UNIVERSITAT POLITÈCNICA DE CATALUNYA

Ph.D. Program:

AUTOMATIC CONTROL, ROBOTICS AND COMPUTER VISION

Ph.D. Thesis

Physics-based Motion Planning for Grasping and Manipulation

Muhayy Ud Din

Thesis Advisors: Jan Rosell Gratacòs and Esteban Peña Pitarch

27 September 2018

Physics-based Motion Planning for Grasping and Manipulation

Submitted in partial fulfillment of the requirements for the degree of Ph.D. in

Automatic Control, Robotics and Computer Vision

Supervised by

Jan Rosell Gratacòs and Esteban Peña Pitarch

Institut d'Organització i Control de Sistemes Industrials

Universitat Politècnica de Catalunya

27 September 2018

This thesis is dedicated to my parents

Acknowledgments

All praise and glory be to Allah who is the greatest benefactor, and Whose help enabled me to complete this thesis.

I would like to thank my advisor Prof. Jan Rosell Gratacòs and Esteban Peña Pitarch for their guidance, motivation, and help to complete this thesis. I would also like to thank Prof. Lydia Kavraki, Dr. Mark Moll and Dr. Máximo A. Roa for their guidance during my research stays.

I would like thank my colleagues Ali Akbari and Mohammed Diab for their help and useful long discussions. I would also like to thank Prof. Raúl Suárez and all the members of Service of Industrial Robotics research group for their help. I would like to thank Leopold Palomo for making the implementation stuff easy.

This work is supported by the Generalitat de Catalunya through the grant FI-DGR 2014 and Ministry of Economy and Competitiveness Spain through the projects DPI2014-57757-R and DPI2016-80077-R.

Finally, special thanks to my father Syed Bashir-ud-din Ahmed, my mother Nasira Noor and the rest of my family, it was not possible to complete the thesis without their support.

Contents

Acknowledgments **vii**

Abstract **xix**

1 Introduction **1**
- 1.1 Contributions 3
- 1.2 Related work 4
  - 1.2.1 Sampling-based motion planning 4
  - 1.2.2 Physics-based motion planning 6
  - 1.2.3 Motion planning for grasping 7
  - 1.2.4 Motion planning under uncertainties 8
  - 1.2.5 Knowledge-oriented planning 9
- 1.3 Software tools for planning 10
- 1.4 Structure of thesis 11
- 1.5 Publication note 12
- 1.6 List of Publications 12

I Knowledge-oriented motion planning **15**

2 Exploring the use of knowledge to enhance physics-based motion planning **17**
- 2.1 Introduction 17
- 2.2 Workspace representation 18
- 2.3 Problem description and interaction constraints 19
- 2.4 Knowledge representation for physics-based planning 20
- 2.5 Planning algorithm 21
- 2.6 Validation 24
- 2.7 Benchmarking 25
  - 2.7.1 Benchmarking parameters 26
  - 2.7.2 Benchmarking setup 27
  - 2.7.3 Benchmarking results 28
  - 2.7.4 Discussion 32

2.8 Applications of the proposed approach . . . 32
2.9 Conclusions . . . 33

3 Motion planning using knowledge-based reasoning on robot-object interactions 35
3.1 Introduction . . . 35
3.2 Problem formulation . . . 36
3.3 Reasoning-based state transition model . . . 37
3.4 Knowledge representation . . . 39
3.4.1 Abstract knowledge . . . 40
3.4.2 Instantiated knowledge κ_t . . . 42
3.5 Knowledge inference and reasoning process . . . 43
3.5.1 Knowledge Inference Process . . . 43
3.5.2 Reasoning process . . . 44
3.6 The κ-*PMP* approach . . . 46
3.6.1 Proposed framework . . . 46
3.6.2 Algorithm . . . 47
3.6.3 Simulation setup . . . 48
3.7 Results and discussion . . . 52
3.7.1 Quantitative analysis . . . 54
3.7.2 Qualitative analysis . . . 55
3.8 Practical application of the proposed approach . . . 56
3.9 Conclusions . . . 58

4 Physics-based motion planning using temporal logic specification 59
4.1 Introduction . . . 59
4.2 Problem setup to handle temporal goals . . . 60
4.3 Linear temporal logic . . . 61
4.4 LTL semantics for motion trajectories . . . 61
4.5 Problem statement . . . 62
4.6 Framework for physic-based LTL motion planning . . . 63
4.6.1 Reasoning process . . . 64
4.6.2 Planning process . . . 67
4.7 Results and discussion . . . 68
4.8 Conclusions . . . 70

II Planning under uncertainty 71

5 Robust-RRT: A motion planning approach for grasping under uncertainty 73
5.1 Introduction . . . 73
5.2 Problem formulation . . . 74
5.3 Reasoning process . . . 75
5.4 Handling pose uncertainty . . . 76
5.4.1 State validity checker . . . 78
5.4.2 Control evaluation . . . 78

5.4.3 Uncertainty region update 78
5.5 Robust Rapidly-exploring Random Tree (R-RRT) 79
5.6 Results and discussion 81
5.6.1 Bias index 84
5.6.2 Robustness threshold 84
5.7 Conclusions 86

6 Probabilistic-KPIECE: A clutter grasping approach under uncertainties 87
6.1 Introduction 87
6.2 Problem setup 88
6.2.1 Modeling uncertainty 88
6.2.2 Problem statement 89
6.2.3 Solution overview 90
6.3 Probabilistic-KPIECE 91
6.3.1 Handling uncertainty propagation 93
6.3.2 Probabilistic motion sampler 94
6.3.3 Tree exploration strategy 95
6.4 Evaluation 96
6.4.1 Simulation setup 99
6.4.2 Benchmarking 99
6.4.3 Qualitative analysis 102
6.5 Conclusions 103

III Implementation, conclusions and future work 105

7 Implementation framework for knowledge-oriented physics-based planning 107
7.1 Introduction 107
7.2 Implementation framework 108
7.2.1 Dependencies 108
7.2.2 The Kautham Project 109
7.2.3 Modeling 110
7.2.4 Kautham-core 111
7.3 Implementation of physics-based planning 111
7.4 Implementation of knowledge representation and reasoning 113
7.5 Visualization 115
7.5.1 Implemented ROS Nodes 115
7.5.2 Manipulation node 116
7.5.3 Communication manager 116
7.6 Conclusions 116

8 Conclusions and future work 117
8.1 Conclusions 117
8.2 Future work 119

List of Figures

1.1 Flow of the thesis. 11

2.1 Different objects with their *mRegions* (the red cube represents a fixed object and hence have no associated *mRegion*). 18
2.2 State transition process for ontological physics-based planning. 19
2.3 Simulation results of the ontological physics-based motion planner using KPIECE. 23
2.4 Simulation results of the ontological physics-based motion planner using RRT. . . 23
2.5 Simulation results using KPIECE without using instantiated knowledge. 24
2.6 Configuration spaces with the tree structures of the planners used and the corresponding solution paths (in red): a) KPIECE using instantiated knowledge; b) RRT using instantiated knowledge; c) KPIECE without using instantiated knowledge. . 25
2.7 Simulation setup used for the benchmarking. The robot (green sphere) is depicted at the start configuration while the yellow circle represents the goal. 27
2.8 Sequence of snapshots of the execution using ontological physics-based motion planner. 28
2.9 Configuration space and solution path: each row corresponds to a scene, the columns in each row (left to right) represents the configuration space and solution path using KPIECE, RRT, SyCLoP-RRT, and SyCLoP-EST respectively. 29
2.10 Average planning time (10 runs) for each scene and the overall averaged planning time (three scenes). 30
2.11 Success rate of the planners for each scene and the overall averaged success rate (three scenes). 30
2.12 Average amount of action for each scene and the overall averaged amount of action (three scenes). 31
2.13 Average power consumed by the robot while moving along the solution path and the overall average (three scenes). 31
2.14 Smoothness measure for each scene and the overall average (three scenes). . . . 32

3.1 Configuration space: $\mathcal{C}_{\text{obs}}$ shown in black (collisions with fixed obstacles) and gray (collision with manipulatable obstacles), $\mathcal{C}_{\text{interaction}}$ in green and $\mathcal{C}_{\text{move}}$ in white. . 36
3.2 Flow of knowledge from the high-level abstract knowledge to the low-level instantiated knowledge. 38

3.3 State transition model for the κ-*PMP*. 39
3.4 OWL-based semantic knowledge taxonomy. https://sir.upc.edu/projects/ontologies. 40
3.5 Screen shot of the Protégé editor showing the semantic properties of a car-like object. 41
3.6 Sample and propagation using: (a) the simple physics-based motion planner; (b) κ-PMP. The control sampling range is selected based on the dynamics properties of the target object and the manipulation region. Blue samples represent the lower values of control range, whereas orange samples represent the higher values. Yellow samples represents the decrease in forces during the transition between free and contact motions. 44
3.7 Knowledge-oriented physics-based motion planning (κ-*PMP*) for power-efficient motion plan. 45
3.8 Planning scenes: (a) an holonomic mobile robot; (b) a car-like mobile robot; (c) a planar kinamatic chain. Video: https://sir.upc.edu/projects/kautham/videos/k-PMP1.mp4 49
3.9 State space projection onto the workspace for the scenes with the holonomic mobile robot (top), the car-like robot (middle) and the planar manipulator (bottom) using: (1) κ-*KPIECE*, (2) κ-*RRT* and (3) κ-*SyCLoP* planners. 50
3.10 Logarithmic plot of the power consumed while moving along the path for the holonomic mobile robot. 51
3.11 Logarithmic plot of the power consumed while moving along the path for the car-like robot. 51
3.12 Logarithmic plot of the power consumed while moving along the path for the planar manipulator. 51
3.13 Histogram of the average power consumed by the robots: (a) holonomic mobile robot, (b) car-like robot, (c) planar manipulator. 53
3.14 Histogram of the average planning time. Scene1, scene2 and scene3 correspond to the scenes illustrated in Fig. 3.8. 54
3.15 Histogram of the success rate. Scene1, scene2 and scene3 correspond to the scenes illustrated in Fig. 3.8 54
3.16 Execution of the computed motion plan by the ontological physics-based motion planner. It can be observed that the task fails because the manipulatable yellow object ends at the gripper, thus preventing the manipulator to grasp the target green object. 56
3.17 Snapshots of the dynamic interactions between the robot and the yellow object corresponding to the ontological physics-based motion planning (a) and to the κ-*PMP* planner (b), respectively. Videos: https://sir.upc.edu/projects/kautham/videos/k-PMP2.mp4 57

4.1 $tr(Traj)$ generates the word $[\pi_0, \pi_1, \pi_0, \pi_3, \pi_0, \pi_2, \pi_0]$. The trajectory satisfies the formula if the word finished at the accepting state of the automaton. 62
4.2 Framework for physics-based LTL motion planning. 63
4.3 Example of the simplification process, where ϕ and ψ are the actual and the simplified formulas respectively. Each $\mathcal{L}_{(i,j)}$ is a list with i and j representing the depth and the order in the parent list, respectively. 65

4.4 Example scenarios A: the goal is to visit the propositional regions $P_1, \cdots, P_4$, being the access to them obstructed by the blue boxes. B: visit all propositional regions in an ordered way, being two of them occupied by bodies. Video: https://sir.upc.edu/projects/kautham/videos/IFAC2017.mp4 68
4.5 Sequence of snapshots the execution of first scenario. 69
4.6 Sequence of snapshots the execution of second scenario. 70

5.1 OWL semantic knowledge taxonomy. 75
5.2 (a) Initial pose uncertainty of three objects; (b) Pose uncertainty of the target object assumed to fit within the gripper aperture; (c) Enlarging of the pose uncertainty of an object being displaced due to the results of interactions. 77
5.3 Graphical representation of the computation of the confidence index for the control leading from E_{near} to E_{new}, where valid resultant states are shown in green whereas invalid ones in red. 77
5.4 (a) The 7 degree-of-freedom KUKA robot; (b,c,d) Scenarios where the path to the target object (in green) is blocked by manipulatable objects (in blue) and fixed objects (in red). 81
5.5 Snapshots of four task executions with different object poses: the first three rows correspond to successful executions, whereas the last one corresponds to a failed execution. Video:https://sir.upc.edu/projects/kautham/videos/RAL-2.mp4 82
5.6 Success rate of the planner using different values of $\mathcal{B}$. 83
5.7 Success rate of the execution for different values of confidence index. 83
5.8 Execution on the real robot, by changing the pose of the object. The goal is to grasp the yellow cylinder. The path to the goal is obstructed with the white Video: https://sir.upc.edu/projects/ kautham/videos/RAL-1.mp4 85

6.1 Control selection process. The motions in green belong to the tree, motions in black are the candidate motions and the motions in blue are the particle motions that are used to compute the belief of the candidate motions. 90
6.2 Sequence of the snapshots of the execution with: a) YuMi; b) Kuka-LWR. Videos: https://goo.gl/RzLVi1 and https://goo.gl/socxhb 97
6.3 Sequence of the snapshots of the execution with TORO. 97
6.4 Sequence of snapshots of the execution on the real YuMi robot. Video: https://goo.gl/bHspjZ 98
6.6 Graph of average planning time of 30 runs by varying the clutterness. 98
6.7 Graph of average planning success rate of 30 runs by varying the clutterness. . . . 99
6.8 Graph of average planning time of 60 runs by varying the number of particle motions. 100
6.9 Graph of average planning success rate of 60 runs by varying the number of particle motions. 100
6.10 Histogram of average planning time of 60 runs by varying the cell size. 101
6.11 Sequence of the snapshot of the executions with the YuMi and the Kuka-LWR, for the comparison with the task and motion planning approach. Videos: https://goo.gl/ZorqCF and https://goo.gl/F4fZVL 102

7.1 Simulation framework for knowledge-oriented physics-based motion planning. . 108
7.2 (a) an example of a problem file. (b) an example of a control file. 109
7.3 (a) a part of the URDF file of the robot. (b) the URDF model of the table 110
7.4 Examples of Prolog predicates, (a) represents the predicate for the object properties, (b) describes the predicate for the object data properties. (c) describes the predicate for reasoning over the object types. 113
7.5 Visualization of the robot motion with the *Kautham-GUI* 114
7.6 Example scenes of the visualization of the triangular mesh view and the actual scene view for different robots using *DrawStuff* 115

List of Tables

6.1 Comparison of p-KPIECE with task and motion planning approaches. Obj. represents the number of objects used, TP and TPU represent the approaches presented in Srivastava et al. (2014) and Hadfield-Menell et al. (2015), respectively. pKP and pKPU represent the p-KPIECE with and without uncertainty, respectively. . . . 103

Abstract

This thesis develops a series of knowledge-oriented physics-based motion planning algorithms for grasping and manipulation in cluttered an uncertain environments. The main idea is to use high-level knowledge-based reasoning to define the manipulation constraints that define the way how robot should interact with the objects in the environment. These interactions are modeled by incorporating the physics-based model of rigid body dynamics in planning.

The first part of the thesis is focused on the techniques to integrate the knowledge with physics-based motion planning. The knowledge is represented in terms of ontologies, a prolog-based knowledge inference process is introduced that defines the manipulation constraints. These constraints are used in the state validation procedure of sampling-based kinodynamic motion planners. The state propagator of the motion planner is replaced by a physics-engine that takes care of the kinodynamic and physics-based constraints. To make the interaction human-like, a low-level physics-based reasoning process is introduced that dynamically varies the control bounds by evaluating the physical properties of the objects. As a result, power efficient motion plans are obtained. Furthermore, a framework has been presented to incorporate linear temporal logic within physics-based motion planning to handle complex temporal goals.

The second part of this thesis develops physics-based motion planning approaches to plan in cluttered and uncertain environments. The uncertainty is considered in *1)* objects' poses due to sensing and due to complex robot-object or object-object interactions; *2)* uncertainty in the contact dynamics (such as friction coefficient); *3)* uncertainty in robot controls. The solution is framed with sampling-based kinodynamic motion planners that solve the problem in open-loop, i.e., it considers uncertainty while planning and computes the solution in such a way that it successfully moves the robot from the start to the goal configuration even if there is uncertainty in the system.

To implement the above stated approaches, a knowledge-oriented physics-based motion planning tool is presented. It is developed by extending The Kautham Project, a C++ based tool for sampling-based motion planning. Finally, the current research challenges and future research directions to extend the above stated approaches are discussed.

Chapter 1

Introduction

Motion planning is an essential component to perform manipulation tasks either for mobile robots, industrial manipulators or humanoid robots. Human motions involve implicit manipulation actions performed by interacting with the environment, such as push actions executed to remove those objects that are blocking the access to the target. To incorporate this type of human-like motion capabilities in robots is a challenging task. It requires knowledge of the appropriate contact regions from where the robot can interact with the objects and an efficient way to evaluate the consequences of these interactions. The former requires the rich semantic description of the objects in the workspace. Whereas, the latter needs the replication of the real world behavior in simulation considering the environment dynamics to reproduce the behavior of the objects in the result of interactions. The class of motion planning techniques that cope with this is known as physics-based motion planning, and is the topic of this thesis.

Typically, motion planners used in manipulation planning (to compute the motions for pick and place) are geometric planners. These planners consider a simple geometric world where dynamics are not considered, e.g., to pick an object, it is attached to the gripper and moved with the robot in the simulation world as part of the gripper. Complex manipulation actions, such as pushing a heavy object, require the complex physics-based models to determine how the object will behave in the results of applied forces. Some approaches already considered the physics-based model in planning, but without taking into account how interactions should take place, i.e., from which part should the robot manipulate the object. This is a key issue and can be determined with the aid of knowledge-based reasoning processes.

This thesis develops motion planning strategies for manipulation. These strategies, on the one hand, incorporate the realistic physics-based models of the objects in the environment and, on the other hand, use knowledge-based reasoning to guide the motion planner toward human-like ways of manipulation by defining manipulation constraints. These constraints help the robot to determine how to interact with the objects, thus allowing to find feasible paths in an efficient way. This thesis addresses the following research problems:

- **Knowledge integration in physics-based planning:** To efficiently manipulate objects during planning it is required to efficiently determine the part of the objects from where the robot can interact. For instance, to move a car-like object, the robot should only interact from its front or rear side. Such constraints can be modeled using knowledge representations, as well as an efficient knowledge-based reasoning process to extract the interactions constraints, and the integration of this knowledge (about the manipulation world) with the planning.

- **Adaptive manipulation interactions:** Humans dynamically vary the interaction forces during manipulation of the objects, that mostly depend on the physical properties of the target object. For instance, we never exert the similar range of forces during the pushing of a table, the pushing of a plate on the table or when moving the hand freely. To incorporate such capabilities, robots require the consideration of physics-based reasoning process to evaluate the physical properties of the target object to dynamically vary the interaction forces.

- **Handling complex temporal goals:** To handle complex temporal goals such as *visit region A followed by region B and avoid region C* within the framework of physics-based planning requires the careful evaluation of the interactions. Interaction constraints and temporal constraints must be satisfied at the same time. The evaluation of the formula that describes the temporal goals is required before starting the planning process in order to analyze the compatibility of the temporal constraints with the interaction constraints.

- **Grasping in clutter and uncertain environments:** Grasping in situations where the robot needs to put the objects away in order to reach the target object is a challenging task, particularly if the position of the objects is not certain. The robot needs to pick the objects and place them elsewhere. In case of uncertainty, the robot may miss a grasp, which leads to a fail in the picking action. Alternatively, pushing actions can be useful in this situation, although pushing the objects using simple straight-line motions of the gripper may not lead to a solution. An efficient approach is required that pushes the objects away, not necessarily using straight motions of the gripper.

- **Handling uncertainties due to complex multi-body interactions:** Interactions involve various dynamic parameters such as the exact value of the friction coefficient at the contact point, the interacting force directions and the pressure distribution under the object supporting surface. To compute the exact physics-model of the object along with the precise values of dynamic parameters is difficult. The difference between the approximated and the exact values can be tackled by considering uncertainty in the dynamic parameters. Furthermore, once interaction takes place, the uncertainty in the initial state is propagated into the future states, which must be tackle in a robust way.

1.1 Contributions

The main contributions of the thesis are the following:

- *A knowledge-oriented physics-based motion planning approach.* This approach aims to integrate ontological knowledge within a physics-based planning framework. The physics-based model of the world is defined using a dynamic engine and it is used as state propagator within sampling-based kinodynamic motion planners. In order to determine the interaction constraints, a knowledge-based reasoning process is introduced that queries over the knowledge to extract the manipulation constraints (Muhayyuddin et al., 2015).

- *Benchmarking criteria for physics-based motion planning.* These criteria includes the dynamic measures such as power consumption while moving along the path, together with the computational measures such as planning time and success rate. The proposed criteria are used to evaluate the performance of various sampling-based kinodynamic motion planners within the framework of knowledge-oriented physics-based motion planners (Muhayyuddin et al., 2016).

- *A physics-based motion planning method equipped with a control sampling strategy that allows the search of a power-efficient motion plan.* The method partitions the configuration space into different regions as a function of whether the robot can either move freely or interact with manipulatable objects. In order to determine appropriate interaction forces, a physics-based reasoning process is introduced that analyzes the configuration space regions and the physical properties of the target object to update the control sampling ranges (Muhayyuddin et al., 2017a).

- *A framework to handle temporal goals within physics-based motion planning for mobile robots.* This framework uses knowledge-based reasoning to handle the temporal and interaction constraints. The feasibility of the task (defined in terms of a linear temporal logic formula) is evaluated and in case of incompatibility between temporal and manipulation constraints, possible simplification in the task is performed (Muhayyuddin et al., 2017b).

- *An approach for physics-based grasping under uncertainty.* The approach uses knowledge-based reasoning to split the workspace into regions to guide the planner and reason about the result of the dynamical interactions between rigid bodies. The proposed planner is a variant of RRT called robust-RRT that uses a region-biased state sampling strategy and a smart validity checker that takes into account uncertainty in the pose of the objects (Muhayyuddin et al., 2018a).

- *A physics-based motion planner that permits robot-object and object-object interactions.* The planner avoids an explicit high-level reasoning of the task by providing the motion planner the capacity to evaluate possible complex multi-body dynamical interactions. The approach is able to solve the problem in complex scenarios by considering uncertainty in the objects' pose and in the contact dynamics. The work enhances the control sampler and the tree exploration strategy of a kinodynamic motion planner called KPIECE. The enhanced algorithm is called probabilistic-KPIECE (Muhayyuddin et al., 2018b).

- *Open source software development.* To implement and test the above stated approaches, an open-source simulation tool for knowledge-oriented physics-based motion planning is proposed by extending The Kautham Project, a C++ based open-source simulation tool for motion planning. The proposed simulation tool provides a flexible way to incorporate the physics, knowledge and reasoning in planning process. Moreover, it provides ROS-based interface to handle the manipulation actions (such as push/pull) and an easy way to communicate with the real robots (Rosell et al., 2014; Muhayyuddin et al., 2017c).

1.2 Related work

Motion planning problems deal with computing collision-free trajectories from a given start to a goal state in the configuration space ($\mathcal{C}$), the set of all possible configurations of the robot (Lozano-Pérez, 1983). The geometrically accessible region of $\mathcal{C}$ is called $\mathcal{C}_{\text{free}}$ and the obstacle region is known as $\mathcal{C}_{\text{obs}}$. At an early stage, the focus of motion planning problems was on planning under geometric constraints, often referred as the piano-mover's problems. Several planning approaches such as, grid-based planning and potential field methods were used to solve this problem. The grid-based planning approaches map a grid of cells on $\mathcal{C}$, being those completely in C_{free} labeled as free cells. The robot is allowed to move along the neighboring grid cells provided that they are free cells C_{free} (LaValle, 2006). A graph is constructed to represent the grid and graph searching techniques (such as A^* and Dijkstra's algorithm) are used to compute the collision-free path from start to goal configurations. The potential field methods compute the collision-free trajectories by generating an artificial potential field in $\mathcal{C}$ to drive the robot, which is a point, towards the goal while avoiding obstacles. This is achieved by the goal generating an attractive potential, and the obstacles a repulsive one (Latombe, 1991). Within this scope, to avoid local minima problems, several potential functions are proposed such as harmonic potential functions (Kim and Khosla, 1991; Garrido et al., 2010) and probabilistic harmonic functions (Rosell and Iñiguez, 2005). All these classical methods required the explicit characterization of $\mathcal{C}_{\text{obs}}$ and do not scale well to high-dimensional configuration spaces.

1.2.1 Sampling-based motion planning

Sampling-based algorithms for motion planing such as Probabilistic RoadMaps (Kavraki et al., 1996) and the Rapidly-exploring Random Trees (Lavalle and Kuffner, 2001) are essentially different from classical motion planning approaches. These approaches do not require the explicit knowledge $\mathcal{C}$, and due to this reason the explicit representation of $\mathcal{C}_{\text{obs}}$ is not required. These algorithms connect collision-free configurations with either a graph or a tree to capture the connectivity of $\mathcal{C}_{\text{free}}$ and find a path along these data structures to connect the initial and the goal configurations. In addition to geometric constraints, planning for real robotic systems also requires to consider the dynamic constraints, which can be handled with sampling-based approaches.

Kinodynamic motion planning refers to the problems in which the motion of the robot must simultaneously satisfy the kinematic constraints (such as joint limits and obstacle avoidance) as well as some dynamic constraints (such as bounds on the applied forces, velocities and accelerations (Donald et al., 1993)). Tree-based planners are best suited to take into account kynodynamic constraints (Tsianos et al., 2007), since the dynamic equations are used to determine the resulting motions used to grow the tree. The general functionality of sampling-based kinodynamic planners is to search a state space S of higher dimensions that records the system's dynamics. The state of a robot for a configuration $q \in \mathcal{C}$ is defined as $s = (q, \dot{q})$. To determine a solution, the planning will be performed in state space, in a similar way as in $\mathcal{C}$. This section briefly reviews the existing most commonly used kinodynamic motion planners, that can be categorized into three groups: a) RRT and EST belong to the class of planning algorithms that sample the states; b) KPIECE belongs to the class that samples the motions or path segments; c) SyCLoP is an hybrid planner that splits the planning problem into a discrete and a continuous layer.

Rapidly exploring Random Trees (RRT):

It is a sampling-based kinodynamic motion planning algorithm (LaValle and Kuffner, 2001) that has the ability to efficiently explore high dimensional configuration spaces. The working mechanism of RRT-based algorithms is to randomly grow a tree rooted at the start state ($q_{start} \in C_{free}$), until it finds a sample at the goal state ($q_{goal} \in C_{free}$). The growth of the tree is based on two steps, *selection* and *propagation*. In the first step a sample is randomly selected (q_{rand}), and its nearest node in the tree is then searched (q_{near}). The second step applies, from q_{near}, random controls (that satisfy the constraints) during a certain amount of time. Among the configurations reached, the one nearest to q_{near} is selected as q_{new} and an edge from q_{near} to q_{new} is added to the tree. Using this procedure, all the paths on the tree will be feasible, i.e. by construction they will satisfy all the kinodynamic constraints.

Expansive-Spaces Tree planner (EST):

This approach constructs a tree-shaped road-map T in the *state*$\times$*time* space (Hsu et al., 1997, 2002). The idea is to select a milestone of T and from there randomly apply sampled controls for a certain amount of time. If the final state is in free-space, it will be added as a milestone in T. The selection of the milestone for expansion is done in a way that the resultant tree should neither be too dense nor sparse. This kinodynamic planner works in three steps: *milestone selection*, *control selection* and *endgame connection*. In the first step, a milestone m in T is selected with probability inversely proportional to the number of neighboring milestones of m. In the second step, controls are randomly sampled and applied from the selected milestone m. Since by moving under kinodynamic constraints it may not be possible to reach exactly the goal state, the *endgame connection* step is the final step that defines a region around the goal in such a way that any milestone within this region is considered as the goal state.

Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE):

This planner is particularly designed for complex dynamical systems (Şucan and Kavraki, 2012). KPIECE grows a tree of motions by applying randomly sampled controls for a randomly sampled time duration from a tree node selected as follows. The state space is projected onto a lower-dimensional space that is partitioned into cells in order to estimate the coverage. As a result of this projection, each motion will be part of a cell, being each cell classified as an interior or exterior cell depending on whether the neighboring cells are occupied or not. Then, the selection of the cell is performance based on the *importance* parameter that is computed based on: 1) the *coverage* (the cells that are less covered are preferred over the others); 2) the *selection* (the cells that have been selected less number of time are preferred); 3) the *neighbors* (the cells that have less neighbors are preferred); 4) the *selection time* (recently selected cells are preferred); 5) the *expansion* (easily expanded cells are preferred over the cells that expand slowly). The cell that has maximum importance will be chosen, and a node of one of the motions of the cell will be randomly selected. The process continues until the tree of motions reaches the goal region.

Synergistic Combination of Layers of Planning (SyCLoP):

This is a meta approach that considers motion planning as a search problem in a hybrid space (of a continuous and a discrete layer) for efficiently solving the problem under kinodynamical constraints (Plaku et al., 2010). The continuous layer is represented by the state space (that is explored by a sampling-based motion planner like RRT or EST) and the discrete layer is determined by the decomposition of the workspace. The decomposition is used to compute a cost parameter called *lead* that guides the motion planner towards the goal. SyCLoP works based on the following steps: *lead computation* and *region selection*. The lead is computed based on the coverage and the frequency of the selection. The former is obtained by the sampling-based motion planner (continuous layer) and the latter is computed by determining how many times a cell has been selected from discrete space. The selection of the region will be performed based on the available free volume of the region (high free volume regions are preferred for the exploration). The process continues until the planner finds a sample in the goal region. SyCLoP will be recalled SyCLoP-RRT or SyCLoP-EST based on the planner used in the continuous layer.

1.2.2 Physics-based motion planning

All the above stated kinodynamic motion planning strategies are focused on computing a collision-free trajectory from start to the goal state, i.e. interactions with the objects in the environment are forbidden. This constraint leads to the neglecting of the physics-based dynamic features such as friction between the objects and the ground, pressure distribution under the objects surfaces, gravitational effect over the objects in the environment and the interaction dynamics (such as direction of interaction force and momentum). If interactions between the robot and the objects are allowed, however, all these features should be considered. Therefore, physics-based motion planning has recently emerged as a step further towards physical realism. It simultaneously considers the kinodynamic constraints and physics-based constraints (such as friction and gravity), and can also incorporate the purposeful manipulation of objects by evaluating the dynamic interactions between rigid bodies simulated using the basic physics (rigid-body dynamics).

Physics-based planning approaches implicitly use a sampling-based kinodynamic planner, that is responsible for sampling the states and constructing the solution path, but using for the propagation step a rigid-body dynamic simulator (dynamic engine), such as Open Dynamic Engine (ODE-`http://www.ode.org/`) or Bullet Physics Library (`http://bulletphysics.org/wordpress/`). The dynamic engine models the dynamical world with all the physical properties and has the ability to simulate the properties of the physical interactions, such as force-based inter-body collision and momentum. The physics-based planner evaluates the results of the action after propagation, if it satisfies all the constraints then the action is selected, and discarded otherwise.

The complexity of the physics-based motion planning is very high due to the high-dimensional state space, large search space and highly constrained solution set. A few physics-based motion planning approaches have been proposed that addressed the above mentioned issues, such as the Behavioral Kinodynamic Balanced Growth Trees (BK-BGT) and the Behavioral Kinodynamic Rapidly-Exploring Random Trees (BK-RRT) proposed in (Zickler and Veloso, 2009) that use a strategy to reduce the search space based on a nondeterministic tactic modeled using a finite state machine, along with skills used to control the sampling. The propagation step is performed using PhysX (`https://developer.nvidia.com/physx-sdk`).

Within the framework of physics-based motion planning, approaches are proposed for problems like rearrangement planning (Haustein et al., 2015a; King et al., 2016), object placement (Cosgun et al., 2011), object sorting (Gupta et al., 2015) or bin picking (Mahler and Goldberg, 2017). In this thesis we developed knowledge-oriented physics based motion planning approaches for grasping in clutter and uncertain environments.

1.2.3 Motion planning for grasping

In the past decades, motion planning has been considered under deterministic conditions, such as the positions of the objects and the motion of the robots being precisely defined. Recently, the incorporation of uncertainties and probabilistic treatment of the planning problem has gained a lot of interest, with the objective to compute robust motion plans for real environments. In this line, planning the grasping motions in unstructured and uncertain environments is a challenging task, particularly when a collision-free trajectory from start to goal does not exist. To tackle this issue, two approaches are mainly followed: *task and motion planning* (Dantam et al., 2016; Lagriffoul et al., 2014; Srivastava et al., 2014; Hadfield-Menell et al., 2015; Stilman and Kuffner, 2005; Stilman et al., 2007) and *clutter grasping* (Dogar and Srinivasa, 2010, 2011; Dogar et al., 2012; Laskey et al., 2016). The former breaks down the problem into a repeated sequence of actions to remove the objects obstructing the path towards the target: select an object to move, select the grasping points such as in León et al. (2012) (or contact points to push) on the object, compute a collision-free trajectory for grasping (pushing), and find an appropriate position to place the grasped (pushed) object. Finally, a collision-free trajectory is computed to grasp the target object. However, these approaches are still object-centric; in each action (either push or grasp) the focus is on moving a particular object while avoiding the interaction with the rest of the environment. On the other hand, clutter grasping is based on planning motions such that interactions with the objects are allowed, in order to clear the path to grasp the target object.

The *task and motion planning* approaches work well for structured or semi-structured environments, but encounter difficulties in cluttered and uncertain environments. On the one hand, task and motion planning approaches require the detailed semantic description of the scene and the explicit reasoning about each robot-object interaction (to carefully move objects in clutter). On the other hand, uncertainty has to be also considered at task level regarding the action effects, which leads to computationally intensive methods that may fail in highly cluttered and uncertain environments, i.e., it may not be possible to find a robust sequence of actions to free the path towards the target object. These challenging issues, however, could be tackled more easily using *clutter grasping* approaches. These strategies free the path towards the target object by pushing the objects away without explicitly reasoning at task level about each interaction.

