{\theta}_j^{(i)}(k) - \theta_j$.

- $\widehat{\cdot}$ , $\widetilde{\cdot}$

  The estimate and estimation error of a parameter regardless of who makes the estimation.

## 7.2 Evaluation of the Interactions in Multi-Agent Systems

### 7.2.1 The Multi-Agent Model

Denote the system state as $x \in \mathbb{R}^n$ where $n$ is the dimension of the state space. The agents in the system are indexed as $i = 1, ..., N$, and they provide inputs $u_1, u_2, ..., u_N$ to the system, where $u_i \in \mathbb{R}^{r_i}$ for all $i$. $U \in \mathbb{R}^r$ is the stack of all $u_i$'s, where $r = \sum_i r_i$. Assume the system dynamics is affine with respect to $u_i$'s, e.g.

$$x(k+1) = f(x(k)) + \sum_{i=1}^{N} h_i(x(k))u_i(k). \tag{7.1}$$

Assume there is no direct communication among the agents. Then for any agent $i$, it can only choose its control $u_i$ based on the observations (the current state and the previous inputs of other agents) and its reference goal $G_i \subset \mathbb{R}^n$ by minimizing a cost function $J_i(x, u_i, G_i)$,

$$u_i = \arg\min J_i(x, u_i, G_i). \tag{7.2}$$

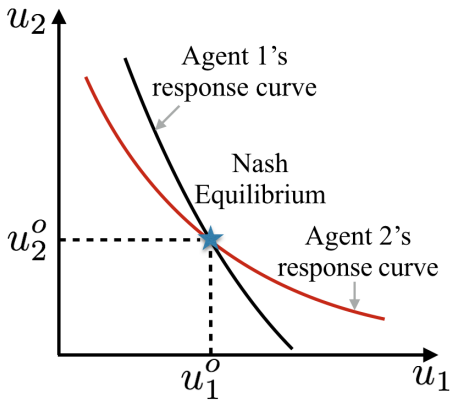
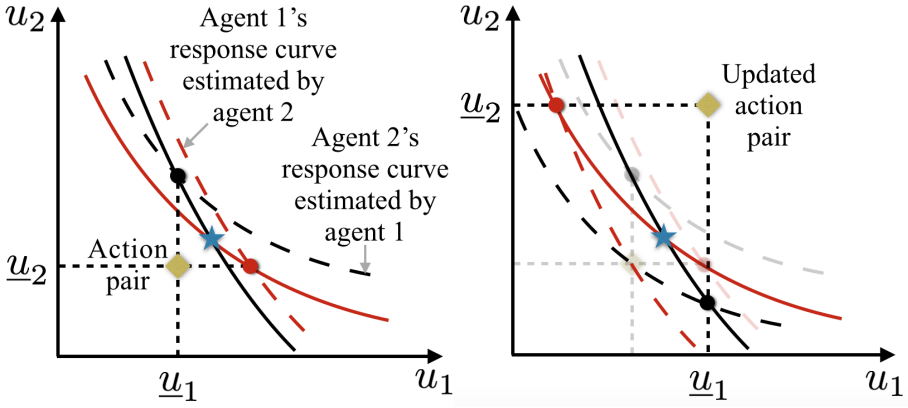



Figure 7.1: The response curve and the Nash Equilibrium



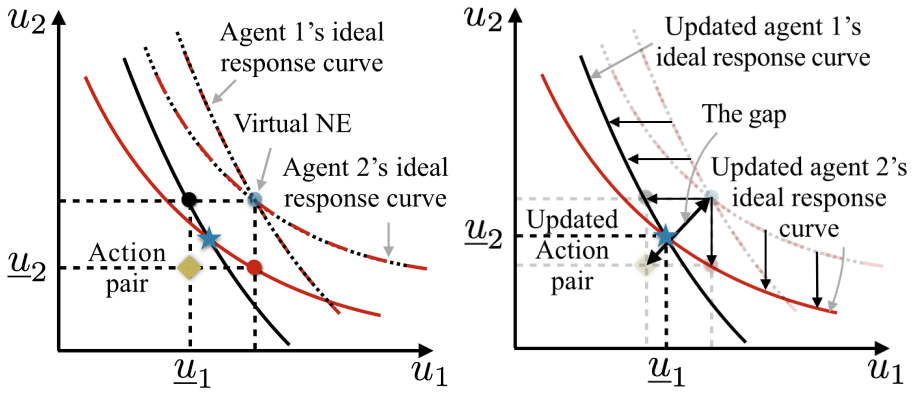

(a) Time step $k$ (b) Time step $k+1$

Figure 7.2: The update of the control law under the adaptive algorithm (the Blame-Me strategy)

For simplicity, it is assumed that $G_i = G$ for all $i$.

### 7.2.2 The Simultaneous Dynamic Game

Since all agents' inputs can affect the system state $x$, the solution of the optimization problem in (7.2) is indeed a response curve, e.g. $u_i = g_i(x, G_i, u_{-i})$ where $u_{-i}$ denotes the inputs of all other agents except the agent $i$. The intersection of the response curves among all agents is the Nash Equilibrium. The resulting control laws $\{u_i^o\}_i$ are optimal in the sense that an agent cannot get a lower cost by deviating from this control law alone. Figure 7.1 illustrates the NE in the $(u_1, u_2)$ function space in a two-agent case. However, the NE is only defined for systems with *complete information*, e.g. the cost functions $J_i$'s or the response




(a) Time step $k$ (b) Time step $k+1$

Figure 7.3: The update of the control law under the Blame-All strategy

curves $g_i$'s are globally known. When those are not globally known, the system is with *incomplete information*. The scenario that the cost functions known to different agents are not identical is called *information asymmetry*. In this chapter, the case that an agent only knows its own cost function is considered, in which case the agents can only act on their estimates of other's response curves.

Figure 7.2 illustrates the reasoning process under the adaptive algorithm (or the Blame-Me strategy). In Fig.7.2a, agent $i$ ($i = 1, 2$) chooses its action $\underline{u}_i$ in the intersection of its response curve (the solid curve) and the estimated response curve of the other agent (the dashed curve of the same color). However, the action pair $\underline{u}_1, \underline{u}_2$ in the yellow dot deviates from the agents' predictions of what the other would do, which provides incentives for the agents to update their estimations. From agent 1's perspective, since the deviation is only attributed to its wrong estimation, the updated estimate of agent 2's response curve should go through the observed action pair $\underline{u}_1, \underline{u}_2$ in Fig.7.2a. A new control law can be chosen based on the new estimation as shown in Fig.7.2b. Agent 2 goes through the same reasoning. The implicit assumption under this strategy is that other agents are optimal given the observed data, which is a common assumption in data-driven estimation of cost functions [16]. However, this strategy is not ideal as it results in 'over-reaction' as the new action pair moves farther away from the NE in Fig.7.2b.

In the Blame-All strategy, it is assumed that other agents are only optimal given their estimates. To predict what others would do, how they are estimating 'me' should also be considered. For example, in Fig.7.3a, the response curve of agent 1 is estimated by both agents (shown as the red and black dashed lines and denoted as Agent 1's ideal response curve). If the two agents follow the same initialization rule, they will have consensus on the estimates of the response curves. The ideal response curves define a virtual NE (the light blue dot). An agent's best action is to choose the corresponding action on its response curve, assuming others choosing actions in the virtual NE. Since the virtual NE is not the true NE,

the observed action pair is still deviating from the predicted. From agent 1's perspective, the deviation is caused both by its wrong estimation and the agent 2's improper action. Hence the estimate of agent 2's response curve is updated only to compensate the gap (between the virtual NE and the action pair) in the $u_2$ direction, while the gap in the $u_1$ direction is compensated by updating agent 1's ideal response curve as shown in Fig.7.3b. In this way, the true response curves are found and the NE is achieved.

### 7.2.3 Evaluation of the Interactions: the Trapped Equilibrium

Following the above procedure, the update of the control law can be continued. The problems of interest are: under different strategies, whether the parameter estimates converge to true values and what is the resulting closed loop system.

**The Trapped Equilibrium** is defined as the time-invariant closed loop system where all agents do not have incentives to change their control laws, as their learning processes converge. In Fig.7.2 and Fig.7.3, being in the TE means that the yellow dot remains in the same location under the specific reasoning strategy. Under the TE, the closed loop stability around the goal point will be analyzed to answer the question whether the agents can cooperate.

## 7.3 The Equilibria in Quadratic Games

In this section, the scope is narrowed down to quadratic games, which rise naturally in multi-robot cooperations [126].

### 7.3.1 The Model and Assumptions

Let the system goal $G$ be driving the state $x$ to the target state in the origin. Consider a quadratic game, i.e.

$$J_i(x(k)) = x^T(k+1)Px(k+1) + u_i^T(k)u_i(k)\theta_i, \forall i, \tag{7.3}$$

where $\theta_i \in \mathbb{R}$ encodes the preference of agent $i$ which is not known to others, and $P \in \mathbb{R}^{n\times n}$ is positive definite, which is chosen such that the closed loop system in the Nash Equilibrium is globally asymptotically stable around the origin. Define $R = diag(\theta_1 I_{r_1}, ..., \theta_N I_{r_N}) \in \mathbb{R}^{r\times r}$. Let $B_i = h_i(x(k))$ be constant and $B = [\ B_1 \quad \cdots \quad B_N\ ] \in \mathbb{R}^{n\times r}$. Denote the matrices that contain the diagonal and the off-diagonal entries of $R + B^T P B$ as $H_1$ and $H_2$ respectively. It is assumed that $H_1, H_2$ are invertible. If $H_1$ is singular, then some $J_i$ does not depend on some entries in $u_i$; if $H_2$ is singular, then there are no cross coupling terms among some control pairs.

### 7.3.2 The Benchmark System in the Nash Equilibrium

For any agent $i$, the response curve $\partial J_i/\partial u_i = 0$ with respect to the cost function (7.3) is

$$\theta_i u_i(k) = -B_i^T P x(k+1), \tag{7.4}$$

where others' inputs are implicitly contained in $x(k+1)$. By (7.1) and using estimates for other agents' inputs,

$$u_i(k) = -[\theta_i I_{r_i} + B_i^T P B_i]^{-1} B_i^T P[f(\underline{x}(k)) + \sum_{j \neq i} B_j \hat{u}_j^{(i)}(k)]. \tag{7.5}$$

Hence the optimal control law is a linear combination of a state feedback control law and a predictive control law (where the actions of other agents are predicted). The predictive control law varies under different strategies, which will be discussed in the following two sections.

In the Nash Equilibrium, agents have correct predictions, i.e. $\hat{u}_j^{(i)}(k) = u_j(k)$. Stacking the control law (7.5) for all agents and moving the inverse terms to the left, then $H_1 U(k) = -B^T P f(\underline{x}(k)) - H_2 U(k)$. Hence the saddle solution with respect to (7.3) is

$$U^o(k) = \begin{bmatrix} u_1^o(k) \\ \vdots \\ u_N^o(k) \end{bmatrix} = -[R + B^T P B]^{-1} B^T P f(\underline{x}(k)), \tag{7.6}$$

which coincides with the solution to a centralized cost function, e.g. $J(x(k)) = x^T(k+1)Px(k+1) + U^T(k)RU(k)$.

Denote the system's feedback gain as $K = [R + B^T P B]^{-1} B^T P$, the agent $i$'s feedback gain as $K_i = T_i K$ where $T_i = [0, ...0, I_{r_i}, 0, ...0] \in \mathbb{R}^{r_i \times n}$. Hence the optimal control law for agent $i$ under complete information is $u_i^o(k) = -K_i f(\underline{x}(k))$. Let $y_{ij} \in \mathbb{R}^{r_i \times r_j}$ be the block entries of $\left[R + B^T P B\right]^{-1}$, which depends on $\theta_1, ...\theta_N$. $y_{ij}$ encodes interactions among agents, as the control strategy of agent $i$ depends on parameters of agent $j$, i.e. $K_i = \sum_j y_{ij} B_j^T P$.

The closed loop system in the Nash Equilibrium is

$$\underline{x}(k+1) = [I - BK] f(\underline{x}(k)). \tag{7.7}$$

### 7.3.3 The Blame-Me Strategy and the Trapped Equilibrium

#### The Adaptive Control Algorithm

In the case that $\theta_j$ is only known to agent $j$, according to the control law (7.6), the adaptive control law of agent $i$ can be written as

$$u_i(k) = -T_i[\hat{R}^{(i)} + B^T P B]^{-1} B^T P f(\underline{x}(k)), \tag{7.8}$$

where $\hat{R}^{(i)} = diag(\hat{\theta}_1^{(i)} I_{m_1}, ..., \theta_i I_{m_i}.., \hat{\theta}_N^{(i)} I_{m_N})$ contains the estimated parameters. Agent $i$'s learning objectives are other agents' response curves (7.4), e.g. the solid curves in Fig.7.1. Since $\theta_j$ is a scalar, it is intuitive to do the parameter estimation using the following equation with reduced order,

$$\theta_j \underline{u}_j^T(k)\underline{u}_j(k) = -\underline{u}_j^T(k) B_j^T P \underline{x}(k+1). \tag{7.9}$$

The parameter $\theta_j$ can be learned by agent $i$ using the recursive least square (RLS) method [67], i.e.

$$\begin{aligned} \hat{\theta}_j^{(i)}(k+1) &= \hat{\theta}_j^{(i)}(k) + e_j^{(i)}(k+1)\underline{u}_j^T(k)\underline{u}_j(k)F(k), \\ e_j^{(i)}(k+1) &= -\underline{u}_j^T(k) B_j^T P \underline{x}(k+1) - \hat{\theta}_j^{(i)}(k)\underline{u}_j^T(k)\underline{u}_j(k), \end{aligned} \tag{7.10}$$

where $F(k) \in \mathbb{R}^+$ is the learning gain. For simplicity, $F(k)$ is set to be $(\underline{u}_j^T(k)\underline{u}_j(k))^{-2}$ if $\underline{u}_j(k) \neq 0$ and 0 otherwise. Note that $\underline{u}_j(k)$ is known to agent $i$ at $k+1$.

### The Closed Loop System

**Proposition 7.1** (**Multiplicative uncertainty**). *The closed loop dynamics of the system when all agents are using the control law (7.8) follow from*

$$\underline{U}(k) = (I + \Delta(k))U^o(k), \tag{7.11}$$

$$\underline{x}(k+1) = [I - B[I + \Delta(k)]K] f(\underline{x}(k)), \tag{7.12}$$

*where $\Delta(k) \in \mathbb{R}^{r \times r}$, whose diagonal entries are all zero.*

*Proof.* Notice that

$$\begin{aligned} \underline{u}_i(k) - u_i^o(k) &= -T_i\{[\hat{R}^{(i)}(k) + B^T P B]^{-1} - [R + B^T P B]^{-1}\} B^T P f(\underline{x}(k)) \\ &= T_i[\hat{R}^{(i)}(k) + B^T P B]^{-1} \tilde{R}^{(i)}(k) [R + B^T P B]^{-1} B^T P f(\underline{x}(k)) \\ &= -T_i[\hat{R}^{(i)}(k) + B^T P B]^{-1} \tilde{R}^{(i)}(k) U^o(k) \\ &= [\hat{y}_{i1}^{(i)}\tilde{\theta}_1^{(i)}, ..., \hat{y}_{i(i-1)}^{(i)}\tilde{\theta}_{i-1}^{(i)}, 0, \hat{y}_{i(i+1)}^{(i)}\tilde{\theta}_{i+1}^{(i)}, ..., \hat{y}_{iN}^{(i)}\tilde{\theta}_N^{(i)}] U^o(k). \end{aligned}$$

Stack the equation for all agents,

$$\underline{U}(k) - U^o(k) = \Delta(k) U^o(k), \tag{7.13}$$

$$\Delta(k) = -\begin{bmatrix} 0 & \hat{y}_{12}^{(1)}\tilde{\theta}_2^{(1)} & \cdots & \hat{y}_{1N}^{(1)}\tilde{\theta}_N^{(1)} \\ \hat{y}_{21}^{(2)}\tilde{\theta}_1^{(2)} & 0 & \cdots & \hat{y}_{2N}^{(2)}\tilde{\theta}_N^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{N1}^{(N)}\tilde{\theta}_1^{(N)} & \hat{y}_{N2}^{(N)}\tilde{\theta}_2^{(N)} & \cdots & 0 \end{bmatrix}. \tag{7.14}$$

Hence (7.11) and (7.12) hold. □

**Remark 7.1.** *(7.12) shows that the uncertainty introduced by information asymmetry is multiplicative and (7.14) indicates the structure of the uncertainty in the Blame-Me strategy.*

**The Trapped Equilibrium**

Note that $\hat{\theta}_j^{(i)}(k) = \hat{\theta}_j^{(m)}(k)$, $\forall i, m \neq j$ if both agent $i$ and agent $m$ are using the same learning algorithm with the same initial conditions. Let $\widehat{\theta}_j(k) = \hat{\theta}_j^{(i)}(k)$, $\forall i \neq j$. Define $\widehat{R}(k) = diag(\widehat{\theta}_1(k)I_{nu_1}, ..., \widehat{\theta}_N(k)I_{nu_N})$.

**Proposition 7.2** (**Convergence of the learning algorithm**). *In system (7.12) with uncertainty (7.14), for all initial conditions, the learning algorithm (7.10) converges, i.e.* $\lim_{k\to\infty} \widehat{R}(k) = \widehat{R}^e$ *and consequently* $\lim_{k\to\infty} \Delta(k) = \Delta^e$, *if*

$$(\Delta^e + I)^T[(\widehat{R}^e + B^T P B)\Delta^e + \widetilde{R}^e] = 0. \tag{7.15}$$

*Proof.* When $\underline{u}_j(k) = 0$, $\widetilde{\theta}_j(k+1) = \widetilde{\theta}_j(k)$ is trivial. Consider the case $\underline{u}_j(k) \neq 0$. Note that (7.4) holds only for $x = x^o$ and $u_j = u_j^o$, e.g. $\theta_j(\underline{u}_j(k) - \delta u_j(k)) = -B_j^T P(\underline{x}(k+1) - B\delta U(k))$. Hence

$$\theta_j \underline{u}_j(k) + B_j^T P \underline{x}(k+1) = (\theta_j T_j + B_j^T P B)\delta U(k). \tag{7.16}$$

Thus the estimate of the reaction curve is biased, so is $\widehat{\theta}_j$. Multiply $\underline{u}_j^T(k)$ on both sides, then $-\widetilde{\theta}_j(k+1)\underline{u}_j^T(k)\underline{u}_j(k) = \underline{u}_j^T(k)T_j(R + B^T P B)\Delta(k)U^o(k)$. Since $\underline{u}_j(k) = T_j(I + \Delta(k))U^o(k)$, the relationship between the estimation error $\widetilde{\theta}_j(k+1)$ and the uncertainty $\Delta(k)$ follows from

$$-\widetilde{\theta}_j(k+1) = \frac{(U^o(k))^T(I + \Delta(k))^T T_j^T T_j(R + B^T P B)\Delta(k)U^o(k)}{(U^o(k))^T(I + \Delta(k))^T T_j^T T_j(I + \Delta(k))U^o(k)}. \tag{7.17}$$

For all initial conditions, the learning algorithm converges if $(I+\Delta^e)^T T_j^T T_j(R+B^T P B)\Delta^e = -\widetilde{\theta}_j^e(I + \Delta^e)^T T_j^T T_j(I + \Delta^e)$, which can be rearranged as:

$$(\Delta^e + I)^T[T_j^T T_j(\widetilde{\theta}_j^e + R + B^T P B))\Delta^e + \widetilde{\theta}_j^e T_j^T T_j] = 0. \tag{7.18}$$

Stacking the equations for all $j$, (7.15) holds. □

**Remark 7.2.** *(7.14) and (7.15) determine the TEs of the MAS, e.g.* $\Delta^e = \widetilde{R}^e = 0$, *or* $\Delta^e = -[\widehat{R}^e + B^T P B]^{-1}\widetilde{R}^e$. *The first TE is efficient as it is identical with the NE. When* $\Delta^e \neq 0$, *the cooperation under the second TE is inefficient.*

**Remark 7.3** (**Unstable modes**). *When* $N = 2$, *suppose* $B_j^T P B_j = c_j I$ *for some scalar* $c_j$. *Then* $\widehat{R}^e = -H_1 + R, \Delta^e = H_2^{-1}H_1$ *satisfies (7.14) and (7.15), hence defines a TE. Moreover,* $(I + \Delta^e)K = H_2^{-1}(H_1 + H_2)[H_1 + H_2]^{-1}B^T P = H_2^{-1}B^T P$. *By (7.12), the closed loop dynamics is*

$$\underline{x}(k+1) = [I - BH_2^{-1}B^T P]f(\underline{x}(k)), \tag{7.19}$$

*which can be unstable when* $||H_2|| \to 0$.

### 7.3.4 The Blame-All Strategy and the Trapped Equilibrium

**The Algorithm**

To overcome the instability in the Blame-Me strategy, the Blame-All strategy is proposed. Putting (7.8) in the form of (7.5), it is clear that the estimate of $u_j(k)$ in the Blame-Me strategy is equivalent to $\hat{u}_j^{(i)}(k) = -[\hat{\theta}_j^{(i)} + B_j^T P B_j]^{-1}[f(\underline{x}(k)) + \sum_{m\neq i,j} B_m \hat{u}_m^{(i)}(k) + B_i u_i(k)]$, which is biased since agent $j$ does not know $u_i(k)$ in advance. The term $u_i(k)$ should also be estimated, e.g.

$$\hat{u}_j^{(i)}(k) = -[\hat{\theta}_j^{(i)} + B_j^T P B_j]^{-1}[f(\underline{x}(k)) + \sum_{m\neq j} B_m \hat{u}_m^{(i)}(k)], \tag{7.20}$$

where the estimate $u_i^{(i)}(k)$ should be calculated using the same algorithm in estimating other agents' behavior as in (7.20). Following the procedure in obtaining (7.6) from (7.5), (7.20) can be solved for all $j = 1, \cdots, N$, e.g.

$$\hat{u}_j^{(i)}(k) = -T_j[\widehat{R}^{(i)}(k) + B^T P B]^{-1} B^T P f(\underline{x}(k)), \tag{7.21}$$

where $\widehat{R}^{(i)} = diag(\hat{\theta}_1^{(i)} I_{r_1}, ..., \hat{\theta}_i^{(i)} I_{r_i} .., \hat{\theta}_N^{(i)} I_{r_N})$. Specifically, $\hat{\theta}_i^{(i)}$ is the estimate of $\hat{\theta}_i^{(j)}$ for $j \neq i$ made by agent $i$ to compensate for others' wrong estimations of $\theta_i$. When all agents are using the same algorithm with the same initial condition, they should agree on their estimates, e.g. $\widehat{R}^{(i)} = \widehat{R}^{(j)} = \widehat{R}$. Then by (7.21), $\hat{u}_j^{(i)}(k)$ does not depend on $i$, e.g. $\hat{u}_j^{(i)}(k) = \widehat{u}_j(k)$, $\forall i$. Stack all $\widehat{u}_j(k)$,

$$\widehat{U}(k) = -[\widehat{R}(k) + B^T P B]^{-1} B^T P f(\underline{x}(k)). \tag{7.22}$$

$\widehat{U}(k)$ is the virtual NE (the light blue dot in Fig.7.3) that is consensual among all agents. Plugging (7.22) into (7.5), agent $i$'s control law becomes

$$u_i(k) = -[\theta_i + B_i^T P B_i]^{-1} B_i^T P[f(\underline{x}(k)) + (B - B_i T_i)\widehat{U}(k)]. \tag{7.23}$$

From agent $i$'s perspective, agent $j$ is only optimal given its estimate $\widehat{U}(k)$. Equation (7.4) needs to be rewritten as

$$\theta_j u_j(k) = -B_j^T P \hat{x}^{(j)}(k+1), \tag{7.24}$$

where $\hat{x}^{(j)}(k+1) = f(\underline{x}(k)) + (B - B_j T_j)\widehat{U}(k) + B_j \underline{u}_j(k)$. In the Blame-Me strategy (7.9), the observed $\underline{x}(k+1)$ is used for the $x(k+1)$ term, which is biased, since it may not be the future state predicted by agent $j$ when it was doing control at time $k$. In (7.24), on the other hand, the predicted future state $\hat{x}^{(j)}(k+1)$ is used. This calculation is made possible by using the consensus $\widehat{U}(k)$. Using $F(k)$ in (7.10), the following unbiased RLS algorithm can be formulated,

$$\begin{aligned} \widehat{\theta}_j(k+1) &= \widehat{\theta}_j(k) + e_j(k+1)\underline{u}_j^T(k)\underline{u}_j(k)F(k), \\ e_j(k+1) &= -\underline{u}_j^T(k) B_j^T P \left[f(\underline{x}(k)) + (B - B_j T_j)\widehat{U}(k) + B_j \underline{u}_j(k)\right] - \widehat{\theta}_j(k)\underline{u}_j^T(k)\underline{u}_j(k). \end{aligned} \tag{7.25}$$

### The Closed Loop System

**Proposition 7.3** (**Multiplicative uncertainty**). *The closed loop dynamics of the system when all agents are using control law (7.23) follow from (7.11) and (7.12) where*

$$\Delta(k) = H_1^{-1} H_2 [\widehat{R}(k) + B^T P B]^{-1} \widetilde{R}(k). \tag{7.26}$$

*Proof.* Subtracting $u_i^0(k)$ from (7.23), $\underline{u}_i(k) - u_i^o(k) = -\left[\theta_i + B_i^T P B_i\right]^{-1} B_i^T P(B - B_i T_i)\widetilde{U}(k) = -T_i H_1^{-1} H_2 \widetilde{U}(k)$, where $\widetilde{U}(k) = \widehat{U}(k) - U^o(k) = -[\widehat{R}(k) + B^T P B]^{-1}\widetilde{R}(k) U^o(k)$. Stack the equations for all agents, then $\underline{U}(k) - U^o(k) = \Delta(k) U^o(k)$ where $\Delta(k)$ follows from (7.26). Equations (7.11) and (7.12) hold. □

### The Trapped Equilibrium

**Proposition 7.4** (**Convergence of the learning algorithm**). *In system (7.12) with uncertainty (7.26) and learning algorithm (7.23), $\widehat{\theta}_j(k)$ is bounded $\forall j = 1, ..N$. It converges to $\theta_j$ in finite time steps if and only if the set $\{u_j(k) \equiv 0\}$ does not contain any trajectory of the closed loop system.*

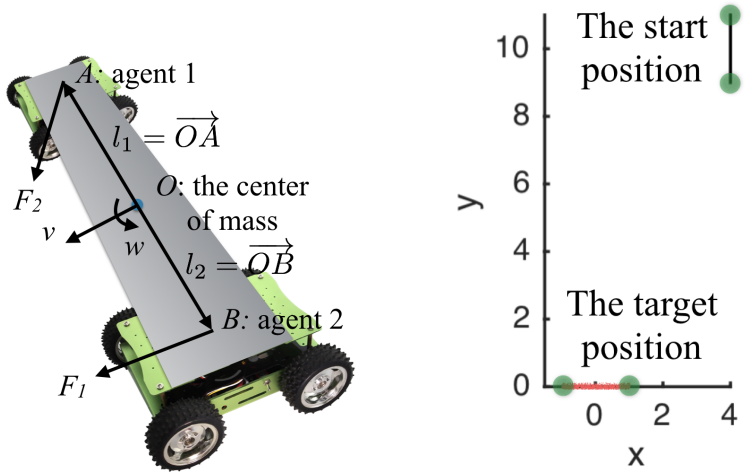
*Proof.* When $\underline{u}_j(k) \neq 0$, $e_j(k+1) = -\widetilde{\theta}_j(k)\underline{u}_j^T(k)\underline{u}_j(k)$. Since $F(k) = (\underline{u}_j^T(k)\underline{u}_j(k))^{-2}$, then $\widehat{\theta}_j(k+1) = \widehat{\theta}_j(k) - \widetilde{\theta}_j(k) = \theta_j$. When $\underline{u}_j(k) = 0$, $\widehat{\theta}_j(k+1) = \widehat{\theta}_j(k)$. Hence $\widehat{\theta}_j(k)$ is bounded by $\max\{\theta_j, \widehat{\theta}_j(0)\}$, and $\widehat{\theta}_j(k)$ converges to true $\theta_j$ if $\exists k^*$ s.t. $\underline{u}_j(k^*) \neq 0$. □

Proposition 7.4 implies two TEs under the Blame-All strategy: 1) the parameter estimation converges in finite steps and the system converges to the benchmark system in (7.7); 2) the system is trapped in the set $\cup_j\{u_j(k) \equiv 0\}$. To get rid of the second TE, agent $j$ is allowed to choose a random control input when the optimal input in (7.23) is always zero, which is called a perturbation to the system.

