Title
Safe and Efficient Human-Robot Collaboration

Author
Cheng, Yujiao

Publication Date
2021

Peer reviewed|Thesis/dissertation

Safe and Efficient Human-Robot Collaboration

by

Yujiao Cheng

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 Roberto Horowitz
Professor Kameshwar Poolla

Fall 2021

The dissertation of Yujiao Cheng, titled Safe and Efficient Human-Robot Collaboration, is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Safe and Efficient Human-Robot Collaboration

Abstract

Safe and Efficient Human-Robot Collaboration

by

Yujiao Cheng

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Masayoshi Tomizuka, Chair

As the emphasis on manufacturing is shifting from mass production to mass customization, the demands for flexible automation keep increasing. Human-robot collaboration (HRC), as an effective and efficient way to enhance flexibility, has attracted lots of attention both in industry and academia in the past decade. These robots are called co-robots. The fundamental research question is how to ensure that co-robots operate efficiently and safely with human partners.

To achieve that, two problems should be addressed: 1) co-robots should know the human's future trajectory and avoid potential collisions to guarantee safety, and 2) co-robots should know the human's intentions and take corresponding actions to ensure task efficiency. Therefore, a robotic system is adopted, which reasons about human behavior and makes human-aware planning using the reasoning information. For reasoning about human behavior, a hierarchical probabilistic modeling method and two online adaptation algorithms are proposed for human plan recognition and human trajectory prediction. The hierarchical probabilistic modeling method explicitly utilizes the hierarchical behavior of the human and uses a pipeline to identify human intention through the trajectory, the trajectory type, the action, and finally the plan, which is explainable and data-efficient.Two online adaptation algorithms, the adaptable neural network and the adaptable lognormal method, are proposed for short-term and long-term trajectory prediction. These two adaptation algorithms enable the prediction models to online accommodate different human behaviors and to deal with the lack of human data. For short-term prediction, the adaptable neural network utilizes the recursive least square-parameter adaptation algorithm to online adapt the last layer of a neural network model, the prediction of which is employed to the real-time collision avoidance. For long-term prediction, the adaptable lognormal method uses an objective-based adaptation algorithm to online adapt the sigma-lognormal model, from which the duration of the whole trajectory is estimated and later will be used in the task planner that optimizes the task completion time. For making human-aware planning, a separation task planner is

proposed, which uses the knowledge of human intention and makes the robot execute actions that are parallel to the human actions, while minimizing the task completion time. Different from those methods that only consider time efficiency, this separation task planner additionally improves human satisfaction and avoids potential conflicts, which can be shown in the experiment results.

This dissertation is organized as follows. Chapter 1 introduces the proposed robotic system. Chapter 2 defines the hierarchical human behaviors. Chapter 3 illustrates the proposed human plan recognition method. Chapter 4 and chapter 5 propose two online adaptation algorithms for short-term and long-term human trajectory prediction. Chapter 6 discusses a human-aware task planner, and Chapter 7 concludes the dissertation.

To my family

Contents

List of Figures

List of Tables

Acknowledgments

Foremost, I would like to express my sincere gratitude to my Ph.D. advisor Professor Masayoshi Tomizuka, who has been an incredible mentor for his immense knowledge, great patience, insightful visions and intense enthusiasm. Professor Tomizuka's guidance helped me all the time through research and writing of this thesis. I could not have come this far without the tremendous help from Professor Tomizuka.

I genuinely thank Professor Roberto Horowitz, Professor Kameshwar Poolla, and Professor Anca Dragan for serving on my dissertation committee. Professor Horowitz also served as the chair on my qualifying exam committee. Meanwhile, I am thankful to Professor Mark Mueller and Professor Somayeh Sojoudi for serving on my qualifying exam committee.

I am very grateful to Dr. Liting Sun and Professor Changliu Liu, who gave me valuable guidance and huge support. Professor Changliu Liu helped me discover my research interest and Dr. Liting Sun offered many professional suggestions and insightful discussions. Moreover, they are passionate and persistent researchers, and I thank them for being such good models for me to follow.

Special thanks go to Berkeley Fellowship and FANUC Corporation, Japan, for sponsoring me for the works in this dissertation. Discussions with employees in FANUC Corporation inspired me a lot and strengthened my dissertation from an industrial perspective.

Besides, my thanks also go to members in the Mechanical System Control (MSC) laboratory. In the past five years, I received enormous helps from former MSC members: Dr. Wenjie Chen, Dr. Te Tang, Dr. Yongxiang Fan, Dr. Yu Zhao, Dr. Hsien-Chung Lin, Dr. Peng Cheng, Dr. Jianyu Chen, Dr. Zining Wang, Dr. Zhuo Xu, Dr. Jiachen Li, and Dr. Yeping Hu, who provided many valuable suggestions to me in my life and career. Additionally, I am also grateful to current colleagues in MSC lab and thank you all: Dr. Wei Zhan, Chen Tang, Shiyu Jin, Jessica Leu, Hengbo Ma, Changhao Wang, Xinghao Zhu, Yiyang Zhou, Huidong Gao, Lingfeng Sun, Zheng Wu, Jinning Li, Xiang Zhang, Wu-Te Yang, Catherine Faulkner, Saman Fahandezhsaadi, Qiyang Qian, Jen-Wei Wang and Keita Kobashi.

Last but not least, my deepest gratitude goes to my parents and sister for your love, support and encouragement.

Chapter 1

An Overview: a Robot System for Human-Robot Collaboration

1.1 Introduction

As the emphasis on manufacturing is shifting from mass production to mass customization, the demands for flexible automation keep increasing [20] [68]. Human-robot collaboration, as an effective and efficient way to enhance flexibility, has attracted lots of attention both in industry and academia in the past decade. These robots are called co-robots. As shown in Fig. 1.1, co-robots can take advantage of both manual manufacture and automation: on the one hand, the number of human labors can be decreased, human labor costs can be reduced, and the task efficiency and quality can be improved; on the other hand, the human's flexibility and adaptability can be introduced to automation, and humans can help conduct some manipulation that robots cannot do.

Collaboration between humans and intelligent robots can be categorized into three levels: 1) low-level collision avoidance, 2) middle-level efficient cooperation with task plan recognition and trajectory prediction, and 3) high-level automatic task assignments. Many researches have been conducted for the three levels. For example, in the first category, [7] [38] regard humans as moving obstacles and designed algorithms to let the robot avoid collisions with humans, and in the second category, [6] [88] [59] reason about human behavior and studied the human plan recognition and human trajectory prediction algorithms. In the third category, [22] studies the task assignment algorithms in peer-to-peer human-robot interaction where humans and robots work as partners. In this dissertation, the focuses are on the second and third categories. Robots reason about human behavior and plan their own actions accordingly, which is beneficial for safe and efficient human-robot collaboration, because:

- Robots predict the human trajectory so that they can plan their own trajectory to avoid potential collisions, which guarantees **safety**;

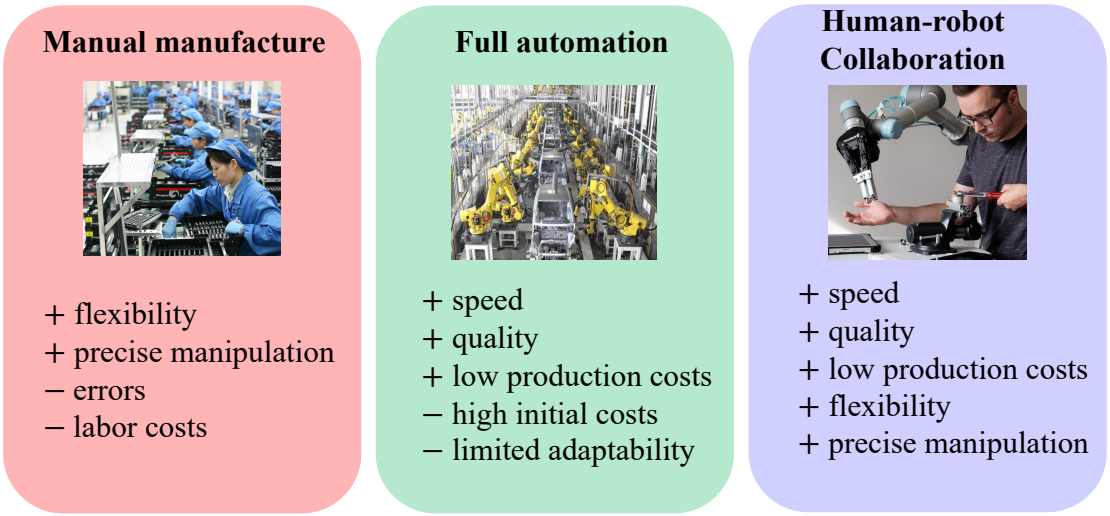


Figure 1.1: The demand for human-robot collaboration.

- Robots recognize the human plans, and they make their own plans accordingly, which boosts task **efficiency**.

1.2 A Robot System

To ensure safety and efficiency, an integrated human-robot collaboration framework is proposed, the architecture of which is shown in Fig. 1.2. It includes both an offline database and online modules. Online modules include a sensor module, a perception module, a human plan recognition module, a human trajectory prediction module, a task planner, a motion planner module and the actuators (the robot).

- *Sensor Module* uses a Microsoft Kinect sensor, which is a RGBD camera, to detect the environment and humans.
- *Perception Module* takes visual information as inputs and outputs the 3D positions of objects as well as the 3D human poses.
- *Human Plan Recognition Module* is a key module in our proposed framework. It aims to identify the action being executed by the human and infers human's plan by observing the trajectories of their key joints. The action estimate will be sent to both the planner module and the trajectory prediction module.
- *Task Planner Module* assigns the next action to the robot based on the current states and the recognized plan and the current action of the human. The action command from the planner is sent to the motion planner module.

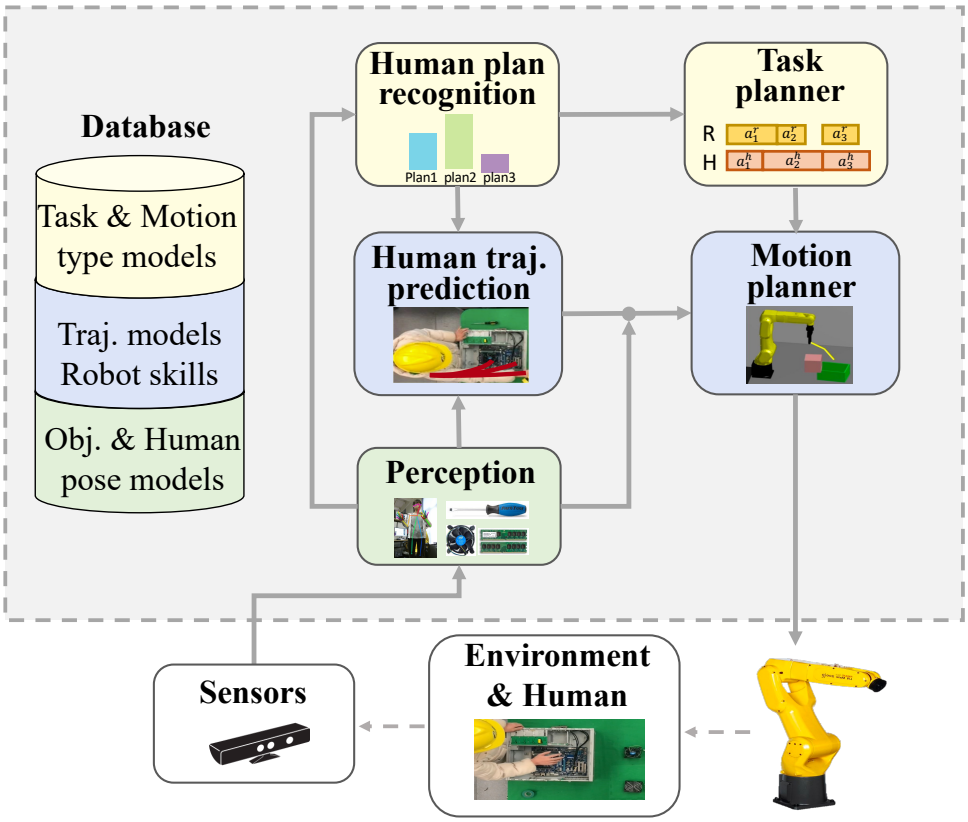


Figure 1.2: A robotic system for human-robot collaboration.

- *Human Trajectory Prediction Module* aims to predict the future trajectories of the human. Instead of directly predicting the future trajectories based on only current and historical human trajectories, we leverage the action labels from the plan recognition module.

- *Motion Planner Module* includes two controllers: an efficiency controller and a safety controller, as in [51]. The efficiency controller is a long-term global controller to assure the efficiency of the robot, and the safety controller is a short-term local controller to guarantee real-time safety under uncertainties.

This robot system aims to make safe and efficient human-robot collaboration. At the trajectory planning level, robots take the human future trajectory into account to avoid potential collisions, which improves safety. At the task planning level, robots perceive the human actions, infer the human plan, and adapt to human actions in advance, which boosts task efficiency. These two aspects can be illustrated in Fig. 1.3.

In the following chapters, the **human plan recognition** module, the **human trajectory prediction** module, and the **task planner** module will be discussed in detail.

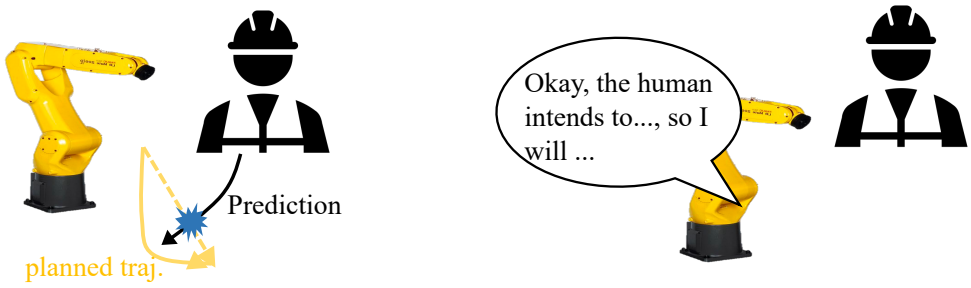


Figure 1.3: Safe and efficient human-robot collaboration. Left: the robot takes the human future trajectory into account to avoid potential collisions. Right: the robot infers the human plan and collaborates accordingly.

1.3 A Running Example Task

Figure 1.4: A running example task for the dissertation: a desktop assembly task.

Throughout the dissertation, a running example is utilized, which is a desktop assembly task. This task aims to assemble a desktop case. The procedure includes assembling the CPU fan, assembling memories, routing cables, assembling system fans, etc.. In the Foxconn factory, the whole assembly line, as shown in Fig. 1.4, is occupied by human labors. To decrease human labor costs and to free human labors from repetitive work, it is necessary to introduce robots to the assembly line to collaborate with humans. Therefore, this collaborative desktop assembly is studied.

1.4 Chapter Conclusion

This chapter displayed that human-robot collaboration arose as it could solve some problems that neither manual manufacturing nor full automation could solve. To guarantee human safety and to boost task efficiency, this chapter showed that it is important to do reasoning about human behavior, and to plan the robot actions using the reasoning results. A robot system was proposed for the safe and efficient human-robot collaboration, which featured a human plan recognition module, a human trajectory prediction module, and a human-aware task planner module. Besides, a running example task, i.e. a desktop assembly task, was illustrated.

Chapter 2

Hierarchical Human Behavior

2.1 Introduction

Human behavior is the potential and expressed capacity (mentally, physically, and socially) of human individuals or groups to respond to internal and external stimuli throughout their life [25]. For human-robot collaboration, some researchers focus on physical human motions, which can be captured by vision sensors [61][89], some researchers analyze humans by modeling human mental states [81][14], and some researchers study social human-robot interaction [8][2]. For us, we reason about human behavior at the physical level. Robots are designed to make safe and efficient collaboration taking account of human physical motions, while considering human mental comfort without explicitly modeling the human mental feelings.

It is hard to learn human physical motions. On the one hand, human motions are very complicated. They are highly nonlinear, time-varying, and stochastic [73]. On the other hand, there is not enough human data to learn the human motions. A lot of data-driven methods require a huge amount of data, while there is no existing database that can provide sufficient data for the assembly setting. Public databases, such as CMU Graphics Lab Motion Capture Database (Mocap) [1], captures human motions for daily activities, and they do not apply to our collaborative assembly task. If we collect human data by ourselves, it would be very time-consuming, and it will take a lot of effort. Therefore, the fundamental challenge of reasoning human behavior is **how to learn human behavior from limited data**. To cope with this challenge, two aspects are proposed:

- Defining hierarchical human behavior;
- Proposing adaptation algorithms for learning human behavior online.

First, instead of learning human behavior from end to end, i.e. learning the high-level human plan from image inputs, we define hierarchical human behavior and use hierarchical

[1]Repository URL: http://mocap.cs.cmu.edu/

modeling methods to reduce the dimension of the problem and reduce the data pressure. The definitions of hierarchical human behavior are explained in the next section, and the hierarchical modeling method is explained in Chapter 3. Second, from limited data, some patterns of human behavior may be missing and we could not learn a universal model offline, so online we continue to adjust our model to suit different human behavior by using adaptation algorithms. Two adaptation algorithms, the adaptable neural network and adaptable lognormal functions, are proposed, and they will be discussed in Chapter 4 and Chapter 5, respectively.

2.2 Definitions of Hierarchical Human Behavior

To understand human behavior, a set of hierarchical human behavior is defined as follows:

- **Trajectory**: A time series of the joint positions of the human in Cartesian space. It represents the continuous movements of an agent.
- **Trajectory Type**: A discrete variable/label to represent different types/patterns of trajectories. For instance, typical trajectory types in factory scenarios include "Fetching", "Screwing" and "Taping". Different trajectories can be generated to perform the same trajectory pattern.
- **Action**: A paired discrete variable/label including a trajectory type and a target object to act on, i.e., action = {trajectory type, object}. For example, we can define actions as "{fetching, screwdriver}" and "{taping, cables}".
- **Subtask**: An element of completing a larger *task* (defined below), whose initial states and the goal states do not depend on other subtasks. It might be implemented with several sequences of actions, depending on their orders.
- **Plan**: A sequence of ordered subtasks. It represents the preferences to finish a *task* (defined below). Different orders of actions in different plans come from either the orders of subtasks, or the orders of actions within subtasks.
- **Task**: The work to be conducted by agents. It specifies the initial states, the goal states and the participants. A task can be decomposed into a set of subtasks and executed via a variety of plans.

Fig. 2.1 illustrates the trajectory, the trajectory type, the action, and the plan in a desktop assembly example. Furthermore, Fig. 2.2 illustrates the hierarchical relationship of the task, the subtask, and the action. A task ("desktop assembly") can be decomposed into three subtasks ("installing a CPU fan", "installing a system fan", "taping cables"). Suppose each of them has a unique action order, then the permutation of three subtasks totally generates

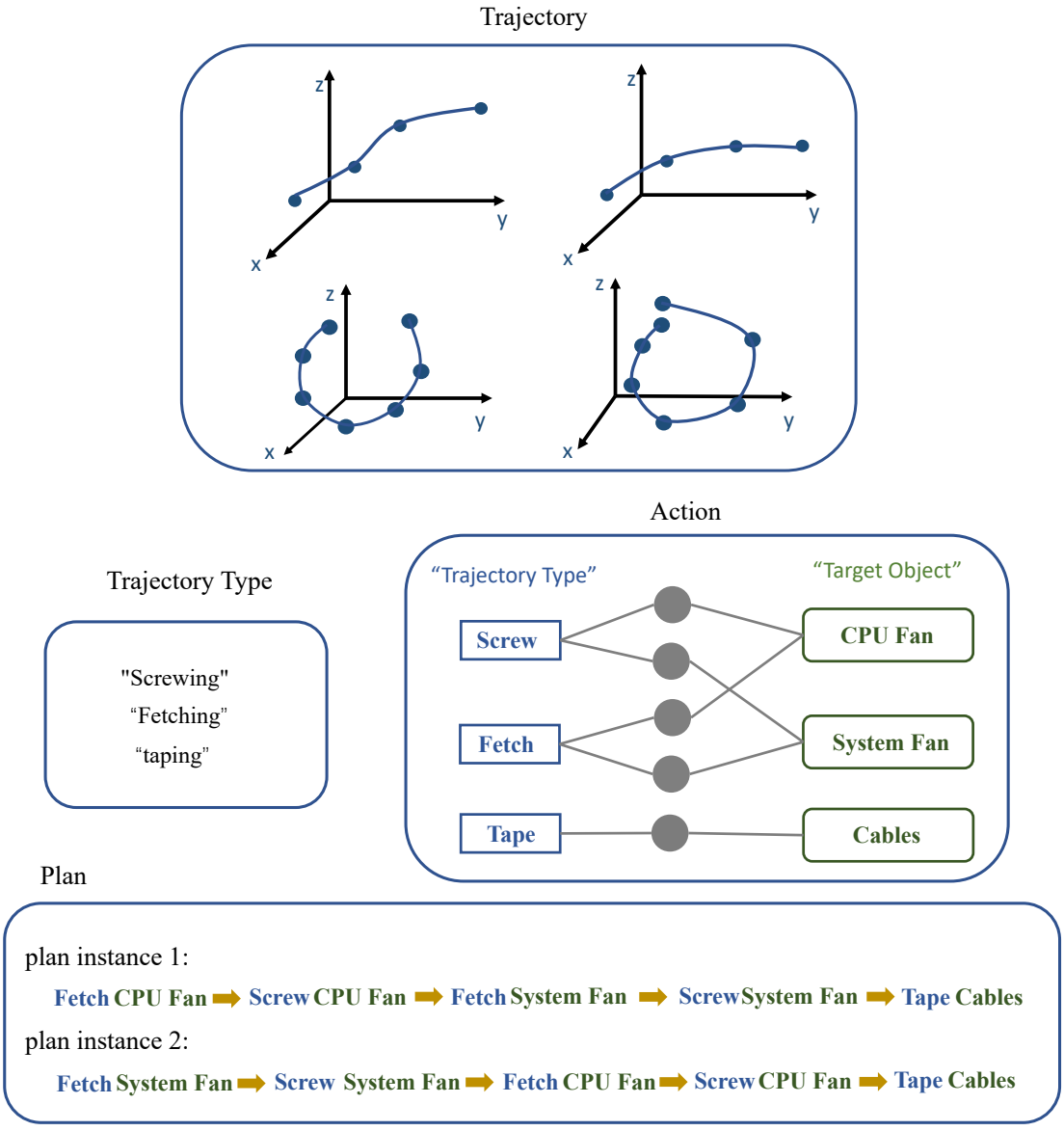
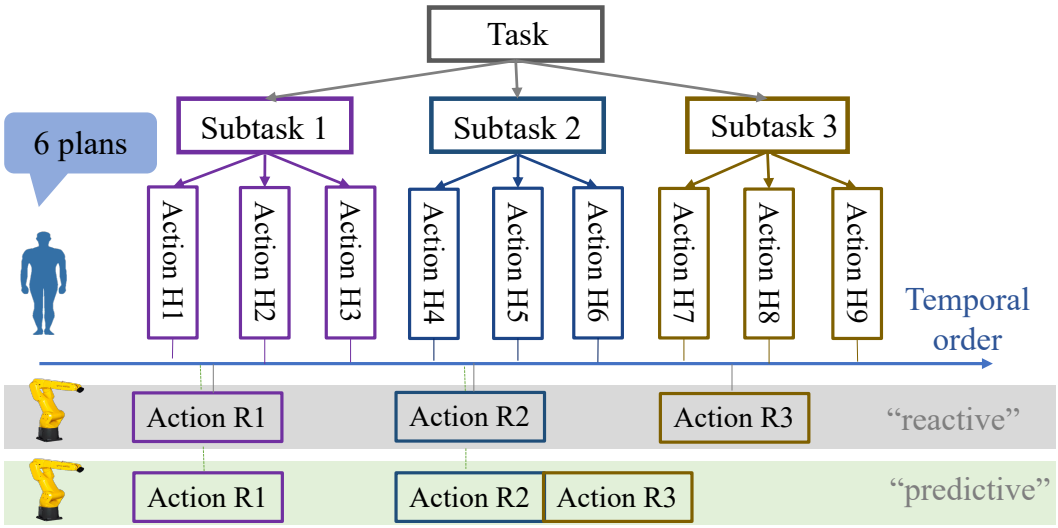


Figure 2.1: Examples for hierarchical human behavior.

six different plans, all of which are stored in the plan library as action[2] sequences of the human and the robot. Furthermore, within each action, the trajectory type can be executed by infinite many (theoretically) trajectories.

[2]In the desktop assembly example, Action H1-H9 are “fetching the CPU fan”, “receiving the screwdriver A”, “screwing the CPU fan”, “fetching the system fan”, “receiving the screwdriver B”, “screwing the system fan”, “taping the cables”, “receiving scissors”, and “cutting the tape”. Action R1-R3 are “delivering screwdriver A”, “delivering screwdriver B” and “delivering scissors”



Plan 1: Subtask 1-> Subtask 2-> Subtask 3; Plan 2: Subtask 1-> Subtask 3-> Subtask 2
Plan 3: Subtask 2-> Subtask 1-> Subtask 3; Plan 4: Subtask 2-> Subtask 3-> Subtask 1
Plan 5: Subtask 3-> Subtask 1-> Subtask 2; Plan 6: Subtask 3-> Subtask 2-> Subtask 1

Figure 2.2: A hierarchical and temporal decomposition of a task.

Trajectory prediction is to forecast the future movement of the human, thus the robot can make safe trajectory planning avoiding potential collisions. *Plan recognition* is to choose the correct plan in a human's mind, which is to choose the predefined action sequence in the plan library. As shown in Fig. 2.2, without plan recognition, the robot (the "reactive" robot) can only acquire its next action after the human finishes some key actions (such as Action H1, Action H4, and Action H7), while with a plan recognition, the robot (the "predictive" robot) can foresee the future actions of the human and execute its following actions in advance to boost the efficiency of the collaboration.

2.3 Chapter Conclusion

This chapter showed that human behavior is complicated, and learning complex human behavior requires a lot of data. To deal with the challenge of insufficient data, hierarchical human behavior was defined, and adaptation algorithms for learning human behavior online were proposed. This chapter explained and illustrated the definitions of hierarchical human behavior, which laid the foundation for the plan recognition and trajectory prediction in the later chapters.

Chapter 3

Human Plan Recognition

3.1 Introduction

Human plan recognition is to identify human intentions for plans. There are many ways that humans can communicate this intention. Some researchers study verbal conversation [69][62]. Robots can query the human plan by asking humans questions. Since human makes decisions on the fly, robots should constantly check in case that the human changes the plan. Thus, after some time, humans can be annoyed by repeated questions. Human gaze can also carry intention information [64][71], but such gaze communication really relies on the very fine detection of eye movement. Some researchers also study gestures [55][54]. Since humans have to use some specific gestures before every decision, it is highly possible that humans either forget from time to time or humans get exhausted physically and mentally. To release the communication burdens from humans, we let the humans do their own work, and we recognize the human plan based on the perceived human behavior.