1.2.4 Motion planning under uncertainties

In a manipulation task, the main sources of uncertainty are the imprecise knowledge of the initial state of the system (sensing uncertainty), of the robot dynamics (motion uncertainty) and of the future state of the environment when interactions exist (environment uncertainty). Sensing uncertainty is usually tackled by introducing a stochastic parameter (with zero mean Gaussian distribution) in the robot model that is used for the step propagation of an RRT-based algorithm, as well as a Gaussian distribution around the measured initial state (Sánchez-Miralles and Sanz-Bobi, 2004; Luders et al., 2010; Bry and Roy, 2011; Van Den Berg et al., 2011; Pilania and Gupta, 2017). The approach presented in (Sánchez-Miralles and Sanz-Bobi, 2004) for instance describes the obstacle map using Gaussian distributions, and construct a road-map with minimum probability of collision. Motion uncertainty is considered in different ways. Some approaches (Bry and Roy, 2011) consider the uncertainty in the robot dynamics and in the initial state of the system as a zero mean Gaussian noise, and use a variant of RRT that, at each step, computes the distribution of the state in a way such that the robot keeps a safe distance from the obstacles. Others use a linear quadratic regulator to maximize the probability to reach the goal or to minimize an expected cost function (Van Den Berg et al., 2011), or rely on *Markov Decision Processes* (MDPs) to compute the global control policy over the environment to maximize the probability of success (Alterovitz et al., 2007). Uncertainty in the environment is considered in (Melchior and Simmons, 2007; Du Toit and Burdick, 2012), which treats the extension step of an RRT as a stochastic process, simulating multiple times and using clustering techniques to create nodes, thus finding inherently safer paths.

Regarding the uncertainty in grasping and manipulation, the tasks are mainly affected by the uncertainty in the pose of the objects. In this sense, several authors take it into account in planning to make approaches more robust, like the approach in (Berenson et al., 2011) that proposes the use of task space regions (TSRs), computed using object pose estimations, to define the goal regions for an RRT-based planner. Within the scope of RRT planners, a metric is introduced in (Johnson et al., 2016) to guide the search towards more convergent trajectories, which increases robustness, as demonstrated with a manipulator rearranging objects. Uncertainty in rearrangement problems is also tackled in (King et al., 2017) using a learned policy from user demonstrations, and with a similar idea in (Laskey et al., 2016) for grasping in clutter. Learning from demonstrations avoid assuming explicit knowledge of objects or dynamics models, although this has been done to include uncertainty while planning for clutter grasping, assuming straight-line motions and quasi-static push mechanics (Dogar and Srinivasa, 2010; Dogar et al., 2012).

1.2.5 Knowledge-oriented planning

Knowledge representation deals with the way to handle information for the computing system to perform complex tasks. This representation can be inspired by the human way of representing knowledge and reason over it to understand the complex situations. There are many ways of representing knowledge, the most dominant in the robotic domain is the ontological representation of knowledge. An ontology represents knowledge in terms of *classes*, *individuals* and *properties*. *Classes* are collections of entities that have common characteristics. *Individuals* define the particular instances of the classes. *Properties* define how the individuals of different classes can relate with each other.

Ontologies can be encoded using the Web Ontology Language (OWL) (Antoniou and van Harmelen, 2003). OWL is intended to collect and organize ontology-based knowledge on a world-wide accessible database in an XML-based file format in order that multiple systems can use and share such knowledge. Ontologies can be written using the *Protégé* editor (`http://protege.stanford.edu/`). *Protégé* is an open source platform providing an ontology editor to develop knowledge-based applications. It, moreover, can be used to portray the visualization of ontology in terms of showing relations of classes or individuals as a graph.

The reasoning predicates can be defined using logic-based languages such as Prolog and SPARQL. Several frameworks for knowledge representation and reasoning are proposed to cover the particular class of planing problems such as navigation (Lim et al., 2011) or manipulation (Tenorth and Beetz, 2009). These knowledge processing frameworks for robotic systems provide reasoning tools that work over ontology models.

Typically, the knowledge processing frameworks are used to facilitate the planning process such as by providing the understanding of the workspace and by providing the way of manipulation of the objects in the environment. For instance, (Akbari et al., 2016b, 2015) used knowledge-based reasoning to understand the environment and to guide the task planner for planning in an efficient way. Diab et al. (2017) proposed the knowledge processing framework for physics-based planning. In the similar line, Tosello et al. (2017) provides the cloud based representation of the knowledge in terms of ontologies for manipulation tasks. In order to provides the common vocabulary to share the knowledge among robots, it follows the standardized concepts for representing the knowledge provided by IEEE RAS Ontology for Robotics and Automation Working Group (Prestes et al., 2013). The use of knowledge in this thesis is to develop the human-like interaction capabilities for robotic systems.

1.3 Software tools for planning

Dynamic simulations play an important role in robotics. During grasping and manipulation, dynamic simulations are essential to describe the behavior of the objects under the influence of various forces such as: the contact forces, the gravitational forces, and the inertial forces. Simulation of dynamics is a challenging task due to the fact that robotic systems are nonlinear in nature and due to the difficulty in determining the exact values of the parameters involved.

Various methods have been proposed to model the dynamics of the objects and the robots, the most commonly used methods are the Newton-Euler formalism, and the Lagrangian formalism. The Newton-Euler formalism describes the equations of motion for the rigid bodies based on the forces that are acting on the system. Whereas, Lagrangian formalism develops the equations of motion in terms of energy (kinetic energy and potential energy) of the system. Using the different dynamic modeling approaches, various dynamic engines have been proposed. These engines provide the core dynamical functionalities, such as how the object will behave under the action of forces. The most commonly used dynamic engines are Bullet Physics Library and Open Dynamic Engine. They are covered below. To provide the dynamic environment for planning, different dynamic simulators have been proposed using these engines. The most commonly used dynamic simulators are: Virtual Robot Experimentation Platform (V-REP) (Freese et al., 2010), Gazebo (`http://gazebosim.org/`), and Webots (`http://www.cyberbotics.com/`). V-REP is an open-source dynamic environment for robotics that uses Bullet as dynamic engine. Gazebo (an open-source dynamic simulator) and Webots (a commercial one), both use ODE as dynamics engine.

Open Dynamic Engine is an efficient and stable rigid body dynamic simulator, developed in C++. It is the most popular dynamic engine in robotics (Evan et al., 2010). In ODE the equations of motion are driven through the Lagrangian formalism. It has its own implementation for the collision checking and provide the facility to check collisions for different shapes such as: convex hulls, spheres, cylinders, and triangular meshes.

To plan collision-free trajectories considering the dynamical properties of the robots and the objects, different simulation frameworks have been proposed such as: *Open Motion Planning Library (OMPL)* (Şucan and Kavraki, 2012), *MoveIt!* (Şucan and Chitta, 2013), *Robotic Library (RL)* (Andre, 2011), and *The Kautham Project* (Rosell et al., 2014). The softwares such as *Simox* (Vahrenkamp et al., 2013), OpenRAVE (Diankov and Kuffner, 2008), and *OpenGRASP* (León

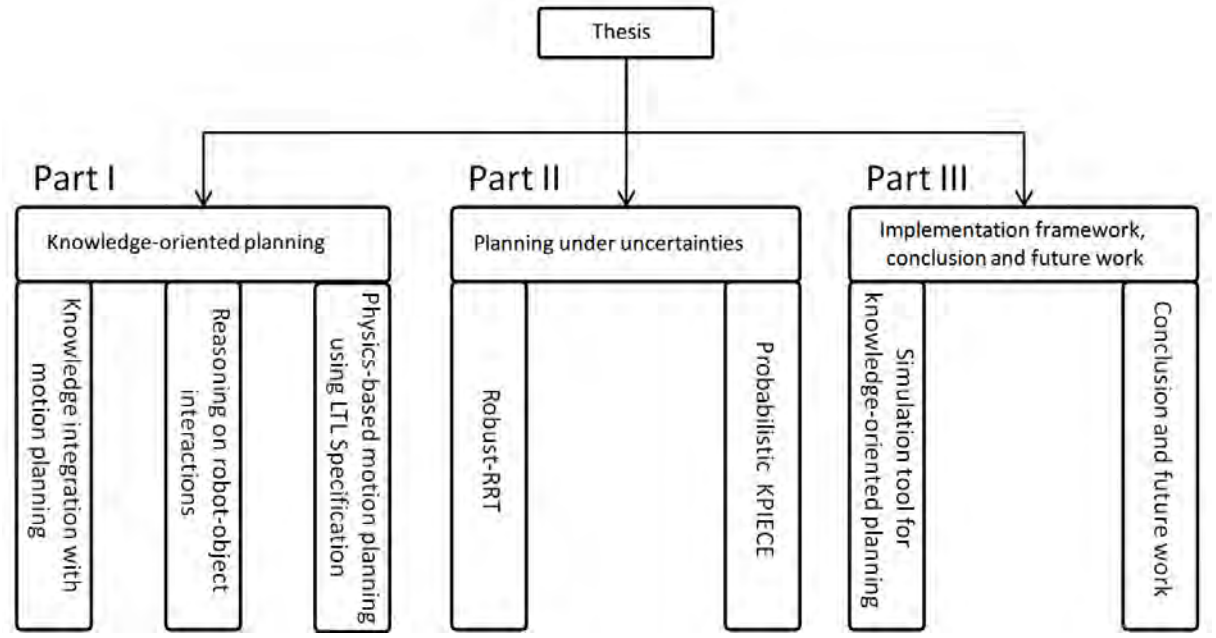

Thesis
Part I
Part II
Part III
Knowledge-oriented planning
Planning under uncertainties
Implementation framework, conclusion and future work
Knowledge integration with motion planning
Reasoning on robot-object interactions
Physics-based motion planning using LTL Specification
Robust-RRT
Probabilistic KPIECE
Simulation tool for knowledge-oriented planning
Conclusion and future work

Figure 1.1: Flow of the thesis.

et al., 2010) are able to perform multiple functionalities like motion and grasp planning.

Beside the core robotic software, various middle-ware frameworks (such as ROS Quigley et al. (2009) and OROCOS Bruyninckx et al. (2003)) are proposed to manage the communication between simulation and robot hardware. These middle-wares help greatly to simplify interprocess communications and synchronization issues.

To implement the approaches presented in this thesis, a simulation tool that simultaneously takes into account knowledge-based reasoning and dynamic of the environment in the planning process is required. The simulation tool with these capabilities is developed by extending The Kautham Project.

1.4 Structure of thesis

The rest of the thesis is structured in three parts, as depicted in Fig. 1.1. Part I is focused on knowledge-oriented planning, it consists of three chapters. Chapter 2 explains the integration of knowledge with physics-based motion planning and evaluates the performance of sampling-based kinodynamic motion planners within the proposed framework. Chapter 3 introduces the knowledge-based reasoning on robot-object dynamic interaction to interact with the objects in a human-like way. Chapter 4 presents a framework to handle the temporal goals within knowledge-oriented physics-based motion planning.

Part II of the thesis is focused on developing the approaches to handle uncertainties in motion planning, it consists of two chapters. Chapter 5 presents an approach to handle object pose uncertainty during the planning process. This approach extends the RRT by incorporating the process of robustness evaluation of the controls and strategies to handle the uncertainty propagation into the future states. Chapter 6 describes an approach to handle object pose uncertainty that arises due to the complex multi-body dynamical interactions while planning the motion in highly cluttered environments. This approach extends the KPIECE algorithm by developing a control sampling strategies and a tree-growing process to bias the tree growing towards the stats that are more robust.

Part III of the thesis presents the implementation framework for knowledge-oriented physics-based motion planning that is developed by extending The Kautham Project. The final chapter of the thesis presents the conclusions and the possible future direction of research.

It is important to note that each planner introduced in each chapter is explained in a quite complete manner. Some sections in different chapters, like those related to knowledge, may have similar descriptions, although they have their own particularities which are highlighted.

1.5 Publication note

The main part of chapter 2 appears in (Muhayyuddin et al., 2015) and (Muhayyuddin et al., 2016). Chapter 3 covers the approach mainly presented in (Muhayyuddin et al., 2017a). Chapter 4 describes the approach presented in (Muhayyuddin et al., 2017b). The approaches related to planning under uncertainties presented in chapter 5 and 6 appeared in (Muhayyuddin et al., 2018a) and (Muhayyuddin et al., 2018b), respectively. The implementation framework presented in chapter 7 appears in (Rosell et al., 2014) that is a joint work of the Service and Industrial Robotics research group at the Institute of Industrial and Control Engineering, Universitat Politècnica de Catalunya (IOC-UPC) and in (Muhayyuddin et al., 2017c).

Complementarity, some of the planners introduced in this thesis have been used the approaches that contain task and motion planning levels, like (Akbari et al., 2015), (Akbari et al., 2016c) and (Akbari et al., 2016a).

1.6 List of Publications

- Muhayyuddin, Moll, M., Kavraki, L., and Rosell, J. (2018). Randomized physics-based motion planning for grasping in cluttered and uncertain environments.In IEEE Robotics and Automation Letters, Pages 3(2):712-719.

- Muhayyuddin, Akbari, A., and Rosell, J. (2018). Knowledge-oriented physics-based motion planning for grasping under uncertainty. In ROBOT: Third Iberian Robotics Conference, pages 502-515. Springer International Publishing.
- Muhayyuddin, Akbari, A., and Rosell, J. (2018). A tool for knowledge-oriented physics-based motion planning and simulation. In EAI International Conference on Future Intelligent Vehicular Technologies. Springer International Publishing.
- Akbari, A., Muhayyuddin, Rosell, J. κ-TMP: Knowledge-oriented Task and Motion Planning for Multiple Mobile Robots. Accepted in: Journal of Experimental and Theoretical Artificial Intelligence.
- Muhayyuddin, Akbari, A., and Rosell, J. (2017). κ-PMP: Enhancing physics-based motion planners with knowledge-based reasoning. Journal of Intelligent Robotic Systems, pages 1-19. Springer International Publishing.
- Diab, M., Muhayyuddin, Akbari, A., Rosell, J. (2018). An ontology framework for physics-based manipulation planning. In ROBOT: Third Iberian Robotics conference, pages 452-464. Springer International Publishing.
- Muhayyuddin, Akbari, A., and Rosell, J. (2017). Physics-based motion planning with temporal logic specifications. IFAC-PapersOnLine, 50(1):8993-8999.
- Muhayyuddin, Akbari, A., and Rosell, J. (2016). Physics-based motion planning: Evaluation criteria and benchmarking. In ROBOT: Second Iberian Robotics Conference, pages 43-55. Springer.
- Akbari, A., Muhayyuddin, and Rosell, J. (2016). Reasoning-based evaluation of manipulation actions for efficient task planning. In ROBOT: Second Iberian Robotics Conference, pages 69-80. Springer.
- Akbari, A., Muhayyuddin, and Rosell, J. (2016). Task planning using physics-based heuristics on manipulation actions. In IEEE International Conference on Emerging Technologies Factory Automation (ETFA), pages 1-8.
- Akbari, A., Muhayyuddin, Rosell, J. (2016). Integrated Task and Motion Planning Using Physics-based Heuristics. RSS Workshop on Task and Motion Planning.
- Muhayyuddin, Akbari, A., and Rosell, J. (2015). Ontological physics-based motion planning for manipulation. In IEEE International Conference on Emerging Technologies Factory Automation (ETFA), pages 1-7.
- Akbari, A., Muhayyudin, and Rosell, J. (2015). Task and motion planning using physics-based reasoning. In IEEE International Conference on Emerging Technologies Factory Automation (ETFA), pages 1-7.
- Rosell, J., Alexander, P., Akbari, A., Muhayyuddin, Leopold, P., and Nétor, G. (2014). The kautham project: A teaching and research tool for robot motion planning. In IEEE international Conference on Emerging Technologies and Factory Automation (ETFA).

Part I

Knowledge-oriented motion planning

Chapter 2

Exploring the use of knowledge to enhance physics-based motion planning

2.1 Introduction

This chapter presents an approach to integrate knowledge with physics-based motion planning and new benchmarking criteria to evaluate different kinodynamic motion planners within the proposed framework. This work is presented in (Muhayyuddin et al., 2015) and (Muhayyuddin et al., 2016) research articles. Muhayyuddin et al. (2015) developed an approach where physics-based motion planning is enhanced with ontological knowledge and with an inference process that is used to reason on manipulation actions as well as on the geometry of the objects. This integration results in a robust and efficient way of planning complex motions. The approach is validated by a simple example in which the goal region is occupied by a car-like object and the mobile robot needs to push the object away in order to reach the goal. The results described show that the proposed algorithm has a better performance than the simple physics-based planning. Muhayyuddin et al. (2016) proposed a new benchmarking criterion to evaluate the performance of sampling based kinodyanamic motion planners within physics-based motion planning frameworks. This criterion takes into account the computational efficiency and dynamic measures (such as power consumed, action, smoothness) to determine the quality of the solution path. This proposed criterion is used to evaluate the performance of the ontological physics-based motion planner. The benchmarking setup is performed by setting several scenes with different level of complexity in such a way that the path to the goal is obstructed with moveable objects. In order to reach the goal the mobile robot has to clear the path by pushing them away. RRT, EST, KPIECE and SyCLoP-RRT and SyCLoP-EST are used for the benchmarking within the ontological physics-based planning framework.

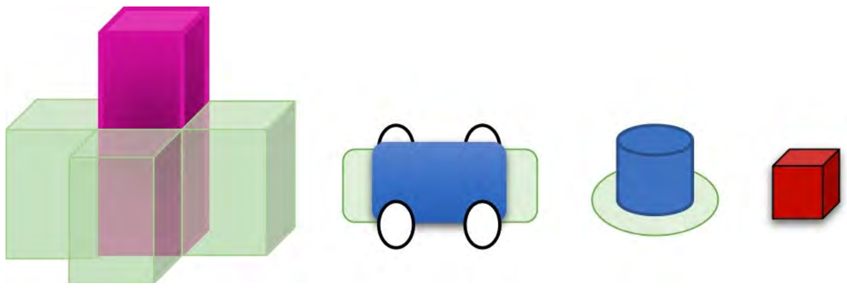


Figure 2.1: Different objects with their *mRegions* (the red cube represents a fixed object and hence have no associated *mRegion*).

2.2 Workspace representation

We consider a dynamic workspace with several objects (each one composed of one or more rigid bodies), that are categorized into *fixed objects* and *manipulatable objects*. *Fixed objects*, remain static during the entire planning process (such as walls) and no collision is allowed with them. *manipulatable objects*, on the contrary, can be pushed and hence their pose is not fixed. *Manipulatable objects* are further categorized into *constraint-oriented manipulatable objects* (*co-mObjects*) and *free-manipulatable objects* (*free-mObjects*), depending on whether they are subject to some kinematic constraints or not (e.g. a car-like object can only be pushed forwards or backwards in a single direction). All manipulatable objects have associated regions, called manipulation regions (*mRegions*), from where the robot can exert forces in order to move them, i.e. the robot can interact with the object through these regions and it is not allowed to contact the object from any other part. For instance, as shown in Fig. 2.1, a car-like robot has one *mRegion* located at its front and one at its rear, and a vertical box has the *mRegions* around it but below its center of mass.

The state s of each object is represented by its position r, orientation o, linear velocity v, angular velocity w, and the associated manipulation constraints η:

$$s = \{r, o, v, w, \eta\} \tag{2.1}$$

The state of a workspace composed of n objects at a given instant of time t is then represented as:

$$\mathbf{q}_t = \{s_1, s_2 \ldots, s_n, t\} \tag{2.2}$$

Also, in order to model the problem, the knowledge about the task and the workspace is organized in two levels: a *high level* knowledge representation called abstract knowledge K, and a *low level* knowledge representation called *instantiated knowledge* κ_t. The former codes the semantic-based knowledge about the world, such as the type of objects (i.e. fixed and manipulatable) and other features like mass or the manipulation constraints in case of *co-mObjects*.

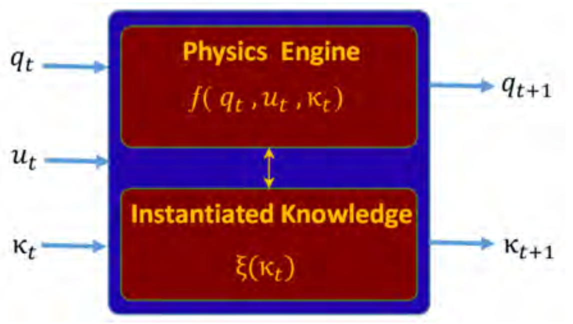


Figure 2.2: State transition process for ontological physics-based planning.

This knowledge remains unchanged throughout the whole planning process. The latter, on the other hand, is a dynamic knowledge, i.e. the instantiated knowledge infers from the abstract knowledge and is continually updated by making use of the reasoning process. For instance, if at a given instance of time the car-like body has its front manipulatable region overlapped by another manipulatable object, then this region cannot be accessed by the robot and the body cannot be pushed backward. In this case the front manipulatable region is tagged as non-valid in the instantiated knowledge.

Then, the state of the environment at any instant of time can be described as the tuple $(\kappa_t, \mathbf{q}_t)$, with κ_t being the instantiated knowledge about the world at time t and $\mathbf{q}_t$ the state of the workspace represented by Eq. (2.2). This state evolves with the state transition process depicted in Fig. 2.2, that is composed of two modules, the physics engine module and the instantiated knowledge module. Given the current state $(\kappa_t, \mathbf{q}_t)$ and the controls $\mathbf{u}_t$ to be applied, the physics engine module implements the dynamic state propagator that computes $\mathbf{q}_{t+1}$ and the instantiated knowledge module is then used to update the current instantiated knowledge:

$$\mathbf{q}_{t+1} = f(\mathbf{q}_t, \mathbf{u}_t, \kappa_t) \tag{2.3}$$

$$\kappa_{t+1} = \xi(\kappa_t) \tag{2.4}$$

This state transition process will be at the core of the physics-based planners to be used, and will only return the state whenever it is valid, i.e. the application of a control $\mathbf{u}_t$ that produces a non-allowable collision will not be accepted (collisions are not allowed with fixed objects nor with *co-mObjects* if the robot is not located at one of the *mRegions*).

2.3 Problem description and interaction constraints

Consider a motion planning problem where a robot must move in an environment with obstacles that can be fixed or manipulatable (i.e. obstacles that can be pushed away). Let Q be the state space of the robot, $q_{init} \in Q$ the initial state and $Q_{goal} \in Q$ the goal region. The robot must move from q_{init} to $q_{goal} \in Q_{goal}$ avoiding collisions with fixed obstacles and, if necessary, push away manipulatable obstacles to clear the way to find a solution path (for instance a manipulatable obstacle has to be removed if it is placed at the goal region thus preventing a collision-free path to exist).

The problem will be solved with a physics-based motion planner enhanced with a knowledge-based reasoning process that, on the one hand, modifies how manipulatable objects behave, and on the other modifies some parameters of the planner. The key points regarding κ_t and the reasoning process are:

- All *fixed objects* are set as non-collisionable (the robot is not allowed to collide with them).
- All *free-mObjects* are set as collisionable.
- Any *co-mObject* is set as collisionable if the robot is located at one of its *mRegions*, otherwise it is set as non-collisionable.
- If the *mRegion* of a *co-mObject* is occupied, then this region is set as non-valid.
- If a *co-mObject* result with no valid *mRegions*, then the object is set as fixed.
- If the state transition process results with a non-allowed collision, then the new generated state is discarded by the planner.
- If the goal region is occupied by a *co-mObject* and the planner used has a sampling bias, then this sampling-bias is randomly set to one of the *mRegions* of the body occupying the goal region, with the aim of pushing it away.

2.4 Knowledge representation for physics-based planning

For representing a manipulation problem, it is required to specify actions, objects with their properties, initial and goal states of a robot, and valid regions. Six classes, derived from a general *"Manipulation"* class have been defined:

- Class *"InitialState"* contains the initial location of the robot.
- Class *"GoalState"* contains the goal location of the robot.
- Class *"Actions"* contains different manipulation actions: PickAction, PlaceAction, and PushAction, although only the latter one will be used in this thesis.
- Class *"Regions"* defines different types of regions: *mRegion*, *ObjectRegion*, and *GoalRegion*. The *mRegions* are the regions around the constrained-oriented manipulatable objects from where the interaction between the robot and the objects are allowed (these regions are defined as boxes), i.e. the robot can only interact with the objects through the *mRegions*. *ObjectRegion* is the bounding box of an object. The *GoalRegion* is a circular region defined around the goal state.

- Class *"ObjectsType"* collects the types of objects in the world: *Manipulatable Object* or *Fixed Object*. If necessary, the manipulatable status of an object can be temporary changed to fixed by using the assertion process.
- Class *"ObjectsElements"* describes elements and features of objects.

Robots can be made more autonomous to carry out their complex tasks if some kind of reasoning process upon knowledge is provided, instead of having a purely symbolic knowledge representation. Reasoning on manipulation actions or on robot geometries provides the possibility of understanding the tasks in an analogous way a humans do, e.g. of being able to answer questions like *"Which specific actions should be selected for a given type of object?*", or *"Do I need to move some manipulatable objects to reach the goal?*". Two types of reasoning are proposed: reasoning about the manipulation actions and about the geometry of the objects. The aim of the reasoning about the manipulation actions is to guide the motion planner to find an appropriate action in accordance with the object type. As introduced before, objects are classified into *fixed* and *manipulatable*, and those manipulatable are divided into *constraint-oriented manipulatable* and *free manipulatable*. The type of object can be predefined, although it can be identified by a reasoning process based on the parts and features of the object and on the constraints they define, since these constraints may restrict the movement of the object. For instance, if when defining a car-like object the body is attached with a wheel-drive, the allowable directions of motion are automatically constrained along the plane of the wheel and the pushing manipulation actions can then only be exerted from some parts of the object (the front and the rear). Also, some parameters like the size and the weight of objects may constrain which actions can be selected according to the capabilities of a robot.

The reasoning about the geometry of objects tackles two matters. First, it determines whether the goal region is free or occupied by other objects (using the bounding boxes of the objects and of the goal region). Second, it determines possible manipulation regions on the object (from where to apply pushing forces). To address this, a reasoning process is done by inferring over the properties of the objects.

2.5 Planning algorithm

The ontological physics-based planning procedure is sketched in Algorithm 1. It takes as input the initial configuration $\boldsymbol{q}_{init}$, the goal region $\boldsymbol{Q}_{goal}$ and a time limit T_{max}, and returns a path from $\boldsymbol{q}_{init}$ to $\boldsymbol{q}_{goal} \in \boldsymbol{Q}_{goal}$, if found, or NULL otherwise. The planning is performed by any of the OMPL control planners, such as RRT or KPIECE, tuned to use ODE as state propagator. The instantiated knowledge is updated at each iteration and conditions the behavior of the bodies in the ODE world.

The steps of the algorithm are the following:

- *ReadTheWorld*: Sets the initial state of the robot and the environment by reading the p, o,

Algorithm 1: Ontological Physics-based Motion Planner

```
inputs : Initial state $\boldsymbol{q}_{\text{init}}$, Goal region $\boldsymbol{Q}_{\text{goal}} \in \mathcal{C}$, Threshold $T_{max}$
output: A path from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q} \in \boldsymbol{Q}_{\text{goal}}$
1  ReadTheWorld()
2  $\mathcal{K} \leftarrow$ FormulateOntology()
3  while $t < T_{max}$ do
4      $\kappa_t \leftarrow$ KnowledgeReasoner($\mathcal{K}, \kappa_{t-1}$)
5      SelectNodeToExpand()
6      $u \leftarrow$ SampleControls()
7      $\boldsymbol{q}_{\text{new}} \leftarrow$ Propagate($u$,$\kappa_t$)
8      if Valid( $\boldsymbol{q}_{new}$) then
9          return UpdateConnections()
10     if $\boldsymbol{q}_{new} \in \boldsymbol{Q}_{goal}$ then
11         return Path($\boldsymbol{q}_{new}$)
12 return NULL
```

v, w, and η of each body.

- *FormulateOntology*: Extracts the abstract knowledge K about the world, such as the type of the bodies, the manipulation constraints and the geometrical positions of the bodies, and determines whether the GoalRegion is occupied or not. Further it computes the *mRegions* for each *co-mObject*. All these attributes are stored in the form of abstract knowledge.

- *KnowledgeReasoner*: Evaluates the state of the world and updates the instantiated knowledge κ_t. For instance, if the GoalRegion is occupied by an object, then the sampling of the planner will be biased towards one of the *mRegion* of the object occupying it (instead of the standard bias towards the goal). When the robot enters the *mRegion*, the standard bias is reset and the instantiated knowledge is updated with the manipulation constraints of that object.

- *SelectNodeToExpand*: Selects the node to expand the tree data structure of the planner.

- *SampleControls*: Samples the controls within the given range.

- *Propagate*: Applies the controls to the bodies (controls can be applied by applying a force, a torque or by setting a linear and angular velocity), while applying the controls, the state propagator will take into account the manipulation constraints provided by the instantiated knowledge, and returns the new generated state.

- *Valid*: Returns true if no forbidden collision has occurred during the generation of the new state, and false otherwise.

- *UpdateConnections*: Updates the tree data structure by adding the edge between the previous and the newly generated state.

- *Path(q)*: Returns a path from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q} \in \boldsymbol{Q}_{\text{goal}}$ by backtracking along the planner data strcuture.

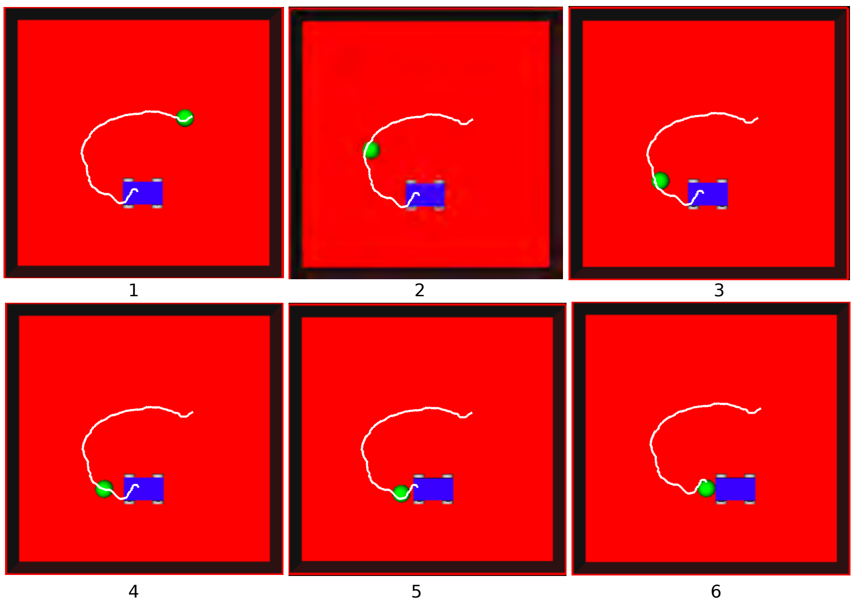


Figure 2.3: Simulation results of the ontological physics-based motion planner using KPIECE.

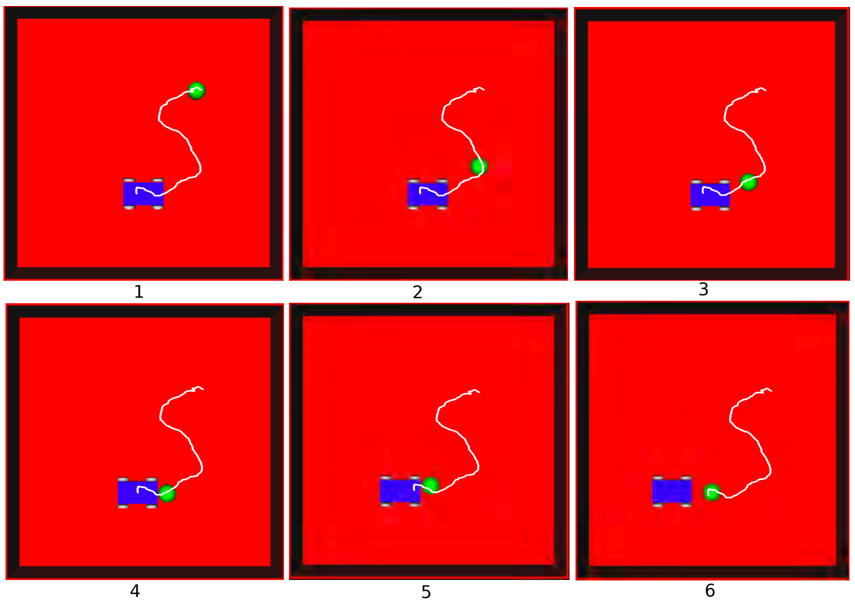


Figure 2.4: Simulation results of the ontological physics-based motion planner using RRT.

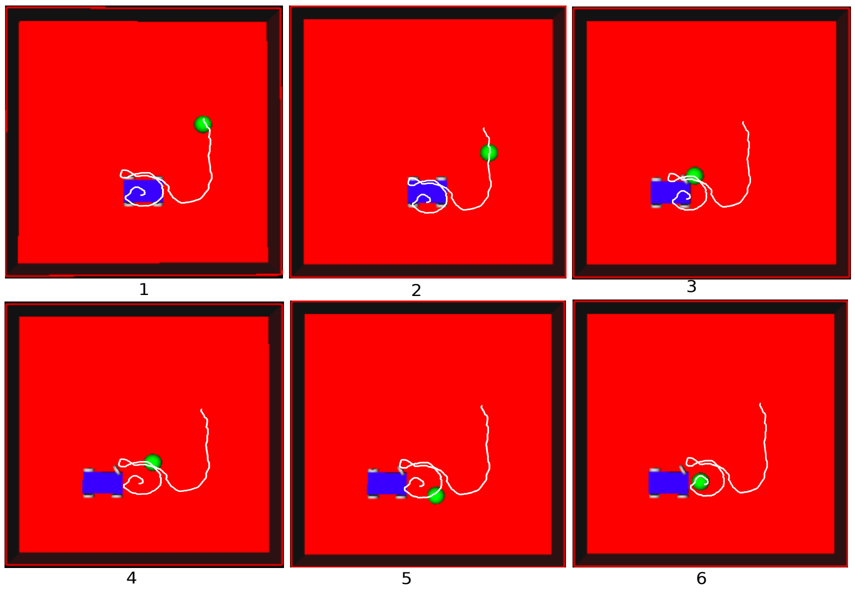


Figure 2.5: Simulation results using KPIECE without using instantiated knowledge.

2.6 Validation

To validate the proposed approach, a simple scenario has been set in order to put the focus on how the planner makes use of the knowledge to manage a complex situation involving a manipulatable object. It consists of: a) the walls of a room which are defined as fixed bodies; b) a spherical shaped robot with two degrees of freedom; and c) a car-like obstacle which is defined as a *co-mObject* that can only be moved in the forward and backward directions. The goal region $\boldsymbol{Q}_{goal}$ is occupied by the car-like obstacle, so no collision-free path exists, i.e. the robot needs to push the car-like obstacle away in order to reach $\boldsymbol{Q}_{goal}$.