**Theorem 7.5** (**System performance under the Blame-All strategy**). *For any $N$, the closed loop system under the Blame-All strategy will converge to the benchmark system (7.7) in the Nash Equilibrium, if the agent $j$ is allowed to perturb the system when the system is trapped in $\{\underline{u}_j(k) \equiv 0\}$, but $B_j^T P \underline{x}(k) \neq 0$.*

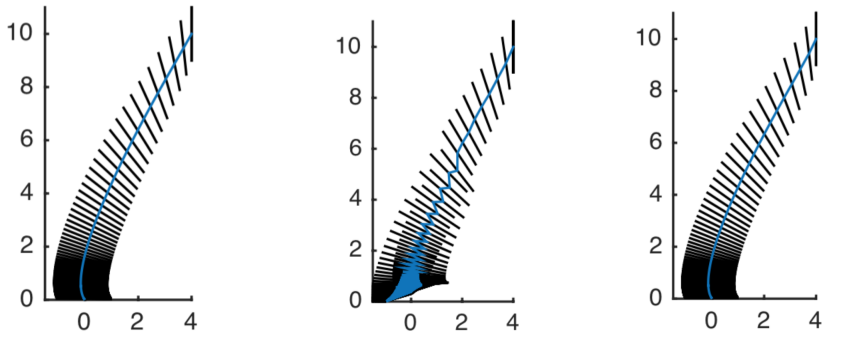
*Proof.* The proof is in two steps. First, we will show that under certain initial conditions, the system will converge to the benchmark system in two time steps. Then we show that a perturbation to the system is equivalent to starting over with a new set of initial conditions. Hence the system will always converge to the benchmark system.

By Proposition 7.4, if $u_j(0) = -T_j(\Delta(0) + I)Kf(x(0)) \neq 0, \forall j$, the parameters converge at $k = 1$, i.e. $\widehat{R}(1) = R$. By (7.26), $\widehat{R}(1) = R$ implies that $\Delta(1) = 0$, which further implies that $\underline{U}(1) = U^o(1)$ and $\underline{x}(2) = x^o(2)$ and $\widehat{R}(k) = R, \Delta(k) = 0$ for all $k \geq 2$. Hence the system converges to the benchmark system in two time steps if $T_j(\Delta(0) + I)Kf(x(0)) \neq 0, \forall j$.




(a) The real world scenario (b) The simulation

Figure 7.4: Multi-robot cooperation




(a) Complete information (b) The BM strategy (c) The BA strategy

Figure 7.5: The trajectories under different strategies

If the system is trapped in the set $\{u_j(k) \equiv 0\}$, i.e. $T_j(\Delta(k) + I) \equiv 0$ for some $j$. As $B_j^T P \underline{x}(k) \neq 0$, the pair $\underline{u}_j(k-1) = 0$ and $\underline{x}(k)$ is not optimal with respect to $J_j$. Suppose agent $j$ chooses a random input $\underline{u}_j(k) \neq 0$ at $k$. At $k+1$, the system starts over again with the new initial conditions: $\widehat{R}(k+1) \neq \widehat{R}(k)$, and $\underline{x}(k+1)$. When the perturbation goes to the right direction, $T_j(\Delta(k+1)+I)Kf(\underline{x}(k+1)) \neq 0$. Then the system will converge to the benchmark system in two steps. □

**Remark 7.4.** *The perturbation only works in the Blame-All case, since the convergence is only affected by the initial conditions (in contrast, in the Blame-Me case, the convergence highly depends on the parameter $\theta_i$'s, according to Proposition 7.2). The improved performance in the Blame-All case is due to the introduction of the consensus $\widehat{U}(k)$ among all agents, which decouples learning and control strategies.*

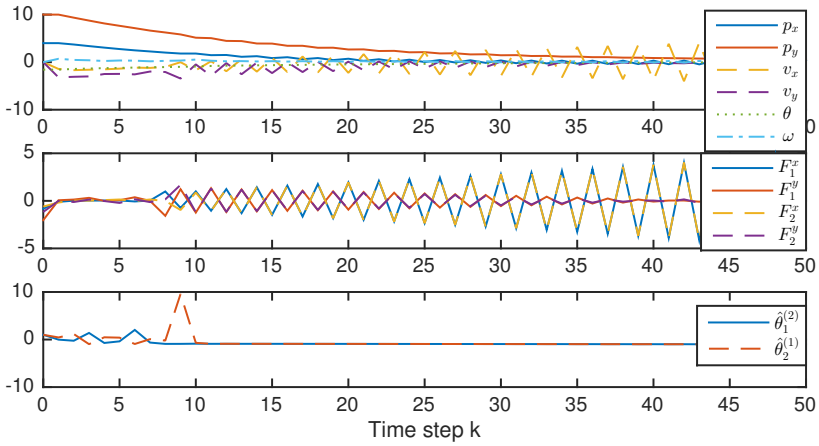



Figure 7.6: The simulation profile under the Blame-Me strategy

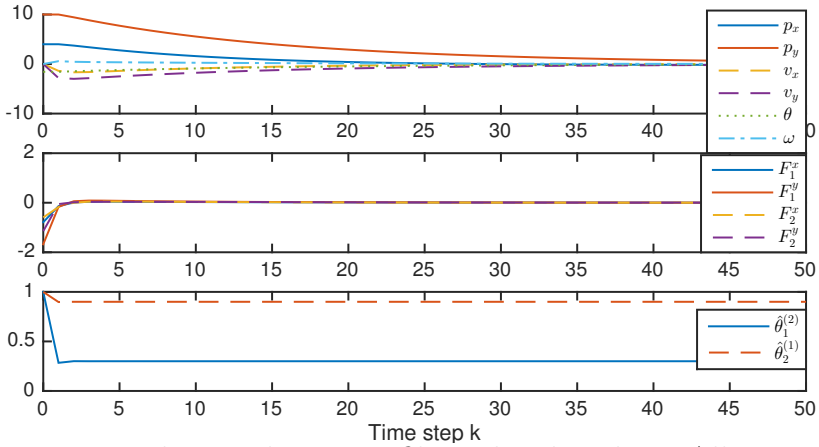



Figure 7.7: The simulation profile under the Blame-All strategy

## 7.4 Example: Robot-Robot Cooperation

Consider the case where two mobile robots cooperate in moving a large object as shown in Fig.7.4a. Suppose there are rigid connections among the robots and the object. Define $x = [p^T, v^T, \alpha, \omega]^T \in \mathbb{R}^6$ where $p$ and $v$ are the position and velocity of the center of mass (which is assumed to be at the middle of the object), $\alpha$ is the orientation of the object and $\omega$ is the angular velocity around the center of mass. The inputs of the agents are the forces, e.g. $u_i = F_i \in \mathbb{R}^2$. Then the system dynamics is in the form of (7.1),

$$x(k+1) = \begin{bmatrix} p(k) + v(k)dt \\ v(k) - \frac{dt}{m} fr1(v(k)) \\ \alpha(k) + \omega(k)dt \\ \omega(k) - \frac{dt}{M} fr2(\omega(k)) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{dt}{m} I_2 & \frac{dt}{m} I_2 \\ 0 & 0 \\ \frac{dt}{M} l_1 & \frac{dt}{M} l_2 \end{bmatrix} U(k), \tag{7.27}$$

where $m$ is the mass, $M$ is the inertia, $dt$ is the sampling time, $fr1(\cdot)$ and $fr2(\cdot)$ are the frictions, and $l_1$ and $l_2$ are the vectors pointing from the center of mass to the robot 1 and

2 respectively. The task for the robots is to move the object from $(4, 10, 0, 0, -\pi/2, 0)$ to $(0, 0, 0, 0, 0, 0)$ as shown in Fig.7.4b. The cost function is given by (7.3).

In the simulation, $dt = m = 0.2$, $M = 0.067$, $|l_1| = |l_2| = 1$, $fr1(v) = 0.02v$ and $fr2(\omega) = 0.0067\omega$. The parameters in the cost function are $\theta_1 = 0.3$, $\theta_2 = 0.9$ and

$$P = diag\left(\begin{bmatrix} 4 & 0 & 0.3 & 0.1 \\ 0 & 4 & 0.1 & 0.3 \\ 0.3 & 0.1 & 1 & 0.1 \\ 0.1 & 0.3 & 0.1 & 1 \end{bmatrix}, \begin{bmatrix} 9 & 0.3 \\ 0.3 & 1 \end{bmatrix}\right).$$

The agents initiate the learning process by setting the estimates to be 1. The simulated trajectories are shown in Fig.7.5 where (a) shows the trajectory under complete information, (b) and (c) show the trajectories under the two strategies when information is asymmetric.

**TE under the Blame-Me Strategy**: When $\alpha \approx 0$, $B_i = h_i(x)$ is almost constant and $\hat{\theta}_1^{(2)} = \hat{\theta}_2^{(1)} = -2$ is a TE by Remark 7.3. The resulting closed loop matrix $I - BH_2^{-1}B^TP$ in (7.19) has an eigenvalue at 10 (the maximum singular value is 10.8), which causes instability as shown in Fig.7.5b. Due to the instability, the target position was not reached. Figure 7.6 shows the simulation profile under this strategy and illustrates how the oscillation starts.

**TE under the Blame-All Strategy**: Figure 7.7 shows the system performance under the Blame-All strategy. The Blame-All strategy outperforms the Blame-Me strategy as the parameter estimation converges in two time steps and the state goes to zero asymptotically, which verifies Theorem 7.5.

## 7.5 Conclusion

This chapter investigated methods to analyze the closed loop performance of the multi-agent system under different agent behaviors. In particular, the Trapped Equilibrium was introduced for system evaluation.

Quadratic simultaneous games were analyzed in the chapter as they rise naturally from multi-robot cooperations. The conclusions are: 1) In a quadratic simultaneous game, the uncertainty introduced by asymmetric information is multiplicative (Proposition 7.1 and Proposition 7.3). 2) The Blame-Me strategy can cause instability in the quadratic simultaneous game (Remark 7.3). 3) Under the proposed Blame-All strategy, the only Trapped Equilibrium assures parameter convergence to true values and system convergence to the benchmark system in the Nash Equilibrium (Theorem 7.5).

This method presented in this chapter can serve as a building block in analyzing interactions with even lesser shared information.

# Part II

# Applications

# Chapter 8

# The Robustly-Safe Automated Driving (ROAD) System

Road safety is one of the major concerns for automated vehicles. In order for these vehicles to interact safely and efficiently with the other road participants, the behavior of the automated vehicles should be carefully designed. In this chapter, we discuss the Robustly-safe Automated Driving system (ROAD) which prevents or minimizes occurrences of collisions of the automated vehicle with other road participants while maintaining efficiency. The design of the ROAD system is a direct application of the methods discussed in previous chapters. In this chapter, a set of design principles will be elaborated, including robust perception and cognition algorithms for environment monitoring and high level decision making and low level control algorithms for safe maneuvering of the automated vehicle.

## 8.1 Overview

Automated driving is widely viewed as a promising technology to revolutionize today's transportation systems [21], so as to free the human drivers, ease the road congestion and lower the fuel consumption among other benefits. Substantial research efforts are directed into this field from research groups and companies [33], which are also encouraged by policy makers [120].

When the automated vehicles drive on public roads, safety is a big concern. While existing technologies can assure high-fidelity sensing, real-time computation and robust control, the challenges lie in the interactions between the automated vehicle and the environment which includes other manually driven vehicles and pedestrians [143]. For road safety, the driving behavior for the automated vehicles should be carefully designed.

Conservative strategies such as "braking when collision is anticipated", known as the Automatic Emergency Braking (AEB) function in existing models [147], are not the best actions in most cases (although they may be necessary in certain cases). Taking into account the dynamics and future course of surrounding vehicles, the automated vehicle has multiple

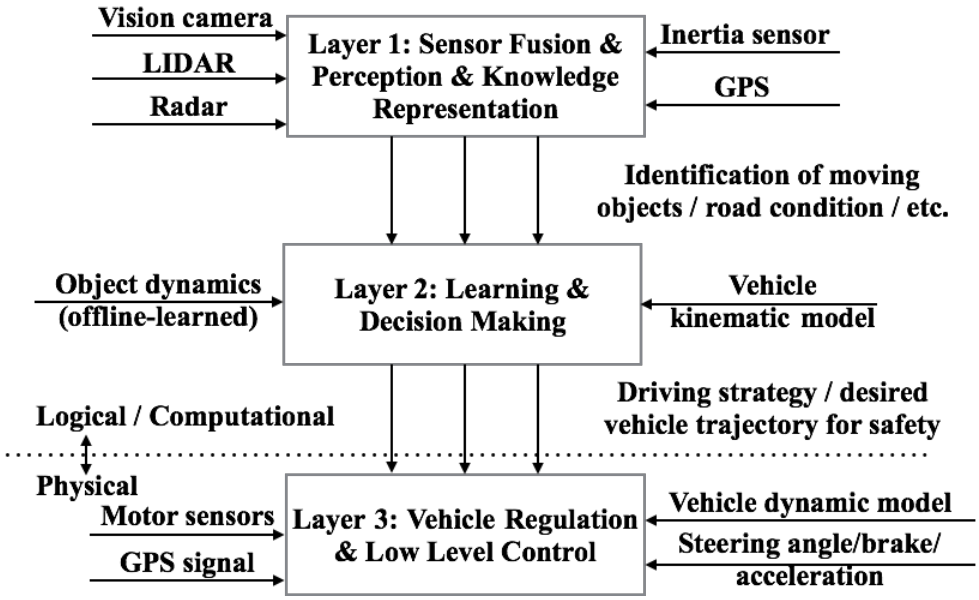



Figure 8.1: Architecture for the robustly-safe automated driving (ROAD) system

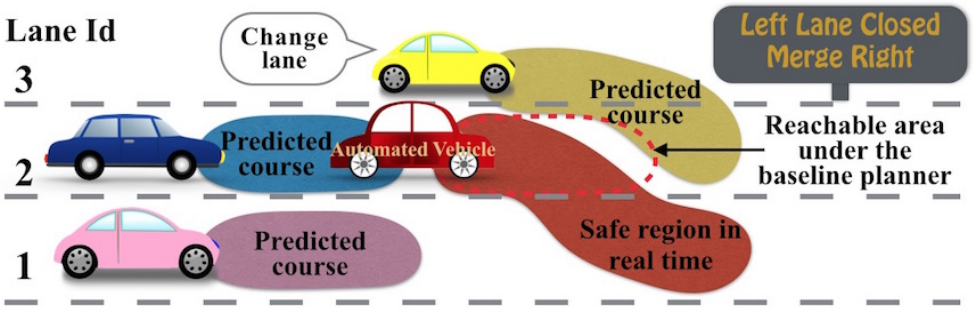
choices for a safe maneuver, i.e. i) slow down to keep a safe headway till the headway reaches the safe limit; ii) steer to the left or right to avoid a collision; iii) even speed up if it can get out a dangerous zone by so doing, etc. The autonomy in driving needs high level machine intelligence [144].

This chapter discusses a framework in designing the driving behavior for automated vehicles to prevent or minimize occurrences of collisions among vehicles, pedestrians and obstacles while maintaining efficiency (e.g. maintaining high speed on freeway). The three-layer Robustly-Safe Automated Driving (ROAD) system is considered, as shown in Fig.8.1, which has a "see-think-do" structure. The expected performance of the ROAD system is illustrated in Fig.8.2, where the automated vehicle predicts the future courses of all surrounding road participants (vehicles or pedestrians) and confines its own trajectory in a safe region in adherence to the predictions for safety.

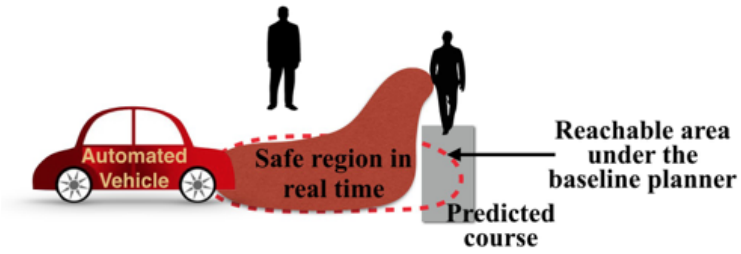
This chapter is based on [80, 89]. The remainder of the chapter is organized as follows: in Section 8.2, a multi-agent traffic model will be introduced and the control problem for automated driving will be formulated. The ROAD system will be discussed in Section 8.3. Simulation studies will be presented in Section 8.4. Section 8.5 concludes the chapter.

## 8.2 The Multi-Agent Traffic Model

The scenario shown in Fig.8.2 is modeled in the framework of multi-agent systems. All vehicles (the automated vehicle and other manually driven vehicles) on freeway are viewed as agents, which have several important characteristics: 1) autonomy: the agents are self-




(a) Interaction with surrounding vehicles.




(b) Interaction with pedestrians.

Figure 8.2: Illustration of the function of the ROAD system.

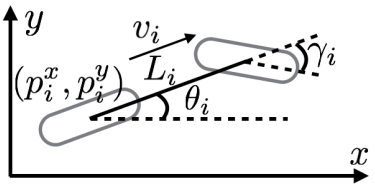



Figure 8.3: The kinematic bicycle model.

aware and autonomous; 2) local views: no agent has a full global view of the system; 3) decentralization: there is no designated controlling agent. From a global view, there can be thousands of agents (vehicles) in the system (e.g. on the freeway). But only the local interactions among the vehicles are of interest. Hence, in the controller of the automated vehicle, only the behavior of the surrounding vehicles will be analyzed. Surrounding vehicles refer to the vehicles that can be detected and within certain distances to the automated vehicle.

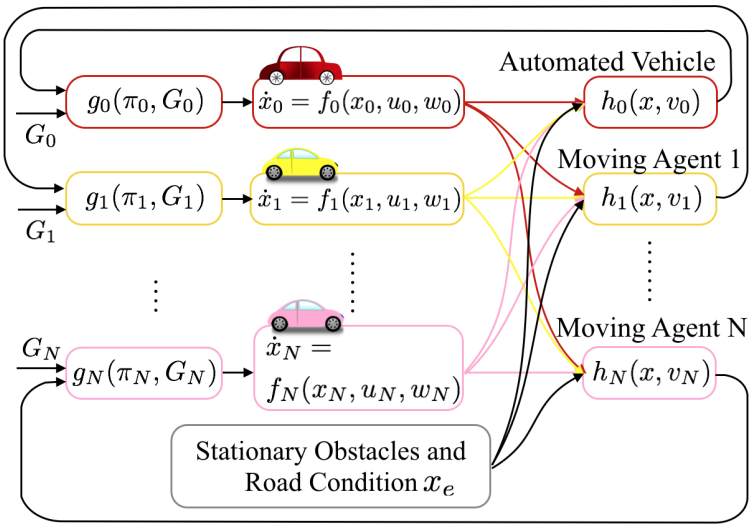



Figure 8.4: The block diagram for local interactions among road participants.

### 8.2.1 The System Model

Suppose there are $N$ surrounding vehicles locally and are indexed from 1 to $N$. Let $H = \{1, .., N\}$ be the set of indices for all surrounding vehicles. The automated vehicle has index 0. Since the surrounding vehicles are changing from time to time, mathematically, the topology of the MAS is time varying [119].

For vehicle $i$, denote its state as $x_i$, control input as $u_i$. According to (2.5), agent $i$'s dynamic equation can be written as

$$\dot{x}_i = f_i(x_i, u_i, w_i), \tag{8.1}$$

where $w_i$ is the disturbance introduced by the environment, e.g. wind. Let $x_e$ be the state of the environment, e.g. speed limit $v_{lim}$, stationary obstacles and so on. Then the system state $x$ is defined as $x = [x_0^T, x_1^T, \cdots, x_N^T, x_e^T]^T$.

Agent $i$ chooses the control $u_i$ based on its information set $\pi_i$ and its objective $G_i$ (which can be intended behaviors or desired speed). In this chapter, it is assumed that there is no direct communication among vehicles. In this way, agent $i$'s information set at time $T$ contains all the measurements up to time $T$, i.e. $\pi_i(T) = \{y_i(t)\}_{t\in[0,T]}$ where $y_i$ follow from (2.3). The controller for agent $i$ can be written as

$$u_i = g_i(\pi_i, G_i). \tag{8.2}$$

Based on (2.3), (8.1) and (8.2), the block diagram for the multi-agent system is shown in Fig.8.4, which is a variation of Fig.2.2. All agents are coupled together in the closed loop system due to measurement feedbacks.