To solve this problem, some existing works focus on deep learning frameworks with RGBD images as inputs [87] [67] [99]. Typically, the features selected mainly focus on humans, for instance, the body pose, hand positions, motion information and histogram of oriented gradients (HOG). No information about the objects of interaction is included. However, the objects can provide rich information for inferring what the human is doing via the intrinsic hierarchy among actions, trajectory types and the objects. Hence, in this chapter, we explore such hierarchy to design a more robust plan recognition algorithm based on the hierarchical human behavior defined in Chapter 2. Such a hierarchical modeling method decouples the problem of learning plans to learning the trajectory, the trajectory type, the action, and finally the plan. This can reduce the dimension of the problem and reduce the data pressure.

We evaluated our hierarchical modeling method through experiments and simulations. Experimental results showed that the average task completion time is significantly reduced, i.e., more efficient human-robot collaboration can be achieved. In addition, our system is robust with respect to noises in the model inputs and errors in the intermediate steps

such as trajectory type classification. We combine a long short-term memory network with algorithms based on Bayesian inference instead of end-to-end learning. This not only helps improve the robustness of the algorithm, but also reduces the dimension of the problem, and enhances its generalization ability using less data.

The main notation used in this chapter is summarized in Table 3.1

Table 3.1: Table of Notation.

k	$\in N$	Time step.
m	$\in \mathcal{N}$	Trajectory type.
n_m	$\in \mathcal{N}$	Number of trajectory types.
$\mathbf{h}$	$\in R^3$	Hand Position.
l	$\in \mathcal{N}$	Number of finger keypoints.
$\mathbf{w}$	$\in N^{2l}$	Finger keypoints.
o	$\in N$	Object label.
a	$\in N$	Action label.
g	$\in N$	Plan label.

3.2 A Hierarchical Modeling Method

By utilizing the definitions of the hierarchical human behavior, which is explained in Section 2.2, the pipeline for human plan recognition is proposed as shown in Fig. 3.1. From continuous human trajectories, the discrete trajectory types are first recognized via a long short-term memory (LSTM) network. Based on the trajectory types and the trajectories, target objects are estimated through Bayesian inference. After the target object is known, the human action is obtained by a combination of the trajectory type and the target object. By using the history of the action sequence, the plan is then recognized based on the dynamic time warping (DTW) algorithm. The upper figure of Fig. 3.1 shows the data flow, and the lower figure of Fig. 3.1 shows the proposed methods.

3.2.1 Trajectory Type Classification

Trajectory type classification aims to categorize different trajectories given segments of trajectories of the human's key joints. Long-short-term-memory (LSTM) neural networks have been extensively proved to be an effective approach to model the dynamics and dependencies in sequential data [57]. Hence, we design an LSTM recurrent neural network for trajectory type classification. The structure of the LSTM network is depicted in Fig. 3.2. The input data is the human pose from the *Perception* module. To be more specific, in an assembly task, the input vector at time step k is

$$\mathbf{x}_k = \{\mathbf{w}_k, \mathbf{h}_k\},$$

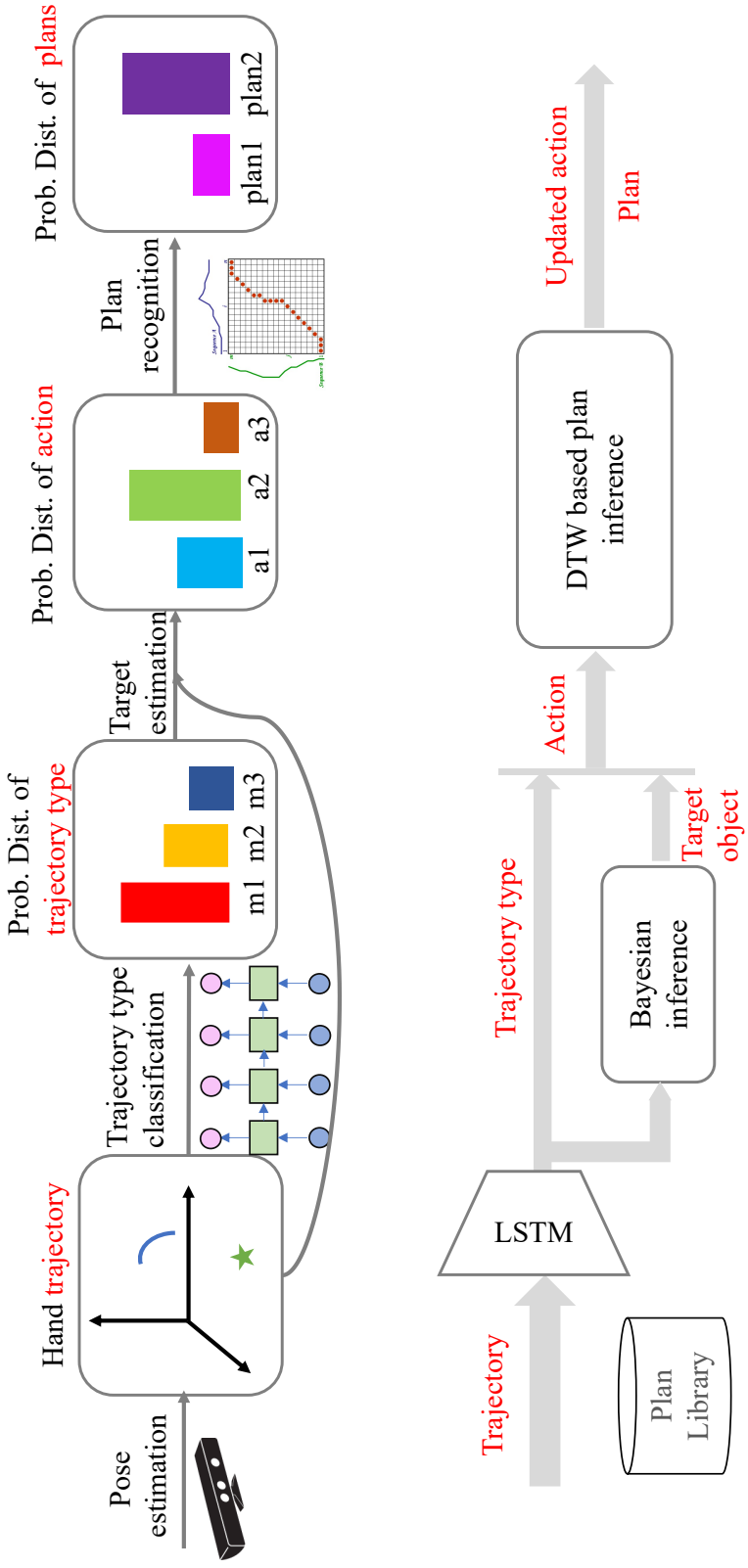


Figure 3.1: The architecture of the plan recognition module.

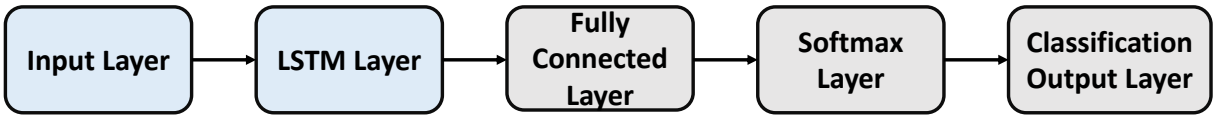


Figure 3.2: The LSTM network for trajectory type classification.

where $\mathbf{h}_k$ is the hand position in the world frame at time step k and $\mathbf{w}_k$ are the velocities of selected key points on the human fingers. The output at time step k is a trajectory type label $m_k \in \{1, 2, ...n_m\}$, where $n_m \in \mathbb{N}$ is the number of trajectory types. The LSTM is trained using the "Motion Model" database in Fig. 1.2.

3.2.2 Target Object Estimation

Given the classified trajectory type labels and a history of human pose, Bayesian inference is commonly used to update the beliefs on different target objects, e.g. [21]. Let o_k be an object at time step k, $\mathcal{O}$ be the object set, $m_{1:k}$ be the historical trajectory type labels, and $h_{1:k}$ be the historical human poses. Then we need to obtain the robot's beliefs on the object, i.e., a probability $P(o_k|h_{1:k}, m_{1:k})$. Applying the Markov assumption, the following equation holds:

$$
\begin{aligned}
&P(o_k|h_{1:k}, m_{1:k}) \propto \\
&P(m_k|o_k, h_{k-1}, m_{k-1}) \cdot P(h_k|o_k, h_{k-1}, m_k) \\
&\sum_{o_{k-1} \in \mathcal{O}} P(o_k|h_{k-1}, m_{k-1}, o_{k-1}) \cdot P(o_{k-1}|h_{1:k-1}, m_{1:k-1})
\end{aligned}
$$

We compute the $P(h_k|o_k, h_{k-1}, m_k)$ with an assumption that humans are optimizing some value function as [5] suggests. Then a Boltzmann policy can be applied:

$$P(h_k|o_k, h_{k-1}, m_k) \propto \exp(\beta V_g(h_k, o_k; m_k))$$

where V_g is the value function. We model V_g for each trajectory type as a function of distance and velocity.

To compute $P(m_k|o_k, h_{k-1}, m_{k-1})$ and $P(o_k|h_{k-1}, m_{k-1}, o_{k-1})$, we impose conditional independence assumption of m_k and h_{k-1} given o_k and m_{k-1}, and conditional independence assumption of o_k and h_{k-1} given o_{k-1} and m_{k-1}. Then, with predefined or learned models of $P(m_k|m_{k-1}, o_k)$ and $P(o_k|m_{k-1}, o_{k-1})$, $P(o_k|h_{1:k}, m_{1:k})$ can be updated iteratively.

3.2.3 Plan Inference

With results from trajectory type classification and object estimation, we can uniquely determine a sequence of actions by observing the human trajectories. Note that a plan is a sequence of subtasks, and each subtask is represented by one action or an ordered sequence of actions. Hence, a plan can be uniquely represented by a temporal sequence of actions. Therefore, we first build a plan library offline in the *Database* where each plan is represented by a reference sequence of actions. Then we utilize the reference sequences to online infer potential plans based on Bayes' rule,

$$P(g|a_{1:k}) \propto P(a_{1:k}|g)P(g),$$

where $P(g)$ is a prior belief of plan g, and $P(g|a_{1:k})$ is a posterior belief based on the likelihood of observed action sequence $a_{1:k}$ given plan g. Similarly, with Boltzmann policy, the likelihood of the action trajectory can be defined as

$$P(a_{1:k}|g) \propto \exp(-d(a_{1:k}; g)),$$

where the function d is a distance function measuring the similarity between the observed action sequence (A, namely $a_{1:k}$) and the reference action sequence (R) of the plan g. The larger the distance is, the less likely the human is following the plan [39]. We adopt the open-end dynamic time warping (OE-DTW) algorithm [91] to calculate d. This algorithm is to best match the query sequence to a reference sequence and to calculate the dissimilarity between the matched portion. Given a reference time series $R = (r_1, r_2, ..., r_N)$ and a query sequence $A = (a_1, a_2, ..., a_M)$, the OE-DTW distance between A and R is calculated via minimizing the dynamic time warping distances (DTW) between A and any references R^j truncated from reference R at point $j = 1 : N$.

$$D_{OE}(A, R) = \min_{j=1,...,N} D_{DTW}(A, R^j).$$

Here is a short introduction to DTW. The indices of the two series will be mapped through ϕ_t and ψ_t, $t = 1, 2, ..., T$, that satisfy the following constraints [91]:

- Boundary condition: $\phi_1 = 1$, $\psi_1 = 1$ and $\phi_T = N$, $\psi_T = M$
- Monotonic conditions: $\phi_{t-1} \leq \phi_t$ and $\psi_{t-1} \leq \psi_t$
- Continuity conditions: $\phi_t - \phi_{t-1} \leq 1$, and $\psi_t - \psi_{t-1} \leq 1$
- Local slope constraints: certain step patterns are allowed.

The optimal $\hat{\Phi} = (\hat{\phi}_t, \hat{\psi}_t)$ minimizes the distance between the two warped time series:

$$(\hat{\phi}_t, \hat{\psi}_t) = arg \min_{\phi_t, \psi_t} \sum_{t=1}^{T} \frac{d(r_{\phi_t}, a_{\psi_t}) m_{t,\Phi}}{\sum_t m_{t,\Phi}},$$

where $d(\cdot,\cdot)$ is any distance function and $m_{t,\Phi}$ is a local weighting coefficient. Therefore, the dynamic time warping distance between A and R is

$$D_{DTW}(A,R) = \sum_{t=1}^{T} \frac{d(r_{\hat{\phi}_t}, a_{\hat{\psi}_t}) m_{t,\Phi}}{\sum_t m_{t,\Phi}}.$$

3.2.4 Posterior Action Correction

As we obtain the posterior estimate of the plan g^*, the best-matched reference sequence R^{*j^*} is also obtained. We correct the action label estimate a_k by retrieving the action in the best-matched reference plan as follows,

$$a_k^{post} = R^*(j^*) = r^*_{j^*}.$$

This step is of key importance to reduce the sensitivity of the learning models to noises so that the robustness of the plan recognition can be improved. The effectiveness of this step is verified in experiments.

3.3 Experiment and Simulation

3.3.1 A Desktop Assembly Scenario

We evaluate our proposed framework in a desktop assembly task in industrial settings. The task of the HRC team is to assemble a desktop (desktop assembly example explained in Section 1.3). This task can be decomposed into three un-ordered subtasks: installing a CPU fan, installing a system fan and taping cables. Each subtask is implemented by only one action sequence. Thus, by a hierarchical decomposition, as shown in Fig. 2.2, there are at most six different plans to finish the task. The robot is designed to assist the human by delivering necessary tools to the human as he/she needs.

What we expect to see in the experiments is that the robot can recognize the human plan correctly and respond to the human in a timely and proactive manner as shown in Fig. 2.2. A predictive robot with an effective plan recognition will recognize the human's plan in the second subtask, and proactively execute the following actions in sequence. For example, if the human is doing the first plan, namely, installing a CPU fan, installing a system fan, and finally taping the cables. We expect that the correct plan is inferred when the human is fetching the system fan and then the robot will execute the following actions ("delivering the screwdriver" and "delivering the scissors") in sequence.

3.3.2 The Robot System

As shown in Fig. 1.2, the output of the *Plan recognition* module is sent to the *Planner* to generate commands for the *Motion planner* module. With the identified plan and the

human action estimate, the *Task planner* module acquires the next action of the robot from the plan library and sets the goal states of the *Motion planner* module to generate safe and executable trajectories.

Since the plan recognition results are probabilistic, we need to design a decision-making mechanism to decide on the robot actions. There are two cases that we might encounter. The first one is that the probability of one plan is prominently higher than that of others. This gives the *Task planner* a clear idea about the plan the human is executing, and it can directly acquire all the following actions for the robot from the plan library. The other case is where there are two or more candidate plans with similar probabilities from the plan recognition algorithms. Under this situation, the *Task planner* will look at the two most likely plans, and find out whether the next action of the robot for each plan is the same. If the next action is the same, *Task planner* will directly let the robot execute it. Otherwise, the *Task planner* will wait and collect the human's information to clear out the confusion.

The *Task planner* also takes changes of a plan into consideration. If the robot's next action is not consistent with what the robot is doing, it will recover the current action and responsively adjust its action. For example, if the robot is delivering a screwdriver to the human, while suddenly the next action becomes bringing the scissors. The robot will put back the screwdriver if it already grabs it, and go to scissors immediately.

The pseudo-code for the workflow of our proposed system is presented as Algorithm 1.

Algorithm 1 Proposed HRC system

Input plan library $\mathcal{Q}$, motion models $\mathcal{M}$; trajectory prediction models f^*; object set $\mathcal{O}$

Init: RobotIsDoing = {}, NewHumanPose = {}, RobotActionBuffer={}
while true **do**
  NewHumanPose = getValidPoseFromPerception()
  if notEmpty(NewHumanPose) **then**
    record historical human joint trajectory $h_{1:k}$
    $m_k \leftarrow$ TrajectoryClassification($h_{1:k}$, $\mathcal{M}$)
    $o_k \leftarrow$ TargetObjectEstimation ($m_{1:k}$, $o_{1:k-1}$, $h_{1:k}$)
    $h_{k+1:k+M} \leftarrow$ TrajectoryPrediction($h_{1:k}$, f^*, m_k, o_k)
    obtain $a_k = \{m_k, o_k\}$
    generate action trajectory $a_{1:k}$
    $p(\mathbf{g}|a_{1:k})$, $a_k^{post} \leftarrow$ OEDTWPlanInference($\mathcal{Q}$, $a_{1:k}$)
    $\hat{g}^{[1]}$, $\hat{g}^{[2]} \leftarrow$ the best and second best plan estimates
    if $p(\hat{g}^{[1]}|a_{1:k})$ >Threshold **then**
      RobotActionBuffer$\leftarrow$ nextActionSequence($\hat{g}^{[1]}$,a_k^{post},$\mathcal{Q}$)
      action1 $\leftarrow$ nextAction($\hat{g}^{[1]}$,a_k^{post},$\mathcal{Q}$)
      action2 $\leftarrow$ nextAction($\hat{g}^{[2]}$,a_k^{post}, $\mathcal{Q}$)
      if action1==action2 **then**
        RobotActionBuffer$\leftarrow$action1
      end if
    end if

```
        if notEmpty(RobotActionBuffer) then
            if notEmpty(RobotIsDoing)& RobotActionBuffer{1}!=RobotIsDoing then
                recover what robot is doing
                RobotIsDoing ← RobotActionBuffer{1}
            end if
        end if
    end if
    Execute(RobotIsDoing, h_{k+1:k+M})
    if ActionExecutionFinished then
        if notEmpty(nextInBuffer) then
            RobotIsDoing ← nextInBuffer
        else
            RobotIsDoing = {}
        end if
    end if
end while
```

3.3.3 Experiment Design

Hypothesis

We evaluate the effectiveness of the proposed plan recognition by verifying the following three hypotheses.

- H1: Human plan recognition improves the efficiency of the HRC team.
- H2: The performance of the proposed pipeline for plan recognition is robust to noises or errors caused by some intermediate steps such as trajectory type classification.
- H3: The human subjects are more satisfied with our collaborative robots than with a responsive robot in terms of some criteria

Experiment setup

We test our system on an industrial robot FANUC LR Mate 200iD/7L. A Kinect V2 for windows is placed close to the table on which a robot arm and a human worker do the task together. Some necessary tools lie in the tool area and a CPU fan and a system fan are in the part area.

We conduct experiments with the human in the loop. Eight human subjects participate in the experiments.

Manipulated variables

To evaluate the effectiveness of the proposed framework, we manipulated two controlled variables in our experiments: *plan recognition schemes* and *trajectory prediction schemes*. Plan recognition schemes include:

- "plan recognition = 0": no plan recognition;
- "plan recognition = 1": recognition ground truths provided by human subjects;
- "plan recognition = 2": recognition results generated by the proposed algorithm.

When "plan recognition = 0", the robot is completely reactive, meaning that it receives the information of the human action after the human completes the action. The robot only starts to move once it detects the human's actions, and it can only collaborate with the human within subtasks. When "plan recognition = 1", the robot has perfect plan knowledge, and it moves based on the ground truths of human's actions and the plan. When "plan recognition = 2", the proposed algorithm will let the robot automatically identify the human's actions and infer about the potential plan, so that the human and the robot can collaborate across subtasks. For trajectory prediction schemes, they include:

- "trajectory prediction = 0": no predictions of human trajectory;
- "trajectory prediction = 1": prediction via our proposed method, which will be explained in Chapter 4.

By manipulating the two variables, we have six groups of experiments. Under each group, every human subject performs the task using any plan three times. Thus, there will be 24 trials in each experimental group and in total we collect 144 trials for all groups.

Dependent measures

- To quantify efficiency, we use a timer to keep track of the task completion time. The timer starts when a human subject starts to move, and ends when the task is finished.
- To quantify the plan recognition performance, we calculate the plan recognition accuracy and the action recognition accuracy which is an intermediate result of the plan recognition module. Plan and action recognition accuracy is the percentage of plan estimates and action estimates that conform to the true values labeled by human subjects. Notice that plan recognition takes place at every time step, and there might be multiple plan labels in the early phase. As long as the estimate is one of the labels, it is regarded as correct.
- To quantify human's satisfaction with our collaborative robots, we ask the eight human subjects to rate the following statements on Likert scale from 1 (strongly disagree) to 5 (strongly agree), similar to [44]:

– The robot was collaborative and helped;
– The robot did the right thing at the tight time;
– I am satisfied working with the robot;
– I will work with this robot again in the future.

3.3.4 Implementation Details

The hardware configuration of the robot system is shown in Fig. 3.3. A workstation PC, a host PC, a target PC and a FANUC LR Mate 200iD/7L is connected via cables and networks. The workstation PC has a high image processing power with four EVGA GeForce GTX 1080 Ti GPUs. The perception module, the trajectory prediction module, and plan recongition module are implemented in this workstation. The planner modules, i.e. the trajectory planner and the task planner, are implemented in the host PC. The target PC receives signals from the host PC and controls the robot to move. For software

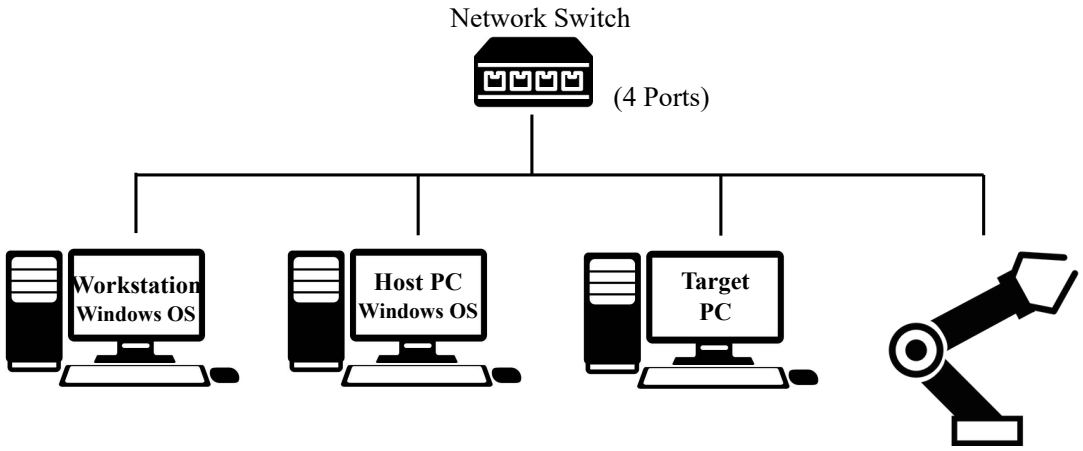


Figure 3.3: Hardware configuration of the system.

implementation, the input feature of the trajectory type classifier is $\mathbf{x} = \{\mathbf{w}, \mathbf{h}\} \in N^{42} \times R^3$, where $\mathbf{h} \in R^3$, is the right hand position of the human in the world frame, and $\mathbf{w} \in N^{42}$ is the coordinates of 21 keypoints [85] of the fingers of the right hand in the image frame. The LSTM layer of the trajectory type classifier has 50 hidden units, and the fully connected layer has 3 units, which is the number of the trajectory types ("fetching", "screwing" and "winding"). To train the LSTM newtork, the MATLAB deep learning toolbox is used. The batch size is set to 10, the initial learning rate is set to 0.01, and the maximum epoches is set to 15. For target object estimation, the object set is {CPU fan, system fan, tape}. The $P(m_k|m_{k-1}, o_k)$ and $P(o_k|m_{k-1}, o_{k-1})$ are predefined as the illustrated in the left and right subfigure of Fig. 3.4 respectively. σ is set to 0.01. The value functions of the Boltzmann policy is cosine distance of the human hand movement vector and the displacement vector

between the human hand and the object for "fetching" , sine distance of the human hand movement vector and the displacement vector between the human hand and the object for "winding" for constant distance for "screwing" .

m_k / m_{k+1}	"Fetch"	"screw"	"wind"
"fetch"	1−σ	1−σ	σ
"screw"	σ	σ	σ
"wind"	1−σ	σ	1−σ
"fetch"	1−σ	1−σ	σ
"screw"	σ	σ	σ
"wind"	1−σ	σ	1−σ
"fetch"	1−σ	σ	1−σ
"screw"	1−σ	1−σ	σ
"wind"	σ	σ	σ

o_{k-1} / o_k	Cpu fan	System fan	tape
Cpu fan	1−σ	σ	σ
System fan	σ	1 −σ	σ
tape	σ	σ	1−σ
Cpu fan	1−σ	σ	σ
System fan	1−σ	σ	σ
tape	1−σ	σ	σ
Cpu fan	σ	σ	1−σ
System fan	σ	σ	1−σ
tape	σ	σ	1−σ

o_k CPU fan System fan tape m_{k-1} "fetch" "screw" "wind"

Figure 3.4: Predefined probability matrices.

3.3.5 Results

H1: Human plan recognition improves the efficiency of the HRC team. The task completion time for different plan recognition schemes was recorded among trials with the eight human subjects. As shown in Fig. 3.5, without plan recognition ("plan recognition = 0"), the average task completion time is $90.0 \pm 10.9s$, which is the longest. With our proposed plan recognition algorithm ("plan recognition = 2"), the average task completion time is $64.6 \pm 10.6s$, which is reduced by 29.1%. Thus, the proposed framework with plan recognition significantly improves ($p < 0.01$) the efficiency of the HRC team compared to the system without the plan recognition. As a matter of fact, our system can achieve similar performance as a system with perfect plan recognition ("plan recognition = 1") with $1.2s$ more average task completion time and larger variance.

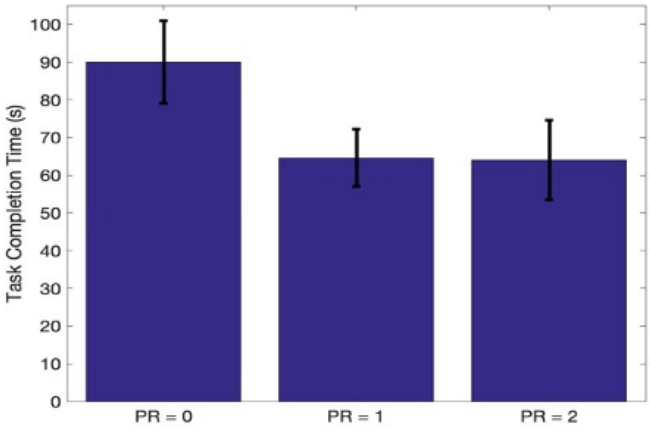


Figure 3.5: Task completion time fro different plan recognition schemes. "plan recognition = 0" means no plan recognition, "plan recognition = 1" means recognition ground truths provided by human subjects, and "plan recognition = 2" means the recognition results generated by the proposed algorithm

Table 3.2: Quantitative experimental results for recognition. TT means trajectory type, and PR means plan recognition.