The knowledge-based reasoning engine reads the initial state of the world and extracts the abstract knowledge related to the world such as the type of objects, the manipulation constraints, and the state of the goal. Based on a reasoning process, the instantiated knowledge is inferred from the abstract knowledge and fills the data structures in the motion planning layer (such as the current manipulation constraints, the valid manipulation regions, the goal state - occupied or free -, etc.), that are periodically updated. In this example, the knowledge-based reasoning engine determines that the goal is occupied and extracts the manipulation constraints associated to the car-like obstacle (they have been defined as two ManipulationRegions corresponding to its front and rear parts, since contact is allowed with the front and the rear sides and is forbidden with the two lateral sides).

The planning of paths has been performed using two different physics-based kinodynamic motion planners, KPIECE and RRT. At each instance of time, the instantiated knowledge evaluates the state of the world and the manipulation constraints, changing the standard sampling

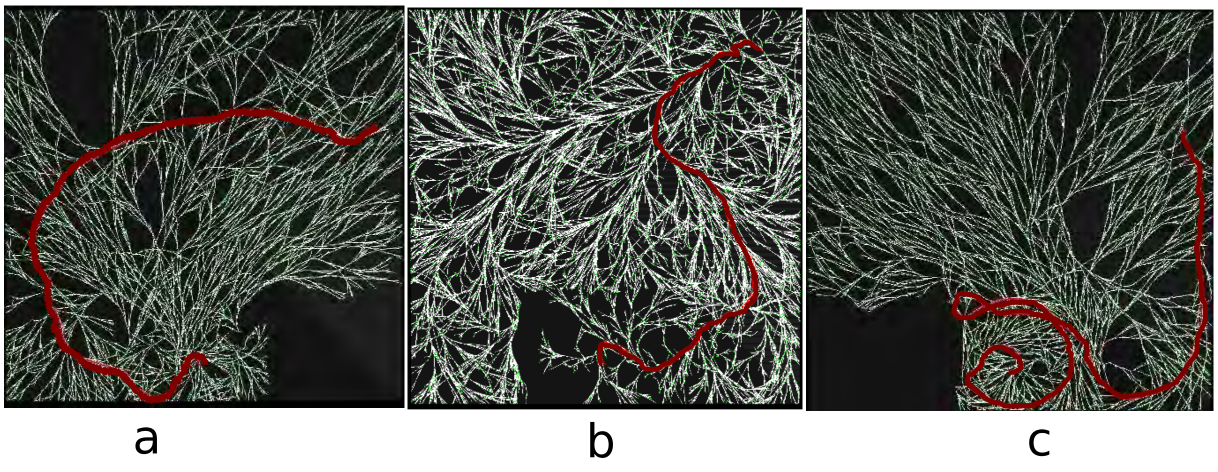


Figure 2.6: Configuration spaces with the tree structures of the planners used and the corresponding solution paths (in red): a) KPIECE using instantiated knowledge; b) RRT using instantiated knowledge; c) KPIECE without using instantiated knowledge.

bias towards the goal by a bias towards one of the ManipulationRegions of the car-like obstacle that occupies the GoalRegion. Once the robot reaches the ManipulationRegion, the standard bias towards the goal is restored. Fig. 2.3 and 2.4 show, respectively, sequences of snapshots of the solutions found using KPIECE and RRT. In both cases, it can be appreciated how the robot moves towards one of the ManipulationRegions and then hits the car-like obstacle at its front/rear side and pushes it to free the GoalRegion and reach the goal.

Without using the aid provided by the instantiated knowledge, these planners were not able to find good solutions, as illustrated in Fig. 2.5. Since in this case the planner did not know how to manipulate the car-like object, many tree edges tried unfruitfully to grow towards the goal by hitting the car at its side (in the solution shown the robot hits the car at the wheel and bounces back). Therefore, the computational cost was greater (it took approximately twice the time used by the ontological physics-based planners, either with KPIECE or RRT). The configuration spaces for all the above stated executions are shown in Fig. 2.6.

2.7 Benchmarking

The complexity of sampling-based kinodynamic motion planners is usually measured in terms of planning time. Since physics-based planning simultaneously evaluates the kinodynamical and the physics-based constraints, the evaluation based on just planning time may not be sufficient. New evaluation criteria are required because a number of other dynamical parameters (such as power consumed, action, smoothness) may significantly influence the planning decisions, such as in the the task planning approach proposed by Akbari et al. (2015) that uses a physics-based reasoning process to determine the feasibility of a plan by evaluating the cost in terms of power consumed and the action. With this in mind, a new benchmarking criteria for the above stated physics-based planner is presented.

2.7.1 Benchmarking parameters

A wide variety of the kinodynamic motion planners is available, the planning strategy of these algorithms is conceptually different from one another, and this can significantly effect the performance of the physics-based planning. A criterion is proposed to evaluate the performance of the kinodynamic motion planners for the physics-based planning. It is suggested that, along with the computational complexity, it is also necessary to evaluate the dynamical parameters that determine the quality of the computed solution path. This is determined by estimating the power consumed by the robot while moving along the computed path, by the total amount of action of the computed trajectory and by the smoothness of the trajectory. The computational complexity is computed based on the planning time and the average success rate of each planner. A planner is said to be the most appropriate if it is optimal according to the above said criteria.

The choice of a particular physics engine (such as ODE, Bullet and PhysX) may not affect the simulation results because the design philosophy of all of them is based on the basic physics, and the performance and accuracy may vary a little, and the simulation results of all the physics engines are almost the same.

The following performance parameters have been established to evaluate different kinodynamic motion planners within the framework of the ontological physics-based motion planner. The trajectories given by the kinodynamic motion planners are described by a list of forces and their duration that have to be consecutively applied to move the robot (either in a collision-free way or possibly pushing some manipulatable objects):

- ***Action:*** It is a dynamical property of a physical system, defined in the form of a functional $\mathcal{A}$ that takes a sequence of moves that define a trajectory as input, and returns a scalar number as output:

$$\mathcal{A} = \sum_{i}^{n} |\mathbf{f_i}| \Delta t_i \varepsilon_i, \tag{2.5}$$

  where $\mathbf{f}_i$, Δt_i and ε_i are, respectively, the applied control forces, their duration and the resulting covered distances.

- ***Power consumed:*** The total amount of power consumed $\mathcal{P}$ by the robot to move from start to the goal state is computed as:

$$\mathcal{P} = \frac{\sum_{i}^{n} \mathbf{f}_i \cdot \mathbf{d}_i}{T}, \tag{2.6}$$

where $\mathbf{f}_i$, $\mathbf{d}_i$ and T are, respectively, the applied control forces, the resultant displacement vectors and the total time.

- ***Smoothness:*** The smoothness $\mathcal{S}$ of a trajectory can be measured as a function of jerk (Hogan, 1984), the time derivative of acceleration:

$$\mathcal{J}(t) = \frac{d\,a(t)}{dt}. \tag{2.7}$$

For a given trajectory τ the smoothness is defined as the sum of squared jerk along τ:

$$\mathcal{S} = \int_{t_i}^{t_f} \mathcal{J}(t)^2\, dt, \tag{2.8}$$

where t_i and t_f are the initial and final time, respectively.

- ***Planning time:*** It is the total time consumed by the ontological physics-based planner to compute a solution trajectory.

- ***Success Rate:*** It is computed based on the number of successful runs.

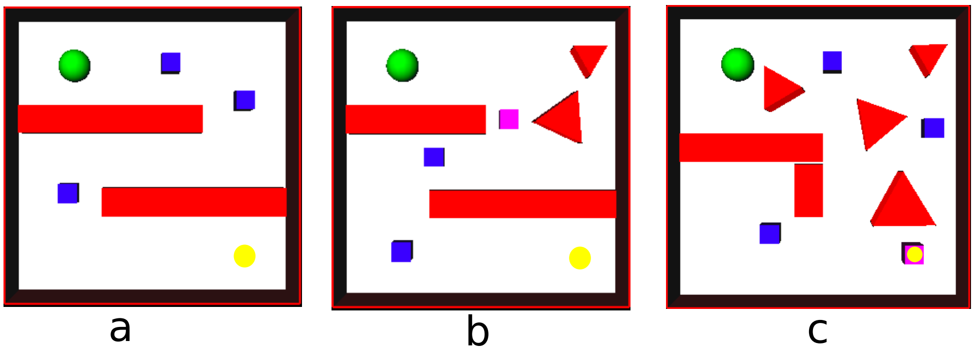


Figure 2.7: Simulation setup used for the benchmarking. The robot (green sphere) is depicted at the start configuration while the yellow circle represents the goal.

2.7.2 Benchmarking setup

In this section the results of benchmarking is presented. The benchmarking setup consists of a robot (green sphere), free manipulatable bodies (blue cubes), constraint oriented manipulatable bodies (purple cubes), and the fixed bodies (triangular prisms, walls, and floor). The benchmarking is performed with the three different scenes shown in Fig. 2.7, that are differentiated based on the degree of clutter. The robot is depicted at its initial configuration, being the goal robot configuration painted as a yellow circle. Fig. 2.7-a describes the simplest scene that consists of a robot, free manipulatable bodies, and fixed bodies. The second scene, represented in Fig.2.7-b, consists of a robot, free manipulatable bodies, fixed bodies, and a constraint-oriented

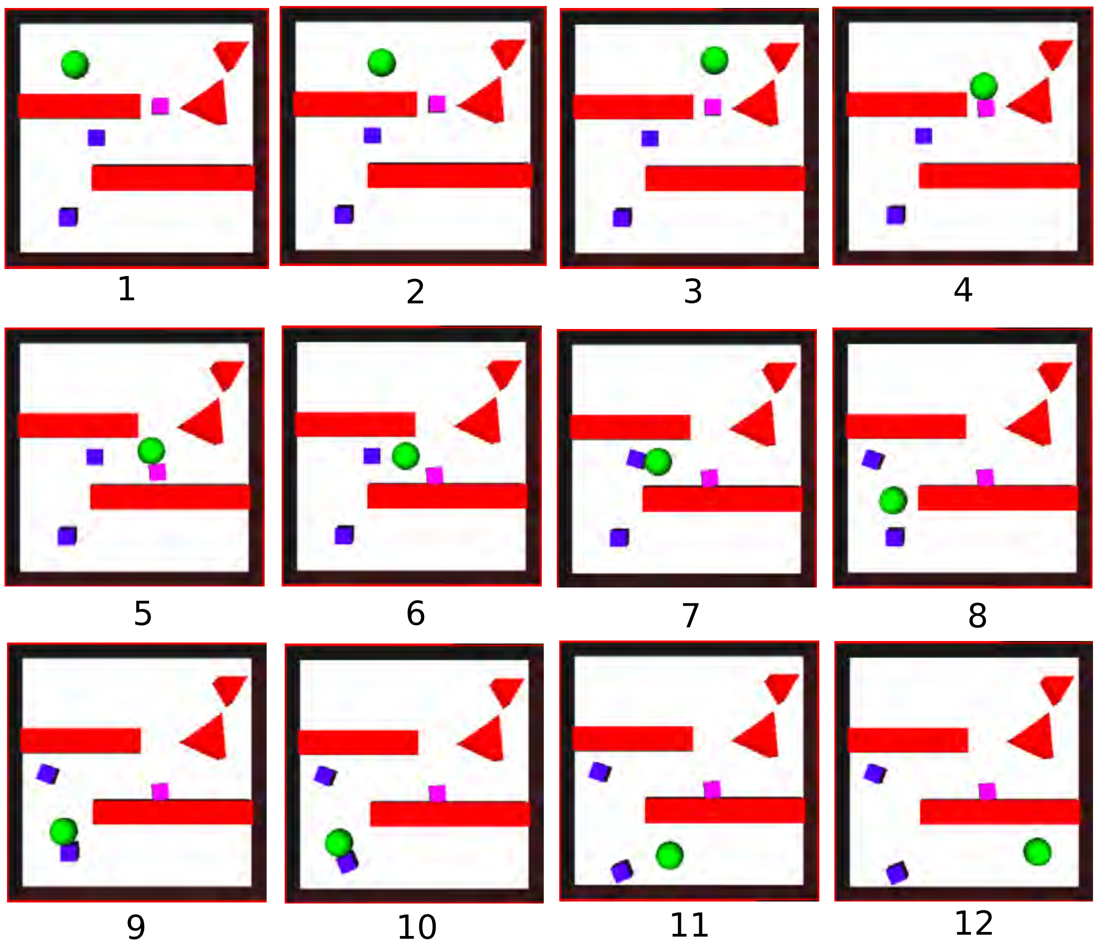


Figure 2.8: Sequence of snapshots of the execution using ontological physics-based motion planner.

body. In this scene the narrow passage is occupied by the constraint-oriented manipulatable body (it can only be pushed vertically, along the y-axis) that has to be pushed away by the robot in order to clear the path towards the goal. The final scene is depicted in Fig. 2.7-c and has the highest degree of clutter. The goal is occupied with a *co-mObject* (it can only be pushed horizontally, along the x-axis); in order to reach the goal region the robot needs to free it by pushing the body away.

2.7.3 Benchmarking results

The ontological physics-based motion planner has been run 10 times for each scene and for each of the kinodynamic motion planners summarized in Section 1.2.1. The average values of the benchmarking parameters are presented in the form of histograms. Fig. 2.8 shows as sequence

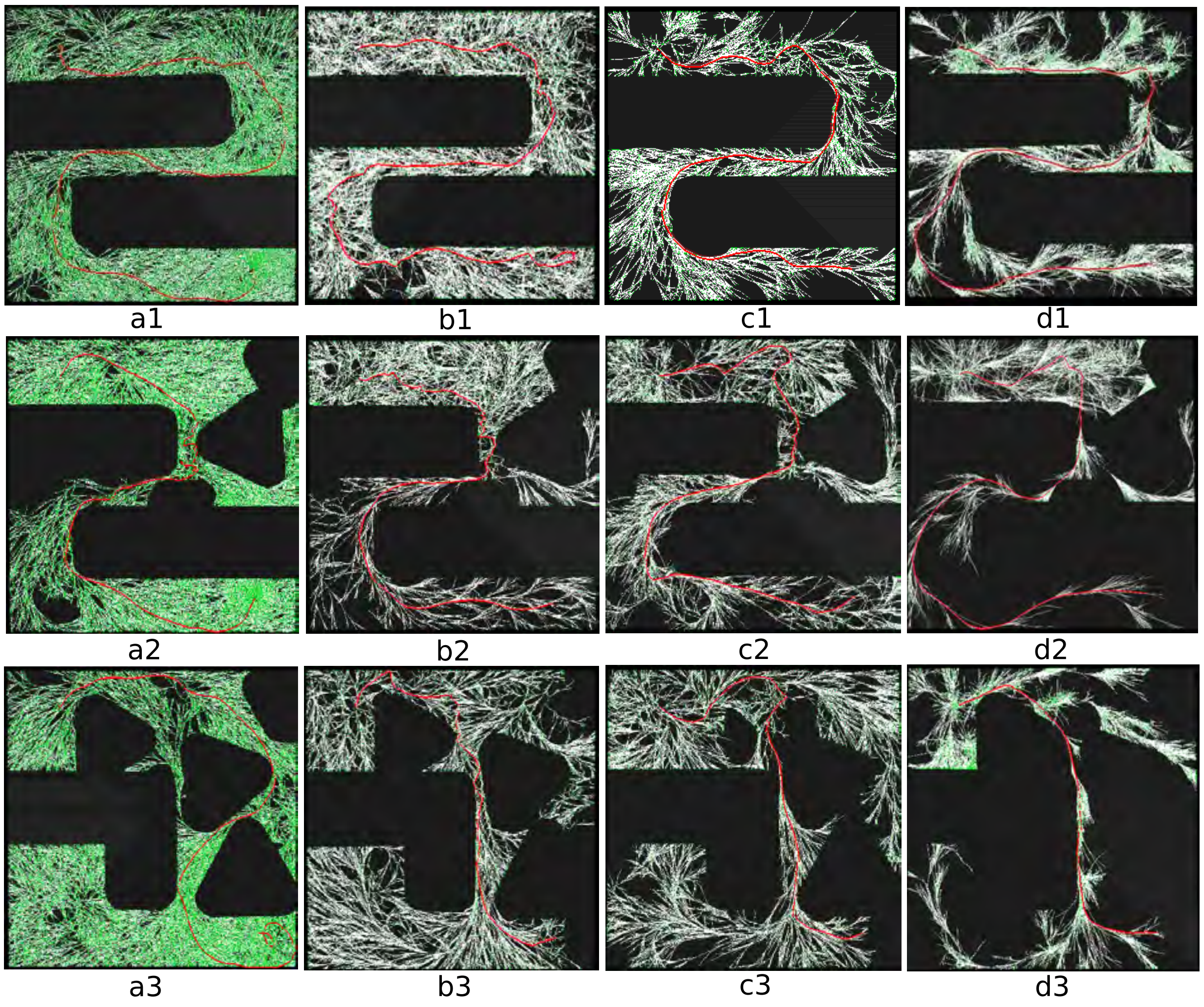


Figure 2.9: Configuration space and solution path: each row corresponds to a scene, the columns in each row (left to right) represents the configuration space and solution path using KPIECE, RRT, SyCLoP-RRT, and SyCLoP-EST respectively.

of snapshots of a a sample execution of the ontological physics-based motion planner using KPIECE. In order to estimate the coverage of configuration space, and the solution trajectory Fig. 2.9 depicts the configuration spaces and solution paths computed by different kinodynamic planners.

Fig.2.10 shows the average planning time (being the maximum allowed planning time set to 500 s). All the planners have been able to compute the solution within the maximum range of planning time except EST. Among all the planners KPIECE computed the solution most efficiently.

The success rate of the planners is described in Fig. 2.11. It is computed for each scene

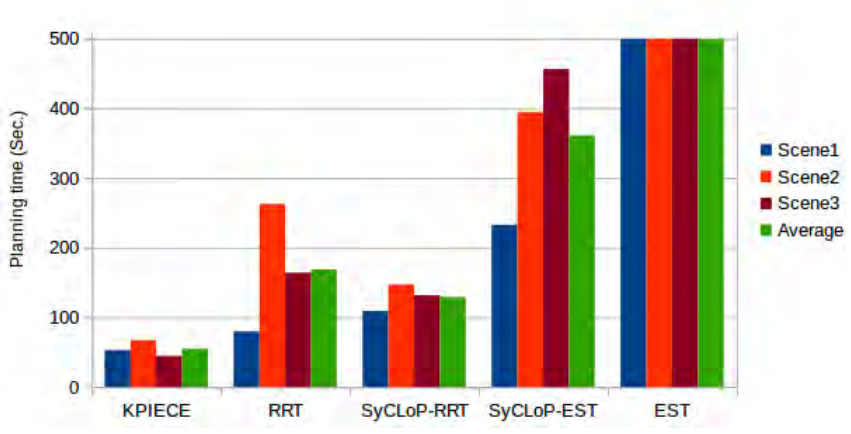


Figure 2.10: Average planning time (10 runs) for each scene and the overall averaged planning time (three scenes).

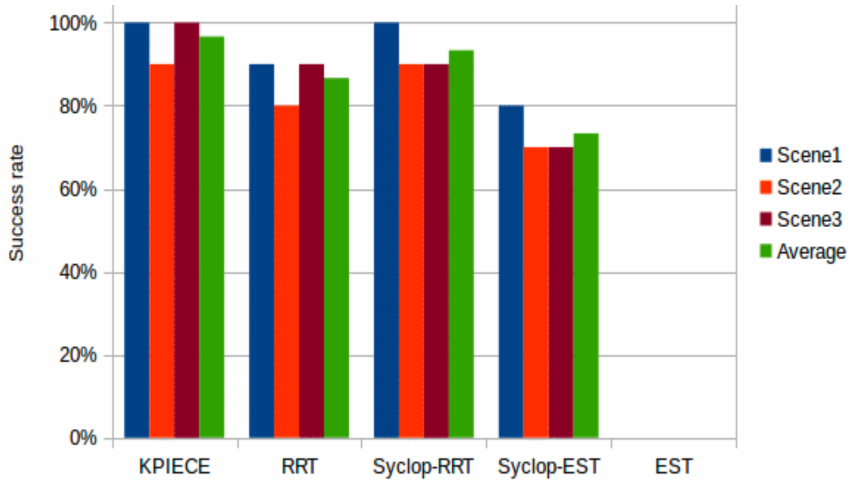


Figure 2.11: Success rate of the planners for each scene and the overall averaged success rate (three scenes).

individually based on the number of successful runs (i.e. how many times the planner computes the solution within the maximum limit of time), then the average success rate of each planner is determined by computing the average successful runs for the three scenes. Results show that the KPIECE has the highest overall success rate. The SyCLoP-RRT also shows an impressive success rate, whereas, EST has zero success rate.

The results of the dynamical parameters, action and power, are shown in Fig. 2.12 and Fig. 2.13, respectively. Regarding action, the solution trajectory with minimum amount of action is considered as the most appropriate. Among the planners that computed the solution within the given time, on average, KPIECE has the minimum amount of action and SyCLoP-RRT has the maximum one. Regarding power, it is desirable that the robot consume a minimum amount of power while moving along the solution. Our analysis shows that SyCLoP-RRT finds the best

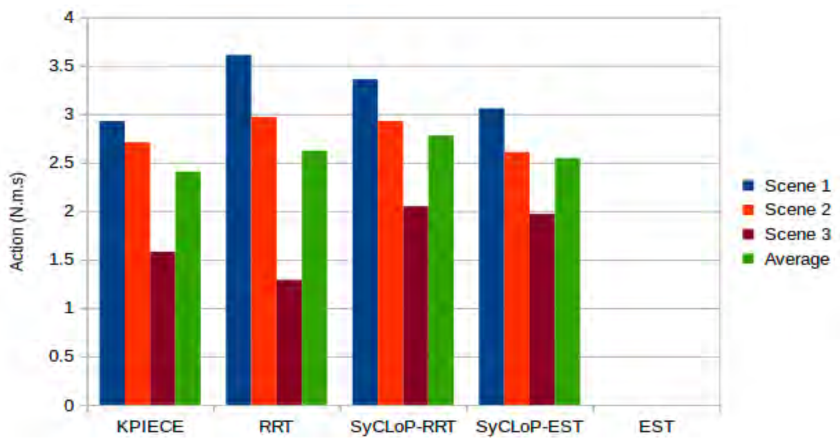


Figure 2.12: Average amount of action for each scene and the overall averaged amount of action (three scenes).

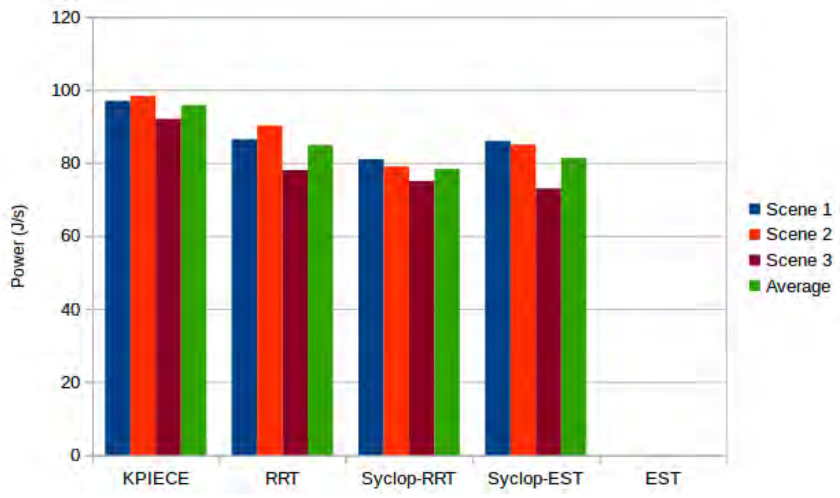


Figure 2.13: Average power consumed by the robot while moving along the solution path and the overall average (three scenes).

trajectory in terms of power whereas KPIECE is the worst. The results for RRT and SyCLoP-EST are almost the same. Regarding trajectory smoothness, Fig. 2.14 shows the comparison. The SyCLoP-RRT computes the most smooth trajectory among all the planners, whereas, KPIECE show the worst results in term of smoothness. Since EST was not able to compute the solution within the given time, the action, power and smoothness for EST are set to infinity and not shown in the histograms.

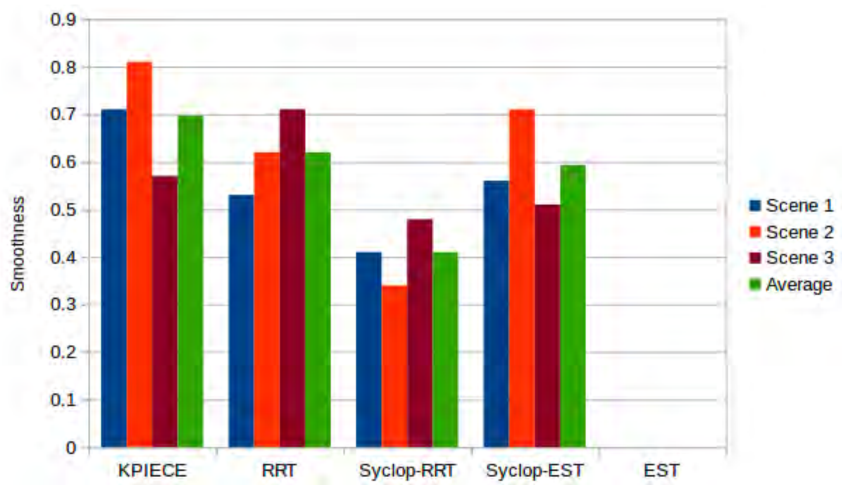


Figure 2.14: Smoothness measure for each scene and the overall average (three scenes).

2.7.4 Discussion

We proposed benchmarking criteria for physics-based planning. Based on the proposed criteria, the performance of physics-based planning is evaluated using different kinodynamic motion planners. Our analysis shows that in terms of planning time, success rate and the action value, KPIECE is the most suitable planner. SyCLoP-RRT shows significant results in terms of smoothness of the computed trajectory and power consumed by the robot moving along the computed path. The planning time and success rate for SyCLoP-RRT is also impressive. RRT shows average results throughout the evaluation. SyCLoP-EST is good in terms of action value and power consumption but the value of planning time is very high and it has a low success rate. The EST was not able to compute the solution within time and has zero success rate.

2.8 Applications of the proposed approach

The ontological physics-based motion planner has been used in task planning studies. The Integrated task and motion planning approach presented in (Akbari et al., 2015) is based on Planning Graph i.e., a task planning approach that searches in the planning space the sequence of actions to perform the given task (Ghallab et al., 2004). Within this approach, the cost of each action is computed using the ontological physics-based motion planner that computes the path and provide the dynamic measures such as power consumption and action for the computed path. These dynamic measures are used for physics-based reasoning to decide which action is suitable from the current state. Another task planning approach that used the ontological physics-based motion planning approach is presented in (Akbari et al., 2016c). It uses a heuristic-based task planning algorithm called Fast-Forward (FF). The heuristic that guide the search is computed based on the feedback (such as power consumption) provided by the ontological physics-based motion planner. As a result, the computed sequence of actions computed by the proposed planner is the one that consumes the minimum power to perform the given task, instead of the one that has less number of actions, as standard FF does.

2.9 Conclusions

The chapter has presented the integration of ontological knowledge with physics-based motion planning. Aiming to enhance the planning process for manipulation problems. The results described show that the proposed algorithm has a better performance than the simple physics-based planning. Moreover, evaluation criteria for the physics-based motion planners have been presented. This benchmarking criteria compute dynamic parameters (such as power consumed by the robot to move along the solution path) for the evaluation of the quality of the computed solution path. Further, based on the proposed benchmarking criteria, the performance of the ontological physics-based motion planner (using different kinodynamic motion planners) has been evaluated.

Chapter 3

Motion planning using knowledge-based reasoning on robot-object interactions

3.1 Introduction

The preliminary work presented in the previous chapter demonstrated the importance of using knowledge (described by ontologies) to guide physics-based motion planning. The approach presented in this chapter greatly enhances the state transition model, which is the core of the proposal, by introducing the low-level geometric reasoning process that allows the use of an adaptive control range, thus obtaining power-efficient solutions. On the other hand, it reduces the computational cost by introducing an offline inference process that prevents online queries to the ontological knowledge. This approach is originally presented in (Muhayyuddin et al., 2017a). It consists of a knowledge-oriented physics-based motion planning method equipped with a control sampling strategy that allows the search of a power-efficient motion plan, which may include free robot motions along with interactions with manipulatable objects that may be obstructing the path. The proposal is called κ-PMP. The main components of the proposal are: *(1)* the partition of $\mathcal{C}$ into different regions as a function of whether the robot can move freely or interact with manipulatable objects; *(2)* an instantaneous reasoning process that performs low-level geometric reasoning in order to analyze the configuration space regions and update the sampling range; *(3)* a detailed representation of knowledge, which is categorized into semantic knowledge and manipulation knowledge, to help improving the knowledge-based inference process. The proposed framework consists of a high-level and a low-level layer, that are connected through a ROS-based communication layer. The high-level layer contains the knowledge about the robot and the environment, that is used by the low-level motion planner for reasoning about the sampled controls and to update the manipulation constraints. This hierarchical structure results in power-efficient motion plans for the robot to efficiently interact with the objects in the environment. The performance of the presented planning approach is tested with three different

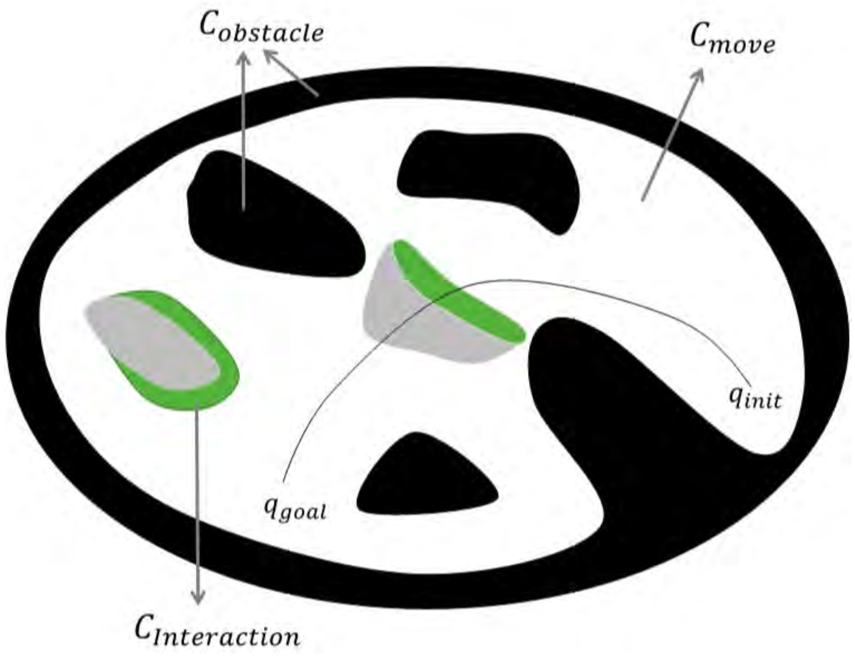


Figure 3.1: Configuration space: $\mathcal{C}_{\text{obs}}$ shown in black (collisions with fixed obstacles) and gray (collision with manipulatable obstacles), $\mathcal{C}_{\text{interaction}}$ in green and $\mathcal{C}_{\text{move}}$ in white.

scenarios: an holonomic mobile robot, a car-like mobile robot and a planar manipulator. The results are compared (in terms of power consumption, planning time and success rate) with simple physics-based planning approaches.

3.2 Problem formulation

Let a physics-based motion planning problem be defined as the tuple $(\mathcal{X},\mathcal{U}, f, \mathcal{K}, \mathcal{F}, x_{init}, \mathcal{X}_{goal})$, where:

- $\mathcal{X}$ represents the state space; it is a differential manifold.
- $\mathcal{U}$ represents the control space; it contains the set of all possible control inputs that can be applied to the robot.
- $f : \mathcal{X}^i \times \mathcal{U} \longrightarrow \mathcal{X}^{i+1}$ is the propagation function.
- $\mathcal{K}$ is the abstract knowledge containing all the available knowledge about the world such as object classification, manipulation regions, and physical properties. $\kappa \subset \mathcal{K}$ is the instantiated knowledge that represents the knowledge that is valid for a particular instance of time.

- $\mathcal{F} : \kappa \times \mathcal{X} \longrightarrow \{0,1\}$ is the physics-based state validity checker. It evaluates the state generated by applying f, and returns 1 if it satisfies all the constraints imposed by κ, or returns 0 otherwise.
- $x_{init} \in \mathcal{X}$ is the initial state.
- $\mathcal{X}_{goal} \subset \mathcal{X}$ is the goal region.

Consider a motion planning problem where no collision free trajectory from start to the goal exist (e.g. either the goal or the way towards it might be occupied with manipulatable objects). The objective is to determine the set of efficient control inputs $\{u_1, \dots, u_n\} \in \mathcal{U}$ and the set of associated time durations $\{t_1, \dots, t_n\}$ such that, if sequentially applied to the system using f and starting from x_{init}, a goal state $x_n \in \mathcal{X}_{goal}$ is reached, being the resultant trajectory power-efficient and satisfying all the constraints (i.e. a trajectory that avoids collisions with fixed obstacles but that may collide with manipulatable objects to push them away from the solution path). The proposed approach (at each step) will determine the control to be applied using the low level reasoning about the object dynamics, in such a way that the resultant solution will be power efficient (no optimization of any kind such as path length will be considered).

The robot's configuration space $\mathcal{C}$, i.e. the set of all possible configurations of the robot, can be divided into the set of geometrically accessible regions, called $\mathcal{C}_{\text{free}}$, and the set of the forbidden ones (those corresponding to collision with obstacles), called $\mathcal{C}_{\text{obs}}$. The condition $\mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}} = \mathcal{C}$ holds. In this work, $\mathcal{C}_{\text{free}}$ is further divided into $\mathcal{C}_{\text{move}}$ and $\mathcal{C}_{\text{interaction}}$ representing, respectively, the regions where the robot can move freely and those where the robot can enter in contact with the manipulatable objects, i.e. the set of configurations corresponding to the robot[1] being placed in an *mRegion* (see Fig. 3.1). The condition $\mathcal{C}_{\text{move}} \cup \mathcal{C}_{\text{interaction}} = \mathcal{C}_{\text{free}}$ holds.