### 8.2.2 The Optimal Control Problem

In the ROAD system, the driving behavior for the automated vehicle should be designed considering the following two factors: efficiency and safety. Equations (8.3) provide the mathematical formulation. The efficiency factor requires that the objective of the automated vehicle (such as lane following in a constant speed or going to a desired position) be achieved in an optimal manner through minimizing a cost function $J(x_0, u_0, G_0)$. As the state $x_0$ is not directly known, the cost should be minimized in the sense of expectation given the measurement $y_0$, e.g. $E(J(x_0, u_0, G_0)|y_0)$ as shown in (8.3a). Meanwhile, the safety factor requires that the efficiency requirement be fulfilled safely as the motion of the automated vehicle should be constrained with respect to other road participants' behaviors. The constraint on the automated vehicle can be written as $x_0 \in R_S(x_1, \cdots, x_n, x_e)$ where $R_S$ is a safe set in the state space of the automated vehicle that depends on the states of other road participants and the state of the environment. As all of the states are not directly known, the constraint should be considered in the stochastic sense such that it is satisfied almost surely with probability $1 - \epsilon$ for $\epsilon \to 0$ given the measurement $y_0$, e.g. $P(\{x_0 \in R_S(x_1, \cdots, x_n, x_e)\}|y_0) \geq 1 - \epsilon$ as shown in (8.3e). Then the following optimal control problem can be formulated

$$\min_{u_0} \; E\left[J(x_0, u_0, G_0)|y_0\right], \tag{8.3a}$$

$$s.t. \; u_0 \in \Omega, E(x_0|y_0) \in \Gamma, \tag{8.3b}$$

$$\dot{x}_0 = f(x_0, u_0, w_0), \tag{8.3c}$$

$$y_0 = h(x_0, x_1, \cdots, x_n, x_e, v_0), \tag{8.3d}$$

$$P\left(\{x_0 \in R_S(x_1, \cdots, x_n, x_e)\}|y_0\right) \geq 1 - \epsilon, \tag{8.3e}$$

where $\Omega$ is the control space constraint for vehicle stability, and $\Gamma$ is the constraint regarding the speed limit and other regulations. Equation (8.3c) and (8.3d) are the dynamic equation and the measurement equation respectively, where $w_0$ and $v_0$ are noise terms.

The ROAD system solves the above problem by 1) estimating the states $x_i$'s using the measurement $y_0$ in layer 1 and 2) estimating the dynamics of all road participates (e.g. dynamics of $x_i$'s) and solving for the optimal control $u_0$ using the estimated states in Layer 2 and 3) tracking the computed optimal control trajectory in layer 3.

## 8.3 The Functions in the ROAD System

In this chapter, we focus on Layer 2. In Layer 2, the control sequence $u_0$ is obtained by solving the optimization problem (8.3). However, the problem is in general hard to solve due to the safety constraint (8.3e), as the dynamics of the states $x_i$'s are unknown and the set $R_S$ is non-convex. To solve the problem efficiently, the behavior design architecture discussed in Chapter 2 will be implemented. The behaviors of other road participants will be identified

and $x_i$'s will be predicted online. A parallel planning structure will be used to solve the non-convex optimization efficiently.

For simplicity, only the kinematic model will be considered in Layer 2. Figure 8.3 shows the bicycle model used for all vehicle $i$, where $(p_i^x, p_i^y)$ is the position of the center of the rear axle, $v_i$ the forward speed, $\theta_i$ the vehicle heading ($\theta_i = 0$ when the vehicle is following the lane), $\gamma_i$ the steer angle and $L_i$ the vehicle wheelbase. Assuming no tire slip angle, the kinematics of vehicle $i$ follow from $\dot{p}_i^x = v_i \cos(\theta_i), \dot{p}_i^y = v_i \sin(\theta_i), \dot{\theta}_i = \frac{v_i}{L_i} \tan\gamma_i$. Since the mapping from $\gamma_i$ to $\dot{\theta}_i$ is homeomorphic given $v_i$, $\dot{\theta}_i$ is chosen as an input signal instead of $\gamma_i$. For vehicle $i$, define $x_i = [p_i^x, p_i^y, v_i, \theta_i]^T$ and $u_i = [\dot{v}_i, \dot{\theta}_i]^T$. Hence (8.1) can be simplified as

$$\dot{x}_i = f(x_i) + \mathbb{B}u_i + \mathbb{B}w_i, \forall i = 0, \cdots, N, \tag{8.4}$$

where $f(x_i) = \begin{bmatrix} v_i \cos\theta_i \\ v_i \sin\theta_i \\ 0 \\ 0 \end{bmatrix}, \mathbb{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = [\mathbb{B}_1, \mathbb{B}_2].$

### 8.3.1 Cognition: Understanding Other Road Participants

Instead of predicting other vehicles' trajectories directly, human drivers may classify other drivers' intended behavior first. If the intended driving behaviors are understood, the future trajectories can be predicted using empirical models. Mimicking what humans would do, the learning structure in Fig.8.5 is designed as an instance of the structure in Fig.2.8 for the automated vehicle to make predictions of the surrounding vehicles, where the process is divided into two steps: 1) the behavior classification, where the observed trajectory of a vehicle goes through an offline trained classifier; and 2) the trajectory prediction, where the future trajectory is predicted based on the identified behavior, by using an empirical model which contains adjustable parameters to accommodate the driver's time-varying behavior.[1] The classification step is needed when the communications among vehicles are limited. Otherwise, vehicles can broadcast their planned behaviors as discussed in [88].

In this section, the design of the classifier, the empirical models and the online learning algorithm will be discussed.

#### The Driving Behaviors

Denote the intended behavior of vehicle $i$ at time step $k$ as $b_i(k)$. In this chapter, five behaviors are considered:

- Behavior 1 ($B_1$): Lane following;

[1]Consider Fig.8.4. When $G_i$ denotes vehicle $i$'s intended behavior, the behavior classification is a backward process to identify $G_i$, while the trajectory prediction is a forward process to predict vehicle $i$'s closed loop behavior $\dot{x}_i = f(x_i) + \mathbb{B}g_i(\pi_i, G_i)$ based on identified $G_i$.

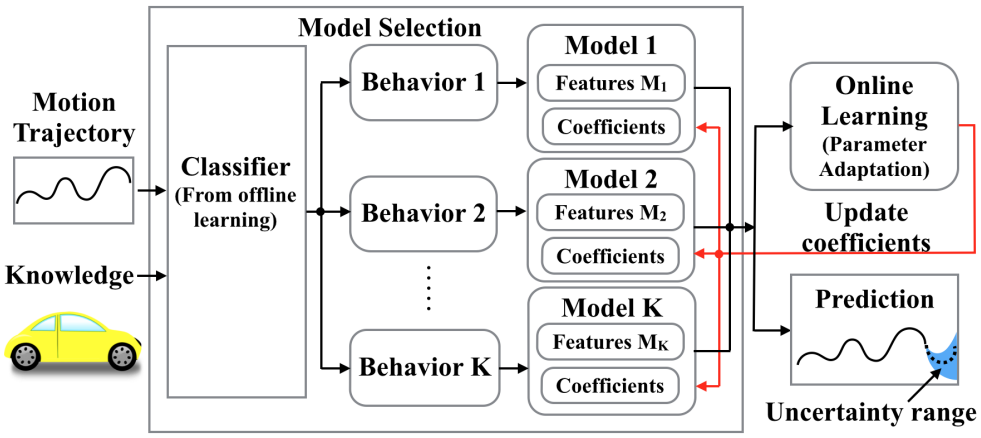



Figure 8.5: The structure of the learning and prediction center.

- Behavior 2 ($B_2$): Lane changing to the left;
- Behavior 3 ($B_3$): Lane changing to the right;
- Behavior 4 ($B_4$): Lane merging;
- Behavior 5 ($B_5$): Lane exiting;

where $B_1$ is the steady state behavior; $B_2$, $B_3$ and $B_4$ are driving maneuvers; and $B_5$ is the exiting behavior. It is assumed that there must be gaps (lane following) between two maneuvers ($B_2$, $B_3$ and $B_4$). The transitions among behaviors follow from the model shown in Fig.8.6a.

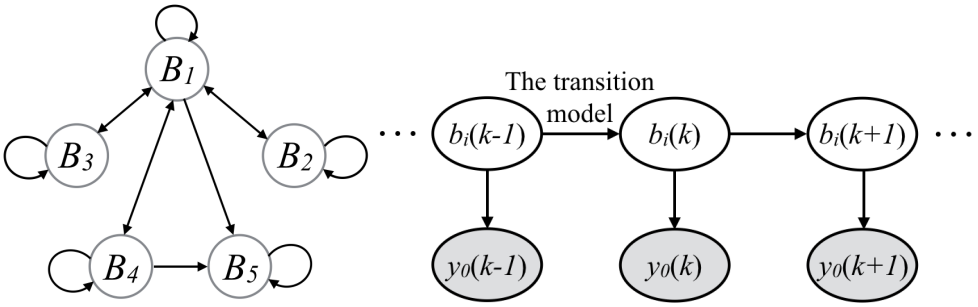
Let $P(b_i(k)|\pi_0(k)) \in \mathbb{R}^5$ be the probability vector that vehicle $i$ intends to conduct $B_1, \cdots, B_5$ at time step $k$ given information up to time step $k$. The relationship between $P(b_i(k)|\pi_0(k))$ and $P(b_i(k+1)|\pi_0(k))$ can be described by a Markov matrix $A = P(b_i(k+1)|b_i(k)) \in \mathbb{R}^{5\times 5}$, e.g.

$$P(b_i(k+1)|\pi_0(k)) = A * P(b_i(k)|\pi_0(k)), \tag{8.5}$$

where $A$ represents the transition model[2] in Fig.8.6a.

The transition model should be invoked in calculation only when vehicle $i$ is following the lane and is about to conduct a maneuver. When vehicle $i$ is conducting a maneuver, there is no need to calculate the probability distribution over other behaviors. The transition model can be used again when the maneuver is completed or aborted and vehicle $i$ starts to follow the lane. The intuition is that: although the intention of a driver is unknown (thus needs to be inferred), but his action is observable (thus when he turns his intention into action, there is no need to guess).

[2] $A$ can be trained using real world labeled data $\underline{b}_i(k)$, e.g. $A_{pq} = P(b_i(k+1) = B_p | b_i(k) = B_q) := \sum_{i,k} I(\underline{b}_i(k+1) = B_p, \underline{b}_i(k) = B_q) / \sum_{i,k} I(\underline{b}_i(k) = B_q)$ where $I$ is the indicator function.




(a) The transition model (b) The HMM model

Figure 8.6: The behavior transition model and the hidden Markov model.

### The Classifier and the Features

The intentions of a driver at different time steps form a Markov process, which, however, are unknown to the automated vehicle. Hence the behavior classification problem becomes an inference problem in the Hidden Markov Model (HMM) [23] as shown in Fig.8.6b. According to Bayes' rule, at time step $k$, for $j = 1, 2, \cdots, 5$,

$$\begin{aligned} P(b_i(k) = B_j | \pi_0(k)) &\propto P(b_i(k) = B_j, y_0(0), \cdots, y_0(k)), \\ &\propto P(y_0(k) | b_i(k) = B_j) P(b_i(k) = B_j | \pi_0(k-1)), \end{aligned}$$

where $P(b_i(k)|\pi_0(k-1))$ encodes the temporal transitions of the intended behaviors and can be obtained using (8.5). $P(y_0(k)|b_i(k))$ is the measurement model, which can be constructed from data offline[3]. To represent the high-dimension data efficiently, the measurement $y_0$ is divided into several features for each surrounding vehicle $i$ (not limited to the following ones):

- Feature 1: Longitudinal acceleration $f_i^1 = \dot{v}_i$.
- Feature 2: Deceleration light $f_i^2 = (1, 0) = (\text{on}, \text{off})$.
- Feature 3: Turn signal $f_i^3 = (1, 0, -1) = (\text{left,off,right})$.
- Feature 4: Speed relative to the traffic flow $f_i^4 = v_i - \bar{v}$.
- Feature 5: Speed relative to the front vehicle $f_i^5 = v_i - v_{front}$.[4]
- Feature 6: Current lane Id $f_i^6$. $f_i^6 = -1$ if the vehicle is occupying two lanes.
- Feature 7: Current lane clearance $f_i^7 = (1, 0, -1) = (\text{blocked}, \text{clear}, \text{ended})$.

[3]Similar to the transition model $A$, the measurement model can also be obtained by supervised training, together with necessary curve fitting.

[4]Feature 4 and 5 encodes the interactions among vehicles.

initialization $P(b_i(0)) = [1, 0, 0, 0, 0]^T$, $k = 0$;
**while** *Classifier is Active* **do**
  $k = k + 1$;
  read current $y_0(k)$, calculate $f_i^1(k), \cdots, f_i^{10}(k)$;
  **if** $f_i^6(k) == -1$ **then**
    $B^* = \arg\max_{B=B_3,B_3,B_4} P(b_i(k-1) = B)$;
    $P(b_i(k) = B^*) = 1$, $P(b_i(k) \neq B^*) = 0$;
    **while** $f_i^6(k) == -1$ **do**
      $k = k + 1$, read $y_0(k)$, calculate $f_i^6(k)$;
      $P(b_i(k)) = P(b_i(k-1))$;
    **end**
    $P(b_i(k)) = [1, 0, 0, 0, 0]^T$;
  **else**
    $P(b_i(k)|\pi_0(k-1)) = A * P(b_i(k-1)|\pi_0(k-1))$;
    calculate $M = P(y_0(k)|b_i(k))$;
    $P(b_i(k)|\pi_0(k)) = M. * P(b_i(k)|\pi_0(k-1))$;
    normalize $P(b_i(k)|\pi_0(k))$;
  **end**
**end**

**Algorithm 8.1:** Behavior classification for vehicle $i$

- Feature 8: Lateral velocity, e.g. $f_i^8 = v_i \sin\theta_i$.
- Feature 9: Lateral deviation from the center of its current lane $f_i^9$.
- Feature 10: Lateral deviation from the center of its target lane (if in the lane changing mode) $f_i^{10}$.

Algorithm 8.1 is designed based on the previous discussion. The probability distributions over all possible behaviors are calculated at each time step when the vehicle is following the lane (not occupying two lanes). The update of the distribution stops when the vehicle is conducting a maneuver. After the maneuver is completed, the probabilities will be initialized. Figure 3.1 illustrates the behavior classification results under Algorithm 8.1, where the measurement model is set empirically as $P(y_0|b_i = B_1) \propto exp(-(f_i^8)^2 - c(f_i^9)^2)$, $P(y_0|b_i = B_3) \propto 1 - exp(-(f_i^8)^2 - c(f_i^9)^2)$ and $c$ is a constant. In the beginning, the vehicle is following a lane. Then the probability of lane changing rises since the lateral speed $f_i^6$ goes up. When the vehicle crosses the boundary of two lanes, the probability of $B_3$ goes to 1. After lane changing, the vehicle starts to follow the new lane.

**The Empirical Models for Trajectory Prediction**

The future trajectory is predicted according to the most likely predicted behavior, i.e. $\arg\max_{B_j} P(b_i(k) = B_j)$. For lane following $B_1$, $\theta_i \approx 0$ and the lateral deviation of the vehicle can be ignored. The vehicle $i$ only regulates its longitudinal speed to match the speed of the traffic flow and the speed of its front vehicle, e.g.

$$\dot{x}_i = f(x_i) + \mathbb{B}_1[k_1 f_i^4 + k_2 f_i^5], \tag{8.6}$$

where $k_1, k_2 \in \mathbb{R}$ are online-adjustable parameters, which can be identified online using the method discussed in Chapter 3.3. For lane changing $B_2$ and $B_3$, the vehicle not only regulates the longitudinal speed, but also regulates the lateral position, hence the turning rate. The empirical model can be described as

$$\dot{x}_i = f(x_i) + \mathbb{B}_1[k_1 f_i^4 + k_2 f_i^5] + \mathbb{B}_2[k_3 f_i^8 + k_4 f_i^{10}], \tag{8.7}$$

where $k_1, k_2, k_3, k_4 \in \mathbb{R}$ are online-adjustable parameters.

### 8.3.2 Online Motion Planning and Control

Based on the predictions of the surrounding vehicles, the automated vehicle needs to find a safe and efficient trajectory satisfying the optimal control problem (8.3). The decision making architecture in Fig.8.7 is considered as an variation of the parallel architecture in Fig.2.3. The baseline planner solves the problem in a long time horizon without the safety constraint (8.3e), and the safety planner takes care of the safety constraint in real time [77].

**The Baseline Planner** The baseline planner solves the optimal control problem (8.3) to ensure efficiency, which is similar to the planner in use when the automated vehicle is navigating in an open environment. When the cost function and the control constraint are designed to be convex, (8.3) become a convex optimization problem.

Suppose the objective $G_0$ (target behavior and target speed $v_r$) is specified. The baseline planner tries to plan a trajectory to accomplish $G_0$. When the objective is to follow the lane,

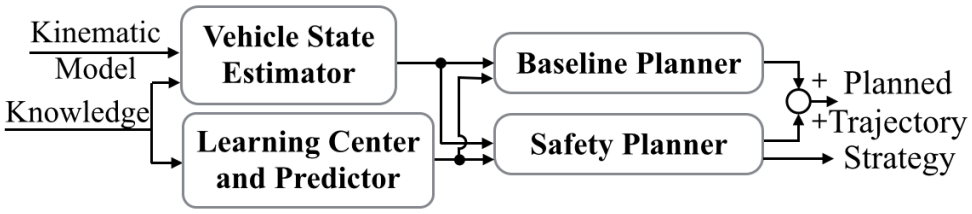



Figure 8.7: The structure of the decision making center.

the cost function is designed as

$$J = \int_0^\infty \left[(v_0 - v_r)^2 + q(f_0^9)^2 + u_0^T R u_0\right] dt, \tag{8.8}$$

where $q \in \mathbb{R}^+$ and $R \in \mathbb{R}^{2\times 2}$ is positive definite. When the objective is to change lane, the automated vehicle should change the lane smoothly within time $T$. The cost function is designed as

$$J = \int_0^T \Phi(t)dt + \Gamma(f_0^{10}(T), \theta_0(T)), \tag{8.9}$$

where $\Phi = (v_0 - v_r)^2 + q_1(f_0^{10})^2 + q_2(\theta_0)^2 + u_0^T R u_0$ is the cost to go; $\Gamma = s_1(f_0^{10}(T))^2 + s_2(\theta_0(T))^2$ is the terminal cost; $q_1, q_2, s_1, s_2 \in \mathbb{R}^+$; and $R \in \mathbb{R}^{2\times 2}$ is positive definite.

The computation in the baseline planner can be done offline. The resulting control policy will be stored for online application, to ensure real-time planning.

**The Safety Planner** The safety planner modifies the trajectory planned by the baseline planner locally to ensure that it will lie in the safe set $X_S$ in real time as discussed in Chapter 4.

During lane following, the safety constraint requires the automated vehicle to keep a safe headway. Thus $X_S(B_1) = \{x : d_1(x) \geq d_{min}\}$ where $d_1(x)$ calculates the minimum distance between the automated vehicles and the vehicle or obstacles in front of it. During lane changing, the safety constraint requires the automated vehicle to keep a safe distance from vehicles on both lanes. Thus $X_S(B_2), X_S(B_3) = \{x : d_2(x) \geq d_{min}\}$ where $d_2(x)$ calculates the minimum distance between the automated vehicle and all surrounding vehicles and obstacles in the two lanes.

Mathematically, the safe set is described using a safety index $\phi$, which is a real-valued continuously differentiable function on the system's state space. The state $x$ is considered safe only if $\phi(x) \leq 0$ as shown in Fig.4.4. In Fig.8.2, the safe region for the automated vehicle is affected by the future trajectory of the surrounding vehicles. Based on the prediction of other vehicles, if the baseline trajectory leads to $\phi \geq 0$ now or in the near future, the safety planner will generate a modification signal to decrease the safety index by making $\dot{\phi} < 0$.

The safety index is chosen as $\phi = D - d_j^2(x) - \alpha \dot{d}_j(x)$ $(j = 1, 2)$ where $D > d_{min}^2$, and $\alpha > 0$ are constants. To ensure safety, the control input of the automated vehicle must be chosen from the set of safe control $U_S(t) = \{u_0(t) : \dot{\phi} \leq -\eta_0 \; when \; \phi \geq 0\}$ where $\eta_0 \in \mathbb{R}^+$ is a safety margin. By (8.4), the set of safe control when $\phi \geq 0$ can be written as

$$U_S(t) = \{u_0(t) : L(t)\, u_0(t) \leq S(t)\}, \tag{8.10}$$

where $L(t) = \frac{\partial \phi}{\partial x_0}\mathbb{B}$, $S(t) = -\eta_0 - \sum_{j\in H} \frac{\partial \phi}{\partial x_j}\dot{x}_j - \frac{\partial \phi}{\partial x_0} f$ and $\dot{x}_j$ is the prediction made by the trajectory predictor.

If the baseline control input $u_0(t)$ is anticipated to violate the safety constraint, the safety controller will map it to the set of safe control $U_S(t)$ according to the following quadratic cost function

$$u_0/U_S = \min_{u \in U_S \cap \Omega} J_0(u) = \frac{1}{2}(u-u_0)^T W (u-u_0), \tag{8.11}$$

where $W$ is a positive definite matrix and defines a metric in the vehicle's control space. To obtain optimality, $W$ should be close enough to the metric imposed by the cost function $J$ in (8.8) and (8.9), e.g. $W \approx d^2J/du_0^2$ where $J$ is convex in $u_0$. In the lane following mode, if the lateral deviation $f_0^9$ is large due to obstacle avoidance and the safety controller continues to generate turning signal $\theta \neq 0$, the vehicle will enter the lane changing mode.

The constraints on the vehicle input $U_S$ in different scenarios are illustrated in Fig.8.8 to Fig.8.9. For simplicity, a constraint on planar acceleration is used to illustrate the safety constraint $U_S$ as the constraint on planar acceleration can be transformed to the constraint on throttle and wheel angles[5].

Figure 8.8 shows the safety constraint with respect to a front vehicle. The red dot represents zero acceleration and the circle represents the boundary of maximum acceleration in any direction (which may also in be other shapes, e.g. a ellipsoid). The shaded area represents $U_S$. When the front vehicle is relatively static with respect to the automated vehicle, $U_S$ depends only on the relative distance. When the headway is far enough, all directions of acceleration are safe. When the headway is too short, only decelerations are safe. When there is a wall, turning against the wall is not safe. When the front vehicle has relative motions with respect to the automated vehicle, $U_S$ depends on the relative distance as well as relative movement. Figure 8.9 illustrate the safety constraint when there is a vehicle in the adjacent lane. In mixed traffic, the constraint $U_S$ is the intersection of the constraint computed with respect to each surrounding vehicle.

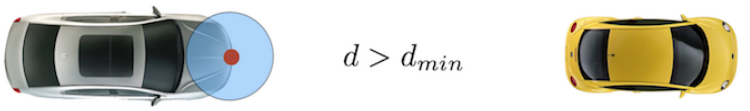
## 8.4 Performance

Case studies are performed to illustrate the performance of the ROAD system during on-road driving (e.g. freeway driving) and unstructured driving (e.g. driving in parking lots). The simulations are done using parameters from a Lincoln MKZ shown in Fig.8.10.

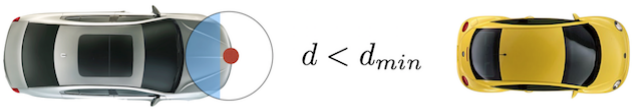
### 8.4.1 On-Road Autonomous Driving

In freeway driving, there are typically two kinds of objectives : lane following with desired speed (in any lane) and lane changing (to a target lane). The baseline planners for the two objectives are obtained offline by computing optimal control policies for problem (1-4). The safety planner checks online whether the planned trajectory is safe to execute with respect

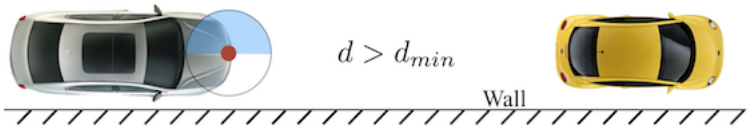
[5]Note that in a kinematic model when the turning angle is small and there is no tire slip, the longitudinal acceleration is proportional to the throttle angle and the lateral acceleration is proportional to the wheel angle.



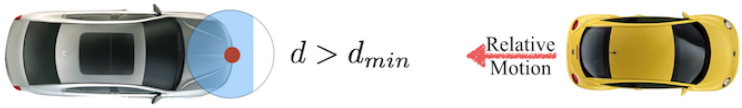

(a) $U_S$ when $d > d_{min}$ and $\dot{d} = 0$.



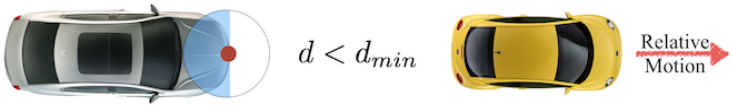

(b) $U_S$ when $d < d_{min}$ and $\dot{d} = 0$.




(c) $U_S$ when there is a wall.




(d) $U_S$ when $d > d_{min}$ and $\dot{d} < 0$.



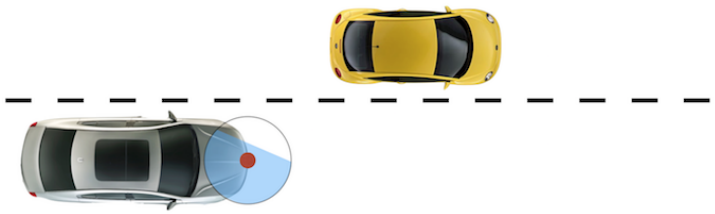

(e) $U_S$ when $d < d_{min}$ and $\dot{d} > 0$.

Figure 8.8: The safety constraint $U_S$ with respect to a front vehicle.