Subjects	**TT accuracy(%)**	**PR accuracy (%)**
Subject 1 (6 trials)	85.3 ±1.3	97.6 ±0.3
Subject 2 (6 trials)	69.6 ±15	98.5 ±0.5
Subject 3 (6 trials)	81.4 ±1.1	97.4 ±1.2
Subject 4 (6 trials)	87.9 ±7.0	97.6 ±1.1
Subject 5 (6 trials)	79.4 ±6.5	96.4 ±1.8
Subject 6 (6 trials)	80.7 ±12.0	97.9 ±0.1
Subject 7 (6 trials)	84.8 ±9.5	97.6 ±0.6
Subject 8 (6 trials)	85.8 ±8.8	90.5 ±0.3

H2: The performance of the proposed pipeline for plan recognition is robust to noises or errors caused by some intermediate steps such as trajectory type classification. This hypothesis can be proved via quantitative results in Table 3.2. One can see that although some trajectory type classification accuracy is low[1], the plan recognition accuracy still remains high. The Pearson product-moment correlation coefficient for the two variables is $-0.11 (p < 0.01)$, which indicates a weak correlation. This means that the

[1]The online trajectory type classification deteriorated mainly because there were a lot of random or transition movements during the experiments that do not belong to any class of the four trajectory types.

overall performance of the proposed plan recognition algorithm is not sensitive to the errors in the intermediate LSTM step. This is mainly benefiting from the Bayesian inference step and the dynamic time warping step. These two steps serve as a low pass filter, eliminating the wrong trajectory type estimates. Besides, the plan is actually estimated by the nearest neighbor in DTW step, and the six plans as action sequences lie sparsely in an increasingly high dimensional space, and they get farther away from each other over time. As long as the estimates do not deviate from the true point too much, the plan estimate should be correct.

To further validate the robustness brought by the DTW step, we assumed that the target object detection was perfect and did simulations by varying the trajectory type classification (MC) performance and then tested the plan recognition (PR) accuracy. First, we obtained the true positive rates for each trajectory type throughout all the experiments with "plan recognition = 2": 83.4% for "screwing", 64.2% for "fetching", 59.1% for "receiving", and 84.2% for "taping". Then we varied each true positive rate by $\delta(\%)$, which took values of $0, -5, -10, -15, -20, -30, -40$, and -45. Based on these sets of true positive rates, we simulated 15 trials of action sequences for each of the six plans in the desktop assembly task, and so we had 90 trials for each δ. As we can see in the Fig 3.6, when δ is equal to -30 (true positive rate for each trajectory type is $53.4\%, 24.2\%, 29.1\%, 54.2\%$) and the overall trajectory type classification is 30%, the plan recognition accuracy remains higher than 85%, which shows robustness of our plan recognition to the trajectory type accuracy.

H3: The human subjects are more satisfied with our collaborative robots than with a responsive robot in terms of some criteria. Fig. 6.9 shows the comparison of human subjects' ratings for the six types of robots on four criteria mentioned above. Human subjects rated the robot with our proposed plan recognition algorithm ("plan recognition = 2") significantly higher ($p < 0.01$) than the robot without plan recognition ("plan recognition = 0") on all four criteria. Between robots with ground truths of plan recognition ("plan recognition = 1") and robots with our plan recognition algorithm ("plan recognition = 2"), there is no significant difference ($p > 0.05$) on all the criteria except for the criteria "The robot did the right things at the right time" ($p = 0.04$). Furthermore, there is also no significant difference ($p > 0.05$) between the robots with trajectory prediction ("trajectory prediction" = 1) and the robots without trajectory prediction ("trajectory prediction" = 0). This might be because the trajectory prediction is too short to influence the human's feedback. Recalling the fact that "plan recognition" has a significant influence, we can see that humans care more about efficient plan recognition.

Aside: The sensitivity of the threshold in Algorithm 1 is not obvious. We designed a new experiment by varying the threshold value in Algorithm 1 on the experiment data we collected. It was found that the plan recognition results remained the same when the threshold was dropped from 0.70 to 0.58, where 0.70 is the threshold we used in other experimental results. Therefore, the sensitivity of threshold in Algorithm 1 is not obvious. To further study the sensitivity of the threshold requires more experiments in the future.

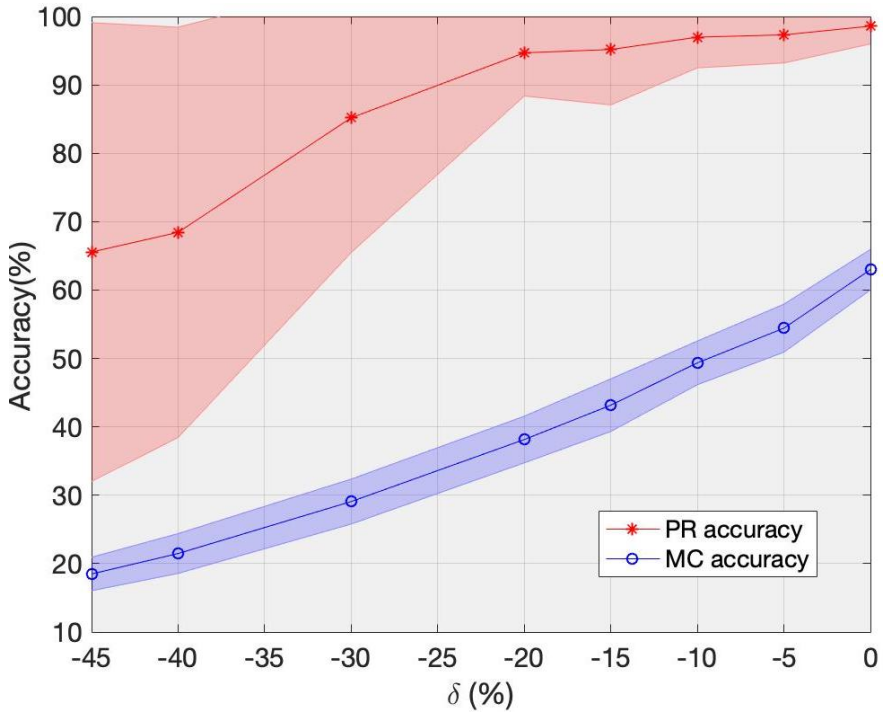


Figure 3.6: Simulations for plan recognition accuracy when reducing the trajectory type classification accuracy by δ. MC represents trajectory type classification, and PR represents plan recognition.

3.3.6 Extension to Multiple Tasks

Table 3.3: The result table for the three tasks.

Tasks	**Precision**		**Recall**		**$F_{0.5}$**	
	ours	MEMM	ours	MEMM	ours	MEMM
Drinking water	97.0	87.9	95.4	80.8	96.7	86.4
Cooking (stirring)	88.9	65.5	98.4	43.9	90.6	59.7
Opening pill container	84.3	86.4	87.0	58.0	84.4	78.7

Our plan recognition algorithm also works to distinguish different tasks. We test our algorithm on three different tasks in the CAD-60 Cornell Activity Dataset [87]: cooking (stir), opening a pill container and drinking water. Table 3.3 shows the comparison of the results in the "new person" setting using our algorithm and the two-layered maximum entropy Markov model (MEMM) method in [87]. We can see that our plan recognition algorithm can also achieve very high accuracy compared to the approach in [87]. This

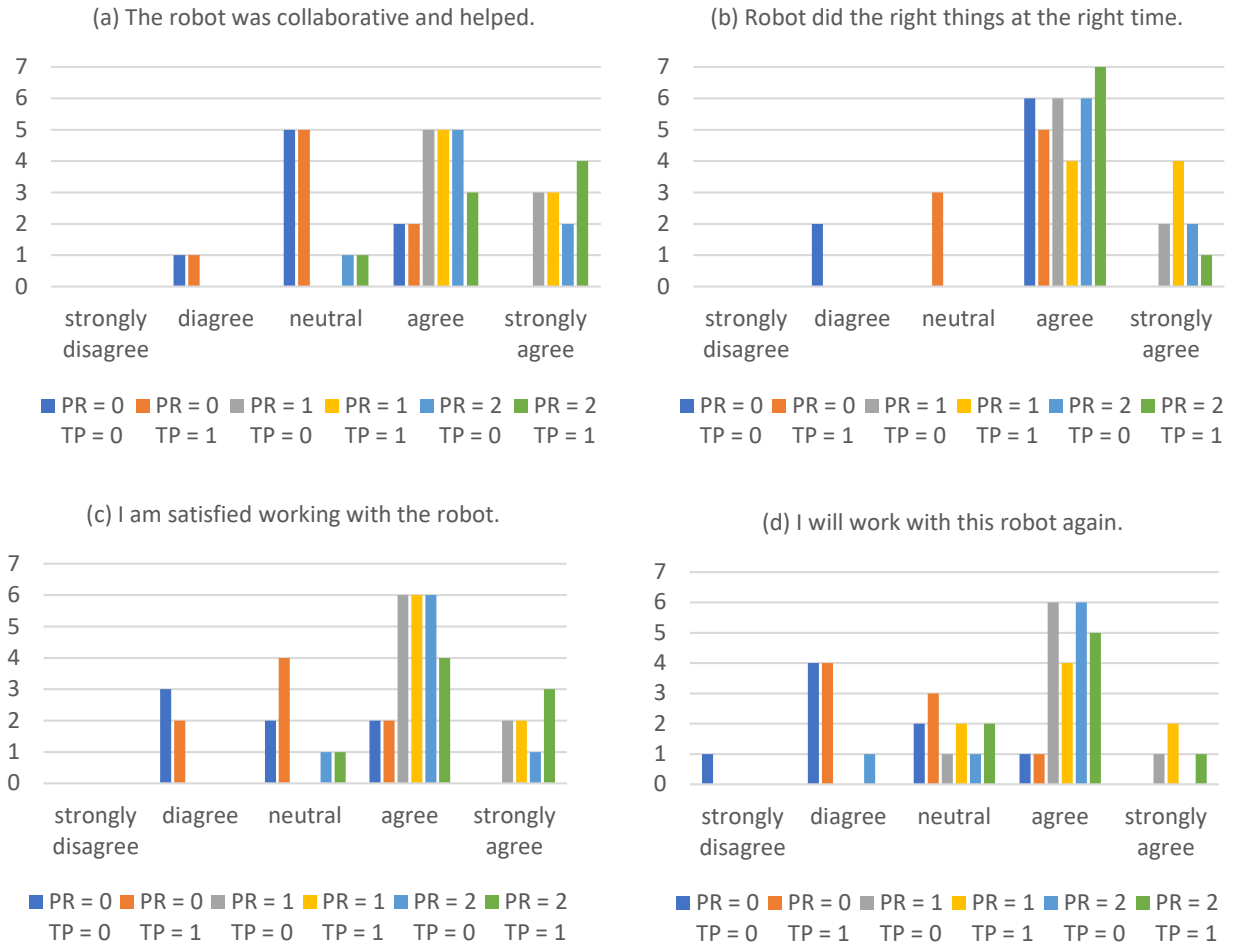


Figure 3.7: Human subjects' ratings for six types of robots on four different criteria. PR is short for plan recognition and TP is short for trajectory prediction.

verifies our claim that exploiting the rich object information can help improve the task/plan recognition performance.

In addition, compared to end-to-end learning, our algorithm advantages in two aspects:

- The learning process is easier, since a hierarchical combination of trajectory type classification and target estimation reduces the dimension of the classification problem;
- The learning pipeline is more interpretable and predictable.

As a cost, however, the proposed method requires stronger prior knowledge, i.e., all possible plans of the new task should be predefined offline, which might be hard when the task is complicated.

3.4 Change of Plans

It is possible that humans change the plans during operations. By using our plan recognition method, the robot can quickly recognize the human's new plan, and adapt its actions accordingly. We evaluate this point by an experiment. Fig. 3.8 shows several clips of the video[2] captured when conducting this experiment. The human intends to assemble the CPU fan at first. While the human is reaching for the CPU fan, the robot recognizes this intention and moves to the screwdriver, which is the tool needed for the subtask of assembling CPU fan. However, the human then changes to tape the cables, when the robot has already picked up the screwdriver. The robot identifies that the human changes the plan, so it releases the screwdriver and goes to scissors, which is the tool needed for the subtask of taping the cables. Therefore, by using our algorithm, the robot detects the change of the plan correctly.

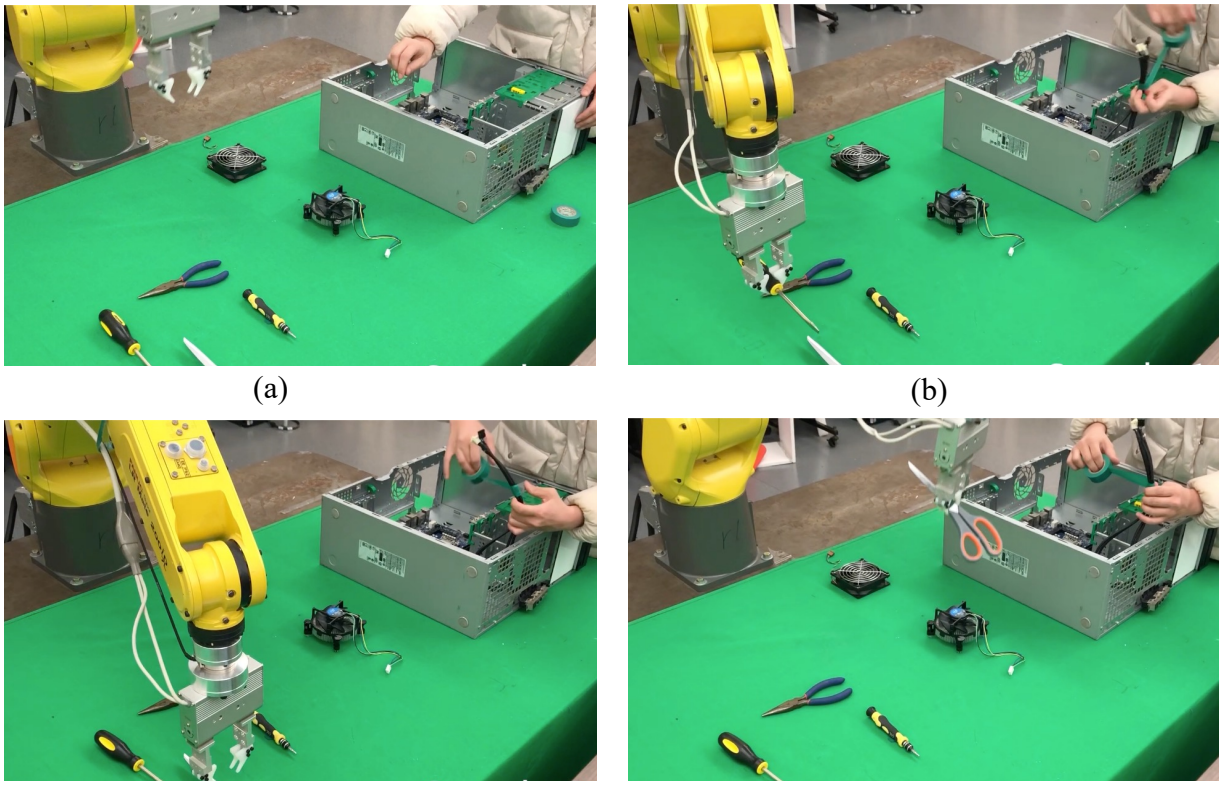


Figure 3.8: The robot's response to the human's change of plans. (a) The human goes to the CPU fan, and the robot goes to the screwdriver. (b) While the robot is picking up the screwdriver, the human changes to tape the cables. (c) The robot releases the screwdriver and goes to pick up the scissors. (d) The robot delivers the scissors.

[2]The video of this experiment can be found in https://www.youtube.com/watch?v=4DlgnFjfwkY.

3.5 Chapter Conclusion

In this chapter, we designed a robust plan recognition algorithm based on neural networks and Bayesian inference. by explicitly leveraging the hierarchical relationships among plans, actions and trajectories. Experiments with humans in the loop were conducted on a desktop assembly task. The results showed that with our proposed framework, the average task completion time was reduced by 29.1%. Moreover, the proposed plan recognition algorithm was robust and reliable. High plan recognition accuracy was achieved even when the trajectory type labels via neural networks were of low accuracy. We verified the effectiveness of the proposed algorithms on both a designed desktop assembly experiment and the CAD60 dataset compared with another human activity recognition approach.

Chapter 4

Short-Term Human Trajectory Prediction

4.1 Introduction

Smooth interactions among intelligent entities depend on a clear understanding of what the others would do in various circumstances. For example, soccer players predict the motions of their teammates for better cooperation; pedestrians have a notion of where others are going so as to avoid collisions. Similarly, robots that interact in proximity with humans are required to know what the human is going to do in the near future. The benefits are that, based on the predictions, the robot can plan collision-free trajectories to assure the human's safety and schedule their actions in advance to improve task efficiency. Human motion prediction has applications well beyond human-robot interaction [98] [42] [23], it also plays a key role in computer vision. Adequate prediction of human motion can facilitate 3D people recognition and tracking [33], motion generation in computer graphics (CG) [45], and psychology biological motion modeling [92].

However, human motion is inherently difficult to predict due to the nonlinearity and stochastic in human behavior [73]. In addition, individual differences are also prominent. Prediction models that work for one person may not apply to another.

Early attempts have been made to predict human motion using Kalman filter and particle filter [43] [12], where the problem is posed as a tracking problem. Another category of approaches assumes that human is rational with respect to certain cost functions. Human trajectories can then be predicted by optimizing the cost function [37]. The difficulty of this method is that the cost functions of humans are hard to obtain due to stochasticity and complexity in human intention. Another domain of work prominently focuses on latent variable based probabilistic models. Wu et al. [97] use hidden Markov models (HMMs) combined with multi-layer perceptrons to model the evolution patterns of motion trajectory.

Similar to HMMs, recurrent neural networks (RNNs) have distributed hidden states to store information about the past, and many works on RNNs have obtained big success on hu-

man motion prediction [29] [1], but they still suffer from several problems. The first problem is that RNNs are hard to train. Heroic efforts of many years still fail to accelerate the training speed of RNNs. The second problem is that predictions from RNNs are deterministic, which is not satisfactory in human-robot interaction, since the robot needs the uncertainty level of human's future motion for safe motion planning. The last serious problem is that the RNN models are fixed and they cannot adapt to time-varying human behaviors.

We aim to solve these problems by proposing a semi-adaptable neural network. To be specific, a neural network is trained offline to represent the human motion transition model, and then a recursive least square parameter adaptation algorithm (RLS-PAA) is adopted for online parameter adaptation of the last layer in the neural network and for uncertainty estimation. The proposed method advantages human motion prediction in three aspects. First, it is computationally more efficient to use a feedforward neural network than to use a RNN for approximation of the human transition model. In the meanwhile, the mechanism for adaptable feedforward neural networks is equally applicable to adaptable RNNs. Second, it adapts the model to time-varying behaviors and individual differences in human motion, which yields more accurate predictions. Third, it computes the uncertainty level of the predictions, which is important for the safe motion planning of robots. To verify the effectiveness of our adaptation scheme, we compare our method with the state-of-the-art online learning algorithm called the identifier-based algorithm [78]. The identifier-based algorithm adapts all the parameters in the offline-trained neural network model online, using gradient descent to minimize the prediction error. Results demonstrate that our method achieves a higher prediction accuracy, and the performance is maintained across a variety of motion categories and motion datasets.

The main notation used in this chapter is summarized in Table 4.1

Table 4.1: Table of Notation.

M	$\in \mathcal{N}$	Prediction horizon.
N	$\in \mathcal{N}$	Past horizon.
$\mathbf{x}$	$\in R^{3M}$	Future M-step human hand position.
$\mathbf{x}^*$	$\in R^{3N}$	Past N-step human hand position.
a	$\in N$	Action.
k	$\in N$	Time step.

4.2 Problem Formulation

Predicting human motion is important for smooth human-robot interaction, because first, if the robot knows what the human is going to do, it can adapt its actions to collaborate with humans in an efficient way, and second, plan collision-free trajectories to guarantee human safety [53]. This work concerns the prediction of one human joint (e.g., wrist), which is reasonable because when a human works in close proximity to a robot, special attention

should be paid to the movement of a human's hand. Moreover, one joint motion prediction is extendable to that of multiple joints, which will be discussed in Section 5.5.

The transition model of human joint motion is formulated as

$$\mathbf{x}(k+1) = f^*(\mathbf{x}^*(k), a) + w_k, \tag{4.1}$$

where $\mathbf{x}(k+1) \in \mathbb{R}^{3M}$ denotes human's M-step positions of the joint at future time steps $k+1, k+2, \ldots, k+M$ in a Cartesian coordinate system. $M \in \mathbb{N}$ is the prediction horizon. Denoting the Cartesian position of the joint at time step k by $p(k) \in \mathbb{R}^3$, $\mathbf{x}(k+1)$ is obtained by stacking $p(k+1), p(k+2), ..., p(k+M)$. $\mathbf{x}^*(k) \in \mathbb{R}^{3N}$ denotes human's past N-step positions of the joint. It is also constructed by stacking the position vectors $p(k), p(k-1), ..., p(k-N+1)$. $a \in \mathbb{N}$ is an action label to distinguish different motions, and this label is obtained by the action recognition module of the system [53]. $w_k \in \mathbb{R}^{3M}$ is a zero-mean white Gaussian noise. The function $f^*(\mathbf{x}^*(k), a) : \mathbb{R}^{3N} \times \mathbb{N}^1 \to \mathbb{R}^{3M}$ represents the transition of the human motion, which takes historical trajectory and current action label as inputs, and outputs the future positions of the joint, which is illustrated in Fig. 4.1.

Since human behavior differs greatly across individuals and is highly time-varying, function f^* may not be a time-invariant function. Though f^* takes the discrete parameter action label a as one of the inputs to accommodate some of the variances, an adaptable model of f^* is still desired to account for continuous changes online in order to provide accurate prediction.

4.3 Semi-adaptable Neural Networks

Since human motion is not only time-varying but also highly nonlinear, we propose to use a neural network to construct the model f^*, because neural networks have good model capacity. To make it adaptable, notice that if we remove the last layer, the pre-trained neural network becomes an effective feature extractor [4], the features from which are better than handcrafted ones [84]. Therefore we only adapt the weights of the output layer of the neural network online, which fixes the weights of the remaining layers, hence fixing the extracted features. This approach combines offline training and online adaptation for function approximation, which takes advantages of both methods. Offline training of the neural network extracts global features from the data. Online adaptation changes the way that the global features are combined by adjusting the coefficients in the last layer of the network locally to minimize real-time prediction error. The proposed procedure is as follows.

1. We first design a neural network architecture;
2. We train the model f^* offline;
3. During online execution, we adapt the parameters of the last layer of the neural network using an efficient adaptation algorithm;

4. We then compute the uncertainty level of predictions given the adaptation result.

The algorithm is shown in Algorithm 2.

4.3.1 Training the Neural Network

To train the transition model f^*, we choose an n-layer neural network with ReLU activation function which takes the positive part of the input to a neuron.

$$f^*(\mathbf{x}^*(k), a) = W^T \max(0, g(U, s_k)) + \epsilon(s_k), \tag{4.2}$$

where $s_k = [\mathbf{x}^*(k)^T, a, 1]^T \in \mathbb{R}^{3N+2}$ is the input vector, g denotes $(n-1)$ - layer neural network, whose weights are packed in U. $\epsilon(s_k) \in \mathbb{R}^{3M}$ is the function reconstruction error, which goes to zero when the neural network is fully trained. $W \in \mathbb{R}^{n_h \times 3M}$ is the weights of the last layer, where $n_h \in \mathbb{N}$ is the number of neurons in the hidden layer of the neural network [78].

4.3.2 Parameter Adaptation Algorithm

To accommodate both the time-varying behavior of human and individual differences among different people, it is important to update the parameters online. In this work, we applied the recursive least square parameter adaptation algorithm (RLS-PAA) with forgetting factor [31] to asymptotically adapt the parameters in the neural network.

By stacking all the column vectors of W, we get a time varying vector $\theta \in \mathbb{R}^{3Mn_h}$ to represent the weights of last layer. θ_k denotes its value at time step k. To represent the

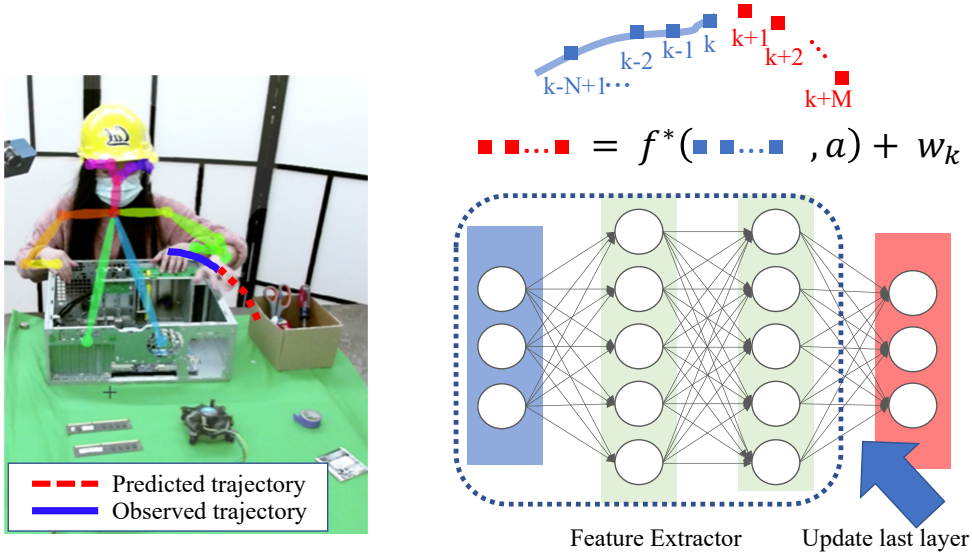


Figure 4.1: The adaptable neural network for human trajectory prediction.