3.3 Reasoning-based state transition model

In order to find a power-efficient solution, we have developed an approach inspired by our daily life experience where, in order to perform the task robustly, we adapt the forces according to the interaction with the environment, e.g. we do not exert the same force while pushing a plate, a table, or when moving freely. We propose the use of any kinodynamic planner, such as KPIECE or RRT, and to equip it with a state transition model that computes the next state based on a dynamic engine and a reasoning process that uses instantiated knowledge. This knowledge defines from where objects should be manipulated and with which range of forces. The knowledge representation used is detailed in Section 3.4 and the knowledge inference and reasoning process in Section 3.5.

The knowledge about the task and the workspace is modeled in two levels:

[1]In the case of manipulators the pose of the tool will be considered.

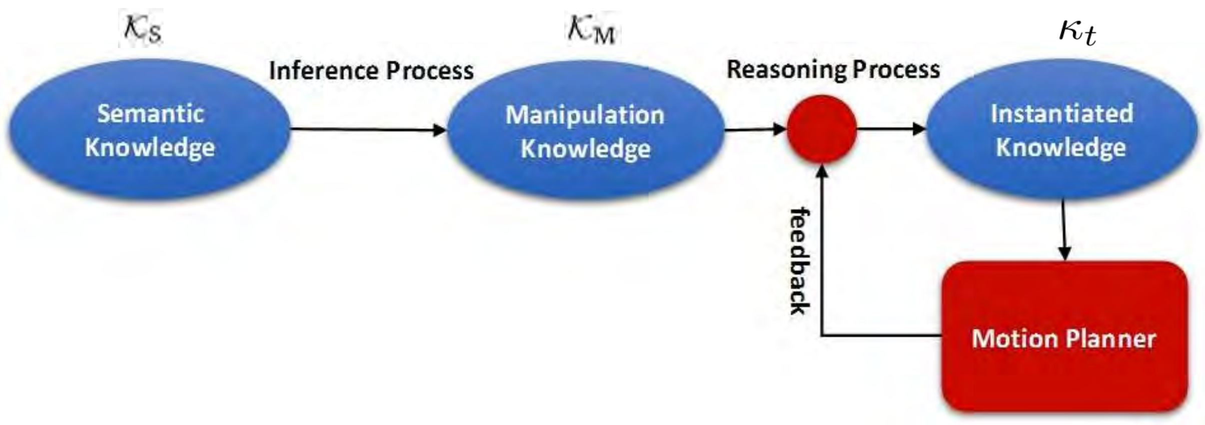


Figure 3.2: Flow of knowledge from the high-level abstract knowledge to the low-level instantiated knowledge.

- The *abstract knowledge* $\mathcal{K}$: It is a high-level representation of knowledge composed of the *semantic knowledge* $\mathcal{K}_S$ and the *manipulation knowledge* $\mathcal{K}_M$. The semantic knowledge describes, using ontologies, information of the task such as the kinematic and dynamic properties of the robot, of the objects, and the manipulation constraints. From $\mathcal{K}_S$ the manipulation knowledge $\mathcal{K}_M$ is inferred. It contains all the necessary information that is required for the motion planner, such as the type of objects and how they can be manipulated. Abstract knowledge remains the same during the whole planning process.

- The *instantiated knowledge* κ_t: It is a low-level representation, updated at each instance of time by the reasoning process, based on the manipulation knowledge and on the feedback received from the motion planner (e.g. if at a particular instance of time one of the *mRegion* of an *mObject* is occupied by some other object, then if there is an *mRegion* in the opposite side, it will be deactivated by the reasoning process and κ_t will be updated accordingly). The instantiated knowledge is used in the state transition model as explained next.

The dynamic flow of knowledge is shown in Fig. 3.2, where it is illustrated that the manipulation knowledge is extracted from semantic knowledge (ontologies) and used by the reasoning process along with the feedback from the motion planner to update the instantiated knowledge. This instantiated knowledge, as shown in Fig. 3.3, is the key module of the state transition process, that takes the state of the world $\mathbf{q}_t$ as input and generates the next state $\mathbf{q}_{t+1}$ by applying an appropriate control based on this knowledge:

$$\kappa_t = \xi(\mathcal{K}_M, \mathbf{q}_t) \tag{3.1}$$

$$\mathbf{q}_{t+1} = f(\mathbf{q}_t, \mathbf{u}_t(\kappa_t)) \tag{3.2}$$

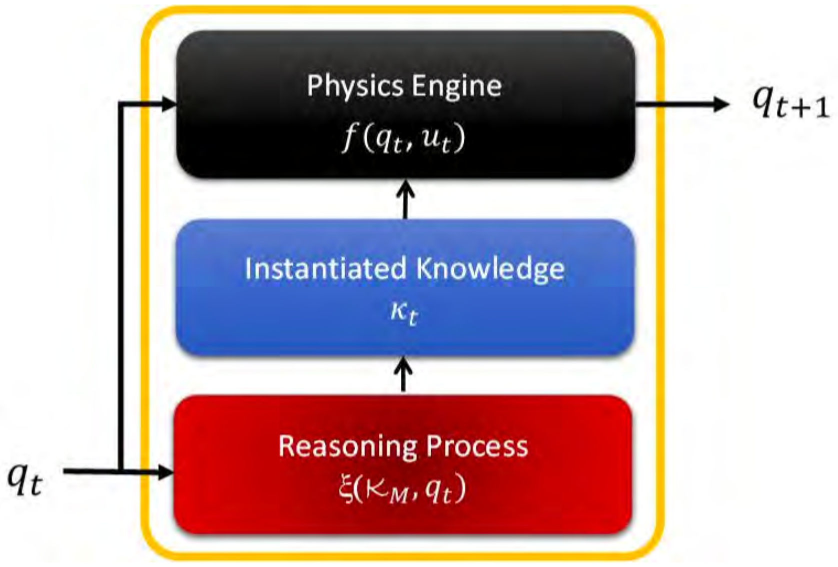


Figure 3.3: State transition model for the κ-*PMP*.

That is, the state transition process consists of three modules: the physics engine, the instantiated knowledge, and the low-level reasoning. The reasoning process is responsible for updating the instantiated knowledge κ_t with the manipulation constraints (determining which are the active manipulation regions) and with the region of the configuration space where the robot is located (either $\mathcal{C}_{\text{move}}$ or $\mathcal{C}_{\text{interaction}}$). With this information the instantiated knowledge determines the set of controls and selects one to be applied (the set will be determined in such a way that if the robot is in a region where interaction with an object is possible then the range of controls must allow the manipulation of the object). Finally, the physics engine (i.e. physics-based state propagator) generates $\mathbf{q}_{t+1}$ by applying the randomly sampled control $\mathbf{u}_t$ to $\mathbf{q}_t$. The validity of the resulting state will be checked using the physics-based state validity checker $\mathcal{F}$ that takes into account the instantiated knowledge (i.e. a collision state is only valid if the robot collides with manipulatable object from one of its manipulation regions).

3.4 Knowledge representation

This section presents the high level and low level representation of knowledge for motion planning. The abstract knowledge $\mathcal{K}_S$ is represented using ontologies, the manipulation knowledge $\mathcal{K}_M$ is inferred from $\mathcal{K}_S$ and the instantiated knowledge κ_t used at the motion planning level.

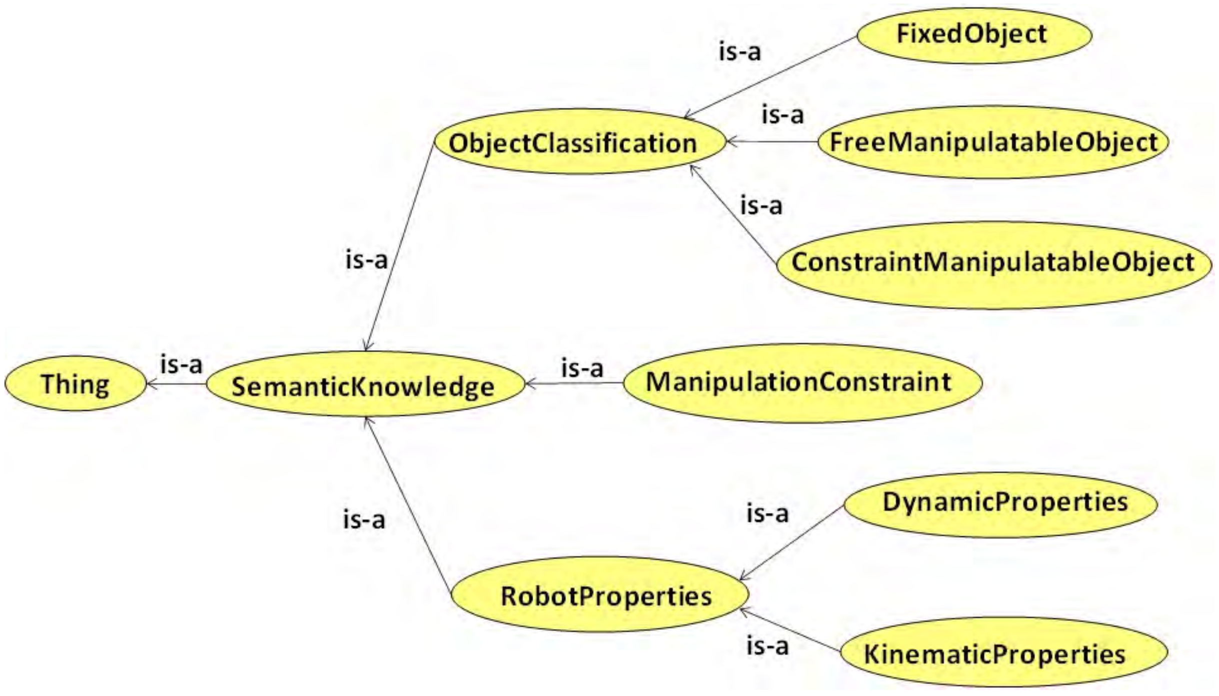


Figure 3.4: OWL-based semantic knowledge taxonomy.
https://sir.upc.edu/projects/ontologies.

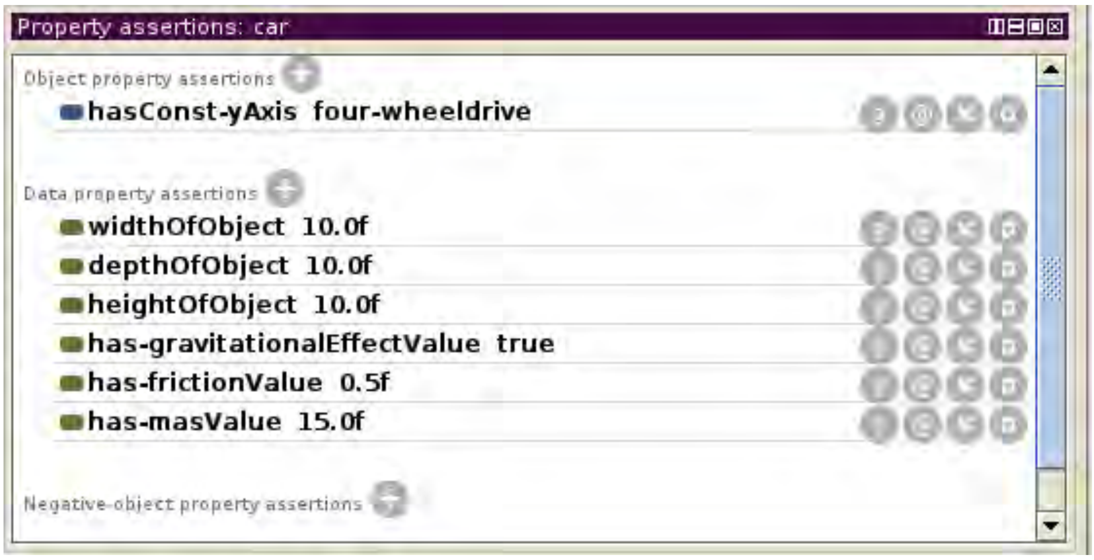
3.4.1 Abstract knowledge

The abstract knowledge is the high-level representation of knowledge which remains fixed throughout the planning process. It is divided into the *Semantic Knowledge* containing information about the workspace and the robot, coded as an OWL taxonomy, as depicted in Fig. 3.4, and the *Manipulation Knowledge* which involves knowledge related to how the robot can interact with the workspace, and which is inferred from the semantic knowledge.

Semantic knowledge ($\mathcal{K}_S$)

Semantic knowledge categorizes information within the following classes:

- Class *"RobotProperties"* describes the properties of the robot in two subclasses. Geometric constraints of the robot such as joint limits are stored in the class *KinematicProperties*; differential properties of the robot such as bounds on forces, torques, velocities, and accelerations (global properties that condition the maximum capacity of the robot) are stored in the class *DynamicProperties*.


Property assertions: car

Object property assertions

hasConst-yAxis four-wheeldrive

Data property assertions

widthOfObject 10.0f

depthOfObject 10.0f

heightOfObject 10.0f

has-gravitationalEffectValue true

has-frictionValue 0.5f

has-masValue 15.0f

Negative object property assertions

Figure 3.5: Screen shot of the Protégé editor showing the semantic properties of a car-like object.

- Class *"ObjectClassification"* is used to describe the objects in the workspace such as, fixed (*fixedObject*), free manipulatable (*free-mObject*) and constraint-oriented manipulatable (*co-mObject*).
- Class *"ManipulationConstraint"* describes orientation constraints on the motion of bodies and objects.

The hierarchy of knowledge among the classes can be represented using Description Logic (DL). For instance the hierarchy for a constraint-oriented manipulatable object is described below:

$\exists hasSuperclass(Thing, SemanticKnowledge)$
$\wedge \exists hasClass(SemanticKnowledge, ObjClassification)$
$\wedge \exists hasSubclass(ObjectClassification, co\text{-}mObject)$

where $\wedge$ and $\exists$ represent *conjunction* and *exist*, respectively.

The semantic properties (that are stored in OWL) are divided into object properties and data properties. The former are used to describe the relationships between the individuals, and the latter are used to assign the values to the physical attributes. As an example, some of the semantic properties of a car-like object are depicted in Fig. 3.5 and explained below in terms of DL.

$Object := Car$

$\wedge\exists hasWheel(Car, Wheel)$
$\wedge\exists hasBody(Car, Body)$
$\wedge\exists hasWheel(Wheel, alongYaxis)$
$\wedge\exists canMove(Car, alongXaxis)$

The above stated DL description of an object explains that car is composed of wheels and body, the motion of the car-like object is constraint by the wheel axis being in the y direction so that the car can only move along the associated x-axis.

The data properties in terms of description logic are represented as follows:

$Object := Car$
$\exists hasWidth(Car, Value)$
$\wedge\exists hasDepth(Car, Value)$
$\wedge\exists hasHeight(Car, Value)$
$\wedge\exists hasGravity(Car, Value)$
$\wedge\exists hasFriction(Car, Value)$
$\wedge\exists hasMass(Car, Value)$

It describes the dimension of the car, response to the gravitational (if considered as dynamic object, the value will be true and false otherwise) and the values of the friction coefficient (between wheels and road) and the mass of the car respectively.

Manipulation knowledge ($\mathcal{K}_M$)

Manipulation knowledge is inferred from $\mathcal{K}_S$ using a Prolog inference process that will be detailed later in Sec. 3.5.1. It contains the classification of objects (*fixedObject*, *free-mObject*, or *co-mObject*), a complete set of *mRegions* for manipulatable objects, physical attributes of objects, and kinematic and dynamic properties of the robot (such as joint limits and bounds on forces and velocities). Manipulation knowledge remains fixed throughout the motion planning process. $\mathcal{K}_S$ contains the maximum possible knowledge about the world and $\mathcal{K}_M$ that related to a particular motion planning problem (e.g. it must be updated if the features of the robot change).

3.4.2 Instantiated knowledge κ_t

Instantiated knowledge is the low level representation of knowledge (at the motion planning level). It is dynamic and valid for a particular instant of time (containing all possible constraints that are valid for that instant of time) and is updated for the next time step. The instantiated knowledge contains two main components: *(1)* the knowledge about the valid and invalid manipulation regions, as well as dynamical properties (such as masses, friction coefficients) of the objects; *(2)* the control sampling range (such as bounds on control forces and torques) to be used by the motion planner at the current instant of time (by using the state propagator, i.e. the dynamic engine).

The instantiated knowledge is updated by the low-level reasoning process based on the high-level manipulation knowledge and the feedback from the motion planner. The reasoning process is detailed in Sec. 3.5.2.

3.5 Knowledge inference and reasoning process

The flow of knowledge was graphically illustrated in Fig. 3.2. It includes the knowledge inference process and the reasoning process. The knowledge inference process is the pre-processing step responsible of inferring the manipulation knowledge $\mathcal{K}_M$ from the ontological semantic knowledge $\mathcal{K}_S$. On the other hand, the reasoning process is the responsible of updating the instantiated knowledge at each instant of time while planning, based on $\mathcal{K}_M$ and the current state fed back by the motion planner.

3.5.1 Knowledge Inference Process

The inference process is performed using Knowrob (Tenorth and Beetz, 2009), a knowledge-based processing tool for robotics applications that allows operating over OWL data bases, extended with the following predicates (coded in Prolog) tailored to extract the necessary information for the physics-based motion planner:

- *object_classification(?Obj, ?ObjType)*: Given an object *Obj* returns the type of the associated object *ObjType* by evaluating its category. Those manipulatable objects that are too heavy for the robot to be manipulated are changed to fixed.
- *manipulatable_region(?Obj, ?ManipRgns)*: Given a manipulatable object *Obj* computes the set of associated manipulatable regions *ManipRgns* taking into account the manipulation constraints, if any.
- *object_properties(?Obj, ?PhysicalProps, ?Dimension)*: Given an object *Obj* returns its physical properties *PhysicalProps*, including mass and friction values, as well as gravitational effect and the dimension *Dimension* of the object.
- *robot_properties(?DynamicsProps, ?KinematicProps)*: Returns the dynamics properties of the robot (forces and velocities limits) in *DynamicsProps*, and the kinematic properties (joint limits) in *KinematicProps*.

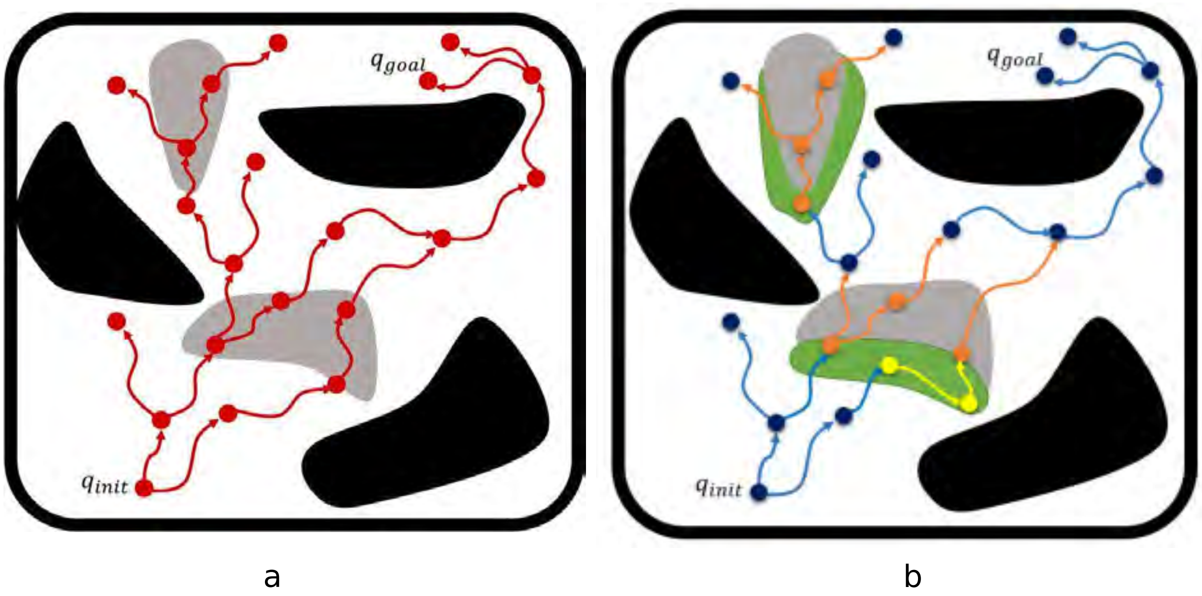


Figure 3.6: Sample and propagation using: (a) the simple physics-based motion planner; (b) κ-PMP. The control sampling range is selected based on the dynamics properties of the target object and the manipulation region. Blue samples represent the lower values of control range, whereas orange samples represent the higher values. Yellow samples represents the decrease in forces during the transition between free and contact motions.

At any given state fed back by the motion planner (corresponding to the node to be expanded), the reasoning process uses geometric reasoning to update the instantiated knowledge with the current information on the manipulation regions and on the control sampling range. Those manipulation regions that become useless are deactivated; the others are set active. A manipulation region becomes useless if it is occupied by an obstacle (i.e. the robot cannot access it), or if the motion of the object is not possible when the robot interacts with it from the manipulation region, e.g. if the front manipulation region of the car-like object is blocked with a *mObject* it is deactivated, as well as the rear *mRegion* because the robot can not exert forces from there until the front *mRegion* becomes free.

3.5.2 Reasoning process

Kinodynamic planners sample both the direction and module of the control to be applied to extend the tree data structure. The proposed reasoning process computes the module range from where to sample depending on whether the robot is located in $\mathcal{C}_{\text{move}}$ or $\mathcal{C}_{\text{interaction}}$, and in this latter case depending on whether the robot is in collision or not. The normal control range to move the robot freely in $\mathcal{C}_{\text{move}}$ will be diminished when being in $\mathcal{C}_{\text{interaction}}$, and if contact occurs then it will be increased based on the weight of the object to be pushed. Let F be the module range from where to sample, and let F take the value $F = [f_{\text{min}}, f_{\text{max}}]$ when the robot

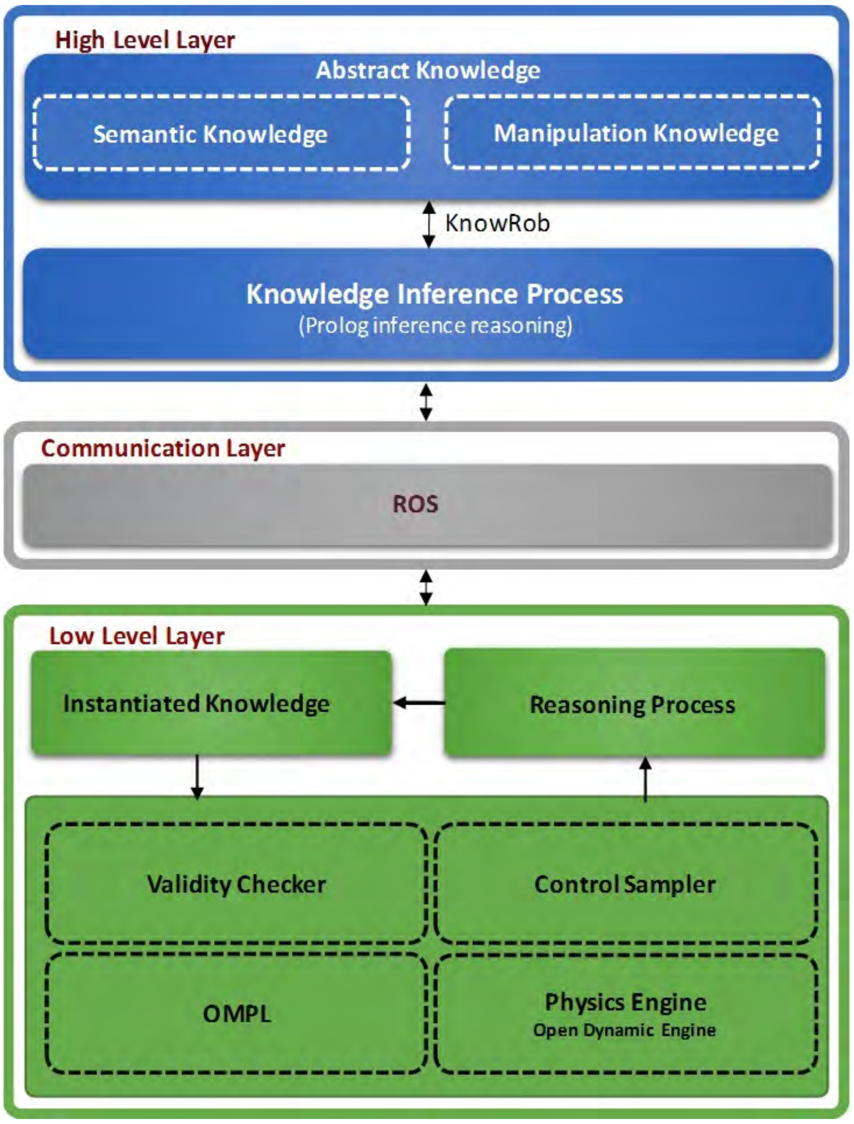


Figure 3.7: Knowledge-oriented physics-based motion planning (κ-*PMP*) for power-efficient motion plan.

is in $\mathcal{C}_{\text{move}}$. Then, when the robot is in $\mathcal{C}_{\text{interaction}}$:

- $F = \alpha[f_{\text{min}}, f_{\text{max}}]$ with $\alpha < 1$ if no contact occurs,
- $F = [f_{\text{min}} + f_{\text{obj}}, f_{\text{max}} + f_{\text{obj}}]$ with $f_{\text{obj}} = \mu_{\text{obj}}\, m_{\text{obj}}\, g$, if contact occurs,

where μ_{obj} is the friction coefficient between the object and the floor, m_{obj} is the mass of the object, and g the gravitational force. In case of a manipulator (kinematic chain), the range of forces is converted to a range of joint torques using the transposed Jacobian.

To illustrate this, Fig. 3.6 qualitatively depicts the difference between the use of the proposed control sampling range and a fixed range, as done in standard physics-based motion planning approaches (Muhayyuddin et al., 2015; Şucan and Kavraki, 2012). In our proposal, the higher value of control forces will only be used when the robot is in contact with an object, and according to its weight. If a fixed lower control range is set then it may not be able to push the objects (to clear the regions) and fails to compute the path, on the other hand, if a higher control range is set then it consumes unnecessary power and may result in a huge displacement of the object. Moreover, prior to contact, the proposed controls slow down the robot to have a smooth transition from no contact to contact.

3.6 The κ-*PMP* approach

3.6.1 Proposed framework

The proposed framework for power-efficient physics-based motion planning is depicted in Fig. 3.7. It is a hybrid planning framework that consists of three main layers: the high-level layer devoted to the knowledge management and inference process, the communication layer, and the low-level layer devoted to the physics-based motion planning.

The high-level layer that contains the high-level representation of knowledge is divided into the semantic knowledge coded in the form of ontologies, and the manipulation knowledge inferred from the semantic knowledge using the knowledge inference process module.

The low-level layer consists of the instantiated knowledge κ_t (that contains the temporary manipulation constraints and the bounds on the control forces), the reasoning process (responsible of updating the instantiated knowledge from the current state and the manipulation knowledge), and the motion planner (whose main modules are the physics-based state validity checker $\mathcal{F}$, the control sampling module, a standard kinodynamic planner like KPIECE or RRT from the Open Motion Planning Library (Şucan et al., 2012a), and the ODE-based physics engine). This layer is developed within *The Kautham Project* (Rosell et al., 2014), an open source framework for motion planning that includes geometric, differential and physics-based motion planners (including those that require ontological knowledge).

The communication layer is based on ROS and communicates the high-level abstract knowledge and the low-level motion planning layer: the abstract knowledge is encapsulated as a ROS service and the motion planning layer accesses it as a ROS client.

Algorithm 2: κ-PMP

```
inputs : Initial state $\boldsymbol{q}_{\text{init}}$, Goal region $\boldsymbol{Q}_{\text{goal}} \in \mathcal{C}$, Threshold $T_{max}$
output: A path from $\boldsymbol{q}_{\text{init}}$ to $\boldsymbol{q} \in \boldsymbol{Q}_{\text{goal}}$
1  WorkspaceInit()
2  $\mathcal{K}_{\text{S}}$ ←SemanticKnowledgeGenerator()
3  $\mathcal{K}_{\text{M}}$ ←ManipulationKnowledgeInference($\mathcal{K}_{\text{S}}$)
4  while $t < T_{max}$ do
5      $\kappa_t \leftarrow$ ReasoningProcess($\mathcal{K}_{\text{M}}, \mathbf{q}$)
6      SelectNodeToExpand()
7      $\{u, n\} \leftarrow$ SampleControlsAndSteps($\kappa_t$)
8      for $i = 0 \; \rightarrow \; i < n$ do
9          $\boldsymbol{q}_{\text{new}} \leftarrow$ Propagate($\boldsymbol{q}, u$)
10         if StateValidityChecker($\boldsymbol{q}_{new}$,$\kappa_t$) then
11             UpdateConnections()
12         else
13             break
14     if $\boldsymbol{q}_{new} \in \boldsymbol{Q}_{goal}$ then
15         return Path($\boldsymbol{q}_{\text{new}}$)
16     return NULL
```

Algorithm 3: ReasoningProcess($\mathcal{K}_{\text{M}}, \mathbf{q}$)

```
1  $\Gamma \leftarrow$ UpdateManipulationConstraints($\mathcal{K}_M, \mathbf{q}$)
2  $\mathcal{L} \leftarrow$ ComputeRobotLocation($\mathbf{q}$)
3  $\{f_{\text{min}}, f_{\text{max}}\}$ ←ComputeControlRange($\mathcal{L}$)
4  $\kappa_{\text{new}}$ ←UpdateInstantiatedKnowledge($\Gamma, \{f_{\text{min}}, f_{\text{max}}\}$)
5  return $\kappa_{\text{new}}$
```

3.6.2 Algorithm

The planning process is sketched in algorithm 2. It takes as input the initial state $\mathbf{q}_{init}$, the goal region $\mathbf{Q}_{goal}$, and the maximum allowed planning time T_{max}. If a solution is found, it returns the path from $\mathbf{q}_{init}$ to $\mathbf{q}_{goal} \in \mathbf{Q}_{goal}$, or NULL otherwise. To sample the states and construct the planner data structure, the approach provides the flexibility to use any sampling-based kinodynamic motion planner (such as RRT, KPIECE, SyCLoP) offered by OMPL, together with the Open Dynamic Engine as state propagator.

Lines [1-3] of the algorithm are the preprocessing steps for the motion planner; lines [4-15] contain the planning process. The main functions used are the following:

- *WorkspaceInit:* Iinitializes the state of the robot and of the objects in the environment.
- *SemanticKnowledgeGenerator:* Creates the semantic knowledge $\mathcal{K}_S$ from the ontologies.

- *ManipulationKnowledgeInference:* Infers $\mathcal{K}_M$ from $\mathcal{K}_S$.
- *ReasoningProcess:* Updates the instantiated knowledge by updating the manipulation constraints and the range of the controls, as detailed in algorithm 3.
- *SelectNodeToExpand:* Selects the node to expand following the node selection process of the kinodynamic planner used.
- *SampleControlsAndSteps:* Samples the controls and the number of steps describing the number of times the selected control will be applied repeatedly. To sample the controls any control sampling strategy, such as steering control sampling, can be used.
- *Propagate:* Applies the sampled control $\mathbf{u}$ on the state $\mathbf{q}$ for Δt time using the Open Dynamic Engine as state propagator.
- *StateValidityChecker*: Is the physics-based state validity checker ($\mathcal{F}$) that validates the newly generated state by taking into account the instantiated knowledge.
- *UpdateConnections:* Adds the accepted state to the planner data structure and updates the connections accordingly.
- *Path:* Returns a path from $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$ if the last generated state lies in the goal region, and NULL otherwise.

Algorithm 3 implements the reasoning process, it contains the following steps:

- *UpdateManipulationRegions:* Activates and deactivates the manipulation regions using the geometric reasoning based on $\mathcal{K}_M$ and the current state.
- *ComputeRobotLocation:* Determines the region $\mathcal{L}$ where the robot lies.
- *ComputeControlRange:* Determines the control range as a function of $\mathcal{L}$.
- *UpdateInstantiatedKnowledge:* Updates the data structures of the instantiated knowledge with the updated manipulation regions and the control sampling range.

The key point of the algorithm is that, during planning, the instantiated knowledge is updated at each instant of time by the reasoning process (Line 5), conditioning the sampling of the controls (Line 7) and the state validity checking procedure (Line 9).

3.6.3 Simulation setup

The proposed approach is validated using three different robot models, depicted in Fig. 3.8. The scenario presented in Fig. 3.8-a is for an holonomic mobile robot. It consists of a robot (sphere),

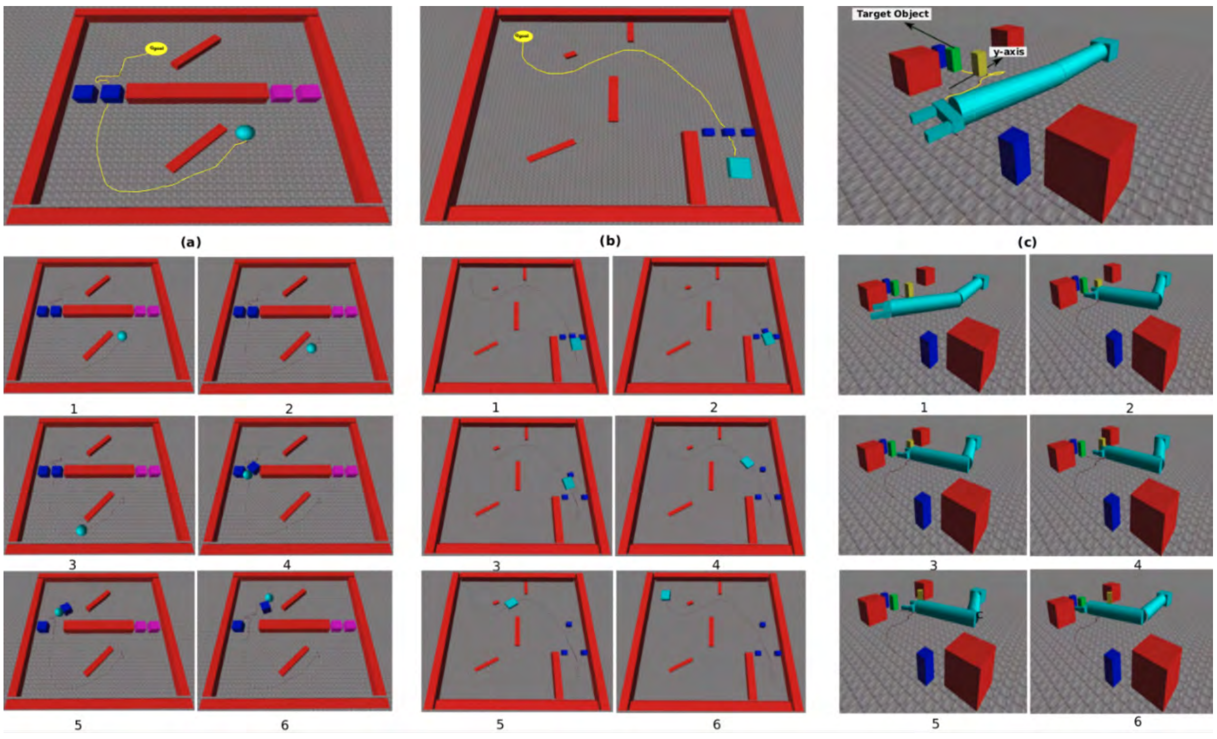


Figure 3.8: Planning scenes: (a) an holonomic mobile robot; (b) a car-like mobile robot; (c) a planar kinamatic chain. Video: https://sir.upc.edu/projects/kautham/videos/k-PMP1.mp4

free-mObjects (blue and purple cubes, being the purple ones heavier than blue cubes) and *fixed-objects* (red walls). The problem is to go from $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$, being the possible solutions blocked by the blue or the purple cubes. Since the motion of this robot is controlled by applying the control force, the range of the control force will be selected by the reasoning process according to the object to be pushed (blue or purple), as explained in Sec. 3.5.2. The higher values of controls will only be applied when the robot is in interaction with the object, unlike the conventional planners that fix the control range at the start, our approach dynamically varies the control range. The power $\mathcal{P}$ consumed by the robot while moving along the path is computed as

$$\mathcal{P} = \sum_{i}^{n} \frac{\mathbf{f}_i \cdot \mathbf{d}_i}{\Delta t_i}, \tag{3.3}$$

with $\mathbf{f}$ and $\mathbf{d}$ being, respectively, the applied force and displacement vectors, and Δt the time duration.

