

**REAL-TIME ROBOT MOTION PLANNING ALGORITHMS AND
APPLICATIONS UNDER UNCERTAINTY**

Jae Sung Park

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2020

Approved by:

Dinesh Manocha

Marc Niethammer

Mohit Bansal

Ming C. Lin

Jia Pan

© 2020
Jae Sung Park
ALL RIGHTS RESERVED

ABSTRACT

Jae Sung Park: Real-time Robot Motion Planning Algorithms and
Applications Under Uncertainty
(Under the direction of Dinesh Manocha)

Robot motion planning is an important problem for real-world robot applications. Recently, the separation of workspaces between humans and robots has been gradually fading, and there is strong interest in developing solutions where collaborative robots (cobots) can interact or work safely with humans in a shared space or in close proximity. When working with humans in real-world environments, the robots need to plan safe motions under uncertainty stemming from many sources such as noise of visual sensors, ambiguity of verbal instruction, and variety of human motions.

In this thesis, we propose novel optimization-based and learning-based robot motion planning algorithms to deal with the uncertainties in real-world environments. To handle the input noise of visual cameras and the uncertainty of shape and pose estimation of surrounding objects, we present efficient probabilistic collision detection algorithms for Gaussian and non-Gaussian error distributions. By efficiently computing upper bounds of collision probability between an object and a robot, we present novel trajectory planning algorithms that guarantee that the collision probability at any trajectory point is less than a user-specified threshold. To enable human-robot interaction using natural language instructions, we present a mapping function from grounded linguistic semantics to the coefficients of the motion planning optimization problem. The mapping function considers task descriptions and motion-related constraints. For collaborative robots working with a human in close proximity, we present human intention and motion prediction algorithms for efficient task ordering and safe motion planning. The robot observes the human poses in real-time and predicts the future human motion based on the history of human poses. We also present an occlusion-aware robot motion planning algorithm that accounts for occlusion in the visual sensor data and uses learning-based techniques for trajectory planning. We highlight the benefits of our collision detection and robot motion planning algorithms with a 7-DOF Fetch robot arm in simulated and real-world environments.

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF TABLES	xiii
CHAPTER 1: INTRODUCTION	1
1.1 Components of Robot Motion Planning under Uncertainty	2
1.1.1 Environment Uncertainty and Collision Probability	3
1.1.2 Understanding Intention from Human Spoken Language	4
1.1.3 Understanding Intention from Human Motion	4
1.1.4 Uncertainty from Robot Occlusions	6
1.2 Thesis Statement	6
1.3 Main Results	8
1.3.1 Probabilistic Collision Detection	8
1.3.2 Motion Planning using NLP Instructions	9
1.3.3 Human Intention-aware Robot Motion Planner	11
1.3.4 Occlusion-aware Robot Motion Planner	13
CHAPTER 2: EFFICIENT PROBABILISTIC COLLISION DETECTION	16
2.1 Related Work	16
2.1.1 Probabilistic Collision Detection for Gaussian Errors	16
2.1.2 Probabilistic Collision Detection for Non-Gaussian Errors	17
2.1.3 Probabilistic Collision Detection: Applications	17
2.2 Overview	18
2.2.1 Probabilistic Collision Detection	18
2.2.2 Probabilistic Collision Detection for Gaussian Error	19
2.3 Truncated Gaussian Mixture Model Error Distribution	20

2.3.1	Truncated Gaussian Mixture Models	20
2.3.2	Collision Probability for Truncated Gaussian	21
2.3.3	Efficient Evaluation of the Integral	24
2.3.4	Error Distribution as Weighted Samples	25
2.3.5	Error Distribution as Truncated Gaussian Mixture Models	26
2.4	Performance and Analysis	26
2.4.1	Probabilistic Collision Detection: Performance	26
2.4.2	Sensor Noise Models for Static Obstacles	28
2.4.3	Robot Motion Planning	29
2.5	Conclusion and Limitations	32
2.5.1	Probabilistic Collision Detection	33
2.5.2	Motion Planning	35

CHAPTER 3: NATURAL LANGUAGE PROCESSING FOR SAFE HUMAN-ROBOT INTERACTION 38

3.1	Related Work	39
3.1.1	Natural Language Processing	39
3.1.2	Robot Motion Planning in Dynamic Environments	39
3.2	Dynamic Grounding Graphs	41
3.2.1	Latent Parameters	42
3.2.2	Probabilistic Model	43
3.2.3	Factor Graph using Conditional Random Fields	45
3.3	Dynamic Constraint Mapping With NLP Input	46
3.3.1	Robot Configurations and Motion Plans	47
3.3.2	Cost Functions	47
3.3.3	Parameterized Constraints	48
3.4	Implementation and Results	50
3.4.1	Training DGGs for Demonstrations	50
3.4.2	Simulations and Real Robot Demonstrations	51
3.4.3	Analysis	53
3.5	Benefits and Comparisons	55

3.6	Limitations, Conclusions, and Future Work	59
CHAPTER 4: HUMAN MOTION PREDICTION FOR SAFE ROBOT MOTION PLANNING		60
4.1	Related Work	61
4.1.1	Intention-aware Motion Planning and Prediction	61
4.1.2	Robot Task Planning for Human-Robot Collaboration	61
4.1.3	Motion Planning in Environments Shared with Humans	62
4.2	Notation and Assumptions	62
4.3	Human Action Prediction	66
4.3.1	Learning of Human Actions and Temporal Coherence	66
4.3.2	Runtime Human Intention and Motion Inference	68
4.3.3	Human Motion Prediction with Noisy Input	69
4.4	I-Planner: Intention-aware Motion Planning	71
4.4.1	Upper Bound of Collision Probability	74
4.4.2	Safe Trajectory Optimization	75
4.5	Implementation and Performance	76
4.5.1	Performance of Human Motion Prediction	76
4.5.2	Comparison with Prior Work	79
4.5.3	Robot Motion Planning with Human Motion Prediction	80
4.5.4	Robot Motion Responses to Human Motion Speed	82
4.5.5	Benefits of Our Prediction Algorithm	82
4.5.6	Evaluation Using 7-DOF Fetch Robot	84
4.6	Conclusions, Limitations, and Future Work	85
CHAPTER 5: OCCLUSION-AWARE ROBOT MOTION PLANNING		89
5.1	Related Work	91
5.1.1	Human Motion Prediction for Robotics	91
5.1.2	Human Motion Prediction from Images and Videos	92
5.1.3	Object Recognition under Occlusions in a Cluttered Environment	92
5.2	Overview	93

5.2.1	Problem Statement and Assumptions	93
5.3	Human Motion Prediction with Occluded Videos	95
5.3.1	Neural Network for Occluded Videos	95
5.3.2	Dataset Generation	97
5.4	Occlusion-Aware Motion Planning	100
5.4.1	Optimization-Based Planning of Robot Trajectories	100
5.4.2	Occlusion Sensitive Constraints	101
5.4.3	Real-time Collision Avoidance with Predicted Human Motions	101
5.5	Performance and Analysis	102
5.5.1	Human Action Recognition and Motion Prediction	102
5.5.2	Occlusion-aware Motion Planning	105
5.6	Conclusion and Limitations	106
CHAPTER 6: CONCLUSION AND FUTURE WORK		108
6.1	Limitations	109
6.2	Future Work	110
REFERENCES		111

LIST OF FIGURES

1.1	A summary diagram of our research. We have developed novel motion planning algorithms that can handle many sources of uncertainty. The center block in the diagram is our main motion planning module, which an optimization solver to find collision-free robot trajectories. The left blocks show novel algorithms that are being used by the motion planner. These include human motion prediction modules and probabilistic collision detection. The right block corresponds to applications that are used to demonstrate the benefits of our novel algorithms.	7
1.2	We highlight the benefits of our novel probabilistic collision detection with a Truncated Gaussian error distribution. Our formulation is used to accurately predict future human motions and is integrated with a motion planner for the 7-DOF Fetch robot arm. Compared to prior probabilistic collision detection algorithms based on Gaussian distribution [1], our new method improves the running time by 2.6x and improves the accuracy of collision detection by 9.7x.	8
1.3	The Fetch robot is moving a soda can on a table based on NLP instructions. Initially, the user gives the “pick and place” command. However, when the robot gets closer to the book, the person says “ <i>Don’t put it there</i> ” (i.e. negation) and the robot uses our dynamic constraint mapping functions and optimization-based planning to avoid the book. Our approach can generate appropriate motion plans for such attributes. . . .	10
1.4	Probabilistic collision checking with different confidence levels: A collision probability less than $(1 - \delta_{CD})$ implies a safe trajectory. The current pose (i.e. blue spheres) and the predicted future pose (i.e. red spheres) are shown. The robot’s trajectory avoids these collisions before the human performs an action. The higher the confidence level δ_{CD} , the farther the distance between the human arm and the robot trajectory. (a) $\delta_{CD} = 0.90$. (b) $\delta_{CD} = 0.95$. (c) $\delta_{CD} = 0.99$	11
1.5	A human and a robot are simultaneously operating in the same workspace. The robot arm occludes the camera view and many parts of the human obstacle are not captured by the camera. Three images at the top show the point clouds corresponding to the human in the UtKinect dataset [2] for different camera positions, with the occluded regions in red. The bottom right image highlights the safe motion trajectory between the initial position (blue) and the goal position (yellow). Our safe trajectory is shown in the bottom right as two red curves (with arrows). The occlusion-aware motion planner first moves the arm to reduce the occlusion and then moves it to the goal position.	13
2.1	(a) Contour plots of the bivariate TG distribution. (b) Contour plots of the bounded function \mathbf{F} for TG are not used in the calculation of collision probability and thereby reduce the running time of collision probability computation.	24
2.2	The upper bound of collision probability with uncertainty approximated as Gaussian, Gaussian Mixture, and Weight Samples. The X-axis is the true collision probability computed using Monte Carlo methods, and Y-axis is the computed probability using different methods. The computed upper bound for Gaussian Mixture and Weights Samples are closer to the ground truth/exact answer, than that for a single Gaussian approximation. The collision probability over-estimation with TGMM is reduced by 90%, compared to the one with Gaussian distribution.	24

2.3	(a) Speedup of weighted samples (expected) case compared to Monte Carlo (actual) with between 10 to 100 samples (X-axis). (b) Speedup of Truncated Gaussian case, compared to the running time of probabilistic collision detection with a Gaussian. X-axis is the untruncated volume of Gaussian, meaning 100% is the Gaussian and lower value indicates smaller bound. As the truncation boundary shrinks up to 50% of the volume of Gaussian, the algorithm with TG is 14x faster times than the algorithm Gaussian distribution.	27
2.4	(a) A captured RGBD image. The depth values of the table and the wood block have noises, even in adjacent frames. The TG noise of each point particle of the wood block contributes to the overall TGMM model. (b) A reconstructed 3D model of a wood block with TGMM, which bounded around the wooden block and more accuracy than Gaussian distribution, which has unbounded probability density function. (c) A reconstructed 3D robot environment with error distributions on the table and the wood blocks. The wood blocks placed in a zig-zag pattern result in 4 narrow passages for the robot.	28
2.5	We highlight the benefits of our novel probabilistic collision detection with a Truncated Gaussian error distribution. Our formulation is used to accurately predict the future human motion and integrated with a motion planner for the 7-DOF Fetch robot arm. As compared to prior probabilistic collision detection algorithms based on Gaussian distribution [1], our new method improves the running time by 2.6x and improves the accuracy of collision detection by 9.7x.	29
2.6	The robot trajectory with probabilistic collision detection through narrow passages. (a) The wood block obstacles are captured by RGBD sensors and its positional errors are modeled with Gaussian distributions. In this case, the planner is unable to compute a path in the narrow passage because of the conservative error bounds on the collision probability. (b) Robot motion planning with probabilistic collision detection under error distribution in the form of TGMM can generate a collision-free robot trajectory that passes through the narrow passage, due to tight bounds.	36
3.1	The overall pipeline of our approach highlighting the NLP parsing module and the motion planner. Above the dashed line (from left to right): Dynamic Grounding Graphs (DGG) with latent parameters that are used to parse and interpret the natural language commands, generation of optimization-based planning formulation with appropriate constraints and parameters using our mapping algorithm. We highlight the high-level interface below the dashed line. As the environment changes or new natural language instructions are given, our approach dynamically changes the specification of the constraints for the optimization-based motion planner and generates the new motion plans.	40
3.2	Factor graphs for different commands: In the environment in the right-hand column, there is a table with a thin rectangular object on it. A robot arm is moving a cup onto the table, but we want it to avoid moving over the book when given NLP instructions. (a) The command <i>"Put the cup on the table"</i> is given and the factor graph is constructed (left). Appropriate cost functions for the task are assigned to the motion planning algorithm (middle) and used to compute the robot motion (right). (b) As the robot gets close to the book, another command <i>"Don't put it there"</i> is given with a new factor graph and cost functions.	46
3.3	The simulated Fetch robot arm reaches towards one of the two red objects. (a) When a command <i>"pick up one of the red objects"</i> is issued, the robot moves to the red object on the right because of the DGG algorithm. (b) If the user doesn't want the robot to pick up the object on the right, he/she uses a command <i>"don't pick up that one."</i> Our DGG algorithm dynamically changes the cost function parameters. (c) The robot approaches the object on the right and stops.	52

3.4	In this simulated environment, the human instructs the robot to “ <i>put the cube on the table</i> ” (a). As it approaches the laptop (b), the human uses a negation NLP command “ <i>don’t put it there,</i> ” so the robot places it at a different location (c).	52
3.5	A 7-DOF Fetch robot is operating in a simulated environment avoiding an obstacle. (a) In a traditional optimization-based motion planner, the planner gets stuck at a local minimum. (b),(c) Using natural language commands as guidance, the user guides the robot out of the minimum and towards the goal position.	53
3.6	The Fetch robot is taking real-time commands from the human and moves the soda can on the table.	53
3.7	The Fetch robot is moving a soda can on a table based on NLP instructions. Initially, the user gives the “pick and place” command. However, when the robot gets closer to the book, the person says “ <i>don’t put it there</i> ” (i.e. negation) and the robot uses our dynamic constraint mapping functions and optimization-based planning to avoid the book. Our approach can generate appropriate motion plans for such attributes. . . .	54
4.1	Overview of our Intention-Aware Planner: Our approach consists of three main components: task planner, trajectory optimization, and intention and motion estimation.	63
4.2	Motion uncertainty and prediction: (a) A point cloud and the tracked human (blue spheres). The joint positions are used as feature vectors. (b) Prediction of next human action and future human motion, where 4 locations are colored according to their probability of next human action from white (0%) to black (100%). Prediction of future motion after 1 second (red spheres) from current motion (blue spheres) is shown as performing the action: <i>move right hand to the second position</i> which has the highest probability associated with it.	65
4.3	Human motion prediction results: The result of human motion prediction is represented by Gaussian distributions for each skeleton joint. The ellipsoid boundaries within which the integral of Gaussian distribution probability is 95% are drawn in red and the human skeleton is shown in blue. Bounding ellipsoids have transparency values that are proportional to the action classifier probability. (a) Undistinguished human action class: This occurs when the classifier fails to distinguish the human action class, generating nearly uniform probability distribution among the action classes. (b) Prediction results when untrained human motion is given: These cases result in larger boundary spheres around the human skeleton. This is because, in the Gaussian Process, the output has a uniform mean and a high variance when the input point is outside the range of the training input data.	77
4.4	Performance of human motion prediction: The precision/accuracy of classification and regression algorithms are shown in the graphs, with varying input noise. (a) Classification precision versus input noise: We only take the input points where the probability of an action classification is dominant, i.e. probability > 50%. (b) Regression precision versus input noise: Measured by the integral of distance between the predicted human motion and the ground-truth human motion trajectory. (c) Regression precision versus input noise: The integral of the volume of Gaussian distribution ellipsoids.	78

4.5	Responses to three different human arm movement speeds: While the robot arm moves from left to right, the human moves his arm to block the robot's trajectory at different speeds. (a) The human arm moves slowly. The robot has enough time to predict the human arm motion, generating the smoothest and the least jerky robot trajectory. (b) As the human moves at a medium speed, the robot predicts the human's future motion, recognizes that it will block the robot's path, and therefore changes the trajectory upwards (at $t = 0.8s$) to avoid the obstacle and generate a smooth trajectory. (c) When the human arm moves faster, the robot trajectory abruptly changes to move upwards (at $t = 0.8s$), generating a less smooth trajectory while still avoiding the human.	86
4.6	Different block arrangements: Different arrangements in terms of the positions of the blocks, results in different human motions and actions. Our planner computes their intent for safe trajectory planning. The different arrangements are: (a) 1×4 . (b) 2×2 . (c) 2×4	87
4.7	Probabilistic collision checking with different confidence levels: A collision probability less $(1 - \delta_{CD})$ implies a safe trajectory. The current pose (i.e., blue spheres) and the predicted future pose (i.e. red spheres) are shown. The robot's trajectory avoids these collisions before the human performs its action. The higher the confidence level is, the longer the distance between the human arm and the robot trajectory. (a) $\delta_{CD} = 0.90$. (b) $\delta_{CD} = 0.95$. (c) $\delta_{CD} = 0.99$	87
4.8	A 7-DOF Fetch robot is moving its arm near a human, avoiding collisions. (a) While the robot is moving, the human tries to move his arm to block the robot's path. The robot arm trajectory is planned without human motion prediction, which may result in collisions and a jerky trajectory, as shown with the red circle. This is because the robot cannot respond to the human motion to avoid collisions. (b) The trajectory is computed using our human motion prediction algorithm; it avoids collisions and results in smoother trajectories. The robot trajectory computation uses collision probabilities to anticipate the motion and compute safe trajectories.	88
5.1	A human and a robot are simultaneously operating in the same workspace. The robot arm occludes the camera view and many parts of the human obstacle are not captured by the camera. Three images at the top show the point clouds corresponding to the human in the UtKinect dataset [2] for different camera positions with the occluded regions in red. The bottom right image highlights the safe motion trajectory between the initial position (blue) and the goal position (yellow). Our safe trajectory is shown in the bottom right as two red curves (with arrows). HMPO first moves the arm to reduce the occlusion and then moves it to the goal position.	90
5.2	HMPO: Overall pipeline of our human motion prediction and robot motion planning. We present a new deep learning technique for human motion prediction in occluded scenarios and an optimization-based planning algorithm that accounts for occlusion.	93
5.3	Sample images of original datasets and modifications with occlusion information. (a) UTKinect dataset [2]. (b) Watch-n-patch dataset [3]. (c) Occlusion MoCap dataset [4]. We present 3 image pairs for each dataset in each column. The top image in each pair is the original image from the dataset, and the bottom images are generated by augmenting the original images with robot arm occlusions at the bottom. These augmented images are used for training and cross-validation.	95

5.4	Benefits of Occlusion-Aware Planning: The top row highlights the point cloud with the dynamic human obstacle, and the regions occluded by robot arms (in red). The bottom row highlights the trajectories computed by different planners when as the robot arm needs to move from right to left: (a) The trajectory is generated by the baseline planner, which does not account for occlusion. When the robot occludes the human, the motion prediction error is high and results in collisions. (b) The robot arm motion is generated by our occlusion-sensitive planner. The arm first moves to reduce the level of occlusion (i.e. a detour) and then reaches the goal to compute a safe trajectory.	103
5.5	Average error distance over time for up to 3 seconds between ground truth joint positions and the predicted joint positions. The error distances for the skeleton tracking [4] and Extended Kalman Filter (EKF) are shown with a dashed line, and the error distances for our models are shown with solid lines. The baseline model without occlusion input images has higher error distances. However, the HMPO model has better prediction results with lower error distance values than EKF when the future prediction time is 1.5 seconds or higher.	104

LIST OF TABLES

2.1	Performance of probabilistic collision detection algorithms: Evaluated as part of a motion planner with sensor data. The collision probability over-estimation is shown as percent point (%p) with the minimum and maximum over-estimation. The values corresponds to the average over the time of the robot trajectory with standard deviation in parenthesis. The best performance is obtained PCD-TGMM algorithm in terms of collision probability estimation (i.e. tight bounds), successful handling of narrow passages, computing collision-free trajectories, among different algorithms. . .	30
3.1	Planning performances with varying sizes of training data for the scenario in Fig. 3.3 with 21 different NLP instructions.	54
3.2	Running time (ms) of our DGG and motion planning modules for each scenario. . .	55
3.3	Examples of DGGs with different configurations of correspondence variables.	56
3.4	Examples of DGGs and the latent variables.	57
3.5	Examples of DGGs and the latent variables.	58
4.1	Performances of action classification and motion regression of ATCRF [5] and our human motion prediction algorithm. As ATCRF algorithm does not predict the variance of motion around the trajectory, the motion regression accuracy cannot be measured on the algorithm. ATCRF uses object affordance annotations as additional information in the learning phase, and we got higher precision and recall with ATCRF than with our algorithm. In terms of motion regression, our algorithm outperformed ATCRF in regression precision.	80
4.2	Three different simulation scenarios: <i>Different Arrangements</i> , <i>Temporal Coherence</i> , and <i>Confidence Level</i> . We consider different arrangements of blocks as well as the confidence levels used for probabilistic collision checking. These confidence levels are used for motion prediction.	82
4.3	Performance of our planning algorithm in terms of robot motion simulation. The use of motion prediction results in a smoother trajectory and we observe up to 4X improvement in our smoothness metric, as defined in Equation (4.12). Our resulting planning algorithm (I-Planner) runs in real-time.	83
4.4	Performance of our planning algorithm on a real robot running on a 7-DOF Fetch robot next to dynamic human obstacles. Our online motion planner computes safe trajectories for challenging benchmarks like “moving cans.” We observe almost 5X improvement in the smoothness of the trajectory due to our prediction algorithm. . .	84
5.1	Accuracy Comparison of Prediction Algorithms on Different Datasets: Average error distance (lower is better) between ground truth joint positions and the predicted joint positions after 3 second for different datasets and algorithms. The numbers in parentheses are standard deviations. The baseline is based on tracking methods [4] along with extended Kalman filters on the skeleton-based human motion model. Our approach, HMPO (31.8 cm), reduces the error distance dataset by 38% from the particle filter-based tracking [4] plus Extended Kalman Filter (51.6 cm) and 50% from the baseline (64.0 cm). This demonstrates the accuracy benefits of our occlusion-aware planner.	105

5.2 Accuracy of action classification and human motion prediction algorithms for the Watch-n-Patch dataset (higher is better). The numbers in parentheses are standard deviations. HMPO (36.6%) improves the action classification accuracy in the *Occlusion* dataset by 63% from Wu et al. [3] (22.5%) and 86% from the baseline (19.7%). 106

CHAPTER 1

Introduction

Robot motion planning has been extensively studied for many decades [6]. The goal of robot motion planning is to find a trajectory connecting two robot configurations that does not result in collisions with any obstacles. A robot configuration corresponds to a high-dimensional vector describing the robot's pose. For robot arms, a configuration consists of rotational joint angles. These configurations are used to formulate the configuration space of a robot. The motion planning problem reduces to finding a collision-free one-dimensional path in this high-dimensional space. Robot motion planning has been extensively studied in the field and different solutions based on analytic techniques, sampling-based approaches and optimization-based methods have been proposed [7].

Human-robot interaction (HRI) is an important research area in robotics and has many applications in real-world settings [8]. Traditionally, many industrial robots have been placed away from humans for safety, and this limits their applications. Generally, humans can handle jobs that require better dexterity skills than robots [9, 10]. For some applications, however, it is more efficient for humans and robots to work together while sharing the same workspace. For example, in an assembly job where there are parts to be assembled on a table, a human can assemble parts while a robot arm delivers the parts required for next step of the assembly. If the robot uses a natural language processing module to understand spoken instructions from a human, the person can request that the robot brings the required parts. In such cooperative jobs, humans and robots share the same workspace. In the assembly application, because the human and the robot work on the same table, there is a chance that the robot may collide with the human arms or the human body. Also, the robot should be able to understand the human instructions in terms of natural language processing. If the robot misinterprets the spoken instructions, the robot arm may fail to deliver the correct items, leading to a delay in task completion time.

Robot motion planning plays a crucial role in terms of performing safe and efficient human-robot interactions. In human-robot collaborative tasks, the robot should not only complete its task but

also understand and predict the human’s intention and avoid the human as a dynamic obstacle. In real-world settings, a major challenge of robot motion planning for cooperation between humans and robots is to deal with uncertainties from many sources. Noisy vision sensors are a source of uncertainty. Depth cameras are widely used for collision avoidance in robot motion planning but result in very noisy inputs. Color cameras have less noise [11], but the noisy input still may cause trouble in terms of vision-related robot motion planning tasks. Human intention and human action prediction are another source of uncertainty. Because of the computation time of robot motion plans and the delay between trajectory planning and trajectory execution, the computed trajectories may collide with a human after the person changes his/her pose from the initial pose that was used by the robot motion planning module. Natural language instructions are yet another source of uncertainty. Spoken commands for robots can be implicit and the robot must sometimes guess the implicit meaning from the context. To overcome these problems related to uncertainty, we need to develop new motion planning algorithms that can handle such uncertainties and also provide realtime performance for human-robot interaction.

1.1 Components of Robot Motion Planning under Uncertainty

Practical motion planning methods are based on sampling-based methods or optimization-based methods. Rapidly-exploring Random Tree (RRT) [7] is a widely used sampling-based robot motion planning algorithm with fast computation speed. RRT generates random samples of robot poses around the initial pose as tree nodes, connects the tree nodes if the continuous motion between the initial poses is feasible without collisions, and expands until the connectivity between the initial and the goal poses is computed, resulting in a collision-free robot motion trajectory. However, RRT is not adequate for dynamic environments with uncertainties for the following reasons. The existing edges in the tree may become invalid due to the changing environment with uncertainties. Also, other edges could have been established in the changing environment to generate a better robot trajectory. Optimization-based robot motion planning is a well-known technique for robots to find collision-free trajectories among dynamically moving obstacles [12]. Given the robot’s start pose and the end pose, these algorithms try to find a robot trajectory that connects the two poses while minimizing the objective function within a constrained high-dimensional space. Usually, the objective function includes a smoothness component, and the constraints include a collision-free constraint. Strengths of optimization-based motion planning include its versatility in terms of the optimization formulation.

Specifically, users can add cost functions or constraints in addition to the smoothness cost and the collision-free constraint to meet specific goals in various real-world settings. For example, when a robot moves a cup filled with water, a user can simply add a constraint that the up-vector of the cup should point in the upward direction. Also, because the robot should not spill the water by moving quickly, a cost function that penalizes high speed or the cup can be added in the objective function.

By taking advantage of the strengths of optimization-based robot motion planning, the basic optimization-based motion planner can be extended in many ways and operated with other techniques, such as predicting future human motions, adding various cost functions for safety and efficiency, and adding constraints related to human motions near robots and natural language commands spoken by humans. A robot motion planner for safe human-robot interaction should consider uncertainties coming from real-world environments. In this section, we discuss some of the components of robot motion planning under uncertainty.

1.1.1 Environment Uncertainty and Collision Probability

Efficient collision detection is important for safe robot motion planning under environmental uncertainties. Previous work in exact collision detection mainly dealt with problems with different variants on input shapes, such as collision detection between rigid convex shapes or collision detection between non-rigid dynamic models [13, 14]. The outputs of these methods are binary values, indicating whether two input shapes are in collision.

When navigating and interacting with real-world objects, robots should gather information about the surroundings and handle environmental uncertainties. Unfortunately, robot cameras or cameras installed around the robot have sensory error, making it difficult to obtain exact shapes and poses of objects in the environment. For example, depth representations captured by depth cameras have errors that correspond to lighting, calibration, or object surfaces [11]. This gives rise to probabilistic collision detection, where the goal is to compute the probability of an in-collision state by modeling the uncertainty using some probabilistic distribution.

Many collision detection algorithms have been proposed to account for such uncertainties [15, 16, 17]. In practice, it is difficult to analytically compute the collision probability for all probabilistic representations of uncertainties. Most prior work on probabilistic collision detection is limited to Gaussian distributions [18, 1]. However, these formulations may not work well when objects are captured by a depth sensor. Even the process of capturing static objects can result in depth images,

where the depth values can vary between consecutive frames due to different noise sources. The dynamic objects in the scene can pose additional problems due to sensor noise and the uncertainties introduced by the objects’ motions. Moreover, Gaussian process dynamical models used to represent human motion [19] have an inherited uncertainty in the Gaussian variances because the central motion is represented using Gaussian means.

Collision probability between robots and obstacles can be used in optimization-based robot motion planner. Collision probability can be used for formulating a hard constraint. For example, a robot motion trajectory should have collision probability between the robot and uncertain surroundings lower than a threshold value (e.g., 0.05).

1.1.2 Understanding Intention from Human Spoken Language

Natural language has been used as an interface to communicate a human’s intent to a robot [20, 21, 22, 23] in the field of human-robot interaction. A key challenge in understanding natural language commands is the uncertainty in natural languages which are typically ambiguous and vague [24]. When the robot misinterprets the natural language commands, it fails to satisfy the human intention.

There are many challenges in terms of understanding natural languages for robots. One challenge is that the natural language instructions should be accurately interpreted and represented with grounded linguistic semantics at the motion level, especially considering the surrounding environment and the context. For example, if a human says “Move that to the left” or “Do not move like this,” the robot should learn the correct interpretation of the objectives and constraints, including spatial and motion-based adjectives, adverbs, and negation. Another challenge is that the motion planner should generate appropriate trajectories based on these complex natural language instructions, achieving the intended goal specified by the natural language instructions. To overcome these challenges, the robot motion planner should appropriately set up the motion planning optimization problem based on different motion constraints (e.g., orientation, velocity, smoothness, and avoidance) and compute smooth and collision-free robot trajectories.

1.1.3 Understanding Intention from Human Motion

Human motion prediction is an important part of human-robot interaction in environments where robots work in close proximity to humans. For safe and reliable robot motion planning in real-world settings with humans, it is essential that robot motion planning module should be able to handle uncertainties with respect to future human motion. While humans are moving, it is

important for the robots to predict human actions and motions from visual sensor data and to compute collision-free trajectories. Also, human motion prediction makes task completion efficient by predicting the human motion and planning tasks accordingly, when the robots and the humans are sharing the same workspace and arranging the subtasks. For example, in an assembly job where there are parts to be assembled on a table, a human can assemble parts while a robot arm delivers other required parts for the next step. In such cooperative jobs, humans and robots share the same workspace. In the assembly application, because the human and the robot work on the same table, there is a chance that the robot may collide with the human arms or body.

By predicting human motions and intentions, we can achieve improved safety and efficiency in human-robot interaction, computing a safe, collision-free path for the robot to reach its goal configuration and planning subtasks in a way that the entire task is completed as quickly as possible. The robot should not only complete its task but also predict the human’s motion or trajectory to avoid the human as a dynamic obstacle. There is considerable work on human motion prediction and safe trajectory computation. By predicting future human motion trajectory in advance, the robot motion planner can generate a safe motion trajectory for a few seconds in the future by avoiding the predicted human poses. It can also generate an efficient motion trajectory for moving near the human and doing collaborative tasks. Some recent methods to predict human motions from images or videos are based on Convolutional Neural Networks (CNNs) [25, 26, 27, 28] or Recurrent Neural Networks (RNNs) [29, 30].

To compute reliable motion trajectories in such shared environments, it is important to gather the states of the humans and predict their motions. There is considerable work on online tracking and prediction of human motion in computer vision and related areas [31]. However, the current state-of-the-art in gathering or tracking the motion data results in many challenges. First of all, there are errors in the data due to the sensors (e.g., point cloud sensors) or poor sampling [32]. Secondly, human motion can be sudden or abrupt and this can result in various uncertainties in terms of accurate representation of the environment. One way to overcome some of these problems is to use predictive or estimation techniques for human motion or actions, such as using filters like Kalman filters or particle filters [33]. Most of these prediction algorithms use a motion model that can predict future motion based on the prior positions of human body parts or joints, and corrects the error between the estimates and actual measurements. In practice, these approaches only work

well when there is sufficient information about prior motion that can be accurately modeled by the underlying motion model. In some scenarios, it is possible to infer high-level human intent using additional information, and thereby perform a better prediction of future human motion. These techniques are used to predict the pedestrian trajectories based on environmental information in 2D domains.

1.1.4 Uncertainty from Robot Occlusions

Robots gather information about the surrounding environment from visual sensors or depth sensors. A head-mounted RGBD camera is a typical device for collaborative robots to observe the workspace. When robots are working with humans sharing the same workspace, the moving parts of the robot may occlude the views of these sensors while robots perform actions with their hands or arms. The input color and depth images may not be able to capture information about many parts of the scenes, including the current position of the human working close to the robot [34, 35, 36]. Such occlusion by parts of a robot can prevent accurate tracking and prediction of the human motion and thereby make it difficult to perform safe and collision-free motion planning. When the robot arm occludes the input images, either the robot should determine whether the human motion can be predicted with high certainty or the robot arm should move so that it does not occlude the field of view of the camera.

To overcome the uncertainty caused by robot occlusions, the robot motion planner should predict what is occurring behind the robot occlusion. Fortunately, robot occlusions in the input color and depth images are known from forward kinematics. From the history of an input image sequence, the robot should predict the unobservable or partially observable humans behind the occlusions or remove the occlusion so that the human can be clearly seen.

1.2 Thesis Statement

Our thesis statement is as follows:

Optimization-based methods and learning algorithms can be combined for safe and efficient motion planning for human-robot interaction and handling uncertainties.

Our research focuses on developing algorithms for optimization-based motion planning in the safe human-robot interaction problem under uncertainties. Figure 1.1 shows a high-level overview of our research. The following list describes specific algorithms for developing robot motion planning under uncertainty.

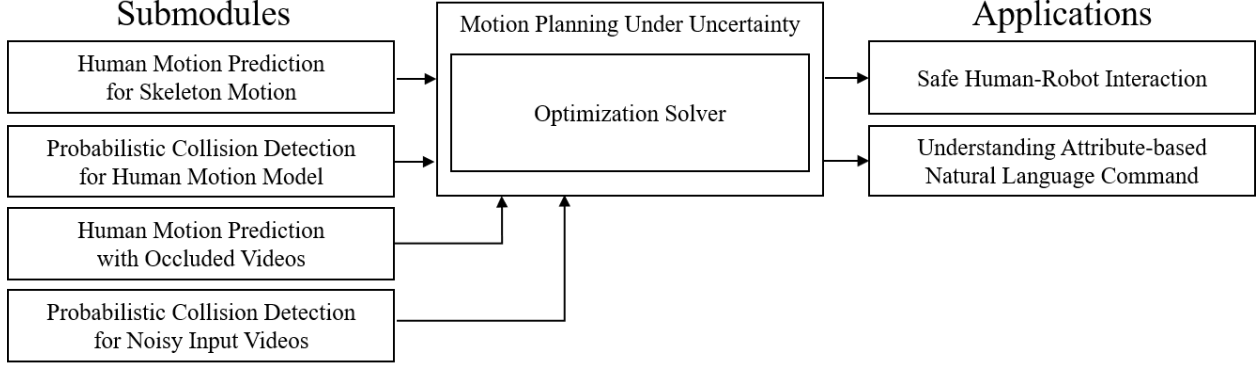


Figure 1.1: A summary diagram of our research. We have developed novel motion planning algorithms that can handle many sources of uncertainty. The center block in the diagram is our main motion planning module, which an optimization solver to find collision-free robot trajectories. The left blocks show novel algorithms that are being used by the motion planner. These include human motion prediction modules and probabilistic collision detection. The right block corresponds to applications that are used to demonstrate the benefits of our novel algorithms.

- Probabilistic collision detection for objects with pose uncertainties:** We present probabilistic collision detection algorithms for human motion models and noisy input data. Computing the probability of collision between a robot trajectory and possible future human movements is important because the predicted human motion may be different from the real human motion. Also, because of noises in the depth sensors, the reconstructed environment surrounding the robot is not precise. Probabilistic collision detection deals with computing collision probability when uncertainties of static obstacles captured by robot cameras and predicted dynamic human obstacles are represented by probability distributions. We develop an efficient algorithm when the probability distribution representing uncertainty is given as Gaussian distribution. We also develop efficient algorithms for probabilistic collision detection algorithms for non-Gaussian distributions, where we use more complex statistical models to represent uncertainties.
- Natural Language Processing (NLP) for optimization-based robot motion planning:** We present a novel approach that combines these algorithms and performs efficient motion planning under uncertainty for many real-world applications. Safe human-robot interaction can be demonstrated with human motion prediction models and probabilistic collision detection algorithms. We show how attribute-based natural language commands are transformed into appropriate constraints for optimization-based motion planning.

- **Human motion and intention prediction for safe robot motion planning:** We present robot motion planning algorithms that can deal with many types of uncertainties. They may come from noisy inputs, unknown human intentions or motions, or spoken natural language instructions. To effectively handle many kinds of uncertainty, we choose optimization-based motion planning algorithms, taking advantage of their versatility.
- **Occlusion-aware robot motion planner for partially-visible scenes:** We present a prediction model for human motion and intention. As the first step, we develop a human motion prediction algorithm for a human skeletal motion model. This algorithm classifies the type of human action and estimates potential future movements in a short time window; the robot motion planning algorithm uses this information to avoid potential collisions in the future. We also develop a new algorithm based on video input in which the scene may be partially occluded by a robot arm.

1.3 Main Results

In this section, we describe the algorithms to overcome the uncertainty problems in more detail.