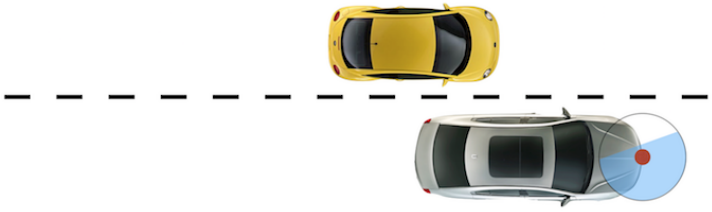
to the predicted motions of the surrounding vehicles. Three behaviors are considered for surrounding vehicles: lane following, lane change to the left, and lane change to the right. A Hidden-Markov-Model-based classifier is trained offline using labelled trajectories of human-controlled road participants in the simulator. The intended behavior of each surrounding vehicle is predicted online using the classifier. And the future motion of a surrounding vehicle is predicted using an empirical model associated with the classified behavior.

### Lane Following

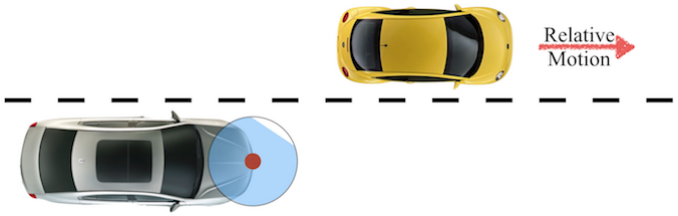
**Case 1 - Stationary Obstacle**. Figure 8.11 shows the case when the automated vehicle suddenly noticed a stationary obstacle 40m ahead. The safety controller went active. By mapping the baseline input $u_0$ to $U_S$ in (8.11), the command for deceleration and turn was generated. Then the automated vehicle slowed down and changed lane to the left to avoid



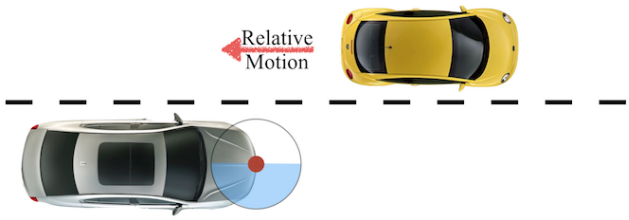
(a) $U_S$ when the vehicle is in the front.



(b) $U_S$ when the vehicle is behind.




(c) $U_S$ when the vehicle is moving away.




(d) $U_S$ when the vehicle is moving closer.

Figure 8.9: The safety constraint $U_S$ with respect to a vehicle in the adjacent lane.

the obstacle. After lane changing, the vehicle accelerated to the desired speed again.

**Case 2 - Slow Front Vehicle**. Figure 8.12 shows the case when the front vehicle was too slow. To illustrate the interaction, the trajectories of both vehicles are down sampled and shown in the last plot in Fig.8.12, where circles represent the automated vehicle and squares represent the slow vehicle. Different colors correspond to different time steps, the lighter the earlier. At the beginning, since it was not possible for the automated vehicle to keep the desired speed behind the slow car, it started to change lane to the left. After changing the lane, it overtook the slow vehicle.

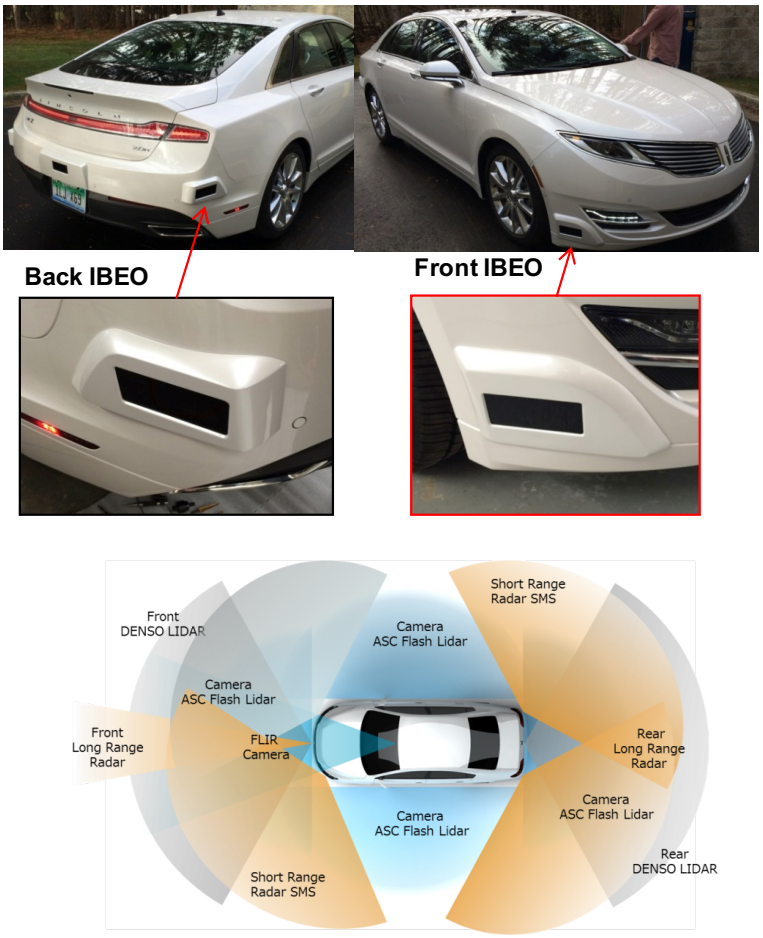



Figure 8.10: The test vehicle: Lincoln MKZ.

**Case 3 - Fast Cut-in Vehicle**. Figure 8.13 shows the scenario when the automated vehicle was overtook by a fast vehicle. When the automated vehicle observed large lateral velocity from the fast vehicle, it predicted that the fast vehicle would change lane. Under the command from the safety controller, the automated vehicle slowed down. After the lane changing of the fast vehicle, the automated vehicle accelerated again to meet the desired speed and keep a safe headway to the fast vehicle.

### Lane Change

In this simulation, the trajectory of the surrounding vehicle is manually controlled, in order to test the real time interactions.

**Case 1 - A Vehicle Moving Side by Side in the Target Lane**. Figure 8.14 shows the case when the vehicle in the target lane was traveling next to the automated vehicle with approximately same speed. It was not safe to change lane in this case. Then the

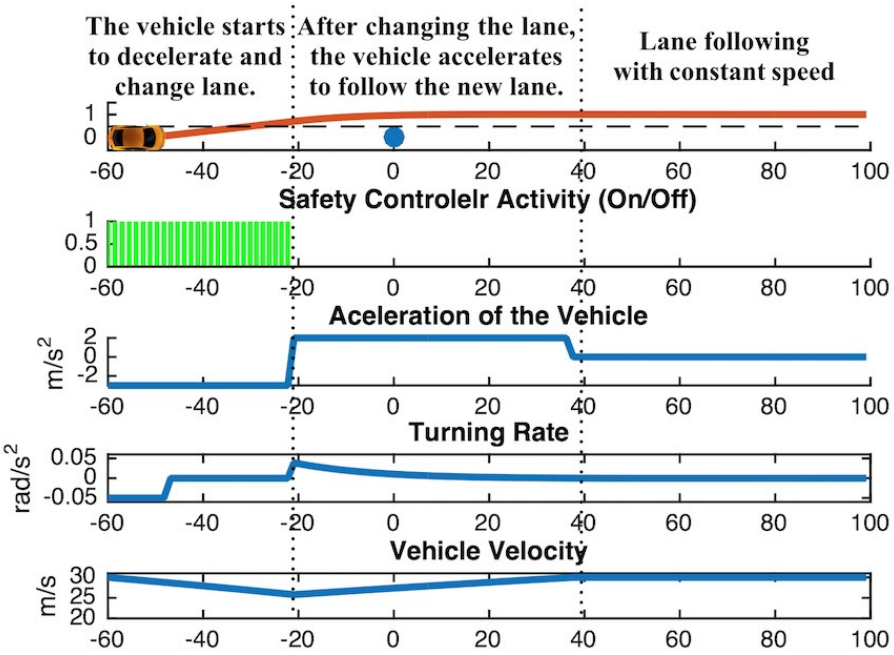



Figure 8.11: Case 1 in lane following: stationary obstacle.

automated vehicle slowed down to create a gap between the two vehicles. When the distance between the two vehicles was big enough, the automated vehicle then changed to the target lane. After adjusting the relative distance to the front vehicle, the automated vehicle then followed the new lane at constant speed.

**Case 2 - A Slowing Down Vehicle in the Target Lane**. Figure 8.15 shows the case when the vehicle in the target lane was slowing down. At first, the automated vehicle tried to use the strategy in case 1. However, when it noticed that the yellow car also slowed down, it then sped up to overtake the yellow car.

**Case 3 - Simultaneous Lane Change from Opposite Directions**. Figure 8.16 shows the case when another vehicle changed to the target lane simultaneously with the automated vehicle, but from the opposite direction. At the beginning, the yellow car was anticipated to follow its lane. Hence it was safe for the automated vehicle to change lane. When the lateral velocity of the yellow car became larger, the probability of $B_3$ went up and a possible future collision was anticipated. Then the safety planner went active. When the yellow car was about to cross the lane boundary, the automated vehicle turned back to its previous lane and slowed down. The automated vehicle finally changed lane using the strategy in case 1: slowing down first and changing lane when the distance between the two vehicles was big enough.

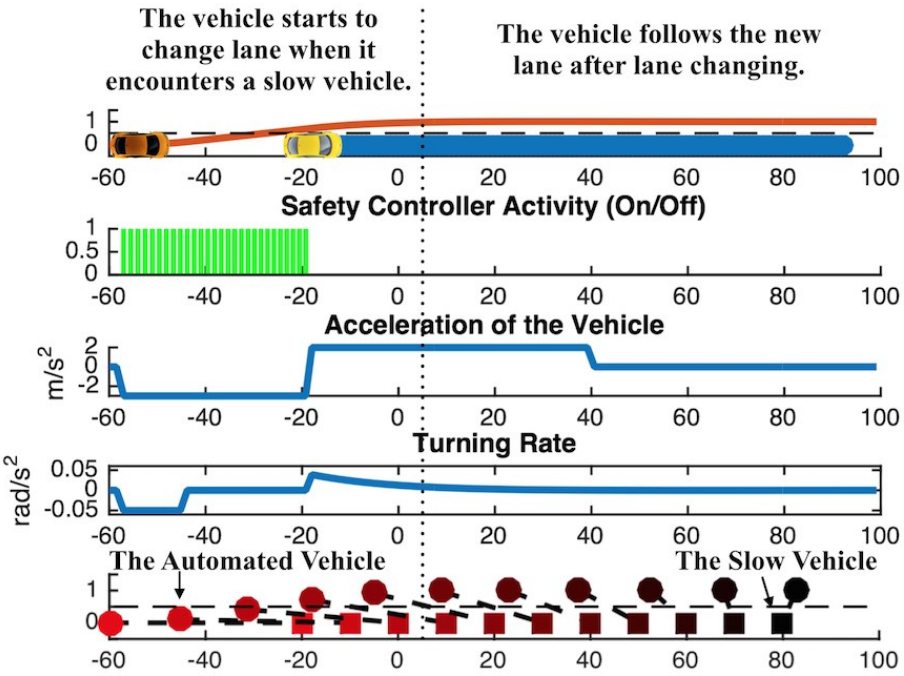



Figure 8.12: Case 2 in lane following: slow front vehicle.

### Mixed Traffic

Figure 8.17 illustrates the active safety measures when traffic is heavy on a curved freeway. The objective $G_0$ is lane following with a desired speed $37m/s$ which is higher than the current traffic speed. To fulfill $G_0$, the vehicle performed several lane changing safely with the help of the safety planner. The distance and velocity profile during the simulation is shown in Fig.8.17b. The dark bar indicates the moment when the safety planner was active, which matches with the moment when the smallest distance to the surrounding vehicles reached a threshold (shown by the dotted line). The smallest distance is only computed for the surrounding vehicles that have the possibility to collide with the automated vehicle, e.g. in the moving direction of the automated vehicle. For example, if there is only one surrounding vehicle which is in the adjacent lane to the automated vehicle and both vehicles are going to follow the lanes, then the smallest distance is infinity as their moving directions are parallel to each other. When the safety planner was on, changes on the vehicle velocity and direction were generated by the safety planner as discussed earlier. The smallest distance was always kept over 4m. The video of this study can be found in [75].

Figure 8.18 shows the active safety performance of ROAD during lane following. There are heavy traffic on the left lanes. As the current speed of the automated vehicle is below the desired speed, the baseline planner generates a acceleration command. The safety constraints with respect to vehicles 1 to 3 are computed and shown as $U_S^1$, $U_S^2$ and $U_S^3$ in the figure and $U_S := \cap_i U_S^i$. As the acceleration command generated by the baseline planner is not

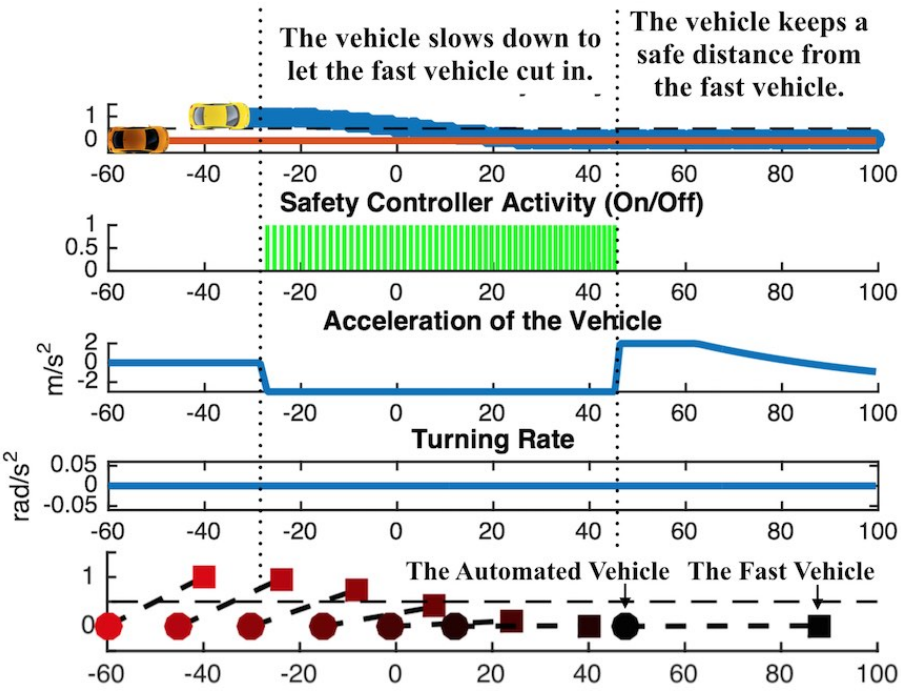



Figure 8.13: Case 3 in lane following: fast cut-in vehicle.

safe, it is modified by the safety planner. The modification signal tries to minimize the length difference between the reference acceleration vector and the modified vector as well as the angular difference between the two vectors. Then a turning command is generated in the modified signal. Under this new command, the vehicle changed to its right lane. The metric for the modification is a design parameter which allows the vehicle to generate diverse behaviors. For example, if the metric penalizes the norm difference between the reference acceleration vector and the modified acceleration vector, there will be no turning command in the modified signal as shown in Fig.8.18c. The automated vehicle will remain in the current lane and adapt to track the velocity of the front vehicle. This behavior is very similar to the one generated by adaptive cruise control (ACC).

Figure 8.19 illustrates the active safety during lane change on freeway. When the vehicle started to change lane, it is not safe to do so. The safety planner cancelled the turning command and increased the longitudinal acceleration. Then the automated vehicle overtook the vehicle in the target lane and finished lane changing when it was safe to do so. The behavior of the automated vehicle depends significantly on the prediction of the behavior of the vehicle in the target lane. If the vehicle is predicted to be moving relatively forward, it is possible that a decelerating modified signal be generated as shown in Fig.8.19c, in which case the automated vehicle will change lane to follow the vehicle in the target lane when a safe "gap" is created by decelerating.

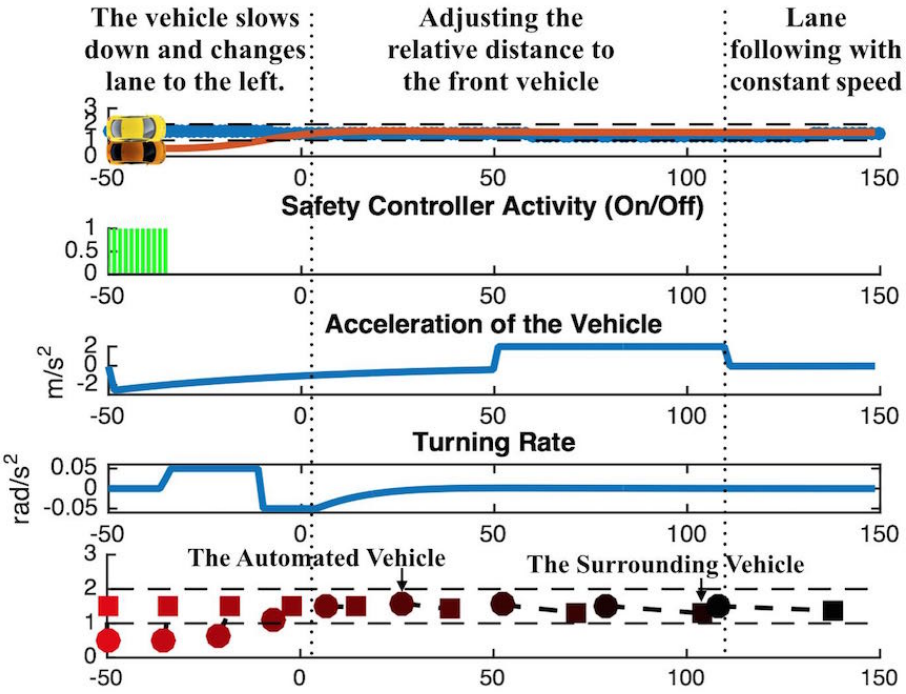



Figure 8.14: Case 1 in lane change: a parallel vehicle in the target lane.

### 8.4.2 Driving in Unstructured Environments

Driving in parking lots is a typical unstructured driving scenario. The automated vehicle needs to interact with pedestrians safely. The objective $G_0$ for the vehicle is to navigate to a desired parking space. Similar to the freeway driving case, the baseline planner is obtained offline. The safety planner checks online whether the planned trajectory is safe to execute with respect to the prediction of the pedestrian motions. However, unlike the freeway driving case, not all directions of acceleration are feasible due to the nonholonomic nature of automobiles. For example, when the vehicle speed is high, it is possible for the vehicle to have acceleration in many directions. When the vehicle speed is low and gear shift is not allowed, the vehicle can only generate a small range of acceleration by steering and pressing the pedal as shown in Fig.8.20. When modifying the trajectories in the safety planner, the nonholonomic constraint also needs to be considered.

***Active Safety for Interacting with Pedestrians*** Figure 8.21a illustrates the active safety for driving in a parking lot. The pedestrian in the simulation environment is controlled by a human subject in real time to test the response of the automated vehicle. The desired parking space is shown by the white lines. In the simulation, the vehicle tried to go to the parking space while the pedestrian moved crossing the path of the vehicle. So the vehicle slowed down to wait for the pedestrian and accelerated only after the pedestrian passed by. The safety constraint for the automated vehicle in this scenario is shown in Fig.8.20c. As the

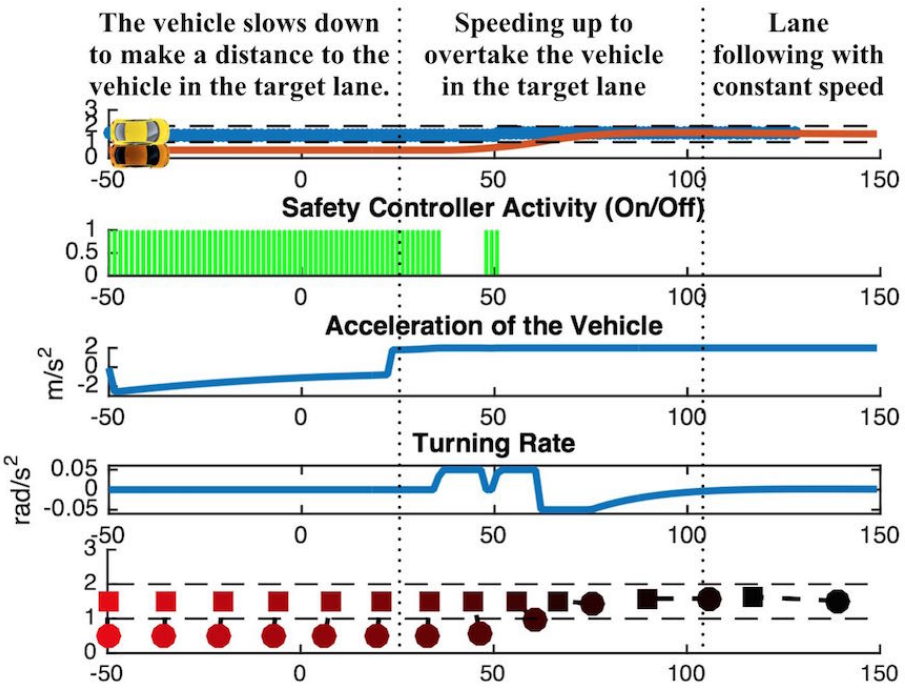



Figure 8.15: Case 2 in lane change: a slowing down vehicle in the target lane.

only action that satisfied both the safety constraint and the nonholonomic constraint was to stay still, the vehicle chose to wait for the human to pass by. The distance and velocity profile is shown in Fig.8.21b. The safety planner was on when the distance between the vehicle and the human was small. The small velocity generated at the beginning was due to the uncertainty in the human motion prediction. The smallest distance was always kept over 4m.

## 8.5 Discussion and Conclusion

In this chapter, the design of Layer 2 in the ROAD system was discussed. The multi-agent traffic model was proposed and an optimal control problem was formulated for vehicle trajectory planning. To solve the problem, the behaviors of surrounding vehicles was identified and their future trajectories was predicted. Based on the predictions, the optimal control problem was solved online using an unique architecture: a parallel combination of a baseline planner which solved the problem without the safety constraint and a safety planner which took care of the safety constraint online. The proposed algorithms were verified in the simulations.