Fig.3.8-b describes the scene for the car-like robot. The position and orientation of the car is controlled by adding the torque to the wheels and to the steering. The path to the goal is blocked with the *free-mObjects* (blue cubes) and in order to reach the goal the car has to clear

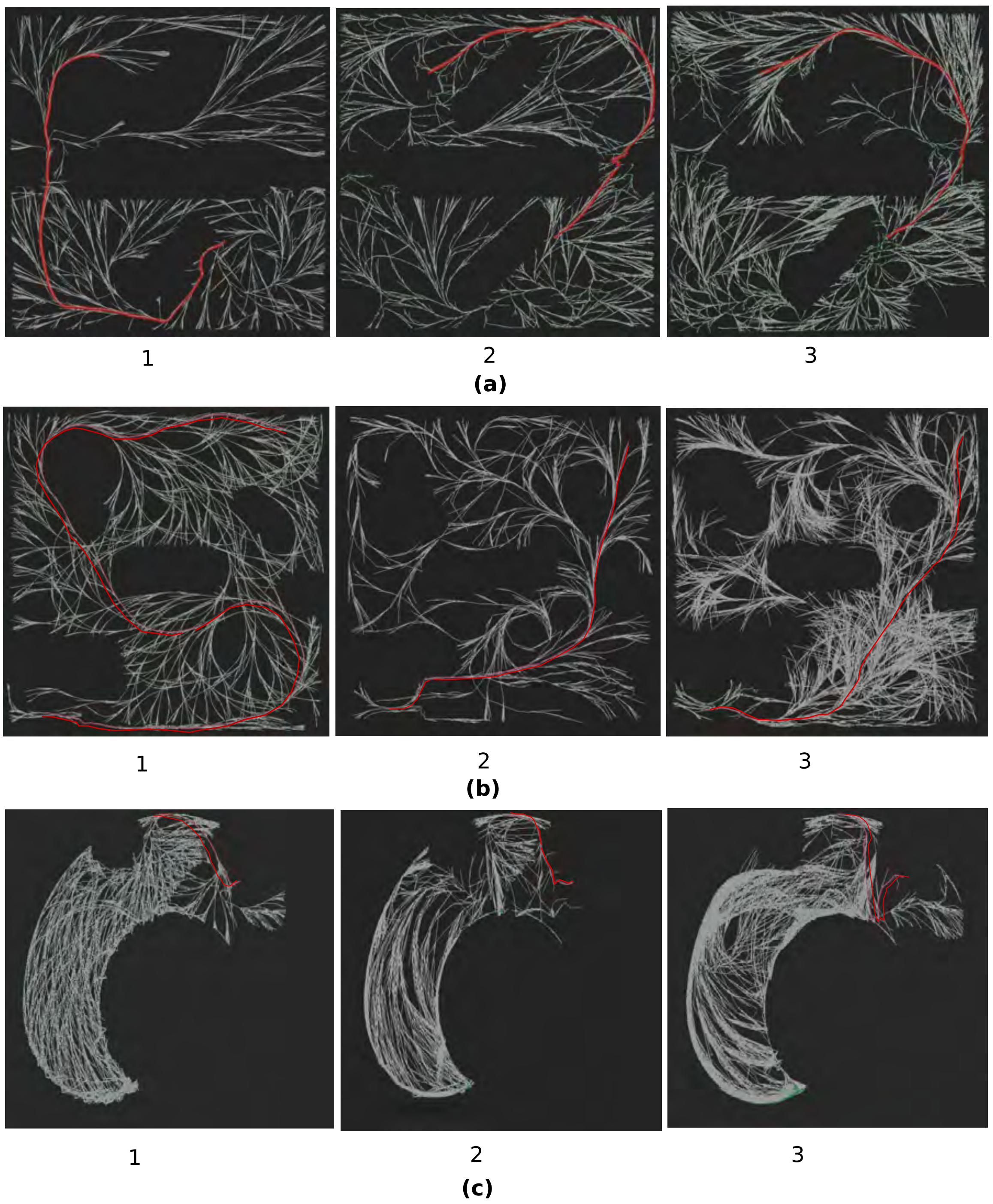


Figure 3.9: State space projection onto the workspace for the scenes with the holonomic mobile robot (top), the car-like robot (middle) and the planar manipulator (bottom) using: (1) κ-*KPIECE*, (2) κ-*RRT* and (3) κ-*SyCLoP* planners.

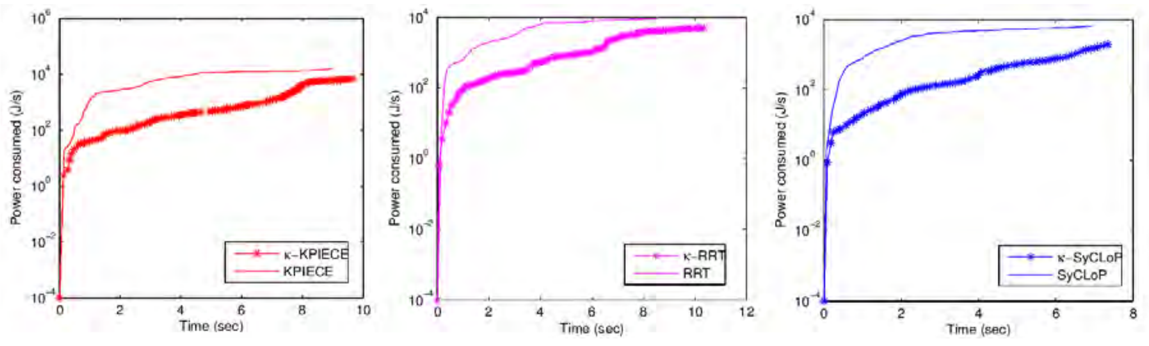


Figure 3.10: Logarithmic plot of the power consumed while moving along the path for the holonomic mobile robot.

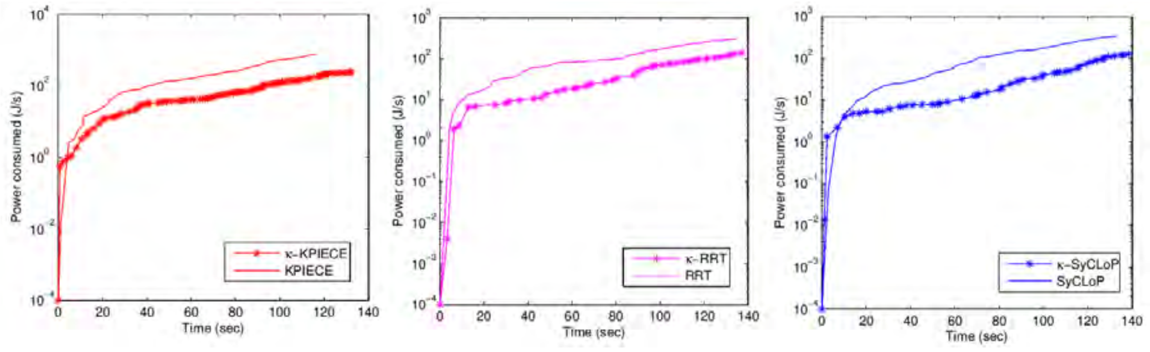


Figure 3.11: Logarithmic plot of the power consumed while moving along the path for the car-like robot.

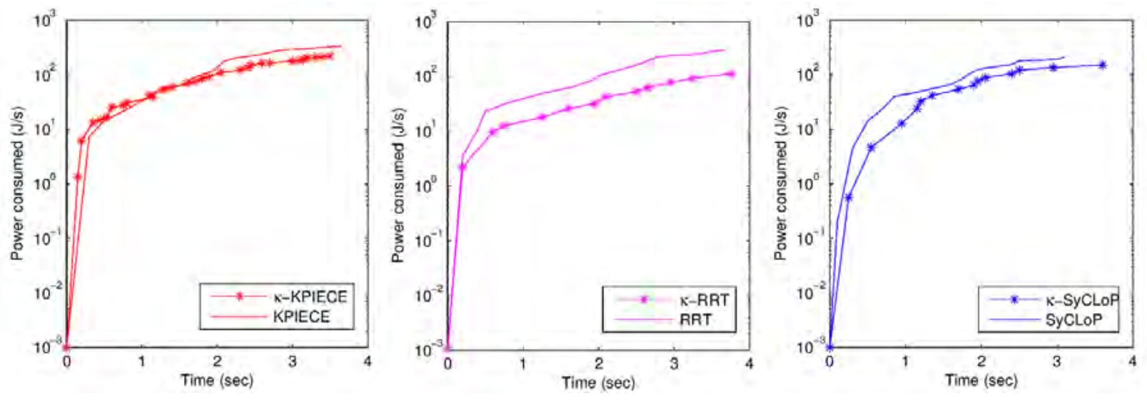


Figure 3.12: Logarithmic plot of the power consumed while moving along the path for the planar manipulator.

the way by pushing the objects away. The high amount of torque is only required while pushing the objects. The reasoning process updates the torque bounds by determining the forces that are required to push the object, and transforms it into the torque exerted by each wheel. The power consumed while moving along the path is computed as

$$\mathcal{P} = \sum_{i}^{n} \tau_i \cdot \omega_i, \tag{3.4}$$

where τ is the torque exerted by the wheels and ω is the corresponding angular velocity.

The scenario for the planar manipulator is depicted in Fig. 3.8-c, it consists of *free-mObject* (yellow and blue boxes), the target object (green box) and the *fixedObjects* (red cubes). The manipulator is shown in its initial state and the goal is to grasp the target object, being the way to that object blocked by the yellow box. Since one manipulation region of the yellow box is occupied with the target object, the reasoning process will change its type to *co-mObject* along y-axis making it manipulatable only (once all the manipulation regions of the yellow box will be free, its type will again update to *free-mObject*). In-order to reach the goal the manipulator has to push it away. The control forces are transformed into joint torques using the transposed Jacobian and the power is computed as

$$\mathcal{P} = \sum_{i}^{n} \tau_i \cdot \omega_i, \tag{3.5}$$

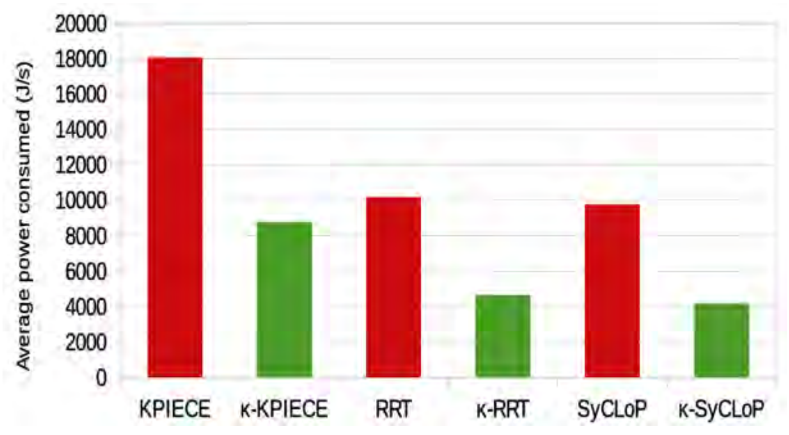
where now τ is the torque exerted by the joints and ω is the corresponding angular joint velocity. Fig. 3.9 depicts the configuration space for the κ-*KPIECE*, κ-*RRT* and κ-*SyCLoP* planners for the above stated scenarios.

3.7 Results and discussion

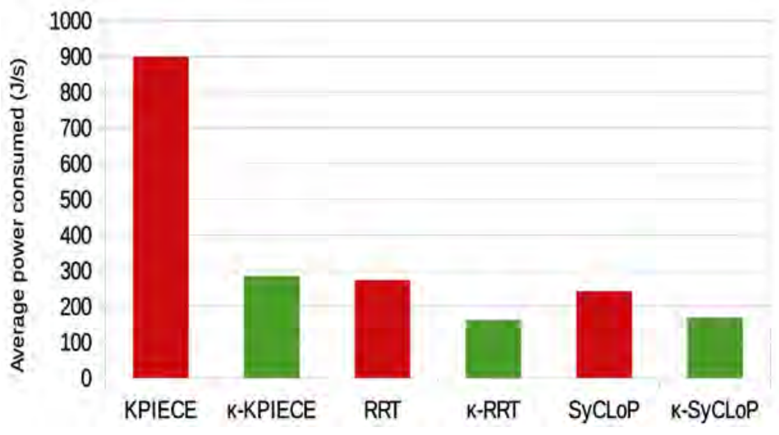
We compared κ-*PMP* with simple physics-based planning, using RRT, KPIECE and SyCLoP-RRT as kinodynamic motion planners, because chapter 2 showed these planners as being the most suitable for physics-based motion planning: KPIECE computed the efficient solutions in terms of time whereas SyCLoP and RRT computed the efficient solution in terms of power. No comparison has been done with other planners that seek different goals, such as (Haustein et al., 2015b; Stilman and Kuffner, 2005) that cope with the rearrangement of the objects in the workspace or such as (Li et al., 2016) that are focused on optimization issues.

The parameter used in the comparison were:

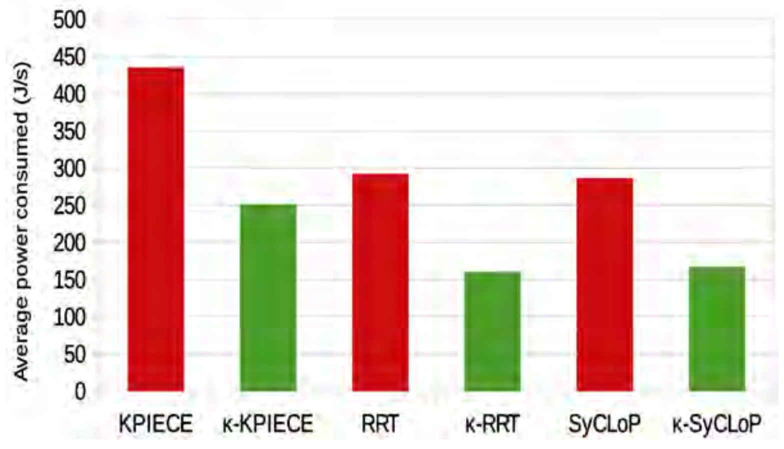
- *Power consumption:* The total power consumed by the robot while moving along the solution path.



(a)



(b)



(c)

Figure 3.13: Histogram of the average power consumed by the robots: (a) holonomic mobile robot, (b) car-like robot, (c) planar manipulator.

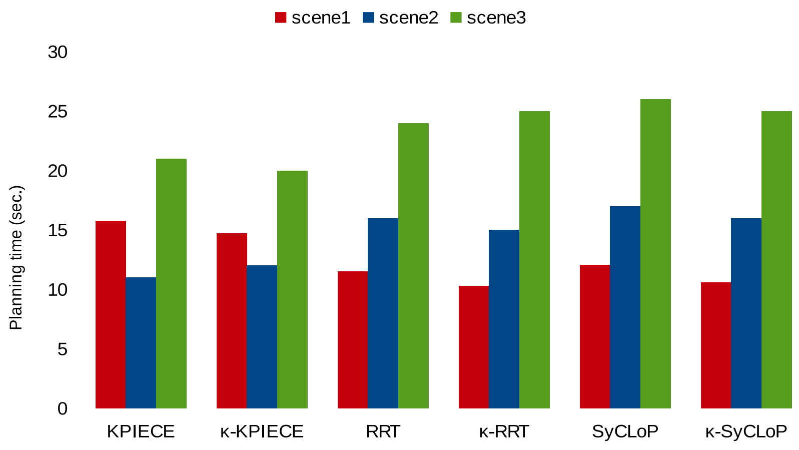


Figure 3.14: Histogram of the average planning time. Scene1, scene2 and scene3 correspond to the scenes illustrated in Fig. 3.8.

- *Planning time:* The total time wasted by the planner to compute the solution path.
- *Success rate:* The rate of successful runs in the maximum limit of planning time.

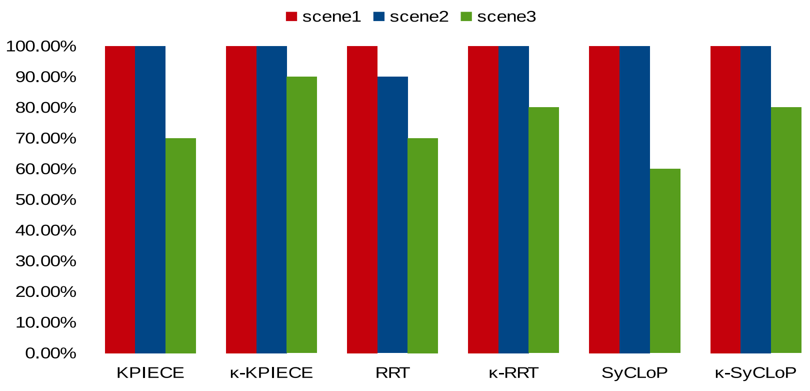


Figure 3.15: Histogram of the success rate. Scene1, scene2 and scene3 correspond to the scenes illustrated in Fig. 3.8

3.7.1 Quantitative analysis

κ-PMP computes the power efficient solution as compared to the simple physics-based motion planning approach. We compared the best results that we obtained using both approaches with different kinodynamic motion planners. Fig. 3.10, Fig. 3.11, and Fig. 3.12 show the logarithmic plot of power consumption vs time corresponding to the holonomic mobile robot, the car-like robot and the planar manipulator, respectively. In all cases *κ-PMP* is the most efficient plan in terms of power. The average power consumed by each planner in several runs is shown in the form of histogram (Fig. 3.13) corresponding to the three different robot models. There is a significant difference in terms of power consumption when using *κ-PMP* and simple physics-based planning approaches.

The average planing time of several runs for three scenes are presented in Fig. 3.14. *κ-PMP* computes the solution almost in the same time as the traditional approach . To compute the success rate, we set the maximum planning time to 150 s and each query is executed 10 times. Fig. 3.15 shows the histogram of the success rate where it can be appreciated that *κ-PMP* has a similar success rate as the traditional one but in some cases (particularly for the manipulator) our approach has a higher success rate than the traditional one.

3.7.2 Qualitative analysis

The analysis of the results shown in the previous section illustrates that *κ-PMP* preserves the behavior of the kinodynamic planner used (such as RRT or KPIECE) and significantly reduce the power consumed. This is due to the fact that *κ-PMP* dynamically varies the control sampling range according to the physical properties of the target object. Moreover, *κ-PMP* uses knowledge to determine the way to manipulate objects in order to reach the goal in a more realistic way. For instance, Fig. 3.16 shows a sequence of snapshots of an execution using the ontological physics-based motion planner (approach presented in the previous chapter). Since it lacks the evaluation of the object-object interaction and the post effect of dynamic interactions, the yellow box ends stuck in the gripper, thus preventing the manipulator to grasp the green box, reducing the success rate of the planner.

κ-PMP manipulates the objects in a natural way without putting any extra constraints (such as quasi static push). Fig. 3.17 shows the results of the dynamic interaction between the robot and the object using both approaches. In most of the cases ontological physics-based planning approach threw the object away (Fig. 3.17-a) because it is unaware of how much force is required to push. In contrast, Fig. 3.17-b shows the smoother interaction resulting from the use of *κ-PMP*. It applies the forces based on the physical properties (such as mass and friction) and pushes the object in a smooth way. Furthermore the solution path computed by the κ-PMP is naturally biased towards the manipulation regions, which helps to effectively manipulate the objects.

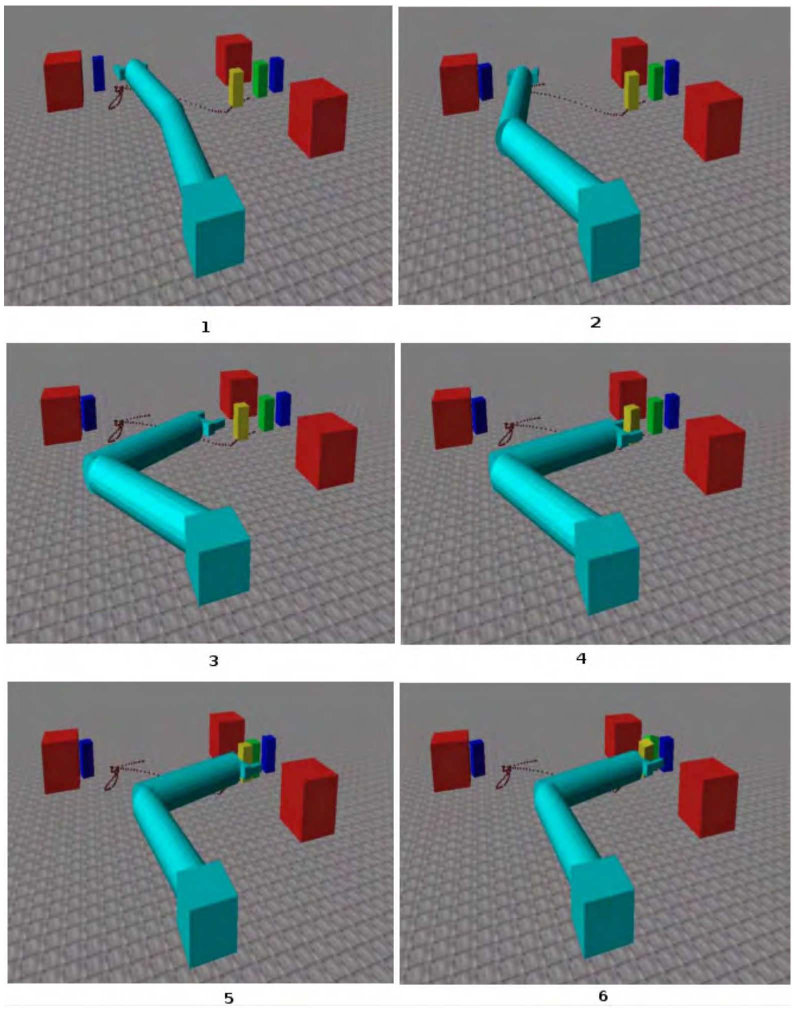


Figure 3.16: Execution of the computed motion plan by the ontological physics-based motion planner. It can be observed that the task fails because the manipulatable yellow object ends at the gripper, thus preventing the manipulator to grasp the target green object.

3.8 Practical application of the proposed approach

From the analysis done, it has been shown that the proposed approach computes more robust and power-efficient motion trajectories, as compared to the kinodynamic or simple physics-based planning approaches (mentioned in the related work). Moreover, the current proposal handles dynamic interaction in a more natural way. With a practical perspective, therefore, this approach may be significantly important in two directions. On the one hand, it can be a part of an integrated task and motion planner. The availability of power-efficient motion trajectories may have a relevant contribution in the final performance of the integrated task and motion planner, since the costs of the actions in a plan are critical and greatly influence the decisions of

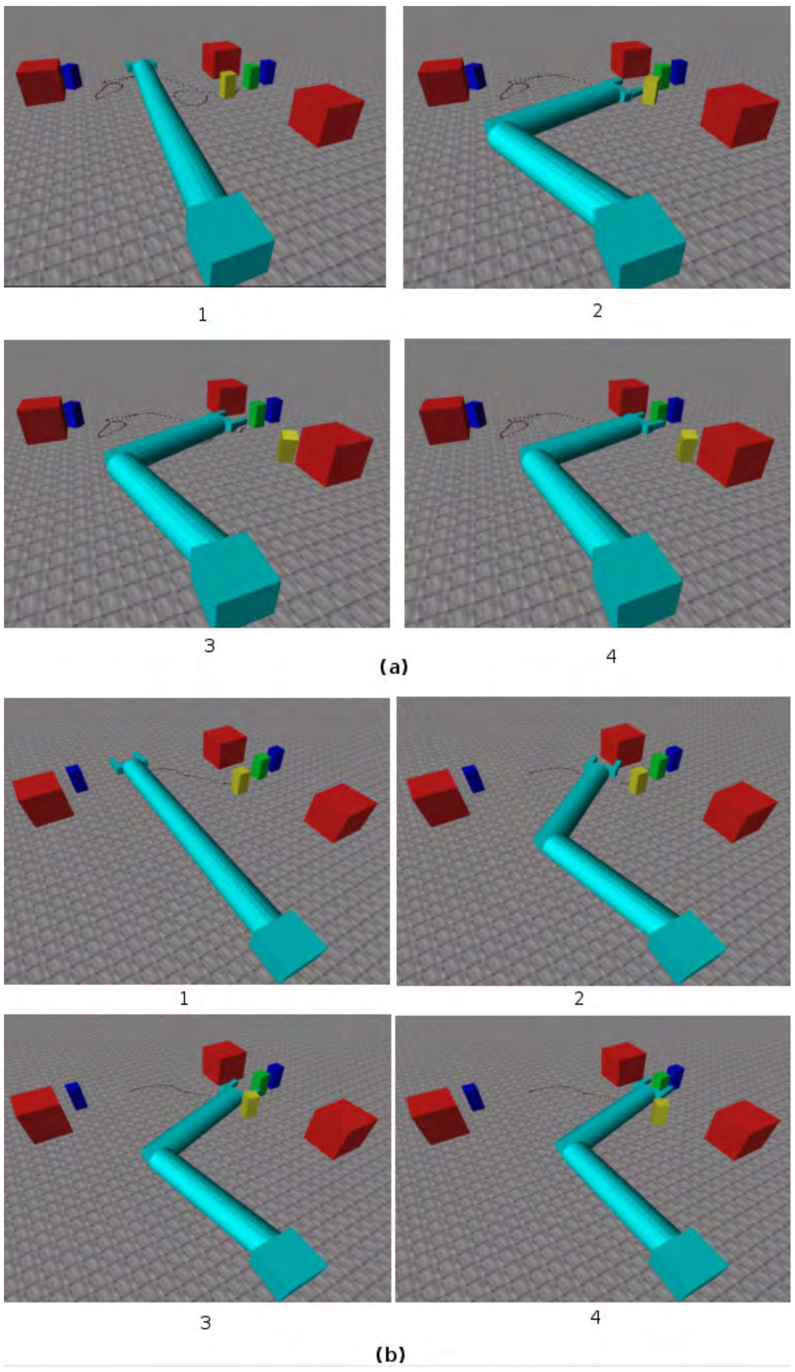


Figure 3.17: Snapshots of the dynamic interactions between the robot and the yellow object corresponding to the ontological physics-based motion planning (a) and to the κ-*PMP* planner (b), respectively. Videos: https://sir.upc.edu/projects/kautham/videos/k-PMP2.mp4

the task planner. On the other hand, as a stand alone planner, it results in a smart and powerful tool to manipulate objects in the clutter, without the need of reasoning at task level, giving practical solutions to quite difficult problems, as developed in next chapters.

3.9 Conclusions

This chapter has presented a framework, called *κ-PMP*, to use knowledge to enhance physics-based motion planners based on any kinodynamic algorithm, like RRT or KPIECE, and on any dynamic engine, like ODE or Bullet. Manipulation knowledge, inferred from an abstract knowledge ontology coded using the Web Ontology Language (OWL), is used to incorporate a reasoning process within the state transition model. This allows to dynamically update: a) the control sampling range based on the region of the configuration space where the robot is located and on the physical properties of the objects to be interacted; b) the regions from where an object can be manipulated. A comparison of physics-based motion planners based on RRT, KPIECE and SyCLoP with and without using the *κ-PMP* framework has been carried out in three different scenarios involving a holonomic mobile robot, a car-like robot and a planar manipulator. The proposal resulted in an improvement of the success rate and of the performance in terms of power consumed and quality of the solutions.

Chapter 4

Physics-based motion planning using temporal logic specification

4.1 Introduction

This chapter presents the framework to integrate the knowledge-oriented physics-based motion planning with linear temporal logic (LTL), originally presented in (Muhayyuddin et al., 2017b). It enables to define complex temporal goals, such as *visit region A followed by region C and avoid region B* using a LTL formula. LTL-based motion planning is a hybrid approach that provides a framework to describe complex motion planning tasks in terms of temporal goals, and that plans in discrete and continuous spaces. The planning is performed in three steps (1) *Workspace decomposition:* decomposes the robot workspace (using for example a triangular decomposition); (2) *High level planning:* constructs the discrete plan over the product space of the decomposed workspace and the automaton representing an LTL formula in such a way that the discrete plan satisfies the LTL formula; (3) *Low level planning:* implements the high level plan at low level in such a way that it also satisfies the formula.

LTL-based motion planning approaches are broadly divided into two main categories: controller based and sampling-based LTL motion planers. The former compute the discrete plan over the decomposed workspace and then the controller looks for the dynamically-feasible and collision-free trajectory for each action (Fainekos et al., 2009). The latter consider the integration of the task and motion planning steps, proposing a probabilistic search over the hybrid space of discrete and continuous components (Bhatia et al., 2010, 2011; Maly et al., 2013; Plaku et al., 2013; He et al., 2015). The discrete component is represented as the product space of decomposed workspace and the automaton that satisfies the LTL formula ϕ, whereas the continuous layer consists of a sampling-based dynamic motion planner that is guided by the discrete layer. All these approaches always seek for a collision-free trajectory. Although some approaches such as (McMahon and Plaku, 2014) incorporate a dynamic engine within an LTL framework, it is used only to consider the robot dynamics and the physics-based constraints, i.e. no dynamic interactions between rigid bodies are considered while planning. The straight extension of these approaches in order to handle manipulation actions is possible, but due to the high complexity of physics-based motion planning (large search space and highly constraint solution set) along with the incorporation of temporal constraints, may lead to a computationally nontractable problem, raising the question about its decidability. Therefore, an efficient and powerful framework is required that has the capacity to handle both the temporal goals and the physics-based constraints, along with the purposeful manipulation of the objects if necessary.

Contributions. The main contributions of this chapter are: *(1)* the integration of the Linear Temporal Logic within the framework of ontological physics-based motion planning, thus allowing the purposeful manipulation of objects, like the execution of push actions to clear regions possibly occupied by objects; *(2)* the proposal of an LTL feasibility evaluation and simplification process, that uses knowledge-based reasoning to evaluate whether the properties of the objects and the robot make the task described by the LTL formula feasible or not and, if required, simplifies the formula by skipping the non-valid propositions defined with disjunction relations.

4.2 Problem setup to handle temporal goals

Let $\mathcal{X}$ be the state space of all the bodies in the environment. At any time t a state $x \in \mathcal{X}$ is represented as $x(t) = \{s_1 \dots s_k\}$ where s_i represents the position and orientation of the i-th object in the environment.

Consider a mobile robot $\mathcal{R}$, and let $\mathcal{S}$ be its state space containing all possible states of the robot. A state $s \in \mathcal{S}$ is represented as $s = \{p, o, v, w\}$ where $p, o, v,$ and w are the position, orientation, linear velocity and angular velocity respectively. The state of the environment can be represented as $E = \mathcal{X} \times \mathcal{S}$. The instantiated knowledge for E (introduced in Sec. 3.4.2) is defined as $\kappa_t = \kappa_t^s \cup \kappa_t^x$, where κ_t^x is the part associated to $\mathcal{X}$ and κ_t^s is the part associated to $\mathcal{S}$.

A trajectory of the robot is defined by the robot dynamics in the results of control inputs, that are applied for a small time duration Δt. It can be written as $s_{\text{new}} = \text{PROPAGATOR}(s, u, \Delta t)$, where $u \in \mathcal{U}$ is a control input from the control space $\mathcal{U}$ containing the set of all possible control inputs that can be applied to the robot. The PROPAGATOR will generate a trajectory between state s and s_{new}. An entire trajectory (*Traj*) of the robot is obtained by applying the control inputs (starting from the start state) repeatedly for small time durations. During the execution of the motion, if the robot interacts with *free-mObjects* or *co-mObjects* (types of objects are explained in Sec. 2.2) the resulting motion will change the state of the bodies. It implies that control inputs are responsible for updating the state of the robot as well as the state of the objects. Generally, the transition function PROPAGATOR can be written as PROPAGATOR: $E_i \times \mathcal{U} \to E_{i+1}$. The validity of newly generated states is evaluated by a function VALIDITYCHECKER: $E \times \kappa_t \to \{\text{true}, \text{false}\}$. It returns true iff the newly generated state satisfies all the constraints imposed by κ_t and false otherwise. For each new state of E, κ_t is updated by the inference process INFERENCE: $E_{i+1} \times \kappa_i \rightarrow \kappa_{i+1}$.

4.3 Linear temporal logic

Linear temporal logic is a formalism used to specify tasks by combining propositions with logic and temporal operators. The combination is called an LTL formula ϕ (Clarke et al., 1999), i.e. an LTL formula ϕ is defined by integrating propositions with the logic operators *negation* ($\neg$), *conjunction* ($\wedge$), *disjunction* ($\vee$), *equivalence* ($\Leftrightarrow$), and *implication* ($\Rightarrow$), along with the temporal operators *next* ($\bigcirc$), *always* ($\Box$), *until* ($\sqcup$), and *eventually* ($\Diamond$).