Algorithm 2 Semi-adaptable neural network for human trajectory prediction

Input Offline trained neural network (4.2) with g, U and W

Output Future trajectory $\mathbf{x}(k+1)$

Variables Adaptation gain F, *a priori* mean squared estimation error of states $X_{\tilde{x}\tilde{x}}$, mean squared estimation error of the parameters $X_{\tilde{\theta}\tilde{\theta}}$, neural network last layer parameters θ, estimated rate of change $\delta\theta$ (approximation of $\Delta\theta$), variance of zero-mean white Gaussian noise $Var(w_k)$

1: **Init:** $F = 1000\mathcal{I}$, θ = column stack of W,$X_{\tilde{x}\tilde{x}} = \mathbf{0}, X_{\tilde{\theta}\tilde{\theta}} = \mathbf{0}$, $\lambda_1 = 0.998$, $\lambda_2 = 1$
2: **while** True **do**
3: Wait for a new joint position p captured by Kinect and current action label a from action recognition module
4: Construct $s_k = [p(k), p(k-1), ..., p(k-N+1), a, 1]^T$
5: Obtain $\Phi(k)$ by diagonal concatenation of $max(0, g(U, s_k))$
6: Update F by (4.7)
7: Adapt the parameters θ in last layer of neural network by (4.6)
8: Calculate future joint trajectory $x(k+1)$ by (4.3)
9: Update $\delta\theta$ and calculate $X_{\tilde{x}\tilde{x}}$ and $X_{\tilde{\theta}\tilde{\theta}}$ by (4.8) and (4.11)
10: send $x(k+1)$ and $X_{\tilde{x}\tilde{x}}$ to robot control.
11: **end while**

extracted features, we define a new data matrix $\Phi_k \in \mathbb{R}^{3M \times 3Mn_h}$ as a diagonal concatenation of M pieces of $\max(0, g(U, s_k))$. Using Φ_k and θ_k, (4.1) and (4.2) can be written as

$$\mathbf{x}(k+1) = \Phi_k \theta_k + w_k. \tag{4.3}$$

Let $\hat{\theta}_k$ denotes the parameter estimate at time step k, and let $\tilde{\theta}_k = \theta_k - \hat{\theta}_k$ be the parameter estimation error. We define the *a priori* estimate of the state and the estimation error as:

$$\hat{\mathbf{x}}\,(k+1|k) = \Phi_k \hat{\theta}_k, \tag{4.4}$$

$$\tilde{\mathbf{x}}\,(k+1|k) = \Phi_k \tilde{\theta}_k + w_k. \tag{4.5}$$

The core idea of RLS-PAA is to iteratively update the parameter estimation $\hat{\theta}_k$ and predict $\mathbf{x}(k+1)$ when new measurements become available. Note that $\mathbf{x}(k+1)$ is available after M steps. The parameter update rule of RLS-PAA can then be summarized as:

$$\hat{\theta}_{k+1} = \hat{\theta}_k + F_k \Phi_k^T \tilde{\mathbf{x}}\,(k+1-M|k-M)\,, \tag{4.6}$$

where F_k is the learning gain updated by:

$$F_{k+1} = \frac{1}{\lambda_1(k)}[F_k - \lambda_2(k)\frac{F_k \Phi_k \Phi_k^T F_k}{\lambda_1(k) + \lambda_2(k)\Phi_k^T F_k \Phi_k}] \tag{4.7}$$

where $0 < \lambda_1(k) \leq 1$ and $0 < \lambda_2(k) \leq 2$. Typical choices for $\lambda_1(k)$ and $\lambda_2(k)$ are:

1. $\lambda_1(k) = 1$ and $\lambda_2(k) = 1$ for standard typical least squares gain.
2. $0 < \lambda_1(k) < 1$ and $\lambda_2(k) = 1$ for least squares gain with forgetting factor.
3. $\lambda_1(k) = 1$ and $\lambda_2(k) = 0$ for constant adaptation gain.

4.3.3 Mean Squared Estimation Error Propagation

To guarantee safety, the uncertainty of the prediction is also quantified during online adaptation [52].

State estimation Note that $\hat{\theta}_k$ only contains information up to the $(k-1)$th time step, and $\tilde{\theta}_k$ is independent of w_k. Thus the *a priori* mean squared estimation error (MSEE) $X_{\tilde{x}\tilde{x}}(k+1|k) = E\left[\tilde{\mathbf{x}}(k+1|k)\,\tilde{\mathbf{x}}(k+1|k)^T\right]$ is

$$X_{\tilde{x}\tilde{x}}(k+1|k) = \Phi_k X_{\tilde{\theta}\tilde{\theta}}(k)\Phi_k^T + Var(w_k), \tag{4.8}$$

where $X_{\tilde{\theta}\tilde{\theta}}(k) = E\left[\tilde{\theta}_k\tilde{\theta}_k^T\right]$ is the mean squared error of the parameter estimate and $Var(w_k)$ is the variance of zero-mean white Gaussian noise.

Parameter estimation Since the system is time varying, $\Delta\theta_k = \theta_{k+1} - \theta_k \neq 0$. According to parameter estimation algorithm in (4.6), the parameter estimation error is

$$\tilde{\theta}_{k+1} = \tilde{\theta}_k - F_k\Phi_k^T\tilde{\mathbf{x}}(k+1-M|k-M) + \Delta\theta_k. \tag{4.9}$$

The estimated parameter is biased and the expectation of the error can be expressed as

$$\begin{aligned} E\left(\tilde{\theta}_{k+1}\right) &= \left[I - F_k\Phi_k^T\Phi_k\right] E\left(\tilde{\theta}_k\right) + \Delta\theta_k \\ &= \sum_{n=0}^{k}\prod_{i=n+1}^{k}\left[I - F_i\Phi^T(i)\,\Phi(i)\right]\Delta\theta_n. \end{aligned} \tag{4.10}$$

The mean squared error of parameter estimate follows from (4.9) and (4.10):

$$\begin{aligned} &X_{\tilde{\theta}\tilde{\theta}}(k+1) \\ =&F_k\Phi_k^T X_{\tilde{x}\tilde{x}}(k+1-M|k-M)\,\Phi_k F_k - X_{\tilde{\theta}\tilde{\theta}}(k)\Phi_k^T\Phi_k F_k \\ &- F_k\Phi_k^T\Phi_k X_{\tilde{\theta}\tilde{\theta}}(k) + E\left[\tilde{\theta}_{k+1}\right]\Delta\theta_k^T \\ &+ \Delta\theta_k E\left[\tilde{\theta}_{k+1}\right]^T - \Delta\theta_k\Delta\theta_k^T + X_{\tilde{\theta}\tilde{\theta}}(k). \end{aligned} \tag{4.11}$$

Since $\Delta\theta_k$ is unknown in (4.10) and (4.11), we define $d\theta_k = \hat{\theta}_k - \hat{\theta}_{k-1}$, and approximate $\Delta\theta_k$ as $\delta\theta_k$ which is the average of $d\theta_i, i = k - n_w + 1, k - n_w, ..., k$, where $n_w \in \mathcal{N}$ is the window size.

At step k, the predicted trajectory $\hat{\mathbf{x}}(k+1|k)$ together with the uncertainty matrix $X_{\tilde{x}\tilde{x}}(k+1|k)$ is then sent to robot control to generate the safety constraint.

4.4 Experiment and Simulation

4.4.1 Four Datasets

To demonstrate the effectiveness of the proposed method, we evaluate the proposed algorithm on four different human motion datasets. Two of them are real data, and two of them are simulated data. The reason that we use synthetic data is because that several factors can contribute to the effectiveness of human motion prediction, including noise-reduced measured data and well-defined adaptation procedures. In order to gain a deeper insight into the mechanism behind the proposed algorithm, we artificially simulate the human motion, where the scale of measurement noise and the noise-free trajectories can be controlled. The four datasets will be explained in the following.

Kinect Dataset

Human trajectory data is collected when a human and a robot collaborate to assemble a desktop. The experimental setup is shown in Fig. 4.2. A Kinect for Windows v2 is utilized to capture the trajectory of the human right wrist at an approximate frequency of $20Hz$. Human has two options: one is to obtain and insert the RAMs in the motherboard, and the other is to fetch and assemble the disk to the desktop case. 50 trajectories for each trajectory type are obtained, of which 80% is utilized for offline neural network training, and the remaining is for the validation of the online adaptation algorithm. To smooth the trajectories, we use a low-pass filter $p_s(k) = 0.6\hat{p}(k-1)+0.4\hat{p}(k)$. $p_s(k) \in \mathbb{R}^3$ is the smoothed position of the joint at time step k, which is the weighted average of joint positions $\hat{p}(k-1)$ and $\hat{p}(k)$ measured at time step $k-1$ and k. We set the number of past and future joint positions N and M both to be 3, which implies that we are doing predictions of three time steps approximately 0.15s. The prediction horizon can be controlled by adjusting the magnitude of M according to the specific applications.

CMU Motion Capture Dataset

Human activities are captured by the Vicon system [1]. Humans wear a black jumpsuit and have 41 markers taped on. The Vicon cameras see the markers in infra-red. The images that the various cameras pick up are triangulated to get 3D data. There are various motion categories, and we choose the data of "walking", "running" and "jumping" of all subjects.

Time Varying System (TV)

In human dynamics model (4.1), the future trajectory depends on the past trajectory $\mathbf{x}^*$. However, the trajectory itself is indeed a 3D curve parameterized by a one dimensional parameter. Here we parameterize the trajectory by time t. We use a polynomial function to

[1] Available at `http://mocap.cs.cmu.edu/motcat.php`

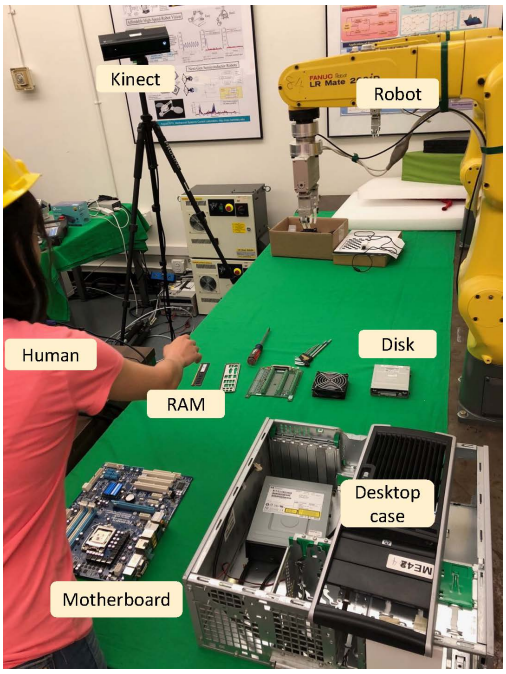


Figure 4.2: Experimental setup for human trajectory prediction.

represent the (x, y, z) trajectory:

$$x(t) = a_x(t+w)^2 + b_x(t+w)$$
$$y(t) = a_y(t+w)^2 + b_y(t+w)$$
$$z(t) = a_z(t+w)^2 + b_z(t+w)$$

where w denotes the artificial noise and $t \in [0, 5s]$. The trajectory is sampled every $0.05s$. It is easy to verify that given the 3D curve, the resulting dynamic model in the form of (4.1) is time-varying. We have applied two sets of parameters $[a_x, b_x, a_y, b_y, a_z, b_z]$ to generate two different simulated motions, $[0.4, -2, 0, 0.9, 0, 1.05]$ and $[0.41, -1.9, 1, 0.9, 0, 0.95]$, the unit for three axes is m. We also add artificial noise to the artificial motion trajectory data since measurement noise exists in real world data. Therefore, uniformly distributed noise w in the range $[-1, 1]$ is added to t. For each motion kind we simulate 50 independent trials.

Time Invariant System (TI)

In contrast to the time varying system, the time invariant system has constant model f that does not depend on time. Here we also choose a quadratic formula to parameterize the

time-invariant state transition model:

$$x(t+\Delta t) = a_x x(t)^2 + b_x x(t) + w$$
$$y(t+\Delta t) = a_y y(t)^2 + b_y y(t) + w$$
$$z(t+\Delta t) = a_z z(t)^2 + b_z z(t) + w.$$

We obtain the TI trajectory with the following strategy: we first randomize the initial position, and then at each sample iteration we get a new position using the quadratic state transition model above provided with the position from last iteration. In the experiment setting, we use two sets of parameters $[a_x, b_x, a_y, b_y, a_z, b_z]$ to represent two kinds of motion classes as well, $[0.06, 0.92, 0, 0.9, 0, 1.05]$ and $[0.061, 0.93, 0, 1.05, 0, 0.96]$. The unit for three axes is dm and the sample time $\Delta t = 0.05s$. Uniformly distributed noise is also added to the simulated data.

4.4.2 One Baseline

We compare our algorithm with the state-of-the-art online learning algorithm called identifier-based algorithm. The identifier-based algorithm adapts all the parameters in the offline-trained neural network model online using gradient descent to minimize the prediction error. Same architecture of the neural network as specified in [78] is utilized for a fair comparison. Both neural networks are trained offline for 100 epochs before online adaptation.

4.4.3 Implementation Details

The past trajectory input to the neural network $\mathbf{x}^* \in R^9$ is past 3-timestep positions in the world frame, and the future trajectory ouput by the neural network $\mathbf{x} \in R^9$ is future 3-timestep positions. We use a 3-layer neural network with 40 nodes in the hidden layer. The loss function is set to be L2 loss. The learning rate is set to 0.001 and the number of epochs is 100.

After obtaining the neural network model for motion transition, we use RLS-PAA to adapt the weights of the last layer. Since the number of nodes in the last layer is 9, and each node has 41 parameters, of which 40 correspond to the outputs from the hidden layer, and the remaining one parameter is a bias term. In total, there are 369 parameters to be adapted online. In the case that the initial values for all the parameters are set to be 10, the three parameters we randomly choose quickly settle into finite bounds as shown in Fig. 4.3.

4.4.4 Results

We evaluate the performance of online adaption algorithms by prediction errors. The comparison between the identifier-based algorithm and our methods is demonstrated in Fig. 4.4. We also compare the prediction errors by the neural network models without online adaptation. RLS-PAA results in the smallest prediction errors on all the datasets, which

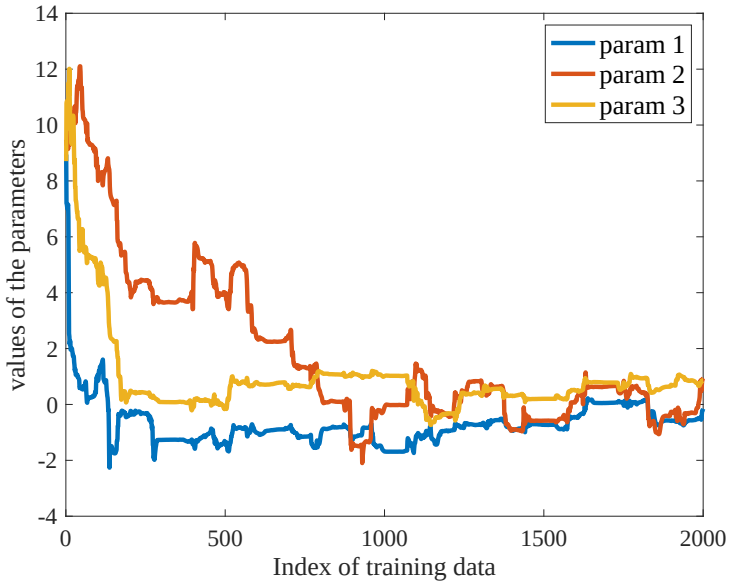


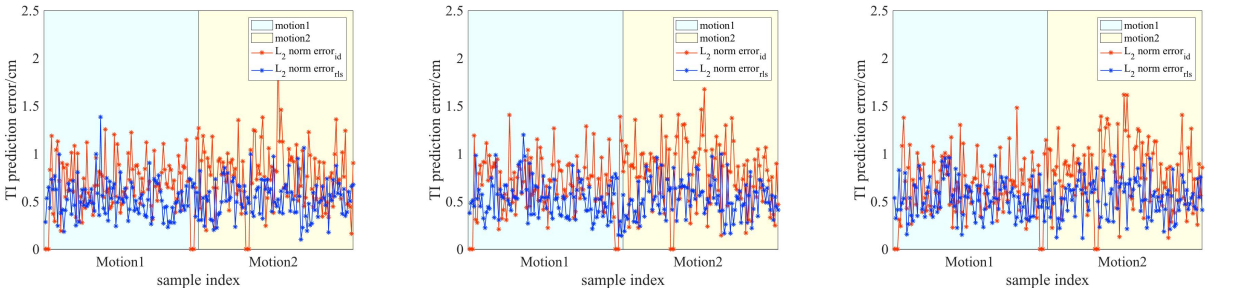
Figure 4.3: Parameter adaptation results.

demonstrates the effectiveness of our proposed algorithm. The mean squared estimation error (MSEE) of all methods are shown in Table 4.2. Compared to the identifier-based algorithm, RLS-PAA has a much smaller MSEE across four different datasets in all x, y, z axes.

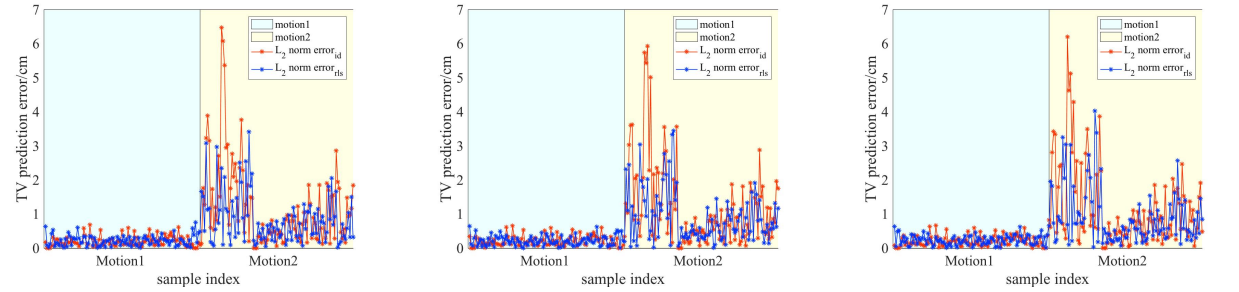
As for the training efficiency, since gradient operation is expensive to compute for large neural networks, we observed that it takes roughly 0.375s for the identifier-based algorithm to adapt one motion sample using standard gradient operation whereas it only takes 0.031s for our proposed method to adapt one sample. Our method is better for real-time applications. For the identifier-based algorithm that adapts all parameters in the neural network, even though it is possible to accelerate gradient operation using central difference approximation, the computation complexity grows exponentially as the complexity of the neural network grows. All the experiments are performed on the MATLAB 2016 platform with $2.7GHz$ Intel Core i5 Processor.

4.4.5 Discussion

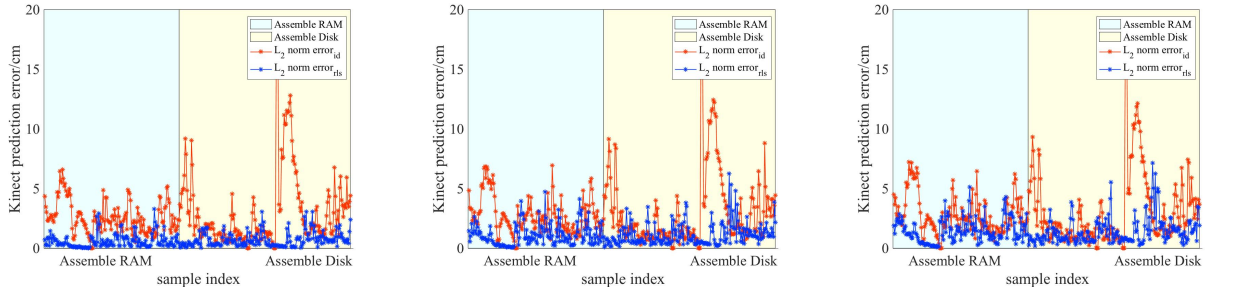
State-of-the-art identifier-based adaptation algorithm requires heavy coefficient tuning. The performance of the identifier-based algorithm highly relies on the scale of update step size. The computation of the gradient has inertia that it cannot adapt to the sudden change quickly. However, sudden change of velocity happens a lot in human motion. Therefore, some peak prediction error can be observed from time to time as shown in Fig. 4.4, especially at the beginning of each motion trial, since there is a clear velocity jump between the beginning



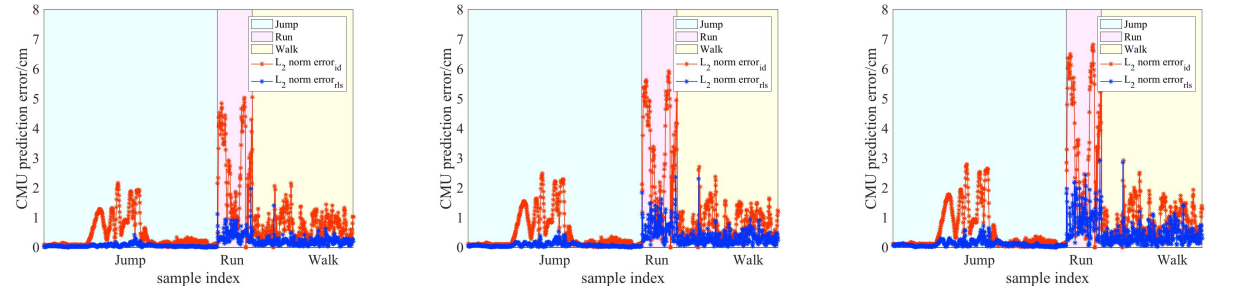
(a) Time Invariant dataset (k+1) (b) Time Invariant dataset (k+2) (c) Time Invariant dataset (k+3)



(d) Time Varying dataset (k+1) (e) Time Varying dataset (k+2) (f) Time Varying dataset (k+3)



(g) Kinect dataset (k+1) (h) Kinect dataset (k+2) (i) Kinect dataset (k+3)



(j) CMU dataset (k+1) (k) CMU dataset (k+2) (l) CMU dataset (k+3)

Figure 4.4: Prediction error comparison between RLS-PAA (blue system) and Identifier-based (red system) algorithm on four datasets. From top to bottom, each row is the experiment result tested on one trial of each motion on artificial time invariant system dataset, artificial time variant system dataset, Kinect dataset, and CMU dataset. In the experiment, when a new measurement is available, three future positions are predicted, and they are shown in the separate three columns respectively. The vertical black dash lines denote the boundaries of different motion classes.

	NN w ID	NN w RLS-PAA	NN w/o ID	NN w/o RLS-PAA
MSEE cm^2(CMU)	21.9248	5.5728	27.5121	30.2721
MSEE cm^2(Kinect)	27.3493	12.0419	29.6521	16.3973
MSEE cm^2(TV)	2.9093	2.2261	3.1889	2.5108
MSEE cm^2(TI)	4.2925	1.3180	5.3221	4.2642

Table 4.2: Mean Squared Estimation Error (MSEE) of predictions on four datasets for identifier-based algorithm (ID), recursive least square parameter adaptation algorithm (RLS-PAA), offline trained neural network in ID case without adaptation (NN w/o ID), and offline trained neural network in RLS-PAA case without adaptation (NN w/o RLS-PAA)

and the end of two different trials.

The number of past joint positions being considered has an impact on the prediction performance. RLS-PAA looks back into historical trajectory with exponentially decayed weights. The forgetting factor controls how far back the algorithm is using the past information. In our experiment on Kinect data, the optimal forgetting factor is 0.9998. Note that to accommodate a more rapidly changing motion, the forgetting factor may be set smaller so that the algorithm looks back in a short range.

Our proposed algorithm can be easily applied to many fields and extended by combination with other algorithms. Generally speaking, we can apply the method to many other time series data prediction, such as vehicle and pedestrian motion prediction. In these cases, we regard the vehicles and pedestrians as joints in our method, and adapt the model and obtain uncertainty level online, which is beneficial in autonomous driving. Our proposed algorithm can also be combined with other algorithms by modeling the velocity state transition pattern, which can be used to adjust the update step size.

The proposed algorithm can handle abnormal human behavior. Since humans are easily distracted from work, and they might do random things during the execution of a collaborative task. Under this situation, our adaptation algorithm can quickly accommodate the unseen behavior.

In many human-robot interaction applications, humans often work with robots over tables, standing or sitting in a fixed position where only the human upper body prediction is required. Note that the human upper body can be viewed as a fixed-base cascade robot arm, which can be represented by a kinematic chain model. In our experiments, we focus on motion prediction for one wrist joint, which can be viewed as the end-point position in the chain model. Whole arm movement can be predicted either using inverse kinematics or dynamic simulation given the prediction of the wrist position.

4.4.6 Integration to the Robot System

The trajectory prediction ensures a safe HRC framework. In Section 3.3.3, to quantify the safety of the proposed framework which includes the proposed trajectory prediction method,

we measure the minimum distance between a human subject and the robot during the entire task in each trial. Smaller the minimum distance, the less safe it is for the human. We compare between two trajectory prediction schemes:

- "trajectory prediction = 0": no predictions of human trajectory;
- "trajectory prediction = 1": prediction via our proposed method.

Through extensive experiments, the minimum distance between the human subjects and the robot for experiments with "trajectory prediction = 0" (72 trials) was $34.9 \pm 3cm$ and for that with "trajectory prediction = 1" (72 trials) was $36.0 \pm 2cm$. Under both experiment conditions, the minimum distances were within a safe distance and no collisions happened. The minimum distances in experiments with "trajectory prediction = 1" were larger than those in the experiments with "trajectory prediction = 0", although not significantly differ[2] ($p > 0.05$). There are two possible reasons: 1) in our scenario, the robot and the human worked in a relatively large space, so predictions of the human trajectory did not make much difference to the robot trajectory planning; 2) human subjects were conservative around the robot, and they intentionally kept a safe distance from the robot.

To show that the trajectory prediction module in our framework improves safety, we did additional tests where the human subjects aggressively move towards the robot end effector (We have a safety mechanism which immediately stops the robot if contact happens.). 100 trials with trajectory prediction and 100 trials without trajectory prediction were collected. The collision rates were 0/100 and 64/100, respectively. Such results qualitatively showed that the trajectory prediction module can improve safety.

4.5 Chapter Conclusion

This chapter proposed a semi-adaptable neural network to predict human motion, which is capable of adapting the model online and provides uncertainty levels for safety constraints. The offline trained neural network takes the historical joint trajectory as input and outputs the predictions. In an online test, the parameters of the last layer in the neural network model are adapted to accommodate individual differences and time-varying behaviors. The extensive experiment results demonstrated that our method significantly outperforms the state-of-the-art method on the majority of the motion datasets in terms of prediction error. Moreover, our model is much more computationally efficient and is free of onerous tuning, and the performance is robust across different motion categories and datasets. Last but not the least, the proposed trajectory prediction was integrated into our robot system, and experiments showed that the trajectory prediction is good for ensuring human safety.

[2]We use paired t-test for all the statistical tests.