1.3.1 Probabilistic Collision Detection

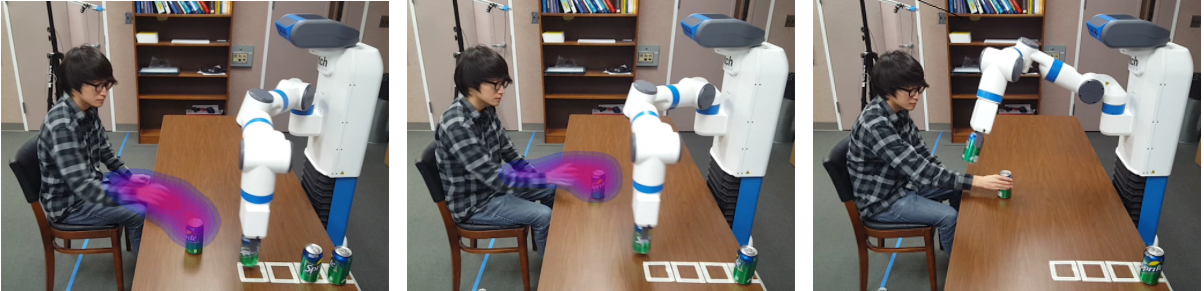


Figure 1.2: We highlight the benefits of our novel probabilistic collision detection with a Truncated Gaussian error distribution. Our formulation is used to accurately predict future human motions and is integrated with a motion planner for the 7-DOF Fetch robot arm. Compared to prior probabilistic collision detection algorithms based on Gaussian distribution [1], our new method improves the running time by 2.6x and improves the accuracy of collision detection by 9.7x.

We present efficient algorithms to compute the collision probability for error distributions corresponding to a variety of non-Gaussian models, including weighted samples and Truncated Gaussian (TG). Our approach is based on modeling the TG error distribution and represents the collision probability using a volume integral. We present efficient techniques to evaluate the integral

and highlight the benefits over prior methods for probabilistic collision detection. We evaluate the performance of these methods on synthetic and real-world datasets captured using depth cameras. Furthermore, we show that our efficient probabilistic collision detection algorithm can be used for real-time robot motion planning of a 7-DOF manipulator in tight scenarios with depth sensors. Some novel components of our work include:

- A novel method to perform probabilistic collision detection for TG Mixture Models based on appropriately formulating the vector field and computing an upper bound using a divergence theorem on the resulting integral. Moreover, we present an efficient method to evaluate this bound for convex and non-convex shapes.
- We show that TG outperforms normal Gaussian, and Truncated Gaussian Mixture Model (TGMM) outperforms Gaussian Mixture Model (GMM). In practice, probabilistic formulation is less conservative than prior methods and results in $5 - 9\times$ better accuracy in terms of collision probability computation (Table 1).
- We have combined our probabilistic collision formulation with an optimization-based realtime robot motion planner that accounts for positional uncertainty from depth sensors. Our modified planner is less conservative in terms of computing paths in tight scenarios.

In Chapter 2, we present an efficient algorithm to compute tight upper bounds of collision probability between two objects with positional uncertainties with error distributions represented with non-Gaussian forms. Our approach can handle noisy datasets from depth sensors with distributions that may correspond to Truncated Gaussian, Weighted Samples, or Truncated Gaussian Mixture Model. We derive tight probability bounds for convex shapes and extend them to non-convex shapes using hierarchical representations. We highlight the benefits of our approach over prior probabilistic collision detection algorithms in terms of tighter bounds (10x) and improved running time (3x). Moreover, we use our tight bounds to design an efficient and accurate motion planning algorithm for a 7-DOF robot arm operating in tight scenarios with sensor and motion uncertainties.

1.3.2 Motion Planning using NLP Instructions

We present an algorithm for generating parameterized constraints for optimization-based motion planning from complex, attribute-based natural language instructions. We use *Dynamic Grounding*

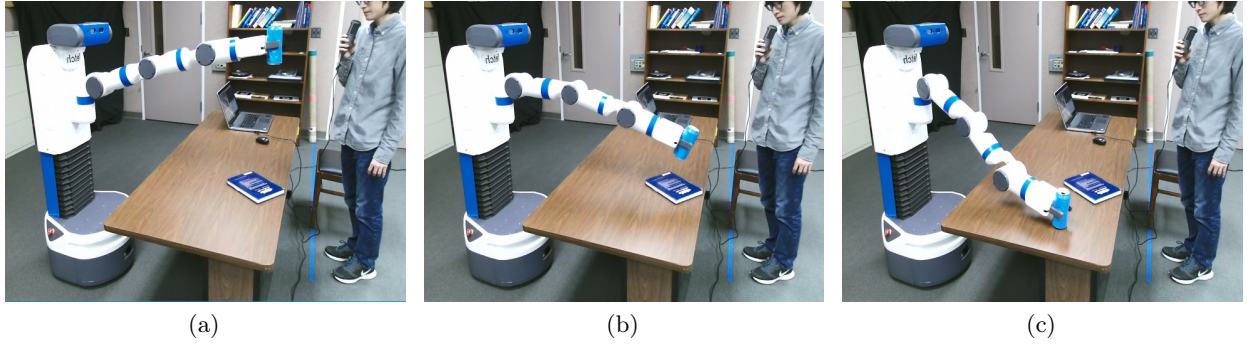


Figure 1.3: The Fetch robot is moving a soda can on a table based on NLP instructions. Initially, the user gives the “pick and place” command. However, when the robot gets closer to the book, the person says “*Don’t put it there*” (i.e. negation) and the robot uses our dynamic constraint mapping functions and optimization-based planning to avoid the book. Our approach can generate appropriate motion plans for such attributes.

Graphs (DGG) to parse and interpret the commands and to generate the constraints. Our formulation includes the latent parameters in the grounding process, allowing us to model many continuous variables in our grounding graph. Furthermore, we present a new dynamic constraint mapping that takes DGG as the input and computes different constraints and parameters for the motion planner. The appropriate motion parameters are speed, orientation, position, smoothness, repulsion, and avoidance. The final trajectory of the robot is computed using a constraint optimization solver. Overall, our approach can automatically handle complex natural language instructions corresponding to spatial and temporal adjectives, adverbs, superlative and comparative degrees, negations, etc. Compared to prior techniques, our overall approach offers the following benefits:

- The inclusion of latent parameters in the grounding graph allows us to model continuous variables that are used by our mapping algorithm. Our formulation computes the dynamic grounding graph based on conditional random fields.
- We present a novel dynamic constraint mapping used to compute different parametric constraints for optimization-based motion planning.
- Our grounding graphs can handle more complex, attribute-based natural language instructions, and our mapping algorithm uses appropriate cost functions as parameters over the continuous space. Compared to prior methods, our approach is much faster and better able to handle more complex and attribute-based natural language instructions.

We highlight the performance of our algorithms in a simulated environment and on a 7-DOF Fetch robot operating next to a human. Our approach can handle a rich set of natural language commands and can generate appropriate paths. These include complex commands such as picking (e.g., *"Pick up a red object near you"*), correcting the motion (e.g., *"Don't pick up that one"*), and negation (e.g., *"Don't put it on the book"*).

In Chapter 3, we present an algorithm for combining natural language processing (NLP) and fast robot motion planning to automatically generate robot movements. Our formulation uses a novel concept called Dynamic Constraint Mapping to transform complex, attribute-based natural language instructions into appropriate cost functions and parametric constraints for optimization-based motion planning. We generate a factor graph from natural language instructions called the Dynamic Grounding Graph (DGG), which takes latent parameters into account. The coefficients of this factor graph are learned based on conditional random fields (CRFs) and are used to dynamically generate the constraints for motion planning. We map the cost function directly to the motion parameters of the planner and compute smooth trajectories in dynamic scenes. We highlight the performance of our approach in a simulated environment and via a human interacting with a 7-DOF Fetch robot using intricate language commands including negation, orientation specification, and distance constraints.

1.3.3 Human Intention-aware Robot Motion Planner

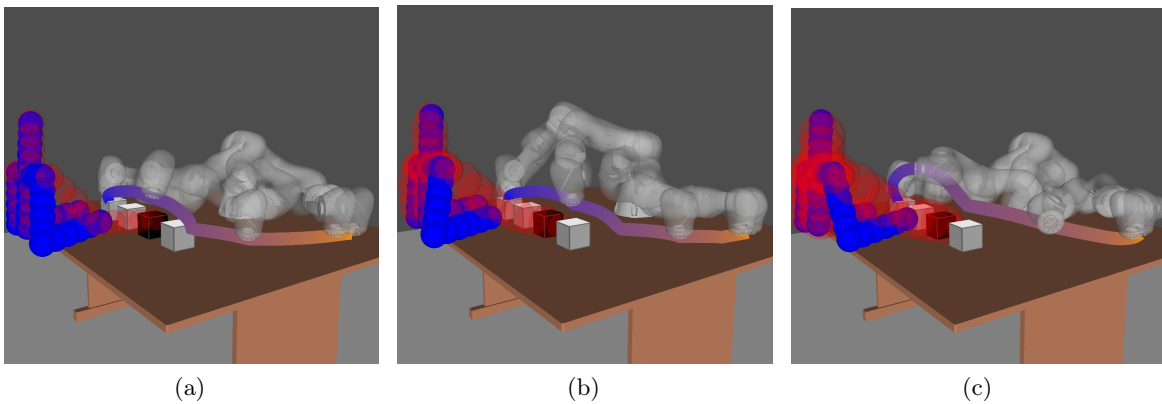


Figure 1.4: **Probabilistic collision checking with different confidence levels:** A collision probability less than $(1 - \delta_{CD})$ implies a safe trajectory. The current pose (i.e. blue spheres) and the predicted future pose (i.e. red spheres) are shown. The robot's trajectory avoids these collisions before the human performs an action. The higher the confidence level δ_{CD} , the farther the distance between the human arm and the robot trajectory. (a) $\delta_{CD} = 0.90$. (b) $\delta_{CD} = 0.95$. (c) $\delta_{CD} = 0.99$.

We present a novel high-DOF motion planning approach to compute collision-free trajectories for robots operating in a workspace with human obstacles or human-robot cooperation scenarios (I-Planner). Our approach is general and doesn’t make assumptions about the environment or the human actions. We track the positions of the human using depth cameras and present a new method for human action prediction using a combination of classification (to predict the type of human motion) and regression (to predict the actual future human motion) methods. Given the sensor noises and prediction errors, our online motion planner uses probabilistic collision checking to compute a high-dimensional robot trajectory that tends to compute safe motions in the presence of uncertain human motion. In contrast to prior methods, the main benefits of our approach include:

1. A novel data-driven algorithm for intention and motion prediction, given noisy point cloud data. Compared to prior methods, our formulation can account for a lot of noise in skeleton tracking in terms of human motion prediction.
2. An online high-DOF robot motion planner for efficient completion of collaborative human-robot tasks that uses upper bounds on collision probabilities to compute safe trajectories in challenging 3D workspaces. Furthermore, our trajectory optimization based on probabilistic collision checking results in smoother paths.

We highlight the performance of our algorithms in a simulator with a 7-DOF KUKA arm and in a real-world setting with a 7-DOF Fetch robot arm in a workspace with a moving human performing cooperative tasks. We have evaluated its performance in some challenging or cluttered 3D environments where the human is close to the robot and moving at varying speeds. We demonstrate the benefits of our intention-aware planner in terms of computing safe trajectories in these scenarios. A preliminary version of this paper was published [36]. Compared to [36], we improve the human motion prediction algorithm using depth sensor data. We present a mathematical analysis of the robustness of our prediction algorithm and highlight its benefits and improved accuracy for challenging scenarios. We also analyze the performance of our algorithm with varying human motion speeds.

In Chapter 4, we present a motion planning algorithm to compute collision-free and smooth trajectories for high-DOF robots interacting with humans in a shared workspace. Our approach uses offline learning of human actions along with temporal coherence to predict the human actions.

Our intention-aware online planning algorithm uses the learned database to compute a reliable trajectory based on the predicted actions. We represent the predicted human motion using a Gaussian distribution and compute tight upper bounds on collision probabilities for safe motion planning. We also describe novel techniques to account for noise in human motion prediction. We highlight the performance of our planning algorithm in complex simulated scenarios and real-world benchmarks with 7-DOF robot arms operating in a workspace with a human performing complex tasks. We demonstrate the benefits of our intention-aware planner in terms of computing safe trajectories in such uncertain environments.

1.3.4 Occlusion-aware Robot Motion Planner

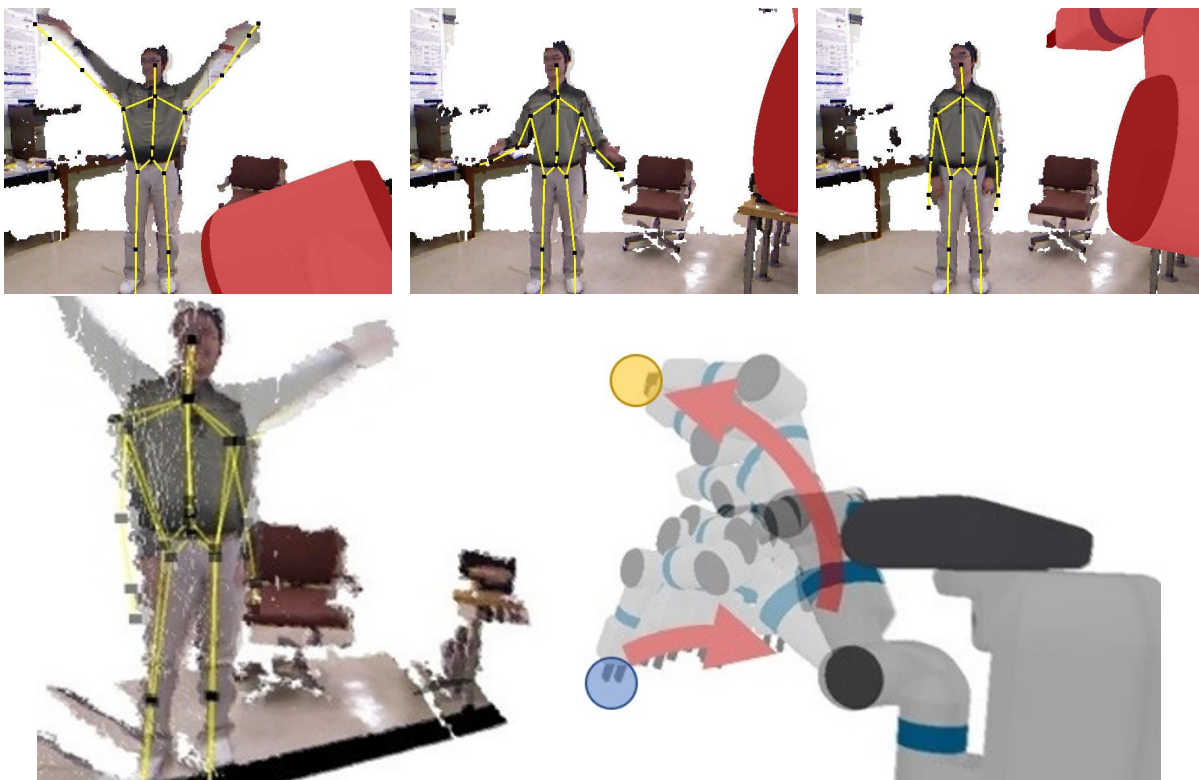


Figure 1.5: A human and a robot are simultaneously operating in the same workspace. The robot arm occludes the camera view and many parts of the human obstacle are not captured by the camera. Three images at the top show the point clouds corresponding to the human in the UtKinect dataset [2] for different camera positions, with the occluded regions in red. The bottom right image highlights the safe motion trajectory between the initial position (blue) and the goal position (yellow). Our safe trajectory is shown in the bottom right as two red curves (with arrows). The occlusion-aware motion planner first moves the arm to reduce the occlusion and then moves it to the goal position.

We address the challenges highlighted above by presenting two novel algorithms: (1) predict

human motion in the presence of obstacles and occlusions; (2) plan a robot’s motion while taking into account the occlusion and the uncertainty in the motion prediction.

- **Human Motion Prediction in Occluded Scenarios:** We present a neural network that uses not only the features from RGBD images, but also features related to occlusion. Our deep learning-based approach predicts the human motion in such occluded scenarios. We use CNNs for feature extraction from RGBD images and feature extraction for robot occlusion. Moreover, we use ResNet-18 [37] to extract visual features from color images with occluded regions. Our learning algorithm classifies the human action and generates the predicted human motion using a skeleton-based human model. We add occluded images of robot scenes to existing RGBD human action prediction datasets [2, 3, 4]. We use these augmented datasets to train and evaluate the performance of our human motion prediction algorithm in the presence of occlusion. In practice, our action classification algorithm improves the prediction accuracy by 63% over prior classification algorithms [3].
- **Occlusion-Aware Motion Planning:** We present a realtime planning algorithm to compute a safe trajectory for a robot in occluded scenes with human obstacles. We use an optimization-based planning framework and add the occlusion constraints in the objective function. Our planner tends to compute collision-free paths and ensures that the human region in the camera image is not occluded by the robot. We have evaluated our planner in complex environments with robots operating close to the human. In practice, our algorithm improves the overall accuracy, measured using error distance between the ground-truth and the predicted human joint positions, by 38%.

We use three human action RGB-D datasets and augment them with occlusion characteristics for training and validation. We highlight the performance of the overall approach (HMPO) in complex environments. We released our source code for augmenting the datasets at <https://github.com/jsonpark/occlusion>

In Chapter 5, we present a novel approach to generate collision-free trajectories for a robot operating in close proximity with a human obstacle in an occluded environment. The self-occlusions of the robot can significantly reduce the accuracy of human motion prediction, and we present a novel deep learning-based prediction algorithm. Our formulation uses CNNs and LSTMs and we

augment human-action datasets with synthetically generated occlusion information for training. We also present an occlusion-aware planner that uses our motion prediction algorithm to compute collision-free trajectories. We highlight the performance of the overall approach (HMPO) in complex scenarios and observe up to 68% performance improvement in motion prediction accuracy, and 38% improvement in terms of error distance between the ground-truth and the predicted human joint positions.

CHAPTER 2

Efficient Probabilistic Collision Detection

Efficient collision detection is an important problem in robot motion planning, physics-based simulation, and geometric applications. Earlier work in collision detection focused on fast algorithms for rigid convex polytopes and non-convex shapes and later extended to non-rigid models [13, 14]. Most of these methods assume that an exact geometric representation of the objects is known in terms of triangles or continuous surfaces [38] and the output of collision query is a simple binary outcome.

As robots navigate and interact with real-world objects, we need algorithms for motion planning and collision detection that can handle environmental uncertainty. In particular, robots operate with sensor data, and it is hard to obtain an exact shape or pose of an object. For example, depth cameras are widely used in robotics applications and the captured representations may have errors that correspond to lighting, calibration, or object surfaces [11]. This gives rise to probabilistic collision detection, where the goal is to compute the probability of in-collision state by modeling the uncertainty using some probabilistic distribution.

In many applications, it is necessary to use non-Gaussian models for uncertainties [39]. These include Truncated Gaussian with bounded domains for sensory noises [40, 41] to represent the position uncertainties for a point robot position [42]. Other techniques model the uncertainty as a Partially Observable Markov Decision Process (POMDP) [43, 44].

2.1 Related Work

We give a brief overview of prior work on probabilistic collision detection.

2.1.1 Probabilistic Collision Detection for Gaussian Errors

Many approaches to compute the collision probability in uncertain robotic environments approximate the noises using a single Gaussian or a mixture of Gaussian distributions to simplify the computations. Such approaches are widely used in 2D environments for autonomous driving cars to

avoid collisions with cars or pedestrians. Xu et al. [18] use Linear-Quadratic Gaussian to model the stochastic states of car positions on the road. Collision detection under uncertainty is performed by computing the Minkowski sum of Gaussian ellipse boundary and the rectangular car model and checking for overlap with other rectangular car model. Park et al. [1] present an efficient algorithm to compute an upper bound of the collision probability with Gaussian error distributions [1]. This approach can be extended to Truncated Gaussian because the probability density function (PDF) of a Truncated Gaussian inside its ellipsoidal domain has the same value as that of the PDF of a Gaussian. Therefore, the upper bound computed using [1] also holds for Truncated Gaussian error distributions, but the bound is not tight. Moreover, a Truncated Gaussian distribution has a bounded ellipsoidal domain and the integral computations outside the domain can be omitted. As compared to this approach, our new algorithm improves the tightness of the upper bound and the running time, as shown in Section 5.

2.1.2 Probabilistic Collision Detection for Non-Gaussian Errors

The collision probability for non-Gaussian error distributions can be computed with Monte Carlo sampling [45]. However, these methods are much slower (10 – 1000 times), as compared to probabilistic algorithms that use Gaussian forms of error distributions [1]. Althoff et al. [46] use a non-Gaussian probability distribution model on the future states of other cars on the road, based on their positions, speeds, and road geometry. They use a 2D grid discretization of the state space and Markov chain to compute the probability that a car belongs to a cell. This method assumes that the environment sensors has no noise. Lambert et al. [47] use a Monte Carlo approach, taking advantage of the probability density function represented as a Gaussian. Other methods have been proposed for point clouds using classification [17] or Monte-Carlo integration [48]. Approaches based on Partially Observable Markov Decision Processes (POMDPs) make efficient decisions about the robot actions in a partially observable state in an uncertain environment [39, 49]. Some applications using POMDPs [50] have been developed to avoid collisions in an uncertain environment, where the uncertainty is represented with a non-Gaussian probability distribution. Our approach for non-Gaussian distributions is different and complementary with respect to these methods.

2.1.3 Probabilistic Collision Detection: Applications

Many approaches have been proposed for collision checking for general applications. Aoude et al. [51] represent the uncertainty model for point obstacles as a Gaussian Process and positional

error is represented by a Gaussian distribution that propagates over a discretized time domain. The upper bound on the collision probability is computed on the Gaussian positional error with an $\text{erf}(\cdot)$ function for a point obstacle. Fisac et al. [52] compute the collision probability between the dynamic human motion and a robot, and use that value for robot motion planning in the 3D workspace. This algorithm models the human motion based on human dynamics, discretizes the 3D workspace into smaller grids, and integrates the cell probabilities over the volume occupied by the robot. Probabilistic collision detection for a Gaussian error distribution [1] has been used for optimization-based robot motion planning. The collision constraint used in the optimization formulation is that the collision probability should be less than 5% at any robot configuration in the resulting trajectory. However, with Gaussian error distributions, the upper bound of collision probability is rather conservative. As a result, these approaches do not work well in tight spaces or narrow passages.

2.2 Overview

In this section, we introduce the terminology used in the paper and give an overview of our approach. Our algorithm is designed for environments, where the scene data is captured using sensors and only partial observations are available. In this case, the goal is to compute the *collision probability* between two objects, when one or both objects are represented with uncertainties and some of the input information such as positions or orientations of polygons or point clouds are given as probability distributions

2.2.1 Probabilistic Collision Detection

The input of the probabilistic collision detection is two 3D shapes A and B , and two 3D positional error distributions P_A and P_B that are probabilistically independent of each other. The positional error distributions P_A and P_B denote the probability density function over 3D space of translations from the origins of objects A and B , respectively. The output of the algorithm is p_{col} , the probability of in-collision state between A and B , where the objects can be translated with the error distributions.

The collision probability p_{col} , given two input shapes A and B and the error distributions p_A

and p_B , can be formulated as

$$p_{col} = \iiint_{\epsilon_A} \iiint_{\epsilon_B} I((A + \epsilon_A) \cap (B + \epsilon_B) \neq \emptyset) p(\epsilon_A) p(\epsilon_B) d\epsilon_A d\epsilon_B, \quad (2.1)$$

$$\epsilon_A \sim P_A, \quad \epsilon_B \sim P_B, \quad (2.2)$$

where $I(\cdot)$ is an indicator function which yields 1 if the condition is true and 0 otherwise, ϵ_A and ϵ_B are the displacement vectors for A and B with the probability distribution P_A and P_B , and \oplus denotes the Minkowski sum operator between two shapes.

To generalize, we shift only one object A by $\epsilon = \epsilon_A - \epsilon_B$ which follows a probabilistic distribution P_{AB} , instead of shifting the two objects separately by ϵ_A and ϵ_B . Because of the independence of probabilistic distributions P_A and P_B , the convolution P_{AB} of P_A and P_B can be expressed as:

$$f_{AB}(\mathbf{x}) = \iiint_{\mathbf{y}} f_A(\mathbf{y}) f_B(\mathbf{x} - \mathbf{y}) d\mathbf{y}, \quad (2.3)$$

where f_{AB} , f_A , f_B are the probability density functions of P_{AB} , P_A , P_B , respectively.

2.2.2 Probabilistic Collision Detection for Gaussian Error

The general probabilistic collision detection problem is hard to solve, when the error distributions P_A and P_B have any arbitrary form. The convolution operator in (Equation (2.3)) can be hard to formulate in the general case. However, it is known that the convolution of two Gaussians is also Gaussian. This generalizes the use of two error distributions into one, yielding the following:

$$p_{col} = \iiint_{\epsilon} I((A + \epsilon) \oplus B \neq \emptyset) p(\epsilon) d\epsilon \quad (2.4)$$

$$= \iiint_{\epsilon} I(\epsilon \in (-A) \oplus B) p(\epsilon) d\epsilon, \quad \epsilon \sim P_{AB}. \quad (2.5)$$

Probabilistic collision detection with the Gaussian distribution condition can be solved efficiently [1], where P_A and P_B also correspond to Gaussian distributions. This algorithm computes a good upper bound on collision probability for convex and non-convex shapes by efficiently linearizing the

Gaussian along the minimum displacement vector direction. In practice, the resulting bounds are conservative.

2.3 Truncated Gaussian Mixture Model Error Distribution

In this section, we present an efficient algorithm for Truncated Gaussian Mixture Model (TGMM) error distributions, which is a more general type of noise model for robotics applications. To compute the collision probability for TGMM, we first introduce the solutions for simpler error distributions corresponding to Truncated Gaussian (TG) and Weighted Samples (WS). We combine these two algorithms to design an algorithm for a multiple Truncated Gaussian error distribution model.

2.3.1 Truncated Gaussian Mixture Models

A TGMM consists of multiple Truncated Gaussian (TG) distributions, each distribution with a truncated domain. The probability density function of a TG, f_{TG} , can be formulated as:

$$f_{TG}(\mathbf{x}; \mu, \Sigma, r) = \begin{cases} \frac{1}{\eta} g(\mathbf{x}; \mu, \Sigma) & (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \leq r \\ 0 & \text{otherwise} \end{cases}, \quad (2.6)$$

where g is the probability density function of a Gaussian, μ is the mean, Σ is the variance, r is the radius of bound in the coordinates of the principal axes, and η is the truncation rate used to compensate the loss of truncated volume of probability outside the bound. A TGMM consists of n TGs with multiple weights w_i . The probability density function of the TGMM, f_{TGMM} , can be formulated using the definition of f_{TG} in Equation (2.6), as:

$$f_{TGMM}(\mathbf{x}) = \sum_{i=1}^n w_i f_{TG}(\mathbf{x}; \mu_i, \Sigma_i, r_i), \quad \sum_{i=1}^n w_i = 1. \quad (2.7)$$

As the radii of TGs decrease and converge to zero, the probability model behaves like a discrete probability distribution, which we call Weighted Samples (WS). The WS is a discrete probability distribution, formulated as:

$$P(\mathbf{X} = \mathbf{x}_i) = w_i, \quad \sum_{i=1}^n w_i = 1 \quad (2.8)$$

where \mathbf{x}_i is a sample in \mathbb{R}^d , and w_i is a weight of the sample, for $i = 1, \dots, n$.

2.3.2 Collision Probability for Truncated Gaussian

The TG is a Gaussian with a specific form of bounded domain. The bounded domain for 3D Truncated Gaussian is an ellipsoid, centered at the Gaussian mean and having the same principal axes as those of Gaussian variances. The TG is formulated with a collision probability function as

$$p_{col} = \iiint_{V_{AB}} f_{TG}(\mathbf{x}; \mu, \Sigma, r) d\mathbf{x}, \quad (2.9)$$

where $V_{AB} = -A \oplus B$, f_{TG} is the probability density function for TG, μ is the mean, Σ is the variance, r is the radius of bound in the coordinates of the principal axes, and η is the normalization constant used to compensate the loss of truncated volume of probability outside the bound. Because f_{TG} has the value of a Gaussian multiplied by η inside the boundary, the integral volume becomes $-A \oplus B \cap V_{TG}$, where V_{TG} is the valid volume of Truncated Gaussian. The collision probability corresponds to

$$p_{col} = \frac{1}{\eta} \iiint_{V_{AB} \cap V_{TG}} f_{TG}(\mathbf{x}; \mu, \Sigma, r) d\mathbf{x}. \quad (2.10)$$

The TG has its center at μ and principal axes with different lengths determined by Σ . To normalize the function, a transformation $T = \Sigma^{-1/2} - \mu I$ is applied to the coordinate system, which changes Equation (2.10) to

$$p_{col} = \frac{1}{\eta \det \Sigma} \iiint_{V'_{AB} \cap V'_{TG}} f_{TG}(\mathbf{x}; \mathbf{0}, I, r) d\mathbf{x}, \quad (2.11)$$

$$V'_{AB} = T(V_{AB}), \quad V'_{TG} = T(V_{TG}). \quad (2.12)$$

In the transformed coordinate system, V'_{TG} is a sphere of radius r .

Unfortunately, there is no explicit or analytic form of solution for the integral of a Gaussian distribution over the intersection of a non-convex volume V'_{AB} and a ball V'_{TG} . In order to simplify the problem, we initially assume that A and B are convex, and so are V_{AB} and V'_{AB} . Instead of computing the exact integral, we compute an upper bound on the collision probability. The

computation of collision probability reduces to the computation of the integral

$$\iiint_{V'} g(\mathbf{x}; \mathbf{0}, I) d\mathbf{x}, \quad (2.13)$$

where $V' = V'_{AB} \cap V'_{TG}$, and $g(\cdot)$ is the Gaussian probability density function.

From the convexity of V'_{AB} , the minimum distance vector \mathbf{d}' between the origin and V'_{AB} can be computed by using the GJK algorithm [53] between A' and B' , which are transformed from A and B by T . Let \mathbf{n}'_d be the unit directional vector of \mathbf{d}' . Then, by the Cauchy-Schwarz inequality $(\mathbf{x} \cdot \mathbf{n}'_d)^2 \leq \|\mathbf{x}\|^2$, the integral is bounded by

$$p_{col} \leq \iiint_{V'} \frac{1}{\sqrt{8\pi^3}} \exp\left(-\frac{1}{2}(\mathbf{x} \cdot \mathbf{n}'_d)^2\right) d\mathbf{x}. \quad (2.14)$$

The integrand of the upper bound term behaves as a 1D Gaussian function instead of being the 3D function. We use the divergence theorem to compute the upper bound on collision probability (2.14).

$$\iiint_{V'} \text{div}(\mathbf{F}) dV = \oint_{S'} (\mathbf{F} \cdot \mathbf{n}_S) dS, \quad (2.15)$$

where \mathbf{F} is a vector field, S' is the surface of V' , dS is an infinitesimal area for integration, and \mathbf{n}_S is the normal vector of dS . This converts the volume integral to a surface integral. Let's define \mathbf{F} as

$$\mathbf{F}(\mathbf{x}) = \frac{1}{2\pi} \left(1 + \text{erf}\left(\frac{\mathbf{x} \cdot \mathbf{n}'_d}{\sqrt{2}}\right)\right) \mathbf{n}'_d, \quad (2.16)$$

where $\text{erf}(\cdot)$ is the 1D Gaussian error function. Note that \mathbf{F} is a vector field with a single direction \mathbf{n}'_d . The directional derivative of $\mathbf{F}(\mathbf{x})$ along any directional vector orthogonal to \mathbf{n}'_d is zero because \mathbf{F} varies only along \mathbf{n}'_d . The divergence of \mathbf{F} thus becomes $(\partial \mathbf{F} / \partial \mathbf{n}'_d)$, and this is equal to the function in Equation (2.14).

We apply the divergence theorem in Equation (2.15) to the volume integral on V' in Equation (2.14). Note that V' is a 3D volume intersection between a non-convex polytope V'_{AB} and a ball V'_{TG} . The surface integral on the intersection between a non-convex polytope and a ball can be

decomposed into two parts and bounded by the sum of two components as

$$\sum_i \oint_{\Delta S'_i} (\mathbf{F} \cdot \mathbf{n}'_i) dS + \oint_{S'_{TG}} (\mathbf{F} \cdot \mathbf{n}_S) dS, \quad (2.17)$$

where S'_i is the i -th triangle of V'_{AB} inside V'_{TG} , \mathbf{n}'_i is the normal vector of $\Delta S'_i$, and S'_{TG} is the spherical boundary of V'_{TG} outside of a plane defined by \mathbf{d}' . The second term corresponds to the spherical domain of the normalized Truncated Gaussian with the truncation rate η . The magnitude of F on the spherical boundary V'_{TG} is upperly bounded by $(1 - \eta)$, because it is the cumulative distribution function on the boundary. The surface area of S'_{TG} is less than $\pi \|\mathbf{d}'\|^2$. This can be used to express a bound based on the following lemma.

Lemma 2.3.1. The collision probability represented in a volume integral is upperly bounded by a surface integral as follows:

$$p_{col} = \iiint_{V'} g(\mathbf{x}; \mathbf{0}, I) d\mathbf{x} \quad (2.18)$$

$$\leq \sum_i \oint_{\Delta S_i} (\mathbf{F} \cdot \mathbf{n}_i) dS + \pi(1 - \eta) \|\mathbf{d}'\|^2, \quad (2.19)$$

where F is a vector field in 3D space whose maximum magnitude is $1/\pi$, and S_i is the i -th triangle of V'_{AB} that is inside V'_{TG} .

Because the error function integral over a triangle domain is hard to compute, the upper bound on the integral is evaluated as

$$\sum_i \oint_{\Delta S_i} (\mathbf{F} \cdot \mathbf{n}_i) dS \quad (2.20)$$

$$\leq \sum_i \left(\max_{j=1,2,3} \mathbf{F}(S_{ij}) \cdot \mathbf{n}_i \right) \text{Area}(\Delta S_i), \quad (2.21)$$

where S_{ij} is the j -th vertex of the triangle S_i for $j \in \{1, 2, 3\}$. The upper bound on the collision probability corresponds to the sum of the maximum of F at the points of each triangle, multiplied by the area of the triangle, over the surface of $V'_{AB} \cap V'_{TG}$.

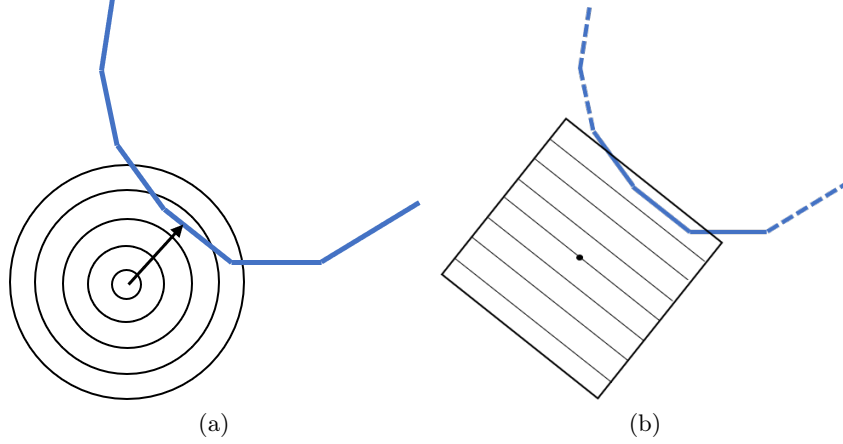


Figure 2.1: (a) Contour plots of the bivariate TG distribution. (b) Contour plots of the bounded function F for TG are not used in the calculation of collision probability and thereby reduce the running time of collision probability computation.

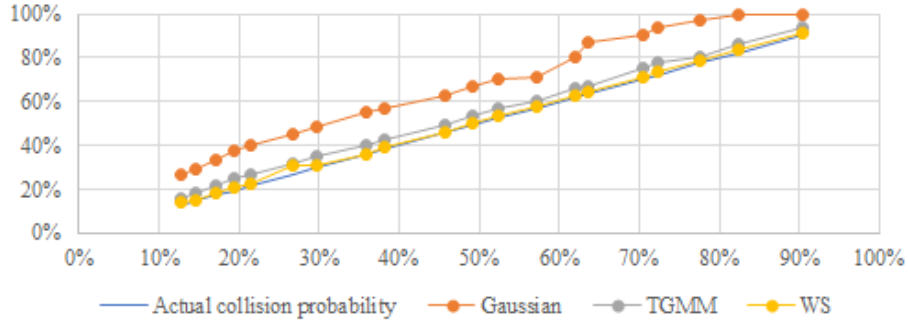


Figure 2.2: The upper bound of collision probability with uncertainty approximated as Gaussian, Gaussian Mixture, and Weight Samples. The X-axis is the true collision probability computed using Monte Carlo methods, and Y-axis is the computed probability using different methods. The computed upper bound for Gaussian Mixture and Weights Samples are closer to the ground truth/exact answer, than that for a single Gaussian approximation. The collision probability over-estimation with TGMM is reduced by 90%, compared to the one with Gaussian distribution.