Let Π be the set of atomic propositions, $\Pi = \{\pi_1, \ldots, \pi_n\}$, each π_i representing a statement such as *"robot is in region P_i"*. Every $\pi_i \in \Pi$ is a formula, and if ϕ and ψ are formulas, then new formulas can be defined using the following grammar:

$$\neg\phi,\ \ \phi \wedge \psi,\ \ \phi \vee \psi,\ \ \Diamond\phi,\ \ \phi \sqcup \psi,\ \ \phi \bigcirc \psi$$

As an example, the formula to visit the regions $P_1,\ P_2,\ P_3$ in an ordered way can be represented as $\phi = \Diamond(\pi_1 \wedge \Diamond(\pi_2 \wedge \Diamond(\pi_3)))$

The semantics of LTL formulas are defined over infinite traces of a system. Let $\sigma = \tau_0, \tau_1, \ldots \tau_\infty$ represent an infinite trace with $\tau_i \in 2^{\Pi}$, then the expression $\sigma \models \phi$ will be used to represent that σ satisfies ϕ iff there exists a finite prefix $\sigma_i = \tau_0, \tau_1, \ldots, \tau_{i-1}$ of σ that satisfies ϕ.

Syntactically co-safe formulas are a special class of LTL formulas. When written in positive normal form (i.e. when the negation operator occurs only in front of atomic propositions), they only contain the eventually, next and until operators. They can be interpreted over a finite trace and their validity can be checked using nondeterministic finite automata (Kupferman and Vardi, 2001). This is the type of formulas used when the focus is in motion planning problems over a finite time horizon.

4.4 LTL semantics for motion trajectories

The robot workspace $\mathcal{W}$ contains different types of rigid bodies and a set of propositional regions $P = \{P_1 \ldots P_n\}$ corresponding to the propositions $\{\pi_1 \ldots \pi_n\}$. The part of the workspace that is accessible by the robot is represented as $\mathcal{W}_{\text{acc}} = \mathcal{W} \setminus \mathcal{W}_{\text{fixed}}$, where $\mathcal{W}_{\text{fixed}}$ is the part of the workspace occupied by *fixed-objects*. Propositional regions are associated with accessible parts of the workspace $P \in \mathcal{W}_{\text{acc}}$. An extra proposition π_0 is associated with a propositional region

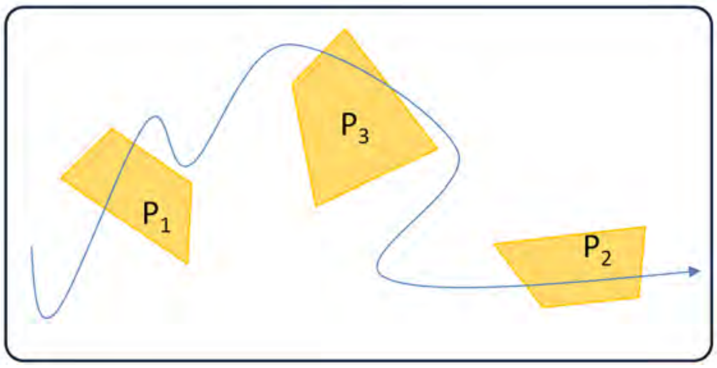


Figure 4.1: $tr(\mathit{Traj})$ generates the word $[\pi_0, \pi_1, \pi_0, \pi_3, \pi_0, \pi_2, \pi_0]$. The trajectory satisfies the formula if the word finished at the accepting state of the automaton.

such that $P_0 = \mathcal{W}_{\text{acc}} \setminus \cup_{i=1}^{n} P_i$. A function $\mathcal{G} : \mathcal{W}_{\text{acc}} \rightarrow \Pi$ maps each point of the workspace over a propositional region.

The discrete trace of a trajectory *Traj* is defined as the sequence of propositional regions that are traversed by *Traj* and is represented as $tr(\mathit{Traj})$. A propositional region P_i is said to be traversed iff $\mathcal{G}(\mathit{Traj}(t)) = \pi_i$ for some $0 \leq t \leq T$. A motion trajectory *Traj* satisfies ϕ iff $tr(\mathit{Traj}) \models \phi$ as depicted in Fig. 4.1.

4.5 Problem statement

Let a motion planning problem with temporal goal be considered as the tuple

$$\langle E_{\text{init}}, \text{PROPAGATOR}, \text{VALIDITYCHECKER}, \mathcal{K}, \kappa_t, \text{ INFERENCE}, \Pi, \phi \rangle.$$

Temporal goals are expressed in terms of an LTL formula ϕ, defined over a set of atomic propositions Π that describe the regions of the workspace that the robot must either visit or avoid. The problem is to evaluate (by taking into account $E_{init}, \mathcal{K}, \Pi, \phi$) whether the robot can access all the propositional regions necessary to satisfy the formula, if not, whether the formula can be simplified by removing the unaccessible propositional regions. If the formula (or possibly simplified formula) is feasible then the problem is to find a sequence of control inputs such that the resulting robot trajectory *Traj* satisfies $tr(\mathit{Traj}) \models \phi$ and is dynamically feasible, avoids the collision with fixed bodies and, if necessary, pushes movable objects away for clearing the regions.

It is important to note that the feasibility evaluation and simplification is different from the validity checking of the LTL formulas that is performed using model-checking techniques

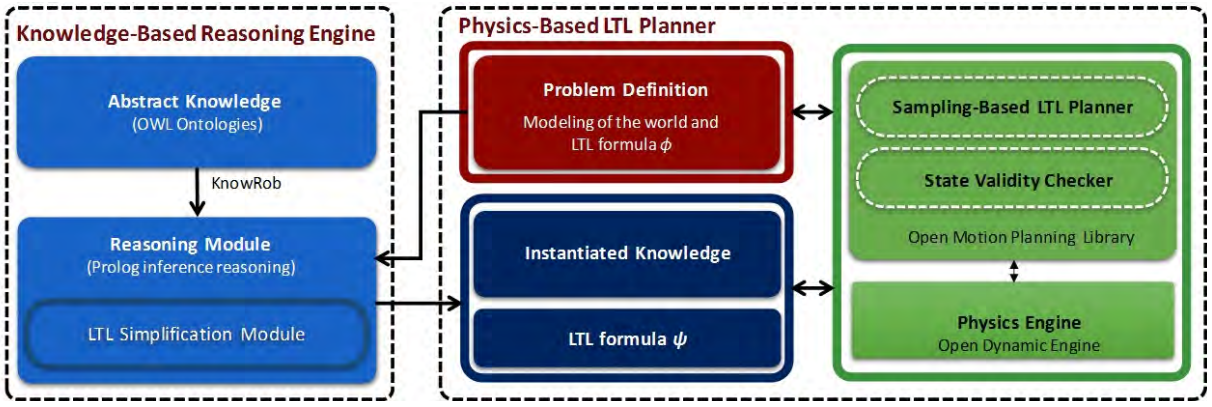

Knowledge-Based Reasoning Engine
Abstract Knowledge
(OWL Ontologies)
KnowRob
Reasoning Module
(Prolog inference reasoning)
LTL Simplification Module
Physics-Based LTL Planner
Problem Definition
Modeling of the world and
LTL formula ϕ
Instantiated Knowledge
LTL formula ψ
Sampling-Based LTL Planner
State Validity Checker
Open Motion Planning Library
Physics Engine
Open Dynamic Engine

Figure 4.2: Framework for physics-based LTL motion planning.

such as automaton generation. A formula is said to be valid if it can be transformed to a valid automaton.

4.6 Framework for physic-based LTL motion planning

To solve the above stated problem, a physics-based LTL motion planning approach is proposed that makes use of ontologies and high-level reasoning, and that considers physics-based motion propagation. The schematic representation of the solution framework is depicted in Fig. 4.2. It consists of two main modules: the knowledge-based reasoning engine and the physics-based LTL planner. The former is responsible for defining the manipulation constraints, the feasibility evaluation and the possible simplification of the LTL formula, whereas the latter computes the motion plan that satisfies the temporal goals.

The knowledge-based reasoning engine contains the abstract knowledge $\mathcal{K}$, and performs a prolog-based reasoning over $\mathcal{K}$ to define the types of the rigid bodies and the associated manipulation constraints (encoded in the instantiated knowledge) using the kinematic and dynamic properties of the robot and the bodies. Furthermore, it is responsible for the feasibility evaluation and the possible simplification of the LTL formula (explained in Sec. 4.6.1).

The physics-based LTL planner stores the problem definition (the world modeling along with the initial LTL formula ϕ that defines the temporal goals), the instantiated knowledge κ and the LTL formula ψ inferred by the reasoning module (ψ contains a simplified version of ϕ, when possible). The automaton will be constructed for ψ, unlike other sampling-based LTL planners that always plan for ϕ. The sampling-based LTL motion planner is responsible for generating the discrete plan (computed over the product space of the decomposed workspace and generated automaton for ψ) and its execution at low-level (continuous motion planning level) to determine the control sequence in such a way that the resultant trajectory satisfies ψ. The state

Algorithm 4: Simplify

```
inputs : List $L$ with nonvalid proposition(s), LTL formula $\phi$
output: LTL formula $\phi$
1 if $L.op \ni \vee$ then
2     return $\phi \backslash L$
3 else
4     if $L.parent = \phi$ then
5         return NULL
6     else
7         Simplify($L.parent, \phi$)
```

propagator makes use of the physics engine for the propagation, and the newly generated states are evaluated by the state validity checker. Different from standard sampling-based LTL planners, the proposed validity checker takes into account the current state of the environment and evaluates it based on the instantiated knowledge that is valid for that particular state.

4.6.1 Reasoning process

The aim of knowledge-based reasoning is to provide autonomy to the robots for performing complex tasks. We use the reasoning process, on one hand, to generate the instantiated knowledge κ and, on the other hand, to evaluate the feasibility of the LTL formula and its potential simplification.

For the generation of κ_t a prolog-based reasoning is employed that reads the abstract knowledge in order to classify the objects into different types, together with their manipulation constraints. This process is performed by evaluating physical properties of the objects and the kinodynamic properties of the robot in a similar way as explained in Chapter 3. At each instant of time κ_t is updated using INFERENCE function (Sec. 4.2). For instance, if after a propagation step one manipulation region is occupied with another object, the motion constraints of the first object are updated accordingly to the new situation.

To evaluate the feasibility and the possible simplification of the LTL formula, the reasoning process is done as follows:

- Let an LTL formula ϕ be defined over a set of propositions $\Pi = \{\pi_1 \dots \pi_n\}$, where each $\pi_i \in \Pi$ is associated with a region P_i of the workspace (that the robot should visit or avoid) called *propositional region*. A proposition is considered nonvalid if the associated propositional region is not accessible by the robot and valid otherwise. Let $\mathcal{M}_p$ be the list of nonvalid propositions and $\mathcal{F}$ be the function that computes them, i.e. $\mathcal{F} : \mathcal{K} \times \Pi \Rightarrow \mathcal{M}_p$. If $\mathcal{M}_p = \emptyset$, the formula is feasible and does not require simplification.
- Let a formula be considered a list L, which can be either an atomic list (defined by a

Algorithm 5: Evaluate

```
inputs : LTL formula $\phi$, Set of propositions $\Pi$, Knowledge $\mathcal{K}$
output: $\phi$ or NULL
1 $\mathcal{M}_p \leftarrow \mathcal{F}(\mathcal{K}, \Pi)$
2 if $\mathcal{M}_p =$ NULL then
3     return $\phi$
4 while $\mathcal{M}_p \neq 0$ do
5     $\phi \leftarrow$Simplify($\pi.L, \phi$)
6     Update($\mathcal{M}_p$)
7     if $\phi =$ NULL then
8         return NULL
9 return $\phi$
```

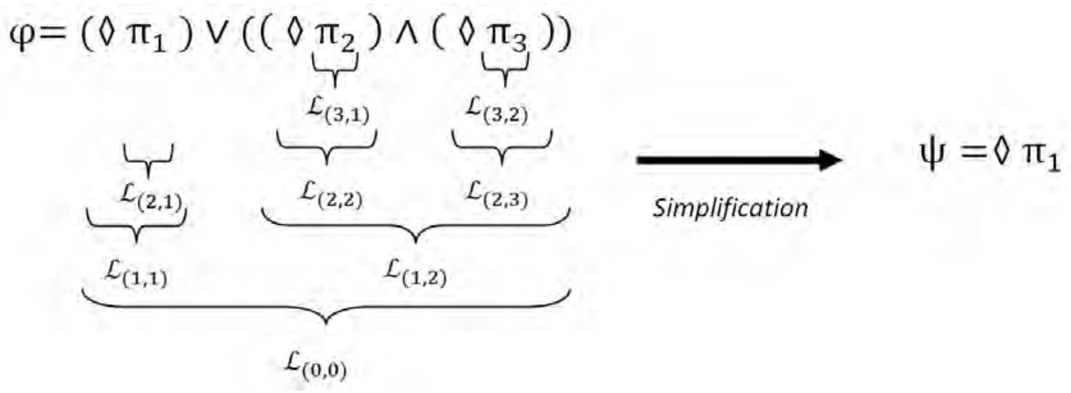


Figure 4.3: Example of the simplification process, where ϕ and ψ are the actual and the simplified formulas respectively. Each $\mathcal{L}_{(i,j)}$ is a list with i and j representing the depth and the order in the parent list, respectively.

single proposition), or a compound list (defined by the grammar introduced in Sec. 4.3). Then, $L.parent$ will indicate the parent list of a list, $L.op$ the set of prefix and postfix operators of L within $L.parent$, and $\pi.L$ the innermost list containing the proposition π (i.e. $\pi.L = \{\pi\}$).

- Let Simplify(L, ϕ) be a recursive function that verifies if L contains disjunction operators and if so returns the formula ϕ without L, as shown in Algorithm 4. Then, Algorithm 5 shows the procedure Evaluate(ϕ) that checks the feasibility of an LTL formula by using function Simplify(L, ϕ) applied to list of non-valid proposition. At each iteration $\mathcal{M}_p$ is updated by removing the proposition that are deleted by the Simplify function.

As an example, consider the formula $\phi = (\Diamond\pi_1) \vee ((\Diamond\pi_2) \wedge (\Diamond\pi_3))$ where the proposition π_3 is assumed to be nonvalid. The list associated to π_3 is $\mathcal{L}_{(3,2)}$, as shown in Fig. 4.3. The recursive Simplify function is initially called for $\mathcal{L}_{(3,2)}$, then for $\mathcal{L}_{(2,3)}$ and $\mathcal{L}_{(1,2)}$, when the $\vee$ operator is

Algorithm 6: Physics-based LTL Motion Planning

```
inputs : Initial state $E_{\text{init}}$, $\Pi$, LTL formula $\phi$, Threshold $T_{max}$
output: A continuous path that satisfies $\phi$.
1  $\mathcal{K} \leftarrow$ OntologyFormulation($E_{init}$)
2  $\kappa_0 \leftarrow$InstantiatedKnowledgeInference($\mathcal{K}$)
3  $\psi \leftarrow$ Evaluate($\mathcal{K}, \Pi, \phi$)
4  if $\psi$ = NULL then
5      return NULL
6  else
7      $\mathcal{T} \leftarrow$InitializeTree($E_{init}$)
8      $\mathcal{A}_\psi \leftarrow$ComputeAutomaton($\psi$)
9      $\mathcal{D} \leftarrow$ ComputeDecomposition() ; $j = 0$
10     while $t < T_{max}$ do
11         $\rho \leftarrow$ DiscretePlanning($\mathcal{A}_\psi$,$\mathcal{D}$ )
12         $v \leftarrow$ SelectHighLevelState($\rho$)
13         $\{u, n\} \leftarrow$ SampleControlAndSteps($v$)
14         for $i = 0$ to $n$ do
15             $E_{\text{new}} \leftarrow$ PROPAGATOR($E, u$,$\Delta t$)
16             if ! VALIDITYCHECKER($E_{new}$,$\kappa_j$) then
17                 Break
18             else
19                 $\kappa_{j+1} \leftarrow$ INFERENCE($E_{new}, \kappa_j$); $j = j + 1$
20                 $v_{new} \leftarrow$UpdateHighLevelState($v$)
21                 $\mathcal{T} \leftarrow$ UpdateTree($E_{new}, u, \Delta t$)
22                 $z \leftarrow$GetAutomatonState($v_{new}$)
23                 if $z \in$ Accepting state of $\mathcal{A}_\psi$ then
24                     return Traj$\leftarrow$RetrieveTrajectory($\mathcal{T}$)
25     return NULL
```

found. At this moment the formula is simplified by deleting $\mathcal{L}_{(1,2)}$ and the simplified formula results $\psi = \Diamond\pi_1$.

The proposed reasoning process works for the syntactically co-safe LTL formulas. The Open Motion Planning Library (OMPL), a C++ based tool for sampling-based motion planning, provides the implementation of the sampling-based LTL motion planner presented by Bhatia et al. (2010). It supports the temporal goals defined using syntactically co-safe LTL formulas such as $\phi = \Diamond\pi_1 \wedge \cdots \wedge \Diamond\pi_n$ (visit all the regions $\pi_1 \ldots \pi_n$ in any order), $\phi = \Diamond(\pi_1 \wedge \Diamond(\pi_2 \wedge \Diamond(\ldots \wedge \Diamond\pi_n)))$ (visit all the regions in an ordered way), $\phi = \Diamond\pi_1 \vee \cdots \vee \Diamond\pi_n$ (visit any of the region from $\pi_1 \ldots \pi_n$), and $\phi = (\Diamond\pi_1 \vee \Diamond\pi_2) \wedge \neg\pi_3$ (visit π_1 or π_2 and do not visit π_3). The reasoning process works over similar types of formulas with similar grammar.

4.6.2 Planning process

The planning process is explained in Algorithm 6, that takes as inputs the initial state, the set of propositions Π, the temporal goal (defined in terms of an LTL formula ϕ over Π), and the maximum allowed planning time T_{max}. As output it returns a continuous path (as a sequence of controls and durations) that satisfies ϕ.

The *OntologyFormulation* function defines the abstract knowledge $\mathcal{K}$ about the world by defining the types of the objects (such as fixed or manipulatable), and their manipulation constraints in terms of *mRegions*(Sec.4.2). *InstantiatedKnowledgeInference* fills the initial state of the instantiated knowledge as explained in Sec. 4.6.1. To determine the feasibility of ϕ, the *Evaluate* function computes the feasibility and performs the possible simplification (if required) as explained in Algorithms 4 and 5. The function *InitializeTree* sets the initial state of the tree as the initial state of the environment.

Lines: (8-13) refer to the general steps of the sampling-based LTL motion planning, as done in (Bhatia et al., 2010), for the high level planning and for the updating of the states (both at low- and high- levels). *ComputeAutomaton* function computes the automaton $\mathcal{A}_\psi$ for the formula ψ, *ComputeDecomposition* performs the triangular decomposition of the workspace not occupied by fixed obstacles, in a way that preserves the propositional regions. As a difference with (Bhatia et al., 2010) we only exclude from the decomposition the workspace occupied by fixed obstacles, i.e. the non-fixed bodies are simply ignored while decomposing the workspace. Function *DiscretePlanning* constructs the discrete plan over the product space of the decomposition and $\mathcal{A}_\psi$, and *SelectHighLevelState* selects the high-level state to be explored. At low level, *SampleControlAndSteps* function samples the controls (that could be a vector of applied forces, joints torques, or velocities), and the number of steps (that refer to the number of times that the sampled controls will repeatedly applied for a duration Δt).

The function PROPAGATOR applies the sampled controls for Δt time on the robot and generates new state of the environment E_{new} using the dynamics engine that allows to handle all the kinodynamic and physics-based constraints. VALIDITYCHECKER evaluates the newly generated state of the environment, based on the instantiated knowledge κ_t. E_{new} will be accepted if it satisfies all the constraints (such as temporal constraints, kinodynamic and physics-based constraints) that are imposed by κ_t and discarded otherwise.

The INFERENCE function updates the instantiated knowledge with the manipulation constraints that are valid for E_{new}. *UpdateHighLevelState* updates the high-level state based on the result of the low-level state and *UpdateTree* updates the tree-data structure. The *GetAutomatonState* function will determine the state of the automaton, if it is the accepting state of $\mathcal{A}_\psi$, the *RetrieveTrajectory* function returns the *Traj* that is a continues trajectory for the robot such that $tr(\mathit{Traj}) \models \phi$.

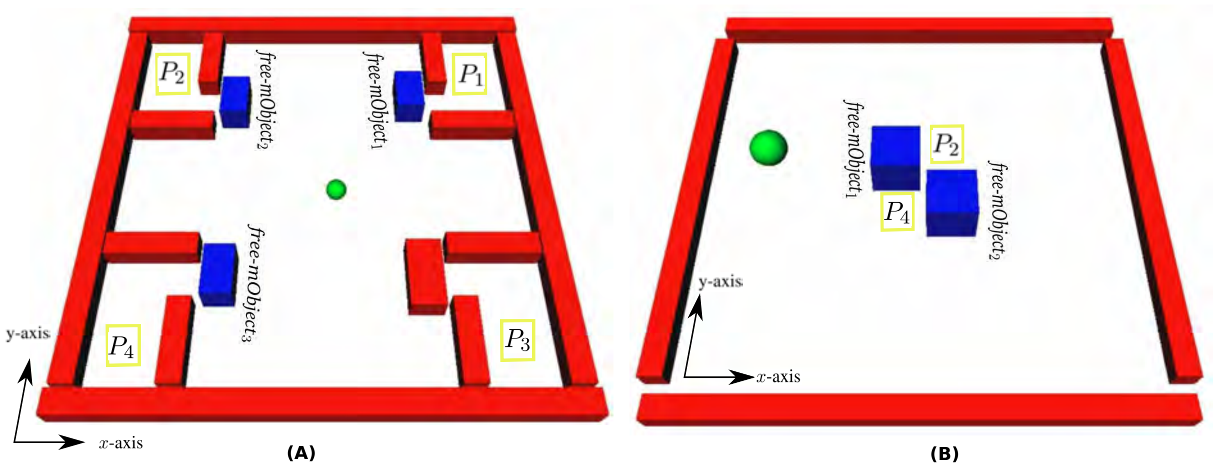


Figure 4.4: Example scenarios A: the goal is to visit the propositional regions $P_1, \cdots, P_4$, being the access to them obstructed by the blue boxes. B: visit all propositional regions in an ordered way, being two of them occupied by bodies. Video: https://sir.upc.edu/projects/kautham/videos/IFAC2017.mp4

4.7 Results and discussion

The simulation setup consists of a robot (green sphere), *co-mObjects* (blue cubes), and *fixed-Objects* (red walls). There are two scenarios presented in Fig. 4.4. In the first scenario, it is assumed that all the propositional regions are surrounded by the objects that could be fixed or movable and no collision free trajectory exist to visit each region. It could be considered as different rooms that robot has to visit but in order to enter each room, the robot has to interact with the freely-movable objects blocking the entrances. The propositional regions $\{P_1 \ldots P_4\}$, are shown as yellow rectangles. The temporal goal is defined by the LTL formula $\phi = \diamondsuit(\pi_1 \wedge \diamondsuit(\pi_2 \wedge \diamondsuit(\pi_3 \vee \pi_4)))$ that is visit P_1, P_2 and then P_3 or P_4 Region associated with P_3 is surrounded with *fixed-Objects* and, therefore, the reasoning process marks π_3 as invalid (it is not accessible by the robot). Since it has disjunction relation with the other propositions, the simplification process will simplify the formula to $\psi = \diamondsuit(\pi_1 \wedge \diamondsuit(\pi_2 \wedge \diamondsuit(\pi_4)))$. Since the length of *free-mObject*$_1$ is greater than the entrance, at its current location its manipulation region along the x-axis of the world frame is occupied with the walls. Therefore, the reasoning process will change the status of the object from freely-movable to constraint-oriented movable, and only allow the robot to push it along the y-axis. If after pushing the body, all the manipulation regions become free, the INFERENCE function will change the type from constraint-oriented to freely movable again. The similar process is applied for *free-mObject*$_2$ and *free-mObject*$_3$.

The temporal goal for the second scenario is described as $\phi = \diamondsuit(\pi_1 \wedge \diamondsuit(\pi_3 \wedge \diamondsuit(\pi_2 \wedge \diamondsuit(\pi_4))))$. That is, visit P_1, P_3, P_2 and P_4 consecutively. Regions associated to P_1 and P_3 are occupied by *free-mObject*$_1$ and *free-mObject*$_2$ respectively. Therefore, in order to visit P_1 (without prior being on P_2 or P_4), the robot must push *free-mObject*$_1$ along the x-axis or $-y$-axis of the world

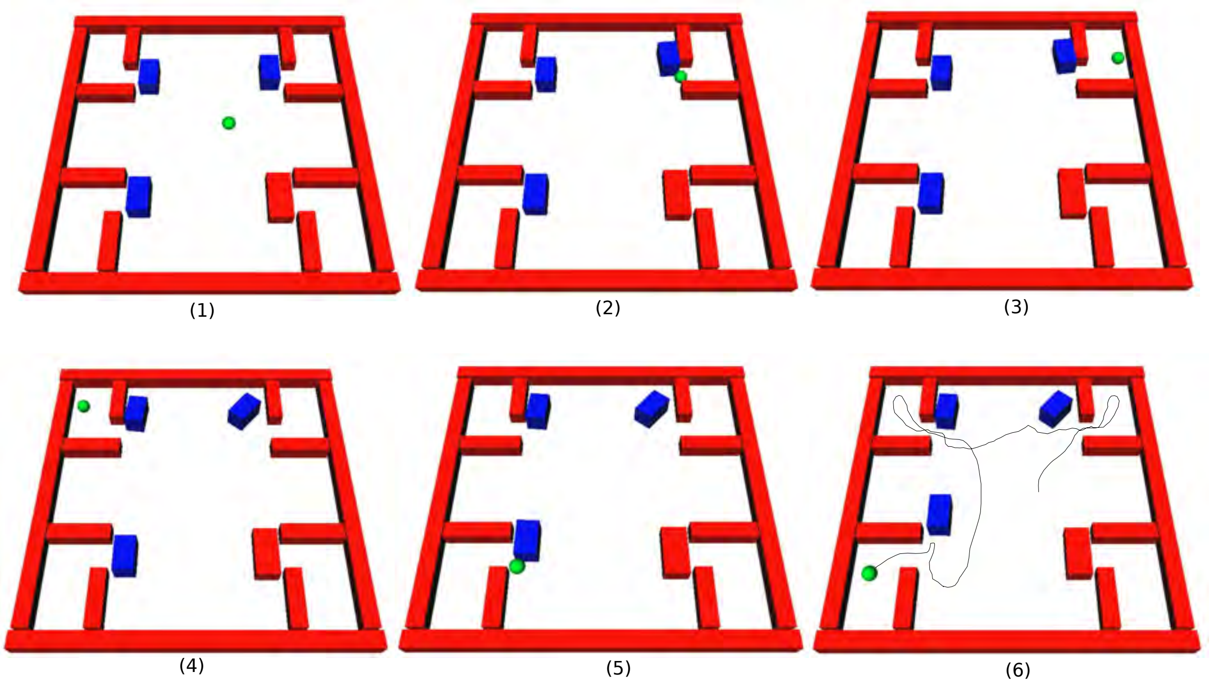


Figure 4.5: Sequence of snapshots the execution of first scenario.

frame. If it pushes *free-mObject*$_1$ along the x-axis then it ends occupying the manipulation region of *free-mObjects*$_2$ that is along y-axis and hence the reasoning process will deactivate the *mRegion* along the $-y$-axis and change the status of the body to constraint-oriented movable. The same reasoning process is repeated for the second body, i.e. The INFERENCE function will update the types of these objects to *co-mObject*$_1$ and *co-mObject*$_2$. To visit P_3, *co-mObject*$_2$ can only be pushed along the $-x$-axis. After visiting P_3 the types of the objects will be restored to *free-mObjects* and the task can continue.

These two examples show that the proposed approach is able, on the one hand, to deal with movable objects that may be obstructing the solution path (changing if necessary the way the robot has to interact with them) and, on the other hand, is able to simplify a formula if part of it is non-feasible. The sequence of the snapshots of the execution is depicted in Fig. 4.5 and Fig. 4.6.

We tested both scenarios with and without instantiated knowledge. The simulation was performed on an Intel Core i7-4500U 1.80GHz CPU with 16 GB memory. For the first scene, the success rate of simple physics-based LTL planner was 30% for 10 runs (maximum allowed time was 300 seconds) whereas the success rate of the proposed approach was 80%. The simple physics-based LTL approach has an average planning time of 230 seconds. In contrast, the proposed approach computes the solution in 46.4 seconds (average of 10 runs).

For the second scenario, the success rate of the proposed approach and the simple physics-

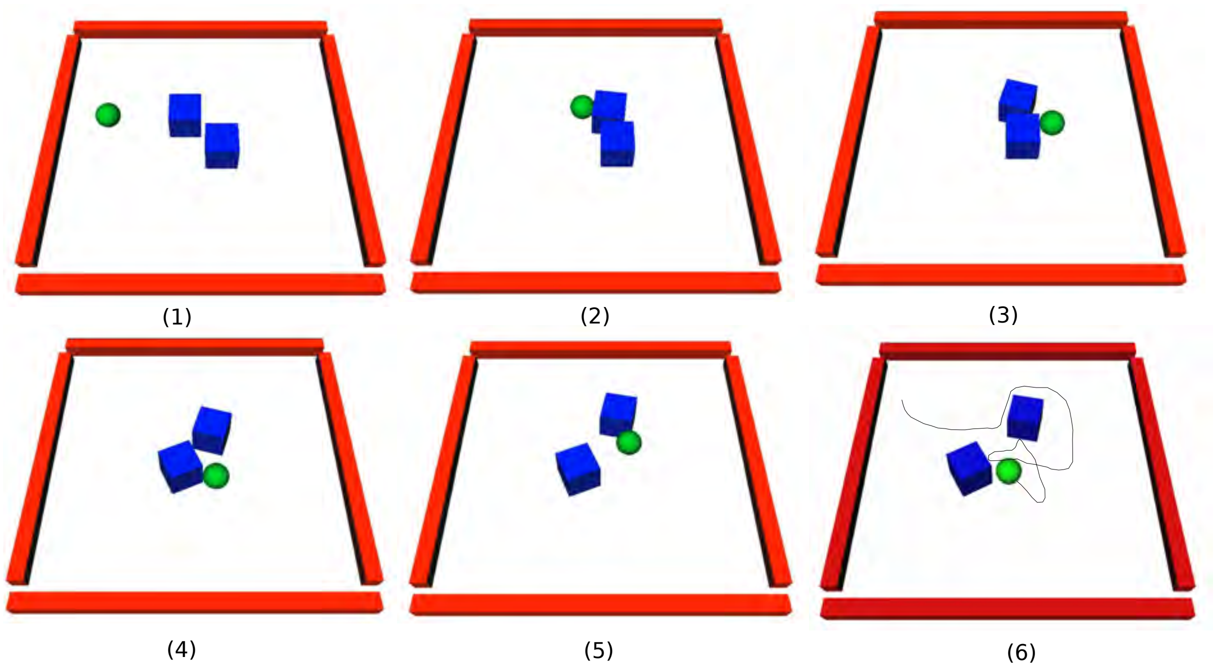


Figure 4.6: Sequence of snapshots the execution of second scenario.

based approach were 100%. But, in the case of the proposed approach the quality of the solution was better, it avoids the unnecessary interactions between the robot and the objects and move the objects only when it is necessary. Regarding planning time, the proposed approach computes the solution in 2.1 seconds (average of 10 runs) and the simple physics-based planning approach takes 23.8 seconds.

4.8 Conclusions

This chapter has presented the integration of LTL planning within the framework of ontological physics-based motion planning in order to provide robustness and autonomy for handling complex temporal goals in a realistic way. Moreover, a simplification process of the LTL formula is proposed, according to the validity or not of the goals to be satisfied and the logical operators involved. The proposed approach has been validated using simulation examples in which some of the propositional regions are occupied with objects or the way to the propositional region is blocked with objects that the robot has to push away, if possible, in order to visit the regions. The results shows that the integration of knowledge makes the planner more efficient and enhance the quality of the solution.

Part II

Planning under uncertainty

Chapter 5

Robust-RRT: A motion planning approach for grasping under uncertainty

5.1 Introduction

Grasping an object in uncertain environments is a challenging task, particularly when a collision-free trajectory does not exits. High-level knowledge and reasoning processes, as well as the allowing of interaction between objects (as presented in previous chapters), can enhance the planning efficiency in such environments. This chapter presents a physics-based motion planning approach (for a hand-arm system) for grasping under objects' pose uncertainty, which is a key issue to be considered when dynamic interactions occur, and that is important to be taken into account during planning. This work is originally presented in (Muhayyuddin et al., 2018a) that extends ontological physics-based motion planning approach (presented in chapter 2) by considering robot manipulators, the reasoning on the post effects of dynamic interactions (both robot-object and object-object interactions), and the presence of uncertainty. It is based on a kinodynamic RRT planner with a dynamic engine as state propagator that allows to consider any type of motion (not only rectilinear motions of the gripper) and any type of interaction (robot-object and object-object). The planner includes a high-level knowledge about the scene, stored in the form of ontologies, that is used to provide a semantic description of the scene and information on how the robot may interact with the obstacles. Moreover, a smart state-validity checker is introduced that evaluates the post-effect of dynamic interactions and computes a confidence index that provides the belief about the robustness of the selected controls in the presence of uncertainty. The proposal is called *robust-RRT*. It is validated for different scenarios in simulation and in real environment using a 7-degree-of-freedom KUKA Lightweight robot equipped with a two-finger gripper. The results show a significant improvement in the success rate of the execution of the computed plan in the presence of object pose uncertainty.