Chapter 5

Long-Term Human Trajectory Prediction

5.1 Introduction

Trajectory prediction is for the purpose of safety. The more we predict into the future, the better it is for the robot trajectory planner to avoid potential collisions. Previous methods always focus on a fixed time step prediction [32][19], such as M step in our our semi-adaptable neural network method in Chapter 4. Now we want to solve for a longer horizon. For example, in Fig. 5.1, the human is reaching for the CPU fan. Based on our plan recognition module, it is predicted that the human will also obtain and install the CPU fan. Therefore, we want to predict the whole trajectory over the three actions. The benefit of such long-term prediction is that we can also estimate the duration of the action/whole trajectory from the long-term prediction, which is very useful for those task planning methods that optimize the task time [13][10].

It is challenging to make accurate predictions of the human movement due to the non-linearity and stochasticity of human behavior and the variety of its external and internal stimuli [80]. However, for the human workers in the assembly line, their movement behavior is mainly driven by the task goals and the layout of the parts, and it is possible to make good predictions using such contextual information. For example, as shown in Fig. 5.1, the human worker is assembling a desktop computer. By using the worker's hand trajectory, task information and the layout of the parts, we can infer that the human is going to reach the CPU fan, and we can also anticipate that the human will obtain the CPU fan and install it in the computer case. Therefore, human hand trajectory can be predicted in the long term, not only for the current action (reaching for the CPU fan), but also for the predicted actions (obtaining the CPU fan and installing the CPU fan). Besides, from the trajectory prediction, we can also estimate the action duration. Action duration is the time required to complete an action, which is an important input for many task planners. Therefore, in this chapter, we study the following problem: *predict the human hand trajectory and estimate the*

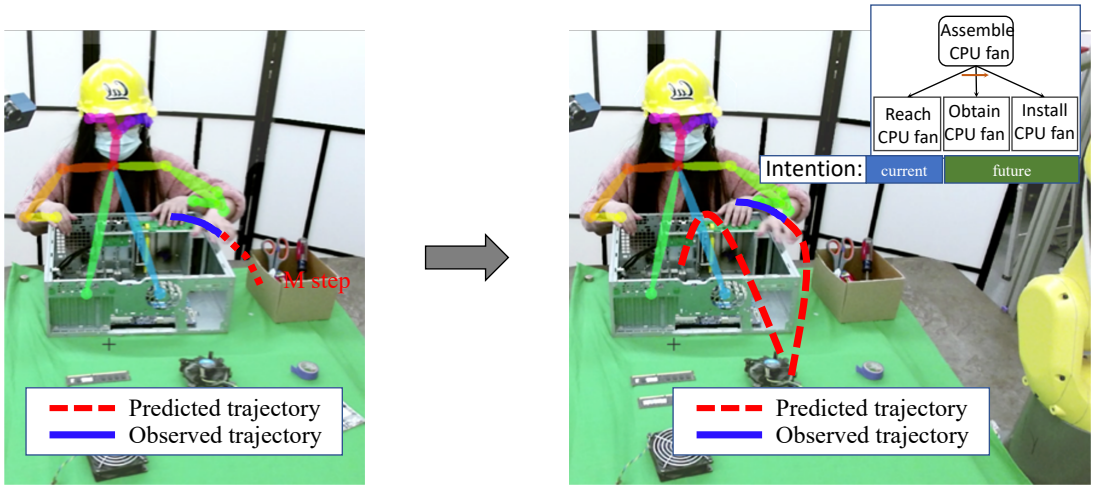


Figure 5.1: From short-term trajectory prediction to long-term trajectory prediction.

durations for the human's current action and future actions with the knowledge of historic trajectory, the task information and the scene context.

For trajectory prediction, two categories of approaches are proposed: 1) learning-based approaches such as recurrent neural networks (RNNs) [61] and inverse reinforcement learning [86], and 2) model-based approaches such as social force model [35] and minimum jerk model [26]. We are interested in developing model-based methods for their intuitive physical meanings and data efficiency property, and we propose to use sigma-lognormal equations to model human kinematics. To the best of our knowledge, it is the first time that this model is applied to the human-robot collaborative field. Besides, most works predict the trajectory by a preset horizon, while our prediction horizon is not fixed and it depends on how far in the future we could predict the human's actions, which is the longest prediction horizon given the results of the human intent recognition.

We propose to use a recognition-then-prediction framework. For the recognition, as shown in Chapter 3, the human's current action and future actions can be recognized by applying a hierarchical modeling method. For prediction, we propose to use a sigma-lognormal function to model and predict the human's velocity profile. The parameters of this model are trained offline, and we propose an adaptive algorithm to change the parameters online to suit different human behaviors. The prediction of the trajectory and the action duration estimation are both based on this model. Experiments validate the effectiveness of the proposed framework and demonstrate the superiority of our prediction method over the other four baseline methods in terms of trajectory prediction accuracy and duration estimation accuracy. Besides, experimental results also show that our adaptation algorithm can quickly adapt the model to human movement with the same motion type but with different starting and ending points.

The main notation used in this chapter is summarized in Table 5.1

Table 5.1: Table of Notation.

$\vec{v}$	$\in R^3$	Measured velocity vector.
t	$\in R$	Time.
v_x, v_y, v_z	$\in R$	Measured velocity on the x, y, and z axis.
$\hat{\vec{v}}$	$\in R^3$	Model prediction of the velocity vector.
$\hat{v}_x, \hat{v}_y, \hat{v}_z$	$\in R$	Model prediction of the velocity on the x, y, and z axis.
$\vec{\alpha} = [t_0, \mu, \sigma]$	$\in R^3$	Parameters for the lognormal function.
$\vec{\beta} = [C_s, t_s, D_s]$	$\in R^3$	Parameters for online adaptation.

5.2 Framework

Our goal is to predict the human hand trajectory over the horizon of the human's current intended action and the human's future actions. There are three aspects to be addressed: (i) obtaining the human pose from the sensor input, (ii) recognizing the human's current action and predicting the future actions, a.k.a. human intent recognition, and (iii) predicting the trajectory over those actions and estimating the action durations.

The pipeline of the proposed approach is schematically illustrated in Fig. 5.2. For the pose estimation module, Kinect Software Development Kit (SDK) is directly applied, which features real-time tracking of a human skeleton. For the intent recognition module, a hierarchical modeling method is utilized as in Chapter 3. Finally, for the long-term trajectory prediction module, we propose to use a sigma-lognormal model, and from this model, the future trajectory and the action durations are predicted.

5.3 Adaptable Lognormal Function

This section explains how we predict the trajectory and estimate the duration. We model human movements for each action in the task as the sigma-lognormal equations, and offline learn the parameters from the training data. Online, the model parameters are adapted every time when the measured data comes. We propose an adaptation algorithm, which utilizes the knowledge of the human intention and the target location. Finally, we predict the trajectory by using the updated model and we estimate the duration by the stable zero-crossing point of the velocity profile with some threshold.

5.3.1 Sigma-lognormal model

Many human movement models have been proposed based on various techniques such as neural networks [27], minimization principals [26], and kinematic theories [75]. Among them,

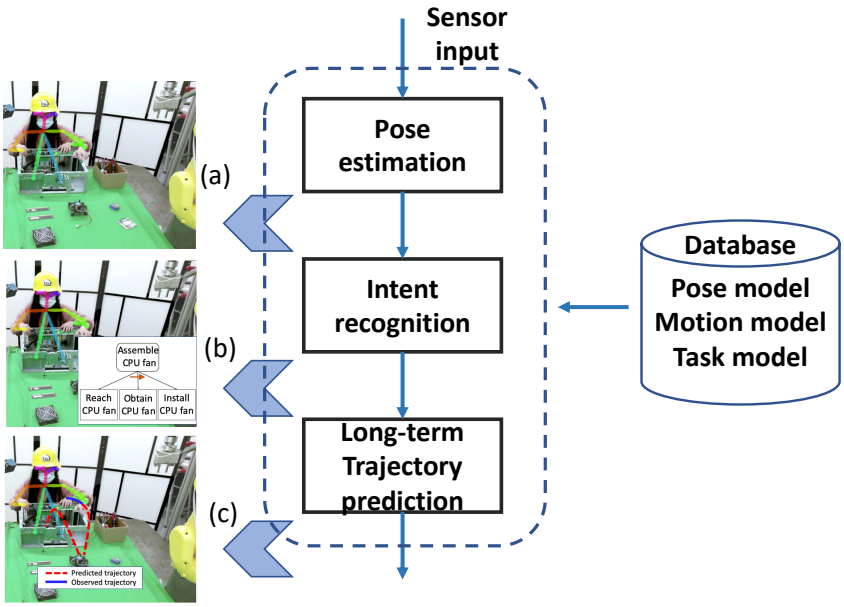


Figure 5.2: The framework for long-term human trajectory prediction. Given the image sensor inputs, we first detect (a) 3D human pose. We then predict the (b) human's intention which includes the current and the future actions. Finally, our model predicts the (c) trajectories over those actions.

the kinematic theory and its sigma–lognormal model explain most of the basic phenomena on human motor control [75], and it is proved that it is ideal for describing the impulse response of a neuromuscular network which is composed of some subsystems controlling the velocity of a movement [75]. It achieves many successes in applications like signature verification [74] and handwriting learning [24]. We now apply it to the human-robot collaboration field, which, to the best of our knowledge, is the first application. The model takes the following form:

$$\hat{\vec{v}}(t) = \sum_{i=1}^{N} \hat{\vec{v}}_i(t) = \sum_{i=1}^{N} \vec{D}_i(t)\Lambda_i(t; t_{0i}, \mu_i, \sigma_i^2); N \geq 3, \tag{5.1}$$

$$\Lambda_i(t; t_{0i}, \mu_i, \sigma_i^2) = \frac{1}{\sigma_i\sqrt{2\pi}(t - t_{0i})} exp(\frac{-(\ln(t - t_{0i}) - \mu_i)^2}{2\sigma^2})$$

where $\hat{\vec{v}}(t)$ is the velocity of the human hand at time t, and $\Lambda(t, t_0, \mu, \sigma^2)$ is a lognormal distribution with the time shift t_0, the expected value of the t's natural logarithm μ and the standard deviation of the t's natural logarithm σ. The velocity $\hat{\vec{v}}(t)$ is composed of N lognormal distributions Λ_i, each scaled by variable $\vec{D}_i, i = 1, ..., N$. Since the human's hand is moving in three dimensions, the number of lognormal distribution N is equal to or larger

than 3. For example, simple reaching or pointing gestures require three weighted lognormals to describe the speed profile. More complex trajectories require more lognormals.

An empirical way of deciding N is to project the velocity profile to the spaces defined by the x-axis, the y-axis and the z-axis, and sum up the observed number of bell shapes in each file. Indeed, Equation (5.1) can be decomposed into three independent scalar equations, and they can be learned and adapted independently. Fig. 5.3 shows an example of using sigma-lognormal functions to represent the velocity profile of one axis. The dashed curves are two lognormal functions, and the solid curve is a velocity profile. The velocity profile can be approximated by the superposition of the two lognormal functions.

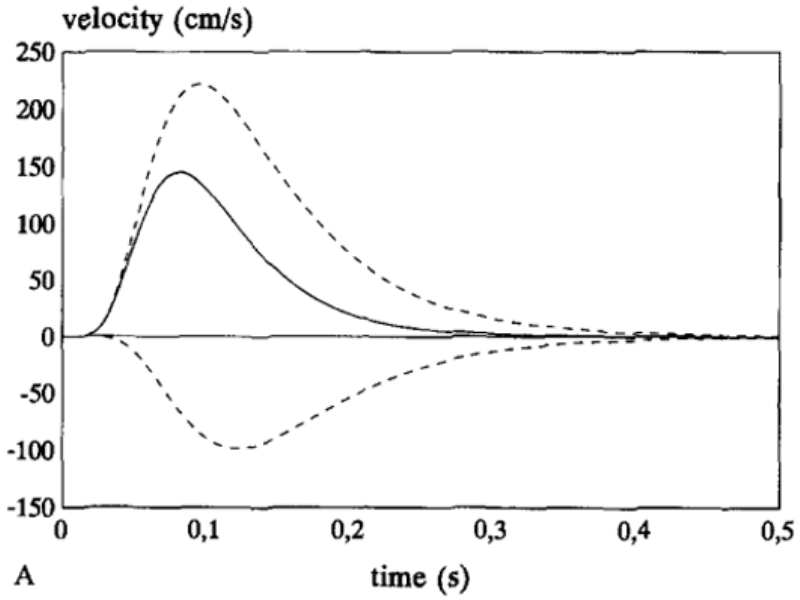


Figure 5.3: An example of using sigma-lognormal functions to represent the velocity profile. The dashed curves are two lognormal functions, and the solid curve is a velocity profile. The velocity profile can be approximated by the superposition of the two lognormal functions.

5.3.2 Offline model fitting

To use the sigma–lognormal model for analyzing the human movement, we need to choose the values for parameters $\vec{\alpha} = \{\vec{D}_i, t_{0i}, \mu_i, \sigma_i^2\}, i = 1, ..., N$ such that the model can fit the training data well. Two categories of parameter extractors have been proposed in the literature: the Xzero-based extractor [70] and the prototype-based extractor [76]. The Xzero-based extractor has closed-form solutions by analyzing several characteristic points located in the original velocity profile, but it lacks the knowledge of the nature of the movement and it is easily influenced by the noise in the data. The prorotype-based extractor takes advantage

of a priori information and works well for stereotypical movements. It first synthesizes the prototype, then it adjusts the parameters by scaling and offsetting. This is a good philosophy for online parameter extraction, but not suitable for offline model fitting.

We propose to use the Levenberg–Marquardt algorithm [83], also known as the damped least-squares method, to solve the following problems: given a set of m data points $(t_j, \vec{v}_j)$ from the collected trajectories, find $\vec{\alpha}$ such that the sum of the squares of the deviations $S(\vec{\alpha})$ is minimized:

$$\vec{\alpha}^* = \arg\min_{\vec{\alpha}} S(\vec{\alpha}) = \arg\min_{\vec{\alpha}} \sum_{j=1}^{m} ||\vec{v}_j - \hat{\vec{v}}(t_j, \vec{\alpha})||_2^2.$$

5.3.3 Online parameter extraction

Since human behavior is time-varying and different people have different motion styles, it is necessary to adapt the offline learned model to accommodate such variations. [96] concludes that the time scaling and shifting account for a large variability of human movement, and only the modification of μ_i and the t_{0i} is required through scaling factor C_{si} and t_{si} in Equation (5.2) and Equation (5.3). In addition to C_{si} and t_{si}, we add one more scaling factor D_{si} for D_i to account for more variations as in Equation (5.4).

$$t_{0is} = C_{si} t_{0i} + t_{si} \tag{5.2}$$
$$\mu_{is} = \mu_i + ln(C_s) \tag{5.3}$$
$$D_{is} = D_{si} D_i \tag{5.4}$$

Therefore, the sigma-lognormal model becomes $\hat{\vec{v}}(t; \vec{\alpha}^*, \vec{\beta})$, where $\vec{\beta} = \{C_{si}, t_{si}, D_{si}\}, i = 1, ..., N$. The problem of adapting the sigma-lognormal model is to learn the values of $\vec{\beta}$. The initialization of $\vec{\beta}$ requires that the sigma-lognormal model starts from the offline learned model in Section 5.3.2. Thus, the scaling factors C_{si} and D_{si} are set to 1, and the time shifting factor t_{si} is set to 0, for $i = 1, ..., N$. As indicated in Section 6.2, the sigma-lognormal equation in Equation (5.1) comprises three scalar equations which can be adapted independently. Hence, we decouple the parameter $\vec{\beta}$ into three parts $\{\vec{\beta}_x, \vec{\beta}_y, \vec{\beta}_z\}$ and adapt them separately.

To adapt $\vec{\beta}$ when a new data point is available, two clues are essential: 1) how the new data point deviates from the model prediction can direct the modification of the model parameters, and 2) the integral of the velocity file should be aligned with the human's intentions and the target locations.

By the first clue, $\vec{\beta}$ is updated using Levenberg–Marquardt algorithm as in Equation (5.5) (5.6) and (5.7). Note that the operator "." indicates an element-wise operation throughout this chapter. $\hat{v}_{x,t_k}, \hat{v}_{y,t_k}, \hat{v}_{z,t_k}$ and $v_{x,t_k}, v_{y,t_k}, v_{z,t_k}$ are the model predictions and the measurements of the velocities in x, y and z directions at time t_k. $\nabla_{\vec{\beta}_x}, \nabla_{\vec{\beta}_y}, \nabla_{\vec{\beta}_z}$ are the gradients of $\hat{v}_{x,t_k}, \hat{v}_{y,t_k}, \hat{v}_{z,t_k}$ with respect to $\vec{\beta}_x, \vec{\beta}_y, \vec{\beta}_z$, and $\vec{\lambda}_x, \vec{\lambda}_y, \vec{\lambda}_z$ are the non-negative damping factors,

which make the algorithm interpolate between the Gauss–Newton algorithm and the method of gradient descent [48]. If values of $\vec{\lambda}$'s components are large, the step goes in the direction for the gradient descent algorithm, and if values are small, the step will be close to the step in the Gauss–Newton algorithm.

$$\vec{\beta}_x \leftarrow \vec{\beta}_x - (\hat{v}_{x,t_k} - v_{x,t_k})\nabla_{\vec{\beta}_x}\hat{v}_{x,t_k}./((\nabla_{\vec{\beta}_x}\hat{v}_{x,t_k})^{.2} + \vec{\lambda}_x) \tag{5.5}$$

$$\vec{\beta}_y \leftarrow \vec{\beta}_y - (\hat{v}_{y,t_k} - v_{y,t_k})\nabla_{\vec{\beta}_y}\hat{v}_{y,t_k}./((\nabla_{\vec{\beta}_y}\hat{v}_{y,t_k})^{.2} + \vec{\lambda}_y) \tag{5.6}$$

$$\vec{\beta}_z \leftarrow \vec{\beta}_z - (\hat{v}_{z,t_k} - v_{z,t_k})\nabla_{\vec{\beta}_z}\hat{v}_{z,t_k}./((\nabla_{\vec{\beta}_z}\hat{v}_{z,t_k})^{.2} + \vec{\lambda}_z) \tag{5.7}$$

Second, to take advantage of the human intentions and the scene information, we further update $\vec{\beta}$ to minimize the following three objectives.

$$J_1(\beta) = || \int_{t_k}^{t_f} \hat{\vec{v}}(s; \vec{\alpha}^*, \vec{\beta})ds - \vec{d}_{togo}||_2^2 \tag{5.8}$$

$$J_2(\beta) = ||\vec{v}_{t_k} - \hat{\vec{v}}(t_k; \vec{\alpha}^*, \vec{\beta})||_2^2 \tag{5.9}$$

$$J_3(\beta) = ||\hat{\vec{v}}(t_f; \vec{\alpha}^*, \vec{\beta})||_2^2 \tag{5.10}$$

The objective function (5.8) implies the predicted displacement from the current time t_k to the final time t_f should be close to the displacement to the goal $\vec{d}_{togo}$, where the final time t_f is obtained from a stable zero-crossing of the model $\hat{\vec{v}}(t; \vec{\alpha}^*, \vec{\beta})$ with a threshold, and $\vec{d}_{togo}$ is obtained by $\vec{p}_O - \vec{p}_{t_k}$. $\vec{p}_{t_k}$ is the current position of the human hand and $\vec{p}_O$ is the position of the target. In addition, the objective function (5.9) and (5.10) ensure the model precision at the current time t_k and the final time t_f, respectively. Therefore, to optimize the three objective functions is to fix the two ends of the future velocity file and to modify velocities in between so that the human's hand can reach the goal at the anticipated final time. Since (5.8)(5.9)(5.10) may not necessarily be minimized for the same value of β, we make weighted sum of the three objectives as $K(t_f, \vec{\beta}) = \gamma_1 J_1(\vec{\beta}) + \gamma_2 J_2(\vec{\beta}) + \gamma_3 J_3(\vec{\beta})$, and update $\vec{\beta}_x$, $\vec{\beta}_y$ and $\vec{\beta}_z$ by:

$$\vec{\beta}_x \leftarrow \vec{\beta}_x + K(t_f, \vec{\beta})\nabla_{\beta_x}K./((\nabla_{\beta_x}K)^{.2} + \vec{\lambda}'_x) \tag{5.11}$$

$$\vec{\beta}_y \leftarrow \vec{\beta}_y + K(t_f, \vec{\beta})\nabla_{\beta_y}K./((\nabla_{\beta_y}K)^{.2} + \vec{\lambda}'_y) \tag{5.12}$$

$$\vec{\beta}_z \leftarrow \vec{\beta}_z + K(t_f, \vec{\beta})\nabla_{\beta_z}K./((\nabla_{\beta_z}K)^{.2} + \vec{\lambda}'_z) \tag{5.13}$$

The outline of the algorithm at time t_k is shown in Algorithm 4. Lines 1-3 receive the position of the human hand $\vec{x}_{t_k}$. Lines 2 represents the intent recognition process, and it determines the current action a_{t_k} and the future actions $\vec{a'_{q^*}}$. Line 3 obtains the current velocity where δT is the time difference between the current time step and the last time step. Lines 4-7 retrieve the current action model and the future action models if the action recognition changes. Lines 8-15 illustrate the model adaptation process. Finally, Lines 16-22 predict the trajectory for the current action and the predicted actions.

Algorithm 3 Long-term motion prediction algorithm at time t_k.

Input $\vec{v}_{1:t_{k-1}}$, $\vec{x}_{1:t_{k-1}}$, $a_{t_{k-1}}$, $\hat{\vec{v}}$, $\hat{\vec{v}}_{1,2,...,np}$
Output t_f, $\hat{\vec{x}}_{t_{k+1},...,t_f}$

$\vec{x}_{t_k}$ = PoseEstimation()
$a_{t_k}, \vec{a'_{q^*}}$ = IntentRecognition($\vec{x}_{1:t_k}$)
$\vec{v}_{t_k} = (\vec{x}_{t_k} - \vec{x}_{t_{k-1}})/\delta T$
if a_{t_k} changed **then**
    $\hat{\vec{v}}(t; \vec{\alpha}^*, \vec{\beta})$ = RetrieveMotionModel(a_{t_k})
    $\hat{\vec{v}}_{1,2,...,np}(t; \vec{\alpha}^*, \vec{\beta})$ = RetrieveMotionModel($\vec{a'_{q^*}}$)
end if
for Iteration = 1,2, ... **do**
    Update $\vec{\beta}$ using Equation (5.5)(5.6)(5.7)
end for
t_f = ZeroCrossing($\hat{\vec{v}}(t; \vec{\alpha}^*, \vec{\beta})$)
for Iteration = 1,2, ... **do**
    Update $\vec{\beta}$ using Equation (5.11)(5.12)(5.13)
    t_f = Zero-Crossing($\hat{\vec{v}}(t; \vec{\alpha}^*, \vec{\beta})$, threshold)
end for
for t = t_{k+1},...,t_f **do**
    $\hat{\vec{x}}(t) = \vec{x}(t_k) + \int_{t_k}^{t} \hat{\vec{v}}(s; \vec{\alpha}^*, \vec{\beta}) ds$
end for
$\vec{t_f}$' = Zero-Crossing($\hat{\vec{v}}_{1,2,...,np}(t; \vec{\alpha}^*, \vec{\beta})$,threshold)
for $i = 1, 2, ..., np$ **do**
    Predict $\hat{\vec{x}}(t)$, for $t = t'_{fi-1} : t'_{fi}$
end for

5.4 Experiments

5.4.1 Scenario

A computer assembly task is considered. The task includes three subtasks, "Assemble CPU fan," "Assemble memory" and "Assemble system fan," where the human should complete "Assemble CPU fan" first, and then complete either "Assemble memory" or "Assemble system fan". Therefore there are two plans in total. For all subtasks, the action sequence is reaching the part, obtaining the part and installing the part.

5.4.2 Hypothesis

We evaluate the effectiveness of the proposed method through experiments by verifying the following hypotheses.

- H1: Our method makes better trajectory prediction and duration estimation than other baseline methods when the human's intentions are known.
- H2: Our method makes good trajectory prediction and duration estimation when the human's intentions are unknown.
- H3: Our proposed adaption algorithm can quickly adapt the movement model to the motions with the same motion type but different starting and ending positions.

The conditions that the human's intentions are known or unknown reflect whether the human intent recognition module in Section 5.2 is involved or not. The first hypothesis shows the effectiveness of our trajectory prediction module in Section 5.3 only, and the second hypothesis shows the effectiveness of the overall structure.

5.4.3 Experiments

We conducted three comparative studies to evaluate the effectiveness of our method. The first study compares different methods. Our algorithm is compared against the following baseline methods:

– Vector field method [15]: This method performs long-term predictions of human motion by employing a map of vector fields. The final time of an action for this method can be estimated by setting a velocity threshold.

– Minimum jerk model [26]: This model states that human arms move by minimizing mean-square jerk for any unconstrained point-to-point movement. As proposed in [46], the value of the final time can be obtained by a search algorithm.

– DTW-based method [60]: This method utilizes dynamic time warping (DTW) algorithm to align the online activity with a reference template, then it estimates the expected duration by proportionally manipulating the alignment time.

– Our method without task context: This is our method without the adaptation steps in Lines 12-15 of Algorithm 4.

The second study considers task types. Two types are considered: (1) the task that only includes a simple reaching task ("reaching the CPU fan" as in Section 5.4.1), where the human's intention is determined and known beforehand, and (2) the whole task as described in Section 5.4.1, where the human's intention is uncertain and needs to be recognized. Our method and all the 4 baseline methods are applied to the first task type, while for the second task type, we only test our method.

The third study studies a set of progress percentages for the test timings. They are 10%, 30%, 40% 50%, 60%, 70%, 80%, 85% and 90%. For the reaching task, at these progress percentages of the reaching motion, we will study trajectory prediction and duration estimation, while for the whole computer assembly task, we will study trajectory prediction

and duration estimation at these progress percentages of the reaching motion of the second subtask, when the human intent recognition starts to take an important role to decide on which subtask the human intends to do.

In these three studies, we have 54 groups of experiments in total. For each group, we use 42 trials for offline training and 7 trials for the test. If the method does not require training data, the set of training trials is set aside. To have a fair comparison, the training and test set come from a random selection, and the two sets are the same through different groups. Besides, the hyperparameters are chosen by grid search cross-validation so that each algorithm can perform its best.

5.4.4 Dependent measures

To quantify the accuracy of trajectory prediction, we measure the mean Euclidean distance between the predictions and the observed data. For duration estimation, we compute the mean absolute percentage error. The absolute percentage error is the absolute time error divided by the groundtruth duration.

5.4.5 Implementation Details

For both task types, the number of lognormal functions are chosen as 3. The velocity profiles are fit in the x, y, and z directions independently, so we are fitting a one dimensional time series to a lognormal function. At offline training, the MATLAB function *nlinfit* is utilized for batch fitting. At online adaptation, the weighting parameters λ_1, λ_2 and λ_3 are fine tuned based on the performance in the validation set, and they are set to 1, 1, and 0.01 respectively. The threshold for identifying zero crossing points in Algorithm 3 is set to 0.1 millimeter per second.