2.3.3 Efficient Evaluation of the Integral

In order to reduce the running time of computing the surface integral, we take advantage of the bound of TG. The domain of surface integral is $V' = V'_{AB} \cap V'_{TG}$, where V'_{AB} consists of many triangles and V'_{TG} is a sphere of radius r . This sphere is tightly bounded by a cube, with one normal parallel to the direction of shortest displacement vector \mathbf{d}' . Therefore, the triangles of V'_{AB} that are outside the cube do not count towards the surface integral. So, we accumulate the upper bound function value in Equation (2.21) only for the triangles that lie inside the cube boundary, and ignore the triangles outside the boundary. For the triangles that intersect the cube boundary, the upper bound function value is computed for the intersecting primitives. Because the approximated integral

for collision probability is bounded by the cube, primitives outside the cube can be ignored in terms of calculating the upper bound of collision probability. Limiting the computation to the truncated primitives can accelerate the running time.

In order to perform this computation for non-convex primitives, we construct bounding volume hierarchies (BVHs) for A and B , with each bounding volume being an oriented bounding box. During the traversal of the BVHs, the oriented bounding boxes are first transformed by T . The transformed bounding volumes are still convex primitives, the surface integral can be obtained using Equation (2.21).

2.3.4 Error Distribution as Weighted Samples

For the weighted samples, the probability distributions are given by multiple points p_i with weights w_i , yielding a discrete probability distribution, as described in Equation (2.8). The collision probability of Equation (2.4-2.5) for the weighted samples is given as:

$$p_{col} = \sum_{i=1}^n w_i I(\mathbf{x}_i \in (-A) \oplus B), \quad \sum_{i=1}^n w_i = 1, \quad (2.22)$$

where w_i is weight and $I(\cdot)$ is the indicator function which yields 1 if the statement inside is true or 0 otherwise. The formulation is the weighted average of n collision detection results. A simple solution to this problem is to run exact collision detection algorithms n times and sum up the weights of in-collision cases. However, this results in an $O(n)$ and we use BVHs to accelerate that computation.

We have the bounding volumes for the weighted samples and the two polyhedra. When there is no overlap between the bounding boxes, it implies that there is no collisions between two shapes for all weighted samples in the corresponding bounding volume. If the bounding volumes overlap, there may be a collision for each weighted samples, and the bounding volumes of the children are checked recursively for collisions. Each of these bounding volume checks can be performed in $O(1)$ time.

If we want to compute an upper bound of collision probability, the running time can be further reduced by replacing detailed computation of collision probability with a simple upper bound. We introduce a user-defined parameter δ which we call the “confidence level”. During the traversal of bounding volume traversal tree, the upper bound of collision is the sum of weights of samples that belong to the bounding volume. If the upper bound is less than the confidence level $1 - \delta$, the traversal stops and the sub-routine returns the sum of weights as an upper bound of collision

probability.

In order to reduce the time complexity for more complex forms of error distributions, we construct a Bounding Volume Hierarchy (BVH) [13] over the error distributions of mixture models with Oriented Bounding Boxes (OBBs) [54] and apply the collision probability algorithm on its nodes, which are convex primitives. We construct a BVH for the weighted samples and for Truncated Gaussian Mixture Models in $O(n)$ time complexity. The BVH is generated from the root node that contains every Truncated Gaussians, and the bounding volume for the root node is computed by minimizing the volume of the oriented bounding box. Next, the bounding volume is split at the center along the longest edge and two child BVH nodes are generated, each containing appropriate samples. This process is repeated till the leaf nodes.

2.3.5 Error Distribution as Truncated Gaussian Mixture Models

For TGMM the probability distribution is given as:

$$p_{col} = \sum_{i=1}^n w_i \iiint_{V_{AB}} \eta_i f_{TGMM}(\mathbf{x}; \mu_i, \Sigma_i, r_i) d\mathbf{x}, \quad (2.23)$$

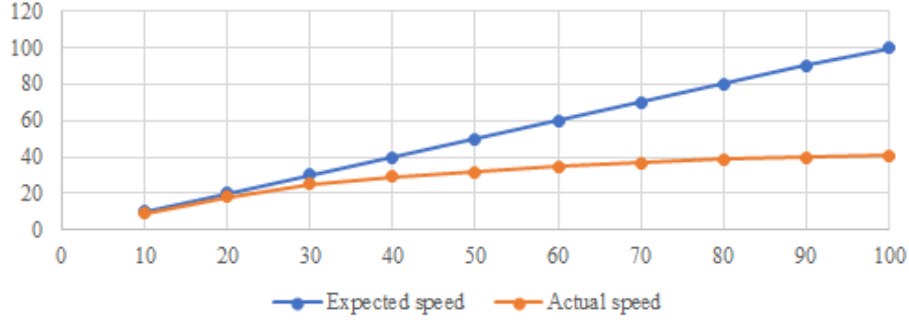
where n is the number of Truncated Gaussians (TGs), w_i is the weight of each TG, and μ_i , Σ_i and r_i are the mean, variance and radius of TGs, respectively. The overall algorithm for TGMM is obtained by combining the two previous algorithms. A change from the algorithm for weighted samples is that the BVH is constructed for n TGs with their ellipsoid bounds instead of the point samples. The details of algorithms and pseudo-codes are given in the appendix [55].

2.4 Performance and Analysis

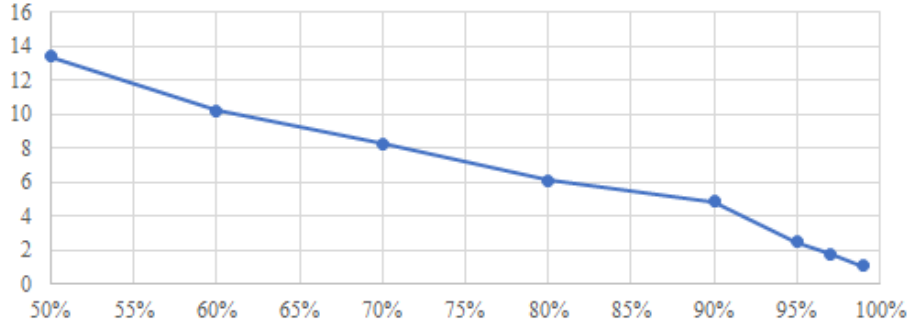
In this section, we describe our implementation and highlight the performance of our probabilistic collision detection algorithms on synthetic and real-world benchmarks. Furthermore, we measure the upper bound of collision probabilities and speedups for algorithms with different noise distributions, compared to the exact collision probability computed by Monte Carlo method.

2.4.1 Probabilistic Collision Detection: Performance

For the translational error distribution, we first generate a ground truth distribution by randomizing the parameters of TGMM. Next, we sample 100,000 points from the distribution. We run expectation-maximization from these samples to find parameters of single Gaussian, single TG, WS, and TGMM. The resultant distributions are different from the ground truth and count toward



(a)



(b)

Figure 2.3: (a) Speedup of weighted samples (expected) case compared to Monte Carlo (actual) with between 10 to 100 samples (X-axis). (b) Speedup of Truncated Gaussian case, compared to the running time of probabilistic collision detection with a Gaussian. X-axis is the untruncated volume of Gaussian, meaning 100% is the Gaussian and lower value indicates smaller bound. As the truncation boundary shrinks up to 50% of the volume of Gaussian, the algorithm with TG is 14x faster times than the algorithm Gaussian distribution.

collision probability over-estimation.

Figure 2.2 shows the collision probabilities of noise models approximated with Gaussian, TGMM of 10 TG distributions, and 100 WS. We observe that the approximation with a single Gaussian yields rather high and conservative value of collision probability, compared to the ground truth collision probability. Figure 2.3 (a) shows the speedup of probabilistic collision detection with WS (Algorithm 2 in the Appendix) over the Monte Carlo method. The collision probability computation with Monte Carlo counts the sum of sample weights for every Weighted Sample that is in collision, and is similar to an exact collision detection algorithm. Thus, the running time of Monte Carlo increases linearly as the number of Weighted Samples increases. Figure 2.3 (b) shows the speedup of probabilistic collision detection with a TG model (Algorithm 1 in the Appendix) compared to probabilistic collision detection with a Gaussian error distribution. In case of TGs, BVH traversal is

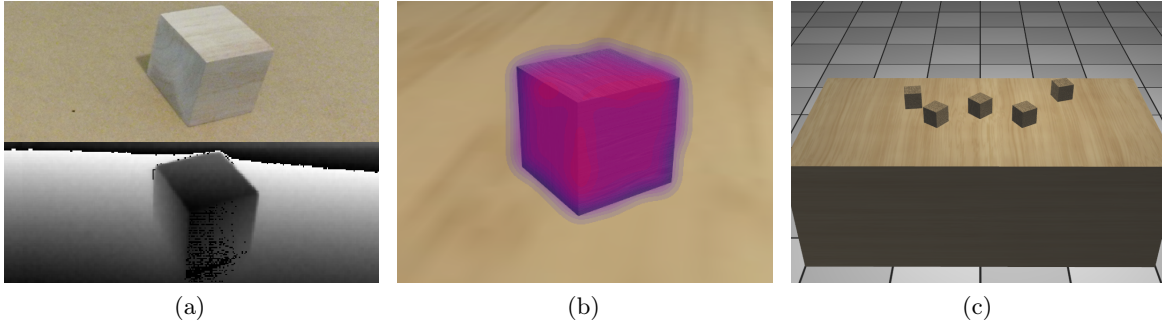


Figure 2.4: (a) A captured RGBD image. The depth values of the table and the wood block have noises, even in adjacent frames. The TG noise of each point particle of the wood block contributes to the overall TGMM model. (b) A reconstructed 3D model of a wood block with TGMM, which is bounded around the wooden block and more accurate than Gaussian distribution, which has unbounded probability density function. (c) A reconstructed 3D robot environment with error distributions on the table and the wood blocks. The wood blocks placed in a zig-zag pattern result in 4 narrow passages for the robot.

not performed when the truncation boundary of TGs does not overlap with the Minkowski sum of bounding volumes for the two objects. On the other hand, for Gaussian distribution there are no truncation boundaries and the BVH traversal continues. Therefore, we observe speedup with TGs over Gaussian, as shown in Figure 1 and Table 1. Overall, the speedup depends on the range of truncation. A smaller truncation boundary results in faster performance of our probabilistic collision detection algorithm.

2.4.2 Sensor Noise Models for Static Obstacles

In a real-world setting, we add a noise model to the point cloud data. The variance is chosen based on the Kinect sensor uncertainty. The input depth images have noise in each pixel and, according to [11], the noise of each pixel can be approximated with a 1D Gaussian. Thus, noise in the pixels of an object are combined with a Gaussian Mixture noise model for the object. Depth images of the wooden blocks and the table have noise in each pixel. We capture sequences of the depth images. In the experiments, the approximate poses of tables and wooden blocks are known a priori. On the boundary of the wooden blocks, some pixels are always classified as a wooden block, some other pixels are classified as a wooden block or as background in different frames, and other pixels are always classified as background. From these boundary pixels, the variance in x- and y-axis noise on the Kinect sensor coordinate frame can be set to the thickness of the boundary. The variance in z-axis motion is computed from the always-wooden-block pixels. The depth values on those pixels differ from frame to frame. After the variance of noise Gaussian is calculated, the mean

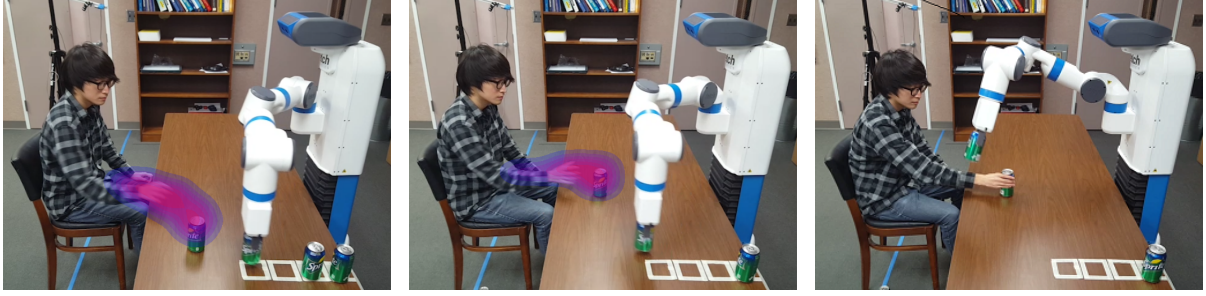


Figure 2.5: We highlight the benefits of our novel probabilistic collision detection with a Truncated Gaussian error distribution. Our formulation is used to accurately predict the future human motion and integrated with a motion planner for the 7-DOF Fetch robot arm. As compared to prior probabilistic collision detection algorithms based on Gaussian distribution [1], our new method improves the running time by 2.6x and improves the accuracy of collision detection by 9.7x.

and variance of a Truncated Gaussian are the same. The truncation rate η with which the integral in the ellipsoidal boundary is set to 90% in our benchmarks. With the fixed truncation rate, the positional error is bounded around the wooden blocks, unlike the positional error represented by Gaussians with an unbound domain. The confidence level δ is related to the robot motion planner, constraining that the collision probability between the robot and the objects should be less than $1 - \delta$ at any robot trajectory point. In our benchmarks, δ is set to 95%.

Figure 2.4 shows a captured depth image, a noise distribution modeled using Truncated Gaussian Mixture Model, and a reconstructed 3D environment with noises. The pixels on the boundary of the object have higher variance in terms of noise. So, the Truncated Gaussian Mixture noise model may have some Gaussians with higher variance. A principal axis for those boundary pixels is perpendicular to the boundary direction.

2.4.3 Robot Motion Planning

The probabilistic collision detection algorithm is used in optimization-based motion planning [12]. We use a 7-DOF Fetch robot arm in the motion planning. We highlight its performance in terms of improved accuracy and faster running time in Figure 2.5. In the robot environment, there is a table in front of the robot and the wood blocks on the table are the static obstacles of the environment. The wood blocks are placed in a manner that pairs of them result in a narrow passage for the robot’s end-effector. The environment is captured using two depth sensors. One sensor is Primesense Carmine 1.09 sensor, the robot head camera. Another is Microsoft Kinect 2.0 sensor installed in the opposite direction with respect to the robot. The point clouds of the table and the wood blocks

Algorithm	Collision Probability Over-estimation (%p)			Running Time (ms)		
	Min	Max	Avg	Min	Max	Avg
CD-Obstacles	0	0	0 (0)	2.8	8.0	3.5 (0.75)
CD-Points	0	100	23 (16)	2300	2800	2500 (130)
PCD-Gaussian [1]	8.6	35	15 (5.2)	15	100	27 (23)
PCD-TG	2.5	13	5.0 (3.3)	5.2	86	12 (4.6)
PCD-WS	0.72	8.1	1.5 (0.60)	130	220	180 (32)
PCD-GMM	1.9	7.7	6.2 (3.3)	85	480	240 (55)
PCD-TGMM	0.26	3.3	0.8 (0.32)	35	130	97 (17)

Algorithm	# Passages	Success Rate	Distance (cm)
CD-Obstacles	4/4	10/10	1.2 (0.09)
CD-Points	1/4	2/10	13 (8.8)
PCD-Gaussian [1]	3/4	8/10	7.9 (2.6)
PCD-TG	3/4	7/10	6.0 (1.3)
PCD-WS	4/4	9/10	4.5 (0.66)
PCD-GMM	3/4	8/10	7.1 (1.0)
PCD-TGMM	4/4	9/10	3.7 (0.46)

Table 2.1: **Performance of probabilistic collision detection algorithms:** Evaluated as part of a motion planner with sensor data. The collision probability over-estimation is shown as percent point (%p) with the minimum and maximum over-estimation. The values corresponds to the average over the time of the robot trajectory with standard deviation in parenthesis. The best performance is obtained PCD-TGMM algorithm in terms of collision probability estimation (i.e. tight bounds), successful handling of narrow passages, computing collision-free trajectories, among different algorithms.

captured by the two sensors are used to reconstruct the environment. In this case, the reconstructed table surface and wood block obstacles have errors due to the noise in the depth sensors. Figure 2.4 (c) shows the reconstructed environment from depths sensors and the error distributions around the obstacles. The robot arm’s task is to move a wood block, drawing a zig-zag pattern that passes through the narrow passages between the wood blocks. The objective is to compute a robot trajectory that minimizes the distance between the robot’s end-effector and the table, and not resulting in any collisions. The following metrics are used to evaluate the performance:

- Collision Probability Over-estimation: Because we compute the upper bounds of collision probability in our algorithm, we measure the extent of collision probability over-estimation, the gap between the upper bound of collision probability and the actual collision probability.
- Running Time: The running time of collision detection algorithm. This excludes the running time of motion planning algorithm.

- # Passages: The successful number of passes the robot makes between wood blocks.
- Success Rate: The number of collision-free trajectories, out of the total number of trajectory executions. For each execution, the wood block’s positions are set following the error distribution.
- Distance: The distance between the robot’s end-effector and the table. A lower value is better, as it implies that the robot can interact with the environment in close proximity.

The collision probability over-estimation, running time, distance values are measured for robot poses of every 1/30 seconds over the robot trajectories.

We compare the performances of 5 different collision detection algorithms: exact collision detection with static obstacles without environment uncertainties (CD-Obstacles), exact collision detection with point clouds (CD-Points), probabilistic collision detection with Gaussian errors (PCD-Gaussian) [1], PCD with Truncated Gaussian (PCD-TG), PCD with weighted samples (PCD-WS), and PCD with Truncated Gaussian Mixture Model (PCD-TGMM). The weighted samples are drawn from the TGMMs. For each TG distribution, one sample is drawn from its center. Other samples are drawn from three icosahedrons with the same centers and different radii by uniformly dividing the truncation radius.

Benefits of Truncated Gaussian: Table 2.1 shows the results of robot motion planning in the various scenarios with different algorithms. The robot motion planning without uncertainties (CD-Obstacles) operates perfectly. However, under sensor uncertainties, the exact collision detection with the point clouds (CD-Points) works poorly. Due to the sensor uncertainties, a high error at a pixel affects the collision detection query accuracy and the performance in narrow passages and success rate. Our probabilistic collision detection algorithms results in better performances than exact collision detection algorithm under the environment with sensor uncertainties. Compared to PCD-Gaussian and PCD-GMM, our algorithms for non-Gaussian distributions (PCD-WS, PCD-TGMM) demonstrate better performances w.r.t. different metrics. Figure 2.6 (appendix) highlights the results: using probabilistic collision detection with Gaussian error distribution (Figure 2.6 (a)); with TGMM error distribution (Figure 2.6 (b)). CD-Obstacles and CD-Points correspond to the exact case, where the poses are known and there is no uncertainty (i.e. the ground truth). The robot’s trajectory is shown in the video.

Human Motion Prediction: We also evaluate our algorithm on scenarios with humans operating close to the robot, as shown in Fig. 1 and the video. In order to handle the uncertainty of future human motion, we use probabilistic collision detection between the robot and the predicted future human pose. Improvement in the accuracy of motion prediction results in better trajectories in terms of being collision-free, smoother and being able to handle tight scenarios [56]. We highlight the benefits of our PCD-TGMM algorithm on the resulting trajectory computation in the video.

2.5 Conclusion and Limitations

We present efficient probabilistic collision detection algorithms for the following forms of non-Gaussian error distributions: Truncated Gaussian, Weighted Samples, and Truncated Gaussian Mixture Model. Compared to the exact collision detection algorithm and prior probabilistic collision detection algorithms for Gaussian error distribution, our new method can compute a tighter upper bound of collision probability and improves the running time. We have integrated this algorithm with a motion planner and highlights its benefits in narrow passage scenarios with a 7-DOF robot arm. Our algorithm can be used to model non-Gaussian error distributions from noisy depth sensors and predicted human motion models.

Our approach has some limitations. The truncation on the Truncated Gaussians has a form of ellipsoid with the center and principal axes that is the same as the original Gaussian distribution. The ellipsoidal shape of truncation boundary may not be sufficient for representing general error distributions. Another realistic possibility would be to truncate using planar boundaries. In our future work, we would like to develop an algorithm for more generic non-Gaussian positional errors. We only consider the positional errors on obstacles and omit the rotational errors. The rotational error cannot be approximated by Truncated Gaussian Mixture Model. As part of future work, we would like to represent a rotational error distribution in the quaternion space, or in the affine space. Furthermore, we assume that the noises of two objects A and B are independent, even though they may arise from the same source.

Appendix

We present some background on truncated Gaussian distributions as well as more details about our probabilistic collision detection algorithm and its applications to motion planning.

Algorithm 1 $p_{col} = \text{PCD_TG}(T_A, T_B, \mu, \Sigma, r, \delta)$

: Compute the collision probability between polyhedra A and B , given precomputed BVHs for polyhedra T_A and T_B , Truncated Gaussian parameters μ, Σ, r , and a confidence level δ .

Input: BVHs T_A and T_B , a Truncated Gaussian $p(\cdot; \mu, \Sigma, r)$, and a confidence level δ

Output: Upper bound on collision probability p_{col}

```

1:  $V_A = T_A.\text{root}$ 
2:  $V_B = T_B.\text{root}$ 
3:  $V'_{AB} = T(-V_A \oplus V_B)$ 
4: if IsNotCollision( $V'_{AB}$ , Sphere( $r$ )) then
5:   return 0
6: end if
7:  $\mathbf{d}' = \text{GJK}(T(V_A), T(V_B))$ 
8:  $V'_{TG} = \text{cube}(\text{norm}(\mathbf{d}'), r)$ 
9:  $p_{col} := 0$ 
10: for all  $i$  of  $V'_{AB}$  do
11:   if  $\triangle S_{i1} S_{i2} S_{i3} \cap V'_{TG} \neq \emptyset$  then
12:     Add Equation (2.21) to  $p_{col}$ 
13:   end if
14: end for
15: if  $p_{col} \leq \delta$  or both  $T_A$  and  $T_B$  are leaf nodes then
16:   return  $p_{col}$ 
17: end if
18: if  $V_A$  has children and  $\text{Vol}(V_A) \geq \text{Vol}(V_B)$  then
19:   return  $p_{col} = \sum_{c_A: \text{child}} \text{PCD\_TG}(c_A, T_B, \mu, \Sigma, r, \delta)$ 
20: else
21:   return  $p_{col} = \sum_{c_B: \text{child}} \text{PCD\_TG}(T_A, c_B, \mu, \Sigma, r, \delta)$ 
22: end if
```

2.5.1 Probabilistic Collision Detection

We present the detailed pseudo-code of our novel probabilistic collision detection algorithms for Truncated Gaussian and weighted samples.

Algorithm 1 describes the overall process of computing the upper bound of collision probability for TG error distribution. Line 3 computes the Minkowski sum of two bounding volumes and transforms that sum using T . Because the bounding volumes are oriented bounding boxes, the time complexity is $O(1)$. Line 7 computes the minimum distance vector in nearly constant time and is used to define the function F for Equation (2.21). The approximated cube boundary V'_{TG} of TG is computed at Line 8, whose center is at the origin and one of the normal directions is parallel to \mathbf{d}' , and whose half-length is r . In Line 11-12, the upper bound of collision probability is computed. If the computed collision probability bound is less than the confidence level, as shown in Line 15, the

Algorithm 2 $p_{col} = \text{PCD_WS}(T_A, T_B, T_{WS}, \delta)$

: Compute the collision probability between polyhedra A and B , given precomputed BVHs for polyhedra T_A and T_B , a BVH T_{WS} for the weighted samples, and a confidence level δ .

Input: BVHs T_A, T_B, T_{WS} , confidence level δ

Output: Upper bound on collision probability p_{col}

```

1:  $V_A = T_A.\text{root}$ 
2:  $V_B = T_B.\text{root}$ 
3:  $V_{WS} = T_{WS}.\text{root}$ 
4: if IsNotCollision( $V_{WS}, -V_A \oplus V_B$ ) then
5:   return 0
6: end if
7: if  $T_{WS}.\text{sum\_weights} < \delta$  or  $V_A, V_B, V_{WS}$  are leaf nodes then
8:   return  $p_{col} = T_{WS}.\text{sum\_weights}$ 
9: end if
10: if  $V_A$  has children and  $\text{Vol}(V_A) \geq \text{Vol}(V_B)$  and  $\text{Vol}(V_A) \geq \text{Vol}(V_{WS})$  then
11:   return  $p_{col} = \sum_{c_A:\text{child}} \text{PCD\_WS}(c_A, T_B, T_{WS}, \delta)$ 
12: else if  $V_B$  has children and  $\text{Vol}(V_B) \geq \text{Vol}(V_A)$  and  $\text{Vol}(V_B) \geq \text{Vol}(V_{WS})$  then
13:   return  $p_{col} = \sum_{c_B:\text{child}} \text{PCD\_WS}(T_A, c_B, T_{WS}, \delta)$ 
14: else
15:   return  $p_{col} = \sum_{c_{WS}:\text{child}} \text{PCD\_WS}(T_A, T_B, c_{WS}, \delta)$ 
16: end if
```

algorithm returns the bound and stops the traverse. Otherwise, the algorithm recursively traverses to child nodes of the BVHs.

Algorithm 2 summarizes the probabilistic collision detection for weighted samples as error distributions. The algorithm is given with three precomputed bounding volume hierarchies. Lines 4-5 check for collisions between the bounding volumes and returns 0 when there is no overlap. Lines 7-8 check whether the upper bound of collision is less than the confidence level δ . If the condition is satisfied, it returns the sum of weights instead of further traversing the tree.

Algorithm 3 corresponds to the combined algorithm of Algorithm 2 and 1, when the input error distribution is a TGMM. The first collision check is performed on Line 4, similar to the case for the Weighted Samples, returning 0 in Line 5 if there is no overlap. Because the maximum possible value of collision probability for a single TG is its weight, the confidence level check in Line 7 is also performed similarly, comparing the sum of weights of the TGs with the confidence level δ . In Line 10, when the traversal reaches a leaf node of the BVH for TGMMs, it returns the collision probability obtained from PCD_TG (Algorithm 1). Otherwise, the BVH traversal continues until it

Algorithm 3 $p_{col} = \text{PCD_TGMM}(T_A, T_B, T_{TGMM}, \delta)$

: Compute the collision probability between polyhedra A and B , given precomputed BVHs for polyhedra T_A and T_B , a BVH T_{TGMM} for the TGMMs, and a confidence level δ .

Input: BVHs T_A, T_B, T_{TGMM} , confidence level δ

Output: Upper bound on collision probability p_{col}

```

1:  $V_A = T_A.\text{root}$ 
2:  $V_B = T_B.\text{root}$ 
3:  $V_{TGMM} = T_{TGMM}.\text{root}$ 
4: if IsNotCollision( $V_{TGMM}, -V_A \oplus V_B$ ) then
5:   return 0
6: end if
7: if  $T_{TGMM}.\text{sum\_weights} < \delta$  then
8:   return  $p_{col} = T_{TGMM}.\text{sum\_weights}$ 
9: end if
10: if  $T_{TGMM}$  is a leaf node then
11:   return  $\text{PCD\_TG}(T_A, T_B, T_{TGMM}.\mu, T_{TGMM}.\Sigma, T_{TGMM}.r, \delta)$ 
12: end if
13: if  $V_A$  has children and  $\text{Vol}(V_A) \geq \text{Vol}(V_B)$  and  $\text{Vol}(V_A) \geq \text{Vol}(V_{TGMM})$  then
14:   return  $p_{col} = \sum_{c_A:\text{child}} \text{PCD\_TGMM}(c_A, T_B, T_{TGMM}, \delta)$ 
15: else if  $V_B$  has children and  $\text{Vol}(V_B) \geq \text{Vol}(V_A)$  and  $\text{Vol}(V_B) \geq \text{Vol}(V_{TGMM})$  then
16:   return  $p_{col} = \sum_{c_B:\text{child}} \text{PCD\_TGMM}(T_A, c_B, T_{TGMM}, \delta)$ 
17: else
18:   return  $p_{col} = \sum_{c_{TGMM}:\text{child}} \text{PCD\_TGMM}(T_A, T_B, c_{TGMM}, \delta)$ 
19: end if
```

reaches the leaf nodes of BVHs.

2.5.2 Motion Planning

Our probabilistic collision detection algorithm has been integrated with optimization-based motion planning and integrated with the 7-DOF Fetch arm (see Figure 1). The tighter bounds on the collision probability enable us to find collision-free paths in tight spaces or narrow passages.

Tabel 2.1 shows the results of robot motion planning in the narrow passage scenario. Exact collision detection between known shapes without uncertainties (CD-Obstacles) always gives a correct collision state, thus 0%p in collision probability over-estimation measurements. The robot motion planning without uncertainties operates perfectly, passing all narrow passages, generating non-collision trajectories, and keeping the closest distance between the end-effector and the table. However, under sensor uncertainties, motion planning with the exact collision detection between the robot and the point clouds (CD-Points) works poorly. Due to the sensor uncertainties, a high error at a pixel may affect the collision detection query, reporting in-collision though the robot

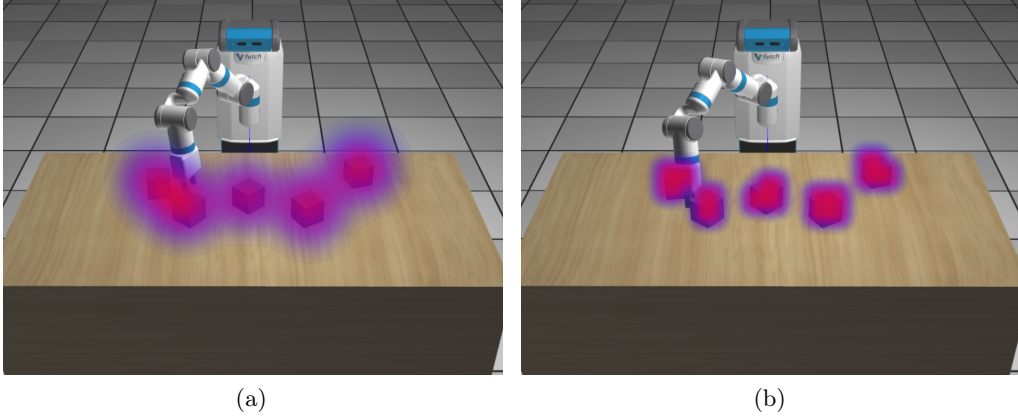


Figure 2.6: The robot trajectory with probabilistic collision detection through narrow passages. (a) The wood block obstacles are captured by RGBD sensors and its positional errors are modeled with Gaussian distributions. In this case, the planner is unable to compute a path in the narrow passage because of the conservative error bounds on the collision probability. (b) Robot motion planning with probabilistic collision detection under error distribution in the form of TGMM can generate a collision-free robot trajectory that passes through the narrow passage, due to tight bounds.

is not in collision, and vice versa. Due to this reason, the average and the standard deviation of collision probability over-estimation is the highest among other algorithms. This also impacts on the worst performance on the number of passages, success rate, and the distance measurements. Motion planning algorithms with probabilistic collision detection algorithms show better performances than the exact collision detection algorithm under the environment with sensor uncertainties. Compared to PCD-Gaussian, the probabilistic collision detection algorithms with non-Gaussian generally show better performances in the metrics in the table. PCD-WS and PCD-TGMM run slower because of the complex shape of error distributions, but compute collision probability more accurately. PCD-TGMM shows the best performance on collision probability estimation, the number of successful passages, successful avoiding obstacles, and the distance between the table surface and the end-effector.

Figure 2.6 shows the results computed using our motion planner with different combination of probabilistic collision detection algorithms on a benchmark that consists of wooden blocks. This wooden block has four narrow passages and have varying sizes. If the collision probability computation results in a higher probability bound, the resulting planner concludes that there is no collision-free trajectory. As a result, the planner is not able to compute a collision-free trajectory for all four cases, which is shown in terms of number of passes (i.e. # Passages) and the success rate in Table. 1. We use this benchmark to evaluate the benefits of computing tighter probabilistic

collision detection bounds. We use an optimization-based planner, ITOMP [12], which repeatedly refines the trajectory while interleaving the execution and motion planning for dynamic scenes. We handle three types of constraints: smoothness constraint, static obstacle collision-avoidance, and dynamic obstacle collision avoidance. In Figure 2.6 (a), the robot motion is planned with the probabilistic collision algorithm used for Gaussian error distribution. However, this planner fails in the configurations corresponding to the narrow passage, even though there is sufficient space for the robot to pass through. Because of the infinite domain of a Gaussian distributions, the non-zero collision probability prevents the robot to pass through, as it conveys a possible collision. In Figure 2.6 (b), the probabilistic collision detection algorithm with TGMM is used in the motion planner. Different from the Gaussian distribution case, the planner can compute a path through the narrow passages. This is because the error distributions have truncation boundaries and the collision detection algorithm yields collision probability of under 5%. Overall, our probabilistic collision detection algorithm provides better bounds than prior methods.

CHAPTER 3

Natural Language Processing for Safe Human-Robot Interaction

In the field of human-robot interaction (HRI), natural language has been used as an interface to communicate a human’s intent to a robot [20, 21, 22, 23]. Much of the work in this area is related to specifying simple tasks or commands for robot manipulation, such as picking up and placing objects. As robots are increasingly used in complex scenarios and applications, it is important to develop a new generation of motion planning and robot movement techniques that can respond appropriately to diverse, attribute-based NLP instructions for HRI, e.g., instructions containing negation based phrases or references to position, velocity, and distance constraints. Furthermore, we need efficient techniques to automatically map the NLP instructions to such motion planners.

Humans frequently issue commands that include sentences with orientation-based or negation constraints such as “put a bottle on the table and keep it upright” or “move the knife but don’t point it towards people,” or sentences with velocity-based constraints such as “move slowly when you get close to a human.” To generate robot actions and movements in response to such complex natural language instructions, we need to address two kinds of challenges:

1. The accurate interpretation of attribute-based natural language instructions and their grounded linguistic semantics, especially considering the environment and the context. For example, a human may say “move a little to the left” or “do not move like this,” and the robot planner needs to learn the correct interpretation of these commands that include spatial and motion-based adjectives, adverbs, and negation.
2. The motion planner needs to generate appropriate trajectories based on these complex natural language instructions. This includes appropriately setting up the motion planning problem based on different motion constraints (e.g., orientation, velocity, smoothness, and avoidance) and computing smooth and collision-free paths.

At a high level, natural language instructions can be decomposed into task description and attributes. Task descriptions are usually verbs or noun phrases that describe the underlying task

performed by a robot. The attributes include various adjectives, adverbs, or prepositional phrases that are used to specify additional conditions the robot must (or must not) satisfy. For example, these conditions may specify some information related to the movement speed, the orientation, the physical space characteristics, or the distances. Therefore, it is important to design motion planners that consider these robotic task descriptions and robot motion constraints.

3.1 Related Work

Most algorithms used to map natural language instruction to robot actions tend to separate the problem into two parts: parsing and motion planning computation. In this section, we give a brief overview of prior work in these areas.

3.1.1 Natural Language Processing

Duvallet et al. [58] use a probabilistic graphical learning model called Generalized Grounding Graphs (G^3) on a ground vehicle for a navigation problem given natural language commands. Branavan et al. [22, 59] use reinforcement learning to learn a mapping from natural language instructions and then apply it to sequences of executable actions. Matuszek et al. [23] use a statistical machine translation model to map natural language instructions to path description language, which allows a robot to navigate while following directions. Duvallet et al. [60] use imitation learning to train the model through demonstrations of humans following directions. Paul et al. [61] propose the Adaptive Distributed Correspondence Graph (ADCG). Arkin et al. [62] further extend DCG, proposing the Hierarchical Distributed Correspondence Graph (HDCG), which defines constraints as discrete inequalities and grounds word phrases to corresponding inequalities. Chung et al. [63] use HDCG on ground vehicles to implement navigation commands and demonstrate performance improvements over G^3 in terms of running time, factor evaluations, and correctness. Oh et al. [64] integrate HDCG with their navigating robot system, measuring performance in terms of completion rates and comparing them to human behaviors. Scalise et al. [65] collected a corpus of natural language instructions from online crowdsourcing that specify objects of interest for “*picking up*” command. The dataset could be used as a training dataset in our method.