5.2 Problem formulation

Consider a hand-arm trajectory planning problem to grasp a target object in an uncertain environment. The environment E is composed of a robot $\mathcal{R}$ and a set of objects $\mathcal{O}$ lying on an horizontal flat surface. Let $\mathcal{X}$ represent the state space of the robot, it is a differential manifold. The state $x \in \mathcal{X}$ of the robot at any time t can be represented as $x(t) = (\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$, where $\boldsymbol{q}_t$ and $\dot{\boldsymbol{q}}_t$ represent the configuration of the robot and its time derivative, respectively. The objects in the environment are classified into a set of fixed objects $\mathcal{O}^f$ and a set of manipulatable objects $\mathcal{O}^m$, that includes the target object $\mathcal{O}^m_{\text{target}}$. The robot is only allowed to interact with the manipulatable objects. Then, objects are represented as $\mathcal{O} = \{\mathcal{O}^m_1 \dots \mathcal{O}^m_{\text{M}}, \mathcal{O}^f_1 \dots \mathcal{O}^f_{\text{F}}\}$ where M and F represent the number of manipulatable and fixed objects, respectively. Let $\mathcal{S}$ be the state space of the objects in the environment. Then, at any time t, the state of the objects can be parametrized as $s(t)=\{s_1, \dots, s_{\text{M}}\}$ where $s_i \in \mathcal{S}$ has a component $\mathbf{p}_i \in SE(2)$ representing the pose, and another one $\mathbf{v}_i$ representing the linear and angular velocities of each object at time t. The state of the environment is then described as $E = \mathcal{X} \times \mathcal{S}$, and the dynamic state propagator $\boldsymbol{f}$ is defined as;

$$\boldsymbol{f} : E_i \times \mathcal{U} \longrightarrow E_{i+1}, \tag{5.1}$$

where, $\mathcal{U}$ is the control space containing all possible controls that can be applied to the system. In this work, controls will be represented in terms of joint velocities.

Consider a situation where the path to reach the target object is blocked with one or several objects in such a way that a collision-free trajectory from the start to a goal configuration does not exist. Then, the objective is to plan the grasp approach trajectory by computing the sequence of controls and corresponding duration in such a way that if sequentially applied to the robot (in the presence of object pose uncertainty), it moves the robot from the start to the (pre-grasping) goal state.

Uncertainty in the knowledge of the actual initial poses due to the sensory noise will be considered. The pose uncertainty region will be modelled as $\boldsymbol{U}_{\text{E}_{\text{init}}} = \mathcal{N}(\mathbf{p}^{\text{init}}, \boldsymbol{\nu}^{\text{init}})$, i.e. a multivariate Gaussian distribution with mean $\mathbf{p}^{\text{init}} = (\mathbf{p}^{\text{init}}_1, \dots, \mathbf{p}^{\text{init}}_{\text{M}})$, the measured initial pose, and variance $\boldsymbol{\nu}^{\text{init}} = (\nu^{\text{init}}_1, \dots, \nu^{\text{init}}_{\text{M}})$. The uncertainty will be propagated to future states as a result of robot-object and object-object interactions. Then, the controls to be selected will be those that, when applied to any actual pose within the pose uncertainty region, are robust enough to bring the system to a new valid state, with a probability higher than a given threshold.

The proposed approach works in two main phases that are the pre-processing phase and the planning phase. The former involves a knowledge-based representation of the environment to provide the semantic description of the scene and a reasoning process that partitions the workspace into regions, define manipulation constraints for the objects, and provides the detailed insight about the task. The latter consists of a sampling-based kinodynamic motion planner that uses the high-level description of the problem (computed in the pre-processing phase) to plan efficiently. Moreover, it proposes a strategy to cope with object pose uncertainty. The integration of these two phases enables the robot to plan robustly in complex environments.

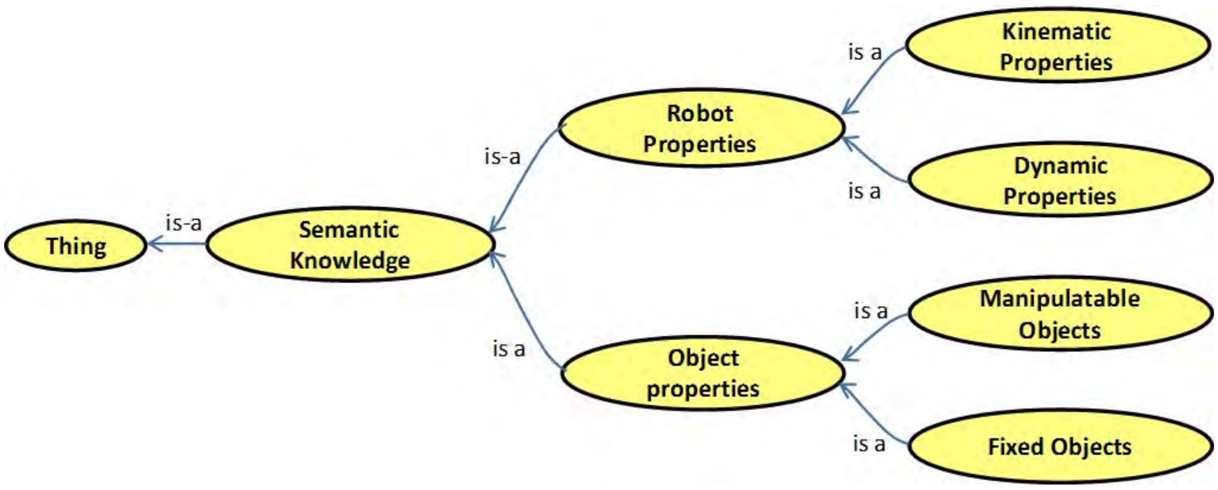


Figure 5.1: OWL semantic knowledge taxonomy.

5.3 Reasoning process

To model and represent semantic knowledge, an ontology-based approach like the one presented in chapter 3 is used. The knowledge regarding the robot as well as obstacles properties is stored in two classes using OWL (Fig. 5.1):

- *Robot properties*: It includes the robot kinematic properties (such as joint limits and collision model) and dynamic properties (such as masses and bounds on forces and velocities), as well as the properties of the gripper.
- *Obstacles Properties*: It involves properties of obstacles such as the poses of fixed and manipulatable objects, their masses, friction coefficients, and manipulation constraints (represented in terms of manipulation regions).

The reasoning process, summarized in Algorithm 7, is performed over this semantic knowledge, called $\mathcal{K}_S$, in order to specify the *target region* as well as the *manipulation region* for each of the manipulatable objects. The target region ($\mathcal{O}^m_{TRgn}$) is determined as a box region around $\mathcal{O}^m_{\text{target}}$, and it is considered as the part of the workspace where efforts to find a robust plan are focused. We assumed that the interactions with the objects that are outside the $\mathcal{O}^m_{TRgn}$ will not effect the grasping process. Therefore, the uncertainty will be considered only for $\mathcal{O}^m_{TRgn}$. The manipulation regions (*mRgn*) are defined as the regions around the manipulatable objects from where the robot can interact with the obstacles (in this respect, for instance, these regions will be located below the center of mass for tall and thin objects). This spatial reasoning process is carried out over the knowledge ontologies described in OWL using the Prolog language, with the following predicates:

Algorithm 7: ReasoningProcess($\mathcal{K}_\text{s}, \mathcal{O}^m_\text{target}, E_\text{init}$)

```
1  $\mathcal{K}_\text{m} \leftarrow \emptyset$
2  $TRgn \leftarrow$compute_target_region($\mathcal{O}^m_\text{target}, E_\text{init}$)
3  $\{\mathcal{O}^m_{TRgn}\}\leftarrow$objects_in_target_region($TRgn$)
4  $\{mRgn\}\leftarrow$compute_manipulation_regions($\mathcal{K}_\text{s}$)
5  return $\mathcal{K}_m$.add($TRgn$, $\{\mathcal{O}^m_{TRgn}\}$, $\{mRgn\}$)
```

Algorithm 8: ConfidenceIndex($\boldsymbol{U}_{\text{E}_\text{near}}, u_{\Delta t}, \mathcal{K}_\text{m}$)

```
1  $\mathcal{E}_\text{near} \leftarrow \emptyset$, $\mathcal{E}_\text{new} \leftarrow \emptyset$, validstatecounter $= 0$
2  for $i = 0$ to $n$ do
3      $E^\text{trial}_\text{near} \leftarrow$ SampleState($\boldsymbol{U}_{\text{E}_\text{near}}$)
4      $E^\text{trial}_\text{new} \leftarrow$ Propagate($E^\text{trial}_\text{near}, u_{\Delta t}$)
5      if StateValidityChecker($E^{new}_{trial}, \mathcal{K}_m$) then
6          validstatecounter $=$ validstatecounter $+ 1$
7          $\mathcal{E}_\text{near}$.Add($E^\text{trial}_\text{near}$)
8          $\mathcal{E}_\text{new}$.Add($E^\text{trial}_\text{new}$)
9  $c =$ validstatecounter$/n$
10 return $\{c, \mathcal{E}_{near}, \mathcal{E}_{new}\}$
```

- *compute_target_region:* Given $\mathcal{O}^m_\text{target}$ and E_init, it computes the location of the target object and applies spatial reasoning to compute the target region (*TRgn*). It will be modeled as a box centered at the target object with the size of the sides depending on the robot and the objects.

- *objects_in_target_region:* Given a *TRgn*, it returns the set $\mathcal{O}^m_{TRgn}$ of manipulatable objects it contains (including $\mathcal{O}^m_\text{target}$).

- *compute_manipulation_region:* It takes the semantic knowledge $\mathcal{K}_\text{s}$ as input and defines the manipulation regions based on the properties of the objects.

The output of the reasoning process is encapsulated as the manipulation knowledge $\mathcal{K}_\text{m}$ that will be used by the motion planner for computing the plan.

5.4 Handling pose uncertainty

This section describes: *1)* the evaluation of the outcomes of dynamic interactions done by the validity checker taking into account uncertainty, *2)* the strategy to define the robustness of controls according to a set of possible resultant states, and *3)* the propagation of the uncertainty into the future states as a result of the dynamic interactions.

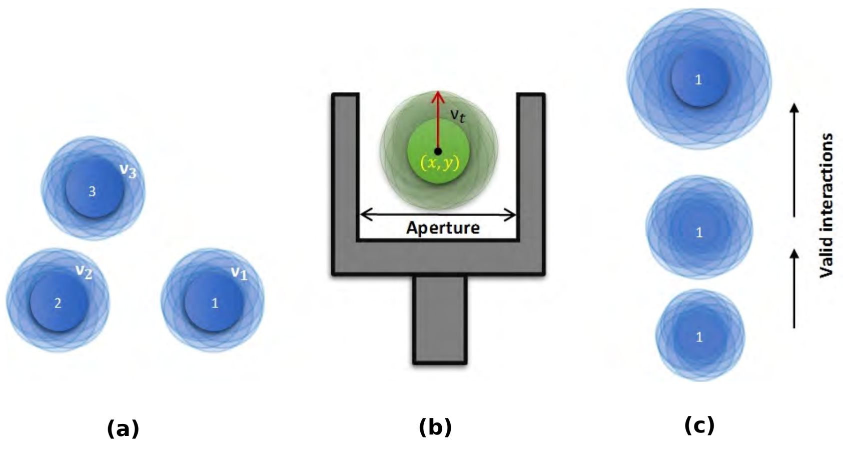


Figure 5.2: (a) Initial pose uncertainty of three objects; (b) Pose uncertainty of the target object assumed to fit within the gripper aperture; (c) Enlarging of the pose uncertainty of an object being displaced due to the results of interactions.

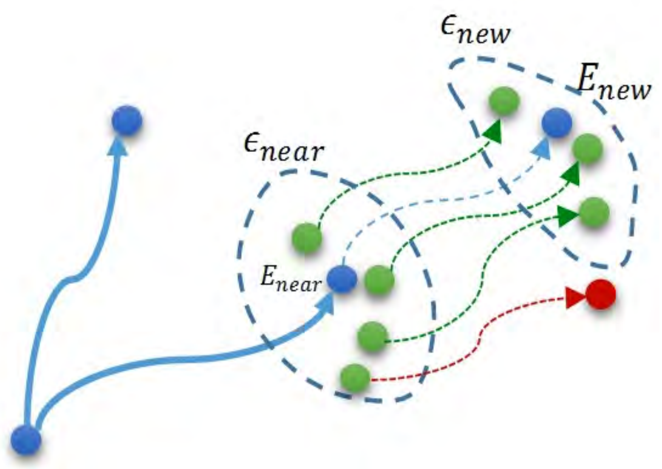


Figure 5.3: Graphical representation of the computation of the confidence index for the control leading from E_{near} to E_{new}, where valid resultant states are shown in green whereas invalid ones in red.

The initial pose uncertainty $\boldsymbol{U}_{\text{E}_{\text{init}}}$ is propagated due to dynamic interactions, i.e. the mean and the deviation vary due to the objects that are moved due to interactions. As an example, Fig. 5.2-a shows the initial pose uncertainty of three objects (as the projection of $\boldsymbol{U}_{\text{E}_{\text{init}}}$ onto the corresponding xy-planes), Fig. 5.2-c the enlarging of the pose uncertainty as the object is displaced due to an interaction, and Fig. 5.2-b the pose uncertainty of the target object (it is assumed that the gripper aperture is big enough to envelope it, and this will always hold since the planning procedure will avoid the interaction with the target object).

5.4.1 State validity checker

The proposed state-validity checker allows the collision with manipulatable objects, while rejects the collision with fixed objects. The result of an interaction with a manipulatable object will be valid, however, only if the interaction takes place when the robot gripper is located at the object manipulation region, the velocity at the contact is below a given threshold (to prevent unwanted large motions of the objects), and no object collide with the target object or falls from the table.

5.4.2 Control evaluation

The effects of applying a given control when interactions occur depend on many factors, like friction, pressure distributions under the object supporting surface, momentum and inertial effects, as well as on the actual pose of the objects (within the uncertainty region) when the control is applied. Therefore, the use of the state propagator dynamics engine is proposed to evaluate its eligibility when applied to a given state E_{near}. The procedure, sketched in Algorithm 8, relies on applying the control (during a given time duration) from a set of n states sampled from the uncertainty region associated to E_{near} (Fig. 5.3), and evaluating the validity of the resulting states. The probability of obtaining a valid resultant state is estimated by the ratio of valid resultant states with respect to the total, and is called confidence index. The main functions used in Algorithm 8 are:

- *SampleState:* Varies the poses of the objects of a given nominal state E_{near} by sampling them from the corresponding pose uncertainty region $\boldsymbol{U}_{\text{E}_{\text{near}}} = \mathcal{N}(\mathbf{p}^{\text{near}}, \boldsymbol{\nu}^{\text{near}})$, and returns the resultant state, called $E^{\text{trial}}_{\text{near}}$.
- *Propagate:* Returns a new state $E^{\text{new}}_{\text{trial}}$ by applying Eq. (5.1) to state $E^{\text{trial}}_{\text{near}}$ using ODE as state propagator, which takes care of all the kinodynamic and physics-based constraints.
- *StateValidityChecker:* Checks the validity of the new state $E^{\text{new}}_{\text{trial}}$ as described above.

The set of sampled states, called $\mathcal{E}_{\text{near}}$, and the set of resulting of valid states, called $\mathcal{E}_{\text{new}}$, together with the confidence index are returned for further processing.

5.4.3 Uncertainty region update

The uncertainty region is updated upon the occurrence of robot-object or object-object interactions, as sketched in Algorithm 9. The following functions are used:

- *displacedObjects:* Evaluates for the states in the set $\mathcal{E}_{\text{new}}$ if the objects poses are different from the corresponding states in $\mathcal{E}_{\text{near}}$, and returns the number of times this is true.

Algorithm 9: UpdateUncertaintyRegion($\mathcal{E}_{\text{near}}, \mathcal{E}_{\text{new}}, \boldsymbol{U}_{\text{E}_{\text{near}}}$)

```
1 if displacedObjects($\mathcal{E}_{near}$, $\mathcal{E}_{new}$) != NULL then
2     $\mathbf{p}^{\text{new}} \leftarrow$ ComputeMeanPose($\mathcal{E}_{\text{new}}$)
3     $\boldsymbol{\nu}^{\text{new}} \leftarrow$ ComputeDeviation($\mathcal{E}_{\text{new}}, \mathbf{p}^{\text{new}}$)
4     return $\mathcal{N}(\mathbf{p}^{new}, \boldsymbol{\nu}^{new})$
5 else
6     return $\boldsymbol{U}_{E_{near}}$
```

- *ComputeMeanPose:* Computes for each object i the mean pose $\mathbf{p}_i^{\text{new}}$, from the pose $\mathbf{p}_i^{E_j}$ of each state $E_j \in \mathcal{E}_{\text{new}}$, i.e.:

$$\begin{aligned} \mathbf{p}^{\text{new}} &= (\mathbf{p}_1^{\text{new}}, \ldots, \mathbf{p}_{\text{M}}^{\text{new}}) \\ \mathbf{p}_i^{\text{new}} &= \frac{1}{|\mathcal{E}_{\text{new}}|} \sum_{\forall E_j \in \mathcal{E}_{\text{new}}} \mathbf{p}_i^{E_j} \end{aligned} \tag{5.2}$$

- *ComputeDeviation:* Computes for each object i the deviation of the poses of the object for the states in $\mathcal{E}_{\text{new}}$, i.e.:

$$\begin{aligned} \boldsymbol{\nu}^{\text{new}} &= (\nu_1^{\text{new}}, \ldots, \nu_{\text{M}}^{\text{new}}) \\ \nu_i^{\text{new}} &= \sqrt{\frac{1}{|\mathcal{E}_{\text{new}}|} \sum_{\forall E_j \in \mathcal{E}_{\text{new}}} [\mathbf{p}_i^{E_j} - \mathbf{p}_i^{\text{new}}]^2} \end{aligned} \tag{5.3}$$

It is important to note that the uncertainty region is only updated if interactions take place, otherwise the uncertainty region of the prior state, $\boldsymbol{U}_{\text{E}_{\text{near}}}$, is returned.

5.5 Robust Rapidly-exploring Random Tree (R-RRT)

For planning the trajectory, an RRT planner is modified to handle the object pose uncertainty. As explained in chapter 1, it is a sampling-based kinodynamic motion planner that efficiently explores high-dimensional state spaces (LaValle and Kuffner, 2001). The work principle of the planning algorithm is to grow a tree rooted at a start state. For the growth, a state $\boldsymbol{x}_{\text{rand}}$ is randomly sampled, and the node of the tree that is nearest to $\boldsymbol{x}_{\text{rand}}$ is selected as $\boldsymbol{x}_{\text{near}}$. From that state, randomly sampled controls are applied for a randomly sampled time duration, and the validity-check is run at each step to find collision-free paths. The control that generates a state closest to $\boldsymbol{x}_{\text{rand}}$ is chosen. The newly generated node is called $\boldsymbol{x}_{\text{new}}$ and is added as a vertex, and the control as the edge connecting $\boldsymbol{x}_{\text{near}}$ to $\boldsymbol{x}_{\text{new}}$. The process continues until a state is found within the goal region, or a predefined threshold planning time has elapsed. The present proposal modifies this algorithm by:

Algorithm 10: Robust-RRT

```
inputs : Semantic knowledge $\mathcal{K}_S$, Initial state $E_{\text{init}}$, Initial pose uncertainty region $\boldsymbol{U}_{E_{\text{init}}}$, Target
         object $\mathcal{O}_m^{\text{target}}$, Time threshold $T_{\max}$, Bias index $\mathcal{B}$, Robustness threshold $R_{\text{threshold}}$
output: A sequence of controls to move the robot to a pre-grasp goal configuration.
1  $\mathcal{K}_m \leftarrow$ReasoningProcess($\mathcal{K}_S, \mathcal{O}_m^{\text{target}}, E_{\text{init}}$)
2  $\mathcal{R}_{\text{goal}} \leftarrow$ComputeEndEffectorGoalRegion($\mathcal{O}_m^{\text{target}}$)
3  $\mathcal{T}$.init($E_{\text{init}}$)
4  while $planning\_time < T_{max}$ do
5      $\boldsymbol{p}_{\text{rand}} \leftarrow$SampleRandomPose($\mathcal{K}_m, \mathcal{B}$)
6      $E_{\text{near}} \leftarrow$SelectNodeToExpand($\mathcal{T}, \boldsymbol{p}_{\text{rand}}$)
7      $\{E_{\text{new}}\ u_{\Delta t}\} \leftarrow$SearchControls($E_{\text{near}}, \mathcal{K}_m$)
8      $\{c, \mathcal{E}_{\text{near}}, \mathcal{E}_{\text{new}}\} \leftarrow$ConfidenceIndex($\boldsymbol{U}_{E_{\text{near}}}, u_{\Delta t}, \mathcal{K}_m$)
9      if $c > R_{threshold}$ then
10         $\mathcal{T}$.AddVertex($E_{\text{new}}$)
11         $\mathcal{T}$.AddEdge($E_{\text{near}}, E_{\text{new}}, u_{\Delta t}$)
12         $\boldsymbol{U}_{E_{\text{new}}} \leftarrow$ UpdateUncertaintyRegion($\mathcal{E}_{\text{near}}, \mathcal{E}_{\text{new}}, \boldsymbol{U}_{E_{\text{near}}}$)
13         if $GetEndEffectorPose(E_{new}) \in \mathcal{R}_{goal}$ then
14             return $Path \leftarrow RetrievePath(\mathcal{T})$
15 return NULL
```

a) Introducing a steering method based on the pose of the end-effector, i.e. to steer the tree growth a pose $\mathbf{p}_{\text{rand}}$ of the end-effector is randomly sampled and the node of the tree to be expanded is the one that corresponds to a configuration of the robot with the end-effector nearest to $\mathbf{p}_{\text{rand}}$. Moreover, a sampling bias is considered towards configurations that place the origin of the gripper reference frame within the target region (a bias index $\mathcal{B} \in [0, 1]$ determines the ratio of the configurations explicitly sampled within the target region).

b) Introducing a knowledge-based state-validity checker to reject states that may lay to a failed execution (detailed in Sec. 5.4.1).

c) Introducing a confidence index to filter out those controls not robust enough (detailed in Sec. 5.4.2).

d) Updating the pose uncertainty when interactions occur (detailed in Sec. 5.4.3).

The planning process for the proposed approach is outlined in Algorithm 10. As an input it takes the semantic knowledge stored in the form of ontologies $\mathcal{K}_S$, the initial state of the environment E_{init} that contains the initial state of the robot and the objects, the initial pose uncertainty region $\boldsymbol{U}_{E_{\text{init}}}$, the target object $\mathcal{O}_m^{\text{target}}$, the time threshold $T_{\max}$, the bias index $\mathcal{B}$, and the robustness threshold $R_{\text{threshold}}$. As an output it returns a trajectory as a sequence of controls along their duration to move the robot to a pre-grasp configuration (that locates the robot end-effector in the goal region $\mathcal{R}_{\text{goal}}$). The key parts of the proposed algorithm are:

1. *Initialization* (lines 1-3): It includes the reasoning process to determine the manipulation knowledge, as detailed in Sec. 5.3, and the determination of the region $\mathcal{R}_{\text{goal}}$ where the

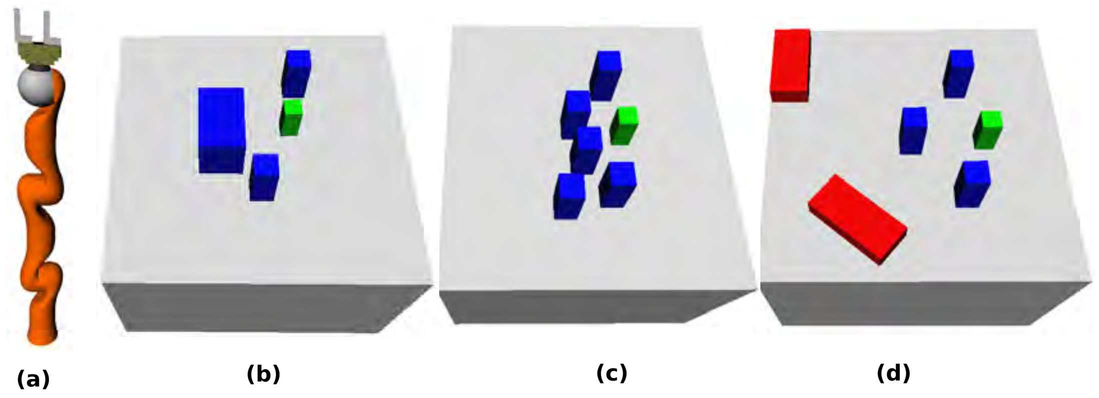


Figure 5.4: (a) The 7 degree-of-freedom KUKA robot; (b,c,d) Scenarios where the path to the target object (in green) is blocked by manipulatable objects (in blue) and fixed objects (in red).

end-effector should lie to grasp the object by closing the fingers (the pose uncertainty of the target object must lie within the gripper aperture, for any configuration within $\mathcal{R}_{\text{goal}}$).

2. *Tree steering* (lines 5-6): It includes the sampling of a pose of the end-effector $\mathbf{p}_{\text{rand}}$ using a region-biased sampler, and the determination of the node of the tree, called E_{near}, with a corresponding end-effector pose nearest to $\mathbf{p}_{\text{rand}}$ (a weighted distance measure between translations and rotations is used).

3. *Control selection* (lines 7-8): It first selects a control using a direct control sampling (Şucan et al., 2012a), i.e. it samples a set of random controls and time durations, applies them to E_{near} using the state propagator and the validity-checker, and selects the one that generates a best valid state E_{new} (in our case the one that brings the end-effector pose closest to p_{rand}). Then, the robustness of the chosen control is evaluated with a confidence index as detailed in Sec.5.4.2.

4. *Tree growing* (lines 9-16): Any control with a confidence index above a given threshold is added as edge and the generated state as a vertex (lines 10-11). Also the uncertainty region is updated (line 12) as detailed in Sec. 5.4.3., and if the end-effector has reached the goal region then the path is retrieved as a sequence of controls with the corresponding time duration (line 14).

5.6 Results and discussion

The proposal is validated using a KUKA lightweight robot with a two-fingers gripper for the three different scenarios presented in Fig. 5.4. The scenes consist of manipulatable objects (in blue), fixed objects (in red) and a target object (in green). The goal in the presented scenarios is to

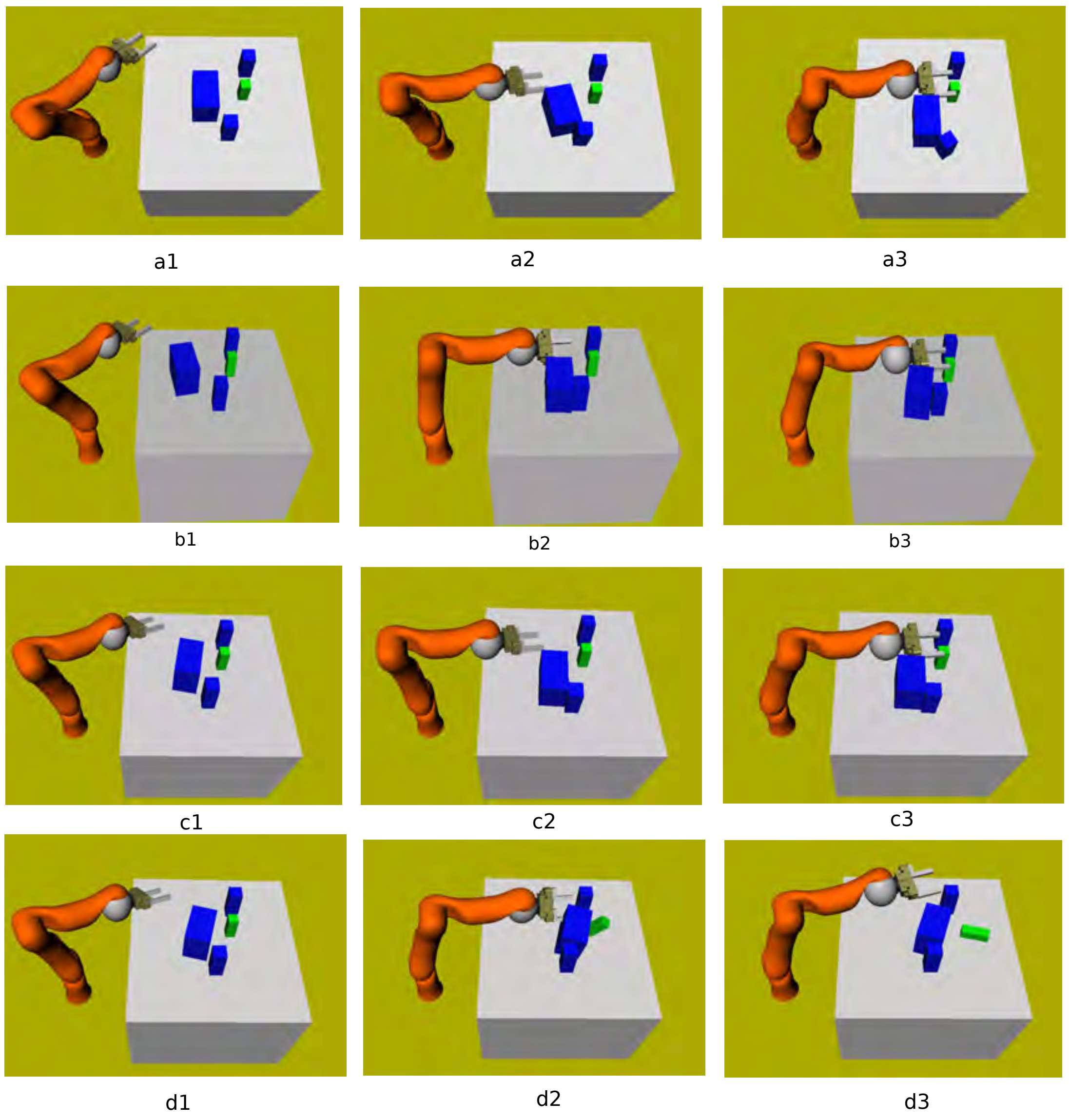


Figure 5.5: Snapshots of four task executions with different object poses: the first three rows correspond to successful executions, whereas the last one corresponds to a failed execution. Video:https://sir.upc.edu/projects/kautham/videos/RAL-2.mp4

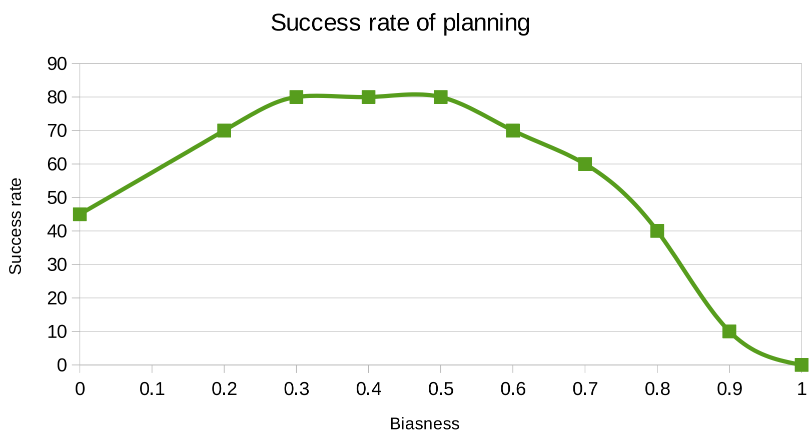


Figure 5.6: Success rate of the planner using different values of $\mathcal{B}$.

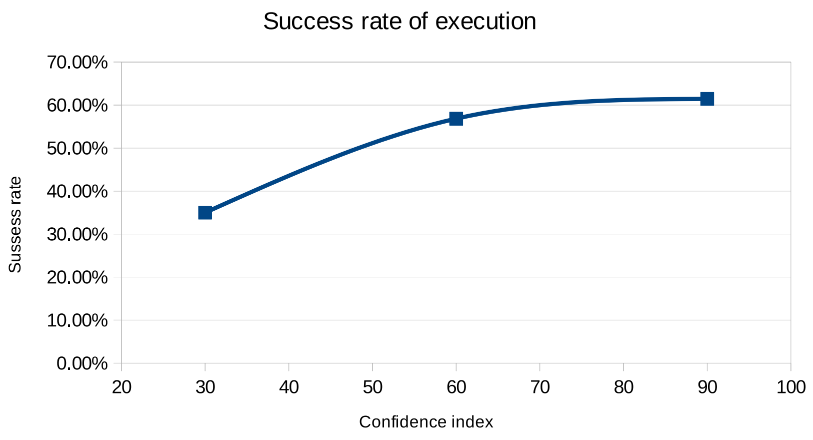


Figure 5.7: Success rate of the execution for different values of confidence index.

move the gripper to a pre-grasping configuration to grasp the target object. Since no collision-free trajectory exist from start to goal, to clear the path towards the goal, the robot has to manipulate the objects in a controlled way. The computed plan should be robust enough to be successfully executed in the presence of object pose uncertainty. Fig. 5.5 depicts the sequence of executions of an example scenario, the computed plan is executed by varying the pose of the object that must be pushed away in order to reach the goal.

The key contribution of this study is the use of high-level knowledge and reasoning process to efficiently plan grasping motions in the presence of object pose uncertainty. To cope with uncertainty, the robustness of the selected controls is evaluated using the confidence index, which is computed by repeatedly applying the control on a set of sampled states and, from the outcomes, estimating the probability of finding valid resultant state. This is a computationally expensive procedure, that is added to the computationally cost of the direct control sampling used in any kinodynamic RRT. The planning time can be reduced, however, by choosing an adequate region bias or by lowering the robustness threshold. In the former case the chosen value can also affect the planning success rate, which is analysed in Sec. 5.6.1. In the later case, the chosen value may affect the execution success rate, which is evaluated in Sec. 5.6.2.

5.6.1 Bias index

The bias index is a critical parameter since its value significantly affects the efficiency of the planner. The planning success rate of the proposed approach is evaluated by varying the bias index between 0 and 1. For each value of $\mathcal{B}$, 20 queries were launched (using the example scenarios), and setting the maximum planning time to 400 s. The simulation was run on an Intel Core i7-4500U 1.80GHz CPU with 16 GB memory. The success rate, computed based on the average number of successful runs, is shown in Fig. 5.6. The highest success rate for all the example scenarios is obtained by setting $0.25 \leq \mathcal{B} \leq 0.5$.

Running the ontological physics-based motion planning approach (Muhayyuddin et al., 2015) the success rate was only around 3%, due to the fact that this planner does not reason on the post-effect of the interactions and since obstacles are very close to the target object, they end by displacing it thus preventing the success of the task execution[1].