5.4.6 Results

H1: Our method makes better trajectory prediction and duration estimation than other baseline methods when the human's intentions are known. Table 5.2 shows the duration estimation results when the human is conducting a simple reaching task. Our method has significantly smaller average percentage errors than other baseline methods at all the test timings except for the timing at the 80% progress, where ours is the second best. By doing a paired t-test, we find that the results of our method are significantly better than any other baseline method ($P < 0.05$). Besides, after the 30% progress, the average percentage error of our method drops below 5%, while for other baseline methods except for our method without task context, the errors are always above 5%. For our method and the minimum jerk method, the average percentage error drops as time elapses, while for the other methods, the average percentage error oscillates or even increases at the latter part. A possible reason is that our method and minimum jerk method explicitly utilize the knowledge of the target positions, while the others do not. Among the baseline methods, the

DTW-based method is the best in the early phase before the 30% progress, the vector field method is the best in the middle phase from the 30% to the 60% progress, and our method without task context is the best in the latter phase after the 60% progress.

Table 5.3 shows the trajectory prediction results when the human is conducting a simple reaching task. Our method has the least average distance errors at all the test timings except for the timings at 85% and 90% progress, where ours is the second best and the third best respectively. By doing a paired t-test, we find that the results of our method are significantly better than any other baseline method ($P < 0.05$). Besides, all the average distance errors of our method remain within 20 mm, and it drops below 10 mm after the 60% progress. For our method, the vector field method and the minimum jerk method, the average percentage errors show a trend of decreasing as time elapses. However, the DTW-based method and our method without task context show a slight increase in the early phase before the 50% progress. Among the baseline methods, the minimum jerk method is the best throughout all the test timings except for the timing at 90% progress, and the DTW-based method is the worst throughout all the test timings except for the timing at 10% progress.

H2: Our method makes good trajectory prediction and duration estimation when the human's intentions are unknown. Table 5.4 displays the results of the trajectory prediction and the duration estimation for the computer assembly task. For the current action, the average percentage error of the duration estimation varies between 1.3% and 14.9%, and it drops below 5% after the 30% progress. The average distance error for trajectory prediction varies between 3.6 mm to 19.7 mm, and it drops below 10 mm after the 60% progress. For the predicted actions, different from the results for the current action, the errors do not feature a decreasing trend, which means $\vec{\beta}$ is not the same among different motions for one task. The average percentage error of the duration estimation varies between 6.9% and 13.2%, and the average distance error of the trajectory prediction varies between 7.7 mm to 21.4 mm. The largest average distance error of the trajectory prediction 21.4 mm appears at the 10% progress, when the human intent recognition module has difficulty distinguishing the human's plan due to the lack of observations. After the 30% progress, the distance error drops exceedingly, because the human's intentions are correctly recognized. Overall, the duration estimation is good, since the error is within 5% for the current action and within 14% for the future actions after the 30% progress; the trajectory prediction is good, since errors are less than 25 mm, which is within the safety distance as in [16].

H3: Our proposed adaption algorithm can quickly adapt the movement model to the motions with the same motion type but different starting and ending positions. We adapt the sigma-lognormal model which is trained on the "reaching for the CPU fan" task to a new task "reaching for the memory", where the motion type is the same, but the starting and the ending positions of the human hand are different. Fig. 5.4 shows the distance errors of the model predictions for 7 trials. The model parameters are adapted through the first trial to the seventh trial, and the distance errors are calculated at 10%, 30%, 40% 50%, 60%, 70%,80%, 85% and 90% progress of each trial. As shown in the figure, the distance errors are large for the first trial, which can reach up to 40 mm. However, the error drops drastically after the first trial, which means the model can adapt to the new reaching

Table 5.2: Average absolute percentage error of duration estimation for a simple reaching task (Unit: %).

Time Percentage	10%	30%	40%	50%	60%	70%	80%	85%	90%
Ours	**13.6**±12.0	**3.9**± 9.7	**2.0**±8.4	**4.0**±7.0	**1.9**±3.5	**3.2**±3.1	1.7±3.6	**1.2**±6.1	**0.1**±5.2
Vector field	25.8±10.3	14.1±7.6	8.7±6.4	8.2±4.5	6.9±3.9	6.7±4.9	10.0±3.8	10.6±5.8	8.4±4.7
Minimum jerk	25.5±5.5	22.1± 6.8	22.4± 7.7	21.5±8.3	18.3± 7.4	14.4±6.2	10.0±5.2	7.7± 4.1	6.1± 3.6
DTW	15.3±18.1	10.9± 12.2	12.1± 11.7	11.7±10.7	10.8±9.8	13.7±7.1	11.5±10.6	14.9± 16.7	12.4± 14.9
Ours w/o task context	16.1±11.6	14.1± 8.8	12.8± 8.2	9.6±6.2	9.9±5.4	6.2±2.3	**1.1**±2.9	3.9±4.9	2.4±5.1

Table 5.3: Average distance error of trajectory prediction for a simple reaching task (Unit: mm).

Time Percentage	10%	30%	40%	50%	60%	70%	80%	85%	90%
Ours	**16.4**±8.7	**17.3**±4.8	**13.0**±3.3	**12.2**±3.4	**10.3**±3.9	**6.3**±2.0	**5.1**±1.7	4.5±1.0	3.2±0.7
Vector field	69.0±14.5	38.9±10.0	27.4±10.3	21.5±9.2	20.2±9.3	22.4±14.9	13.4±14.3	10.4±11.8	7.4 ±8.3
Minimum jerk	23.8±6.5	18.0±4.2	17.3±4.2	15.6±4.2	12.8±3.7	9.4±3.0	5.9±2.4	**4.3**±1.9	3.1±1.5
DTW	42.7±24.6	55.1±16.4	59.5±14.6	60.6±15.8	59.3±16.5	49.9± 17.8	43.9±19.1	40.3±21.9	32.7±16.8
Ours w/o task context	28.1±8.4	28.9±6.0	31.2±6.5	31.5±6.1	29.5±5.4	20.5±4.5	10.1±1.7	5.1±1.2	**2.3**±0.5

Table 5.4: The average percentage error of the duration estimation and the average distance error of the trajectory prediction for the computer assembly task.

Time percentage		10%	30%	40%	50%	60%	70%	80%	85%	90%
time error (%)	current action	14.9±11.8	4.9± 8.6	4.5±4.4	4.4±7.5	2.3±3.7	2.4±2.9	3.1±3.9	2.2±4.7	1.3±4.6
	future actions	11.2±10.6	10.5±13.3	8.7±9.5	9.2±10.6	10.3±11.9	7.4±10.7	6.9±11.6	13.2±9.1	11.2±10.5
trajectory error (mm)	current action	19.7±6.8	14.6±7.6	13.0±5.6	13.4±7.6	9.1±5.9	7.3±3.5	6.3±2.6	5.4±1.0	3.6±1.1
	future actions	21.4±13.6	9.4±8.8	12.3±8.4	14.7±11.4	8.5±6.7	10.4±13.7	8.9±9.6	7.7±8.1	12.7±9.8

Table 5.5: The average percentage error of the duration estimation and the average distance error of the trajectory prediction for adapting to another task.

Time Percentage	10%	30%	40%	50%	60%	70%	80%	85%	90%
time error (%)	15.1±16.2	12.6 ±10.2	5.4±6.3	5.1±6.2	4.2±5.7	3.5±4.9	3.8±5.2	1.8±2.5	1.2±2.3
trajectory error(mm)	19.0±10.3	14.8±7.5	14.2±10.9	13.1±6.3	9.1±4.8	4.3±2.7	3.3±4.8	1.1±0.5	0.6±0.3

task within one trial. Table 5.5 summarizes the statistics for the results of the trajectory prediction and the duration estimation, and it shows that the results are comparable with the results of our method in Table 5.2 and Table 5.3, where the model is trained and tested on the same reaching task. For the duration estimation, the percentage error is below 10% after the 30% progress and below 5% after the 50% progress. For the trajectory prediction, all the average distance errors remain within 20 mm, and it drops below 10 mm after the 50% progress.

5.5 Discussion

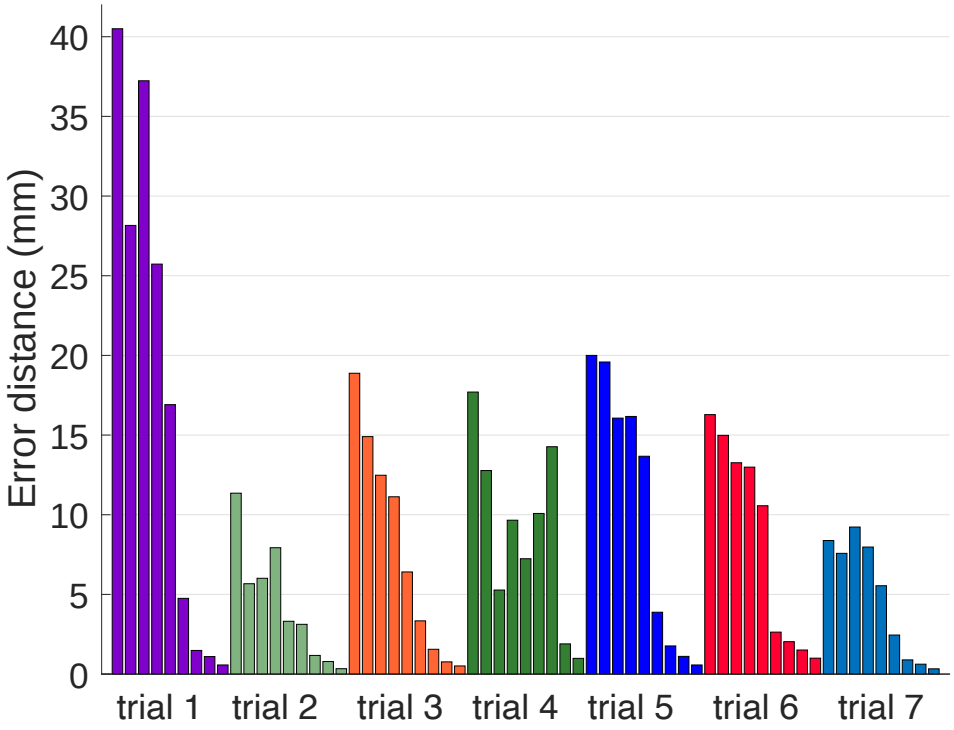


Figure 5.4: The distance errors of the model predictions for adapting to a new task.

D1: The intermediate time estimate t_f is important for the accuracy of the duration estimation and trajectory prediction. We vary t_f in line 19 of Algorithm 4 by setting the value to $\frac{1}{4}T$, $\frac{1}{2}T$, T, $2T$, $3T$ and $4T$, where T is the groundtruth final time. We compare the results of duration estimation and trajectory prediction between our original method and the method using the set of preset intermediate time estimates for the simple reaching task. Table 5.6 displays the p-values of significance using paired t-test. As shown in the table, there are significant differences through all settings except for t_f being T. This indicates

that if the intermediate time estimate deviates far from the groundtruth, the results of our algorithm will be heavily impacted.

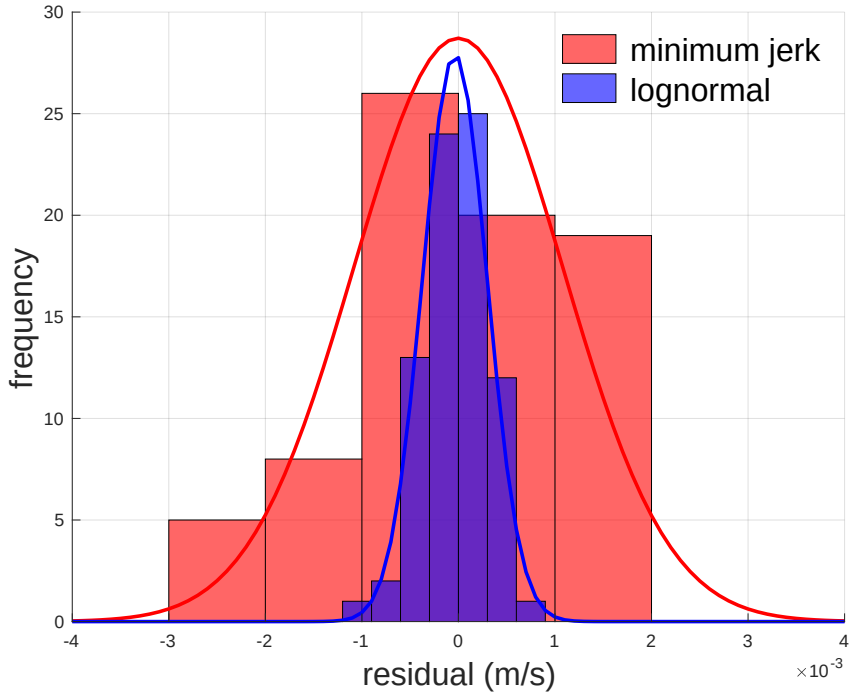


Figure 5.5: Histograms and normal fits for the residuals of the minimum jerk model and the lognormal model.

D2: The sigma-lognormal model fits the human motion data better than the minimum jerk model. Fig. 5.5 shows the residual diagrams for the minimum jerk model and the sigma lognormal model of one trial for the simple reaching task. Both diagrams are unskewed and include no outliers, and both residuals are normally distributed with probability $P = 0.09 > 0.05$ ($\chi^2 = 4.81$, $df = 2$) and probability $P = 0.18 > 0.05$ ($\chi^2 = 3.41$, $df = 2$) respectively for D'Agostino-Pearson's test. Actually, this normality conclusion holds for all the other 6 test trials, which indicates that we can trust the results of the regression analysis. Note that although normality assumption is not needed for nonlinear regression, if it is clearly violated, we may not trust the model. That is why we do the normality test.

Next, we check the mean absolute residual to see which model fits the data better. The mean absolute residual for the minimum jerk models is 1.20 ± 1.05 mm, and the mean absolute residual for the sigma lognormal models is 0.52 ± 0.42 mm. The sigma-lognormal model has 56.5% less error. This shows that the sigma-lognormal model fits the data better than the minimum jerk model.

D3: Our proposed system shown in Fig. 6.3 runs at $15 \sim 30$ hertz, which meets the real-time requirement for human-robot collaboration. The bottleneck is the pose estimation

Table 5.6: P-values of the duration estimation results and the trajectory prediction results between our method and methods using $\frac{1}{4}T$, $\frac{1}{2}T$, T, $2T$, $3T$ and $4T$ as in line 19 of algorithm 4. The shaded cells indicate that there are significant differences when the significance level is set to 0.05.

	$\frac{1}{4}T$	$\frac{1}{2}T$	T	$2T$	$3T$	$4T$
time result	0.000	0.023	0.099	0.793	0.000	0.000
trajectory result	0.000	0.000	0.306	0.041	0.022	0.000

module. The body stream of the Kinect sensor returns body data at 15 frames per second (average quality) and 30 frames per second (high quality) [93]. For the intent recognition module and the prediction module, it takes around 3 and 2 milliseconds, respectively. Therefore, if a higher frequency is required, the pose estimation module should be replaced by a faster method.

D4: The adaptation algorithm takes the goal location as an input, and assumes that humans are focused on the goal. However, humans do not always concentrate on work and might get distracted by random things, such as picking up cell phones. Our adaptation algorithm cannot handle such abnormal activities well.

5.6 Chapter Conclusion

We proposed a recognition-then-prediction framework for long-term human hand prediction and action duration estimation, where the prediction horizon relies on how far in the future we can predict the human actions rather than a preset time. We improved the recognition method by introducing scaling vectors, which takes advantage of the task knowledge of action transitions. The sigma-lognormal equation was proposed to model the human's movement, and an adaptation algorithm was proposed to accommodate different human behaviors. We conducted experiments for a computer assembly task. The results showed the effectiveness of the overall framework and the effectiveness of the trajectory prediction module only. The proposed method had significantly smaller average percentage errors as well as average distance errors. This is a definite advantage in predicting the long-term human hand trajectory for collaborative task planning. The experiments also showed that our adaptation algorithm can quickly adapt the movement model to the trajectories with the same motion type but different starting and ending positions.

Chapter 6

Human-Aware Task Planner

6.1 Introduction

When robots work with humans for a collaborative task, they need to plan their actions while taking humans' actions into account. In Section 3.3.2, an assistive robot planner has already been proposed, where the robot is asked to deliver tools to the human. Actually, this robot planner only plans fixed actions and it does not have other options. Although it improves the time efficiency as indicated in Section 3.3.5, it can be even better if we make the robot be more proactive, and let the robot do real assembly work. As shown in Fig.6.1, the proactive robot assembles the memory and participates in the real assembly while taking account of the human.

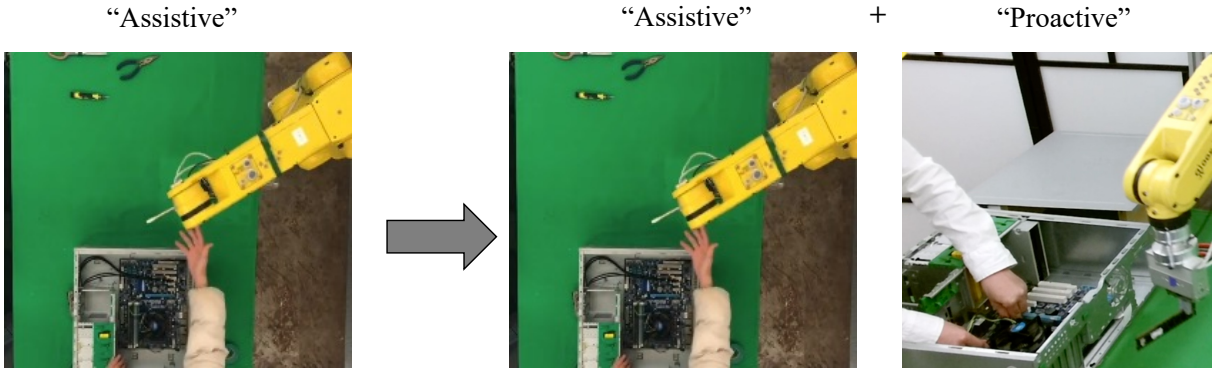


Figure 6.1: From assistive task planner to proactive task planner.

So far, there have been many research efforts on human-aware task planning. Some works such as [36] [47] [63] solve this problem by empowering a human worker or a centralized planner to be a supervisor. The existence of supervisors makes the system less flexible and makes the human more stressed. Other works do not include supervisors. They build a robot

planner which takes as input the human's actions, the human intention, environment states, task models, and etc., and enables the robot to collaborate as a teammate. The mechanism usually consists of two steps: offline obtaining task knowledge and online generating plans. In the literature, task knowledge can be acquired via engineering by experts [17] [49] [95] [18] [65] or learning from demonstration [44]. Online plans can be generated by techniques such as search algorithms [17] [49] [18] [65], optimization of the collaborative cost [95], or a Q-learning method [44]. In this chapter, we adopt this two-step mechanism, where task knowledge is extracted from human demonstration offline, and robot actions are planned by optimization online.

Task knowledge is often described by graphical task models. Depending on the structure of the models, they can be categorized as flat models, such as a plan network [49], or hierarchical models, such as and/or graphs [41] [11]. Hierarchical models have shown superiority over flat models for human-robot collaboration, because levels of abstraction in hierarchical models are close to human intuitions, which can help predict actions of the human and plan predictable actions of the robot. To our advantage, we propose a sequential/parallel task model which is a hierarchical task model. This model inherits the feature from the model in [79] that it explicitly models parallel relationships of subtasks and actions. To learn the hierarchical task models, Hayes and Scassellati in [34] proposed a conjugate task graph and an aggregation algorithm for the identification of the underlying structure of a task. Nevertheless, the introduction of a conjugate task graph shrinks the task space, which makes it less applicable to some use cases. We propose to extract the sequential/parallel task model using the idea of aggregation but without introducing conjugate graphs.

Optimization-based planners formulate the planning problem as finding a robot action from all feasible actions that minimizes the collaborative costs including factors such as completion time [30], human fatigue [50], spatial interfaces [30], and etc.. The more factors considered, the more complex the cost function becomes, and it is nontrivial to decide on the weights for each factor. With poor modeling, the planning results can be far below expectations. We propose an optimization-based planner, the objective of which is to minimize the task completion time. Different from other methods, we give priorities to actions that are parallel to the human's actions. The perspective is that it is worth sacrificing some completion time for conducting parallel actions, because this has potential benefits in reducing spatial interfaces and improving human satisfaction, which is not explicitly included in the objective function.

The main notation used in this chapter is summarized in Table 6.1.

6.2 A Hierarchical Task Model

Hierarchical task models surpass the flat models in the fields of human-robot collaboration in three ways. First, hierarchical task models provide more intuitive abstractions of the task for the robots, which can help improve the human-robot communication about the intermediate goals of the task [28] [66]. Second, since humans follow the hierarchical ab-

Table 6.1: Table of Notation.

m	$\in N$	Number of actions in a demonstration.
a	$\in$ Action space	Action.
ξ	$[a_1, a_2, ..., a_m]$	Action sequence.
n	$\in N$	Number of action sequences.
Ξ	$\{\xi_1, \xi_2, ..., \xi_n\}$	Demonstration set.
s	$\in \{0,1\}^m$	Execution indicator.
req$(\cdot)$	$\in \{0,1\}^m$	Requisite execution indicator.
prod$(\cdot)$	$\in \{0,1\}^m$	Resulting execution indicator.
C_r	$\in$ Action space	Robot capability.
C_h	$\in$ Action space	Human capability.
k	$\in N$	Planning horizon.
x_r	$\in \{0,1\}^k$	Robot task assignment vector.
x_h	$\in \{0,1\}^k$	Human task assignment vector.

stractions when accomplishing a task, hierarchical task models can help predict the human's actions [77] [72]. Finally, it has been found that such hierarchical abstractions can be learned remarkably fast from relatively little data compared with what is needed for learning at lower levels due to overhypotheses [90] [40] [82]. Thus, in this work, we adopt a hierarchical task model named a sequential/parallel task model. Fig. 6.2 shows an example of the sequential/parallel task model for the desktop assembly task. The root node represents the *task*, and all the other nodes represents *subtasks*. Leaf nodes are *atomic subtask*, also known as *actions*. Nodes can be categorized to the following three types according to the relationship among their child nodes.

- **Sequential nodes**: their child nodes must be executed in the order from left to right, which is denoted by the operator "$\rightarrow$". For example, the subtask "Install CPU fan" is a sequential node, and its child "obtain CPU fan" must be conducted before "Insert CPU fan".

- **Parallel nodes**: their child nodes can be executed in parallel, which is denoted by "$\|$". For example, the subtask "Install motherboard" is a parallel node, its children "install CPU fan", "install memory" and "Tape cables" can be executed simultaneously.

- **Independent nodes**: their child nodes can be executed in any orders, which is denoted by "$\perp$". Parallel nodes are a special case of independent nodes. For example, a root node is an independent node but not a parallel node. Its child nodes "Applying labels to hood" and "Assemble main body" have no fixed order, but they cannot be executed in parallel if "close hood" is in progress.

Following the description of [16] [56], actions can be defined as $a = [\{motion, object\}, attribute]$, where *motion* indicates types of the movement and *object* indicates the object of interaction.

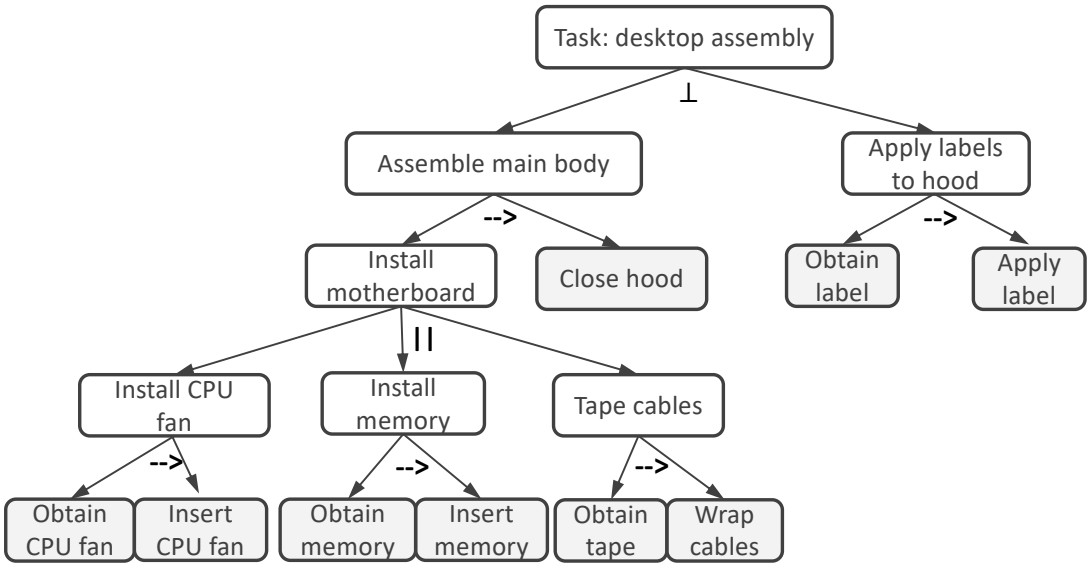


Figure 6.2: The sequential/parallel task model for a desktop assembly task.

The pair of *motion* (e.g. inserting) and *object* (e.g. CPU fan) is sufficient to distinguish different actions. In addition, the *attribute* contains information such as completion time, energy consumption, etc., which are useful in the planning process. For assembly tasks, the relationships of actions also take three forms.

- Two actions a and b are **independent** (written as $a \perp b$ or $b \perp a$) if the occurrence of one action does not rely on the occurrence of the other.
- Action a is **dependent** on the action b (written as $a \not\perp b$) if a can not occur in the absence of the occurrence of action b.
- Actions a and b are **parallel** (written as ab) if a and b are independent and they do not share objects of interaction.