3.1.2 Robot Motion Planning in Dynamic Environments

Many replanning algorithms have been suggested to generate collision-free motion plans in dynamic environments. Fox et al. [66] propose the dynamic window approach to compute optimal

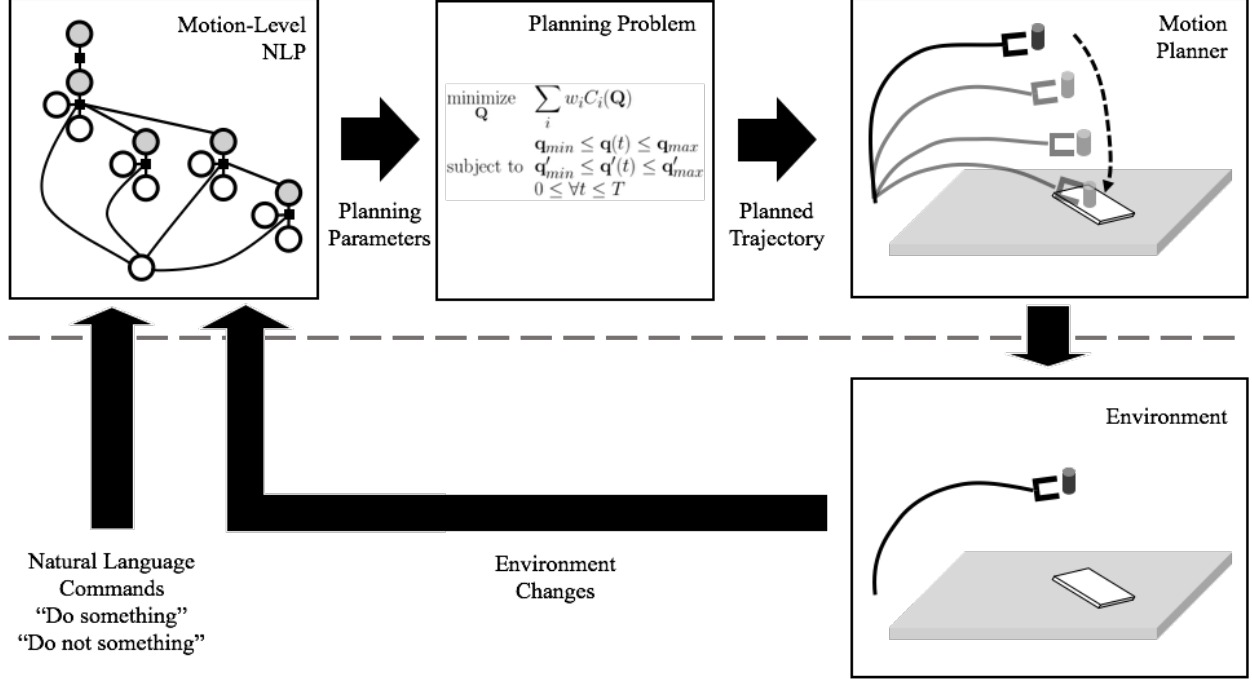


Figure 3.1: The overall pipeline of our approach highlighting the NLP parsing module and the motion planner. Above the dashed line (from left to right): Dynamic Grounding Graphs (DGG) with latent parameters that are used to parse and interpret the natural language commands, generation of optimization-based planning formulation with appropriate constraints and parameters using our mapping algorithm. We highlight the high-level interface below the dashed line. As the environment changes or new natural language instructions are given, our approach dynamically changes the specification of the constraints for the optimization-based motion planner and generates the new motion plans.

velocity in a short time window. Optimization-based motion planners [67, 68, 69] solve a constrained optimization problem to generate smooth and collision-free robot paths. We present an automatic scheme that generates the motion planning problem from NLP instructions.

There is some work on integrating optimization-based motion planning with NLP in 2D workspaces. Silver et al. [70] develop an algorithm for learning navigation cost functions from demonstrations. Howard et al. [21] use a probabilistic graphical model to generate motion planning constraints for a 2D navigation problem. Compared to these methods, our approach can handle 3D workspaces and high-dimensional configuration spaces to generate robot motions corresponding to complex NLP instructions. Other techniques focus on efficiency in human-robot collaborative tasks. Markov Decision Processes (MDP) are widely used to compute the best robot action policies [71, 10]. These techniques are complementary to our approach.

3.2 Dynamic Grounding Graphs

Fig. 3.1 shows the basic pipeline of our approach. When natural language commands are given as input, the NLP module (upper left) creates an optimization problem for a motion planning module (upper middle). The robot motion trajectory is then computed from the motion planning module (upper right). As the planned trajectory is executed (bottom right), the result is fed back to the NLP module. In this section, we present the algorithms used in the NLP module.

We extend the ideas of the Generalized Grounding Graphs (G^3) model and the Distributed Correspondence Graph (DCG) model [21] by including the latent variables in the grounding graph and using them to compute the constraints for motion planning. The input to our algorithm is the natural language instruction. We do not account for any errors due to voice recognition. From a natural language command input, we construct a factor graph, as shown in Fig. 3.2(a), which is based on the parsing of the command. For each node of the parse tree, we generate three types of nodes: word phrase node λ , grounding node γ , and correspondence node ϕ .

The input sentence Λ is parsed using the NLTK library [72]. The word phrase of each node in the parse tree is denoted as λ_i for $i = 1, 2, \dots$. Children of λ_i are $\lambda_{i1}, \dots, \lambda_{im}$. The root node of the parse tree is λ_1 . For example, in Fig. 3.2(a), the input sentence is “*Put the cup on the table.*” The parse tree has the root word phrase $\lambda_1 = \text{“Put”}$. Its noun $\lambda_2 = \text{“the cup”}$ and the preposition $\lambda_3 = \text{“on,”}$ which are the children nodes of the root node. The noun phrase $\lambda_4 = \text{“the table”}$ is the child node of λ_3 . Similarly, in Fig. 3.2(b), the command “*Don’t put it there*” is decomposed into 4 noun phrase nodes. The word phrase $\lambda_1 = \text{“Don’t”}$ is a negation of the verb and its child node is $\lambda_2 = \text{“put.”}$ $\lambda_3 = \text{“it”}$ and $\lambda_4 = \text{“there”}$ are the children nodes of λ_2 . Note that this parse tree is different from the parse tree in Fig. 3.2(a).

Our goal is to compute a mapping from a natural language sentence Λ to the cost function parameters H , given the robotic environment E where the robot is operating. E is a representation of the environment, which is composed of obstacle positions, orientations, and the robot’s configuration. Feature vectors are constructed in the factor graph from the description of the environment. H is a real-valued vector that contains all cost function parameters used in the optimization-based motion planner. It also includes the weights of different types of cost functions used in the optimization formulation. For example, the end-effector position cost function (Eq. (3.13)) requires the 3D coordinates of the target position as parameters. The repulsion cost function (Eq. (3.17)) requires

the repulsion source position and the constant from the exponential function.

We first compute the groundings γ_i of each word phrase λ_i . The grounding of each word phrase is the mapping from the word phrase to its meaning in the real world. Groundings can be objects, locations, motions, tasks, or constraints. In our model, the grounding γ_i depends on its word phrase λ_i and its children grounding nodes $\gamma_{i1}, \dots, \gamma_{im}$, where the tree structure of the grounding nodes follows the parse tree. Correspondence node ϕ_i indicates the correct matching between the word phrase λ_i and the grounding γ_i . It is a binary variable; ϕ_i is *true* if the word phrase and the grounding match and *false* if they do not.

3.2.1 Latent Parameters

A key novel component of our approach is the inclusion of latent variables in the grounding graph. Our primary goal is to compute the best cost function parameters H to be used directly for optimization-based motion planning. We denote $H \in \mathbb{R}^h$, a real vector of size h , as a collection of cost function parameters. In this case, the size h and the number of cost function parameters depend on the types of cost functions that are used.¹ From the predicted groundings γ_i , the cost function parameters in the motion planning formulation (Fig. 3.2(b)) are inferred through the latent variable H . H contains all the cost function parameters (e.g., weights of cost functions, locations, and orientations).

In Fig. 3.2(b), the resulting constraint-based motion planning problems are shown. We use the collision avoidance cost function as the default smoothness cost function and the target location cost function, though weights can vary. The target location, whose 3D coordinates are the cost function parameters, is set on the surface of the table. The cost function parameter node H contains the weights of the parameters and the 3D coordinates of the target location. In the bottom of Fig. 3.2(b), where a new “Don’t” command is given, a repulsion cost function is added. Thus, the cost function weight and the location of the repulsion source (below the robot’s end-effector position) are added to H .

¹In this paper, we set a maximum $h = 22$ to fully specify the smoothness, the end-effector position, the end-effector orientation, the end-effector speed, and the repulsion cost functions. It is a sum of three terms: 5 for weights, 16 for positions and orientations, and 1 for an exponential constant.

3.2.2 Probabilistic Model

We present a new probabilistic model to compute H , Λ , and E . We pose the problem of finding the best cost parameters as an optimization problem:

$$\underset{\mathbf{H}}{\text{maximize}} \quad p(H|\Lambda, E). \quad (3.1)$$

However, modeling the probability function without decomposing the variables and some assumptions about independence is difficult due to the high-dimensionality of H , Λ , and E and the dependencies between them. To simplify the problem, the natural language sentence is decomposed into n word phrases based on a parse tree, i.e.

$$p(H|\Lambda, E) = p(H|\lambda_1, \dots, \lambda_n, E). \quad (3.2)$$

Like G^3 , we introduce the intermediate groundings γ_i of word phrases λ_i and correspondence variables ϕ_i . The correspondence variables ϕ_i are binary random variables. The value 1 indicates that the word phrase λ_i correctly corresponds to the grounding γ_i . 0 means an incorrect correspondence.

We assume the conditional independence of the probabilities so that we can construct a factor graph (see Fig. 3.2(a)). With the independence assumptions, a single factor is connected to a word phrase node and its children grounding nodes, which contain information about the sub-components. These independence assumptions simplify the problem and make it solvable by efficiently taking advantage of the tree structure of the probabilistic graphical learning model. Formally, the root grounding node γ_1 contains all the information about a robot's motion. The factor that connects γ_1 and H implies that, from the root grounding node, the cost function parameters H are optimized without any consideration of other nodes. Other factors connect γ_i , ϕ_i , λ_i , children grounding nodes γ_{ij} and the environment E , where the parent-child relationship is based on a parse tree constructed from the natural language sentence. This graphical representation corresponds to the following equation:

$$p(H|\lambda_1, \dots, \lambda_n, E) = p(H|\gamma_1, E) \prod_i p(\gamma_i|\lambda_i, \phi_i, \gamma_{i1}, \dots, \gamma_{im}, E). \quad (3.3)$$

For the root factor connecting H , γ_1 and E , we formulate the continuous domain of H . We compute

the Gaussian Mixture Model (GMM) on the probability distribution $p(H|\gamma_1, E)$ and model our probability with non-root factors as follows:

$$\begin{aligned}
& p(\gamma_i|\lambda_i, \phi_i, \gamma_{i1}, \dots, \gamma_{im}, E) \\
&= \frac{1}{Z} \psi_i(\gamma_i, \lambda_i, \phi_i, \gamma_{i1}, \dots, \gamma_{im}, E) \\
&= \frac{1}{Z} \exp(-\theta_i^T f(\gamma_i, \lambda_i, \phi_i, \gamma_{i1}, \dots, \gamma_{im}, E)),
\end{aligned} \tag{3.4}$$

where Z is the normalization factor, and θ_i and f are the log-linearization of the feature function. The function f generates a feature vector given a grounding γ_i , a word phrase λ_i , a correspondence ϕ_i , children groundings γ_{ij} , and the environment E . The information from the robotic environment is used in the feature function f and in the log-linearized feature function f . The attributes of objects in the robotic world such as shapes and colors are encoded as multidimensional binary vectors, which indicate whether the object has a given attribute.

The probability distribution of the latent variable H is modeled with m pairs of Gaussian distribution parameters μ_i and σ_i with weights ω_i , as follows:

$$p(H|\gamma_1, E) \sim \sum_{i=1}^m \omega_i \mathcal{N}(\mu_i, \sigma_i^2). \tag{3.5}$$

Word phrases. The feature vector includes binary-valued vectors for the word and phrase occurrences, and Part of Speech (PoS) tags. There is a list of words that could be encountered in the training dataset such as $\{put, pick, cup, up, there, \dots\}$. If the word phrase contains the word “*put*,” then the occurrence vector at the first index is set to 1 and the others are set to 0. If the word phrase is “*pick up*,” then the occurrence vector values at the second, while the fourth is set to 1 and others are set to 0. This list also includes real-valued word similarities between the word and the pre-defined seed words. The seed words are the pre-defined words that the users expect to encounter in the natural language instructions. We used Glove word2vec [73] to measure cosine-similarity (i.e. the inner product of two vectors divided by the lengths of the vectors) between the words. The measurement indicates that the words are similar if the similarity metric value is near 1, that they have opposite meanings if the similarity metric is near -1, and that they have a weak relationship if it is near 0. This provides more flexibility to our model, especially when it encounters new words

that are not trained during the training phase.

Robot states. From the robot state, we collect the robot joint angles, the velocities, the end-effector position, the end-effector velocity, etc. This information can affect the cost function parameters even while processing the same natural language commands. For example, if the robot is too close to a human under the current configuration, then the cost function for end-effector speed C_{speed} or smoothness $C_{smoothness}$ will be adjusted so that the robot does not collide with the human. We also store information about the objects that are close to the robot. This information includes object type, position, orientation, shape, dimension, etc.

3.2.3 Factor Graph using Conditional Random Fields

We represent our dynamic grounding graph as a factor graph. We build a factor graph based on the probabilistic model described in Section 3.2.2 and use that for training and for inferring the meaning of given commands. In particular, we use Conditional Random Fields (CRF) [74] as a learning model for factor graphs because CRFs are a good fit for applying machine learning to our probabilistic graph model with conditional probabilities.

During the training step of CRF, we solve the optimization problem of maximizing the probability of the samples in the training dataset over the feature coefficients θ_i and the GMM parameters ω_i , μ_i and σ_i for every parse tree structure. By multiplying Eq. (3.4-3.5) for all training samples, the optimization problem becomes

$$\underset{\substack{\theta_1, \dots, \theta_n, \\ \omega_1, \dots, \omega_m, \\ \mu_1, \dots, \mu_m, \\ \sigma_1, \dots, \sigma_m}}{\text{maximize}} \prod_k p(H^{(k)} | \gamma_1^{(k)}, E^{(k)}) \quad (3.6)$$

$$\prod_i \frac{1}{Z^{(k)}} \exp(\theta_i^T f(\gamma_i^{(k)}, \lambda_i^{(k)}, \phi_i^{(k)}, \gamma_{i1}^{(k)}, \dots, \gamma_{im}^{(k)}, E^{(k)})), \quad (3.7)$$

where superscripts $(k) = 1 \dots D$ mean the indices of the training samples. The joint optimization problem Eq. (3.6-3.7) of the GMM and the CRF is a hard problem. So, we separate the problem into two and solve each one separately to maximize the objective. To solve Eq. (3.6), the training samples of continuous variable H is collected under the same conditional variable $\gamma_1^{(k)}$. Then, we solve the problem with the collection of H 's via Expectation Maximization (EM) method. Eq. (3.7) is a tree-structured CRF problem.

At the inference step, we used the trained CRF factor graph models to find the best groundings

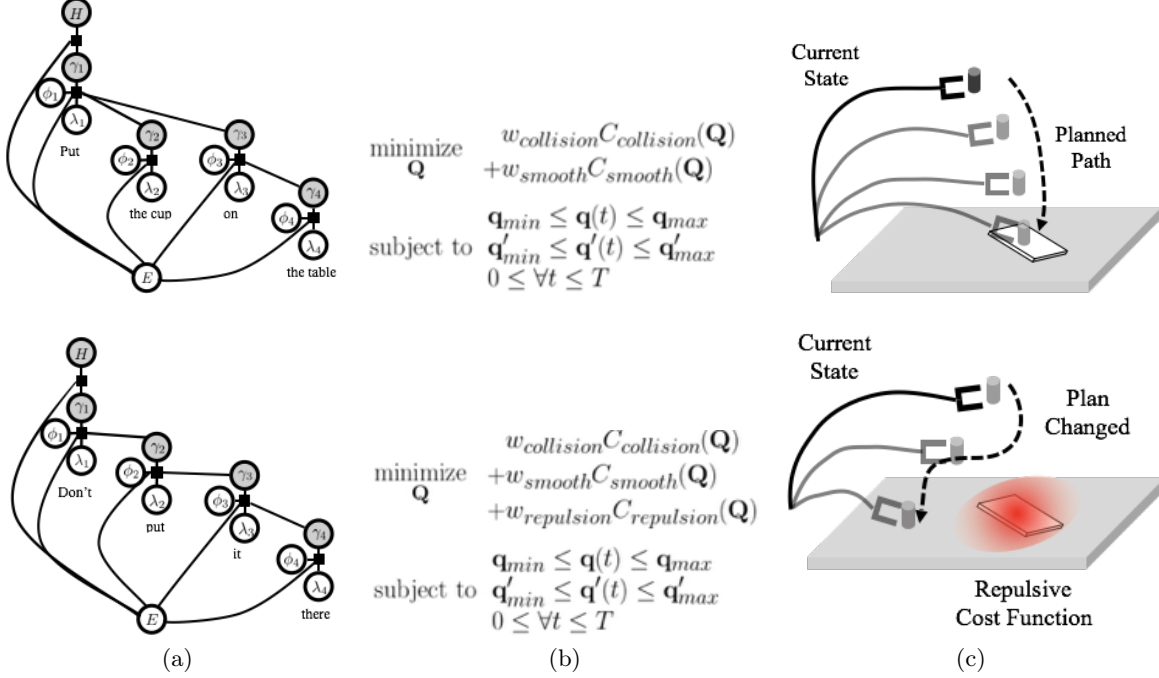


Figure 3.2: **Factor graphs for different commands:** In the environment in the right-hand column, there is a table with a thin rectangular object on it. A robot arm is moving a cup onto the table, but we want it to avoid moving over the book when given NLP instructions. (a) The command “*Put the cup on the table*” is given and the factor graph is constructed (left). Appropriate cost functions for the task are assigned to the motion planning algorithm (middle) and used to compute the robot motion (right). (b) As the robot gets close to the book, another command “*Don’t put it there*” is given with a new factor graph and cost functions.

Γ and the cost function parameters H by solving the CRF maximization problem

$$\underset{H, \gamma_1, \dots, \gamma_n}{\text{maximize}} p(H | \gamma_1, E) \prod_i \frac{1}{Z} \exp(\theta_i^T f(\gamma_i, \lambda_i, \phi_i, \gamma_{i1}, \dots, \gamma_{im}, E)). \quad (3.8)$$

When the nodes $H, \gamma_1, \dots, \gamma_n$ are optimized, they create a tree structure in the factor graph, meaning that we can solve the optimization problem efficiently using dynamic programming. Each factor depends on its parent and children varying variables and other fixed variables connected to it. This implies that we can solve the sub-problems in a bottom-up manner and combine the results to solve the bigger problem corresponding to the root node.

3.3 Dynamic Constraint Mapping With NLP Input

We use an optimization-based algorithm [12] to solve the cost minimization problem. The function and constraints of this cost minimization problem come from DGG, as explained in Sec. 3.2. In this section, we present our mapping algorithm, Dynamic Constraint Mapping, which maps

the word phrase groundings to proper cost function parameters corresponding to natural language instructions.

3.3.1 Robot Configurations and Motion Plans

We denote a single configuration of the robot as a vector \mathbf{q} , which consists of joint-angles or other degrees-of-freedom. A configuration at time t , where $t \in \mathbb{R}$, is denoted as $\mathbf{q}(t)$. We assume $\mathbf{q}(t)$ is twice differentiable, and its derivatives are denoted as $\mathbf{q}'(t)$ and $\mathbf{q}''(t)$. The n -dimensional space of configuration \mathbf{q} is the configuration space \mathcal{C} . We represent bounding boxes of each link of the robot as B_i . The bounding boxes at a configuration \mathbf{q} are denoted as $B_i(\mathbf{q})$.

For a planning task with a given start configuration \mathbf{q}_0 and derivative \mathbf{q}'_0 , the robot's trajectory is represented by a matrix \mathbf{Q} , whose elements correspond to the waypoints [75, 67, 68, 12]:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_0 & \mathbf{q}_1 & & \mathbf{q}_{n-1} & \mathbf{q}_n \\ \mathbf{q}'_0 & \mathbf{q}'_1 & \cdots & \mathbf{q}'_{n-1} & \mathbf{q}'_n \\ t_0 = 0 & t_1 & & t_{n-1} & t_n = T \end{bmatrix}. \quad (3.9)$$

The robot trajectory passes through $n + 1$ waypoints q_0, \dots, q_n , which will be optimized by an objective function under constraints in the motion planning formulation. Robot configuration at time t is interpolated from two waypoints. Formally, for j such that $t_j \leq t \leq t_{j+1}$, the configuration $\mathbf{q}(t)$ and derivative $\mathbf{q}'(t)$ are cubically interpolated using \mathbf{q}_j , \mathbf{q}'_j , \mathbf{q}_{j+1} , and \mathbf{q}'_{j+1} .

The i -th cost functions of the motion planner are $C_i(\mathbf{Q})$. Our motion planner solves an optimization problem with non-linear cost functions and linear joint limit constraints to generate robot trajectories for time interval $[0, T]$,

$$\begin{aligned} & \underset{\mathbf{Q}}{\text{minimize}} && \sum_i w_i C_i(\mathbf{Q}) \\ & \text{subject to} && \mathbf{q}_{min} \leq \mathbf{q}(t) \leq \mathbf{q}_{max}, \quad 0 \leq \forall t \leq T. \\ & && \mathbf{q}'_{min} \leq \mathbf{q}'(t) \leq \mathbf{q}'_{max} \end{aligned} \quad (3.10)$$

In the optimization formulation, C_i is the i -th cost function and w_i is the weight of the cost function.

3.3.2 Cost Functions

The overall optimization formulation is given in Eq. (3.10). To formulate the constraints, we use the following cost functions, which are designed to account for various attributes in the NLP

instructions. In our formulation, we use many types of cost functions such as collision avoidance, robot smoothness, robot end-effector speed, target positions, and target orientations. These cost functions are used to handle many attributes of natural language instructions. Each cost function has its weight and may also have other cost function parameters, if necessary. For example, the robot end-effector speed cost function has parameters corresponding to the direction and the magnitude of the speed, which impose a constraint on the final computed trajectory. If the weight of the end-effector speed cost function is higher than the others, then it contributes more to the overall objective function in the optimization formulation. If the weight is low, then the end-effector speed cost will be compromised and has a lesser impact on the path planner.

The cost functions C_i and the latent parameter H are closely related. H is a collection of parameters that describe all types of C_i and the weights w_i . The cost function parameters of C_i and the weights w_i are all real-valued. Those real values are appended to construct the real-valued vector H .

3.3.3 Parameterized Constraints

To handle various attributes, we use the following parameterized constraints in our optimization formulation.

Collision avoidance: By default, the robot should always avoid obstacles.

$$C_{collision}(\mathbf{Q}) = \int_0^T \sum_i \sum_j dist(B_i(t), O_j)^2 dt, \quad (3.11)$$

where $dist(B_i(t), O_j)$ is the penetration depth between a robot bounding box $B_i(t)$ and an obstacle O_j .

Smoothness: We penalize the magnitude of a robot’s joint angle speed to make the trajectory smooth. This corresponds to the integral of the first derivative of joint angles over the trajectory duration, as follows:

$$C_{smoothness}(\mathbf{Q}) = \int_0^T \sum_i \mathbf{q}'(t)_i^2 dt. \quad (3.12)$$

This function is useful when we need to control the speed of the robot. When the robot should operate at a low speed (e.g. when a human is too close), or we don’t want abrupt movements (e.g.,

for human safety), the smoothness cost can have high weights so that the robot moves slowly without jerky motions.

End-effector position: A user usually specifies the robot’s target position to make sure that the robot reaches its goal. This cost function penalizes the squared distance between the robot’s end-effector position and the target position over the trajectory duration as

$$C_{position}(\mathbf{Q}) = \int_0^T \|\mathbf{p}_{ee}(t) - \mathbf{p}_{target}\|^2 dt, \quad (3.13)$$

where $\mathbf{p}_{ee}(t)$ is the robot end-effector position at time t and \mathbf{p}_{target} is the target position. The target position \mathbf{p}_{target} is considered as a cost function parameter. In the mapping algorithm, a position grounding node encodes the target position parameter. This parameter can be a 3D position or the current object position in the environment. Typically, the target position is specified by an object name in the sentence, such as “*pick up the cup*” or “*move to the box*.” In these cases, the grounding nodes for “*the cup*” and “*the box*” are interpreted as the current 3D coordinates of the target positions, which are the parameters of this cost function.

End-effector orientation: Robotic manipulation tasks are sometimes constrained by the end-effector orientation. This cost function penalizes the squared angular differences between the end-effector orientation and the target orientation over the trajectory duration.

$$C_{orientation}(\mathbf{Q}) = \int_0^T \text{angledist}(\mathbf{q}_{ee}(t), \mathbf{q}_{target})^2 dt \quad (3.14)$$

$$C_{upvector}(\mathbf{Q}) = \int_0^T \text{angledist}(\mathbf{n}_{up}(t), \mathbf{n}_{target})^2 dt, \quad (3.15)$$

where $\mathbf{q}_{ee}(t)$ is the quaternion representation of the robot end-effector’s orientation at time t , \mathbf{q}_{target} is the end-effector orientation that we want the robot to maintain, \mathbf{n}_{up} is the normal up-vector of the robot’s end-effector, and \mathbf{n}_{target} is the target up-vector. As with the *end-effector position* cost, the target orientation \mathbf{q}_{target} is the cost function parameter. The target orientation usually depends on the object the robot picked up. For example, when the robot is doing a peg-hole insertion task under the command “*insert that into the hole*,” the orientation of the robot’s end-effector \mathbf{q}_{ee} should be constrained near the hole. If the robot arm is holding a cup of water, it should be upright so it does not spill the water. In this case, \mathbf{n}_{target} is set to $(0, 0, 1)$.

End-effector speed: This cost function penalizes the robot’s end-effector speed and direction:

$$C_{speed}(\mathbf{Q}) = \int_0^T \|\mathbf{v}_{ee}(t) - \mathbf{v}_{target}\|^2 dt, \quad (3.16)$$

where $\mathbf{v}_{ee}(t)$ is the robot’s end-effector speed at time t , and \mathbf{v}_{target} is the target speed. The parameters of this cost function correspond to \mathbf{v}_{target} . In some cases, we must restrict the robot’s end-effector velocity, e.g., if a user wants to pick up a cup filled with water and doesn’t want to spill it. Spilling can be prevented by limiting the end-effector speed, making the robot move more slowly.

Repulsion: The repulsion functions are commonly represented as potential fields

$$C_{repulsion}(\mathbf{Q}) = \int_0^T \exp(-c\|\mathbf{p}_{ee}(t) - \mathbf{p}_r\|) dt, \quad (3.17)$$

where \mathbf{p}_r is the position to which we don’t want the robot to move. The coefficient $c > 0$ suggests how much the cost is affected by $\|\mathbf{p}_{ee}(t) - \mathbf{p}_{repulsive}\|$, the distance between the end-effector position and the repulsion source. The cost function is maximized when the end-effector position is exactly at the repulsion source, and it decreases as the distance between the end-effector and the repulsion position increases. For example, if the command is “*Don’t put the cup on the laptop,*” we can define a repulsion cost with the laptop position as the repulsion source. The cost function is inversely proportional to the distance between the end-effector and the laptop.

3.4 Implementation and Results

We have implemented our algorithm and evaluated its performance in a simulated environment and on a 7-DOF Fetch robot. All the timings are generated on a multi-core PC with Intel i7-4790 8-core 3.60GHz CPU and a 16GB RAM. We use multiple cores for fast evaluation and parallel trajectory search to compute a good solution to the constrained optimization problem [12].

3.4.1 Training DGGs for Demonstrations

We describe how the training dataset for our DGG model was generated. The training dataset for DGGs requires three components: a natural language sentence, a robotic environment, and the cost function parameters for optimization-based motion planners.

For each demonstration, we write tens of different sentences that specify the take goals the constraints for the motion plans with different nouns, pronouns, adjectives, verbs, adverbs, preposition,

etc. For each sentence, we generate a random robotic environment and an initial state for the robot. In addition, the robot joint values and joint velocities are randomly set as initial states. We collect tens or hundreds of random robotic environments. For a natural language sentence, a random robotic environment, and a random initial state for the robot, the cost function parameters are assigned manually or synthesized from other examples. Crowdsourcing such as Amazon Mechanical Turk can be alternatively used to assign cost function parameters. Hundreds of multiple data samples are generated from generated data samples by switching the correspondence variable in the DGG model from 1 (true) to 0 (false) and changing the grounding variables to the wrong ones to match the false correspondence variable. The training dataset is created with up to 100,000 samples in our experiments. When the cost function parameters are determined, the optimization-based motion planner is used to compute a feasible robot trajectory. In the optimization-based motion planning algorithm, there are some waypoints through which the robot trajectory should pass. For the robot’s safety, we check if the robot trajectory with the given cost function parameters is in-collision and appropriately set a higher value of the coefficient of the collision cost and compute a new trajectory. This process is repeated until the trajectory is collision-free. The training step took up to an hour with 100,000 training samples in our experimental settings, though the training time can vary depending on the complexity of tasks, environments, and natural language instructions.

We use different training data for each scenario. For the scenarios shown in Fig. 3.3, the initial pose of the robot in front of the table and the positions of the blue and red objects on the table are randomly set. For “*Pick up*” commands, appropriate cost function parameters are computed so that the robot picks up a blue or red object depending on the given command. Similarly, in Fig. 3.4, the position and orientation of the laptop is initialized randomly. Given the “*Put*” command, we create an end-effector position cost function so that the robot places the object on the table; and a repulsive cost function to avoid the laptop position.

3.4.2 Simulations and Real Robot Demonstrations

We evaluate the performance on optimization problems that occur in complex environments composed of multiple objects. Based on the NLP commands, the robot decides to pick an appropriate object or is steered towards the goal position in a complex scene. In particular, the user gives NLP commands such as “*move right*,” “*move up*,” “*move left*,” or “*move down*” to guide the robot. For each such command, we compute the appropriate cost functions.

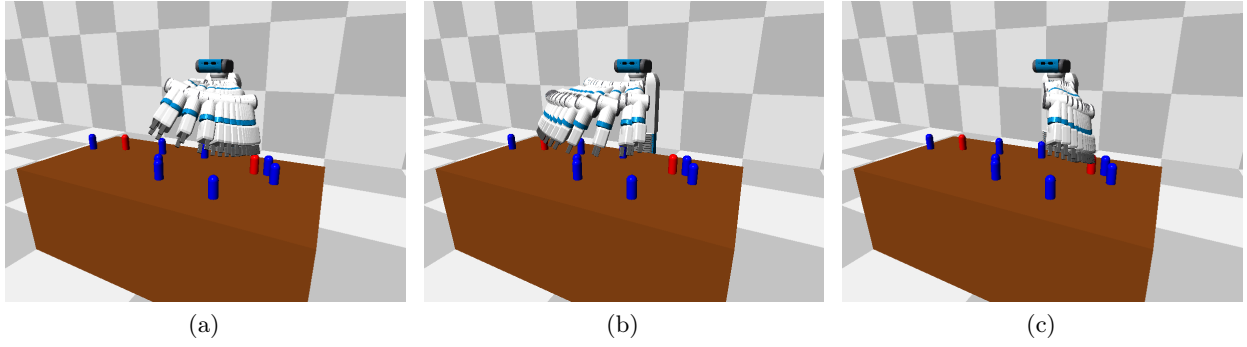


Figure 3.3: The simulated Fetch robot arm reaches towards one of the two red objects. (a) When a command “*pick up one of the red objects*” is issued, the robot moves to the red object on the right because of the DGG algorithm. (b) If the user doesn’t want the robot to pick up the object on the right, he/she uses a command “*don’t pick up that one.*” Our DGG algorithm dynamically changes the cost function parameters. (c) The robot approaches the object on the right and stops.

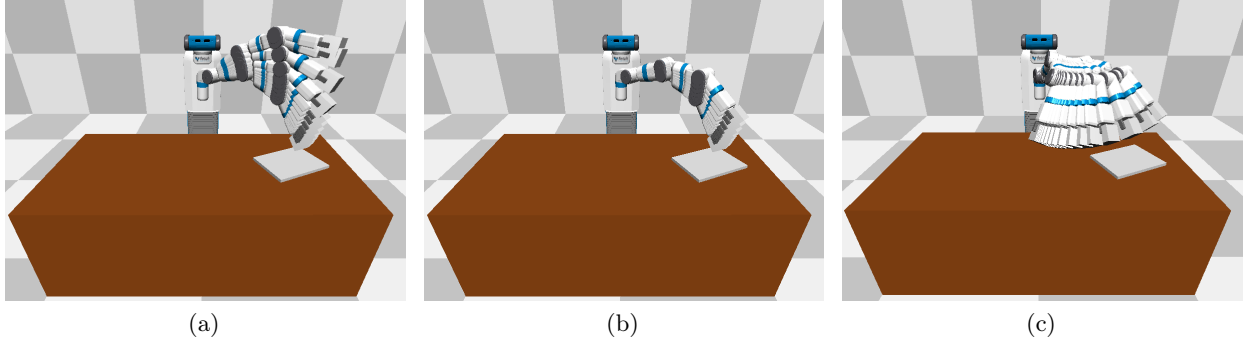


Figure 3.4: In this simulated environment, the human instructs the robot to “*put the cube on the table*” (a). As it approaches the laptop (b), the human uses a negation NLP command “*don’t put it there,*” so the robot places it at a different location (c).

We also integrate our NLP-based planner with ROS and evaluated its performance on the 7-DOF Fetch robot. In a real-world setting, we test its performance on different tasks corresponding to: (1) moving a soda can on the table from one position to another; (2) not moving the soda can over the book. With a noisy point cloud sensor on the robot, the thin book is not recognized as a separate obstacle by the robot, though the human user wants the robot to avoid it. All the instructions used in these tasks have different attributes, which makes it hard for prior methods. In Fig. 3.6, the two sub-tasks are specified in one sentence at the beginning, as “*move the can on the table, but don’t put it on the book*”. The cost function is used to move the robot’s end-effector to the surface of the table. Another cost function penalizes the distance between the book and the end-effector. In Fig. 3.7, only the first sub-task is given at the beginning. This results in the robot moving the can on the

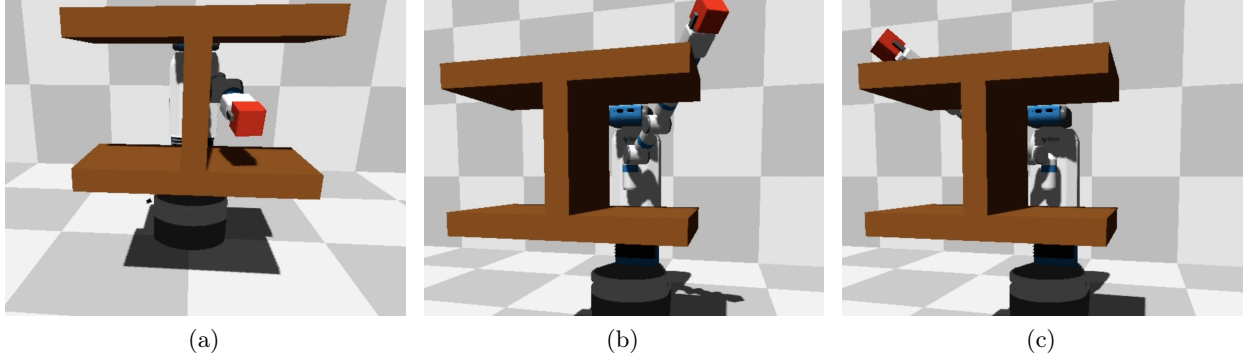


Figure 3.5: A 7-DOF Fetch robot is operating in a simulated environment avoiding an obstacle. (a) In a traditional optimization-based motion planner, the planner gets stuck at a local minimum. (b),(c) Using natural language commands as guidance, the user guides the robot out of the minimum and towards the goal position.

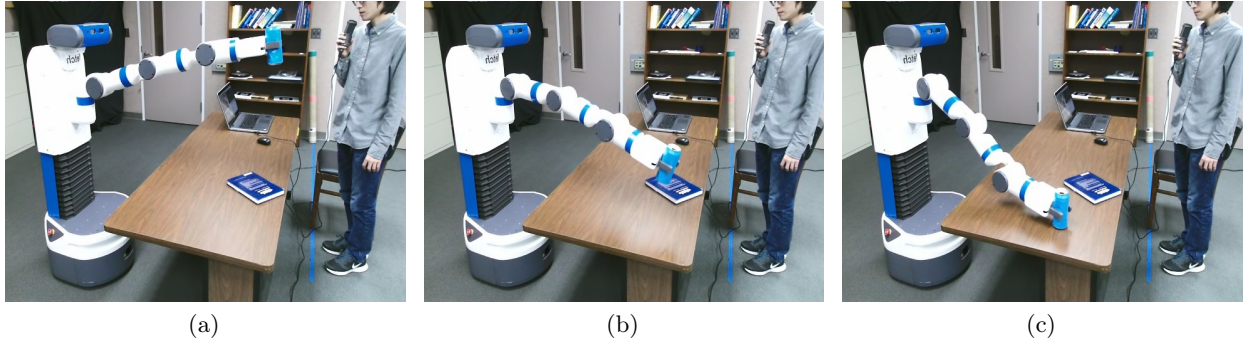


Figure 3.6: The Fetch robot is taking real-time commands from the human and moves the soda can on the table.

book. As the robot gets too close to the book, the person says “stop,” then says “don’t put it there.” The robot recomputes the cost functions and avoids the region around the book.

3.4.3 Analysis

We evaluate the performance based on the following metrics:

Success Rate: The ratio of successful task completion among all trials. Failure includes colliding with the obstacles due to an incorrect mapping of cost function parameters, violating constraints specified by natural language commands, and not completing the task due to some other reason.

Trajectory Duration: The time between the giving of the first NLP command and the robot’s successful completion of the task after trajectory computation. A shorter duration implies a higher performance.