The function of the ROAD system can be divided into two parts: reasoning of other road participants' behaviors and planning the trajectory for the ego vehicle. The first part relies

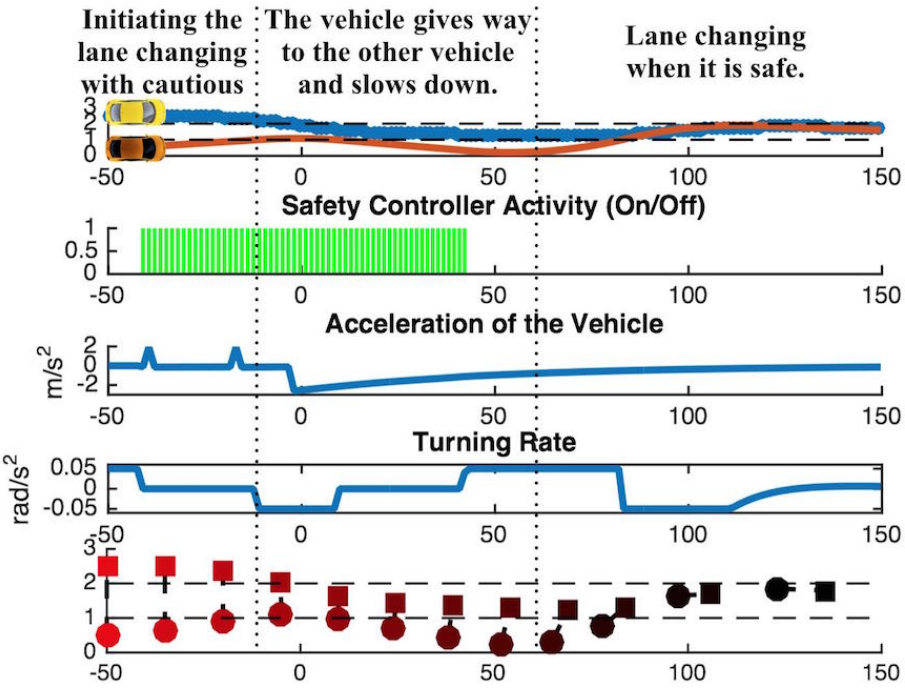
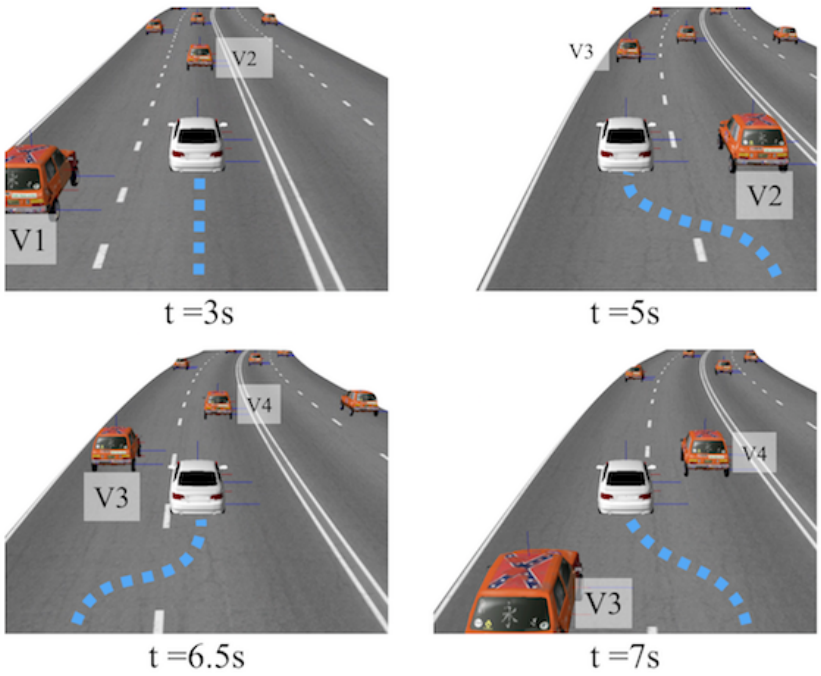


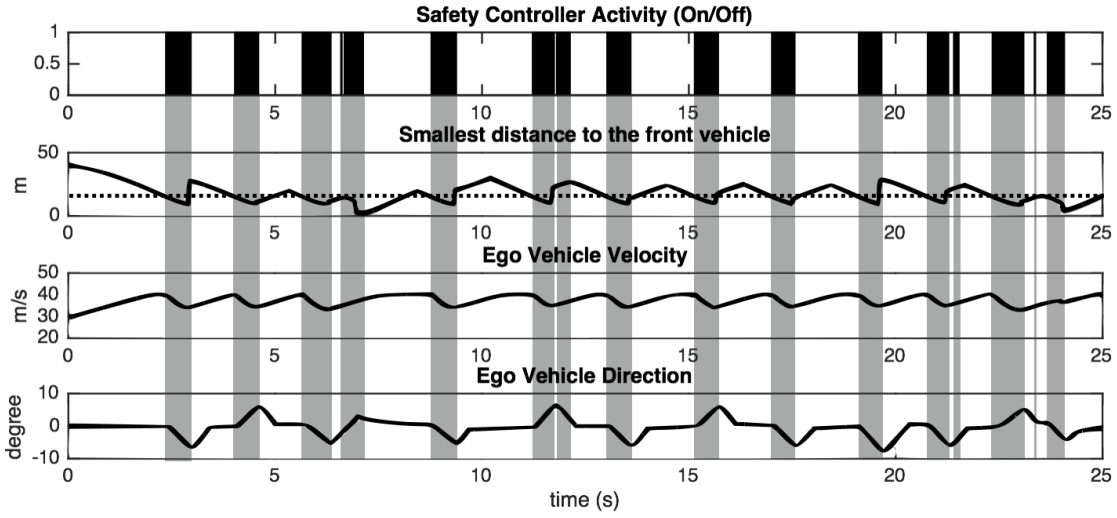

Figure 8.16: Case 3 in lane change: simultaneous lane change from opposite directions.

on offline data collection and learning. The purpose of offline learning is to let the ego vehicle make reasonable predictions of the environment so that it can behave less conservatively. Even if the environment is new and the behaviors of the road participants are never encountered before, the system can still generate safe trajectories using only the online learning method and the parallel planners as discussed in [13]. In the beginning of the interaction, the safety planner can behave defensively by assuming the worst case scenario. During the interaction, the vehicle can fit a reactive behavior model for each road participant using the observed trajectory of that road participant, and refine the model by online adaptation, similar to the method discussed in the case study "Driving in Unstructured Environments". The safety planner then monitors the trajectory generated by the baseline planner given the predictions made by the reactive behavior models, while a larger minimum distance requirement will be chosen if the confidence level of the model is lower. The confidence level of the model can be tracked by comparing the predicted and the observed behaviors of the corresponding road participant. Although the proposed method is mainly to address active safety for automated vehicles, it can also be applied to driving assistive systems for manually driven vehicles, if we replace the baseline planner by a function that gets human's driving command directly as shown in Fig.8.22. Then the safety planner can monitor human's driving commands based on the predicted motions of other road participants.

In the future, the following parts will be improved. An online long-term trajectory planner discussed in Chapter 5 will replace the current offline baseline planner. The long-

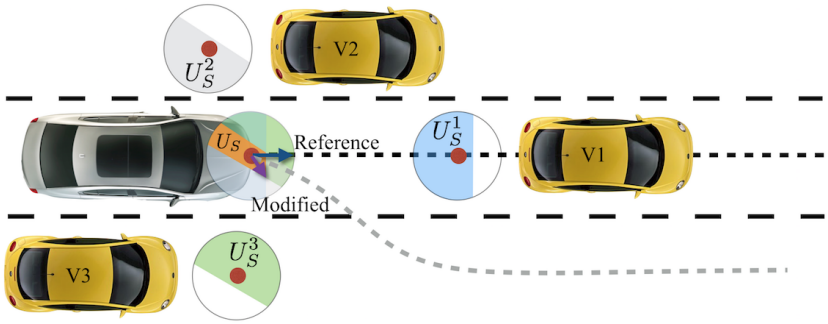



(a) The snapshots of the simulation.

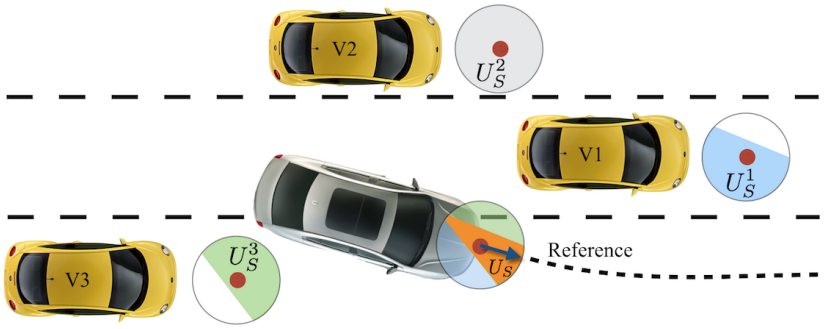



(b) The distance and velocity profile for mixed traffic simulation.

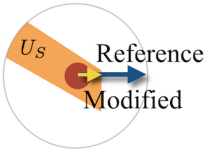
Figure 8.17: Performance of the ROAD system in heavy traffic.




(a)



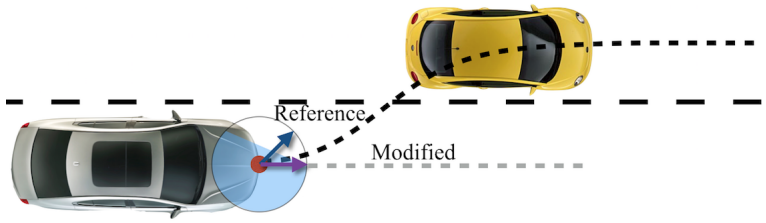

(b)



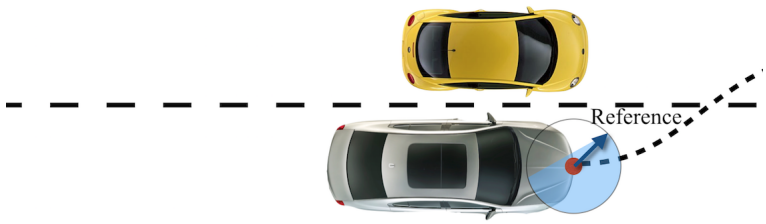

(c) Different safety maneuver during lane following due to different modification metric.

Figure 8.18: The safety constraints and maneuvers for lane following in mixed traffic.

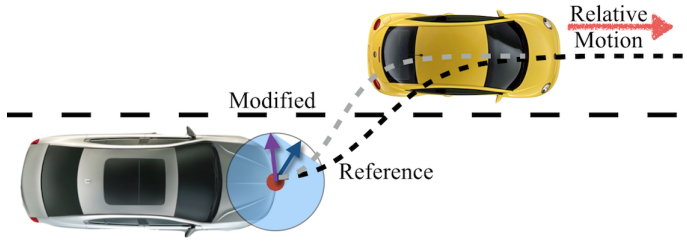
term planner will consider all related issues in driving, including safety, efficiency, comfort and economy, with a relatively long preview horizon (e.g. 10s) and a relatively low sampling frequency (e.g. 4Hz). The safety controller described in this chapter will be running in high frequency (e.g. 20Hz) and be prepared to interrupt the long-term planner in emergencies to guarantee safety. Moreover, to account for diverse behaviors of other road participants, the trajectory predictor will be extended to be stochastic and multimodal.




(a)




(b)

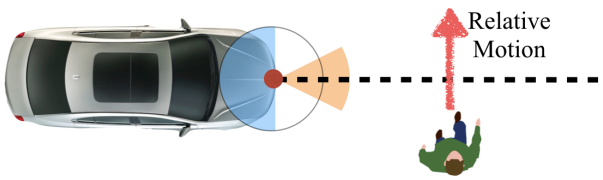



(c) Different Safety Behavior during Lane Change due to Different Prediction

Figure 8.19: The safety constraints and maneuvers for lane change in mixed traffic.

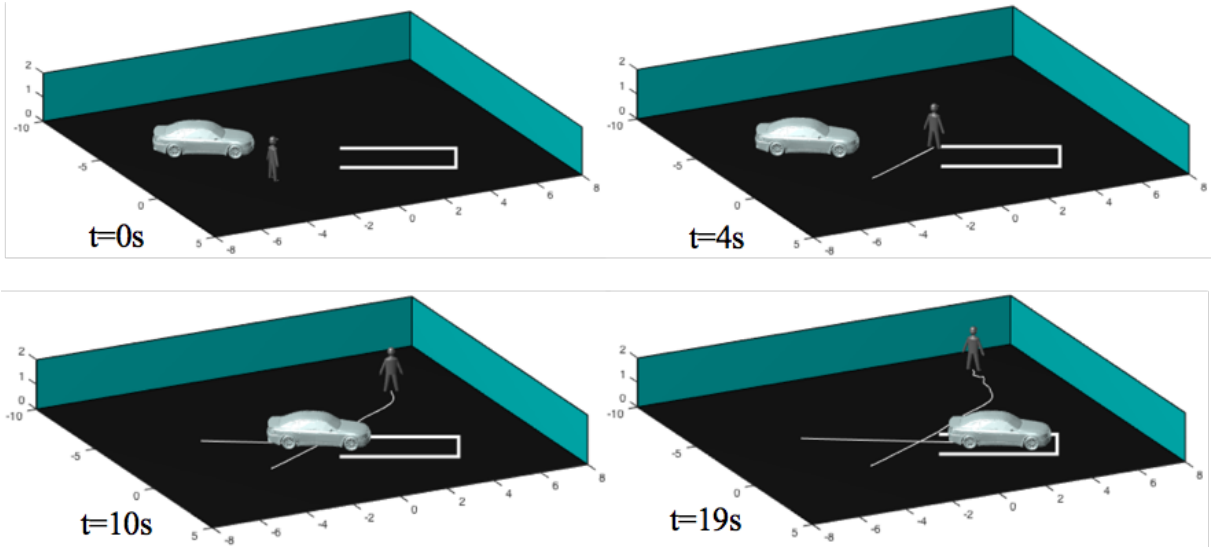
(a) The dynamic constraint for low speed

(b) The dynamic constraint for high speed

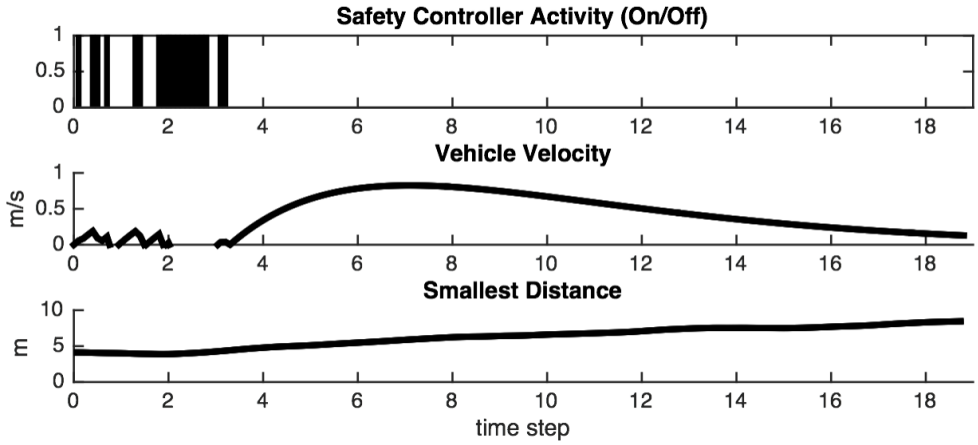



(c) The safety constraint and the dynamic constraint.

Figure 8.20: Considering the dynamic constraint together with the safety constraint.




(a) The snapshots of the simulation.




(b) The distance and velocity profile.

Figure 8.21: Application of the ROAD system for driving in a parking lot

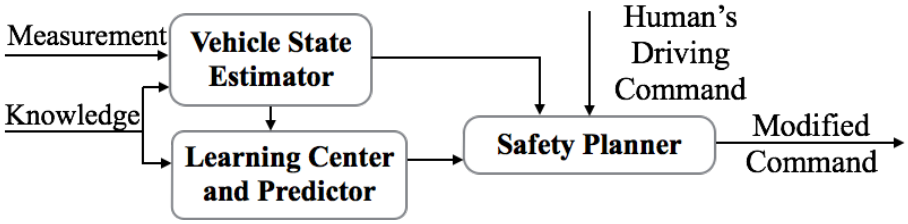



Figure 8.22: A driver assistive system using the ROAD system.

# Chapter 9

# The Robot Safe Interaction System (RSIS)

Human workers and robots are two major workforces in modern factories. For safety reasons, they are separated, which limits the productive potentials of both parties. It is promising if we can combine human's flexibility and robot's productivity in manufacturing. This chapter describes the application of the methods discussed in previous chapters on industrial robots.

## 9.1 Overview

In modern factories, human workers and robots are two major workforces. For safety concerns, the two are normally separated with robots confined in metal cages, which limits the productivity as well as the flexibility of production lines. In recent years, attention has been directed to remove the cages so that human workers and robots may collaborate to create a human-robot co-existing factory [24]. Those robots working in a human-involved environment are called co-robots.

The potential benefits of co-robots are huge and extensive, e.g. they may be placed in human-robot teams in flexible production lines [64] as shown in Fig.9.1, where robot arms and human workers cooperate in handling workpieces, and automated guided vehicles (AGV) co-inhabit with human workers to facilitate factory logistics [141]. Automotive manufacturers Volkswagen and BMW [149] have took the lead to introduce human-robot cooperation in final assembly lines in 2013.

In the factories of the future, more and more interactions among humans and industrial robots are anticipated to take place. In such environments, safety is one of the biggest concerns [132], which attracts attention from standardization bodies [52], as well as from major robot manufacturers including Kuka, Fanuc, Nachi, Yaskawa, Adept and ABB [5]. Several safe cooperative robots or co-robots has been released, such as UR5 from Universal Robots (Denmark) [142] which is implemented by Volkswagen and BMW, Baxter from Rethink

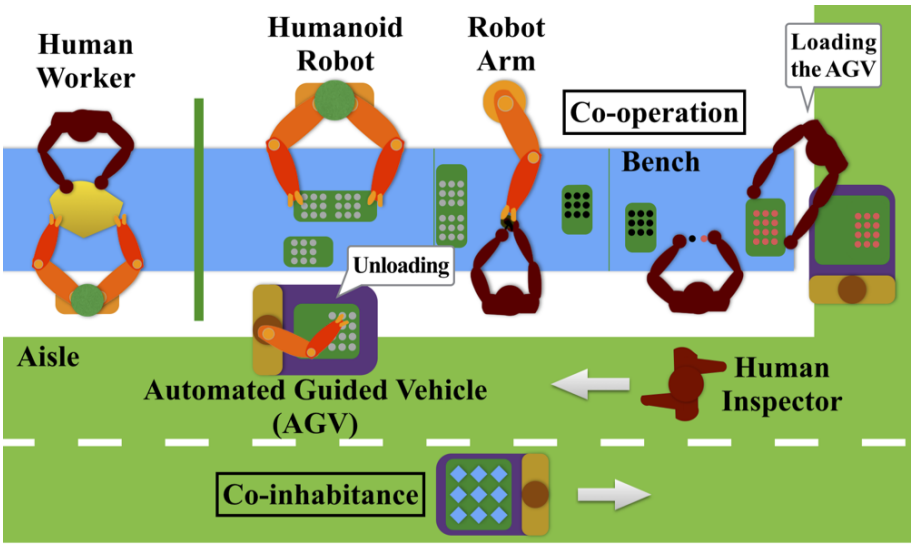



Figure 9.1: Human-robot collaboration and co-inhabitance in future production lines.

Robotics (US) [12], NextAge from Kawada (Japan) [59] and WorkerBot from Pi4_Robotics GmbH (Germany) [19]. However, most of these researches and products focus on intrinsic safety, i.e. safety in mechanical design [55], actuation [156] and low level motion control [95]. Safety during social interactions with humans, which are key to intelligence (including perception, cognition and high level motion planning and control), still needs to be explored.

On the other hand, several successful implementations of non-industrial co-robots have been reported, e.g. home assist robots [151] and nursing robots [111]. Complex software architectures are developed to equip the robots with various cognition, learning and motion planning abilities. However, those robots are mostly of human-size or smaller size with slow motion, which may not be cost-efficient for industrial applications. To fully realize a human-robot co-existing factory, the software design methodology for fast co-robots, especially those that are large in size, with multiple links and complicated dynamics, needs to be explored.

In order to make the industrial co-robots human-friendly, they should be equipped with the abilities [51] to: (1) collect environmental data and interpret such data, (2) adapt to different tasks and different environments, and (3) tailor itself to the human workers' needs. The first ability is a perception problem, while the second and third are control problems that are of interest in this chapter.

The challenges for control are (i) coping with complex and time-varying human motion, and (ii) assurance of real time safety without sacrificing efficiency. An constrained optimal control problem is formulated to describe this problem mathematically. And a modularized controller architecture will be discussed as an variation of the architecture discussed in Chapter 2.2. The modularized architecture 1) treats the efficiency goal and the safety goal separately and allows more freedom in designing robot behaviors, 2) is compatible with existing robot motion control algorithms and can deal with complicated robot dynamics, 3)

guarantees real time safety, and 4) are good for parallel computation.

The remainder of the chapter is organized as follows: in Section 9.2, the constrained optimization problem will be described; in Section 9.3, the controller architecture in solving the optimization problem will be proposed, together with the design considerations of each module. Case studies with robot arms are performed in Section 9.4. Section 9.5 concludes the chapter.

## 9.2 Algorithmic Safety Measures: The Optimization Problem

As shown in Fig.9.1, co-robots can co-operate as well as co-inhabit with human workers. In this chapter, safety in co-inhabitance and contactless co-operation will be addressed as they form basic interaction types during human-robot interactions. Since the interaction is contactless, robots and humans are independent to one another in the sense that the humans' inputs will not affects the robots' dynamics in the open loop. However, humans and robots are coupled together in the closed loop, since they will react to others' motions.

### 9.2.1 Problem Formulation

Denote the state of the robot of interest as $x_R \in \mathbb{R}^n$ and the robot's control input as $u_R \in \mathbb{R}^m$ where $n, m \in \mathbb{N}$ . Assume the robot dynamics is affine[1] , i.e.

$$\dot{x}_R = f(x_R) + h(x_R)u_R. \tag{9.1}$$

The task or the goal for the robot is denoted as $G_R$, which can be 1) a settle point in the Cartesian space (e.g. a workpiece the robot needs to get), 2) a settle point in the configuration space (e.g. a posture), 3) a path in the Cartesian space or 4) a trajectory in the configuration space.

The robot should fulfill the aforementioned tasks safely. Let $x_H$ be the state of humans and other moving robots in the system, which are indexed as $H = \{1, 2, \cdots, N\}$. Then the system state is $x = [x_R^T, x_H^T]^T$. Denote the collision free state space as $X_S$, e.g. $X_S = \{x : d(x_H, x_R) > 0\}$ where $d$ measures the minimum distance among the robot, the humans and all other moving robots. Given the human configuration, the constraint on the robot's state space $R_S(x_H)$ is a projection of $X_S$, e.g. $R_S(x_H) = \{x_R : [x_R^T, x_H^T]^T \in X_S\}$, which is time varying with $x_H$. Hence, two steps are needed to safely control the robot motion:

[1]Any system can have an affine form through dynamic extension. Suppose $\dot{x}_R = F(x_R, u_R)$. Define $x_R^e = [x_R^T, u_R^T]^T$. Let the new control input be $u_R^e = \dot{u}_R$. Then the new system

$$\dot{x}_R^e = \begin{bmatrix} F(x_R^e) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_R^e,$$

is affine.

1) predicting the human motion; and 2) finding the safe region for the robot based on the prediction.

### 9.2.2 The Optimization Problem

**The requirement of the co-robot** is to finish the tasks $G_R$ efficiently while staying in the safe region $R_S(x_H)$, which leads to the following optimization problem [34]:

$$\min_{u_R} \; J(x_R, u_R, G_R), \tag{9.2a}$$

$$s.t.\; u_R \in \Omega, x_R \in \Gamma, \dot{x}_R = f(x_R) + h(x_R)u_R, \tag{9.2b}$$

$$x_R \in R_S(x_H), \tag{9.2c}$$

where $J$ is a goal related cost function to ensure efficiency, $\Omega$ is the constraint on control inputs, $\Gamma$ is the state space constraint (e.g. joint limits, stationary obstacles). The problem is hard to solve since the safety constraint $R_S(x_H)$ is nonlinear, non-convex and time varying with unknown dynamics.

There are numerical methods in solving non-convex optimizations, e.g. sequential convex optimization [123], A* search [127] and Monte-Carlo based rapidly-exploring random trees (RRT) method [66]. However, the computation loads are too high for online applications on industrial co-robots. On the other hand, analytical methods such as potential field methods [108] and sliding mode methods [45] have low computation loads. But they generally do not emphasize optimality. Moreover, the motion patterns of human subjects (or other intelligent robots) are much more complicated than those of general obstacles due to interactions, e.g. $x_H$ may be a function of $x_R$. In Chapter 4, a safe set algorithm (SSA) was discussed to identify the dependency of $x_H$ on $x_R$ online and regulate the control input of the robot in a supervisory loop so as for the system state to stay in the safe set $X_S$. A safe exploration algorithm (SEA) was built upon SSA to reflect the uncertainties in the prediction of $x_H$ in robot motion control. These two methods will be generalized and a modularized controller architecture that can handle 3D interactions will be proposed in the next section.

## 9.3 Algorithmic Safety Measures: The Controller Architecture

### 9.3.1 The Controller Architecture

The proposed controller will be designed as a parallel combination of a baseline controller and a safety controller as shown in Fig.9.2. The baseline controller solves (9.2a-9.2b), which is time-invariant and can be solved offline. The safety controller enforces the time varying safety constraint (9.2c), which computes whether the baseline control signal is safe to execute or not (in the "Safety Constraint" and the "Criteria" module) based on the predictions made in the human motion predictor, and what the modification signal should be (in the "Control

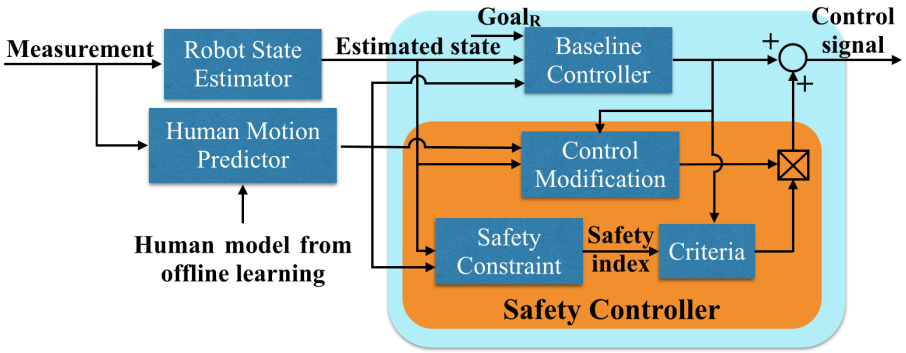



Figure 9.2: The controller architecture.

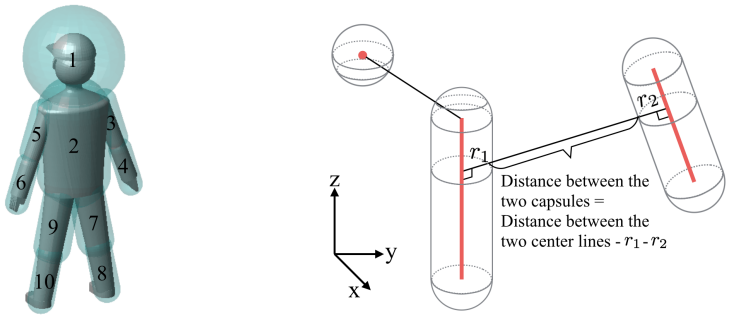
Modification" module). Each module will be elaborated below. The expected outcome of this controller structure is shown in Fig.4.3 on an AGV. In that scenario, the baseline controller will command the AGV to go straight towards its goal. However, the human motion predictor predicts that the human will go to the blue dot and he will be very likely to show up in the gray area. Since the baseline trajectory is no longer in the safe region, the safety controller generates a modified trajectory towards the goal and avoids the human.