An action a is dependent on the action b implies that b produces some consequences that a requires in order to be executed. We use the *execution indicator* to represent this connection, each component representing an action being executed or not. There are totally 9 actions in the example, and thus the indicator is $s \in \{0,1\}^9$. Note that for assembly tasks, the effects of several actions are addictive, thus indicator vectors can be used as procedural states of the assembly. Based on this, we define two types of indicator vectors, req$(\cdot)$ to be *requisite execution indicator* and prod$(\cdot)$ to be the *resulting execution indicator* of an action or an action sequence. In this sense, action a is dependent on action b ($a \not\perp b$) is equivalent to req$(a)\wedge$prod(b)=prod(b), where $\wedge$ is a bit-wise AND operator. In addition, that action d is dependent on action sequence bcd

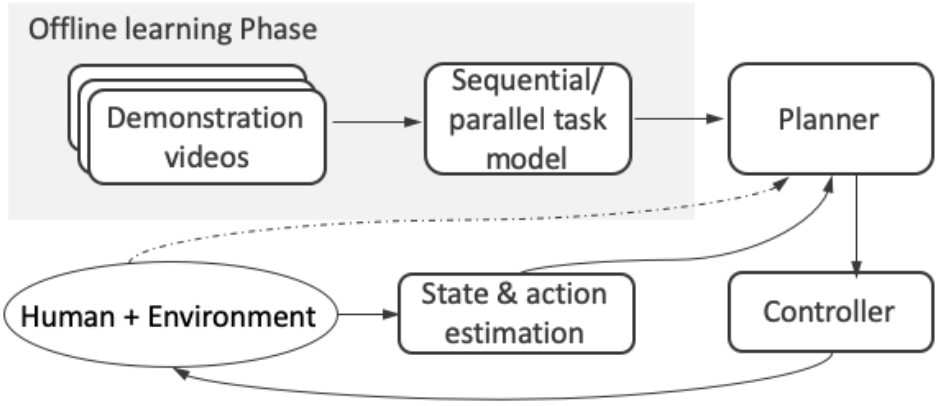


Figure 6.3: System overview.

($d \not\perp abc$) is equivalent to req(d)∧prod(abc)=prod(abc). As the effects of actions are *addictive*, prod(abc)=prod(a)+ prod(b)+ prod(c), hence req(d)∧prod(abc)=prod(abc) is equivalent to req(d)∧prod(a)=prod(a), req(d)∧prod(b)=prod(b) and req(d)∧prod(c)=prod(c). Besides, for actions with independent relationships, action a is independent of action b ($a \perp b$) if and only if req(a)∧prod(b)=$\mathbf{0}$ and req(b)∧prod(a)=$\mathbf{0}$. Action d and action sequence abc is independent if and only if req(d)∧prod(abc)=$\mathbf{0}$ and req(abc)∧prod(d)=$\mathbf{0}$. Similarly, by the addition characteristic of the effect, req(d)∧prod(abc)=$\mathbf{0}$ is equivalent to req(d)∧prod(a)=$\mathbf{0}$, req(d)∧prod(b)=$\mathbf{0}$ and req(c)∧prod(abc)=$\mathbf{0}$.

6.3 Approach

Our goal is to develop a human-aware robot task planner for industrial assembly scenarios. There are two aspects to be addressed: (i) learning the knowledge of assembly plans from human demonstrations, and (ii) designing a human-aware robot task planner using the task knowledge. Fig. 6.3 shows the proposed robotic system. We first learn a sequential/parallel task model from human demonstrations at the offline learning phase. This model represents the plans of a task with levels of abstraction, which provides information about the parallel relationships that could not be directly seen in a collection of action sequences. At the online execution phase, the planner takes as input the task model and human actions, then outputs the robot action command to the controller to execute it. Our planner leverages the parallel information in the sequential/parallel task model to guide the robot towards performing actions parallel to the human's action, while optimizing over task efficiency.

6.4 Automatically Constructing the Task Model

With the fast development in computer vision techniques, action recognition for videos can be accurate and fast using state-of-the-art methods such as [3] [58] [94]. Applying such techniques to make annotations to each human demonstration video, a set of action sequences $\Xi = \{\xi_1, \xi_2, ..., \xi_n\}$ can be obtained, where n is the number of different plans demonstrated and $\xi_i = [a_1^i, a_2^i, a_3^i, ..., a_{m^i}^i]$ is the i-th action sequence with horizon m^i. For industry assembly tasks, the number of steps for any possible plan is often the same, thus we omit the superscript of m^i for simplicity. The set Ξ is complete if and only if it includes all the possible plans.

To construct the sequential/parallel task model from Ξ, the key is to discover the independent relationships and sequential relationships at all levels of abstraction. Once the independent relationships are identified, recognizing parallel relationships is trivial by checking objects of interaction. In the following, we first discuss how to find a sequential relationship among actions, and second, we explain how to discover an independent relationship, and then we introduce a bottom-up algorithm which iteratively aggregates actions and forms a sequential/parallel task model. Finally, we discuss whether there is a requirement on the completeness of the demonstration set and then we display time complexity.

6.4.1 Sequential relationships

To identify the actions with sequential relationships is to find the longest common subsequence (LCS). The longest subsequence is common to all the given sequences $\xi_i \in \Xi, i \in \{1, 2, ..., n\}$, provided that the elements of the subsequence are required to occupy consecutive positions within the original sequences. By saying so, we impose an assumption that the sequential actions are not interrupted by parallel actions in human demonstrations. For example, "Apply labels to hood" cannot take place in between "Obtain CPU fan" and "Insert CPU fan". This is a weak assumption because reasonable humans follow this way when proceeding with tasks.

6.4.2 Independent relationships

Identifying the independent relationships is based on the following theorem.

Theorem 1 *When Ξ is complete, for a subsequence η of $\xi \in \Xi$, and suppose $\bar{\eta}$ is a reversed η, the following two statements are equivalent:*
(a) There exists a $\gamma \in \Xi$ such that $\bar{\eta}$ is a subseqence of γ.
(b) Each pair of actions in η are independent.

(a) "⇒"(b)

Suppose subsequence $\eta = [b_1, b_2, ..., b_m], m \geq 2$, then $\bar{\eta} = [b_m, b_{m-1}, ..., b_1]$. From η, we have

$$\begin{aligned} &req(b_1) \wedge prod(b_2 b_3 ... b_m) = \mathbf{0} \\ &req(b_2) \wedge prod(b_3 b_4 ... b_m) = \mathbf{0} \\ &... \\ &req(b_{m-1}) \wedge prod(b_m) = \mathbf{0}, \end{aligned}$$

and from $\bar{\eta}$, we have

$$\begin{aligned} &req(b_m) \wedge prod(b_{m-1} b_{m-2} ... b_1) = \mathbf{0} \\ &req(b_{m-1}) \wedge prod(b_{m-2} b_{m-3} ... b_1) = \mathbf{0} \\ &... \\ &req(b_2) \wedge prod(b_1) = \mathbf{0}. \end{aligned}$$

Using the addition rule, we have $req(b_i) \wedge prod(b_j) = \mathbf{0}$, where $i, j \in \{1, 2, ..., m\}$. Thus, $b_i \perp b_j, i \neq j \in \{1, 2, ..., m\}$, which indicates (b).
(b) "⇒"(a)
since $b_i \perp b_j, i \neq j \in \{1, 2, ..., m\}$, then $req(b_i) \wedge prod(b_j) = \mathbf{0}$, where $i, j \in 1, 2, ..., m$, therefore, any orders of $\{b_1, b_2, ..., b_m\}$ are feasible. Hence (a) holds.

Therefore, an independent relationship can be identified by checking each pair of the sequences $\{\xi_i, \xi_j\}, i \neq j \in \{1, 2, ..., n\}$. If there is a common subsequence of ξ_i and $\bar{\xi}_j$, where $\bar{\xi}_j$ is a reversed ξ_j, actions in this common subsequence are recognized as independent actions.

6.4.3 Algorithm

We propose the algorithm shown in Algorithm 4. It takes as input Ξ, the set of action sequences labeled from demonstrations, and outputs a table T, which stores the information about meta-actions. A meta-action is a fake action by compacting atomic actions or meta-actions into one, which corresponds to the intermediate node in the model. Line 1 initializes the table T to be empty. Lines 2-20 construct a hierarchical tree from bottom to top by iteratively discovering the sequential and independent relationships. Lines 3-8 identify the sequential relationship. First, the longest common subsequence ψ is found. If the length of ψ is greater than 1, the actions in subsequence are compacted to a meta-action A_m, subsequence ψ is replaced with A_m for every sequence in Ξ by the function *refresh*, and in the meantime repeated sequences in Ξ are discarded. Finally, the table T is updated by adding an entry of A_m with information of the relation type and the replaced subsequence ψ, where relation types 's', 'i' and 'p' denotes sequential, independent and parallel relationships. Lines 9-16 identify the independent relationships. The function *refresh* works differently in Line 13. Replacement takes place not for all sequences but for those sequences that contain the subsequence ψ or varying orders of subsequence ψ. The procedure of finding sequential and independent relationships alternates until T table remains the same. Finally, Lines 21-26 discover parallel relationships by checking the objects of interaction among independent

Algorithm 4 Sequential/parallel task model construction

Input Ξ
Output T
init $T = \emptyset$
while True **do**
  ψ = findLCS(Ξ)
  if $|\psi| > 1$ **then**
    create meta-action A_m
    Ξ.refresh(A_m)
    T.update(A_m,relation='s',ψ)
  end if
  for each pair $\{\xi_i, \xi_j\} \in \Xi, i \neq j$ **do**
    ψ = findLCS(reverse(ξ_i), ξ_j)
    if $|\psi| > 1$ **then**
      create meta-action A_m
      Ξ.refresh(A_m)
      T.update(A_m,relation='i',ψ)
    end if
  end for
  if T is not updated in this iteration **then** break
  end if
end while
for each entry t in T with relation='i' **do**
  A_o = retrieveAtomicActions(T, t.ψ)
  if actions in A_o share no objects of interaction **then**
    update t.relation='p'
  end if
end for

actions. Function *retrieveAtomicActions* retrieves all the atomic actions that a meta-action represents. Fig. 6.4 shows the T table and the updating demonstration set for the desktop assembly task when running this algorithm.

6.4.4 Demonstration set requirement

Our algorithm does not require that the demonstration set to be complete for successfully constructing the task model. Suppose the number of independent nodes in the sequential/parallel task graph is p, and the number of children for ith independent node is $l_i \in \mathcal{N}$ for $i = \{1, 2, ...p\}$. If $p \geq 1$, which is often true, the minimum number of different plans required by our algorithm is 2, while the number of all possible plans is $\prod_{i=1}^{p} l_i$. Since $l_i \geq 2$ and $p \geq 1$, thus $2 \leq 2^p \leq \prod_{i=1}^{p} l_i$ holds, indicating that complete demonstration set is not a

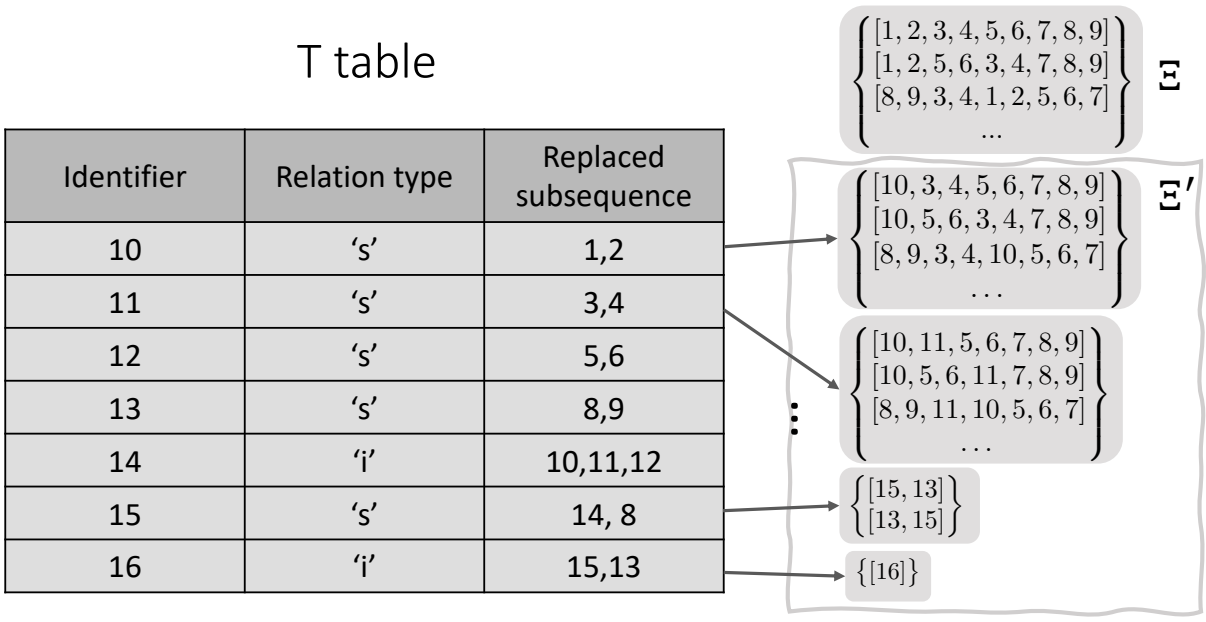


Figure 6.4: T table and the updating demonstration set $\Xi'$for the desktop assembly task when running algorithm 1. Action identifier 1-9 corresponds to the atomic action 'Obtain CPU fan', 'Insert CPU fan', 'Obtain memory', 'Insert memory', 'Obtain tape', 'Wrap cables', 'close hood', 'Obtain label' and 'Apply label'.

must for our algorithm.

6.4.5 Complexity

The dominating complexity factor throughout the algorithm is the discovery of the independent relationship. The step of finding the longest common subsequence can be accomplished in time $\mathcal{O}(m\log(m))$ using the algorithm in [9], where m is the length of the sequence, in our case the number of actions in a demonstration. Therefore, the time complexity of our algorithm is $\mathcal{O}(mn^2\log(m))$, where n is the number of demonstrations.

6.5 A Separation Planner

In this section, we will present a human-aware robot task planner. For simplicity, we assume that there is one robot and one human. However, with a slight modification, the algorithm can be applied to a team of multiple humans and multiple robots. The input of the algorithm includes sequential/parallel task model T, the execution indicator s, the capability of the human C_h and capability of robots C_r, which is a set of actions that the

human or the robot is capable of doing. The planner optimizes a plan, and then sends the robot's first action to the controller to execute, and plans again, repeatedly.

The key of our planner is the planning horizon, which is a collection of subtasks including: 1) the sequential subtask the human is currently conducting, and 2) the subtasks that are parallel to the human's current subtasks and are not executed yet, which is illustrated in Fig. 6.5. Thus, the planning horizon is part of the remaining task and it varies as the task proceeds. Since humans follow abstract hierarchies while doing a task, it is very likely that the human conducts the following actions in the sequential subtasks after finishing the current action, thus we assign the whole sequential subtask to the human to avoid the robot choosing the same action as the human's. The planning problem then becomes a scheduling problem, which is to assign those parallel subtasks to the human and the robot such that the completion time for the planning horizon is minimized. The optimization problem is formulated as follows.

$$
\begin{aligned}
\min_{x_h, x_r} \quad & t \\
\text{s.t.} \quad & x_h^T t_1 + t_o \leq t \\
& x_r^T t_2 \leq t \\
& x_h + x_r = \mathbf{1} \\
& x_h, x_r \in \{0,1\}^k \\
& x_{h\{C-C_h\}} = 0 \\
& x_{r\{C-C_r\}} = 0
\end{aligned}
\tag{6.1}
$$

Decision variables x_h and $x_r \in \{0,1\}^k$ are binary vectors for the assignment of subtasks, where k is the number of subtasks in the planning horizon. The objective function t is the completion time for the planning horizon, which is the upper bound of the human's completion time and the robot's completion time. t_o is the remaining time for the human to complete his current subtask. t_1 and t_2 are the empirical completion time for subtasks, which are obtained based on *Attribute* of an action. $\{C - C_h\}$ denotes the indices of the actions that the human is unable to do, and $\{C - C_r\}$ denotes the indices of the actions that the robot is unable to do.

This planner is suitable to be integrated into a robot system which has two modes of interaction. One mode is the "command" mode, where the human explicitly asks the robot to perform an action through communication. The other mode is the "automation" mode. The human and the robot collaboratively do tasks without communication. Our planner can work under this "automation" mode, and it is switched to the other planner when humans make commands.

Figure 6.5: Two examples of planning horizons. These are two sequential/parallel task model. Red edges are with sequential relationships and black edges are with independent relationships. "done" and "isDoing" in the atomic action node represents that the action is executed already and that the action is being executed by the human. The subtasks with orange shades are the human's current subtasks, and the subtasks with blue shades are the parallel subtasks to the human's current subtasks. These two kinds of subtasks constitute the planning horizon.

6.6 Simulations and Experiments

6.6.1 Three Scenarios

We use the following three different scenarios to decrease the impact of different tasks for a fair comparison.

A desktop assembly scenario

We evaluate our proposed planning algorithm in a desktop assembly task whose sequential/parallel task model is shown in Fig. 6.2.

Two book shelving scenarios

We device two book shelving scenarios, corresponding to the two models in Fig. 6.5. The initial state of the task is that the shelf is empty, and the goal state is shown in Fig. 6.6. The difference of the two scenarios is the constraints on the shelving order. For the first scenario, books must be shelved from bottom to top, which means that books must be shelved at 1st layer first and then 2nd layer and finally 3rd layer. This constraint still holds to small layers such as the left two layers at the 1st layer. This task corresponds to the left model in Fig. 6.5. For the second scenario, there is no constraint on the vertical axis, however, within each layer, books must be shelved from left to right. This corresponds to the right model in Fig. 6.5.

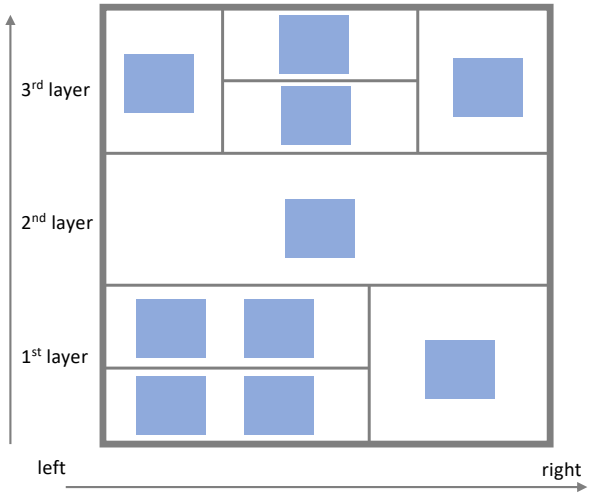


Figure 6.6: The book shelving scenarios. The solid boxes represent books, and the gray frames constitute the bookshelf.

6.6.2 Hypothesis

We evaluate the effectiveness of the proposed planning algorithm through simulations by verifying the following four hypotheses.

- H1: The proposed algorithm generates efficient plans.
- H2: The proposed algorithm generates safe plans.
- H3: The performance of the proposed algorithm is robust to different levels of incompleteness in demonstrations.
- H4: The human subjects are more satisfied with our collaborative robot planning than with other baseline planners.

6.6.3 Simulation setup

We test our algorithm on a simulator built in MATLAB. Human subjects manipulate the objects by using a mouse, dragging and dropping the objects with the mouse bottom held down and released. Complex actions such as inserting and wrapping in the desktop assembly scenario are simplified by dropping the objects. The robot actions are expressed by the object moving and a sentence shown in the interaction window. The capability of both the human and the robot is set to the whole action space for all the tasks.

Six human subjects participated in the simulation. After being instructed about the task and the simulation environment, they operated each action for practice, and that was when the completion time of actions for humans was collected. For robots, the completion time of actions is manually set in the simulations.

6.6.4 Instructions to the human subjects

First, we teach the task knowledge to the human subjects. The task knowledge includes: (1) the goals of the tasks, (2) which subtasks can be executed in random orders, and (3) which subtasks must be executed in certain orders. Second, we explain the simulation interfaces, and elaborate how to complete a task by moving a mouse and dragging a part in the interface. Take computer assembly for example. In the simulation interface shown in Fig. 6.7, parts that need to be installed or manipulated are represented by solid rectangles, and the dashed boxes are where they should be installed or manipulated. All the assembly procedures are simplified by locating, clicking, dragging, and releasing the part in the simulation. As long as you drag a part inside the corresponding dashed box, that subtask is considered done. For example, to do the subtask “install CPU fan”, you should drag the green rectangle into the dashed box annotated with “fan”, and to conduct the subtask “closing hood”, you should drag the gray solid rectangle inside the dashed box annotated with “computer case”. Based on the mouse behavior operated by the human subjects, our algorithm computes the “robot’s” task planning and visualizes the “robot’s” action by making the part go to

the designated area automatically. For example, if the "robot's" action is "installing the memory", the solid red rectangle goes to the dashed box annotated with "memory" by itself. We also notice the human subjects that before they start to do the task, they should put the mouse in the square box annotated with "human starts here", so that once the mouse leaves that square, the timer starts and our algorithm begins to track the mouse behavior and recognizes human's intentions and actions. Finally, we let the human subjects practice for a while, and we observe and correct them if they make mistakes.

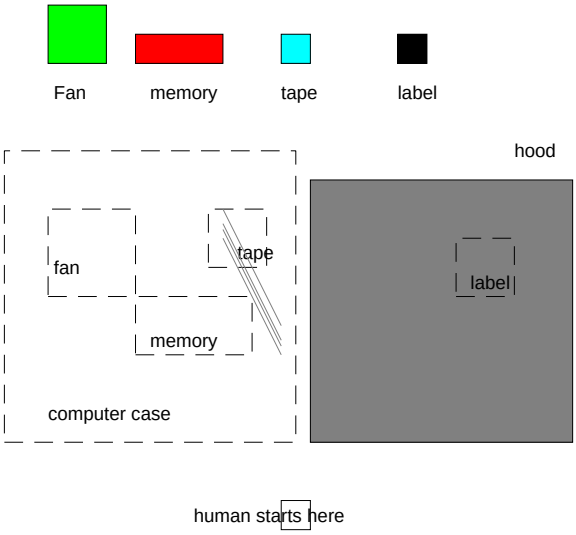


Figure 6.7: Simulation interface of desktop computer assembly task.

6.6.5 Manipulated variables

To evaluate the effectiveness of the proposed algorithm, we manipulated two controlled variables in our simulations. The first one is *planing schemes*. Our algorithm is compared with the following three baselines:

- Random policy: This planner models the task as a set of action sequences, which is a single-level plan library. It tries to recognize the human's plan by aligning the current procedure/ action sequence to the action sequences in the plan library, and chooses the most likely action sequences as the human's intended plans, which can be many. After that, the robot action is generated uniformly at random from the set of next actions of those possible human's plans.

- Anticipatory planning [44]: This algorithm models the human's and the robot's behavior with an MDP and learns the policy from demonstrations by reinforcement learning.

At the execution time, it also models that humans may not take the optimal action by predicting the ratio of the human taking optimal action and human following habits.

- Shortest completion time optimizer: Similar to our algorithm, this algorithm also aims to minimize the completion time, but the planning horizon is the whole task to go.

The other manipulated variable is *level of incompleteness* of the demonstration set. The levels are set to be 0%, 30%, 50% which means these percentages of action sequences in the demonstration set will be randomly dismissed.

Since the third baseline assumes the task model is already known, it does not rely on the demonstration set. Thus, by manipulating the two variables, we have 10 groups of simulations for each scenario. Under each group, every human subject performs the task using any plan twice. Thus, there will be 20 trials in each simulation group and in total 120 trials for each scenario.

To have a fair comparison, all the algorithms use the same sets of action sequences as inputs under all the incompleteness conditions for each scenario. For the hyperparameters in the algorithm of anticipatory planning, grid search cross-validation is utilized to choose the hyperparameters that perform the best.

6.6.6 Dependent measures

To quantify the safety of the proposed framework, we measure the percentage of robot actions that conflict with the human's. For efficiency, we set a timer to keep track of the task completion time. The timer starts when a human subject starts to act, and ends when the task is finished. As for the measurement of human's satisfaction with our collaborative robots, we ask the six human subjects to rate the following statements on Likert scale from 1 (strongly disagree) to 5 (strongly agree), similar to [44]:

- The robot was collaborative and helped;
- The robot did the right thing at the right time;
- I am satisfied working with the robot;
- I will work with this robot again in the future.

6.6.7 Implementation Details

The desktop assembly task includes a action space ($\{a \mid a \in N, 1 \leq a \leq 9\}$) as illustrated in Fig 6.2. C_h is set as the whole action space, and C_r is set as the whole action space excluding the "closing hood" action ($\{a \mid a \in N, 1 \leq a \leq 9, a \neq 7\}$). For the book shelving task, both C_h and C_r are the whole action space ($\{a \mid a \in N, 1 \leq a \leq 10\}$). We use MATLAB function *intlinprog* to solve the optimization problem 6.1.

6.6.8 Results

H1: Table 6.2 shows the average task completion time for the three scenarios using different planners. Our planner is significantly more efficient than the planners of random policy and anticipatory planning under all situations[1] ($p < 0.01$). Compared with the shortest completion time optimizer, our planner can achieve similar results, as the average task completion time is close and there is no significant difference among all trials ($p > 0.1$). Possible reasons are that:

- Although shortest time optimizer computes the most efficient plan, the human subject may not follow that plan;
- Shortest time optimizer does not consider humans' preference of doing the next actions with a sequential relationship. Thus, this planner is more likely to cause conflicts, which increases the task completion time.

The random policy is most inefficient. Opposite to our algorithm, the random policy has no knowledge of the parallel relationships, and it strictly plans the next actions of the current plan, which results in a lot of waiting time. Compared with random policy, anticipatory planning has a shorter task completion time, as it enables the robot to anticipate more in the collaboration. However, its limitation is that it utilizes learned policies from single-agent demonstrations. In the two-agent scenarios, humans can follow a different reward model, which makes the learned policies less effective. Although this planner tries to adapt to the human on the fly, it takes time.

H2: The average percentages of conflicts are $2.7 \pm 0.2\%$, $15.0 \pm 0.7\%$, $8.1 \pm 0.4\%$ and $6.2 \pm 0.2\%$ for our algorithm, random policy, anticipatory planning and shortest completion time optimizer correspondingly, as shown in the Fig. 6.8. Our planner is the safest planner among the other planners, as the robot conflicts with the human the least with significant difference ($p < 0.01$). The planner guides the robot to do subtasks that are parallel to the human's current action, which naturally decreases the possibility of conflicts. This result also validates the assumption stated in Section 6.4 that humans complete sequential actions consecutively without switching to another parallel subtask. The planner with random policy is the most unsafe one, as it causes conflicts the most with significant difference ($p < 0.01$). This planner has no knowledge of the parallel relationships and it always plans the next actions of the current recognized plan. However, when humans are conducting subtasks with sequential actions, they also prefer to do the next actions, which can cause conflicts. The planners with anticipatory planning and shortest completion time optimizer do not explicitly utilize parallel information, nor do they explicitly utilize sequential information, and their results are in-between.

H3: Through various simulations on different levels of incompleteness of demonstrations, we show that our method consistently delivers comparable performance as shown in Table 6.2. The average task completion times do not increase as the level of the incompleteness

[1]We use paired t-test for all the statistical tests.