Trajectory Smoothness Cost: A cost based on evaluating the trajectory smoothness according

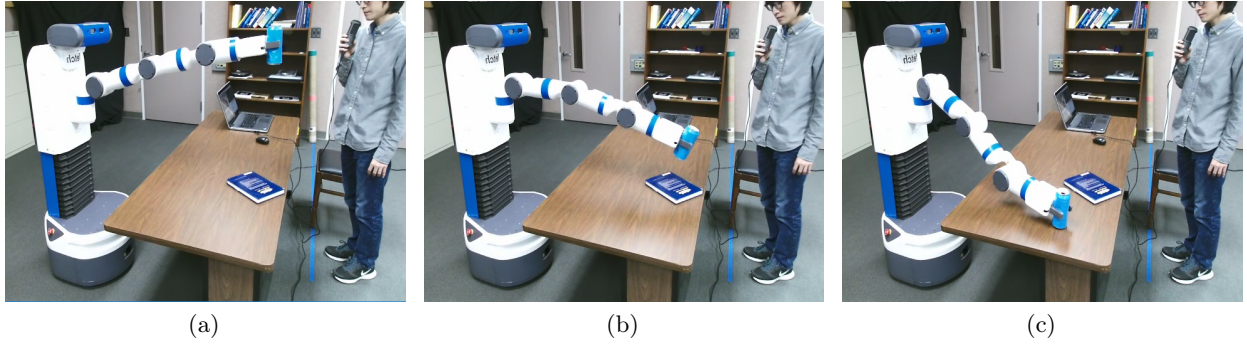


Figure 3.7: The Fetch robot is moving a soda can on a table based on NLP instructions. Initially, the user gives the “pick and place” command. However, when the robot gets closer to the book, the person says “*don’t put it there*” (i.e. negation) and the robot uses our dynamic constraint mapping functions and optimization-based planning to avoid the book. Our approach can generate appropriate motion plans for such attributes.

# Training Data	Success Rate	Duration	Smoothness Cost
1,000	5/10	23.46s (5.86s)	8.72 (5.56)
3,000	9/10	16.02s (3.28s)	2.56 (0.64)
10,000	10/10	13.16s (1.24s)	1.21 (0.32)
30,000	10/10	12.81s (0.99s)	0.78 (0.12)
100,000	10/10	12.57s (0.97s)	0.72 (0.10)

Table 3.1: Planning performances with varying sizes of training data for the scenario in Fig. 3.3 with 21 different NLP instructions.

to standard metrics and dividing it by the trajectory duration. A lower cost implies a smoother and more stable robot trajectory.

Table 3.1 shows the results on our benchmarks with varying numbers of training data samples in the simulation environment shown in Fig. 3.3. When the number of training data samples increases, the success rate also increases while the trajectory duration and the trajectory smoothness cost decrease. Table 3.2 shows the running time of our algorithm and the distances from the obstacle on the table in the real-world scenarios. We use 8 parallel threads for parallel trajectory search in the motion planning module.

Table 3.3, 3.4 and 3.5 show the examples of the dataset. In Table 3.3, DGGs with word phrase nodes, grounding nodes and correspondence variables are shown. The correspondence variables of the graphs on the left column are all *true*, and the groundings are matched correctly. Whereas, the correspondence variables on the right column are mixed with *true* and *false*. The groundings are not matched if the correspondence variable is *false*. Many data samples are generated by flipping the

Scenarios	Instructions	$ H $	DGG Time	Planning Time
Pick up an object (Fig. 4)	10	12	32ms	93ms
Don't put on the laptop (Fig. 5)	20	13	16ms	98ms
Move around obstacle	45	9	16ms	95ms
Static Instructions (video)	20	18	73ms	482ms
Dynamic Instructions (Fig. 1)	21	18	58ms	427ms

Table 3.2: Running time (ms) of our DGG and motion planning modules for each scenario.

correspondence variables between *true* and *false* to increase the accuracy of the DGG inference step. In Table 3.4 and Table 3.4, the latent variables are shown for the examples of the grounding graphs and the environment. The cost function weights and other necessary cost function parameters are manually set in the data generation program.

3.5 Benefits and Comparisons

Most prior methods that combine NLP and motion planning have focused on understanding natural language instructions to compute robot motion for simple environments and constraints. Most of these methods are limited to navigation applications [64, 63, 58] or simple settings [59], or they are not evaluated on real robots [62]. Nyga et al. [76, 77, 78, 79] use probabilistic relation models based on knowledge bases to understand natural language commands that describe visual attributes of objects. This is complementary to our work. Broad et al. [80] extended DCG for a robot manipulator so that it will handle natural language correction for robot motion in realtime. In our approach, the goal is to generate appropriate high-DOF motion trajectories in response to attribute-based natural language instructions like negation, distance or orientation constraints, etc. Unlike prior methods, the output of our NLP parsing algorithm is directly coupled with the specification of the motion planning problem as a constrained optimization method.

It may be possible to extend prior methods [20, 21] to handle attribute-based NLP instructions. For example, distance attributes require a number of constraints in the motion planning formulation. In natural language instructions such as *“Pick up the blue block and put it 20 cm to the left of the red block”* or *“Pick up one of the two blocks on the rightmost, and place it 10 inches away from the block on the leftmost,”* the exact distance specifications are the distance attributes. Prior methods that use G^3 , DCG, or the Hybrid G^3 -DCG models have only been evaluated with a small number of attributes (distance, orientation, and contact) to solve constrained motion planning problems.

Pick up one of the blue objects.							
Grounding Graph		DGG Nodes		Grounding Graph		DGG Nodes	
		λ_1	"Pick up"			λ_1	"Pick up"
		γ_1	Command(pick up)			γ_1	Object ₅
		ϕ_1	true			ϕ_1	false
		λ_2	"one"			λ_2	"one"
		γ_2	Object ₁			γ_2	Object ₅
		ϕ_2	true			ϕ_2	true
		λ_3	"of"			λ_3	"of"
		γ_3	Select(nearest)			γ_3	Select(nearest)
		ϕ_3	true			ϕ_3	true
		λ_4	"blue"			λ_4	"blue"
		γ_4	Color(blue)			γ_4	Color(red)
		ϕ_4	true			ϕ_4	false
λ_5	"the objects"	λ_5	"the objects"				
γ_5	{Object ₁ , ..., Object ₅ }	γ_5	{Object ₁ , ..., Object ₅ }				
ϕ_5	true	ϕ_5	true				

Place it on the table.							
		λ_1	"Place"			λ_1	"Don't"
		γ_1	Command(place)			γ_1	Negation
		ϕ_1	true			ϕ_1	false
		λ_2	"it"			λ_2	"it"
		γ_2	Object ₁			γ_2	Object ₂
		ϕ_2	true			ϕ_2	false
		λ_3	"on"			λ_3	"on"
		γ_3	Location(on)			γ_3	Location(on)
		ϕ_3	true			ϕ_3	true
		λ_4	"the table"			λ_4	"the table"
		γ_4	Object ₂			γ_4	Object ₁
		ϕ_4	true			ϕ_4	false

Don't put it there.							
		λ_1	"Place"			λ_1	"Place"
		γ_1	Negation			γ_1	Command(place)
		ϕ_1	true			ϕ_1	false
		λ_2	"put"			λ_2	"put"
		γ_2	Command(place)			γ_2	Negation
		ϕ_2	true			ϕ_2	false
		λ_3	"it"			λ_3	"it"
		γ_3	Object ₁			γ_3	Object ₁
		ϕ_3	true			ϕ_3	true
		λ_4	"there"			λ_4	"there"
		γ_4	Location(robot)			γ_4	Object ₂
		ϕ_4	true			ϕ_4	false

Table 3.3: Examples of DGGs with different configurations of correspondence variables.

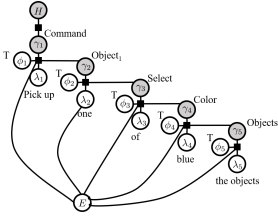
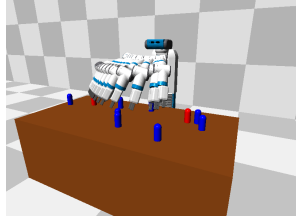
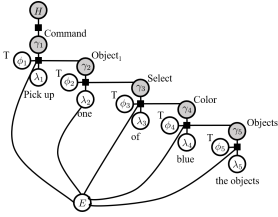
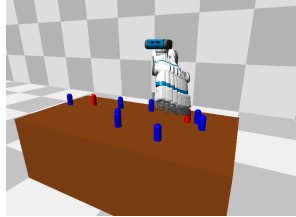
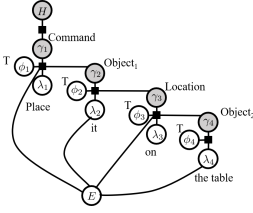

Pick up one of the blue objects			
<div>Grounding Graph</div> 	DGG Nodes		<div>Trajectory</div> 
	γ_1	Command(pick up)	
	γ_2	Object ₁	
	γ_3	Select(nearest)	
	γ_4	Color(blue)	
	γ_5	{Object ₁ , ..., Object ₅ }	
	Latent Variables		
	Collision avoidance	3.00	
	Smoothness	1.00	
	End-effector position	10.0	
		(0.75, 0.2, 0.81)	
	End-effector orientation	30.00	
		(0.00, 0.00, 1.00)	
	End-effector speed	0.00	
Repulsion	0.00		
<div>Grounding Graph</div> 	DGG Nodes		<div>Trajectory</div> 
	γ_1	Command(pick up)	
	γ_2	Object ₁	
	γ_3	Select(nearest)	
	γ_4	Color(blue)	
	γ_5	{Object ₁ , ..., Object ₅ }	
	Latent Variables		
	Collision avoidance	1.00	
	Smoothness	3.00	
	End-effector position	10.0	
		(-0.43, 0.26, 0.81)	
	End-effector orientation	30.00	
		(0.00, 0.00, 1.00)	
	End-effector speed	0.00	
Repulsion	0.00		

Table 3.4: Examples of DGGs and the latent variables.

Place it on the table.		
Grounding Graph	DGG Nodes	
	γ_1	Command(place)
	γ_2	Object ₁
	γ_3	Location(on)
	γ_4	Object ₂
	Latent Variables	
	Collision avoidance	1.00
	Smoothness	3.00
	End-effector position	10.0 (0.00, 0.30, 0.70)
	End-effector orientation	100 (0.00, 0.00, 1.00)
	End-effector speed	0.00
	Repulsion	0.00
Trajectory		
		

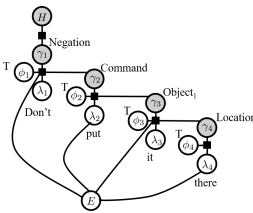

Don't put it there.		
Grounding Graph	DGG Nodes	
	γ_1	Negation
	γ_2	Command(place)
	γ_3	Object ₁
	γ_4	Location(robot)
	Latent Variables	
	Collision avoidance	1.00
	Smoothness	3.00
	End-effector position	10.0 (0.00, 0.30, 0.70)
	End-effector orientation	100 (0.00, 0.00, 1.00)
	End-effector speed	0.00
	Repulsion	3.00 10.00, (0.05, 0.32, 0.70)
Trajectory		
		

Table 3.5: Examples of DGGs and the latent variables.

These prior techniques use discretized constraints [21], each of which can be active (i.e. $f(x) > 0$), inverted ($f(x) < 0$), or ignored (i.e. not included). Therefore, it is not possible to represent an explicit constraint corresponding to the value of the continuous variable *distance* in their formulation.

3.6 Limitations, Conclusions, and Future Work

We present a motion planning algorithm that computes appropriate motion trajectories for a robot based on complex NLP instructions. Our formulation is based on two novel concepts: dynamic grounding graphs and dynamic constraint mapping. We highlight the performance in simulated and real-world scenes with a 7-DOF manipulator operating next to humans. We use a high dimensional optimization algorithm and the solver may get stuck in local minima, though we use multiple initializations to solve this problem. Furthermore, the accuracy of the mapping algorithm varies as a function of the training data.

As future work, we would like to overcome these limitations and evaluate the approach in challenging scenarios with moving obstacles while performing complex robot tasks. More work is needed to handle the full diversity of a natural language, especially for rare words, complicated grammar styles, and hidden intentions or emotions in human speech. We plan to incorporate stronger natural language processing and machine learning methods such as those based on semantic parsing, neural sequence-to-sequence models, etc. We also plan to collect more natural language data from a variety of sources such as recipes or demonstration videos.

CHAPTER 4

Human Motion Prediction for Safe Robot Motion Planning

Motion planning algorithms are used to compute collision-free paths for robots among obstacles. As robots are increasingly used in workspace with moving or unknown obstacles, it is important to develop reliable planning algorithms that can handle environmental uncertainty and the dynamic motions. In particular, we address the problem of planning safe and reliable motions for a robot that is working in environments with humans. As the humans move, it is important for the robots to predict the human actions and motions from sensor data and to compute appropriate trajectories.

In order to compute reliable motion trajectories in such shared environments, it is important to gather the state of the humans as well as predict their motions. There is considerable work on online tracking and prediction of human motion in computer vision and related areas [31]. However, the current state of the art in gathering motion data results in many challenges. First of all, there are errors in the data due to the sensors (e.g., point cloud sensors) or poor sampling [32]. Secondly, human motion can be sudden or abrupt and this can result in various uncertainties in terms of accurate representation of the environment. One way to overcome some of these problems is to use predictive or estimation techniques for human motion or actions, such as using various filters like Kalman filters or particle filters [33]. Most of these prediction algorithms use a motion model that can predict future motion based on the prior positions of human body parts or joints, and corrects the error between the estimates and actual measurements. In practice, these approaches only work well when there is sufficient information about prior motion that can be accurately modeled by the underlying motion model. In some scenarios, it is possible to infer high-level human intent using additional information, and thereby perform a better prediction of future human motions [81, 82]. These techniques are used to predict the pedestrian trajectories based on environmental information in 2D domains.

4.1 Related Work

In this section, we give a brief overview of prior work on human motion prediction, task planning for human-robot collaborations, and motion planning in environments shared with humans.

4.1.1 Intention-aware Motion Planning and Prediction

Intention-Aware Motion Planning (IAMP) denotes a motion planning framework where the uncertainty of human intention is taken into account [81]. The goal position and the trajectory of moving pedestrians can be considered as human intention and used so that a moving robot can avoid pedestrians [83].

In terms of robot navigation among obstacles and pedestrians, accurate predictions of humans or other robot positions are possible based on crowd motion models [84, 85] or integration of motion models with online learning techniques [86] for 2D scenarios and they are orthogonal to our approach.

Predicting the human actions or the high-DOF human motions has several challenges. Estimated human poses from recorded videos or realtime sensor data tend to be inaccurate or imperfect due to occlusions or limited sensor ranges [32]. Furthermore, the whole-body motions and their complex dynamics with many high-DOF makes it difficult to represent them with accurate motion models [87]. There has been a considerable literature on recognizing human actions [88]. Machine learning-based algorithms using Gaussian Process Latent Variable Models (GP-LVM) [89, 90] or Recurrent neural network (RNN) [30] have been proposed to compute accurate human dynamics models. Recent approaches use the learning of human intentions along with additional information, such as temporal relations between the actions [71, 91] or object affordances [5] to improve the accuracy. Inverse Reinforcement Learning (IRL) has been used to predict 2D motions [92, 93] or 3D human motions [94].

4.1.2 Robot Task Planning for Human-Robot Collaboration

In human-robot collaborative scenarios, robot task planning algorithms have been developed for the efficient distribution of subtasks. One of their main goal is reducing the completion time of the overall task by interleaving subtasks of robot with subtasks of humans with well designed task plans. In order to compute the best action policy for a robot, Markov Decision Processes (MDP) have been widely used [95]. [71] use MDP models based on mental model convergence of human and robots. [10] use Q-learning to train MDP models where the graph model has the transitions corresponding to the human action and robot action pairs. [96] used a Bayesian learning algorithm on hand motion

prediction and tested the algorithm in a human-robot collaboration tasks. Our MDP models extend these approaches, but also take into account the issue of avoiding collisions between the human and the robot.

4.1.3 Motion Planning in Environments Shared with Humans

Prior work on motion planning in the context of human-robot interaction has focused on computing robot motions that satisfy cognitive constraints such as social acceptability [97] or being legible to humans [98].

In human-robot collaboration scenarios where both the human and the robot perform manipulation tasks in a shared environment, it is important to compute robot motions that avoid collisions with the humans for safety reasons. Dynamic window approach [66] (which searches the optimal velocity in a short time interval) and online replanning [99, 12, 69] (which interleaves planning with execution) are widely used approaches for planning in such dynamic environments. As there are uncertainties in the prediction model and in the sensors for human motion, the future pose is typically represented as the *Belief* state, which corresponds to the probability distribution over all possible human states. [100] explicitly construct an occupied workspace voxel map from the predicted Belief states of humans in the shared environment and avoid collisions.

Partially Observable Markov Decision Process (POMDP) techniques are widely used for motion planning with uncertainty in the robot state and in the environment. These approaches estimate the robot environment states, represent them in a probabilistic manner, and tend to compute the best action or the best robot trajectory considering likely possibilities. Because the search space of the exact POMDP formulation is too large, many practical and approximate POMDP solutions have been proposed [101, 102] to reduce the running time and obtain almost realtime performance. [49] use an approximate and realtime POMDP motion planner on autonomous driving carts. Our algorithm solves the problem in two steps: first, the future human motion trajectory is predicted; second, our planning algorithm generates a collision-free trajectory by considering the predicted trajectory. Our current formulation does not fully account for the uncertainty in the robot state and can be combined with POMDP approaches to handle this issue.

4.2 Notation and Assumptions

In this section, we first introduce the notation and terminology used in the paper and give an overview of our motion planning algorithms.

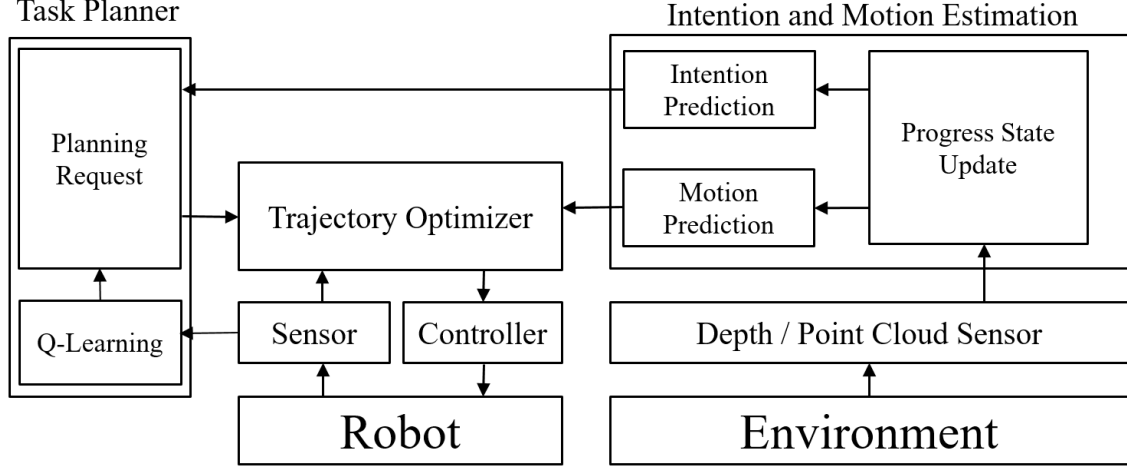


Figure 4.1: **Overview of our Intention-Aware Planner:** Our approach consists of three main components: task planner, trajectory optimization, and intention and motion estimation.

In the context of this paper, we use the term human ‘intention’ as a predetermined simple action to interact with the robotic environment. This is in the user’s mind before the action is taken. The intention is a combination of what to do (e.g. reaching his/her hand to grab an object) and how to achieve it (e.g. moving his/her arm from an idle pose to the cup while avoiding the robot). More formally in the context of learning algorithms, the former is the action class in a discrete domain, the latter is a set of possible motions in a continuous domain, and ‘intention’ is a combination of both components.

I-Planner (Intention-Aware Motion Planner) is our robot motion planning algorithm that predicts human intentions and plans robot motions with the information of the predicted human intentions, given a whole task to be completed by the human and the robot working cooperatively. The overview of I-Planner is shown in Figure 4.1. A task consists of multiple subtasks that the human or the robot can accomplish. The goal of I-Planner is to complete the given task efficiently, while guaranteeing safety for the human by avoiding any collisions. We specifically focus on three parts: predicting human intention in a short time window, determining a subtask for the robot to be achieved next, and planning a robot’s motion trajectory that avoids the human for his/her safety. The predicted human action class (the first component of ‘intention’) is used in determining a subtask for the robot. The predicted human motion trajectory (the second component of ‘intention’) is used in planning the robot trajectory. An example of tasks, human action and human motion prediction is shown in Figure 4.2.

As we need to learn about human actions and short-term motions, a large training dataset of human motions is needed. We collect N demonstrations of how human joints typically move while performing some tasks and in which order subtasks are performed. Each demonstration is represented using $T^{(i)}$ time frames of human joint motion, where the superscript (i) represents the demonstration index. The motion training dataset is represented as following:

- ξ is a matrix of tracked human joint motions. $\xi^{(i)}$ has $T^{(i)}$ columns, where a column vector represents the different human joint positions during each time frame.
- F is a feature vector matrix. $F^{(i)}$ has $T^{(i)}$ columns and is computed from $\xi^{(i)}$.
- \mathbf{a}^h is a human action (or subtask) sequence vector that represents the action labels over different time frames. For each time frame, the action is categorized into one of the m^h discrete action labels, where the action label set is $A^h = \{a_1^h, \dots, a_{m^h}^h\}$.
- $L = \{(\xi^{(i)}, F^{(i)}, \mathbf{a}^{h,(i)})\}_{i=1}^N$ is the motion database used for training. It consists of human joint motions, feature descriptors and the action labels at each time frame.

During runtime, MDP-based human action inference is used in our task planner. The MDP model is defined as a tuple (P, A^r, T) :

- $A^r = \{a_1^r, \dots, a_{m^r}^r\}$ is a robot action (or subtask) set of m^r discrete action labels.
- $P = \mathcal{P}(A^r \cup A^h)$, a power set of union of A^r and A^h , is the set of states in MDP. We refer to the state $\mathbf{p} \in P$ as a *progress state* because each state represents which human and robot actions have been performed so far. We assume that the sequence of future actions for completing the entire task depends on the list of actions completed. \mathbf{p} has $m^h + m^r$ binary elements, which represent corresponding human or robot actions have been completed ($\mathbf{p}_j = 1$) or not ($\mathbf{p}_j = 0$). For cases where same actions can be done more than once, the binary values can be replaced with integers, to count the number of actions performed.
- $T : P \times A^r \rightarrow \Pi(P)$ is a transition function. When a robot performs an action a^r in a state \mathbf{p} , $T(\mathbf{p}, a^r)$ is a probability distribution over the progress state set P . The probability of being state \mathbf{p}' after taking an action a^r from state \mathbf{p} is denoted as $T(\mathbf{p}, a^r, \mathbf{p}')$.

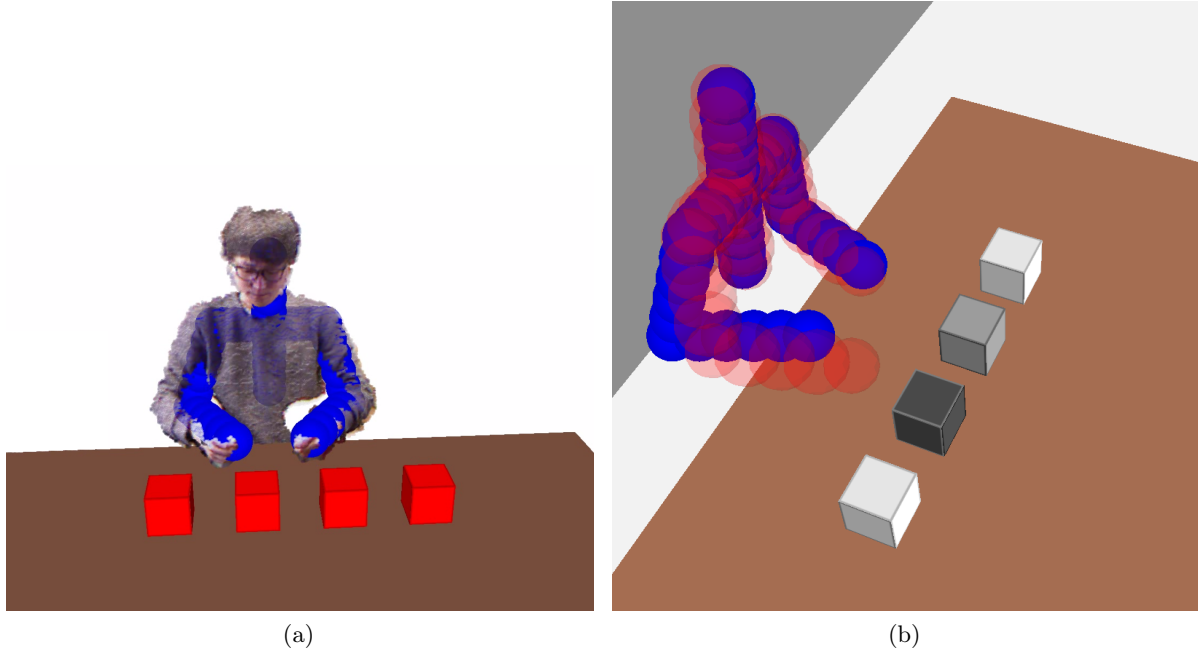


Figure 4.2: **Motion uncertainty and prediction:** (a) A point cloud and the tracked human (blue spheres). The joint positions are used as feature vectors. (b) Prediction of next human action and future human motion, where 4 locations are colored according to their probability of next human action from white (0%) to black (100%). Prediction of future motion after 1 second (red spheres) from current motion (blue spheres) is shown as performing the action: *move right hand to the second position* which has the highest probability associated with it.

- $\pi : P \rightarrow A^r$ is the action policy of a robot. $\pi(\mathbf{p})$ denotes the best robot action that can be taken at state \mathbf{p} , which results in maximal performance.

We use Q-learning [103] to determine the best action policy during a given state, which rewards the values that are induced from the result of the execution.

For the robot representation, we denote a single configuration of the robot as a vector \mathbf{q} that consists of joint-angles. The n -dimensional space of configuration \mathbf{q} is the configuration space \mathcal{C} . We represent each link of the robot as R_i . The finite set of bounding spheres for link R_i is $\{B_{i1}, B_{i2}, \dots\}$, and is used as a bounding volume of the link, i.e., $R_i \subset \cup_j B_{ij}$. The links and bounding spheres at a configuration \mathbf{q} are denoted as $R_i(\mathbf{q})$ and $B_{ij}(\mathbf{q})$, respectively. In our benchmarks, where the robot arms are used, these bounding spheres are automatically generated using the medial axis of robot links. We also generate the bounding spheres $\{C_1, C_2, \dots\}$ for humans and other obstacles.

For a planning task with start and goal configurations \mathbf{q}_s and \mathbf{q}_g , the robot's trajectory is

represented by a matrix \mathbf{Q} ,

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_s & \mathbf{q}_1 & \cdots & \mathbf{q}_{n-1} & \mathbf{q}_g \\ t_0 & t_1 & \cdots & t_{n-1} & t_n \end{bmatrix},$$

where robot trajectory passes through the $n + 1$ waypoints. We denote the i -th waypoint of \mathbf{Q} as $\mathbf{x}_i = \begin{bmatrix} \mathbf{q}_i^T & t_i \end{bmatrix}$.

4.3 Human Action Prediction

In this section, we describe our human action prediction algorithm, which consists of offline learning and online inference of actions.

4.3.1 Learning of Human Actions and Temporal Coherence

We collect N demonstrations to form a motion database L . The 3D joint positions are tracked using OpenNI library [104], and their coordinates are concatenated to form a column vector $\xi^{(i)}$. For full-body motion prediction, we used 21 joints, each of which has 3D coordinates tracked by OpenNI. So, $\xi^{(i)}$ is a 63-dimensional vector. For upper-body motion prediction, there are 10 joints and thus $\xi^{(i)}$ is length 30. Then, feature vector $F^{(i)}$ is derived from $\xi^{(i)}$. It has joint velocities and joint accelerations, as well as joint positions.

To learn the temporal coherence between the actions, we deal with only the human action sequences $\{\mathbf{a}^{h,(i)}\}_{i=1}^N$. Based on the progress state representation, for any time frame s , the prefix sequence of $\mathbf{a}^{h,(i)}$ of length s yields a progress state $\mathbf{p}_s^{(i)}$ and the current action $c_s^{h,(i)} = \mathbf{a}_s^{h,(i)}$. The next action label $n_s^{h,(i)}$ performed after frame s can also be computed, at which the action label differs at the first time while searching in the increasing order from frame $s + 1$. Then, for all possible pairs of demonstrations and frame index (i, s) , the tuples $(\mathbf{p}_s^{(i)}, c_s^{h,(i)}, n_s^{h,(i)})$ are collected to compute histograms $h(n^h; \mathbf{p}, c^h)$, which counts the next action labels at each pair (\mathbf{p}, c^h) that have appeared at least once. We use the normalized histograms to estimate the next future action for the given \mathbf{p} and c^h . i.e.,

$$p(n^h = a_j^h | \mathbf{p}, c^h) = \frac{h(a_j^h; \mathbf{p}, c^h)}{\sum_{k=1}^m h(a_k^h; \mathbf{p}, c^h)}. \quad (4.1)$$

In the worst case, there are at most $O(2^m)$ progress states since there are m binary values per action. However, in practice, only $O(N \cdot m)$ progress states are generated. This is because the number of unique progress states is less than m , and the subtask order dependency may allow only

a few possible topological orders.

In order to train the human motion, the motion sequence $\xi^{(i)}$ and the feature sequence $F^{(i)}$ are learned, as well as the action sequences $\mathbf{a}^{(i)}$. Because we are interested in short-term human motion prediction for collision avoidance, we train the learning module from multiple short periods of motion. Let n_p be the number of previous consecutive frames and n_f be the number of future consecutive frames to look up. n_f and n_p are decided so that the length of motion is short-term motion (about 1 second) that will be used for short-term future collision detection in the robot motion planner. At the time frame s , where $n_p \leq s \leq T^{(i)} - n_f$, the columns of feature matrix $F^{(i)}$ from column index $s - n_p + 1$ to s are denoted as $F_{prev,s}^{(i)}$. Similarly, the columns of motion matrix from index $s + 1$ to $s + n_f$ are denoted as $\xi_{next,s}^{(i)}$.

Tuples $(F_{prev,s}^{(i)}, \mathbf{p}_s^{(i)}, c_s^{h(i)}, \xi_{next,s}^{(i)})$ for all possible pairs of (i, s) are collected as the training input. They are partitioned into groups having the same progress state \mathbf{p} . For each progress state \mathbf{p} and current action c^h , the set of short-term motions are regressed using SPGP with the DTW kernel function [105], considering $\{F_{prev}\}$ as input and $\{\xi_{next}\}$ as multiple channeled outputs. We use SPGPs, a variant of Gaussian Processes, because it significantly reduces the running time for training and inference by choosing M pseudo-inputs from a large number of an original human motion inputs. The final learned probability distribution is

$$\begin{aligned} p(\xi_{next}|F_{prev}, \mathbf{p}, c^h) &= \prod_{c:\text{channels}} p(\xi_{next,c}|F_{prev}, \mathbf{p}, c^h), \\ p(\xi_{next,c}|F_{prev}, \mathbf{p}, c^h) &\sim \mathcal{GP}(m_c, K_c), \end{aligned} \quad (4.2)$$

where $\mathcal{GP}(\cdot, \cdot)$ represents trained SPGPs, c is an output channel (i.e., an element of matrix ξ_{next}), and m_c and K_c are the learned mean and covariance functions of the output channel c , respectively.

The current action label $c_s^{h(i)}$ should be learned to estimate the current action. We train $c_s^{h(i)}$ using Tuples $(F_{prev,s}^{(i)}, \mathbf{p}_s^{(i)}, c_s^{h(i)})$. For each state \mathbf{p} , we use Import Vector Machine (IVM) classifiers to compute the probability distribution:

$$p(c^h = a_j^h | F_{prev}, \mathbf{p}) = \frac{\exp(f_j(F_{prev}))}{\sum_{a_k^h} \exp(f_k(F_{prev}))}, \quad (4.3)$$

where $f_j(\cdot)$ is the learned predictive function [106] of IVM.

4.3.2 Runtime Human Intention and Motion Inference

Based on the learned human actions, at runtime we infer the next most likely short-term human motion and human subtask for the purpose of collision avoidance and task planning, respectively. The short-term future motion prediction is used for the collision avoidance during the motion planning. The probability of future motion is given as:

$$p(\xi_{next}|F, \mathbf{p}) = \sum_{c^h \in A^h} p(\xi_{next}, c^h|F, \mathbf{p}).$$

By applying the Bayes theorem, we get

$$p(\xi_{next}|F, \mathbf{p}) = \sum_{c^h \in A^h} p(c^h|F, \mathbf{p})p(\xi_{next}|F, \mathbf{p}, c^h). \quad (4.4)$$

The first term $p(c^h|F, \mathbf{p})$ is inferred through the IVM classifier in (4.3). To infer the second term, we use the probability distribution in (4.2) for each output channel.

We use Q-learning for training the best robot action policy π in our MDP-based task planner. We first define the function $Q : P \times A^r \rightarrow \mathbb{R}$, which is iteratively trained with the motion planning executions. Q is updated as

$$\begin{aligned} Q_{t+1}(\mathbf{p}_t, a_t^r) = & (1 - \alpha_t)Q_t(\mathbf{p}_t, a_t^r) \\ & + \alpha_t(R_{t+1} + \gamma \max_{a^r} Q_t(\mathbf{p}_{t+1}, a^r)), \end{aligned}$$

where the subscripts t and $t + 1$ are the iteration indexes, R_{t+1} is the reward function after taking action a_t^r at state \mathbf{p}_t , and α_t is the learning rate, where we set $\alpha_t = 0.1$ in our experiments. A reward value R_{t+1} is determined by several factors:

- Preparation for next human action: the reward gets higher when the robot runs an action before a human action which can be benefited by the robot's action. We define this reward as $R_{prep}(\mathbf{p}_t, a_t^r)$. Because the next human subtask depends on the uncertain human decision, we predict the likelihood of the next subtask from the learned actions in (4.1) and use it for the

reward computation. The reward value is given as

$$R_{prep}(\mathbf{p}_t, a_t^r) = \sum_{a^h \in A^h} p(n^h = a^h | \mathbf{p}_t) H(a^h, a_t^r), \quad (4.5)$$

where $H(a^h, a^r)$ is a prior knowledge of reward, representing the amount of how much the robot helped the human by performing the robot action a^r before the human action a^h . If the robot action a^r has no relationship with a^h , the H value is zero. If the robot helped, the H value is positive, otherwise negative.

- Execution delay: There may be a delay in the robot motion’s execution due to the collision avoidance with the human. To avoid collisions, the robot may deviate around the human and make it without delay. In this case the reward function is not affected, i.e. $R_{delay,t} = 0$. However, there are cases that the robot must wait until the human moves to another pose because the human can block the robot’s path, which causes delay d . We penalize the amount of delay to the reward function, i.e. $R_{delay,t} = -d$. Note that the delay can vary during each iteration due to the human motion uncertainty.

The total reward value is a weighted sum of both factors:

$$R_{t+1} = w_{prep} R_{prep}(\mathbf{p}_t, a_t^r) + w_{delay} R_{delay,t}(\mathbf{p}_t, a_t^r),$$

where w_{prep} and w_{delay} are weights for scaling the two factors. The preparation reward value is predefined for each action pairs. The delay reward is measured during runtime.

4.3.3 Human Motion Prediction with Noisy Input

In most scenarios, our human motion prediction algorithm deals with the noisy data. As a result, it is important to analyze the performance of our approach in relation to these limitations. To analyze the robustness of our human motion prediction algorithm, we account for input noise in our Gaussian Process model.

Equation 4.2 is the Gaussian Process Regression for human motion prediction where the input variable is F_{prev} and the output variables are represented as $\xi_{next,c}$. Following the standard notation of the Gaussian Process, we use the symbol \mathbf{x} as a D -dimensional input vector instead of F_{prev} , y as an output variable instead of $\xi_{next,c}$, and $y = f(\mathbf{x}) + \epsilon_y$ instead of $p(\xi_{next,c} | F_{prev}) \mathcal{GP}(m_c, K_c)$. We

add an input noise term ϵ_x to the standard GP model,

$$\begin{aligned} y &= \tilde{y} + \epsilon_y, \epsilon_y \sim \mathcal{N}(0, \sigma_y^2), \\ \mathbf{x} &= \tilde{\mathbf{x}} + \epsilon_x, \epsilon_x \sim \mathcal{N}(\mathbf{0}, \Sigma_x), \end{aligned}$$

where we assume that the input noise is Gaussian and that the D -dimensional input vector is independently noised, i.e. Σ_x is diagonal. With the input noise term, the output becomes

$$y = f(\tilde{\mathbf{x}} + \epsilon_x) + \epsilon_y,$$

and the first term Taylor expansion on the function f yields

$$y = f(\tilde{\mathbf{x}}) + \epsilon_x^T \partial_f(\mathbf{x}) + \epsilon_y,$$

where $\partial_f(\mathbf{x})$ is the D -dimensional derivative of f with respect to \mathbf{x} . We have N training data items, represented as $(\mathbf{x}_i, y_i)_{i=1}^N$. \mathbf{y} is an N -dimensional vector $\{y_1, \dots, y_N\}^T$. If we follow the derivation of the Gaussian Process (GP) with the additional error term presented in [107], GP with noisy input becomes

$$\begin{aligned} m_c(\mathbf{x}_*) &= \mathbf{k}_* (K + \sigma_y^2 I + \text{diag}(\Delta_f \Sigma_x \Delta_f^T))^{-1} \mathbf{y}, \\ K_c(\mathbf{x}_*) &= k_{**} - \mathbf{k}_* (K + \sigma_y^2 I + \text{diag}(\Delta_f \Sigma_x \Delta_f^T))^{-1} \mathbf{k}_*, \end{aligned}$$

where \mathbf{x}_* is the input of the mean function m_c and of the variance function K_c ; k_{**} is the kernel function value on \mathbf{x}_* , i.e. $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$; K is a matrix of kernel function values on all pairs of input points, i.e. $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$; and Δ_f is a matrix whose i -th row is the derivative of f at \mathbf{x}_i . $\text{diag}(\cdot)$ results in a diagonal matrix, leaving the diagonal elements and reducing the non-diagonals to zero. Compared to the standard GP, the additional term is the diagonal matrix $\text{diag}(\Delta_f \Sigma_x \Delta_f^T)$, which acts as the output noise term $\sigma_y I$.