### 9.3.2 The Baseline Controller

The baseline controller solves (9.2a-9.2b), which is similar to the controller in use when the robot is working in the cage. The cost function is usually designed to be quadratic which penalizes the error to the goal and the magnitude of the control input, e.g. when $G_R$ is a trajectory, $J = \int_0^T [(x_R - G_R)^T P (x_R - G_R) + u_R^T R u_R] dt$ where $P$ and $R$ are positive definite matrices. The control policy can be obtained by solving the problem offline. The collision avoidance algorithms discussed in Chapter 5 can be used to avoid stationary obstacles described by the constraint $x_R \in \Gamma$. This controller is included to ensure that the robot can still perform the tasks properly when the safety constraint $R_S(x_H)$ is satisfied.

### 9.3.3 The Human Model and the Human Motion Predictor

In different applications, human body should be represented at various levels of details. For AGVs, mobile robots and planar arms, since the interactions with humans happen in 2D, a human can be tracked as a rigid body in the 2D plane with the state $x_H$ being the position and velocity of the center of mass and the rotation around it. For robot arms that interact with humans in 3D, the choice of the human model depends on his distance to the robot. When the robot arm and the human are far apart, the human should also be treated as one rigid body to simplify the computation. In the close proximity, however, the human's limb movements should be considered. As shown in Fig.9.3a, the human is modeled as a




(a) The human model (b) The distance between capsules

Figure 9.3: The human model and the capsules.

connection of ten rigid parts: part 1 is the head; part 2 is the trunk; part 3, 4, 5 and 6 are upper limbs; and part 7, 8, 9 and 10 are lower limbs. The joint positions can be tracked using 3D sensors [125]. The human's state $x_H$ can be described by a combination of the states of all rigid parts.

The prediction of future human motion $x_H$ needs to be done in two steps: inference of the human's goal $G_H$ and prediction of the trajectory to the goal. Once the goal is identified (using the method of model selection discussed in Chapter 3), a linearized reaction model can be assumed for trajectory prediction [154], e.g.

$$\dot{x}_H = Ax_H + B_1 G_H + B_2 x_R + w_H, \tag{9.3}$$

where $w_H$ is the noise, $A, B_1$ and $B_2$ are unknown matrix parameters which encode the dependence of future human motion on his current posture, his goal and the robot motion. Those parameters can be identified using parameter identification algorithms discussed in Chapter 3.3, while the prediction can be made using the identified parameters. Note that to account for human's time varying behaviors, the parameters should be identified online. This method is based on the assumption that human does not 'change' very fast. Moreover, to reduce the number of unknown parameters, key features that affect human motion can be identified through offline analysis of human behavior. Those low dimension features $\{f_i\}$ can be used in the model (9.3) to replace the high dimension states $x_H$ and $x_R$, e.g. $\dot{x}_H = \sum_i a_i f_i + B_1 G_H + w_H$.

### 9.3.4 The Safety Controller

**The Safety Index** The safe set $X_S$ is a collision free subspace in the system's state space, which depends on the relative distance among humans and robots. Since humans and robots have complicated geometric features, simple geometric representations are needed for efficient online distance calculation. Ellipsoids [139] were used previously. However, it's hard to obtain the distance between two ellipsoids analytically. To reduce the computation load,

capsules (or spherocylinders) [97], which consists of a cylinder body and two hemisphere ends, are introduced to bound the geometric figures as shown in Fig.9.3a, Fig.9.4b and in Fig.9.6b. A sphere is considered as a generalized capsule with the length of the cylinder being zero. The distance between two capsules can be calculated analytically, which equals to the distance between their center lines minus their radiuses as shown in Fig.9.3b. In the case of a sphere, the center line reduces to a point. In this way, the relative distance among complicated geometric objects can be calculated just using several skeletons and points. The skeleton representation is also ideal for tracking the human motion.

Given the capsules, the design of $X_S$ is mainly the design of the required minimum distances among the capsules. The design should not be too conservative, while larger buffer volumes are needed to bound critical body parts such as the head and the trunk, as shown in Fig.9.3a. The safe set in the 3D interactions can be designed as:

$$X_S = \{x : \frac{d(p_{ij}, x_R)}{d_{ij,min}} > 1, \forall i = 1, \cdots, 10, \forall j \in H\}, \tag{9.4}$$

where $d(p_{ij}, x_R)$ measures the minimum distance from the capsule of body part $i$ on the human (or the robot) $j$ to the capsules of the robot $R$. $d_{ij,min} \in \mathbb{R}^+$ is the designed minimum safe distance. $d_{1j,min}$ should be large since the head is most vulnerable.

To describe the safe set $X_S$ efficiently, a safety index $\phi$ is introduced, which is a Lyapunov-like function over the system's state space as illustrated in Fig.4.4 that satisfies the three conditions discussed in Chapter 4.2. The safety index for the safe set in (9.4) is designed as:

$$\phi = 1 + \gamma - (d^*)^c - k_1 \dot{d^*} - \cdots - k_{l-1}(d^*)^{(l-1)}, \tag{9.5}$$

where $d^* = \frac{d(p_{i^*j^*}, x_R)}{d_{i^*j^*,min}}$ and the capsule $i^*$ on the human (or the robot) $j^*$ is the capsule that contains the closest point (the critical point) to the robot $R$ . $l \in \mathbb{N}$ is the relative degree from the function $d(\cdot, x_R)$ to $u_R$ in the Lie derivative sense. In most applications, $l = 2$ since the robot's control input can affect joint acceleration. $c > 1$ is a tunable parameter, while larger $c$ means heavier penalties on small relative distance. $\gamma > 0$ is a safety margin that can uniformly enlarge the capsules in Fig.9.3a. $k_1, \cdots, k_{l-1}$ are tunable parameters that need to satisfy the condition that all roots of $1 + k_1 s + \cdots + k_{l-1} s^{l-1} = 0$ should be on the negative real axis in the complex plane. The higher order terms of $d^*$ are included to make sure that the robot does not approach the boundary of the safe set in a large velocity, so that the state can always be maintained in the safe set even if there are constraints on the robot control input, e.g. $u_R \in \Omega$.

**The Criteria** Given the safety index, the criteria module determines whether or not a modification signal should be added to the baseline controller. There are two kinds of criteria: (I) $\phi(t) \geq 0$, or (II) $\phi(t + \Delta t) \geq 0$. The first criterion defines a reactive safety behavior, i.e. the control signal is modified once the safety constraint is violated. The second criterion defines a forward-looking safety behavior, i.e. the safety controller considers whether the

safety constraint will be violated $\Delta t$ time ahead. The prediction in the second criterion is made upon the estimated human dynamics and the baseline control law. In the case when the prediction of future $x_H$ has a distribution, the modification signal should be added when the probability for criteria (II) to happen is non-trivial, e.g. $P(\{\phi(t+\Delta t) \geq 0\}) \geq \epsilon$ for some $\epsilon \in (0,1)$.

**The Set of Safe Control and the Control Modification** The set of safe control $U_R^S$ is the equivalent safety constraint on the control space, i.e. the set of control that can drive the system state into the safe set as shown in Fig.4.4. According to (4.5-4.8), the set of safe control when $\phi \geq 0$ is

$$U_R^S = \{u_R : \frac{\partial \phi}{\partial x_R} h(x_R) u_R \leq -\eta - \frac{\partial \phi}{\partial x_R} f(x_R) - \frac{\partial \phi}{\partial x_H} \dot{x}_H\} \tag{9.6}$$

where $\eta \in \mathbb{R}^+$ is a margin and $\dot{x}_H$ comes from human motion predictor. When $\dot{x}_H$ has a distribution, let $\Pi$ be the compact set that contains major probability mass of $\dot{x}_H$, e.g. $P(\{\dot{x}_H \in \Pi\}) \geq 1-\epsilon$ for a small $\epsilon$. Then the inequality in (9.6) should hold for all $\dot{x}_H \in \Pi$.

The non-convex state space constraint $R_S(x_H)$ is then transferred to a linear constraint on the control space in (9.6). In this way, the modification signal is the optimal value to be added to the baseline control law such that the final control lies in the set of safe control,

$$\Delta u_R = \quad \arg\min_{u_R^o + u \in U_R^S \cap \Omega \cap U_\Gamma} \quad u^T Q u \tag{9.7}$$

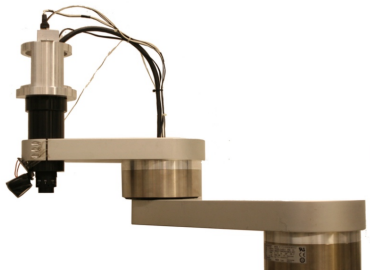
where $Q \in \mathbb{R}^{m \times m}$ is positive definite which determines a metric on the robot's control space. To obtain optimality, $Q$ should be close enough to the metric imposed by the cost function $J$ in (9.2a), e.g. $Q \approx d^2 J / du_R^2$ where $J$ is assumed to be convex in $u_R$. $U_\Gamma$ is the equivalent constraint on the control space of the state space constraint $\Gamma$, which can be constructed following the same procedure of constructing $U_R^S$. Equation (9.7) is a convex optimization problem and is easy to solve. In the case that $U_R^S \cap \Omega \cap U_\Gamma$ is empty, a smaller margin $\eta$ can be chosen so that the feasible control set becomes nonempty.

## 9.4 Case Studies

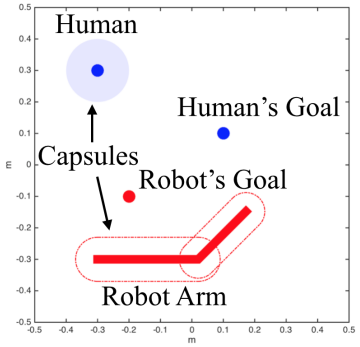
Case studies are performed to evaluate the safety measures on scenarios shown in Fig.9.1. The cases for AGVs and mobile robots are studied in Chapter 4. In this chapter, the interactions among robot arms and humans will be studied. The architecture of the evaluation platform is discussed in Appendix A.3. The result with the evaluation platform in Appendix A.4 can be found in [73].

### 9.4.1 Planar Robot Arm

The planar robot arm is shown in Fig.9.4a. Denote the joint angle as $\theta = [\theta_1, \theta_2]^T$. The dynamic equation of the robot arm is $M(\theta)\ddot{\theta} + N(\theta, \dot{\theta}) = \tau_R$ where $M(\cdot)$ is the generalized



(a) The planar robot arm




(b) The simulation environment

Figure 9.4: The planar robot arm and the simulation environment.

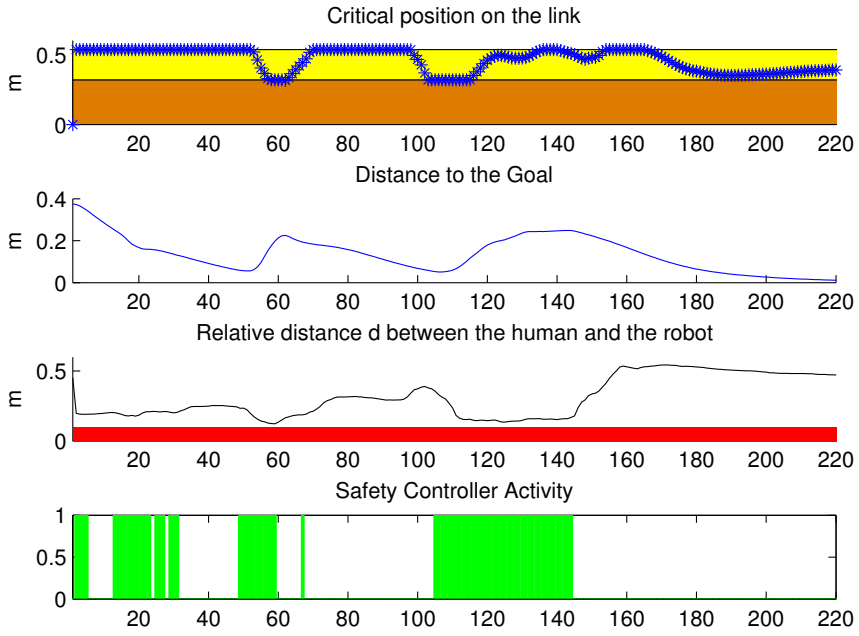
inertia matrix and $N(\cdot,\cdot)$ is the Coriolis and centrifugal forces [25]. Both functions depend on the robot state $x_R = [\theta^T, \dot{\theta}^T]^T$. $u_R = \tau_R$ is the torque input. The state space equation of the planar robot is affine

$$\dot{x}_R = \begin{bmatrix} \dot{\theta} \\ -M^{-1}(\theta)N(\theta,\dot{\theta}) \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}(\theta) \end{bmatrix} u_R. \tag{9.8}$$

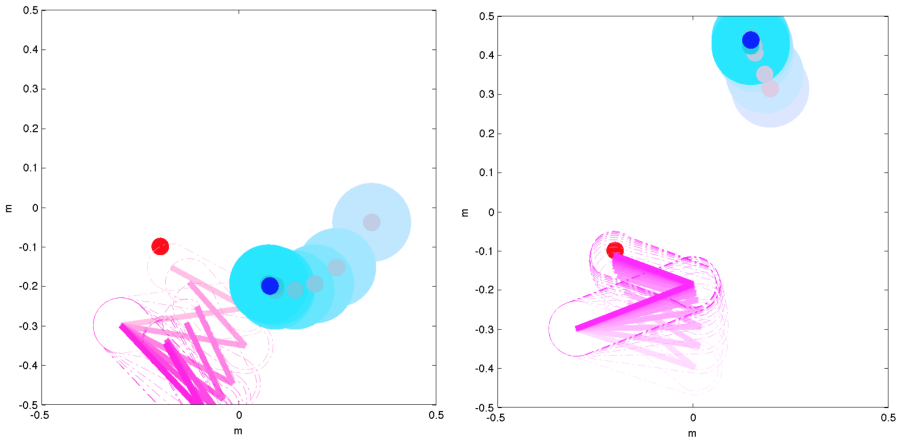
The simulation environment is shown in Fig.9.4b where the robot arm is wrapped in two capsules. The vertical displacement of the robot arm is ignored. The human is shown as a blue circle, which is controlled in real time by a human user through a multi-touch pad. Both the human and the robot need to approach their respective goal points in minimum time. New goals will be generated when the old one is approached.

The baseline controller is designed as a computed torque controller with settle point $G_R$. The safety index is designed as $\phi = D - d^2 - \dot{d}$, where $d$ measures the minimum distance between the human and the robot arm and $D = d_{min}^2(1+\gamma)$. The sampling frequency is $20hz$. Due to the limitation of bandwidth, both reactive and forward-looking criteria are used, in order not to violate the safety constraints between two samples. The set of safe control $U_R^S(k)$ at time $k$ is the intersection of the two sets: $U_1 = \{u_R(k) : \dot{\phi}(k) \leq \eta$ when $\phi(k) \geq 0\}$ and $U_2 = \{u_R(k) : \phi(k+1) < 0\}$. The computation of $U_1$ follows from (9.6). The computation of $U_2$ is similar and is discussed in details in [83]. The metric $Q$ is chosen to be $M(\theta)$, which puts larger penalties on the torque modification applied to heavier link, thus is energy efficient.

The simulation result is shown in Fig.9.5. The first plot in Fig.9.5a shows the critical point on the arm that is the closest to the human capsule. The orange area represents the first link ($y = 0$ is the base) and yellow area represents the second link ($y = 0.55m$ is the endpoint). The second plot shows the distance from the robot endpoint to the robot's goal position. The third plot shows the relative distance $d$ between the robot capsules and the human capsule, while the red area represents the danger zone $\{d < d_{min}\}$. The bars in

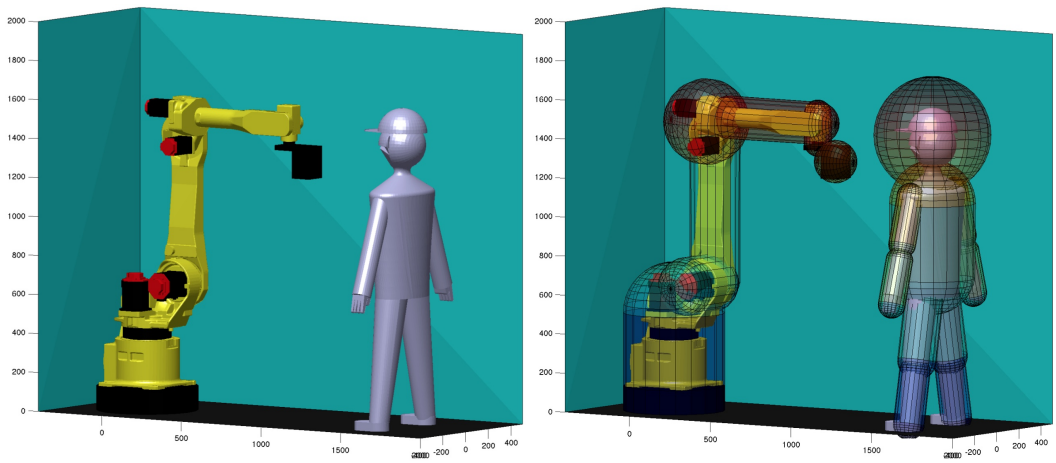



(a) The simulation profile.




(b) The snapshots at $k = 110 : 5 : 140$ (c) The snapshots at $k = 160 : 5 : 220$

Figure 9.5: The simulation result of the planar robot.



(a) The 3D simulation environment (b) The capsules

Figure 9.6: The 6DoF robot arm and the simulation environment.

the fourth plot illustrate whether the safety controller is active (green) or not (white) at each time step. During the simulation, the robot was close to its goal at $k = 55$ and at $k = 110$ before it finally approached it at $k = 220$. However, since the human was too close to the robot in that two cases, going to the goal was dangerous. Then the safety controller went active and the robot arm detoured to avoid the human. This scenario is also illustrated in Fig.9.5b, the 5-times down-sampled snapshots from time step 110 to 140, denote as $k = 110 : 5 : 140$. Lighter color corresponds to smaller $k$. Due to the safety controller, the relative distance was always maintained above the danger zone. Figure 9.5c shows the snapshots at $k = 160 : 5 : 220$. As the human was far from the robot arm, the safety controller was inactive and the robot finally approached its goal.

### 9.4.2 Six Degree of Freedom Robot Arm

In this case study, the Fanuc M16iB robot arm is used as shown in Fig.1.1a and the simulation environment is shown in Fig.9.6a. Capsules are calculated for both the human and the robot as shown in Fig.9.6b. The radius of the capsules are designed such that one uniform minimum distance requirement $d_{min} = 0.2m$ can be used for all capsules. Denote the robot state as $x_R = [\theta^T, \dot{\theta}^T]^T$ where $\theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]^T$ are the joint angles. $u_R = \ddot{\theta}$ is the joint acceleration. The control modification is done in the kinematic level. A perfect low level tracking controller is assumed. The state space equation of the robot arm is linear:

$$\dot{x}_R = Ax_R + Bu_R, \tag{9.9}$$

where $A_R = \begin{bmatrix} 0_{6\times6} & I_{6\times6} \\ 0_{6\times6} & 0_{6\times6} \end{bmatrix}$ and $B_R = \begin{bmatrix} 0_{6\times6} \\ I_{6\times6} \end{bmatrix}$.

$G_R$ is to follow a path in the Cartesian space. The baseline controller is a feedback and feedforward controller. The human is moving around the robot arm. The safety index is the same as in the previous case, e.g. $\phi = D - d^2 - \dot{d}$, where $d$ is computed analytically [35]. The sampling frequency is $20hz$. The forward-looking criteria is used. The set of safe control is $U_R^S(k) = \{u_R(k) : \phi(k+1) < 0\}$, and $Q = I$.

The simulation results are shown in Fig.9.7, Fig.9.8 and Fig.9.9. The first plot in Fig.9.7 shows the critical capsule ID on the robot arm that contains the closest point to the human and the second plot shows the critical capsule ID on the human that contains the closest point to the robot. During interactions, those critical points changed from time to time. The minimum distance between the human and the robot is shown in the third figure, which was maintained above the danger zone during the simulation. The tracking error is shown in the fourth plot. When the human was far from the robot, perfect tracking can be achieved from $k = 100$ to $k = 200$. When the human went close to the robot at $k = 230$, the safety controller took over and moved the robot arm away from the human, at the cost of large tracking error. The snapshots at $k = 230 : 5 : 250$ are shown in Fig.9.8. Another human avoidance behavior at $k = 310 : 10 : 350$ is shown in Fig.9.9, with the solid spheres representing the reference path at each time step. The robot stopped tracking the path that moved towards the human by moving backward. In this simulation, the human subject can only control the planar movement of the dummy. The simulation that captures human's whole body movement using Kinect is shown in the video attachment.

The algorithms are run in Matlab on a MacBook of 2.3 GHz using Intel Core i7. The running time of the safety controller is shown in Table 9.1. The average running time of the safety controller is 9.5ms, which is dominated by the time in finding the critical points, e.g. calculating the minimum distance between the robot and the human. This is because finding the critical points involves $6 \times 10$ distance calculations between capsules. If only the first three joints of the robot arm are considered, e.g. only three robot capsules are used in calculation, the running time is reduced to 5.5ms. If the number of human capsules is reduced to two, the running time for the safety controller is reduced to 2.7ms. Moreover, the running time of the safety controller is only 0.77ms if the human geometry is represented using spheres. However, the spheres cannot describe the geometry as accurate as the capsules do and may be too conservative. In conclusion, current algorithms can support at least $100Hz$ sampling frequency and the computation time can be further reduced if faster algorithms are developed for distance calculation. The video of this study can be found in [74].

## 9.5 Discussion and Conclusion

This chapter discussed the algorithmic safety measures for industrial robots working in a human-involved environment. The control problem was posed as a constrained optimal control problem and a unique parallel controller structure was proposed to solve the problem. The control problem was separated into two parts: the efficiency goal with time-invariant constraints and the time-varying safety constraint. The first part was solved by the baseline

Table 9.1: Running time of the safety controller.

| Robot arm: degree of freedom | Human model | Running time of the safety controller | Running time in finding critical points |
|---|---|---|---|
| 6 DoF | 10 capsules | 9.5ms | 8.8ms |
| 3 DoF | 10 capsules | 5.5ms | 5.0ms |
| 6 DoF | 2 capsules | 2.7ms | 2.0ms |
| 3 DoF | 10 spheres | 0.77ms | 0.40ms |

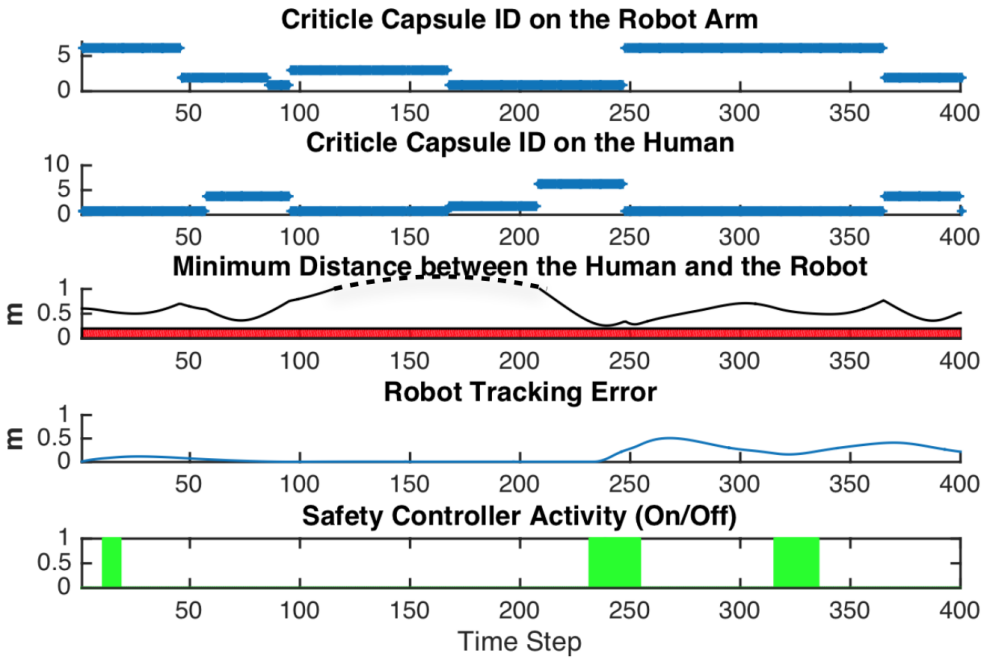
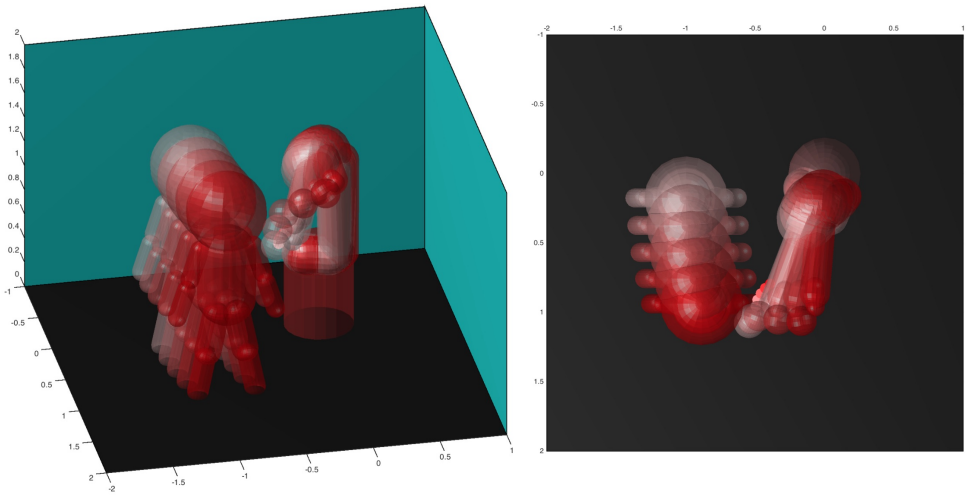



Figure 9.7: The simulation profile of the 6DoF robot arm

controller and the safety constraint was enforced by the safety controller. This separation is ideal due to the following reasons:

- There is no need to solve the original problem in a long time horizon, since the uncertainties of the human motion will accumulate. And the safety constraint $R_S(x_H)$ is only active in a small amount of time as evidenced in the simulations. The separation respects different natures of the constraints, by allowing the baseline controller to do long term planning without the time varying constraint and letting the safety controller to do local modification regarding the time varying constraint.