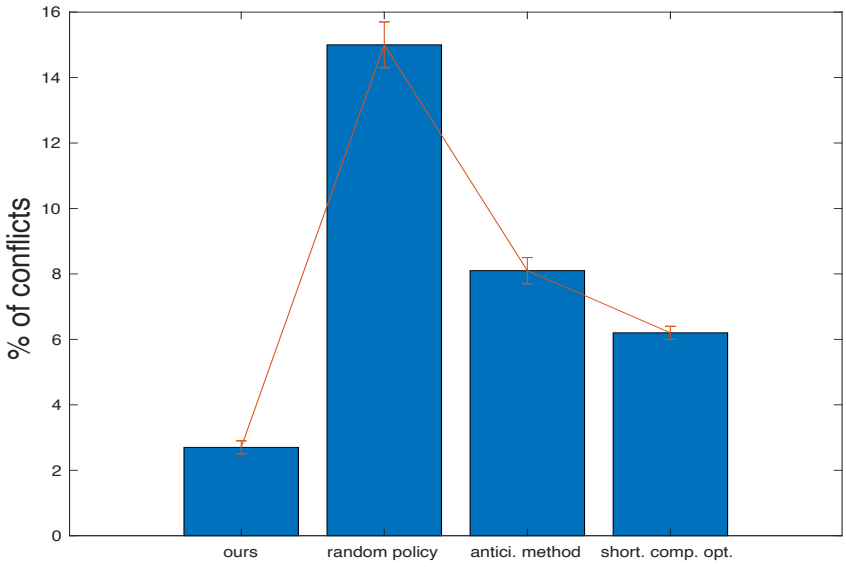


Figure 6.8: The average percentages of conflicts for our algorithm, random policy, anticipatory planning and shortest completion time optimizer.

increases. For example, when 30% of the demonstrations are deleted from the desktop assembly scenario, the average task completion time even drops by 0.8 seconds. Besides, through statistical tests, we find that there is no significant difference in the task completion time and percentage of conflicts among all levels of incompleteness for each scenario ($p < 0.01$). Actually, the sequential/parallel task models retrieved from the demonstration set are the same for all the tasks with different incompleteness levels. Thus, our algorithm is robust to the incomplete demonstrations.

H4: Fig. 6.9 shows the comparison of human subjects' ratings for the four types of planners on the four criteria: 1. The robot was collaborative and helped; 2. The robot did the right thing at the right time; 3. I am satisfied working with the robot; 4. I will work with this robot again in the future. Human subjects rated the our planner significantly higher than the other planners on all four criteria ($p < 0.01$). This indicates that human subjects are more satisfied with our planner. It is also interesting to note that scores of the anticipatory planning are significantly higher than that of random policy ($p < 0.01$), and scores of shortest completion time optimizer are significantly higher than that of anticipatory planning ($p < 0.01$).

Table 6.2: The result table for the average task completion time (unit: second). M1, M2 and M3 are methods of random policy, anticipatory planning and optimization respectively.

Incompleteness	**Desktop assembly Scenario**				**book shelving scenario 1**				**book shelving scenario 2**			
	ours	M1	M2	M3	ours	M1	M2	M3	ours	M1	M2	M3
0%	**8.3**	12.6	10.0	8.5	16.6	23.7	18.9	**15.5**	**13.2**	20.6	17.7	14.1
30%	**7.5**	15.0	13.4	-	**15.7**	25.2	21.7	-	**14.5**	27.4	25.3	-
50%	**8.1**	15.9	15.8	-	**17.5**	28.1	27.5	-	**15.1**	28.0	27.4	-

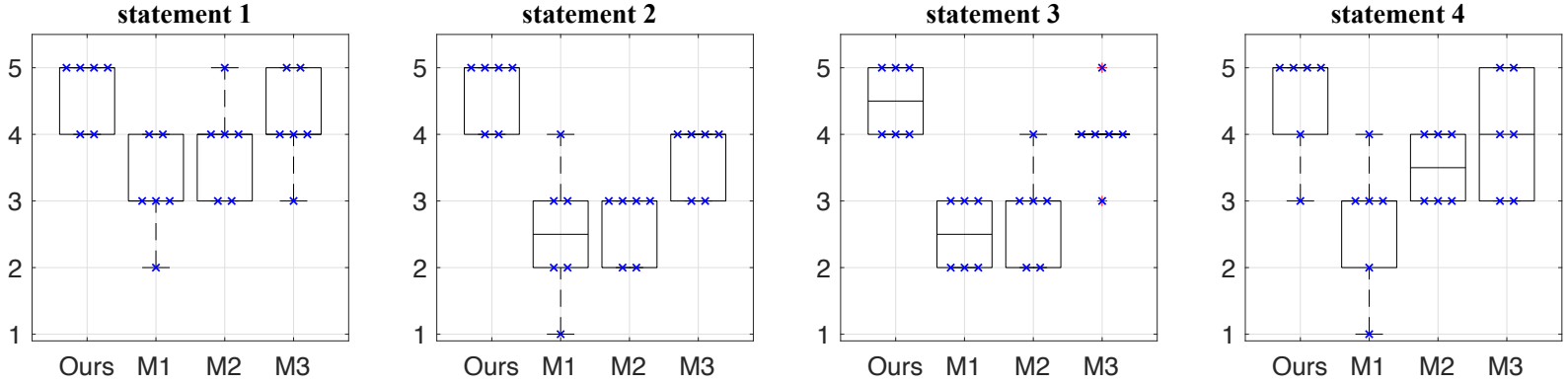


Figure 6.9: Human subjects' ratings for the four robot planners on four different statements. M1, M2 and M3 are for planners of random policy, anticipatory planning and optimization correspondingly. Score 1 to 5 corresponds to 1. strongly disagree, 2. disagree, 3. neutral, 4. agree and 5. strongly agree.

6.6.9 Experiments

Figure 6.10: A robot and a human collaborate to assemble a desktop.

We test our planner on an industrial robot FANUC LR Mate 200iD/7L. A Kinect V2 for windows is placed close to the table for detecting the human and the objects. Fig. 6.10 shows that the robot is collaborating with the human to complete the desktop assembly task, where the human is assembling the CPU fan, while the robot is going to assemble the memory based on the plan computed by our planner. The video of the experiment is available at: https://msc.berkeley.edu/research/serocs.html.

6.7 Conclusion

To solve the collaborative task planning problem, we proposed an algorithm to automatically construct a hierarchical task model from human demonstrations, which captures the sequential and parallel relationships of the task at all levels of abstraction. This algorithm is very robust to the levels of incompleteness of demonstration. Actually, for any task, two specific demonstrations are enough to retrieve the sequential/parallel task model. We then proposed an optimization-based planner, which exploits the parallel relationships in the task model and gives priorities to the actions that are parallel to the humans', since conducting parallel actions can reduce spatial interfaces, reduce task completion time and improve human satisfaction. These benefits were verified through various simulations for three different scenarios when compared with other three baselines.

Chapter 7

Conclusion

In this dissertation, the goal is to achieve a safe and efficient human-robot collaboration. Specifically, two problems are studied:

- To guarantee human **safety**, robots should predict the human trajectory so that they can plan their own trajectory to avoid potential collisions.
- To boost task **efficiency**, robots should recognize the human plans and intentions so that they can make plans accordingly.

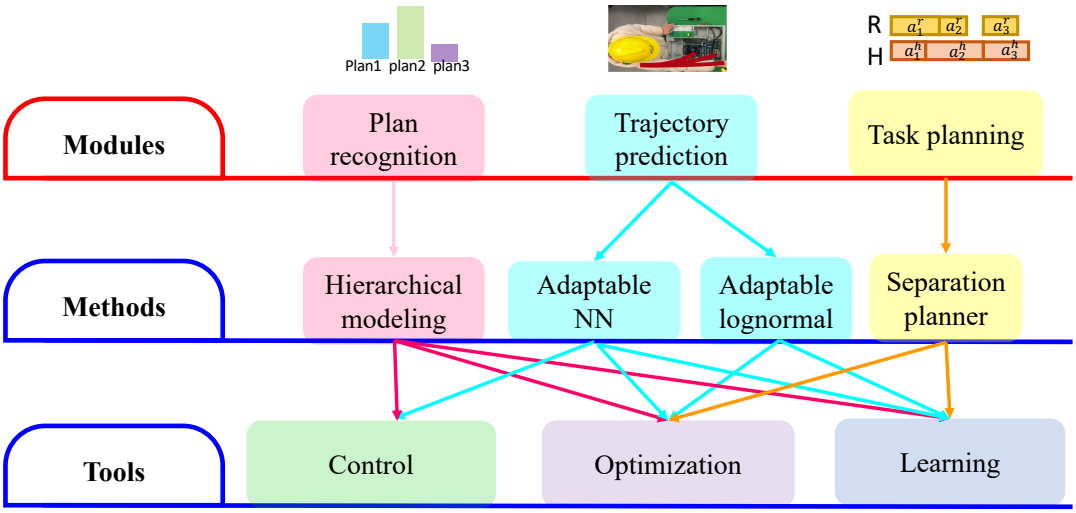


Figure 7.1: Summary of the dissertation work.

A robot system, as shown in Fig. 1.2, is proposed, where the key modules are **human plan recognition**, **human trajectory prediction** and **human-aware task planning**. The main content of the dissertation is focused on these three modules, outlined by Fig. 7.1.

Human plan recognition and human trajectory prediction are for reasoning about human behavior. The fundamental challenge for reasoning about human behavior is that there is no sufficient human data to learn human behavior. In order to learn from limited human data, a hierarchical modeling method and two adaptation algorithms are proposed for human plan recognition and human trajectory prediction, respectively. For human plan recognition, instead of learning from end-to-end, the hierarchical modeling method decouples the problem to learn the trajectory, the trajectory type, the action, and finally the plan from bottom to top. This decreases the dimension of the problem and reduces the data pressure. For trajectory prediction, the adaptable neural network method and the adaptable lognormal method deal with the short-term and long-term prediction, respectively. Since human data is insufficient, some patterns of human behavior may be missing, and we could not learn a universal model offline, so online we continue to adjust our model to suit different human behavior by using adaptation algorithms. Comparing the two adaptable methods, the adaptable neural network can be applied to collision avoidance with the capability to handle abnormal human activities; the adaptable lognormal method estimates the durations for humans' actions/whole trajectories, which is of great use for the task planners that optimize the task time.

Human-aware task planning plans robot actions while taking account of human actions. A separation planner is proposed. Researchers often separate humans and robots at the trajectory level to ensure safety, but it is the first planner that separates the human and the robot at the task planning level, and it is shown that this planner can achieve good time efficiency and good human satisfaction.

Bibliography

[1] Alexandre Alahi et al. "Social lstm: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 961–971.

[2] Beatrice Alenljung et al. "User experience in social human-robot interaction". In: *Rapid automation: Concepts, methodologies, tools, and applications*. IGI Global, 2019, pp. 1468–1490.

[3] Helena de Almeida Maia et al. "Action Recognition in Videos Using Multi-stream Convolutional Neural Networks". In: *Deep Learning Applications*. Springer, 2020, pp. 95–111.

[4] Ben Athiwaratkun and Keegan Kang. "Feature representation in convolutional neural networks". In: *arXiv preprint arXiv:1507.02313* (2015).

[5] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. "Action understanding as inverse planning". In: *Cognition* 113.3 (2009), pp. 329–349.

[6] Chris L Baker and Joshua B Tenenbaum. "Modeling human plan recognition using Bayesian theory of mind". In: *Plan, activity, and intent recognition: Theory and practice* 7 (2014), pp. 177–204.

[7] Lucian Balan and Gary M Bone. "Real-time 3D collision avoidance method for safe human and robot coexistence". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 276–282.

[8] Christoph Bartneck and Jodi Forlizzi. "A design-centred framework for social human-robot interaction". In: *RO-MAN 2004. 13th IEEE international workshop on robot and human interactive communication (IEEE Catalog No. 04TH8759)*. IEEE. 2004, pp. 591–594.

[9] Ratul Bhowmick et al. "An approach for improving complexity of longest common subsequence problems using queue and Divide-and-Conquer method". In: *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*. IEEE. 2019, pp. 1–5.

[10] Kyle EC Booth et al. "Mixed-integer and constraint programming techniques for mobile robot task planning". In: *IEEE Robotics and Automation Letters* 1.1 (2016), pp. 500–507.

[11] Taylor Boyd et al. "Hierarchical Task Learning Through Human Demonstration". In: (2019).

[12] Allison Bruce and Geoffrey Gordon. "Better Motion Prediction for People-tracking". In: *Proc. of the Int. Conf. on Robotics & Automation (ICRA), Barcelona, Spain*. 2004.

[13] Wolfram Burgard et al. "Coordinated multi-robot exploration". In: *IEEE Transactions on robotics* 21.3 (2005), pp. 376–386.

[14] Micah Carroll et al. "On the utility of learning about humans for human-ai coordination". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 5174–5185.

[15] J Frederico Carvalho et al. "Long-term Prediction of Motion Trajectories Using Path Homology Clusters." In: *IROS*. 2019, pp. 765–772.

[16] Yujiao Cheng et al. "Towards Efficient Human-Robot Collaboration With Robust Plan Recognition and Trajectory Prediction". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2602–2609.

[17] Marcello Cirillo, Lars Karlsson, and Alessandro Saffiotti. "Human-aware task planning for mobile robots". In: *2009 International Conference on Advanced Robotics*. IEEE. 2009, pp. 1–7.

[18] Marcello Cirillo, Lars Karlsson, and Alessandro Saffiotti. "Human-aware task planning: An application to mobile robots". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 1.2 (2010), pp. 1–26.

[19] Enric Corona et al. "Context-aware human motion prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 6992–7001.

[20] Eva Coupeté, Fabien Moutarde, and Sotiris Manitsaris. "Gesture recognition using a depth camera for human robot collaboration on assembly line". In: *Procedia Manufacturing* 3 (2015), pp. 518–525.

[21] Debasmit Das and CS Lee. "Cross-scene trajectory level intention inference using gaussian process regression and naive registration". In: (2018).

[22] Sandra Devin, Aurélie Clodic, and Rachid Alami. "About decisions during human-robot shared plan achievement: Who should act and how?" In: *International Conference on Social Robotics*. Springer. 2017, pp. 453–463.

[23] Myron A Diftler et al. "Robonaut 2-the first humanoid robot in space". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2178–2183.

[24] Salim Djeziri et al. "Learning handwriting with pen-based systems: computational issues". In: *Pattern Recognition* 35.5 (2002), pp. 1049–1057.

[25] B Farnsworth. *Human Behavior: The Complete Pocket Guide*. 2019.

[26] Tamar Flash and Neville Hogan. "The coordination of arm movements: an experimentally confirmed mathematical model". In: *Journal of neuroscience* 5.7 (1985), pp. 1688–1703.

[27] Garipelli Gangadhar, Denny Joseph, and V Srinivasa Chakravarthy. "An oscillatory neuromotor model of handwriting generation". In: *International journal of document analysis and recognition (ijdar)* 10.2 (2007), pp. 69–84.

[28] Andrew Garland, Kathy Ryall, and Charles Rich. "Learning hierarchical task models by defining and refining examples". In: *Proceedings of the 1st international conference on Knowledge capture*. ACM. 2001, pp. 44–51.

[29] Partha Ghosh et al. "Learning human motion models for long-term predictions". In: *3D Vision (3DV), 2017 International Conference on*. IEEE. 2017, pp. 458–466.

[30] Matthew C Gombolay, Ronald J Wilcox, and Julie A Shah. "Fast scheduling of robot teams performing tasks with temporospatial constraints". In: *IEEE Transactions on Robotics* 34.1 (2018), pp. 220–239.

[31] Graham C Goodwin and Kwai Sang Sin. *Adaptive filtering prediction and control*. Courier Corporation, 2014.

[32] Anand Gopalakrishnan et al. "A neural temporal model for human motion prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12116–12125.

[33] Ankur Gupta et al. "3D pose from motion for cross-view action recognition via non-linear circulant temporal encoding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 2601–2608.

[34] Bradley Hayes and Brian Scassellati. "Autonomously constructing hierarchical task networks for planning and human-robot collaboration". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 5469–5476.

[35] Dirk Helbing and Peter Molnar. "Social force model for pedestrian dynamics". In: *Physical review E* 51.5 (1995), p. 4282.

[36] Lars Johannsmeier and Sami Haddadin. "A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes". In: *IEEE Robotics and Automation Letters* 2.1 (2016), pp. 41–48.

[37] Mrinal Kalakrishnan et al. "STOMP: Stochastic trajectory optimization for motion planning". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 4569–4574.

[38] Yue Kang, Danilo Alves de Lima, and Alessandro Corrêa Victorino. "Dynamic obstacles avoidance based on image-based dynamic window approach for human-vehicle interaction". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2015, pp. 77–82.

[39] Jagat Narain Kapur and Hiremaglur K Kesavan. "Entropy optimization principles and their applications". In: *Entropy and energy dissipation in water resources*. Springer, 1992, pp. 3–20.

[40] Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. "Learning overhypotheses with hierarchical Bayesian models". In: *Developmental science* 10.3 (2007), pp. 307–321.

[41] Ross A Knepper et al. "Distributed assembly with and/or graphs". In: *Workshop on AI Robotics at the Int. Conf. on Intelligent Robots and Systems (IROS)*. 2014.

[42] Will Knight. "Smart robots can now work right next to auto workers". In: *MIT Technology Review* 17 (2013).

[43] Markus Kohler et al. *Using the Kalman filter to track human interactive motion: modelling and initialization of the Kalman filter for translational motion*. Dekanat Informatik, Univ., 1997.

[44] Hema S Koppula, Ashesh Jain, and Ashutosh Saxena. "Anticipatory planning for human-robot teams". In: *Experimental Robotics*. Springer. 2016, pp. 453–470.

[45] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. "Motion graphs". In: *ACM SIGGRAPH 2008 classes*. ACM. 2008, p. 51.

[46] Chiara Talignani Landi et al. "Prediction of human arm target for robot reaching movements". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019.

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

[48] Kenneth Levenberg. "A method for the solution of certain non-linear problems in least squares". In: *Quarterly of applied mathematics* 2.2 (1944), pp. 164–168.

[49] Steven James Levine and Brian Charles Williams. "Concurrent plan recognition and execution for human-robot teams". In: *Twenty-Fourth International Conference on Automated Planning and Scheduling*. 2014.

[50] Kai Li et al. "Sequence planning considering human fatigue for human-robot collaboration in disassembly". In: *Procedia CIRP* 83 (2019), pp. 95–104.

[51] Changliu Liu and Masayoshi Tomizuka. "Robot Safe Interaction System for Intelligent Industrial Co-Robots". In: *arXiv preprint arXiv:1808.03983* (2018).

[52] Changliu Liu and Masayoshi Tomizuka. "Safe exploration: Addressing various uncertainty levels in human robot interactions." In: *ACC*. 2015, pp. 465–470.

[53] Changliu Liu et al. "SERoCS: Safe and Efficient Robot Collaborative Systems for Next Generation Intelligent Industrial Co-Robots". In: *arXiv preprint arXiv:1809.08215* (2018).

[54] Hongyi Liu and Lihui Wang. "Gesture recognition for human-robot collaboration: A review". In: *International Journal of Industrial Ergonomics* 68 (2018), pp. 355–367.

[55] Hongyi Liu and Lihui Wang. "Latest developments of gesture recognition for human–robot collaboration". In: *Advanced Human-Robot Collaboration in Manufacturing*. Springer, 2021, pp. 43–68.

[56] Jingen Liu, Benjamin Kuipers, and Silvio Savarese. "Recognizing human actions by attributes". In: *CVPR 2011*. IEEE. 2011, pp. 3337–3344.

[57] Jun Liu et al. "Global context-aware attention lstm networks for 3d action recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1647–1656.

[58] Li Liu et al. "Deep learning for generic object detection: A survey". In: *International journal of computer vision* 128.2 (2020), pp. 261–318.

[59] Ruikun Luo, Rafi Hayne, and Dmitry Berenson. "Unsupervised early prediction of human reaching for human–robot collaboration in shared workspaces". In: *Autonomous Robots* 42.3 (2018), pp. 631–648.

[60] Riccardo Maderna et al. "Robust real-time monitoring of human task advancement for collaborative robotics applications". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 11094–11100.

[61] Julieta Martinez, Michael J Black, and Javier Romero. "On human motion prediction using recurrent neural networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 4674–4683.

[62] Laëtitia Matignon, Abir Beatrice Karami, and Abdel-Illah Mouaddib. "A model for verbal and non-verbal human-robot collaboration". In: *2010 AAAI fall symposium series*. 2010.

[63] George Michalos et al. "A method for planning human robot shared tasks". In: *CIRP journal of manufacturing science and technology* 22 (2018), pp. 76–90.

[64] Mohammad Mehdi Moniri et al. "Human gaze and focus-of-attention in dual reality human-robot collaboration". In: *2016 12th International Conference on Intelligent Environments (IE)*. IEEE. 2016, pp. 238–241.

[65] Vincent Montreuil et al. "Planning human centered robot activities". In: *2007 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. 2007, pp. 2618–2623.

[66] Giulio Mori, Fabio Paternò, and Carmen Santoro. "CTTE: support for developing and analyzing task models for interactive system design". In: *IEEE Transactions on software engineering* 28.8 (2002), pp. 797–813.

[67] Snehasis Mukherjee, Leburu Anvitha, and T Mohana Lahari. "Human Activity Recognition in RGB-D Videos by Dynamic Images". In: *arXiv preprint arXiv:1807.02947* (2018).

[68] Stefanos Nikolaidis and Julie Shah. "Human-robot teaming using shared mental models". In: *ACM/IEEE HRI* (2012).

[69] Stefanos Nikolaidis et al. "Planning with verbal communication for human-robot collaboration". In: *ACM Transactions on Human-Robot Interaction (THRI)* 7.3 (2018), pp. 1–21.

[70] Christian O'Reilly and Réjean Plamondon. "Development of a Sigma–Lognormal representation for on-line signatures". In: *Pattern recognition* 42.12 (2009), pp. 3324–3337.

[71] Oskar Palinko et al. "Robot reading human gaze: Why eye tracking is better than head tracking for human-robot collaboration". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 5048–5054.

[72] Mingtao Pei, Yunde Jia, and Song-Chun Zhu. "Parsing video events with goal inference and intent prediction". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 487–494.

[73] Zhen Peng, Tim Genewein, and Daniel Alexander Braun. "Assessing randomness and complexity in human motion trajectories through analysis of symbolic sequences". In: *Frontiers in human neuroscience* 8 (2014), p. 168.

[74] Rejean Plamondon. "The design of an on-line signature verification system: from theory to practice". In: *International Journal of Pattern Recognition and Artificial Intelligence* 8.03 (1994), pp. 795–811.

[75] Rejean Plamondon, Chunhua Feng, and Anna Woch. "A kinematic theory of rapid human movement. Part IV: a formal mathematical proof and new insights". In: *Biological cybernetics* 89.2 (2003), pp. 126–138.

[76] Réjean Plamondon et al. "Recent developments in the study of rapid human movements with the kinematic theory: Applications to handwriting and signature synthesis". In: *Pattern recognition letters* 35 (2014), pp. 225–235.

[77] Siyuan Qi et al. "Predicting human activities using stochastic grammar". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1164–1172.

[78] Harish Chaandar Ravichandar and Ashwin P Dani. "Human Intention Inference Using Expectation-Maximization Algorithm With Online Model Learning." In: *IEEE Trans. Automation Science and Engineering* 14.2 (2017), pp. 855–868.

[79] Alessandro Roncone, Olivier Mangin, and Brian Scassellati. "Transparent role assignment and task allocation in human robot collaboration". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1014–1021.

[80] Andrey Rudenko et al. "Human motion trajectory prediction: A survey". In: *The International Journal of Robotics Research* 39.8 (2020), pp. 895–935.

[81] Dorsa Sadigh et al. "Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state". In: *Autonomous Robots* 42.7 (2018), pp. 1405–1426.

[82] Ruslan Salakhutdinov, Joshua Tenenbaum, and Antonio Torralba. "One-shot learning with a hierarchical nonparametric bayesian model". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 2012, pp. 195–206.

[83] George AF Seber and Christopher John Wild. "Nonlinear regression. hoboken". In: *New Jersey: John Wiley & Sons* 62 (2003), p. 63.

[84] Ali Sharif Razavian et al. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.

[85] Tomas Simon et al. "Hand keypoint detection in single images using multiview bootstrapping". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 1145–1153.

[86] Liting. Sun, Wei. Zhan, and Masayoshi. Tomizuka. "Probabilistic Prediction of Interactive Driving Behavior via Hierarchical Inverse Reinforcement Learning". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Nov. 2018, pp. 2111–2117.

[87] Jaeyong Sung et al. "Human activity detection from RGBD images". In: *Workshops at the twenty-fifth AAAI conference on artificial intelligence*. 2011.

[88] Kartik Talamadupula et al. "Coordination in human-robot teams using mental modeling and plan recognition". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2957–2962.

[89] Yongyi Tang et al. "Long-term human motion prediction by modeling motion context and enhancing motion dynamic". In: *arXiv preprint arXiv:1805.02513* (2018).

[90] Joshua B Tenenbaum et al. "How to grow a mind: Statistics, structure, and abstraction". In: *science* 331.6022 (2011), pp. 1279–1285.

[91] Paolo Tormene et al. "Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation". In: *Artificial intelligence in medicine* 45.1 (2009), pp. 11–34.

[92] Nikolaus F Troje. "Decomposing biological motion: A framework for analysis and synthesis of human gait patterns". In: *Journal of vision* 2.5 (2002), pp. 2–2.

[93] Javier Vargas et al. "Kinect sensor performance for Windows V2 through graphical processing". In: *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*. 2018, pp. 263–268.

[94] Ba-Ngu Vo and Ba-Tuong Vo. "A multi-scan labeled random finite set model for multi-object state estimation". In: *IEEE Transactions on Signal Processing* 67.19 (2019), pp. 4948–4963.

[95] Weitian Wang et al. "Robot action planning by online optimization in human–robot collaborative tasks". In: *International Journal of Intelligent Robotics and Applications* 2.2 (2018), pp. 161–179.

[96] A Woch, R Plamondon, and C O'Reilly. "Kinematic characteristics of successful movement primitives in young and older subjects: a delta-lognormal comparison". In: *Hum. Mov. Sci* 30.1 (2011), pp. 1–17.

[97] Di Wu and Ling Shao. "Leveraging hierarchical parametric networks for skeletal joints based action segmentation and recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 724–731.

[98] Kimitoshi Yamazaki et al. "Home-assistant robot for an aging society". In: *Proceedings of the IEEE* 100.8 (2012), pp. 2429–2441.

[99] Chenyang Zhang and Yingli Tian. "RGB-D camera-based daily living activity recognition". In: *Journal of computer vision and image processing* 2.4 (2012), p. 12.