In our human motion prediction algorithm, we use the human joint positions as input and output, so the input and the output have the same amount of noise. In other words, $\Sigma_x = \sigma_x I$ and $\sigma_x = \sigma_y$. Instead of differentiating between input and output noise, we use $\sigma = \sigma_x = \sigma_y$. Also, the

derivative ∂_f is proportional to the joint velocities because f is proportional to the joint position values. Therefore, the elements of the diagonal matrix can be expressed as

$$(\text{diag}(\Delta_f \Sigma_x \Delta_f))_{ii} = \sigma_x \|\partial_f(\mathbf{x}_i)\|^2 = \sigma \|\mathbf{v}_i\|^2,$$

where \mathbf{v}_i is the joint velocity. Because the joint velocity is a variable for every training input data, instead of taking joint velocities, we set the joint velocity limits \mathbf{v} , satisfying $\|\mathbf{v}_i\|^2 \leq \|\mathbf{v}\|^2$. As a result, the Gaussian Process regression for human motion prediction can be given as:

$$\begin{aligned} m_c(\mathbf{x}_*) &= \mathbf{k}_* (K + \sigma^2 (1 + \|\mathbf{v}\|^2) I)^{-1} \mathbf{y}, \\ K_c(\mathbf{x}_*) &= k_{**} - \mathbf{k}_* (K + \sigma_y^2 (1 + \|\mathbf{v}\|^2) I)^{-1} \mathbf{k}_*. \end{aligned}$$

If we keep the joint velocity small, the Gaussian Process with input noise behaves robustly, like how it behaves without input noise. The mean function is over-smoothed and the variance function becomes higher if the joint velocity limit is set high.

4.4 I-Planner: Intention-aware Motion Planning

The main goals of our motion planner are: (1) planning high-level tasks for a robot by anticipating the most likely next human action and (2) computing a robot trajectory that reduces the probability of collision between the robot and the human or other obstacles by using motion prediction.

At the high-level task planning step we use MDP, which is used to compute the best action policies for each state. The state of an MDP graph denotes the progress of the whole task. The best action policies are determined through reinforcement learning with Q-learning. Then, the best action policies are updated within the same state. The probability of choosing an action increases or decreases according to the reward function. Our reward computation function is affected by the prediction of intent and the delay caused by predictive collision avoidance.

We also estimate the short-term future motion from learned information to avoid future collisions. From the joint position information, motion features are extracted based on human poses and surrounding objects related to human-robot interaction tasks, such as human joint positions, positions of a hand relative to other objects, etc. The motions are classified over the human action set A^h . To classify the motions, we use an Import Vector Machine (IVM) [106] for classification and a Dynamic

Time Warping (DTW) [105] kernel function for incorporating the temporal information. Given the human motions database of each action type, we train future motions using Sparse Pseudo-input Gaussian Process (SPGP) [108]. Combining these two prediction results, the final future motion is computed as the weighted sum over different action types weighed by the probability of each action type that could be performed next. For example, if the action classification results in probability are 0.9 for action *Move forward* and 0.1 for action *Move backward*, the future motion prediction (the results of SPGP) for *Move forward* dominates. If the action classification results in a probability of 0.5 for both actions, the predicted future motions for each action class will be used in avoiding collisions, but with weights of 0.5. However, in this case, the current motion does not have specific features to distinguish the action, meaning that the future motion in the short term will be similar and there will be an overlapped region in the 3D space that works as the future motion of weight 1.

After deciding which robot task will be performed, a robot motion trajectory that tends to avoid collisions with humans is then computed. An optimization-based motion planner [12] is used to compute a locally optimal solution that minimizes the objective function subject to many types of constraints such as robot related constraints (e.g., kinematic constraint), human motion related constraints (e.g., collision free constraint), etc. Because future human motion is uncertain, we can only estimate the probability distribution of the possible future motions. Therefore, we perform probabilistic collision checking to reduce the collision probability in future motions. We also continuously track the human pose and update the predicted future motion to re-plan safe robot motions. Our approach uses the notion of online probabilistic collision detection [17, 109, 110] between the robot and the point-cloud data corresponding to human obstacles to compute reactive costs and integrate them with our optimization-based planner.

Our motion planner is based on an optimization formulation, where $n + 1$ waypoints in the space-time domain \mathbf{Q} define a robot motion trajectory to be optimized. Specifically, we use an optimization-based planner, ITOMP [12], that repeatedly refines the trajectory while interleaving the execution and motion planning for dynamic scenes. We handle three types of constraints: smoothness constraint, static obstacle collision-avoidance, and dynamic obstacle collision avoidance. To deal with the uncertainty of future human motion, we use probabilistic collision detection between the robot and the predicted future human pose.

Let s be the current waypoint index, meaning that the motion trajectory is executed in the time

interval $[t_0, t_s]$, and let m be the replanning time step. A cost function for collisions between the human and the robot can be given as:

$$\sum_{i=s+m}^{s+2m} p \left(\bigcup_{j,k} B_{jk}(\mathbf{q}_i) \cap C_{dyn}(t_i) \neq \emptyset \right) \quad (4.6)$$

where $C_{dyn}(t)$ are the workspace volumes occupied by dynamic human obstacles at time t . The trajectory being optimized during the time interval $[t_s, t_{s+m}]$ is executed during the next time interval $[t_{s+m}, t_{s+2m}]$. Therefore, the future human poses are considered only in the time interval $[t_{s+m}, t_{s+2m}]$.

The collision probability between the robot and the dynamic obstacle at time frame i in (4.6) can be computed as a maximum between bounding spheres:

$$\max_{j,k,l} p(B_{jk}(\mathbf{q}_i) \cap C_l(t_i) \neq \emptyset), \quad (4.7)$$

where $C_l(t_i)$ denotes bounding spheres for a human body at time t_i whose centers are located at line segments between human joints. The future human poses ξ_{future} are predicted in (4.4) and the bounding sphere locations $C_l(t_i)$ are derived from it. Note that the probabilistic distribution of each element in ξ_{future} is a linear combination of current action proposal $p(c^h|F, \mathbf{p})$ and Gaussians $p(\xi_{future}|F, \mathbf{p}, c^h)$ over all c^h , i.e., (4.7) can be reformulated as

$$\max_{j,k,l} \sum_{c^h} p(c^h|F, \mathbf{p}) p(B_{jk}(\mathbf{q}_i) \cap C_l(t_i) \neq \emptyset).$$

Let \mathbf{z}_l^1 and \mathbf{z}_l^2 be the probability distribution functions of two adjacent human joints obtained from $\xi_{future}(t_i)$, where the center of $C_l(t_i)$ is located between them by a linear interpolation $C_l(t_i) = (1-u)\mathbf{z}_l^1 + u\mathbf{z}_l^2$ where $0 \leq u \leq 1$. The joint positions follows Gaussian probability distributions:

$$\begin{aligned} \mathbf{z}_l^i &\sim \mathcal{N}(\mu_l^i, \Sigma_l^i) \\ \mathbf{c}_l(t_i) &\sim \mathcal{N}((1-u)\mu_l^1 + u\mu_l^2, (1-u)^2\Sigma_l^1 + u^2\Sigma_l^2) \end{aligned} \quad (4.8)$$

$$= \mathcal{N}(\mu_l, \Sigma_l), \quad (4.9)$$

where $\mathbf{c}_l(t_i)$ is the center of $C_l(t_i)$. Thus, the collision probability between two bounding spheres is bounded by

$$\int_{\mathbb{R}^3} I(\|\mathbf{x} - \mathbf{b}_{jk}(\mathbf{q}_i)\| \leq (r_1 + r_2)) f(\mathbf{x}) d\mathbf{x}, \quad (4.10)$$

where $\mathbf{b}_{jk}(\mathbf{q}_i)$ is the center of bounding sphere $B_{jk}(\mathbf{q}_i)$, r_1 and r_2 are the radius of $B_{jk}(\mathbf{q}_i)$ and $C_l(t_i)$, respectively, $I(\cdot)$ is an indicator function, and $f(\mathbf{x})$ is the probability distribution function. The indicator function restricts the integral domain to a solid sphere, and $f(\mathbf{x})$ is the probability density function of $\mathbf{c}_l(t_i)$, in (4.9). There is no closed form solution for (4.10), therefore we use the maximum possible value to approximate the probability. We compute \mathbf{x}_{max} at which $f(\mathbf{x})$ is maximized in the sphere domain and multiply it by the volume of sphere, i.e.

$$p(B_{jk}(\mathbf{q}_i) \cap C_l(t_i) \neq \emptyset) \leq \frac{4}{3}\pi(r_1 + r_2)^3 f(\mathbf{x}_{max}). \quad (4.11)$$

Since even \mathbf{x}_{max} does not have a closed form solution, we use the bisection method to find λ with

$$\mathbf{x}_{max} = (\Sigma^{-1} + \lambda I)^{-1}(\Sigma^{-1}\mathbf{p}_{lm} + \lambda \mathbf{o}_{jk}(\mathbf{q}_i)),$$

which is on the surface of sphere, explained in Generalized Tikhonov regularization [111] in detail.

The collision probability, computed in (4.10), is always positive due to the uncertainty of the future human position, and we compute a trajectory that is guaranteed to be nearly collision-free with sufficiently low collision probability. For a user-specified confidence level δ_{CL} , we compute a trajectory that its probability of collision is upper-bounded by $(1 - \delta_{CL})$. If it is unable to compute a collision-free trajectory, a new waypoint \mathbf{q}_{new} is appended next to the last column of \mathbf{Q} to make the robot wait at the last collision-free pose until it finds a collision-free trajectory. This approach computes a guaranteed collision-free trajectory, but leads to delay, which is fed to the Q-learning algorithm for the MDP task planner. The higher the delay that the collision-free trajectory of a task has, the less likely the task planner selects the task again.

4.4.1 Upper Bound of Collision Probability

Using the predicted distribution and user-specified threshold δ_{CL} , we can compute an upper bound using the following lemma.

Lemma 4.4.1. *The collision probability is less than $(1 - \delta_{CL})$ if $\frac{4}{3}\pi(r_1 + r_2)^2 f(\mathbf{x}_{max}) < 1 - \delta_{CL}$.*

Proof. This bound follows Equations (4.10) and (4.11).

$$\begin{aligned} p(B_{jk}(\mathbf{q}_i) \cap C_l(t_i) \neq \emptyset) &\leq \frac{4}{3}\pi(r_1 + r_2)^2 f(\mathbf{x}_{max}) \\ &< 1 - \delta_{CL}. \end{aligned}$$

□

We use this bound in our optimization algorithm for collision avoidance.

4.4.2 Safe Trajectory Optimization

Our goal is to compute a robot trajectory that will either not collide with a human or that will reduce the probability of collision below a certain threshold. Sometimes, there is no feasible collision-free trajectory for the robot. However, if there is any trajectory for which the collision probability is less than a threshold, we want to be able to compute it. Our optimization-based planner also generates multiple initial trajectories and finds the best solution in parallel. In this manner, it expands the search space and reduces the probability of the robot being stuck in a local minima configuration. This can be expressed as the following theorem:

Theorem 4.4.2. *An optimization-based planner with n parallel threads will compute a global solution trajectory with a collision probability less than $(1 - \delta_{CL})$, with the probability $1 - (1 - \frac{|A(\delta_{CL})|}{|S|})^n$, if it exists, where S is the entire search space. $A(\delta_{CL})$ corresponds to the neighborhood around the optimal solutions with a collision probability being less than $(1 - \delta_{CL})$, where the local optimization converges to one of the global optima. $|\cdot|$ is the measure of the search or configuration space.*

We give a brief overview of the proof. In this case, $\frac{|A(\delta_{CL})|}{|S|}$ measures the probability that a random sample lies in the neighborhood of the global optima. In general, $|A(\delta_{CL})|$ will become smaller as the environment becomes more complex and has more local minima. Overall, this theorem provides a lower bound on the probability for our intention-aware planner with n threads. If the limit for n increases, the planner will compute the optimal solution, if it exists. This can be stated as:

Corollary 4.4.3 (Probabilistic Completeness). *Our intention-aware motion planning algorithm with n trajectories becomes probabilistically complete as n increases.*

Proof. When the number of threads increases, we have a higher chance of computing the global optimal trajectory and, in the same manner, the increasing number of threads improves the probability that the planner computes an acceptable solution, according to the following:

$$\lim_{n \rightarrow \infty} 1 - \left(1 - \frac{|A(\delta_{CL})|}{|S|}\right)^n = 1.$$

□

Theorem 4.4.2 and Corollary 4.4.2 both implicate that the use of more threads with different initial random function seed values increases the probability of finding a valid solution, where validity is a trajectory with the upper bound of collision probability at less than a certain safety value, as shown in Lemma 4.4.1. Since I-Planner uses an optimization-based formulation and all optimization threads run independently, the parallel algorithm significantly increases the probability of finding a reliable trajectory with a small collision probability while not taking additional running time. From the implications of Lemma 4.4.1, Theorem 4.4.2, and Corollary 4.4.2, our I-Planner algorithm with multiple optimization threads can efficiently find a valid and safe robot trajectory with a collision probability less than a certain safety value.

4.5 Implementation and Performance

We highlight the performance of our algorithm in a situation where the robot is performing a collaborative task with a human and computing safe trajectories. We use a 7-DOF KUKA-IIWA robot arm. The human motion is captured by a Kinect sensor operating with a 15Hz frame rate, and only upper body joints are tracked for collision checking. We use the ROS software [112] for robot control and sensor data communication. The motion planner has a 0.5s re-planning timestep, The number of pseudo-inputs M of SPGPs is set to 100 so that the prediction computation is performed online. Videos of the following experiments with a real robot are available at: <http://gamma.cs.unc.edu/SafeMP>.

4.5.1 Performance of Human Motion Prediction

We have tested our human motion prediction model on labeled motion datasets corresponding to a human’s reaching action. Our human motion prediction model allows human joint position errors, as discussed in the previous section. To demonstrate the robustness of our model against input noise errors, we measure the classification accuracy and regression accuracy, varying the sensor

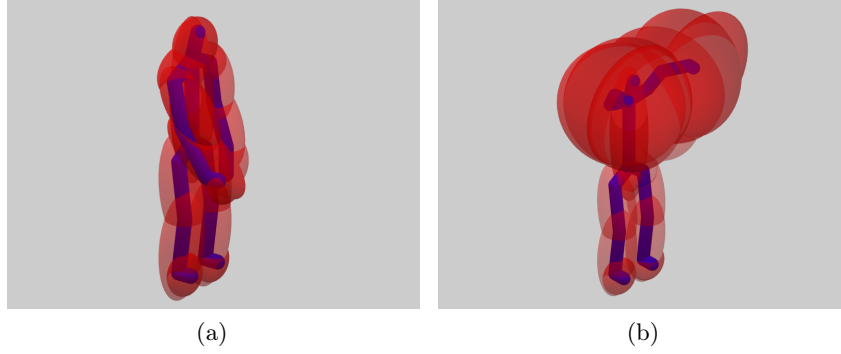


Figure 4.3: **Human motion prediction results:** The result of human motion prediction is represented by Gaussian distributions for each skeleton joint. The ellipsoid boundaries within which the integral of Gaussian distribution probability is 95% are drawn in red and the human skeleton is shown in blue. Bounding ellipsoids have transparency values that are proportional to the action classifier probability. (a) Undistinguished human action class: This occurs when the classifier fails to distinguish the human action class, generating nearly uniform probability distribution among the action classes. (b) Prediction results when untrained human motion is given: These cases result in larger boundary spheres around the human skeleton. This is because, in the Gaussian Process, the output has a uniform mean and a high variance when the input point is outside the range of the training input data.

error parameter and the maximum human joint velocity limits.

In the human reaching action motion datasets, the human is initially in a static pose in front of a table. Then, his or her left or right arm moves and reaches towards one of the target locations on the table. Then the arm returns to the initial pose. The dataset contains 8 different target positions on the table and 30 reaching motions for each target position. Because the result of our human motion prediction model depends on human joint velocities, we synthesize fast and slow motions by changing the speed of captured motions.

We measure the correctness of human motion classification and human motion regression in the following ways:

- Motion classification precision: Because human motion is continuous and the transition between motions can be difficult to measure, we only count the motion frames in which the multi-class classifier yields the highest probability greater than 50%.
- Motion regression precision: At a certain time, human motion trajectory for the next T seconds is predicted. We compute the regression precision as the integral of distance between the



Figure 4.4: **Performance of human motion prediction:** The precision/accuracy of classification and regression algorithms are shown in the graphs, with varying input noise. (a) Classification precision versus input noise: We only take the input points where the probability of an action classification is dominant, i.e. probability $> 50\%$. (b) Regression precision versus input noise: Measured by the integral of distance between the predicted human motion and the ground-truth human motion trajectory. (c) Regression accuracy versus input noise: The integral of the volume of Gaussian distribution ellipsoids.

predicted and ground-truth motions over the T -second window as:

$$\int_0^T \sum_{i:joint} \|\mathbf{p}_{pred,i}(t) - \mathbf{p}_{true,i}(t)\|^2 dt,$$

where $\mathbf{p}_{pred}(t)$ is the collection of resulting mean values of the Gaussian Process with noisy input for joint i .

- Motion regression accuracy: As with the precision, accuracy is the integral of the volume of ellipsoids generated by the Gaussian distributions:

$$\int_0^T \sum_{i:joint} \det(\mathbf{K}_{i,pred}(t)) dt,$$

where $\mathbf{K}_{i,pred}(t)$ is the collection of resulting variance of the Gaussian Process with noisy input for joint i . In this case, a lower value results in a better prediction result.

Figure 4.3 shows the human motion prediction results. The Gaussian Process regression algorithm corresponds to the use of Gaussian distribution ellipsoids around the predicted mean values of the joint positions. In (a), when the human is in an idle position, the motion classifier results in near uniform distribution among the action classes and the motion regression algorithm generates human motion trajectories that progress slightly towards the target positions. In (b), when the human arm gets close to the target position, the motion classifier predicts the motion class with a dominant

probability and the motion regression algorithm infers a correct future motion trajectory that is more accurate than (a). In (c), when an untrained human motion is given, the motion classifier and the motion regression algorithm result in uniform values and high variances, respectively, generating a conservative collision bound around the human. This is a normal behavior in terms of classification and regression because the given input point is outside the range of training data input points.

Figure 4.4 shows the classification precision, regression precision, and accuracy with varying input noises. When the input noise is high, the accuracy of human motion classification and regression can decrease.

4.5.2 Comparison with Prior Work

We compared our method against Anticipatory Temporal Conditional Random Field (ATCRF) [5], a prior method for human activity classification and future motion regression. The ATCRF algorithm creates a learning model based on Conditional Random Field, learning human intentions from a dataset with annotated human action classes, human motion trajectories, and object affordances (the functionality of the object). For this comparison, we used the CAD-120 dataset [113], which contains 120 RGB-D videos of 4 different subjects and 10 activity classes. We also train their algorithm on our dataset, which contains 100 RGB-D videos with 5 activity classes performed by one subject.

By using additional information about properties of objects close to humans, ATCRF can predict the action classes and generate future human motion trajectories. However, compared to our human motion prediction method, ATCRF cannot handle noisy input or predict the variance around the predicted motion trajectories. Thus, we measure the motion classification and regression precision to compare the performance of our method with that of ATCRF. For a fair comparison, the input noise parameter in our human motion prediction is set to zero.

Table 4.1 shows the results of ATCRF and our human motion prediction method. The classification precision and recall of our algorithm is lower than that of the ATCRF algorithm. Note that ATCRF uses object affordance annotations as additional information in the learning phase, whereas our algorithm does not. There is a big difference in motion regression result. With our definition of regression precision, our algorithm predicts human motion trajectory 2 times closer to the ground-truth human motion trajectory than that of ATCRF. Also, we could measure regression accuracy with our algorithm that could be used in probabilistic collision detection, whereas ATCRF could not.

Algorithm	CAD-120 Dataset			
	Action Classification		Motion Regression	
	Precision	Recall	Precision	Accuracy
ATCRF	74.8	66.2	0.305	N/A
Our model	68.3	60.3	0.153	1.25

Table 4.1: Performances of action classification and motion regression of ATCRF [5] and our human motion prediction algorithm. As ATCRF algorithm does not predict the variance of motion around the trajectory, the motion regression accuracy cannot be measured on the algorithm. ATCRF uses object affordance annotations as additional information in the learning phase, and we got higher precision and recall with ATCRF than with our algorithm. In terms of motion regression, our algorithm outperformed ATCRF in regression precision.

4.5.3 Robot Motion Planning with Human Motion Prediction

In the simulated benchmark scenario, the human is sitting in front of a desk. In this case, the robot arm helps the human by delivering objects from one position that is far away from the human to target position closer to the human. The human waits till the robot delivers the object. As different tasks are performed in terms of picking the objects and their delivery to the goal position, the temporal coherence is used to predict the actions. The action set for a human is $A^h = \{Take_0, Take_1, \dots\}$, where $Take_i$ represents an action of taking object i from its current position to the new position. The action set for the robot arm is defined as $A^r = \{Fetch_0, Fetch_1, \dots\}$.

We quantitatively measure the following values:

- Modified Hausdorff Distance (MHD) [114]: The distance between the ground-truth human trajectory and the predicted mean trajectory is used. In our experiments, MHD is measured for an actively moving hand joint over 1 second. This evaluates the quality of the anticipated trajectory of the human motion.
- Smoothness: We also measure the smoothness of the robot’s trajectory with and without human motion prediction results. The smoothness is computed as

$$\frac{1}{T} \int_0^T \sum_{i=1}^n \ddot{\mathbf{q}}_i(t)^2 dt, \quad (4.12)$$

where the two dots indicate acceleration of joint angles.

- Jerkiness: Jerkiness is defined as

$$\max_{0 \leq t \leq T} \sum_{i=1}^n \ddot{\mathbf{q}}_i(t)^2. \quad (4.13)$$

- Distance: The closest distance between the robot and the human during task collaboration.
- Efficiency: The ratio of the task completion time when the robot and the human collaborate to complete all the subtasks to the task completion time when the human performs all the tasks without any help from the robot. This is used to evaluate the capability of the resulting human-robot system.

To compare the performance of our I-Planner, we use the following algorithm:

- **ITOMP**. This model is the same as the realtime motion planner for dynamic environments, ITOMP [12], without human motion prediction.
- **I-Planner, no noisy input (I-Planner, no NI)**. This is our motion planning algorithm with human motion prediction, but the motion prediction does not assume noisy input. The details of this algorithm are given in the preliminary paper [36].
- **I-Planner, noisy input (I-Planner, NI)**. This is the modified algorithm presented in the previous section, which also considers noise in the human motion prediction data.

Table 4.2 highlights the performance of our algorithm in three different variations of this scenario: *arrangements of blocks*, *task order* and *confidence level*. The numbers in the "Task Order" column indicates the identifiers of human actions. Parentheses mean that the human actions in the parentheses can be performed in any order. Arrows mean that the right actions can be performed only if the left actions are done. For example, $(0, 1) \rightarrow (2, 3)$ means that the possible action orders are $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$, $0 \rightarrow 1 \rightarrow 3 \rightarrow 2$, $1 \rightarrow 0 \rightarrow 2 \rightarrow 3$ and $1 \rightarrow 0 \rightarrow 3 \rightarrow 2$. Table 4.3 shows the performance of our algorithm in a simulation compared to algorithms without human motion prediction or input noise. The 3 different algorithms were tested in 9 different scenarios and we found that our complete algorithm outperformed other versions of the algorithm in various measurements in most scenarios. Table 4.4 highlights the performance of our algorithm with a real robot. Our

algorithm has been implemented on a PC with 8-core i7-4790 CPU. We used OpenMP to parallelize the computation of future human motion prediction and probabilistic collision checking.

Scenarios	Arrangement	Task Order	Confidence Level	Average Prediction Time
Different Arrangements	1×4	$(0, 1) \rightarrow (2, 3)$	0.95	52.0 ms
	2×2	$(1, 5) \rightarrow (2, 6)$	0.95	72.4 ms
	2×4	$(0, 4) \rightarrow (1, 5) \rightarrow (2, 6) \rightarrow (3, 7)$	0.95	169 ms
Temporal Coherence	1×4	$0 \rightarrow 1 \rightarrow 2 \rightarrow 3$	0.95	52.1 ms
	1×4	Random	0.95	105 ms
	1×4	$(0, 2) \rightarrow (1, 3)$	0.95	51.7 ms
Confidence Level	1×4	$0 \rightarrow (1, 2) \rightarrow 3$	0.90	47.2 ms
	1×4	$0 \rightarrow (1, 2) \rightarrow 3$	0.95	50.7 ms
	1×4	$0 \rightarrow (1, 2) \rightarrow 3$	0.99	155 ms

Table 4.2: Three different simulation scenarios: *Different Arrangements*, *Temporal Coherence*, and *Confidence Level*. We consider different arrangements of blocks as well as the confidence levels used for probabilistic collision checking. These confidence levels are used for motion prediction.

4.5.4 Robot Motion Responses to Human Motion Speed

Figure 4.5 shows the robot’s responses to three different speeds of human movements in a human-robot scenario. In this scenario, the human arm serves as a block or an obstacle to the robot’s motion from left to right. The human moves at a slow speed in (a), a medium speed in (b), and a fast speed in (c). Because the human motion prediction and the robot motion planning process operate in parallel at the same time, the motion planner needs to account for the current results of the motion predictor. If the human moves slowly, the robot motion planner is given the future predicted human motion and the planner has enough planning time to adjust the robot’s trajectory. This results in smooth and collision-free trajectories for the robot. However, if the robot moves quickly, the prediction is not very accurate due to the limited processing time. At the next planning timestep, the robot’s trajectory may abruptly change to avoid the current human pose, generating a jerky or non-smooth motion. This highlights how the performance of the prediction algorithm affects the smoothness of a robot’s trajectory.

4.5.5 Benefits of Our Prediction Algorithm

In the *Different Arrangements* scenarios, the position and layout of the blocks changes. Figure 4.6 shows three different arrangements of the blocks: 1×4 , 2×2 and 2×4 . In the two cases 2×2 and 2×4 , where positions are arranged in two rows unlike the 1×4 scenario, the human arm blocks a

Scenarios	Model	Pred. Time	MHD	Smooth.	Jerkiness	Distance	Efficiency
Different Arrangements (1)	ITOMP	N/A	N/A	2.96	5.23	3.2 cm	1.2
	IP, no NI	52.0 ms	6.7 cm	1.08	1.52	6.7 cm	1.6
	IP, NI	58.5 ms	7.2 cm	0.92	1.03	8.1 cm	1.7
Different Arrangements (2)	ITOMP	N/A	N/A	5.78	7.19	2.3 cm	1.1
	IP, no NI	72.4 ms	6.2 cm	1.04	1.60	8.2 cm	1.6
	IP, NI	70.8 ms	7.0 cm	0.84	1.32	10.7 cm	1.6
Different Arrangements (3)	ITOMP	N/A	N/A	4.82	6.82	1.6 cm	1.2
	IP, no NI	169 ms	10.4 cm	1.15	1.30	6.2 cm	1.6
	IP, NI	150 ms	12.6 cm	1.02	1.20	8.9 cm	1.6
Temporal Coherence (1)	ITOMP	N/A	N/A	1.79	3.22	6.0 cm	1.3
	IP, no NI	52.1 ms	4.3 cm	0.65	1.56	9.3 cm	1.5
	IP, NI	48.9 ms	5.2 cm	0.62	1.53	9.7 cm	1.5
Temporal Coherence (2)	ITOMP	N/A	N/A	5.49	7.30	2.0 cm	1.0
	IP, no NI	105 ms	8.2 cm	1.21	1.28	8.8 cm	1.6
	IP, NI	110 ms	9.9 cm	1.08	1.10	9.9 cm	1.6
Temporal Coherence (3)	ITOMP	N/A	N/A	3.12	3.18	8.0 cm	1.3
	IP, no NI	51.7 ms	6.8 cm	1.00	1.20	12.1 cm	1.5
	IP, NI	60.0 ms	8.2 cm	0.79	0.93	13.2 cm	1.5
Confidence Level (1)	ITOMP	N/A	N/A	2.90	3.40	7.2 cm	1.2
	IP, no NI	47.2 ms	7.9 cm	1.17	1.58	9.5 cm	1.6
	IP, NI	52.1 ms	9.1 cm	1.08	1.32	10.2 cm	1.7
Confidence Level (2)	ITOMP	N/A	N/A	3.12	3.80	10.3 cm	1.2
	IP, no NI	50.7 ms	7.9 cm	1.28	1.71	13.3 cm	1.6
	IP, NI	48.2 ms	9.1 cm	1.20	1.66	14.1 cm	1.8
Confidence Level (3)	ITOMP	N/A	N/A	3.76	4.33	13.0 cm	1.2
	IP, no NI	155 ms	7.9 cm	1.40	1.90	16.2 cm	1.7
	IP, NI	129 ms	9.1 cm	1.35	1.73	17.7 cm	1.8

Table 4.3: Performance of our planning algorithm in terms of robot motion simulation. The use of motion prediction results in a smoother trajectory and we observe up to 4X improvement in our smoothness metric, as defined in Equation (4.12). Our resulting planning algorithm (I-Planner) runs in real-time.

movement from a front position to the back position. As a result, the robot needs to compute its trajectory accordingly.

Depending on the temporal coherence present in the human tasks, the human intention prediction may or may not improve the performance of our the task planner. It is shown in the *Temporal Coherence* scenarios. In the sequential order coherence, the human intention is predicted accurately with our approach with 100% certainty. In the random order, however, the human intention prediction step is not accurate until the human hand reaches the specific position. The personal order varies for each human, and reduces the possibility of predicting the next human action. When the right arm

Scenarios	Model	Pred. Time	MHD	Smoothness	Jerkiness	Distance
Waving Arms	ITOMP	N/A	N/a	4.88	6.23	2.1 cm
	IP, no NI	20.9 ms	5.0 cm	0.91	1.33	9.3 cm
	IP, NI	23.5 ms	6.1 cm	0.83	1.25	10.5 cm
Moving Cans	ITOMP	N/A	N/A	5.13	7.83	3.9 cm
	IP, no NI	51.7 ms	7.3 cm	1.04	1.82	8.7 cm
	IP, NI	50.0 ms	8.8 cm	0.93	1.32	13.5 cm

Table 4.4: Performance of our planning algorithm on a real robot running on a 7-DOF Fetch robot next to dynamic human obstacles. Our online motion planner computes safe trajectories for challenging benchmarks like “moving cans.” We observe almost 5X improvement in the smoothness of the trajectory due to our prediction algorithm.

moves forward a little, $Fetch_0$ is predicted as the human intention with a high probability whereas $Fetch_1$ is predicted with low probability, even though position 1 is closer than position 0.

In the *Confidence Level* scenarios shown in Figure 4.7, we analyze the effect of confidence level δ_{CD} on the trajectory computed by the planner, the average task completion time, and the average motion planning time. As the confidence level becomes higher, the robot may not take the smoothest and shortest path so as to compute a collision-free path that is consistent with the confidence level.

In all cases, we observe the prediction results in smoother trajectory, using the smoothness metric defined as Equation (4.12). This is because the robot changes its path in advance before the human obstacle actually blocks the robot’s shortest path if human motion prediction is used.

4.5.6 Evaluation Using 7-DOF Fetch Robot

We integrated our planner with the 7-DOF Fetch robot arm and evaluated in complex 3D workspaces. The robot delivers four soda cans from start locations to target locations on a desk. At the same time, the human sitting in front of the desk picks up and takes away the soda cans delivered to the target positions by the robot, which can cause collisions with the robot arm. In order to evaluate the collision avoidance capability of our approach, the human intentionally starts moving his arm to a soda can at a target location, blocking the robot’s initially planned trajectory, when the robot is delivering another can moving fast. Our intention aware planner avoids collisions with the human arm and results in a smooth trajectory compared to motion planner results without human motion prediction.

Figure 4.8 shows two sequences of robot’s trajectories. In the first row, the robot arm trajectory is generated an ITOMP [12] re-planning algorithm without human motion prediction. As the human

and the robot arm move too fast to re-plan collision-free trajectory. As a result, the robot collides (the second figure) or results in a jerky trajectory (the third figure). In the second row, our human motion prediction approach is incorporated. The robot re-plans the arm trajectory before the human actually blocks its way, resulting in collision-free path.

4.6 Conclusions, Limitations, and Future Work

We present a novel intention-aware planning algorithm to compute safe robot trajectories in dynamic environments with humans performing different actions. Our approach uses offline learning of human motions and can account for large noise in terms of depth cameras. At runtime, our approach uses the learned human actions to predict and estimate the future motions. We use upper bounds on collision guarantees to compute safe trajectories. We highlight the performance of our planning algorithm in complex benchmarks for human-robot cooperation in both simulated and real world scenarios with 7-DOF robots. Compared to [36], our improved human motion prediction model can better handle input noise and can generate smoother robot trajectories.

Our approach has some limitations. As the number of human action types increases, the number of states of MDP can increase significantly. In this case, it may be useful to use POMDP for robot motion planning under uncertainty [115]. Our classification can be improved with more information of the robotic environment such as object affordances. So we would like to improve our algorithm by using additional annotations as future work. Our probabilistic collision checking formulation is limited to environment uncertainty and does not take into account robot control errors. The performance of motion prediction algorithm depends on the variety and size of the learned data. Currently, we use supervised learning with labeled action types, but it would be useful to explore unsupervised learning based on appropriate action clustering algorithms. Furthermore, our analysis of human motion prediction noise assumes the use of a Gaussian Process model and it would be useful to extend it other noise models. Moreover, we would like to measure the impact of robot actions on human motion, and thereby establish a two-way coupling between robot and human actions. Our realtime planner can also be combined with natural language processing to generate robot movements [116].

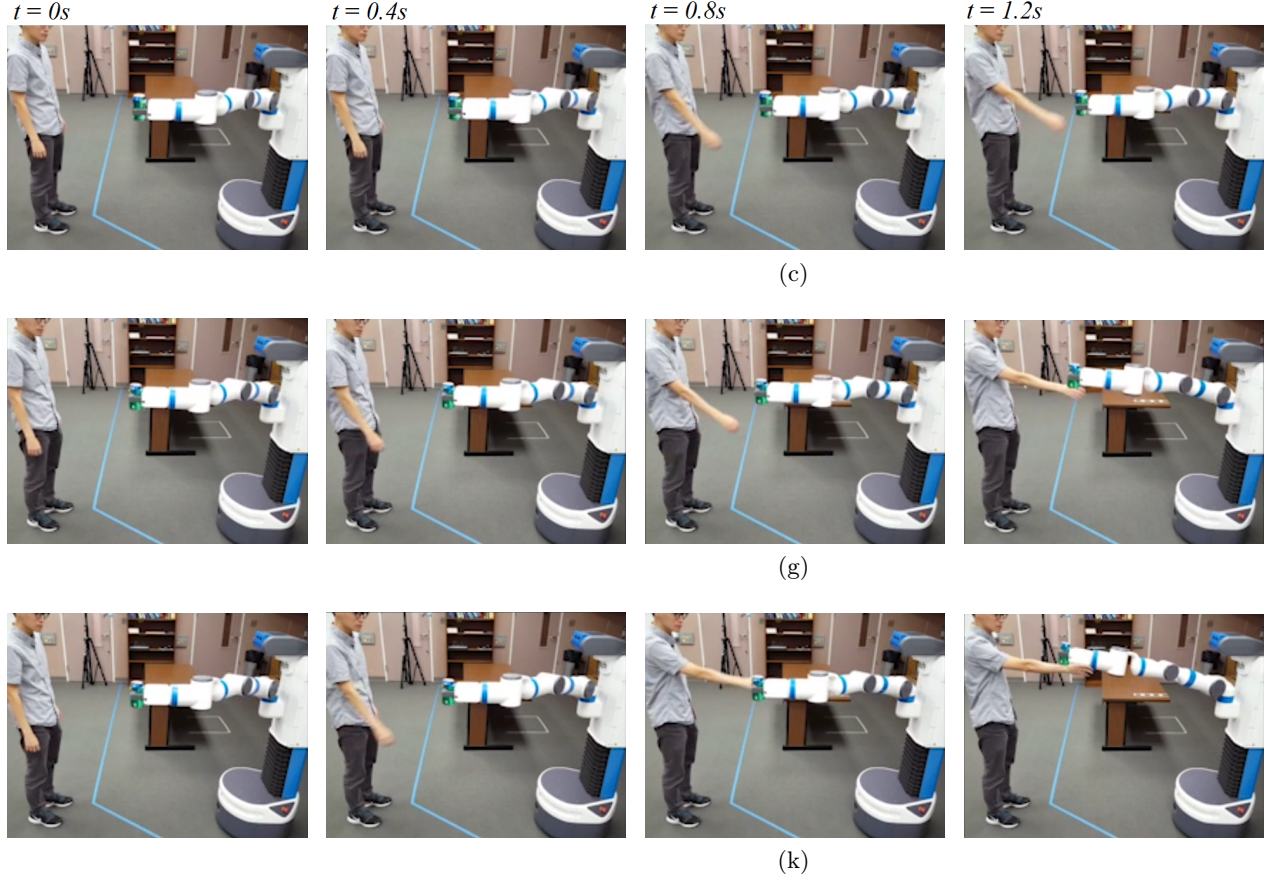


Figure 4.5: Responses to three different human arm movement speeds: While the robot arm moves from left to right, the human moves his arm to block the robot's trajectory at different speeds. (a) The human arm moves slowly. The robot has enough time to predict the human arm motion, generating the smoothest and the least jerky robot trajectory. (b) As the human moves at a medium speed, the robot predicts the human's future motion, recognizes that it will block the robot's path, and therefore changes the trajectory upwards (at $t = 0.8s$) to avoid the obstacle and generate a smooth trajectory. (c) When the human arm moves faster, the robot trajectory abruptly changes to move upwards (at $t = 0.8s$), generating a less smooth trajectory while still avoiding the human.