- This separation can also be validated by analytically solving the optimal control problem. Suppose $G_R = \{x_R = 0\}$ and $J = \int_0^T (x_R^T P x_R + u_R^T R u_R) dt$. Let $\Omega, \Gamma$ be the whole



(a) The lateral view (b) The top view

Figure 9.8: The simulated response of the 6DoF robot arm: scenario 1

space and $R_S(x_H) = \{x_R : \phi(x_R, x_H) < 0\}$. Assume $h(x_R) = B$. Then the Lagrangian [54] of the optimal control problem (9.2a-9.2c) is

$$L = x_R^T P x_R + u_R^T R u_R + \lambda(f(x_R) + B u_R) + \eta\dot{\phi}, \tag{9.10}$$

where $\lambda, \eta$ are adjoint variables and $\eta = 0$ if $\phi < 0$. The partial derivatives from $L$ to $u_R$ is $L_u = 2(Ru_R)^T + \lambda B + \eta\phi_{x_R} B$. Setting $L_u = 0$, the optimal control law becomes

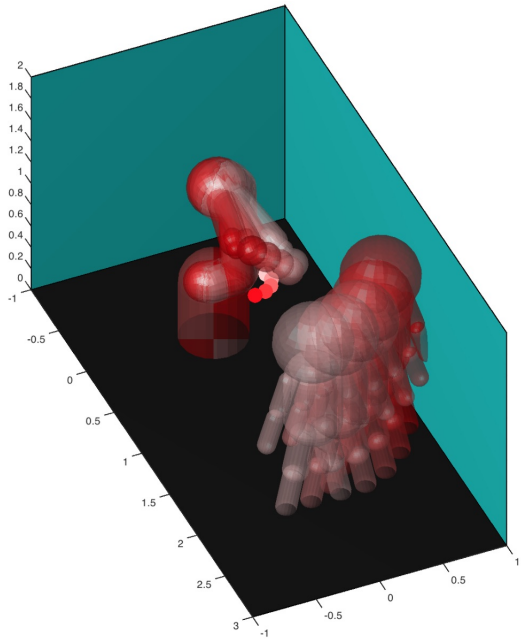
$$u_R = -\frac{1}{2}R^{-1}B^T\lambda^T - \frac{1}{2}\eta R^{-1}B^T\phi_{x_R}^T, \tag{9.11}$$

where the first term on the RHS is not related to the safety constraint, which can be viewed as the baseline control law; the second term is concerned with the safety constraint, which is nontrivial only if $\phi \geq 0$, e.g. the safety constraint is violated. Nonetheless, the optimality of this separation will be studied for more complicated problems in the future.

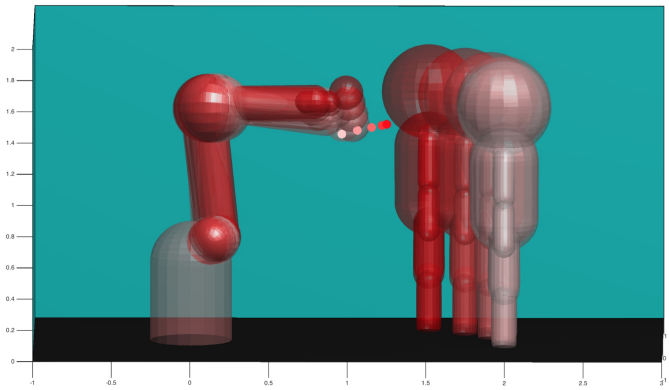
Moreover, the separation offers more freedom in designing the robot behavior and is good for parallel computation.

In conclusion, the controller design procedure is:

1. Design the baseline controller that can handle the goal and the time invariant constraints.
2. Wrap every moving rigid body with a capsule to simplify the geometry.
3. Design the safe set $X_S$ which specifies the required distance among capsules.



(a) The lateral view



(b) The left view

Figure 9.9: The simulated response of the 6DoF robot arm: scenario 2

4. Design the safety index $\phi$ based on the safe set $X_S$ and the robot dynamics.

5. Choose the control modification criteria and design the control modification metric $Q$.

6. Design the human motion predictor.

To fully realize the scenario in Fig.9.1, more aspects in the controller design needs to be investigated. For example, as the number of agents in the system increases, the non-convexity of the problem will increase. Methods to avoid local optima need to be developed.

Moreover, the safe control method for human-robot cooperation that involves contacts also needs to be studied.

Nonetheless, the controller structure proposed is of importance as it is a method to handle constraints of different natures and to deal with multiple objectives, whose effectiveness is demonstrated both in simulation and in analysis.

# Chapter 10

# Final Words

In this dissertation, we explored the methods to design the robot behavior toward safe and efficient human-robot interactions in various scenarios. A three component behavior system was proposed, which consisted of knowledge, logic and learning. The design of the knowledge, especially the internal cost, was discussed with respect to specific applications in Chapter 8 and Chapter 9. The design of the logic in terms of motion planning was discussed in Chapter 4 and Chapter 5. To equip the robot with a global perspective and ensure timely responses in emergencies, we developed a parallel planning and control architecture, which consisted of an efficiency-oriented long term planner and a safety-oriented short term planner. Chapter 6 discussed the solvers to speed up the computation in the long term motion planning. The design of the learning process was discussed in Chapter 3, where the humans' behaviors were identified and predicted. The overall architecture for behavior design as well as the multi-agent model for the human-robot system were discussed in Chapter 2. The method to evaluate the performance of the multi-agent system given the design behavior was discussed in Chapter 7.

There are many directions for future work, which are listed below.

- To account for diverse modes of interactions

  This dissertation focus on parallel human-robot relationship. In the future, hierarchical human-robot relationships will be explored, especially the interaction between a human driver and an driver-assistive system and the interaction between a human and a human-assistive device such as exoskeleton. How to increase the "transparency" of such robots to make them understandable to human users is still challenging.

- To consider communication among agents

  Action of an agent may have various purposes, which can be divided into two categories, *target action* and *communicative action*. Target action is directed toward task performance. Communicative action is to inform others one's intention so that the task can be performed smoothly, which contributes to task performance indirectly. For example, in a lane change maneuver, an automated vehicle would switch on the turn signal

and turn the steering wheel. The wheel motion is a target action and the turn signal light is a communicative action. This dissertation focused on target action achieved by physical movement. However, physical movement can also be communicative as it reveals one's intention to observers. In [32], such motion is called legible motion. Communicative motion will be explicitly incorporated into the design in the future. In addition, it will also be beneficial to explore the effect of facial expression [101] and spoken language [31] during interactions. On the other hand, with the development of dedicated short range communication (DSRC) technology [60], vehicles are able to perform vehicle to vehicle (V2V) and vehicle to infrastructure (V2I) communication. Such communication will significantly change the interactions among vehicles. For example, if vehicles can directly exchange information about their intended behaviors, the difficulty of predicting other vehicles' trajectories (as discussed in Chapter 3 and Chapter 8) would be greatly reduced. A method to incorporate the communicated information into motion planning is discussed in [88]. In the future, the effect of direct communication among agents will be considered.

- To improve computation efficiency

  Computation efficiency of the designed algorithms is extremely important. In this dissertation, we developed algorithms to speed up the computation of non-convex optimization problems that had convex objectives and satisfied several geometric properties. However, the assumptions narrowed the scope of the proposed algorithms, hence should be relaxed in the future. In addition to algorithm innovation, another approach to improve computation efficiency is to solidify the software into hardware. For example, we can directly build the functions that will be frequently called, such as the non-convex optimization, into the hardware. The possibility will be explored in the future.

- Analysis, synthesis and evaluation of complex human-robot systems

  The analysis of complex human-robot systems remains challenging. We discussed one tool (e.g. the trapped equilibrium) to analyze the quadratic games under information asymmetry in this dissertation. However, methods to analyze more complicated dynamics are needed. Such methods should also provide a way to compare against different designs.

With the development of technology, it may no longer be a fantasy to have intelligent and autonomous robots that think, behave and interact with the world in the way that human beings do, so that they can better serve, assist and collaborate with people in their daily lives across work, home and leisure.

# Appendix A

# Evaluation Platforms

## A.1 Overview

To evaluate the performance of the designed behavior system, pure simulation is not enough. However, to protect human subjects, it is desirable if we can separate human subjects and robot physically during the early phase of deployment. In this chapter, we discuss possible evaluation platforms and the platforms we developed to evaluate the designed behavior system. Figure A.1 shows five different evaluation platforms.

The first platform is the virtual reality-based human-in-the-loop platform. The robot's motion is simulated in the robot simulator. A human subject observes the robot movement through the virtual reality display (e.g. virtual reality glasses, augmented reality glasses or monitors). The reaction of the human subject is captured by sensors (e.g. Kinect or touchpad). The sensor data then sent to the behavior system for compute desired control input. The advantage of such platform is that it is safe to human subjects and convenient for idea testing. The disadvantage is that the robot simulator may neglect dynamic details of the physical robot, hence not reliable.

The second platform is the virtual reality-based hardware-in-the-loop platform. The only difference from the first platform is that the robot motion is no longer simulated, but directly measured from the robot hardware. This solves the problem of the mis-match problem that the robot simulator may have. However, as the interaction happens virtually, the human subjects may not react in the way that they will do with real robots.

The third platform is the dummy-robot interaction platform. The interaction happens physically between the dummy and the robot. The human directly observes the robot motion and controls the dummy to interact with the robot. As there are physical interactions, an emergency check module should be added to ensure safety. The advantage of such platform is that it is safe to human subjects, while it is able to test interactions physically. However, the disadvantage is that the dummy usually doesn't have as many degrees of freedom as a human subject does.

The fourth platform is the robot-robot interaction platform. Each robot is regarding

the other as "human". In general, such platform allows us to inquiry whether the designed behavior system can cope with intelligent entities other than human. Moreover, it can also tell us whether the multi-agent system consists of those intelligent robots are stable or not.

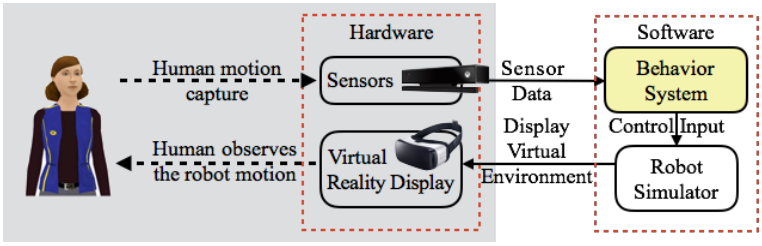
The fifth platform is the human-robot interaction platform where human subjects directly interact with robots. Such platform should be designed to illustrate the performance of the robot in real tasks.

In this dissertation, we focus on platform one and platform three. Section A.2 introduces a multi-vehicle human-in-the-loop platform, which is used in Chapter 8. Section A.3 introduces a human-in-the-loop simulation platform for industrial robots, which is used in Chapter 9. Section A.4 introduces a dummy-robot interaction platform for industrial robots. The videos with those platforms can be found in [73–75].

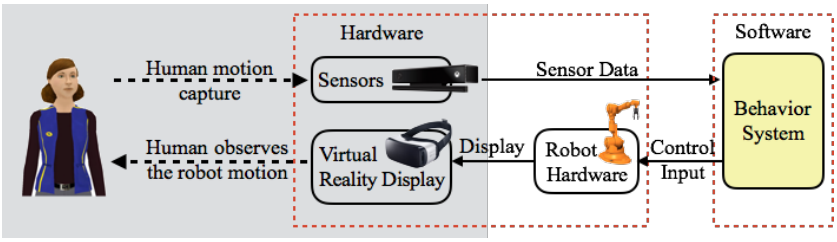
## A.2 A Multi-Vehicle Human-in-the-Loop Simulation Platform

To ensure safety, a simulation environment is needed in order to evaluate the algorithms before the road test. Existing vehicle and traffic simulators often suffer from imbalanced coverage of macro traffic dynamics and micro vehicle dynamics, e.g. some either contain only high level decision making and omit the realistic vehicle dynamics, others involve too many details of the low-level vehicle control and dynamics and omit the high-level decision making and traffic. In order to simulate the real-world driving scenarios, a multi-vehicle simulator is needed to consider both interactions between vehicles, and the dynamic details of every single vehicle. For this purpose, we developed an object-oriented simulator. A physic engine (bullet) is embedded in the simulator to simulate the real-world physical phenomenon (e.g. collision, friction, and gravity) as well as the vehicle dynamics to make it realistic. The simulator consists of four modules: environment, sensor, agent and vehicle, as an analogy of real-world driving, where each human driver is considered as an agent, who uses his sensors (e.g. eyes) to get information (e.g. distance from the front vehicle) from the environment, provides control inputs (e.g. turning the steering wheel) to move the vehicle so as to influence the environment.

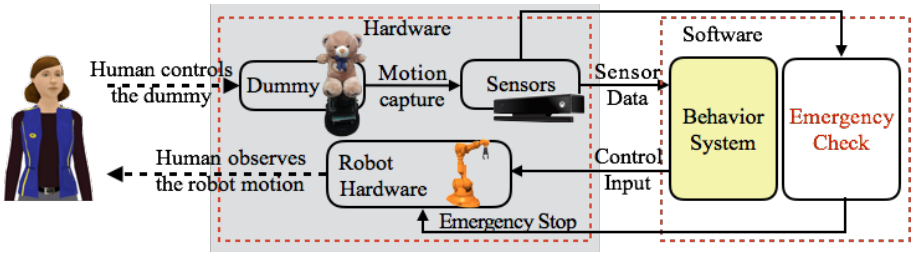
In the simulator shown in Fig.A.2, multiple vehicles are running in the environment and are interacting with the environment (e.g. friction and contact with the road, collision with surrounding vehicles). The environment contains road map and all road participants. Each road participant has its own dynamics and is associated with an agent which controls the motion of the road participant. The agent may be software-controlled (e.g. agents that run the ROAD system autonomously) or human-controlled (e.g. agents that read human commands from input devices such as a wheel or a keyboard). Each software agent is associated with a sensor through which it can acquire information from the environment. For now, the sensors can access a noisy measurement of the positions and velocities of the ego vehicle and the surrounding vehicles. In the future, we will add more realistic sensor



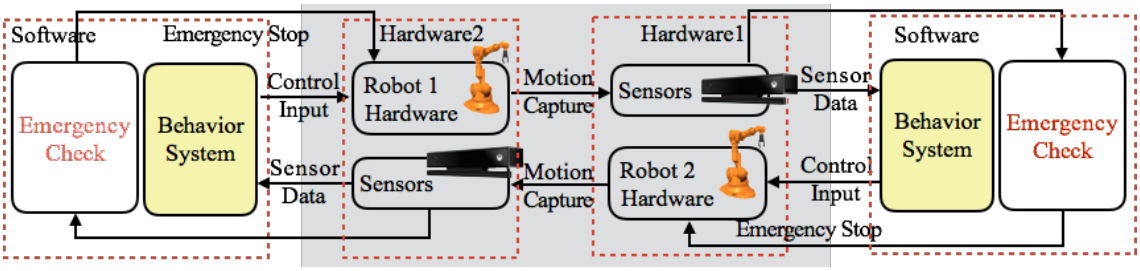

(a) Virtual reality-based human-in-the-loop platform.



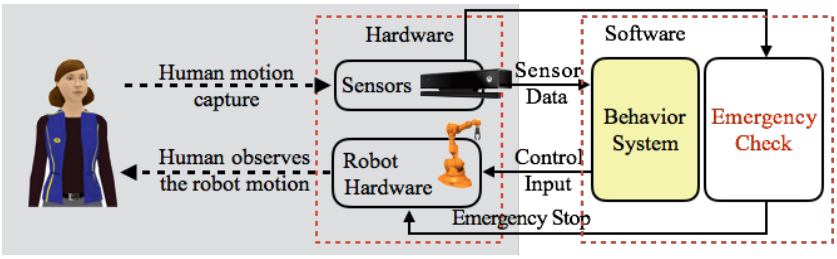

(b) Virtual reality-based hardware-in-the-loop platform.




(c) Dummy-robot interaction platform.




(d) Robot-robot interaction platform.




(e) Human-robot interaction platform.

Figure A.1: The evaluation platforms.

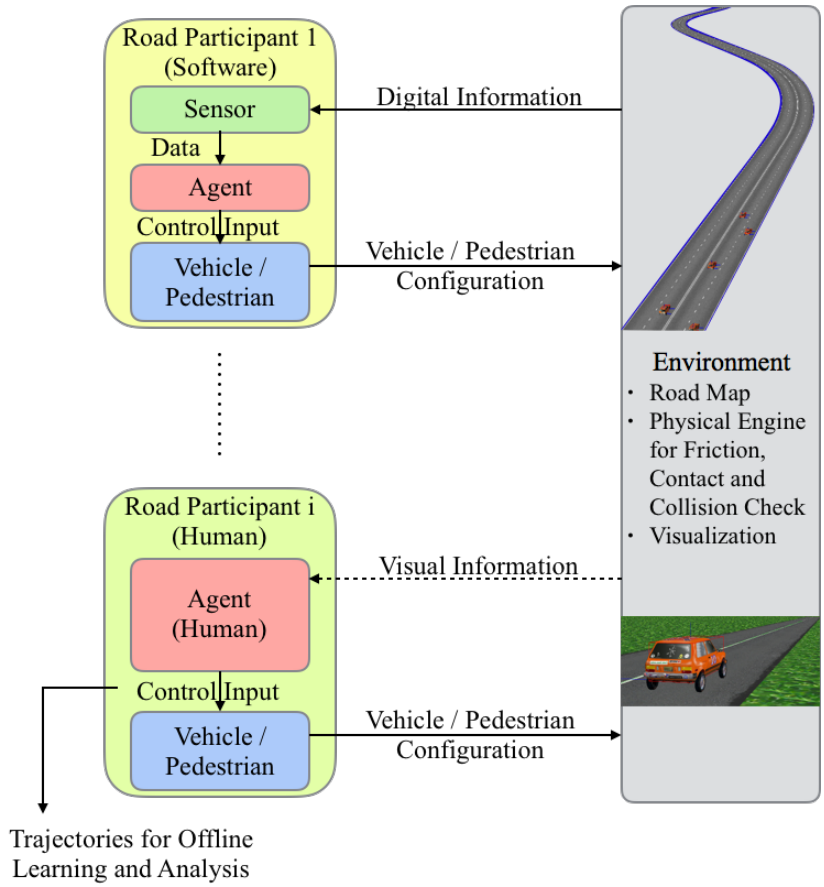
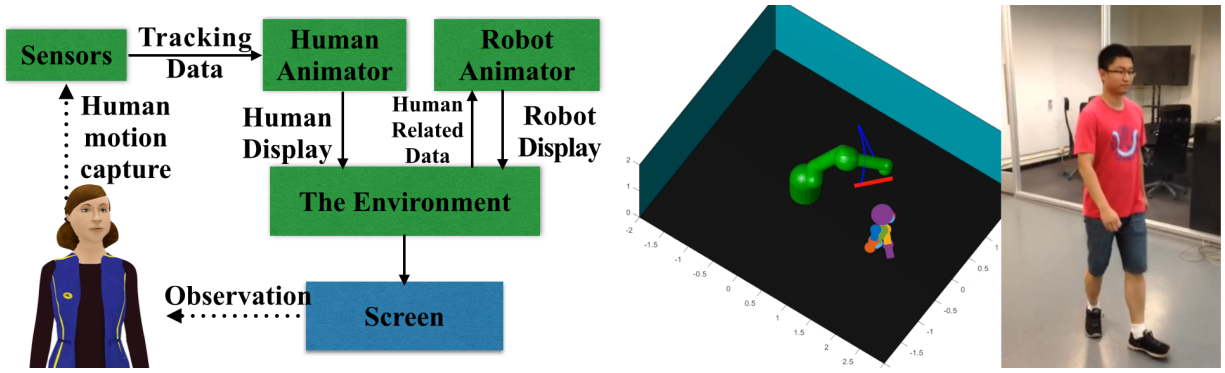



Figure A.2: A multi-vehicle platform to evaluate autonomous driving

models to simulate Lidar, radar and camera. Using the sensor data, the software agent for a vehicle then makes the driving decisions and sends out the steering, throttle and brake signals to the vehicle. Other than ROAD system, we developed various classes of software agents representing different driving characteristics in order to mimic the real-world scenarios. On the other hand, a human agent observes the current configuration of the virtual environment through a visual display and controls the assigned road participant using input devices.

The trajectories of vehicles or pedestrians controlled by human agents are recorded in every simulation, which serve as the data source during offline learning in layer 2. During offline learning, the trajectories are labelled manually regarding the intended driving behavior at each time step. Then the classifier for behavior prediction and the empirical models for trajectory prediction under different driving behaviors are learned from the labelled data. The human subjects are volunteers, who also participate in the evaluation process of the ROAD system during simulation. For different driving scenarios, e.g. freeway driving or urban driving, different classifiers need to be trained as the types of road participants and their behaviors in those scenarios differ. Note that discrepancies exist between the trajecto-




(a) The software architecture. (b) The human-in-the-loop simulation.

Figure A.3: The human-in-the-loop simulation platform for industrial robots.

ries of road participants controlled by human agents in the simulation and the trajectories of manually driven vehicles (or pedestrians) in the real world, since the human subjects may behave differently in virtual reality and in reality.

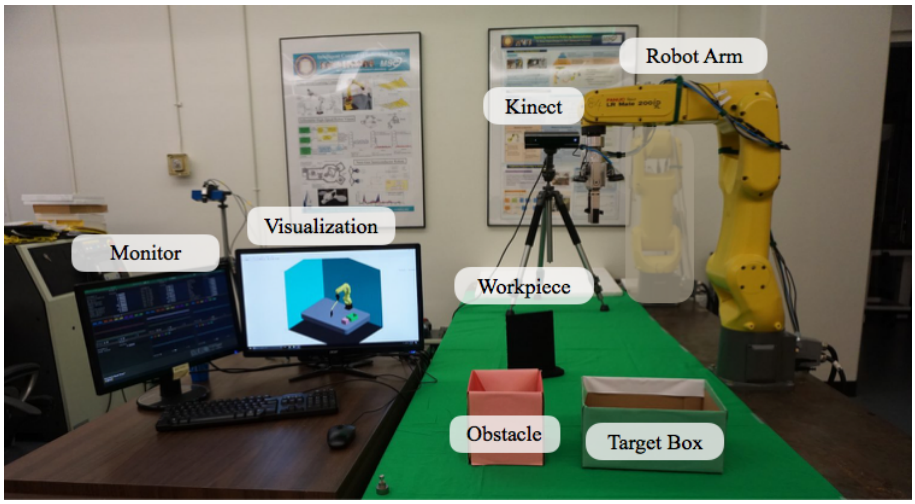
## A.3 A Human-in-the-loop Simulation Platform for Industrial Robots

The platform is shown in Fig.A.3, which consists of the human loop, the robot loop and the environment. The human subject is in the human loop, who can observe the virtual environment through the screen and whose reaction will be captured by the Kinect. The human animator reads the tracking data from the sensors and sends the human figure to the environment for display. In the robot loop, the robot animator reads the noisy human data from the environment, computes the safe and efficient trajectory and then sends the real time robot figure to the environment.