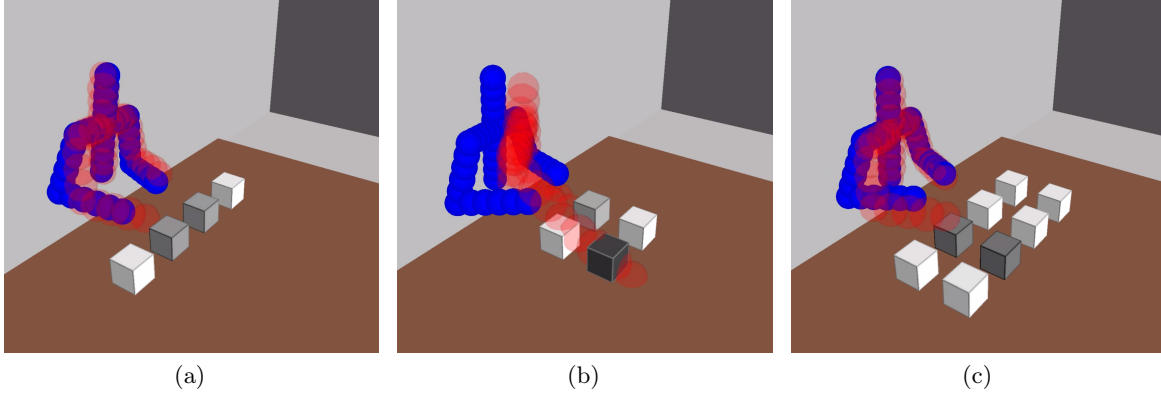


Figure 4.6: **Different block arrangements:** Different arrangements in terms of the positions of the blocks, results in different human motions and actions. Our planner computes their intent for safe trajectory planning. The different arrangements are: (a) 1×4 . (b) 2×2 . (c) 2×4 .

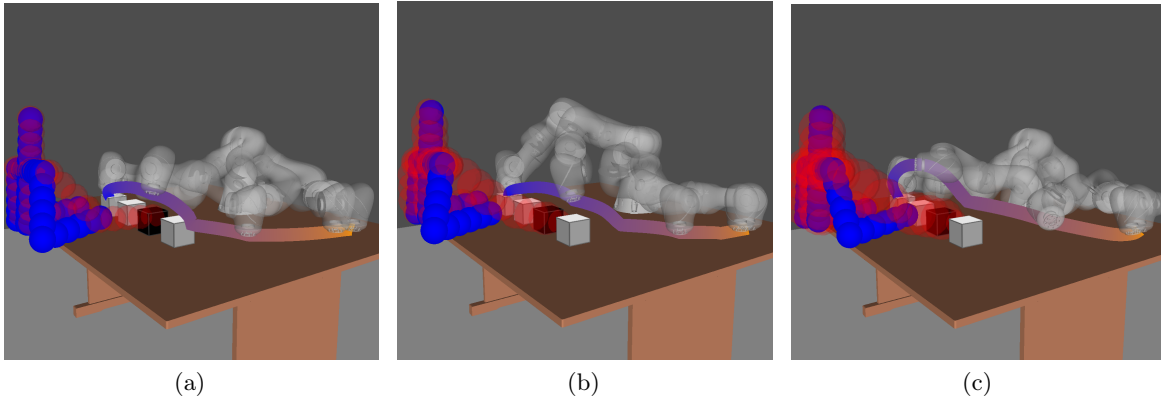


Figure 4.7: **Probabilistic collision checking with different confidence levels:** A collision probability less $(1 - \delta_{CD})$ implies a safe trajectory. The current pose (i.e., blue spheres) and the predicted future pose (i.e. red spheres) are shown. The robot's trajectory avoids these collisions before the human performs its action. The higher the confidence level is, the longer the distance between the human arm and the robot trajectory. (a) $\delta_{CD} = 0.90$. (b) $\delta_{CD} = 0.95$. (c) $\delta_{CD} = 0.99$.

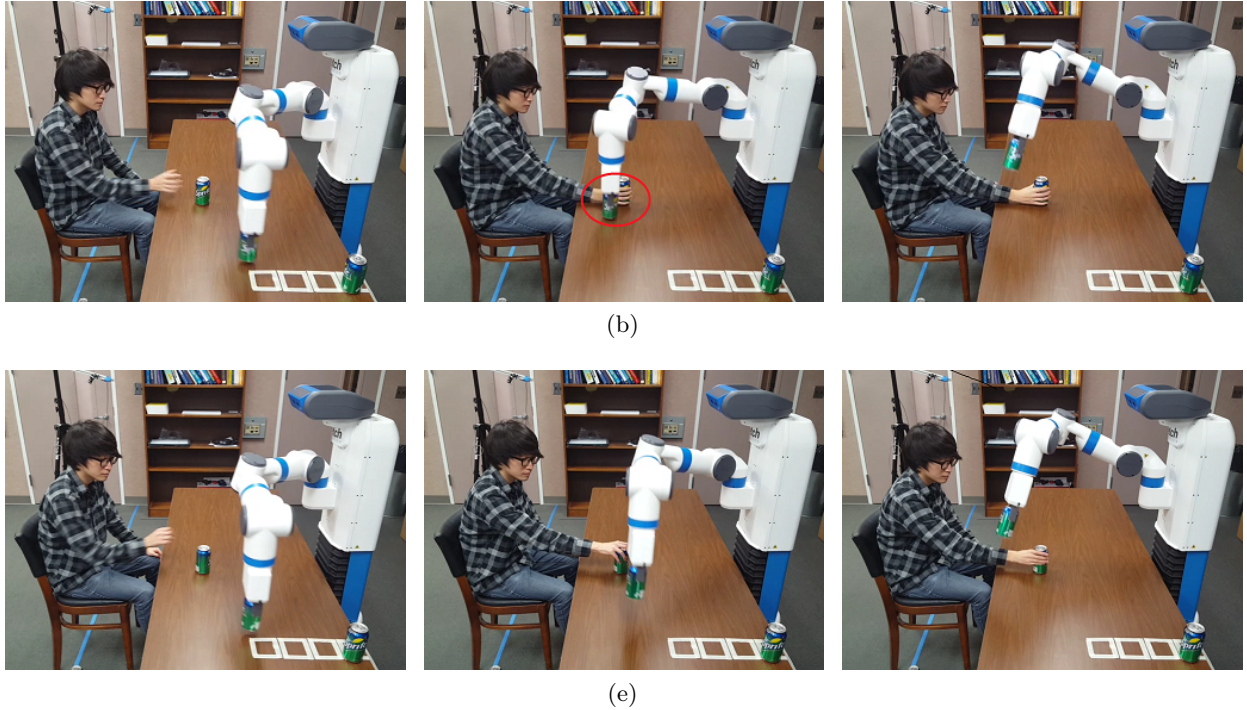


Figure 4.8: A 7-DOF Fetch robot is moving its arm near a human, avoiding collisions. (a) While the robot is moving, the human tries to move his arm to block the robot’s path. The robot arm trajectory is planned without human motion prediction, which may result in collisions and a jerky trajectory, as shown with the red circle. This is because the robot cannot respond to the human motion to avoid collisions. (b) The trajectory is computed using our human motion prediction algorithm; it avoids collisions and results in smoother trajectories. The robot trajectory computation uses collision probabilities to anticipate the motion and compute safe trajectories.

CHAPTER 5

Occlusion-Aware Robot Motion Planning

Human motion prediction is an important part of human-robot interaction in environments where robots work in close proximity to humans. Traditionally, industrial robots were isolated from humans for safety. However, humans can handle jobs that require better dexterity than robots [9, 10], meaning that, for some applications, it is more efficient for humans and robots to work together while sharing the same workspace. In these scenarios, it is important for a robot to observe and predict human motion and plan its tasks accordingly.

A key challenge in achieving safety and efficiency in human-robot interactions is computing a collision-free path for the robot to reach its goal configuration. The robot should not only complete its task but also predict the human’s motion or trajectory to avoid the human as a dynamic obstacle. There is considerable work on human motion prediction as well as safe trajectory computation. Some recent methods predict human motions from images or videos based on Convolutional Neural Networks (CNNs) [25, 26, 27, 28] or Recurrent Neural Networks (RNNs) [29, 30].

When robots are in close proximity with humans, they gather information about the surrounding environment using visual sensors (color cameras, depth cameras, etc.). Typically, head-mounted cameras on the robots observe the workspace. As robots perform actions with their hands or arms, the moving parts of the robot may occlude the views of these sensors. As a result, the resulting images cannot capture information about many parts of the scenes, including the current position of the human working close to the robot [34, 35, 36]. Such occlusion by parts of a robot can prevent accurate tracking and prediction of the human motion, thereby making it difficult to perform safe and collision-free motion planning. When the robot arm occludes the input images, the robot should either determine whether the human motion can be predicted with high certainty or move its arm in such a manner that it does not occlude the field of view of the camera (i.e. remove occlusions), as

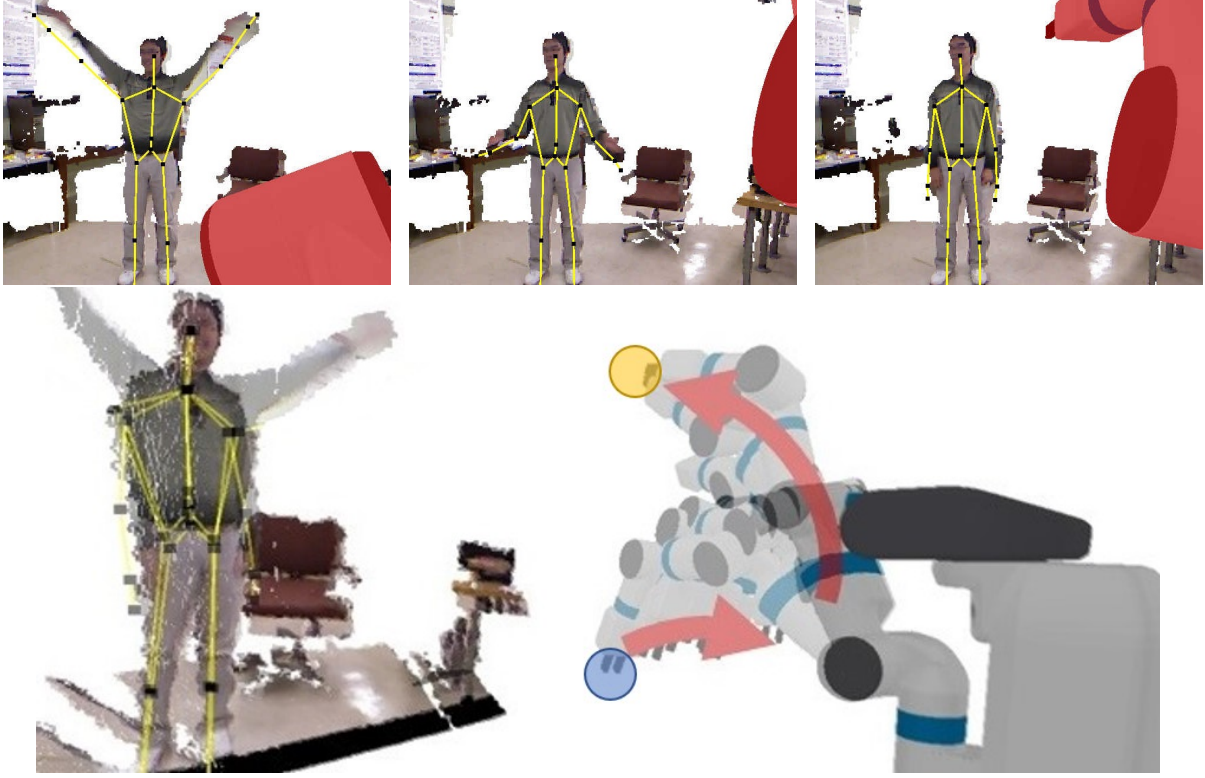


Figure 5.1: A human and a robot are simultaneously operating in the same workspace. The robot arm occludes the camera view and many parts of the human obstacle are not captured by the camera. Three images at the top show the point clouds corresponding to the human in the UtKinect dataset [2] for different camera positions with the occluded regions in red. The bottom right image highlights the safe motion trajectory between the initial position (blue) and the goal position (yellow). Our safe trajectory is shown in the bottom right as two red curves (with arrows). HMPO first moves the arm to reduce the occlusion and then moves it to the goal position.

shown in Fig. 5.1. This results in two main challenges:

- The human motion predictor should be aware of the region of overlap between the human obstacle and the robot in the input image. These regions occur when the human moves into the shadow region of the camera or when the robot parts occlude the region corresponding to the human in the input image. In such scenarios, prior human motion predictors do not work well.
- The robot motion planner should respond in realtime when the human motion cannot be accurately predicted due to occlusion. The robot motion planner should compute a safe path by considering these occlusion constraints.

We present a novel approach to overcome the challenges in human motion prediction and robot

motion planning under occlusion.

- Human motion prediction based on neural networks uses additional input images to improve the prediction accuracy. From the color camera images and the joint angle values of the robot, the occluding parts can be segmented from forward kinematics. The occluded images can be easily obtained from occlusion-free human motion datasets by overlaying images from these datasets with virtual robot images. When the additional input images are fed to the neural networks, we observe improved prediction accuracy in both human action classification and human pose prediction, compared to the neural network model using only raw input images.
- An optimization-based robot motion planner with an occlusion-related cost function helps avoid collisions and maintains reliable human skeleton tracking without occluding humans. An occlusion sensitive cost function uses the results of human motion prediction under occlusion. It reduces the degree of occlusion in the future and helps avoid future collision due to untrackable human motion behind occlusions.

5.1 Related Work

In this section, we give a brief overview of prior work on prediction and occlusion handling in computer vision and robotics.

5.1.1 Human Motion Prediction for Robotics

Human motion prediction has been shown to be useful to guide collaborative robots in human-robot interaction systems [117]. The Multiple-Predictor System is a method combining multiple data-driven human motion predictors [118]. The goal-set Inverse Optimal Control algorithm plans human motion trajectories and considers them as moving obstacles in the robot motion planning step [119]. Probability models for future human motions can be used in generating collision-free robot motions. For 2D navigation robots, the probability distribution of a human’s future position on a grid map can be predicted based on a human motion model, where parameters of the motion model are approximated and learned from the motion data [52]. For 3D collaborative applications, the whole-body joint poses of humans may be predicted [36]. From the tracked human skeleton joint positions, a Gaussian probability distribution can be constructed and learned through Gaussian Processes [108], and the future human motion is predicted and presented as Gaussian distributions. All of the algorithms require fully observable information about the human motion and do not

account for occlusion. If the human motion is not fully visible, the probability distributions for non-observable human body parts will have high variances; thus the predicted future human motion is not accurate enough to generate collision-free robot motions.

5.1.2 Human Motion Prediction from Images and Videos

Motion prediction algorithms can be categorized as model-based approaches or motion analysis without an a priori human shape model [120, 121]. Human motion models usually have a high degree-of-freedom (DOF) configuration space. For skeleton model-based human models, Hidden Markov Models (HMMs) are used to predict skeleton joint positions [122]. Deep learning-based Recurrent Neural Networks (RNNs) can be used for sequences of high-DOF human shape models [123]. An occlusion removing algorithm for self-occlusion of 3D objects and robot occlusion from robot grippers is used for robot motion planning in [124]. From 3D point cloud stream data, this method recovers points that were not occluded in previous frames, but are occluded in the current frame. After recovering occluded 3D point clouds, they extract features from the point clouds and use them in RNN. However, this approach is mainly designed for deformable objects manipulated by robot grippers. The prediction of high-DOF human motions has additional challenges due to occlusion or limited sensor ranges. Dragan et al. [125] propose improved assistive teleoperation with predictions of the motion trajectory to reach the goal using inverse reinforcement learning. Koppula and Saxena [5] use spatial and temporal relations of object affordances to predict future human actions.

5.1.3 Object Recognition under Occlusions in a Cluttered Environment

Self-occlusions or occlusions from surrounding objects have been investigated in the context of object recognition and object tracking algorithms. Multiple moving cars can be tracked from video data where some cars are occluded by others. Without occlusions, a linear translational and scaling motion model for cars fits for tracking cars and the motions are computed by differentiating consecutive frames of images [126]. Prior works have also used image features to overcome the occlusion problem. Histograms of Oriented Gradients (HOG) and Local Binary Pattern (LBP) have been considered as representative visual features and can be used in a Support Vector Machine (SVM) classifier to segment the occlusions and detect humans behind occlusions [127] from input color images. Human model-based body part tracking under an occluding blanket in hospital monitoring applications has been developed [128]. This is a specialized technique for this application. From input depth images of a human occluded by obstacles [4], human joint positions can be tracked

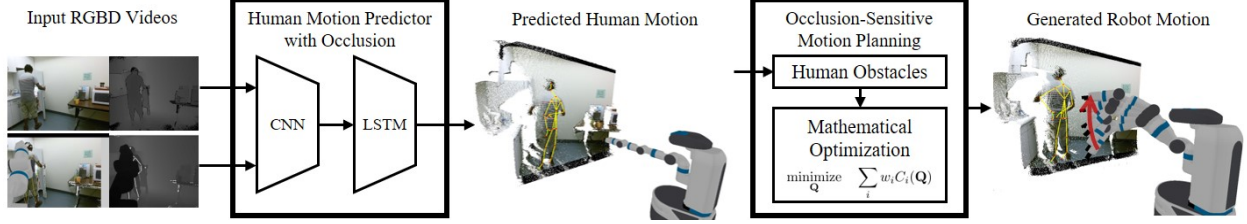


Figure 5.2: **HMPO**: Overall pipeline of our human motion prediction and robot motion planning. We present a new deep learning technique for human motion prediction in occluded scenarios and an optimization-based planning algorithm that accounts for occlusion.

from a hierarchical particle filter, where occlusions are handled with a 3D occupancy grid and a Hidden Markov Model (HMM) is used to represent the state of visibility and occlusion. However, it is unable to track parts that are not visible. To overcome and respond to occlusions in object recognition or human body pose estimation, the visibility of occluded objects or human body parts can be computed using supervised learning [129, 130]. By labeling the visibility of body parts with 0 and 1 in the training data and minimizing the loss function for visibility, the visibility is then inferred as a probability in the range of $[0, 1]$.

Our approach is more general and complimentary than the methods discussed above. Not only do we present a novel deep learning-based method to predict human motion in occluded scenarios, but we also compute a motion trajectory for a robot that reduces occlusions. Moreover, we exploit robot kinematics and self-occlusion capabilities to achieve higher classification accuracy than prior methods.

5.2 Overview

In this section, we describe our problem and the assumptions made by our algorithm. Furthermore, we give an overview of the overall approach combining human motion prediction and occlusion-aware motion planning.

5.2.1 Problem Statement and Assumptions

Figure 5.2 highlights the different components of our approach. In our environment, we assume that there is a collaborative robot with one or more robot arms and a camera. Moreover, the robot is operating in close proximity to a human obstacle, and our goal is to compute a collision-free and safe trajectory for the robot. We assume that the human is active and the robot is passive while the robot arm shares the same workspace with the human. The human either performs actions as if there were no robots nearby or as if he or she believes the robot will avoid collisions.

In these scenarios, the robot tracks and predicts the motion of the human using the camera and uses that information for safe planning. We extract the human skeleton from the image and use the skeleton for motion prediction (see Fig. 5.1). Our approach is designed for environments, where the robot’s motion results in self-occlusions with respect to the camera. This happens for configurations, where the robot arm either fully or partially occludes the human. The input of the human motion predictor is captured from a single RGBD camera attached to the robot’s head. Our approach can also work with 2D RGB cameras. The RGB and depth image frames are fed as input to the human motion predictor at a fixed frame rate, which is governed by the underlying camera hardware and the training datasets. For example, the Kinect V2 sensor streams color and depth images at 30 frames per second. The camera position and angle are set to capture the human’s motion. The outputs of the human motion predictor are the human action, the future human motion with the skeleton-based human model, and the certainty value related to the probability that the human motion can be predicted accurately in the occluded scenarios.

Real-time Planning: We present an occlusion-aware realtime motion planning algorithm. Our planner takes as input the current configuration of the robot, including the arm, and computes a high-dimensional trajectory in the configuration space that is represented in the space corresponding to the robot configuration $\mathbf{q} \in \mathbb{R}^n$ and the time $t \in \mathbb{R}$. The trajectory connects the robot’s configuration at the current time to the goal configuration at a later time. The future motion of the human is predicted from our deep learning-based human motion predictor, represented using a skeleton-based model. Our planner takes this predicted trajectory into account for safe motion planning. Our planner modifies this trajectory in real-time in response to the obstacles in the environment and considers two constraints:

1. Collision avoidance with static obstacles and predicted paths of dynamic obstacles, especially humans.
2. Moving the robot arms so they do not occlude the human from the camera’s point of view.

This way, the accuracy of the human motion predictor will improve in subsequent frames.

We present an optimization-based planner based on these constraints.

5.3 Human Motion Prediction with Ocluded Videos

In this section, we present our novel human motion prediction algorithm that accounts for occlusions in the scene.

5.3.1 Neural Network for Ocluded Videos



Figure 5.3: Sample images of original datasets and modifications with occlusion information. (a) UTKinect dataset [2]. (b) Watch-n-patch dataset [3]. (c) Occlusion MoCap dataset [4]. We present 3 image pairs for each dataset in each column. The top image in each pair is the original image from the dataset, and the bottom images are generated by augmenting the original images with robot arm occlusions at the bottom. These augmented images are used for training and cross-validation.

Our approach is based on convolutional neural networks (CNNs), which have been widely used for image classification and recognition [25, 26, 27, 28]. We first extract the features, which are used by LSTMs, from the pre-trained ImageNet [25]. In addition to the image features, we also take into account occlusion features. The deep neural network is provided with the input color image sequence, the depth image sequence, and an occlusion mask image sequence. To facilitate the robot’s

early response, we need to predict the human action class quickly.

The input image sequence contains the human upper body action. The color and depth images may be occluded by the robot arm, and it is assumed that the robot knows which parts of the images are being occluded, as shown in the red regions in Figure 5.1. We use forward kinematics based on robot joint values and the robot camera position to compute the occlusion region in the image. The output corresponds to the human action class, the future human motion in a short time window, and the confidence value of the human motion prediction. For action classification, our prediction algorithm outputs a discrete probability distribution for various action classes included in the datasets. For the future human motion, the human skeletal joint positions are predicted. Those predictions will have a 100% confidence level, if the robot's configuration does not result in self-occlusions. The confidence level decreases when the human motion is partially occluded; at 0%, the human motion is completely hidden.

Recurrent Neural Networks and Long Short-Term Memory (LSTM) models are useful for constructing deep neural networks for temporal sequences. We exploit these models to predict human actions and future motions with the RGBD input image sequences, which may be partially occluded by the robot arm. In addition to the pre-trained CNN features from the color and depth images, we also use a neural network input for the occlusion image to adjust the human motion prediction results and generate the confidence level of the certainty with which the human motion can be predicted. The feature vectors of color and depth and the occlusion images are fed to the LSTM. The features from depth images and occlusion images are different and are used to generate accurate confidence level results. The output contains the information about action classification, future human joint position, and degrees of occlusions. For each action class, a real value between 0 and 1 represents the likelihood that the human is performing a certain action. The predicted action is the one with the highest value among the action classes.

The input color and depth images are first cropped around the human with the resolution 224×224 to feed the input for resnet-18. The output of the pre-trained CNN is a vector of size 1,000 for each color and depth image. The column vector describing the skeleton joint positions has x, y, z components for each joint. These values are concatenated and connected to a fully-connected layer of size 1,000 followed by LSTM.

The outputs of the neural network are the x, y , and z components of the future human joint

position, future human action class, and the confidence value. Future human joint positions are predicted up to 3 seconds ahead of time. The 3-second time window is discretized using 0.5s timesteps (i.e. “prediction timestep”), resulting in 6 time points at which the joint positions are predicted. The x, y, and z coordinates of each joint compose the output vector. The degree of occlusion is represented by a real value between 0 and 1. A value near 1 implies that the joint position is difficult to predict due to robot occlusions, whereas a value near 0 means the joint is not occluded by the robot. To train the future joint positions, the ground-truth joint positions in the sequence for each timestep ahead of the current time are used as the expected outputs. To avoid the redundancy of temporal relationships from LSTM and the output values, the values for predicted joint positions and the degrees of occlusion in the output layer do not interconnect with those values from different time points.

5.3.2 Dataset Generation

In the field of computer vision, synthetic data has been used widely, reducing the efforts of collecting data and improving prediction performance [131, 132]. There is very little data from real-world scenarios in terms of humans reacting when they are close to robots. Usually, when robot motion planners work in close proximity with humans in the real-world, the color and depth cameras are installed at a location that minimizes the robot occlusions and human self-occlusions while still accurately tracking human skeleton joints. As a result, synthetic datasets are used to generate results for our supervised learning method. Our synthetic datasets have robot images overlaid on the original dataset, as if the robot arm image was captured from the viewpoint of the head-mounted camera.

To train the neural network, we extend three existing datasets for training and cross-validation by adding robot occlusions in the images. There may be some small errors in synthesized datasets, such as pixel color values, depth values, and joint angles of actual motors, compared to real-world captured images with robot occlusions. However, our main problem is predicting the human joint positions and human action class behind the robot occlusion, and the regions of occlusion from forward kinematics. Our approach provides a robust solution to predict human motions accurately with synthesized training data. Furthermore, we added a new action class in these datasets to represent whether the human is occluded by the robot.

UTKinect-Action dataset [2] (Figure 5.3 (a)) contains 10 types of human actions (*Walk, Sit Down,*

etc.) and each action has about 18 to 20 RGBD videos captured with Kinect v1. The resolution of the RGB videos is 640×480 , whereas the resolution of the depth videos is 320×240 . The actions are performed with 10 different subjects. The videos are captured in the same space (a lab) with the same Kinect position and angle.

Watch-n-Patch dataset [3] (Figure 5.3 (b)) provides RGBD videos of 21 types of human actions performed by 7 subjects captured with Kinect v2. The resolution of the RGB videos is 1920×1080 , whereas the resolution of the depth videos is 512×424 . The videos are captured in 8 offices and 5 kitchens with different Kinect positions and angles.

Occlusion MoCap dataset [4] (Figure 5.3 (c)) has RGBD videos of a human with joint tracking Qualysis markers on his body and a static object in the middle of the room. There are 4 videos with lengths between 45 and 60 seconds captured at 15 frames per second. In the videos, a person comes into the space, walks around the chair in the middle of the space, and sits down. The dataset has 640×480 resolution in both color and depth images. While the action labels are not given in the dataset, this one provides more accurate joint positions than the other two datasets highlighted above.

In all the datasets, only one human subject performs the actions and human skeleton tracking data are available. We add a robot arm occlusion in both the RGB videos and depth videos of the UTKinect, Watch-n-Patch, and Occlusion MoCap datasets to make them effective for our prediction algorithm. The robot occlusions are added as if the videos are captured by a camera on a virtual robot, where the robot arm is moving around in the same space that is used to perform human actions. The inserted robot occlusions are rendered with simulated geometric models of the robot and appropriate models of light to simulate the images and occlusion. The regions of occlusion are computed using forward kinematics. It is accurate up to the resolution of the image-based methods. Because the humans in the original dataset are moving without the presence of robot, those captured human motions are neither changed nor affected in the occluded datasets. Therefore, the virtual robot’s goal is to avoid collisions with the humans. In order to generate the virtual robot’s motion, we used the ITOMP optimization-based motion planner [12] to avoid collisions along with probabilistic collision detection [55] to measure collision probability with noisy point cloud data.

The file sizes of the UTKinect, Watch-n-Patch, and Occlusion MoCap datasets are 7GB, 30GB,

and 2GB, respectively, and we generate additional input images with occlusions. Duplicating image files and saving them in storage disks can be inefficient, so we store the synthesized dataset by only storing the robot joint angles for each frame. From the robot joint poses, the RGBD images and occlusion images are obtained by overlaying the robot image on the original images.

When human motions are not fully visible due to occlusions, human action labels cannot be predicted accurately. In this case, we semi-automatically assign an *occluded* label. To determine if the human action can be predicted, we check if the human skeleton tracking data is occluded by the generated virtual robot arm motions. For action labels that are recognized mostly from human hand motions (e.g., *fetch-from-fridge*, *drinking*, or *pouring*), the human action cannot be predicted if the robot arm occludes the human hand. These action labels are changed to *occluded* if the human hand joint is occluded by the virtual robot in the depth image. For other action labels that are recognized from the motion of the whole body (e.g., *walking*, *leave-office*, or *leave-kitchen*), the human action can be predicted if some parts in the RGBD videos are occluded but cannot be predicted if most parts of the human are occluded. These action labels are changed to *occluded* if most of the human joints are occluded by the virtual robot. There are 23 joints in the human skeleton tracking data. We label *occluded* if 20 or more joints are occluded. For the prediction algorithm to be able to predict actions when RGBD videos are not occluded, the original datasets are also included in the training dataset without modification.

The neural network is given the images with occlusions for both training and inference. The synthesized datasets include images without robot occlusions when the robot arm does not occlude the camera. About 50% of the training dataset images have robot occlusions to train human action and joint positions behind occlusions. These data have the *occluded* label and a 0 confidence value for expected output if the robot parts occlude more than half of the human joints. The rest of the images with no occlusions are also necessary to train human action and joint positions without occlusions. These data with and without robot occlusions would be used in the real-world scenarios. The human motion prediction and occlusion-aware motion planner work well without occlusion because the training dataset contains images without occlusions. The robot occlusion does not hide the human, where the certainty values are 1 and the robot motion trajectory is not affected by occlusion-related cost functions. The algorithms also work well with occlusion.

5.4 Occlusion-Aware Motion Planning

In this section, we describe our planning algorithm that uses the human motion prediction results computed in the prior section.

5.4.1 Optimization-Based Planning of Robot Trajectories

We denote a single configuration of the robot as a vector \mathbf{q} , which consists of joint-angles or other degrees-of-freedom. An n -dimensional configuration at time t , where $t \in \mathbb{R}$, is denoted as $\mathbf{q}(t)$. We assume $\mathbf{q}(t)$ is twice differentiable, and its first and second derivatives are denoted as $\mathbf{q}'(t)$ and $\mathbf{q}''(t)$, respectively. We represent bounding boxes of each link of the robot as B_i . The bounding boxes at a configuration \mathbf{q} are denoted as $B_i(\mathbf{q})$.

For a planning task with the given start configuration \mathbf{q}_s and goal configuration \mathbf{q}_g , the robot's trajectory is represented by a matrix \mathbf{Q} , the elements of which correspond to the waypoints [67, 68, 12]:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_0 & \mathbf{q}_1 & \cdots & \mathbf{q}_{n-1} & \mathbf{q}_n \\ \mathbf{q}'_0 & \mathbf{q}'_1 & \cdots & \mathbf{q}'_{n-1} & \mathbf{q}'_n \\ t_0 = 0 & t_1 & \cdots & t_{n-1} & t_n = T \end{bmatrix}. \quad (5.1)$$

The robot trajectory passes through $n + 1$ waypoints q_0, \dots, q_n , which will be optimized by an objective function under constraints in the motion planning formulation. Robot configuration at time t is cubically interpolated from two waypoints.

We use optimization-based robot motion planning [12] for generating robot trajectories in dynamic scenes. The objective function for the optimization-based robot motion planning consists of different types of cost functions. The i -th cost functions of the motion planner are $C_i(\mathbf{Q})$.

$$\begin{aligned} & \underset{\mathbf{Q}}{\text{minimize}} && \sum_i w_i C_i(\mathbf{Q}) \\ & && \mathbf{q}_{min} \leq \mathbf{q}(t) \leq \mathbf{q}_{max}, \\ & \text{subject to} && \mathbf{q}'_{min} \leq \mathbf{q}'(t) \leq \mathbf{q}'_{max}, \quad 0 \leq \forall t \leq T, \\ & && \mathbf{q}_0 = \mathbf{q}_s, \quad \mathbf{q}_n = \mathbf{q}_g \end{aligned} \quad (5.2)$$

for the initial robot configuration \mathbf{q}_s and the goal configuration \mathbf{q}_g . In the optimization formulation, C_i is the i -th cost function and w_i is the weight of the cost function. Every 0.5s timestep, the motion planning problem is updated, and the motion planner adjusts the trajectory with respect to changes

in human motions and prediction of occlusion and human action.

In a static environment where there are no humans or dynamic obstacles, we define the basic cost functions: robot smoothness and collision avoidance with static obstacles.

Smoothness:

$$C_{smoothness}(\mathbf{Q}) = \frac{1}{T} \int_0^T \mathbf{q}'(t)^T \mathbf{D} \mathbf{q}'(t) dt, \quad (5.3)$$

where \mathbf{D} is a diagonal matrix with non-negative values.

Collision avoidance with static obstacles:

$$C_{collision}(\mathbf{Q}) = \frac{1}{T} \int_0^T \sum_i \sum_j dist(B_i(t), O_j)^2 dt, \quad (5.4)$$

where $dist(B_i(t), O_j)$ is the penetration depth between a robot bounding box $B_i(t)$ and a static obstacle O_j .

5.4.2 Occlusion Sensitive Constraints

We account for occlusion characteristics by adding a new soft constraint that prevents the robot from occluding the human obstacle, especially when the certainty in motion prediction is low.

Robot occlusion:

$$C_{occlusion}(\mathbf{Q}) = \frac{1}{T} \int_0^T (1 - \alpha(t))^2 dt, \quad (5.5)$$

where $\alpha(t)$ is the confidence value at time t of human motion prediction, where the robot may have occluded the human image captured by the RGBD sensor. The confidence value is one of the output values of the neural network in Section 5.3.1 and is in the range $[0, 1]$. A confidence value near 1 means that the human is not very occluded by the robot, whereas a value near 0 means that the human motion cannot be accurately predicted. We modify the trajectory to reduce $C_{occlusion}$ and this reduces the overlapping area of the robot and the human portion in the RGBD frames over the duration of the trajectory.

5.4.3 Real-time Collision Avoidance with Predicted Human Motions

In order to avoid collisions with the human obstacle in the 3-second future time period, we add a soft constraint that imposes a penalty in terms of the extent of the penetration depth between the

robot and the predicted human motion.

Collision avoidance with a human:

$$C_{collision}(\mathbf{Q}) = \frac{1}{T} \int_0^T \sum_i \sum_j dist(B_i(t), H_j(t))^2 dt, \quad (5.6)$$

where $dist(B_i(t), H_j(t))$ is the penetration depth between a robot bounding box $B_i(t)$ and the predicted human obstacle $H_j(t)$ at time t . The human obstacle is represented with multiple capsules, each of which connects a pair of joints. $H_j(t)$ represents a capsule with index j , connecting two human joints $\mathbf{h}_{j,1}(t)$ and $\mathbf{h}_{j,2}(t)$, where the joint positions come from the result of the skeleton model-based human motion prediction in Section 5.3.1. For the prediction uncertainty of each joint due to the presence of occlusions, we change the radius of the capsule with respect to the confidence values for the joints $\alpha_{j,1}(t)$ and $\alpha_{j,2}(t)$. To reduce the computation time, we take the average of two confidence values and the radius $r_j(t)$ is linearly interpolated as:

$$\alpha_j(t) = \frac{1}{2}(\alpha_{j,1}(t) + \alpha_{j,2}(t)), \quad (5.7)$$

$$r_j(t) = (1 - \alpha_j(t))r_0 + \alpha_j(t)r_1, \quad r_0 \geq r_1 \quad (5.8)$$

where r_0 and r_1 are user-specified parameters. When the occlusion confidence $\alpha_j(t)$ is 0, this implies that the joints are occluded and the radius is r_0 . On the other hand, when $\alpha_j(t)$ is 1 that implies that the joints are not occluded, and the radius is r_1 .

5.5 Performance and Analysis

5.5.1 Human Action Recognition and Motion Prediction

After generating RGB-D datasets with occlusion characteristics (see Section 5.3.2), we use them for training and evaluation. The Watch-n-patch dataset [3] has a frame rate of 5 frames per second. Each dataset has two types of RGB-D images: *No Occlusion* and *Occlusion* (see Fig. 5.3). We perform 5-fold cross-validation, and these datasets are divided into 5 segments. 4 segments are used for training and the remaining one is used for validation. When splitting the dataset, we split the original dataset into 5 subsamples, and we split the modified dataset with robot occlusions into 5 subsamples. 4 subsamples of the original dataset and 4 subsamples of the modified dataset are used for training, and the remaining subsamples are used for validation.



Figure 5.4: **Benefits of Occlusion-Aware Planning:** The top row highlights the point cloud with the dynamic human obstacle, and the regions occluded by robot arms (in red). The bottom row highlights the trajectories computed by different planners when as the robot arm needs to move from right to left: (a) The trajectory is generated by the baseline planner, which does not account for occlusion. When the robot occludes the human, the motion prediction error is high and results in collisions. (b) The robot arm motion is generated by our occlusion-sensitive planner. The arm first moves to reduce the level of occlusion (i.e. a detour) and then reaches the goal to compute a safe trajectory.

We have tested our neural network models by enabling and disabling the input data channels related to the robot occlusion. These input channels are: *Occlusion Color*, *Occlusion Depth*, and *Skeleton*. *Occlusion Color* is the color image of the robot with a white background. *Occlusion Depth* is the depth image of the robot with a white background. *Skeleton* is the tracked human skeletal joint positions in 3D coordinates with respect to the camera coordinate system. The baseline planning algorithm only accepts the color and depth images and does not acquire information about robot occlusions. We created 7 different models or versions of planners by enabling the three input channels described above. HMPO accepts color image, depth image, color robot occlusion image, depth robot occlusion image, and the tracked human skeleton.

We measure the performance of our joint position prediction and action classification algorithms. Table 5.1 shows the performance of the future human joint position prediction for the different classification models. The average *error distance* is measured as follows:

$$d_{err}(t) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{h}_i(t) - \mathbf{h}_{truth,i}(t)\|, \quad (5.9)$$

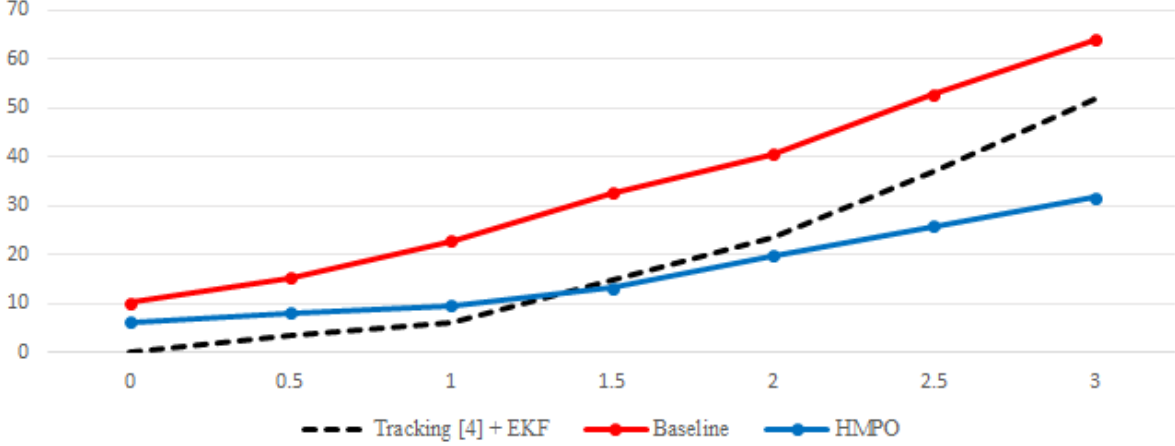


Figure 5.5: Average error distance over time for up to 3 seconds between ground truth joint positions and the predicted joint positions. The error distances for the skeleton tracking [4] and Extended Kalman Filter (EKF) are shown with a dashed line, and the error distances for our models are shown with solid lines. The baseline model without occlusion input images has higher error distances. However, the HMPO model has better prediction results with lower error distance values than EKF when the future prediction time is 1.5 seconds or higher.

where N is the number of human skeleton joints, $\mathbf{h}_i(t)$ is the predicted i -th human 3D joint position at time t , and $\mathbf{h}_{truth,i}(t)$ is the ground-truth human joint position. The human skeleton model-based joint tracking with particle filter [4] has an average error distance of 16.0 cm for tracking. An Extended Kalman Filter with linear motion of joint angles is used to predict the future joint positions. With the particle filter and the Extended Kalman Filter, the average prediction error is 34.0 cm, which is a significant increase over the average tracking error of 16.0 cm. When occlusion characteristics are added to the RGB-D images, the error distance increases to 51.6 cm. The error distance of HMPO in the *Occlusion* dataset is 31.8 cm. HMPO reduces the error distance dataset by 38% from the particle filter-based tracking [4] plus Extended Kalman Filter (51.6 cm) and 50% from the baseline (64.0 cm).

Table 5.2 highlights the performance of human action class prediction for different classification models. Wu et al. [3] highlighted 31.6% accuracy on action classification for the original Watch-n-patch dataset with 21 different types of human action classes. When robot occlusion is added to this dataset, human skeleton-based visual features cannot be extracted. This results in lower accuracy of classification (22.5%) for both the original action class labels and the *occluded* label. However, when more input channels containing information about occlusions are added to the baseline, the classification accuracy increases. We observe that *Occlusion Depth* and *Skeleton* inputs play a more

Error Distance (cm)	UTKinect [2]	Watch-n- Patch [3]	Occlusion MoCap [4]
Tracking [4] + EKF			51.6 (17.7)
Baseline	91.3 (26.8)	116 (28.4)	64.0 (16.7)
Occlusion Color	94.1 (20.4)	110 (22.9)	63.4 (14.5)
Occlusion Depth	83.1 (21.6)	105 (28.2)	41.0 (9.3)
Skeleton	79.9 (15.2)	96.8 (19.7)	38.6 (9.2)
Occlusion Color + Depth	72.9 (15.0)	91.4 (21.4)	35.4 (14.9)
Occlusion Color + Skeleton	70.9 (13.0)	82.7 (21.4)	34.0 (4.9)
Occlusion Depth + Skeleton	65.3 (12.1)	77.1 (22.7)	35.1 (4.0)
HMPO	61.9 (15.8)	76.8 (14.3)	31.8 (6.9)

Table 5.1: **Accuracy Comparison of Prediction Algorithms on Different Datasets:** Average error distance (lower is better) between ground truth joint positions and the predicted joint positions after 3 second for different datasets and algorithms. The numbers in parentheses are standard deviations. The baseline is based on tracking methods [4] along with extended Kalman filters on the skeleton-based human motion model. Our approach, HMPO (31.8 cm), reduces the error distance dataset by 38% from the particle filter-based tracking [4] plus Extended Kalman Filter (51.6 cm) and 50% from the baseline (64.0 cm). This demonstrates the accuracy benefits of our occlusion-aware planner.

significant role in terms of action classification for the *Occlusion* dataset than *Occlusion Color*. Overall, the accuracies of the *Occlusion Depth* and *Skeleton* for *Occlusion* datasets increase from the accuracy of the baseline (19.7%) by 4.7pp and 9.1pp, respectively. However, the accuracy of *Occlusion Color* decreases by 2.8pp from the baseline, though the occlusion color input channel contributes to an increase when combined with the occlusion depth or the skeleton input channels. The classification accuracy of HMPO is 36.6%. HMPO improves the action classification accuracy in the *Occlusion* dataset by 63% from Wu et al. [3] (22.5%) and 86% from the baseline (19.7%). This demonstrates the benefits of our approach.

5.5.2 Occlusion-aware Motion Planning

We use the Fetch robot with an RGB-D camera on its head and a 7-DOF robot arm. The environments are represented as point clouds of human and static objects from the RGB-D datasets. In addition, we add virtual tables and bookshelves to the environments, so that the robot can interact with them as static obstacles. The robot’s task is to move a simple object on the table or bookshelf to a goal location while avoiding collisions with static obstacles and the human (see Fig. 5.4). The initial and goal locations of the object are randomly set for each task. The moving task is repeated with randomized goal locations for our evaluations.

Accuracy (%)	Dataset
	Watch-n-Patch [3]
Wu et al. [3]	22.5
Baseline	19.7 (6.3)
Occlusion Color	16.9 (5.0)
Occlusion Depth	24.4 (5.2)
Skeleton	28.8 (6.1)
Occlusion Color + Depth	28.3 (4.3)
Occlusion Color + Skeleton	30.7 (7.1)
Occlusion Depth + Skeleton	31.0 (5.4)
HMPO	36.6 (4.1)

Table 5.2: Accuracy of action classification and human motion prediction algorithms for the Watch-n-Patch dataset (higher is better). The numbers in parentheses are standard deviations. HMPO (36.6%) improves the action classification accuracy in the *Occlusion* dataset by 63% from Wu et al. [3] (22.5%) and 86% from the baseline (19.7%).

The human joint positions occluded by the robot arm are set to zero (untracked) as they are used as inputs to the LSTM described in Section 5.3.1. Only the inferred future joint positions and the confidence values are used while computing the collision and occlusion cost functions in our planner. To evaluate the performance, robot motion trajectories are generated from a baseline planner without the robot occlusion cost functions (left) and from our occlusion-aware robot motion planner, which uses the robot occlusion cost function (right) in Figures 5.1 and 5.4, respectively. The baseline robot motion planner tends to generate trajectories that collide with the human when the robot arm occludes the human from the robot head camera in the input images. This demonstrates the benefits of our planner, as it is able to compute a collision-free path in a complex environment with occluded dynamic obstacles.

5.6 Conclusion and Limitations

We present a novel approach to generating safe and collision-free trajectories for a robot operating in close proximity with a human obstacle. In these scenarios, parts of the robot (e.g., the arms) can result in self-occlusion and reduce the accuracy of human motion prediction. We present two novel algorithms. The first of these is a deep learning-based method for human motion prediction in occluded scenarios that not only considers image features but also occlusion features for training and evaluation. We use three widely used datasets of human actions and augment them with synthetic occlusion information. Compared to prior classification algorithms, we observe up to 68%

improvement in motion prediction accuracy. Second, we present an occlusion-aware planner that considers the predicted trajectories and the confidence level. It directly computes a safe trajectory or moves the robot arms to reduce the extent of occlusion, thereby increasing the accuracy of human motion prediction for safe planning. We have highlighted the performance in complex scenarios where prior planners are unable to compute collision-free trajectories. Furthermore, we observe up to 38% improvement in terms of the error distance metric. To the best of our knowledge, this is the first general method for safe motion planning in occluded scenarios with human obstacles.

Our work has some limitations. Our augmented datasets with occlusion characteristics are synthesized from human-only action datasets. Those human actions were captured in an environment with no physical robots. The human actions in the real world in an environment shared with a robot may be different. The trajectories computed by our occlusion-aware planner may be less optimal because we may compute path detours while we first attempt to move the arms to reduce occlusion. Our overall planning algorithm uses an optimization framework with occlusion functions and is prone to local minima problems. Our motion prediction algorithm assumes that a good representation of the human skeleton can be computed from a given depth image. There are many avenues for future work. In addition to addressing the limitations, we would like to evaluate our approach in complex scenes with multiple humans, which can result in complex occlusion relationships.

CHAPTER 6

Conclusion and Future Work

We study four different algorithms for robot motion planning under uncertainty: collision probability computation (Chapter 2), natural language processing (Chapter 3), intention-aware motion planning (Chapter 4), and occlusion-aware motion planning (Chapter 5). We applied the algorithms to robot motion planning in simulated scenes and in real-world settings.

In Chapter 2, we study efficient collision probability computation algorithms for positional errors represented in some types of probability distributions: Truncated Gaussian, Weighted Samples, and Truncated Gaussian Mixture Model. Our novel probabilistic collision detection can compute a tight upper bound of collision probability efficiently. By applying divergence theorem to Gaussian distribution and cut off computations outside the truncation boundary, the collision probability between convex shapes is tightly bounded and efficiently computed. The collision probability for non-convex shapes and non-Gaussian probability distributions is computed efficiently by constructing BVHs of shapes and TGMM. Benefits of tight collision probability computation in robot motion planning is shown in a simulated narrow passage scenario.

In Chapter 3, we study a dynamic constraint coefficient mapping algorithm that translates from attribute-based natural language instructions to the coefficients in the optimization problem for the optimization-based robot motion planner. Attribute-based natural language commands are transformed into appropriate constraints for optimization-based motion planning using DGG, our Conditional Random Field model. Safe human-robot interaction was demonstrated on the optimization-based motion planning with probabilistic collision detection algorithms after

In Chapter 4, we study intention-aware robot motion planning and prediction algorithms for human actions and motions. By predicting future human action and motion using an SVM action classifier and a Gaussian Process regressor, the robot can make efficient subtask ordering plans from Q-learning and generate collision-free paths under uncertainty of human motions. Our improved human motion prediction model in robot motion planning can better handle input noise and can

generate smoother robot trajectories, compared to previous motion planning algorithms for dynamic environments.

In Chapter 5, we study occlusion-aware robot motion planning for robots with visual sensors and moving parts that occlude scenes. When a robot arm is located between the visual camera and a human, it makes an occlusion area in the input images. Visual features of the occluded input images and the occluded area are extracted from pre-trained CNNs. The features are fed to RNNs, which compute the human joint positions from the visible area, predict joint positions behind the occluded area, and provide the certainty about how the occluded area is affected, making predictions about the human motion accurate. The predicted human actions and motions are used in robot motion planning, moving the robot parts to remove occlusions when the human is behind the robot.

6.1 Limitations

The collision probability computation and robot motion planning algorithms under uncertainty still have some limitations. First, the collision probability computation algorithm is limited to specific positional forms of the probability density function. It assumes that the shapes of surrounding objects are known. However both positional uncertainty and uncertainties in shapes and orientations exist. Also, the humans are modeled with a simple human skeleton model enclosed by capsules.

Second, the human motions and natural language instructions are limited compared to possible human motions and natural language sets. The human actions are limited to 4-8 arm motions (Chapter 4) or to 12-15 whole body motions (Chapter 5). A factor graph in Chapter 3 for understanding natural language instructions is a good model for a small subset of natural language instructions. However, when the complexity of natural language instruction increases, the computation time grows exponentially as the number of possible groundings increases linearly. Also, the human motion dataset in Chapter 5 assumes that the human is not aware of the existence of the robot and performs actions independently from the robot.

Third, the multiple techniques cannot be easily integrated into a whole system. As more objective functions are added to the optimization problem, the computation time grows exponentially, and the quality of the robot motion trajectory becomes worse. There are two basic objective functions in optimization-based motion planning: smoothness and collision avoidance. These are already non-convex functions in the configuration space. With additional cost functions, such as end-effector orientation, end-effector speed, repulsion (Chapter 3), and occlusion cost (Chapter 5), the objective

function has too many local minima to compute a robot motion trajectory within a motion planning timestep. The motion trajectory solutions are likely to get stuck in a local minimum.

6.2 Future Work

In future work, we would like to develop an integrated robot motion planning system for collaborative robots that combines the submodules introduced in each chapter. By merging the objective functions for natural language understanding and intention- and occlusion-aware motion planning, the system can theoretically be integrated, although we have not run experiments in an environment where a human simultaneously gives verbal instructions and performs motions.

Also, in future work, we would like to develop an unsupervised learning for machine learning algorithms in each chapter. We have data from real-world settings for natural language instruction sets for natural language understanding, human joint position and pose prediction for intention-aware motion planning, and occluded RGBD images for occlusion-aware motion planning. Labeling data manually or semi-automatically is tedious and time-consuming. It would be useful to explore unsupervised learning method for motion prediction algorithms and use them with optimization-based planners.

In addition, we would like to run experiments with multiple robots and multiple humans collaborating. In addition to the interaction between a single robot and a single human, we also need to consider the interaction between the robots and the humans separately.

REFERENCES

- [1] J. S. Park, C. Park, and D. Manocha, “Efficient probabilistic collision detection for non-convex shapes,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 1944–1951, IEEE, 2017.
- [2] L. Xia, C. Chen, and J. Aggarwal, “View invariant human action recognition using histograms of 3d joints,” in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pp. 20–27, IEEE, 2012.
- [3] C. Wu, J. Zhang, S. Savarese, and A. Saxena, “Watch-n-patch: Unsupervised understanding of actions and relations,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4362–4370, 2015.
- [4] A. Dib and F. Charpillet, “Pose estimation for a partially observable human body from rgb-d cameras,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4915–4922, IEEE, 2015.
- [5] H. S. Koppula and A. Saxena, “Anticipating human activities using object affordances for reactive robotic response,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 38, no. 1, pp. 14–29, 2016.
- [6] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
- [7] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.
- [8] S. Haddadin and E. Croft, “Physical human–robot interaction,” in *Springer handbook of robotics*, pp. 1835–1874, Springer, 2016.
- [9] P. A. Lasota and J. A. Shah, “Analyzing the effects of human-aware motion planning on close-proximity human–robot collaboration,” *Human factors*, vol. 57, no. 1, pp. 21–33, 2015.
- [10] H. S. Koppula, A. Jain, and A. Saxena, “Anticipatory planning for human-robot teams,” in *Experimental Robotics*, pp. 453–470, Springer, 2016.
- [11] K. Khoshelham and S. O. Elberink, “Accuracy and resolution of kinect depth data for indoor mapping applications,” *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.
- [12] C. Park, J. Pan, and D. Manocha, “ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments,” in *Proceedings of International Conference on Automated Planning and Scheduling*, 2012.
- [13] M. C. Lin and D. Manocha, “Collision and proximity queries,” in *Handbook of Discrete and Computational Geometry: Collision detection*, pp. 787–808, CRC Press, 2003.
- [14] J. T. Klosowski, M. Held, J. S. Mitchell, H. Sowizral, and K. Zikan, “Efficient collision detection using bounding volume hierarchies of k-dops,” *IEEE transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.
- [15] R. B. Rusu, I. Alexandru, B. Gerkey, S. Chitta, M. Beetz, L. E. Kavraki, *et al.*, “Real-time perception-guided motion planning for a personal robot,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4245–4252, IEEE, 2009.

- [16] K.-H. Bae, D. Belton, and D. D. Lichti, “A closed-form expression of the positional uncertainty for 3d point clouds,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 4, pp. 577–590, 2009.
- [17] J. Pan, S. Chitta, and D. Manocha, “Probabilistic collision detection between noisy point clouds using robust classification,” in *International Symposium on Robotics Research (ISRR)*, 2011.
- [18] W. Xu, J. Pan, J. Wei, and J. M. Dolan, “Motion planning under uncertainty for on-road autonomous driving,” in *ICRA*, pp. 2507–2512, IEEE, 2014.
- [19] J. M. Wang, D. J. Fleet, and A. Hertzmann, “Gaussian process dynamical models for human motion,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 283–298, 2008.
- [20] T. Kollar, S. Tellex, M. R. Walter, A. Huang, A. Bachrach, S. Hemachandra, E. Brunskill, A. Banerjee, D. Roy, S. Teller, *et al.*, “Generalized grounding graphs: A probabilistic framework for understanding grounded language,” *JAIR*, 2013.
- [21] T. M. Howard, S. Tellex, and N. Roy, “A natural language planner interface for mobile manipulators,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 6652–6659, IEEE, 2014.
- [22] S. R. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay, “Reinforcement learning for mapping instructions to actions,” in *ACL-IJCNLP*, Association for Computational Linguistics, 2009.
- [23] C. Matuszek, D. Fox, and K. Koscher, “Following directions using statistical machine translation,” in *HRI*, IEEE, 2010.
- [24] D. Nyga, *Interpretation of Natural-language Robot Instructions: Probabilistic Knowledge Representation, Learning, and Reasoning*. PhD thesis, Universität Bremen, 2017.
- [25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- [26] H. Kataoka, Y. Miyashita, M. Hayashi, K. Iwata, and Y. Satoh, “Recognition of transitional action for short-term action prediction using discriminative temporal cnn feature,” in *BMVC*, 2016.
- [27] Q. Ke, M. Bennamoun, S. An, F. Boussaid, and F. Sohel, “Human interaction prediction using deep temporal features,” in *European Conference on Computer Vision*, pp. 403–414, Springer, 2016.
- [28] J. Butepage, M. J. Black, D. Kragic, and H. Kjellstrom, “Deep representation learning for human motion prediction and classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6158–6166, 2017.
- [29] C. Finn, I. Goodfellow, and S. Levine, “Unsupervised learning for physical interaction through video prediction,” in *Advances in neural information processing systems*, pp. 64–72, 2016.
- [30] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, “Recurrent network models for human dynamics,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4346–4354, 2015.

- [31] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, “Real-time human pose recognition in parts from single depth images,” *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.
- [32] B. Choo, M. Landau, M. DeVore, and P. A. Beling, “Statistical analysis-based error models for the microsoft kinecttm depth sensor,” *Sensors*, vol. 14, no. 9, pp. 17430–17450, 2014.
- [33] D. Vasquez, T. Fraichard, and C. Laugier, “Growing hidden markov models: An incremental tool for learning and predicting human and vehicle motion,” *The International Journal of Robotics Research*, 2009.
- [34] M. Field, D. Stirling, F. Naghdy, and Z. Pan, “Motion capture in robotics review,” in *2009 IEEE International Conference on Control and Automation*, pp. 1697–1702, IEEE, 2009.
- [35] A. Schick, “Hand-tracking for human-robot interaction with explicit occlusion handling,” 2008.
- [36] J. S. Park, C. Park, and D. Manocha, “Intention-aware motion planning using learning based human motion prediction,” in *Robotics: Science and Systems*, 2017.
- [37] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [38] A. Greß, M. Guthe, and R. Klein, “Gpu-based collision detection for deformable parameterized surfaces,” in *Computer Graphics Forum*, vol. 25, pp. 497–506, Wiley Online Library, 2006.
- [39] H. Kurniawati and V. Yadav, “An online pomdp solver for uncertainty planning in dynamic environment,” ISRR, 2013.
- [40] N. L. Johnson, S. Kotz, and N. Balakrishnan, “Lognormal distributions,” *Continuous univariate distributions*, vol. 1, pp. 207–227, 1994.
- [41] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.
- [42] S. Patil, J. Van Den Berg, and R. Alterovitz, “Estimating probability of collision for safe motion planning under gaussian motion and sensing uncertainty,” in *Proceedings of ICRA*, pp. 3238–3244, IEEE, 2012.
- [43] M. Rafieisakhaei, S. Chakravorty, and P. Kumar, “Non-gaussian slap: Simultaneous localization and planning under non-gaussian uncertainty in static and dynamic environments,” *arXiv preprint arXiv:1605.01776*, 2016.
- [44] K. M. Seiler, H. Kurniawati, and S. P. Singh, “An online and approximate solver for POMDPs with continuous action space,” in *Proceedings of ICRA*, pp. 2290–2297, IEEE, 2015.
- [45] N. E. Du Toit and J. W. Burdick, “Probabilistic collision checking with chance constraints,” *Robotics, IEEE Transactions on*, vol. 27, no. 4, pp. 809–815, 2011.
- [46] M. Althoff, O. Stursberg, and M. Buss, “Model-based probabilistic collision detection in autonomous driving,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 2, pp. 299–310, 2009.

- [47] A. Lambert, D. Gruyer, and G. S. Pierre, “A fast monte carlo algorithm for collision probability estimation,” in *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pp. 406–411, IEEE, 2008.
- [48] C. Park, J. S. Park, and D. Manocha, “Fast and bounded probabilistic collision detection in dynamic environments for high-dof trajectory planning,” *arXiv preprint arXiv:1607.04788*, 2016. To appear in *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.
- [49] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, “Intention-aware online pomdp planning for autonomous driving in a crowd,” in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 454–460, IEEE, 2015.
- [50] J. Van den Berg, D. Wilkie, S. J. Guy, M. Niethammer, and D. Manocha, “LQG-Obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 346–353, IEEE, 2012.
- [51] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, “Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns,” *Autonomous Robots*, vol. 35, no. 1, pp. 51–76, 2013.
- [52] J. F. Fisac, A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, S. Wang, C. J. Tomlin, and A. D. Dragan, “Probabilistically safe robot planning with confidence-based human predictions,” *arXiv preprint arXiv:1806.00109*, 2018.
- [53] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, “A fast procedure for computing the distance between complex objects in three-dimensional space,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.
- [54] S. Gottschalk, M. C. Lin, and D. Manocha, “Obbtrees: A hierarchical structure for rapid interference detection,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, ACM, 1996.
- [55] J. S. Park and D. Manocha, “Efficient probabilistic collision detection for non-gaussian noise distributions,” *arXiv preprint arXiv : 1902.10252*, 2019.
- [56] J. S. Park, C. Park, and D. Manocha, “I-planner: Intention-aware motion planning using learning-based human motion prediction,” *The International Journal of Robotics Research*, vol. 38, no. 1, pp. 23–39, 2019.
- [57] A. D. Dragan, K. C. Lee, and S. S. Srinivasa, “Legibility and predictability of robot motion,” in *HRI*, IEEE, 2013.
- [58] F. Duvallet, M. R. Walter, T. Howard, S. Hemachandra, J. Oh, S. Teller, N. Roy, and A. Stentz, “Inferring maps and behaviors from natural language instructions,” in *Experimental Robotics*, pp. 373–388, Springer, 2016.
- [59] S. Branavan, N. Kushman, T. Lei, and R. Barzilay, “Learning high-level planning from text,” in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pp. 126–135, Association for Computational Linguistics, 2012.

- [60] F. Duvallet, T. Kollar, and A. Stentz, “Imitation learning for natural language direction following through unknown environments,” in *ICRA*, pp. 1047–1053, IEEE, 2013.
- [61] R. Paul, J. Arkin, N. Roy, and T. M. Howard, “Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators,” in *Robotics: Science and Systems*, 2016.
- [62] J. Arkin and T. M. Howard, “Towards learning efficient models for natural language understanding of quantifiable spatial relationships,” in *RSS 2015 Workshop on Model Learning for Human-Robot Communication*, 2015.
- [63] I. Chung, O. Propp, M. R. Walter, and T. M. Howard, “On the performance of hierarchical distributed correspondence graphs for efficient symbol grounding of robot instructions,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 5247–5252, IEEE, 2015.
- [64] J. Oh, T. M. Howard, M. R. Walter, D. Barber, M. Zhu, S. Park, A. Suppe, L. Navarro-Serment, F. Duvallet, A. Boularias, *et al.*, “Integrated intelligence for human-robot teams,” in *International Symposium on Experimental Robotics*, pp. 309–322, Springer, 2016.
- [65] R. Scalise, S. Li, H. Admoni, S. Rosenthal, and S. S. Srinivasa, “Natural language instructions for human-robot collaborative manipulation,” *The International Journal of Robotics Research*, p. 0278364918760992, 2018.
- [66] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
- [67] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, “STOMP: Stochastic trajectory optimization for motion planning,” in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 4569–4574, 2011.
- [68] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, “Chomp: Covariant hamiltonian optimization for motion planning,” *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.
- [69] C. Park, J. Pan, and D. Manocha, “Real-time optimization-based planning in dynamic environments using GPUs,” in *Proceedings of IEEE International Conference on Robotics and Automation*, 2013.
- [70] D. Silver, J. A. Bagnell, and A. Stentz, “Learning autonomous driving styles and maneuvers from expert demonstration,” in *Experimental Robotics*, pp. 371–386, Springer, 2013.
- [71] S. Nikolaidis, P. Lasota, G. Rossano, C. Martinez, T. Fuhlbrigge, and J. Shah, “Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action,” in *Robotics (ISR), 2013 44th International Symposium on*, pp. 1–6, IEEE, 2013.
- [72] S. Bird, “Nltk: the natural language toolkit,” in *Proceedings of the COLING/ACL on Interactive presentation sessions*, pp. 69–72, Association for Computational Linguistics, 2006.
- [73] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *EMNLP*, pp. 1532–1543, 2014.

- [74] C. Sutton, A. McCallum, *et al.*, “An introduction to conditional random fields,” *Foundations and Trends® in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012.
- [75] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, “CHOMP: Covariant hamiltonian optimization for motion planning,” *International Journal of Robotics Research*, 2012.
- [76] D. Nyga and M. Beetz, “Everything robots always wanted to know about housework (but were afraid to ask),” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 243–250, IEEE, 2012.
- [77] D. Nyga and M. Beetz, “Cloud-based probabilistic knowledge services for instruction interpretation,” in *Robotics Research*, pp. 649–664, Springer, 2018.
- [78] D. Nyga, M. Picklum, S. Koralewski, and M. Beetz, “Instruction completion through instance-based learning and semantic analogical reasoning,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 4270–4277, IEEE, 2017.
- [79] D. Nyga, M. Picklum, and M. Beetz, “What no robot has seen before-probabilistic interpretation of natural-language object descriptions,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 4278–4285, IEEE, 2017.
- [80] A. Broad, J. Arkin, N. Ratliff, T. Howard, and B. Argall, “Real-time natural language corrections for assistive robotic manipulators,” *The International Journal of Robotics Research*, vol. 36, no. 5-7, pp. 684–698, 2017.
- [81] T. Bandyopadhyay, C. Z. Jie, D. Hsu, M. H. Ang Jr, D. Rus, and E. Frazzoli, “Intention-aware pedestrian avoidance,” in *Experimental Robotics*, pp. 963–977, Springer, 2013.
- [82] A. Bera, S. Kim, T. Randhavane, S. Pratapa, and D. Manocha, “Gmp-realtime pedestrian path prediction using global and local movement patterns,” *ICRA*, 2016.
- [83] V. V. Unhelkar, C. Pérez-D’Arpino, L. Stirling, and J. A. Shah, “Human-robot co-navigation using anticipatory indicators of human walking motion,” in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 6183–6190, IEEE, 2015.
- [84] C. Fulgenzi, A. Spalanzani, and C. Laugier, “Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid,” in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 1610–1616, IEEE, 2007.
- [85] J. van den Berg, S. Patil, J. Sewall, D. Manocha, and M. Lin, “Interactive navigation of multiple agents in crowded environments,” in *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pp. 139–147, ACM, 2008.
- [86] S. Kim, S. J. Guy, W. Liu, D. Wilkie, R. W. Lau, M. C. Lin, and D. Manocha, “Brvo: Predicting pedestrian trajectories using velocity-space reasoning,” *The International Journal of Robotics Research*, 2014.
- [87] M. Hofmann and D. M. Gavrila, “Multi-view 3d human pose estimation in complex environment,” *International journal of computer vision*, vol. 96, no. 1, pp. 103–124, 2012.
- [88] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, “Machine recognition of human activities: A survey,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 11, pp. 1473–1488, 2008.

- [89] C. H. Ek, P. H. Torr, and N. D. Lawrence, “Gaussian process latent variable models for human pose estimation,” in *Machine learning for multimodal interaction*, pp. 132–143, Springer, 2007.
- [90] R. Urtasun, D. J. Fleet, and P. Fua, “3d people tracking with gaussian process dynamical models,” in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 238–245, IEEE, 2006.
- [91] K. P. Hawkins, N. Vo, S. Bansal, and A. F. Bobick, “Probabilistic human action prediction and wait-sensitive planning for responsive human-robot collaboration,” in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, pp. 499–506, IEEE, 2013.
- [92] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning,” in *AAAI*, pp. 1433–1438, 2008.
- [93] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, “Feature-based prediction of trajectories for socially compliant navigation,” in *Robotics: science and systems*, Citeseer, 2012.
- [94] A. D. Dragan and S. S. Srinivasa, “A policy-blending formalism for shared control,” *The International Journal of Robotics Research*, vol. 32, no. 7, pp. 790–805, 2013.
- [95] L. Busoniu, R. Babuska, and B. De Schutter, “A comprehensive survey of multiagent reinforcement learning,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 2, pp. 156–172, 2008.
- [96] C. Pérez-D’Arpino and J. A. Shah, “Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6175–6182, IEEE, 2015.
- [97] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Simeon, “A human aware mobile robot motion planner,” *Robotics, IEEE Transactions on*, vol. 23, no. 5, pp. 874–883, 2007.
- [98] A. D. Dragan, S. Bauman, J. Forlizzi, and S. S. Srinivasa, “Effects of robot motion on human-robot collaboration,” in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pp. 51–58, ACM, 2015.
- [99] S. Petti and T. Fraichard, “Safe motion planning in dynamic environments,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2210–2215, 2005.
- [100] J. Mainprice and D. Berenson, “Human-robot collaborative manipulation planning using early prediction of human motion,” in *Intelligent Robots and Systems (IROS)*, pp. 299–306, IEEE, 2013.
- [101] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, “Motion planning under uncertainty for robotic tasks with long time horizons,” *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 308–323, 2011.
- [102] J. Van Den Berg, S. Patil, and R. Alterovitz, “Motion planning under uncertainty using iterative local optimization in belief space,” *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1263–1278, 2012.
- [103] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

- [104] PrimeSense Inc., *Prime Sensor TM NITE 1.3 Algorithms notes*, 2010. Last viewed 19-01-2011 15:34.
- [105] M. Müller, “Dynamic time warping,” *Information retrieval for music and motion*, pp. 69–84, 2007.
- [106] J. Zhu and T. Hastie, “Kernel logistic regression and the import vector machine,” *Journal of Computational and Graphical Statistics*, 2012.
- [107] A. McHutchon and C. E. Rasmussen, “Gaussian process training with input noise,” in *Advances in Neural Information Processing Systems*, pp. 1341–1349, 2011.
- [108] E. Snelson and Z. Ghahramani, “Sparse gaussian processes using pseudo-inputs,” in *Advances in neural information processing systems*, pp. 1257–1264, 2005.
- [109] C. Park, J. S. Park, and D. Manocha, “Fast and bounded probabilistic collision detection for high-dof robots in dynamic environments,” in *Workshop on Algorithmic Foundations of Robotics*, 2016.
- [110] J. S. Park, C. Park, and D. Manocha, “Efficient probabilistic collision detection for non-convex shapes,” in *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, 2017.
- [111] C. W. Groetsch, *The theory of Tikhonov regularization for Fredholm equations of the first kind*, vol. 105. Pitman Advanced Publishing Program, 1984.
- [112] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.
- [113] H. S. Koppula, R. Gupta, and A. Saxena, “Learning human activities and object affordances from rgb-d videos,” *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 951–970, 2013.
- [114] M.-P. Dubuisson and A. K. Jain, “A modified hausdorff distance for object matching,” in *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, vol. 1, pp. 566–568, IEEE, 1994.
- [115] H. Kurniawati, T. Bandyopadhyay, and N. M. Patrikalakis, “Global motion planning under uncertain motion, sensing, and environment map,” *Autonomous Robots*, vol. 33, no. 3, pp. 255–272, 2012.
- [116] J. S. Park, B. Jia, M. Bansal, and D. Manocha, “Generating realtime motion plans from complex natural language commands using dynamic grounding graphs,” *CoRR*, vol. abs/1707.02387, 2017.
- [117] V. V. Unhelkar, P. A. Lasota, Q. Tyroller, R.-D. Buhai, L. Marceau, B. Deml, and J. A. Shah, “Human-aware robotic assistant for collaborative assembly: Integrating human motion prediction with planning in time,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2394–2401, 2018.
- [118] P. A. Lasota and J. A. Shah, “A multiple-predictor approach to human motion prediction,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2300–2307, IEEE, 2017.

- [119] J. Mainprice, R. Hayne, and D. Berenson, “Goal set inverse optimal control and iterative re-planning for predicting human reaching motions in shared workspaces,” *arXiv preprint arXiv:1606.02111*, 2016.
- [120] J. K. Aggarwal and Q. Cai, “Human motion analysis: A review,” *Computer vision and image understanding*, vol. 73, no. 3, pp. 428–440, 1999.
- [121] I. A. Kakadiaris and D. Metaxas, “Model-based estimation of 3d human motion with occlusion based on active multi-viewpoint selection,” in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 81–87, IEEE, 1996.
- [122] G. T. Papadopoulos, A. Axenopoulos, and P. Daras, “Real-time skeleton-tracking-based human action recognition using kinect data,” in *International Conference on Multimedia Modeling*, pp. 473–483, Springer, 2014.
- [123] J. Martinez, M. J. Black, and J. Romero, “On human motion prediction using recurrent neural networks,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [124] Z. Hu, T. Han, P. Sun, J. Pan, and D. Manocha, “3-d deformable object manipulation using deep neural networks,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4255–4261, 2019.
- [125] A. D. Dragan and S. S. Srinivasa, *Formalizing assistive teleoperation*. MIT Press, July, 2012.
- [126] D. Koller, J. Weber, and J. Malik, “Robust multiple car tracking with occlusion reasoning,” in *European Conference on Computer Vision*, pp. 189–196, Springer, 1994.
- [127] X. Wang, T. X. Han, and S. Yan, “An hog-lbp human detector with partial occlusion handling,” in *2009 IEEE 12th international conference on computer vision*, pp. 32–39, IEEE, 2009.
- [128] F. Achilles, A.-E. Ichim, H. Coskun, F. Tombari, S. Noachtar, and N. Navab, “Patient mocap: Human pose estimation under blanket occlusion for hospital monitoring applications,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 491–499, Springer, 2016.
- [129] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei, “Towards viewpoint invariant 3d human pose estimation,” in *European Conference on Computer Vision*, pp. 160–177, Springer, 2016.
- [130] E. Martinez-Martin and A. P. Del Pobil, “Object detection and recognition for assistive robots: Experimentation and implementation,” *IEEE Robotics & Automation Magazine*, vol. 24, no. 3, pp. 123–138, 2017.
- [131] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser, “Semantic scene completion from a single depth image,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1746–1754, 2017.
- [132] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, “Learning from synthetic humans,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 109–117, 2017.