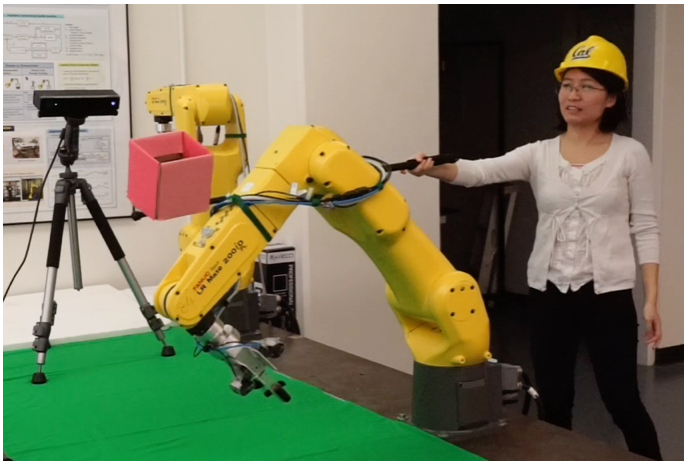
The human skeleton data can be stored for offline analysis in order to let the robot build better cognitive models.

## A.4 A Dummy-Robot Platform for Industrial Robots

The platform is shown in Fig.A.4. This platform is for a pick-and-place task. We have a 6 DOF industrial robot arm, a kinect for environment monitoring, a workpiece, a target box and an obstacle. The environment perceived by the robot is visualized in a screen. The status of the robot is shown in the monitor. The robot needs to pick the workpiece and place it in the target box while avoiding the dynamic obstacle. The scenario can be viewed as an abstraction of the human-robot collaborative assembly we discussed before. The human




(a)



(b)

Figure A.4: The dummy-robot platform for industrial robots

hand that accept the tool is similar to the green target box. The other human hand is similar to the obstacle. Hence the target box and the obstacle can be considered as dummies. The human subjects can control the position of the obstacle at a distance to the robot arm as shown in Fig.A.4b.

# Bibliography


[1] P. Abbeel and A. Y. Ng. "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the International Conference on Machine learning (ICML)*. 2004, p. 1.

[2] B. Açıkmeşe, J. M. Carson, and L. Blackmore. "Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem". In: *IEEE Transactions on Control Systems Technology* 21.6 (2013), pp. 2104–2113.

[3] N. Aharony et al. "Social fMRI: Investigating and shaping social mechanisms in the real world". In: *Pervasive and Mobile Computing* 7.6 (2011), pp. 643–659.

[4] B. Akgun et al. "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective". In: *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*. 2012, pp. 391–398.

[5] T. M. Anandan. *Major Robot OEMs Fast-Tracking Cobots*. 2014. URL: `http://www.robotics.org/`.

[6] T. Arai, R. Kato, and M. Fujita. "Assessment of operator stress induced by robot collaboration in assembly". In: *CIRP Annals-Manufacturing Technology* 59.1 (2010), pp. 5–8.

[7] B. D. Argall et al. "A survey of robot learning from demonstration". In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 469–483.

[8] J. Bae and M. Tomizuka. "Gait phase analysis based on a Hidden Markov Model". In: *IEEE/ASME Transactions on Mechatronics* 21.6 (2011), pp. 961–970.

[9] L. Bainbridge. "Ironies of automation". In: *Automatica* 19.6 (1983), pp. 775–779.

[10] A. Bandura. "Social cognitive theory: An agentic perspective". In: *Annual review of psychology* 52.1 (2001), pp. 1–26.

[11] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Vol. 200. London: Academic Press, 1995.

[12] *Baxter from Rethink Robotics*. URL: `http://www.rethinkrobotics.com/products/baxter/`.

[13] G. A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. MIT press, 2005.

[14] J. van den Berg. "Extended LQR: locally-optimal feedback control for systems with non-linear dynamics and non-quadratic cost". In: *Robotics Research*. Springer, 2016, pp. 39–56.

[15] C. R. Berger and R. J. Calabrese. "Some explorations in initial interaction and beyond: Toward a developmental theory of interpersonal communication". In: *Human Communication Research* 1.2 (1975), pp. 99–112.

[16] D. Bertsimas, V. Gupta, and I. C. Paschalidis. "Data-driven estimation in equilibrium using inverse optimization". In: *Mathematical Programming* (2014), pp. 1–39.

[17] P. T. Boggs and J. W. Tolle. "Sequential quadratic programming". In: *Acta numerica* 4 (1995), pp. 1–51.

[18] P. V. K. Borges, N. Conci, and A. Cavallaro. "Video-based human behavior understanding: a survey". In: *IEEE Transactions on Circuits and Systems for Video Technology* 23.11 (2013), pp. 1993–2008.

[19] S. Bouchard. "With Two Arms and a Smile, Pi4 Workerbot Is One Happy Factory Bot". In: *IEEE Spectrum* (2011).

[20] C. Breazeal. "Social interactions in HRI: the robot view". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews* 34.2 (2004), pp. 181–186.

[21] L. D. Burns. "Sustainable mobility: a vision of our transport future". In: *Nature* 497.7448 (2013), pp. 181–182.

[22] R. H. Byrd, J. Nocedal, and R. A. Waltz. "Knitro: An Integrated Package for Nonlinear Optimization". In: *Large-Scale Nonlinear Optimization*. Ed. by G. Di Pillo and M. Roma. Boston, MA: Springer US, 2006, pp. 35–59.

[23] O. Cappé, E. Moulines, and T. Rydén. *Inference in Hidden Markov Models*. Springer Science & Business Media, 2006.

[24] G. Charalambous. "Human-automation collaboration in manufacturing: Identifying key implementation factors". In: *Proceedings of the International Conference on Ergonomics & Human Factors*. CRC Press, 2013, p. 59.

[25] H. Cheng. "Vision and Inertial Sensor Based Drive Trains Control". PhD thesis. University of California at Berkeley, 2010.

[26] R. Chipalkatty et al. "Human-in-the-loop: MPC for shared control of a quadruped rescue robot". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011, pp. 4556–4561.

[27] F. H. Clarke. "Generalized gradients and applications". In: *Transactions of the American Mathematical Society* 205 (1975), pp. 247–262.

[28] K. Dautenhahn. "Robots we like to live with?!– A developmental perspective on a personalized, life-long robot companion". In: *Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication*. 2004, pp. 17–22.

[29] K. Dautenhahn. "Socially intelligent robots: Dimensions of human–robot interaction". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 362.1480 (2007), pp. 679–704.

[30] M. B. Dias et al. "Sliding autonomy for peer-to-peer human-robot teams". In: *Proceedings of the Intelligent Conference on Intelligent Autonomous Systems (IAS)*. 2008, pp. 332–341.

[31] F. Doshi and N. Roy. "Spoken language interaction with model uncertainty: an adaptive human–robot interaction system". In: *Connection Science* 20.4 (2008), pp. 299–318.

[32] A. Dragan. "Legible Robot Motion Planning". PhD thesis. Carnegie Mellon University, 2015.

[33] *Driverless car market watch*. 2015. URL: `http://www.driverless-future.com`.

[34] N. E. Du Toit and J. W. Burdick. "Robot motion planning in dynamic, uncertain environments". In: *IEEE Transactions on Robotics* 28.1 (2012), pp. 101–115.

[35] D. Eberly. *Robust Computation of Distance Between Line Segments*. 2015.

[36] G. Eichfelder and J. Povh. "On the set-semidefinite representation of nonconvex quadratic programs over arbitrary feasible sets". In: *Optimization Letters* 7.6 (2013), pp. 1373–1386.

[37] G. B. Folland. *Real Analysis: Modern Techniques and Their Applications*. John Wiley & Sons, 2013.

[38] T. Fong, I. Nourbakhsh, and K. Dautenhahn. "A survey of socially interactive robots". In: *Robotics and Autonomous Systems* 42.3 (2003), pp. 143–166.

[39] K. Fragkiadaki et al. "Recurrent network models for human dynamics". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4346–4354.

[40] A. Franchi et al. "Bilateral teleoperation of groups of mobile robots with time-varying topology". In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1019–1033.

[41] M. Golub, S. Chase, and M. Y. Byron. "Learning an internal dynamics model from control demonstration". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2013, pp. 606–614.

[42] D. González et al. "A Review of Motion Planning Techniques for Automated Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (2016), pp. 1135–1145.

[43] M. A. Goodrich and A. C. Schultz. "Human-robot interaction: A survey". In: *Foundations and Trends in Human-Computer Interaction* 1.3 (2007), pp. 203–275.

[44] G. C. Goodwin and K. S. Sin. *Adaptive Filtering Prediction and Control*. Courier Dover Publications, 2013.

[45] L. Gracia, F. Garelli, and A. Sala. "Reactive siding-mode algorithm for collision avoidance in robotic systems". In: *IEEE Transactions on Control Systems Technology* 21.6 (2013), pp. 2391–2399.

[46] E. Gribovskaya, A. Kheddar, and A. Billard. "Motion learning and adaptive impedance for robot control during physical interaction with humans". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 4326–4332.

[47] T. Gu, J. M. Dolan, and J.-W. Lee. "Runtime-bounded tunable motion planning for autonomous driving". In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2016, pp. 1301–1306.

[48] T. Gu et al. "Tunable and stable real-time trajectory planning for urban autonomous driving". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 250–256.

[49] S. Haddadin et al. "Collision detection and reaction: A contribution to safe physical human-robot interaction". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2008, pp. 3356–3363.

[50] S. Haddadin et al. "New insights concerning intrinsic joint elasticity for safety". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 2181–2187.

[51] S. Haddadin et al. "Towards the robotic co-worker". In: *Robotics Research*. Vol. 70. Springer Berlin Heidelberg, 2011, pp. 261–282.

[52] C. Harper and G. Virk. "Towards the development of international safety standards for human robot interaction". In: *International Journal of Social Robotics* 2.3 (2010), pp. 229–234.

[53] M. W. Harris and B. Açıkmeşe. "Lossless convexification of non-convex optimal control problems for state constrained linear systems". In: *Automatica* 50.9 (2014), pp. 2304–2311.

[54] R. F. Hartl, S. P. Sethi, and R. G. Vickson. "A survey of the maximum principles for optimal control problems with state constraints". In: *SIAM Review* 37.2 (1995), pp. 181–218.

[55] G. Hirzinger et al. "On a new generation of torque controlled light-weight robots". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 4. 2001, pp. 3356–3363.

[56] R. Hof. "Toyota: 'Guardian Angel' cars will beat self-driving cars". In: *Forbes* (2016).

[57] T. M. Howard, C. J. Green, and A. Kelly. "Receding horizon model-predictive control for mobile robot navigation of intricate paths". In: *Field and Service Robotics*. 2010, pp. 69–78.

[58] T. A. Johansen, T. I. Fossen, and S. P. Berge. "Constrained nonlinear control allocation with singularity avoidance using sequential quadratic programming". In: *IEEE Transactions on Control Systems Technology* 12.1 (2004), pp. 211–216.

[59] *Kawada's NextAge Robot*. 2011. URL: `http://nextage.kawada.jp/en/`.

[60] J. Kenney. "Dedicated Short-Range Communications (DSRC) Standards in the United States". In: *Proceedings of the IEEE* 99.7 (July 2011), pp. 1162–1182. ISSN: 0018-9219. DOI: `10.1109/JPROC.2011.2132790`.

[61] O. Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *The International Journal of Robotics Research* 5.1 (1986), pp. 90–98.

[62] R. Koeppe et al. "Robot-robot and human-robot cooperation in commercial robotics applications". In: *Robotics Research* (2005), pp. 202–216.

[63] K. Kong, J. Bae, and M. Tomizuka. "Control of rotary series elastic actuator for ideal force-mode actuation in human–robot interaction applications". In: *IEEE/ASME Transactions on Mechatronics* 14.1 (2009), pp. 105–118.

[64] J. Krüger, T. K. Lien, and A. Verl. "Cooperation of human and machines in assembly lines". In: *CIRP Annals-Manufacturing Technology* 58.2 (2009), pp. 628–646.

[65] A. Kuefler et al. "Imitating Driver Behavior with Generative Adversarial Networks". In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2017.

[66] J. J. Kuffner and S. M. LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2. 2000, pp. 995–1001.

[67] I. D. Landau, R. Lozano, and M. M'Saad. *Adaptive control*. Vol. 51. Springer, 1998.

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

[69] J. Leber. *At Volkswagen, Robots Are Coming Out Of Their Cages*. 2013. URL: `http://www.fastcoexist.com/`.

[70] I. Lee, E. Kim, and E. M. Marcotte. "Modes of interaction between individuals dominate the topologies of real world networks". In: *PLOS ONE* 10.3 (2015), pp. 1–12.

[71] X. Li et al. "A practical trajectory planning framework for autonomous ground vehicles driving in urban environments". In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2015, pp. 1160–1166.

[72] M. L. Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 157. 1994, pp. 157–163.

[73] C. Liu. *Real Time Robot Motion Planning in Dynamic Uncertain Environment*. URL: `https://youtu.be/o2X1Wsd046g`.

[74] C. Liu. *Robot Safe Interaction System - Safe Human Robot Co-inhabitance*. URL: `https://youtu.be/GHd9hgpKIjQ`.

[75] C. Liu. *Robustly-Safe Automated Driving (ROAD) System - Freeway Driving in Mixed Traffic*. URL: `https://youtu.be/e3EUHDAeTKs`.

[76] C. Liu, C.-Y. Lin, and M. Tomizuka. "The convex feasible set algorithm for real time optimization in motion planning". In: *SIAM Journal on Control and Optimization* arXiv:1709.00627 (2017), under review.

[77] C. Liu and M. Tomizuka. "Algorithmic safety measures for intelligent industrial co-robots". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 3095–3102.

[78] C. Liu and M. Tomizuka. "Control in a safe set: Addressing safety in human robot interactions". In: *Proceedings of the ASME Dynamic Systems and Control Conference (DSCC)*. 2014, V003T42A003.

[79] C. Liu and M. Tomizuka. "Designing the robot behavior for safe human robot interactions". In: *Trends in Control and Decision-Making for Human-Robot Collaboration Systems*. Springer, 2017, pp. 241–270.

[80] C. Liu and M. Tomizuka. "Enabling safe freeway driving for automated vehicles". In: *Proceedings of the American Control Conference (ACC)*. 2016, pp. 3461–3467.

[81] C. Liu and M. Tomizuka. "Modeling and controller design of cooperative robots in workspace sharing human-robot assembly teams". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, pp. 1386–1391.

[82] C. Liu and M. Tomizuka. "Real Time Trajectory Optimization for Nonlinear Robotic Systems: Relaxation and Convexification". In: *System & Control Letters* 108 (2017), pp. 56–63.

[83] C. Liu and M. Tomizuka. "Safe exploration: Addressing various uncertainty levels in human robot interactions". In: *Proceedings of the American Control Conference (ACC)*. 2015, pp. 465–470.

[84] C. Liu and M. Tomizuka. "Who to blame? Learning and control strategies with information asymmetry". In: *Proceedings of the American Control Conference (ACC)*. 2016, pp. 4859–4864.

[85] C. Liu, Y. Wang, and M. Tomizuka. "Boundary layer heuristic for search-based non-holonomic path planning in maze-like environments". In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 831–836.

[86] C. Liu, W. Zhan, and M. Tomizuka. "Speed profile planning in dynamic environments via temporal optimization". In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 154–159.

[87] C. Liu et al. "Convex feasible set algorithm for constrained trajectory smoothing". In: *Proceedings of the American Control Conference (ACC)*. 2017, pp. 4177–4182.

[88] C. Liu et al. "Distributed conflict resolution for connected autonomous vehicles". In: *IEEE Transactions on Intelligent Vehicles* (2017), under review.

[89] C. Liu et al. "The Robustly-Safe Automated Driving System for Enhanced Active Safety". In: *SAE Technical Paper*. 2017-01-1406. 2017.

[90] C. Liu et al. "Path planning for autonomous vehicles using model predictive control". In: *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 174–179.

[91] J. Liu. *Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization, and Adaptive Computation*. World Scientific, 2001.

[92] X. Liu. "Autonomous trajectory planning by convex optimization". PhD thesis. Iowa State University, 2013.

[93] X. Liu and P. Lu. "Solving nonconvex optimal control problems by convex optimization". In: *Journal of Guidance, Control, and Dynamics* 37.3 (2014), pp. 750–765.

[94] L. Lu and J. T. Wen. "Human-robot cooperative control for mobility impaired individuals". In: *Proceedings of the American Control Conference (ACC)*. 2015, pp. 447–452.

[95] R. C. Luo et al. "Adaptive impedance control for safe robot manipulator". In: *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*. 2011, pp. 1146–1151.

[96] R. Luo and D. Berenson. "A framework for unsupervised online human reaching motion recognition and early prediction". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 2426–2433.

[97] V. Macagon and B. Wünsche. "Efficient collision detection for skeletally animated models in interactive environments". In: *Proceedings of Image and Vision Computing New Zealand (IVCNZ)*. Vol. 3. 2003, pp. 378–383.

[98] J. Mainprice and D. Berenson. "Human-robot collaborative manipulation planning using early prediction of human motion". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, pp. 299–306.

[99] J. L. Marble et al. "Evaluation of supervisory vs. peer-peer interaction with human-robot teams". In: *Proceedings of the Annual Hawaii International Conference on System Sciences*. 2004, pp. 1–9.

[100] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[101] M. Moosaei et al. "Using Facially Expressive Robots to Calibrate Clinical Pain Perception". In: *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*. 2017, pp. 32–41.

[102] X. Na and D. J. Cole. "Linear quadratic game and non-cooperative predictive methods for potential application to modelling driver–AFS interactive steering control". In: *Vehicle System Dynamics* 51.2 (2013), pp. 165–198.

[103] S. Nikolaidis and J. Shah. "Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy". In: *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*. 2013, pp. 33–40.

[104] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[105] *Optimization Toolbox, Constrained optimization, fmincon*. URL: `https://www.mathworks.com/help/optim/ug/fmincon.html`.

[106] U. Ozguner, C. Stiller, and K. Redmill. "Systems for safety and autonomous behavior in cars: The DARPA Grand Challenge experience". In: *Proceedings of the IEEE* 95.2 (2007), pp. 397–412.

[107] R. Parasuraman, T. B. Sheridan, and C. D. Wickens. "A model for types and levels of human interaction with automation". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 30.3 (2000), pp. 286–297.

[108] D.-H. Park et al. "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields". In: *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*. 2008, pp. 91–98.

[109] H.-K. Park et al. "A nursing robot system for the elderly and the disabled". In: *International Journal of Human-Friendly Welfare Robotic Systems* 2.4 (2001), pp. 11–16.

[110] V. Pavlovic et al. "A dynamic Bayesian network approach to figure tracking using learned dynamic models". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Vol. 1. 1999, pp. 94–101.

[111] J. Pineau et al. "Towards robotic assistants in nursing homes: Challenges and results". In: *Robotics and Autonomous Systems* 42.3 (2003), pp. 271–281.

[112] X. Qian et al. "Motion planning for urban autonomous driving using Bézier curves and MPC". In: *Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2016, pp. 826–833.

[113] E. Rasmusen and B. Blackwell. *Games and Information: An Introduction to Game Theory*. Cambridge, MA, 1994.

[114] J. Rasmussen. "Outlines of a hybrid model of the process plant operator". In: *Monitoring Behavior and Supervisory Control*. Springer, 1976, pp. 371–383.

[115] N. Ratliff et al. "CHOMP: Gradient optimization techniques for efficient motion planning". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2009, pp. 489–494.

[116] H. C. Ravichandar and A. Dani. "Human intention inference and motion modeling using approximate E-M with online learning". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 1819–1824.

[117] H. C. Ravichandar, A. Kumar, and A. Dani. "Bayesian human intention inference through multiple model filtering with gaze-based priors". In: *Proceedings of the International Conference on Information Fusion (FUSION)*. 2016, pp. 2296–2302.

[118] J. H. Reif and H. Wang. "Social potential fields: A distributed behavioral control for autonomous robots". In: *Robotics and Autonomous Systems* 27.3 (1999), pp. 171–194.

[119] W. Ren, R. W. Beard, et al. "Consensus seeking in multiagent systems under dynamically changing interaction topologies". In: *IEEE Transactions on Automatic Control* 50.5 (2005), pp. 655–661.

[120] P. E. Ross. "California to issue driving licences to robots". In: *IEEE Spectrum* (2014).

[121] D. Sadigh et al. "Planning for autonomous cars that leverages effects on human actions". In: *Proceedings of the Robotics: Science and Systems Conference (RSS)*. 2016.

[122] A. V. Savkin et al. *Safe Robot Navigation Among Moving and Steady Obstacles*. Butterworth-Heinemann, 2015.

[123] J. Schulman et al. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization." In: *Proceedings of the Robotics: Science and Systems Conference (RSS)*. Vol. 9. 1. 2013, pp. 1–10.

[124] M. Schwager et al. "Data-driven identification of group dynamics for motion prediction and control". In: *Journal of Field Robotics* 25.6-7 (2008), pp. 305–324.

[125] L. A. Schwarz et al. "Human skeleton tracking from depth data using geodesic distances and optical flow". In: *Image and Vision Computing* 30.3 (2012), pp. 217–226.

[126] E. Semsar-Kazerooni and K. Khorasani. "Multi-agent team cooperation: A game theory approach". In: *Automatica* 45.10 (2009), pp. 2205–2213.

[127] E. A. Sisbot et al. "Synthesizing robot motions adapted to human presence". In: *International Journal of Social Robotics* 2.3 (2010), pp. 329–343.

[128] P. Spellucci. "A new technique for inconsistent QP problems in the SQP method". In: *Mathematical Methods of Operations Research* 47.3 (1998), pp. 355–400.

[129] R. G. Strongin and Y. D. Sergeyev. *Global optimization with non-convex constraints: Sequential and parallel algorithms*. Vol. 45. Springer Science & Business Media, 2013.

[130] L. Sun et al. "A fast integrated planning and control framework for autonomous driving". In: *arXiv:1707.02515*. 2017.

[131] S. Tachi and K. Komoriya. "Guide dog robot". In: *Autonomous Mobile Robots: Control, Planning, and Architecture* (1984), pp. 360–367.

[132] T. S. Tadele, T. J. d. Vries, and S. Stramigioli. "The safety of domestic robots: a survey of various safety-related publications". In: *IEEE Robotics and Automation Magazine* (2014), pp. 134–142.

[133] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Vol. 65. Springer Science & Business Media, 2002.

[134] *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*. 2014. URL: `http://doi.org/10.4271/J3016_201401`.

[135] A. L. Thomaz, C. Breazeal, et al. "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance". In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Vol. 6. 2006, pp. 1000–1005.

[136] K. Tone. "Revisions of constraint approximations in the successive QP method for nonlinear programming problems". In: *Mathematical Programming* 26.2 (1983), pp. 144–152.

[137] G. Tonietti, R. Schiavi, and A. Bicchi. "Design and control of a variable stiffness actuator for safe and fast physical human/robot interaction". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2005, pp. 526–531.

[138] P. Trautman and A. Krause. "Unfreezing the robot: Navigation in dense, interacting crowds". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 797–803.

[139] C.-S. Tsai, J.-S. Hu, and M. Tomizuka. "Ensuring Safety in Human-Robot Coexistence Environment". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, pp. 4191–4196.

[140] A. Turnwald et al. "Interactive navigation of humans from a game theoretic perspective". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, pp. 703–708.

[141] G. Ulusoy, F. Sivrikaya-Şerifoğlu, and Ü. Bilge. "A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles". In: *Computers and Operations Research* 24.4 (1997), pp. 335–351.

[142] *UR5 from Universal Robotics*. URL: `http://www.universal-robots.com/GB/Products.aspx`.

[143] C. Urmson et al. "Autonomous driving in urban environments: Boss and the urban challenge". In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466.

[144] K. G. Vamvoudakis et al. "Autonomy and machine intelligence in complex systems: A tutorial". In: *Proceedings of the American Control Conference (ACC)*. 2015, pp. 5062–5079.

[145] J. Van Den Berg, P. Abbeel, and K. Goldberg. "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 895–913.

[146] T. Van Zandt. "Interim Bayesian Nash equilibrium on universal type spaces for supermodular games". In: *Journal of Economic Theory* 145.1 (2010), pp. 249–263.

[147] *Volvo collision avoidance features: Initial results*. Tech. rep. 5. Highway Loss Data Institute, 2012.

[148] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2009.

[149] "Working with Robots: Our Friends Electric". In: *The Economist* (2013). URL: `http://www.economist.com/`.

[150] W. Xu et al. "A real-time motion planner with trajectory optimization for autonomous vehicles". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2012, pp. 2061–2067.

[151] K. Yamazaki et al. "Home-assistant robot for an aging society". In: *Proceedings of the IEEE* 100.8 (2012), pp. 2429–2441.

[152] C. Yang et al. "Human-like adaptation of force and impedance in stable and unstable interactions". In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 918–930.

[153] J. E. Young et al. "Toward acceptable domestic robots: Applying insights from social psychology". In: *International Journal of Social Robotics* 1.1 (2009), pp. 95–108.

[154] W. Zhang et al. "Real-time Kinematic Modeling and Prediction of Human Joint Motion in a Networked Rehabilitation System". In: *American Control Conference (ACC)*. 2015, pp. 5800–5805.

[155] Z. Zhu, E. Schmerling, and M. Pavone. "A convex optimization approach to smooth trajectories for motion planning with car-like robots". In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*. 2015, pp. 835–842.

[156] M. Zinn et al. "A new actuation approach for human friendly robot design". In: *The International Journal of Robotics Research* 23.4-5 (2004), pp. 379–398.