5.6.2 Robustness threshold

Motion plans have been computed for robustness thresholds of 30%, 60%, and 90%, with the value of $\mathcal{B}$ fixed to 0.3. Then, each computed plan has been executed 20 times in simulation by varying the initial state of the environment. For small deviations in the pose, the computed plans always executed successfully, whereas for the large deviations, the computed plan lead sometimes to unsuccessful results. The success rate of the execution of plans computed for

[1]Video: https://sir.upc.edu/projects/kautham/videos/RAL-3.mp4

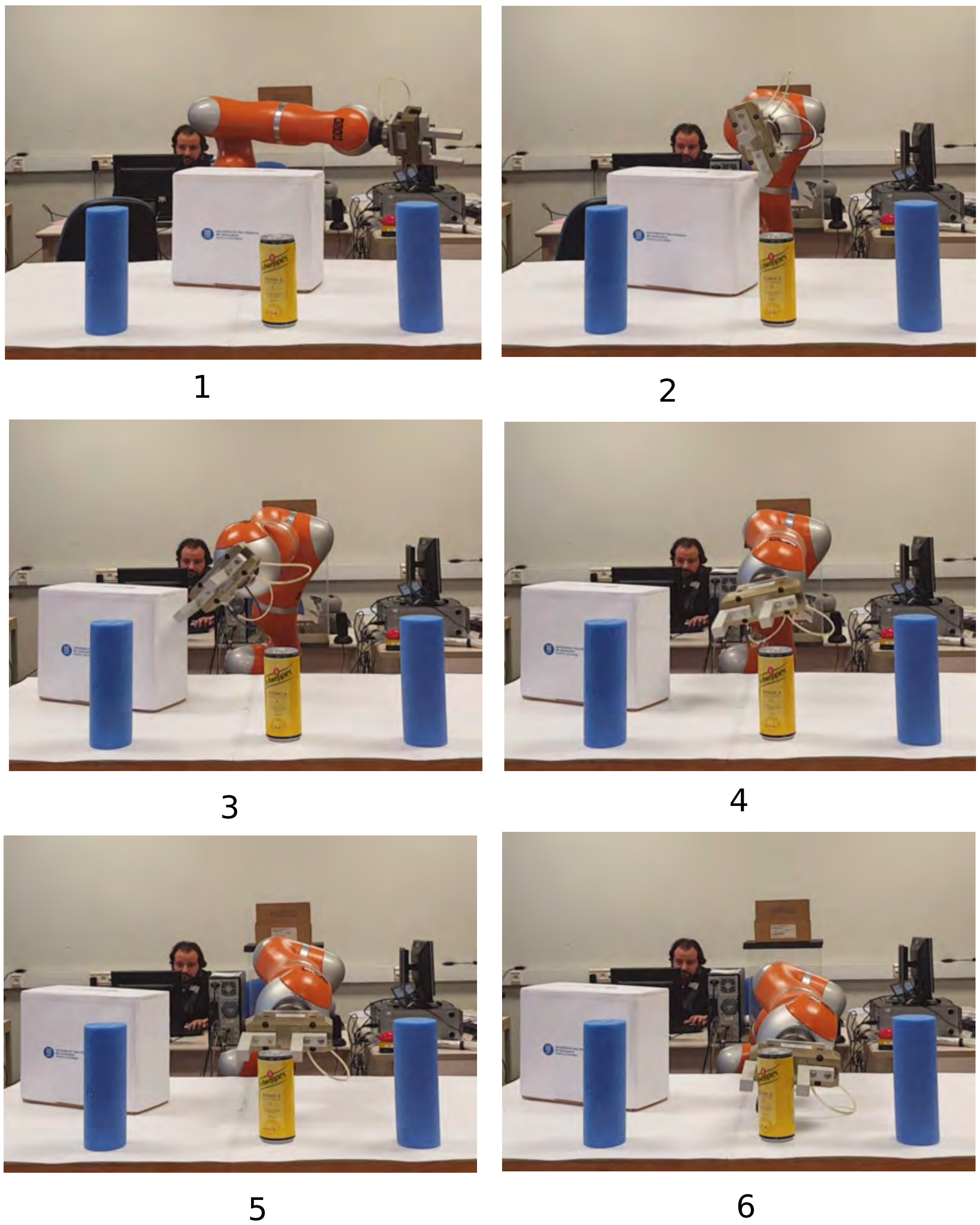


Figure 5.8: Execution on the real robot, by changing the pose of the object. The goal is to grasp the yellow cylinder. The path to the goal is obstructed with the white Video: https://sir.upc.edu/projects/ kautham/videos/RAL-1.mp4

different values of the robustness threshold is depicted in Fig. 5.7. The execution success rate improves with the robustness threshold, being the effect more relevant when changing from from 30 to 60, and more slightly when increasing up to 90. The computed plan for a scenario similar to the first one has also been executed with the real robot varying the poses of the objects, as shown in Fig. 5.8, the same behavior has been observed.

5.7 Conclusions

A knowledge-oriented physics-based motion planning approach is proposed for planning the grasping motions in cluttered environments. The semantic knowledge about the scene is stored using OWL ontologies. A knowledge-based reasoning process is proposed that computes the target region and uncertainty regions. The computed regions are used by the sampling-based kinodynamic motion planner that performs the region bias state sampling to plan efficiently, and by the validity checker that considers as non-valid those states that may prevent the success of the task. The proposed approach is validated for different scenarios. The results show a significant improvement in the execution success rate.

Chapter 6

Probabilistic-KPIECE: A clutter grasping approach under uncertainties

6.1 Introduction

The approach presented in the previous chapter works well for non-cluttered scenarios but the performance of the algorithm drops with increasing number of objects in the scene. This is due to the fact that most of the states are discarded by the state validity checker during the robustness evaluation phase, and the tree growing process is not influenced by the results of this robustness evaluation, i.e., there is no effective bias of the tree growing towards the states that are more robust. Moreover, no uncertainty is considered in contact dynamics. The uncertainty propagation in the object poses is considered as a Gaussian distribution, however the complex multi-body interactions may lead to high variations in the poses and can only be fit using a multi-model distribution. This chapter describes an approach to compute the robust and efficient plan for grasping in complex and uncertain environments. This approach is originally presented in (Muhayyuddin et al., 2018b).

The main contribution of this chapter is the proposal of a physics-based manipulation planning strategy (framed within sampling-based motion planning) for grasping in cluttered and uncertain environments. The main components of the approach are *(1) A probabilistic control sampler* that samples controls and computes the belief about the validity and the robustness of the sampled controls in the presence of object pose uncertainty; (2) *A tree exploration strategy* which integrates the computed belief with the planning data structures, biasing the exploration towards the states that have high belief; (3) *An uncertainty handling strategy* to cope with the propagation to future states of the objects' pose uncertainty which arise from robot-object or object-object interactions. To handle the environment dynamics, Open Dynamic Engine is used as state propagator. The proposal has been implemented as a variant of the KPIECE planner (Şucan and Kavraki, 2012). The enhanced algorithm, called probabilistic-KPIECE (*p-KPIECE*), has been validated in simulation and with real experiments. The results have been compared with an ontological physics-based motion planner and with task and motion planning approaches, resulting in a significant improvement in terms of planning time, success rate and quality of the solution path.

6.2 Problem setup

The world is modeled (including the robot and the objects) in the similar way as presented in the previous chapter. Let $\mathcal{X}$ be a differential manifold representing the state space of the robot. At any time t the state of the robot $x_t \in \mathcal{X}$ is specified as $x_t = (\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$, where $\boldsymbol{q}_t$ and $\dot{\boldsymbol{q}}_t$ describe the configuration of the robot and its time derivative, respectively. The objects in the workspace are classified as fixed, movable and target object, and are represented as $\mathcal{O} = \{\mathcal{O}^{\text{target}}, \mathcal{O}_1^{\text{movable}}, \dots, \mathcal{O}_{\text{M}}^{\text{movable}}, \mathcal{O}_1^{\text{fixed}}, \dots, \mathcal{O}_{\text{F}}^{\text{fixed}}\}$ with M and F being the number of movable and fixed objects, respectively. Their state space $\mathcal{S}$ consists of two components, $\mathcal{S} = \{\boldsymbol{\rho}, \boldsymbol{v}\}$ with $\boldsymbol{\rho}$ representing the pose, and $\boldsymbol{v}$ the linear and angular velocities. At any time t the state space $s_t \in \mathcal{S}$ of the target and movable objects is represented as $s_t = \{s_{\text{target}}, s_1, \dots, s_{\text{M}}\}$. The state of the world is modelled as $\mathcal{W} = \mathcal{X} \times \mathcal{S}$.

The discrete-time dynamic model of the robot is:

$$\mathcal{W}_{t+1} = \boldsymbol{f}(\mathcal{W}_t, \boldsymbol{u}_t), \tag{6.1}$$

where $\boldsymbol{f} : \mathcal{W} \times \mathcal{U} \to \mathcal{W}$ is the state transition function, with control space $\mathcal{U}$ representing all possible controls that can be applied to the system, and $\boldsymbol{u}_t \in \mathcal{U}$ a control input that is applied at time t.

Let $\mathcal{F}$ be a state validity checker defined as $\mathcal{F} : \mathcal{W} \times \boldsymbol{C} \to \{\mathsf{true}, \mathsf{false}\}$ where $\boldsymbol{C}$ represents the set of validity constraints $\boldsymbol{C} = \{kinodynamic, interaction\}$. The $kinodynamic$ constraints refer to the workspace bounds, the joints limits, bounds over the velocities, forces, and torques. The $interaction$ constraints refer to the set of constraints that describe how the robot can interact with the environment. These constraints are computed using knowledge-based reasoning as explained in previous chapters. A state is said to be valid if it satisfies all these validity constraints.

6.2.1 Modeling uncertainty

The current work handles: 1) uncertainty in the initial state of the environment, 2) uncertainty in the robot motions and 3) uncertainty due to dynamic interactions.

1) The uncertainty in the initial pose of the objects can be attributed to the process noise in the sensing of the environment as measured, for instance, with a RGBD camera. In this work

object pose uncertainty is modeled as:

$$\boldsymbol{U}_{\text{init}} \sim \mathcal{N}(\boldsymbol{\rho}^{\text{init}}, \boldsymbol{m}^{\text{init}}), \tag{6.2}$$

where $\mathcal{N}$ represents the multivariate Gaussian distribution with $\boldsymbol{\rho}^{\text{init}}$ being the mean representing the set of measured initial poses of the objects, $\boldsymbol{\rho}^{\text{init}} = \{\rho_1^{\text{init}} \dots \rho_{\text{M}}^{\text{init}}\}$, and $\boldsymbol{m}^{\text{init}}$ the variance, $\boldsymbol{m}^{\text{init}} = \{m_1^{\text{init}} \dots m_{\text{M}}^{\text{init}}\}$. Other uncertainty models could be alternatively used as needed.

2) The stochastic discrete-time dynamic model of the robot is described as:

$$\mathcal{W}_{t+1} = \boldsymbol{g}(\mathcal{W}_t, \boldsymbol{u}_t, \boldsymbol{\varepsilon}_t), \tag{6.3}$$

where $\boldsymbol{g} : \mathcal{W} \times \mathcal{U} \times \boldsymbol{\xi} \to \mathcal{W}$ is the state transition function, with the disturbance vector space $\boldsymbol{\xi}$ containing all possible disturbances that can be applied to the system. Disturbance in the robot controls is modelled as:

$$\boldsymbol{U}_{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{m}_{\varepsilon}). \tag{6.4}$$

The disturbance vector $\boldsymbol{\varepsilon}_t \in \boldsymbol{\xi}$ represents the control uncertainty at time t, and is used to introduce uncertainty in the control input at this instant of time.

3) The uncertainty in the future states is propagated when robot-object and object-object dynamic interactions occur. These dynamic interactions involve various dynamic parameters (such as friction, the pressure distribution under the object surface, the contact forces and the inertial effects), whose values greatly influence the behavior of the system. Physics engines provide a good approximation of the actual contact dynamics. ODE, in particular, does it with three contact parameters $\mathcal{D} = \{\mu, c, e\}$, with μ being the vector of friction coefficients between pairs of objects, c a constraint force mixing parameter to soften constraints and e an error reduction parameter to relax the constraints satisfaction. The uncertainty in the interactions will be modelled by setting a multivariate Gaussian distribution around the predefined approximated values:

$$\boldsymbol{U}_{\mathcal{D}} \sim \mathcal{N}(\mathcal{D}^{\text{aprox}}, \boldsymbol{m}_{\mathcal{D}}), \tag{6.5}$$

with mean values $\mathcal{D}^{\text{aprox}} = \{\boldsymbol{\mu}^{\text{aprox}}, c^{\text{aprox}}, e^{\text{aprox}}\}$ and variance $\boldsymbol{m}_{\mathcal{D}} = \{m_\mu, m_c, m_e\}$. The values of these parameters are tuned in simulation and are qualitatively evaluated; the associated variances are small.

6.2.2 Problem statement

Consider a motion planning problem that asks a robot to grasp an object in a cluttered and uncertain environment, where possibly no collision-free trajectory exists that moves the robot from an initial configuration to a pre-grasping pose. All the objects are assumed to have stable support surfaces and be laying on a flat working region (e.g., a table). The objective is to compute the sequence of robust controls and their durations in such a way that, if applied to the system in the presence of object pose uncertainty and imprecise knowledge of contact dynamics, it moves the robot from the start to the goal state (a pre-grasping pose), by pushing away those

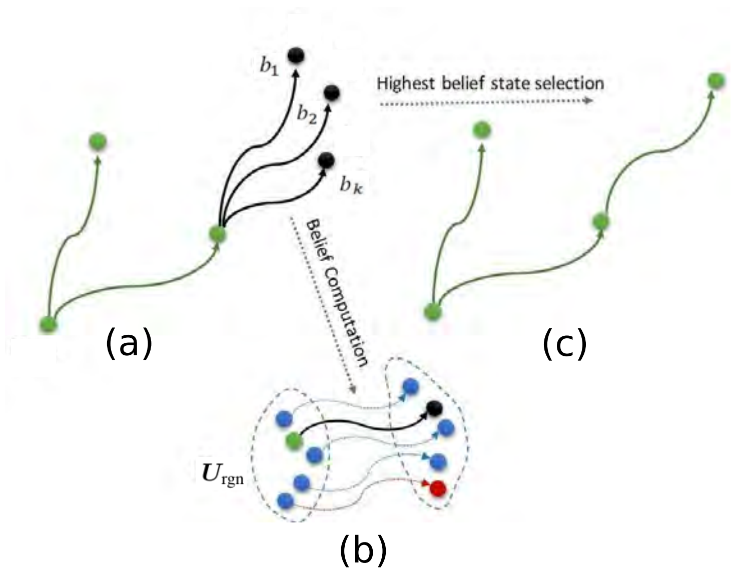


Figure 6.1: Control selection process. The motions in green belong to the tree, motions in black are the candidate motions and the motions in blue are the particle motions that are used to compute the belief of the candidate motions.

moveable objects obstructing the path, while avoiding collisions with fixed obstacles. Object-object interactions between moveable objects are allowed. Note that we are not dealing with the rearrangement problem, i.e. the final pose of the objects is not relevant.

Moreover, in order to ease the finding of a solution, the following constraints are set: 1) no interaction is allowed with the target object; 2) as a result of interactions the object's velocity must be less than a given threshold, and no object should change its support surface or fall from the working region.

6.2.3 Solution overview

The proposed approach considers the above stated motion planning problem as an open-loop problem, i.e., it tries to develop a robust strategy that absorbs the potential deviations in the objects poses and in the results of the interactions occurring during the planning process. The work presents a physics-based motion planner with an underlying sampling-based kinodynamic planner that incorporates a robust tree growing strategy. This strategy works in two phases. The first one is a control sampling phase, once the state from which the tree will grow is selected, the algorithm applies randomly sampled controls called candidate controls for some randomly sampled time durations to generate valid motions called candidate motions (as depicted in Fig. 6.1-a). No uncertainty is considered in this phase. The second phase considers the uncertainty in the system, as stated in Sec. 6.2.1, to compute the belief about the robustness and validity of the candidate motions. This is done for each candidate motion by considering the uncertainty and repeatedly applying the candidate control to obtain a set of associated motions, called particle motions (as shown in Fig. 6.1-b, where $\boldsymbol{U}_{\text{rgn}}$ represents the uncertainty in the object's pose as introduced in Sec. 6.3.1), and evaluating their validity. This belief is used to select the best candidate motion (Fig. 6.1-c), thus enhancing the tree exploration process. Additionally, a strategy to handle the uncertainty due to dynamic interaction is included in the approach.

6.3 Probabilistic-KPIECE

Kinodynamic Motion Planning by Interior and Exterior Cell Exploration (KPIECE) (Şucan and Kavraki, 2009; Şucan and Kavraki, 2012) is used to plan the trajectory. It is a sampling-based kinodynamic motion planner, particularly designed for systems with complex dynamics, briefly introduced in chapter 1.

KPIECE grows a tree of motions. A motion is defined as $\nu = (\boldsymbol{x}, \boldsymbol{u}, d)$, where $\boldsymbol{x}$ is the start state of the motion, $\boldsymbol{u}$ is the control vector that is applied at state $\boldsymbol{x}$ for a time duration d. A key feature of KPIECE is how it estimates coverage during exploration. In order to estimate the coverage, the state space is projected into a low-dimensional space that is partitioned into cells. As a result of the projection, motions are classified into cells (each cell can contain several motions). These cells are divided into interior and exterior, depending on whether neighboring cells are occupied or not. The tree growing procedure is based on first selecting a cell, then selecting a motion of that cell and a state pertaining to it, from where the tree will grow. The cell selection process for the tree growth is based on a parameter called *importance* (cells with high importance are preferred for expansion), defined for a cell z as:

$$\text{Imp}(z) = \frac{\log(\mathcal{I}) \cdot score}{C_{\text{cell}} \cdot (1 + |Neigh(z)|) \cdot Cov(z)}, \tag{6.6}$$

where $\mathcal{I}$ represents the planning iteration when the cell was added to the grid, C_{cell} represents the number of times it has been selected, $Cov(z)$ represents the number of states in the cell, $|Neigh(z)|$ is the number of instantiated neighboring cells, and $score$ represents the exploration progress, a value that is computed by evaluating the increase in the total coverage and the time spent to achieve this increment in coverage (when expanding from the cell). Once the cell is chosen, a motion from that cell is selected using a half normal distribution (over the indices of the motions, ordered starting with the newest). Finally, a state is randomly picked from the selected motion and a new motion is sampled from that selected state. The process continues until the tree of motions reaches the goal state.

The current proposal extends this algorithm with:

1. Strategies to handle the propagation of objects' pose uncertainty into future states resulting from robot-object or object-object interactions (Sec. 6.3.1).

Algorithm 11: Probabilistic KPIECE

```
inputs : Initial State $\mathcal{W}_{init}$, initial pose uncertainty $\boldsymbol{U}_{\text{init}}$, goal region $\mathcal{Q}_{goal}$, maximum time
         $T_{max}$, candidate motions $k$, particle motion $n_p$, displacement threshold $d$, validity
         constraints $C$
output: A sequence of motions that move the robot to the goal region
1  $\nu_0 = \mathcal{W}_{\text{init}}$
2  $G$.empty()
3  $G$.AddMotion($\nu_0$)
4  $\boldsymbol{U}_{\text{rgn}} = \boldsymbol{U}_{\text{init}}$
5  while $T_{max}$ do
6      cell = SelectCell($G$)
7      $\nu$= SelectMotion(cell)
8      $\nu_{new}$=MotionSampler($\nu, k, n_p, d, C, \boldsymbol{U}_{\text{rgn}}$)
9      if $\nu_{new} \in Q_{goal}$ then
10         return path to $\nu_0$
11     if $\nu_{new}! = NULL$ then
12         $G$.AddMotion($\nu_{new}$)
13         $\boldsymbol{U}_{\text{rgn}} \leftarrow$UpdatePoseUncertainty($\boldsymbol{U}_{\text{rgn}}$)
14     figures UpdateParameters()
15     UpdateCellImportance(cell)
16 return NULL
```

2. A motion sampling strategy that samples motions to expand the tree and computes the belief about the robustness of the sampled motions (Sec. 6.3.2).

3. A tree exploration strategy to modify the cell and motion selection process according to these beliefs (Sec. 6.3.3).

The algorithm, inspired by the original KPIECE algorithm, is presented in Algorithm 11. As input it takes the initial state of the world $\mathcal{W}_{init}$, the uncertainty in the initial pose of the objects $\boldsymbol{U}_{\text{init}}$, a goal region $\mathcal{Q}_{goal}$, a time threshold T_{max}, the number of candidate motions k to use, the number of particle motions n_p to evaluate the robustness, the displacement threshold d for evaluating interactions, and the validity constraints C that are to be used by the state validity checker. It starts with ν_0, a motion of zero duration representing the initial state.

Lines 1 to 3, 12 and 14 are the default steps of KPIECE. The contribution of the current proposal is implemented in function UpdatePoseUncertainty (line-13) that updates the object pose uncertainty (Sec. 6.3.1), in function MotionSampler (line-8) that samples k motions and select the one that has highest belief (Sec. 6.3.2), and in functions SelectMotion (line-7) and UpdateCellImportance (line-15) that, respectively, select a motion from the selected cell to grow the tree and update the cell importance (Sec. 6.3.3).

6.3.1 Handling uncertainty propagation

To handle uncertainty, the most robust motion should be chosen, i.e., those with higher chance to reach a valid state. For this purpose, the robustness of the candidate motions is evaluated by generating a set of particle motions. The belief of the candidate motions is computed as a function of the validity of the motion particles. This section explains how the particle motions are generated and used to update the uncertainty pose of the objects when interactions take place.

Particle motion generation

Particle motions are generated to evaluate a nominal candidate motion. Each particle motion is generated by sampling the initial state, the dynamic interaction parameters and the control to be applied:

- The initial state is sampled from the probability density function (PDF) associated to the initial state of the nominal candidate motion (initially represented by Eq. (6.2)).
- The dynamic parameters are sampled using the PDF function described in Eq. (6.5).
- The control is determined by randomly adding noise to the nominal candidate control using Eq. (6.4).

Then, a particle motion is obtained by applying the selected control to the chosen initial state and considering the sampled dynamic interaction parameters. Each particle motion will be evaluated using the state validity checker $\mathcal{F}$. introduced in Sec. 6.2.

Updating object pose uncertainty

Once interactions take place, the object pose uncertainty of the displaced objects is recomputed (objects may collide several times with the robot and/or with other objects). The poses of the objects as a result of interaction are highly variant depending on the contact regions and interaction force directions. Therefore, the resultant poses of the objects after applying several particle motions, can be modelled using the Gaussian Mixture Model (GMM), which is a convenient way to model the data that comes from different sources (Pfaff et al., 2008):

$$\boldsymbol{U}_{\text{rgn}} = \sum_{j=1}^{n} \alpha_j \cdot \mathcal{N}(c_j, m_j) \text{ with } \alpha_j \geq 0 \,, \; \sum_{j=1}^{n} \alpha_j = 1. \tag{6.7}$$

Eq. (6.7) is computed using Expectation-Maximization (EM) algorithm (Dempster et al., 1977).

6.3.2 Probabilistic motion sampler

The function MotionSampler (Line-8, Algorithm 11) is used to select a robust motion to expand the tree. The process, detailed in Algorithm 12 is a probabilistic motion sampler that:

- Samples k candidates (Line 1).
- Computes the belief about their validity (Line 2-21).
- Returns the one that has the highest belief (or with a given probability a random one) to be added to the tree data-structure (Line 22-25).

Algorithm 12 uses the following functions:

- SampleCandidateMotions: Samples a set $\mathcal{M}$ of k candidate motions by first randomly choosing a single state from the input motion, and then by randomly sampling k controls and durations.
- Propagate^c: Uses Eq. (6.1) to compute the state resulting from applying motion ν_i (i.e., sampled control $\boldsymbol{u}$ at the selected state $\boldsymbol{x}$ for the time duration d).
- Propagate^p: Uses Eq. (6.3) to compute the state resulting from applying motion ν_i.
- StateValidityChecker: Evaluates the resultant state of the world, $\mathcal{W}_c$, using the state validity checker $\mathcal{F}$. If $\mathcal{W}_c$ satisfies all the validity constraints, $\mathcal{F}$ will return true, and false otherwise. The validity constraints are explained in Sec. 6.2.
- GetObjectPoses: Retrieves the object poses from $\mathcal{W}_c$.
- SampleObjectPose: Samples the object poses using Eq.(6.2) or Eq.(6.7), depending on whether the object is still at its initial pose or has been moved due to interactions.
- SampleDynamicParameters: Samples the dynamic parameters using the probability distribution function described in Eq.(6.5).
- InteractionEvaluator: Returns true if $|disp| \leq d$ or false otherwise, where $disp$ is the displacement vector containing the displacement(s) covered by the object(s) in the result of interaction and d a given threshold (experiments showed an increase in success rate if object pose uncertainty was reduced by avoiding very large displacements in highly cluttered scenes).
- ComputeBelief: Compute the belief of a motion as $b^i = \mathrm{P}^i_{state}\mathrm{P}^i_{int.}$.
- AddMotionBelief: Add the belief into the corresponding motion data structure in $\mathcal{M}$.
- SelectMotion: Selects the motion from $\mathcal{M}$ that has the highest belief, although with a given small probability, it is replaced by SelectRandom that selects a motion randomly regardless of its belief value.

Algorithm 12: MotionSampler($\nu, k, n_p, d, C, \boldsymbol{U}_{\text{rgn}}$)

```
inputs : Motion to grow the tree $\nu$, pose uncertainty $\boldsymbol{U}_{\text{rgn}}$, candidate motions $k$, particle motion $n_p$,
         displacement threshold $d$, validity constraints $C$
output: A new motion $\nu_{new}$
1  $\mathcal{M}$ ←SampleCandidateMotions($\nu, k$)
2  foreach $\nu_i \in \mathcal{M}$ do
3      $\mathcal{W}_c \leftarrow$ Propagate$^c(\nu_i)$
4      if StateValidityChecker($\mathcal{W}_c, C$) then
5          $\boldsymbol{v}_{\text{state}} = \boldsymbol{v}_{\text{int.}} = 0$
6          for $j = 1$ to $n_p$ do
7              $\rho$ ←GetObjectPoses($\mathcal{W}_c$)
8              SampleObjectPose($\rho, \boldsymbol{U}_{\text{rgn}}$)
9              SampleDynamicParameters($\boldsymbol{U}_{\mathcal{D}}$)
10             $\mathcal{W}_p \leftarrow$ Propagate$^p(\nu_i)$
11             $valid_S$ ←StateValidityChecker($\mathcal{W}_p, C$)
12             $valid_I$ ←InteractionEvaluator($\mathcal{W}_p, disp, d$)
13             if $valid_S$ then
14                 $\boldsymbol{v}_{\text{state}} = \boldsymbol{v}_{\text{state}} + 1$
15             if $valid_I$ then
16                 $\boldsymbol{v}_{\text{int.}} = \boldsymbol{v}_{\text{int.}} + 1$
17         $\boldsymbol{P}^i_{state} \leftarrow \boldsymbol{v}_{\text{state}}/n_p$
18         $\boldsymbol{P}^i_{int.} \leftarrow \boldsymbol{v}_{\text{int.}}/n_p$
19         $\boldsymbol{b}_{\nu_i}$ ←ComputeBelief($\boldsymbol{P}^i_{state}, \boldsymbol{P}^i_{int.}$)
20         AddMotionBelief($\nu_i, \boldsymbol{b}_{\nu_i}$)
21         ExistValidMotion=$true$
22 if ExistValidMotion then
23     if GenerateRandom(0,1) > bias then
24         return $\boldsymbol{\nu}_i$ ←SelectMotion(max($\boldsymbol{b}_{\nu_i}$))
25     return $\boldsymbol{\nu}_i$ ←SelectRandom($\mathcal{M}$)
26 return NULL
```

6.3.3 Tree exploration strategy

The main parameters that control the tree exploration in KPIECE are the cell selection process and the motion selection from the selected cell. In order to enhance the exploration strategy to make the tree to grow through robust regions, our approach modifies these processes as follows.

- The definition of a motion is modified by incorporating the motion belief, i.e., $\boldsymbol{\nu} = (\mathcal{W}, \boldsymbol{u}, d, \boldsymbol{b}_\nu)$, where $\mathcal{W}$ is the state of the world and $\boldsymbol{b}_\nu$ represents the belief about the robustness of the motion.
- A belief value is associated with the cells, according to the beliefs of the motions they contain. It is computed as the mean value of the beliefs of the motions normalized for all

the cells, i.e.:

$$\boldsymbol{b}_{cell} = \frac{\overline{\boldsymbol{b}}_\nu}{\sum_{\forall \text{cell}j} \overline{\boldsymbol{b}}_\nu^j} \quad \text{with} \quad \overline{\boldsymbol{b}}_\nu = \frac{1}{n} \sum_{\forall \nu_i \in \text{cell}} \boldsymbol{b}_{\nu_i} \tag{6.8}$$

- The Imp parameter is modified by integrating $\boldsymbol{b}_{\text{cell}}$ in order to favour those cells with higher belief (the strength of this bias being controlled by a heuristic parameter f, set equal to the number of cells):

$$\text{Imp}(z) = \frac{(1 + f\boldsymbol{b}_{cell}) \cdot \log(\mathcal{I}) \cdot score}{C \cdot (1 + |Neigh(z)|) \cdot Cov(z)} \tag{6.9}$$

Then, once a cell is chosen for the exploration, the motion from that cell is selected based on $\boldsymbol{b}_\nu$. If several motions have the same belief, a random motion will be selected from them. In order to explore the less certain regions, with a fixed small probability, the selection is done as in the standard KPIECE.

6.4 Evaluation

The proposed approach described in the previous section results in a planner that enhances the power of KPIECE by considering uncertainty, object interactions and by favoring the tree exploration towards safer areas, allowing to plan motions in cluttered and uncertain environments. The main parameters of the algorithm are evaluated in this section, where comparisons with other approached are also done.

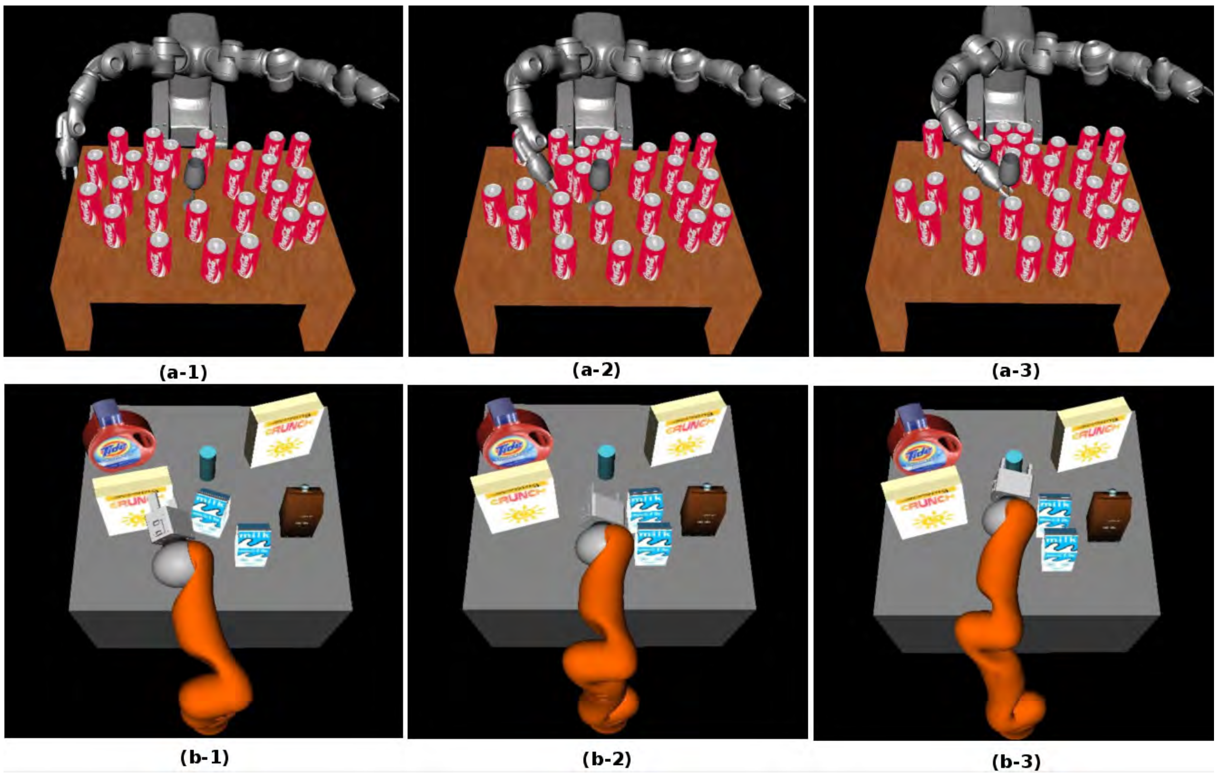


Figure 6.2: Sequence of the snapshots of the execution with: a) YuMi; b) Kuka-LWR. Videos: https://goo.gl/RzLVi1 and https://goo.gl/socxhb

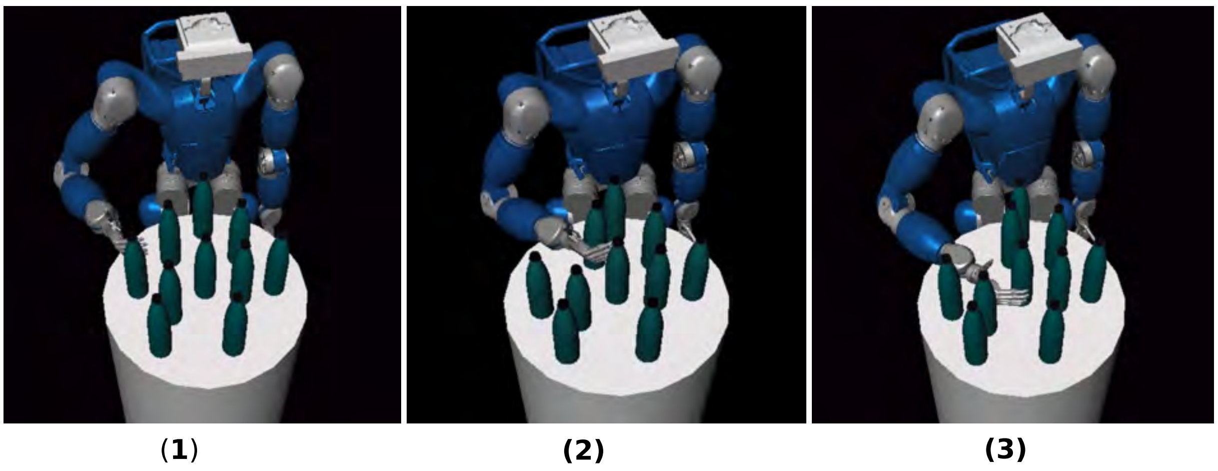


Figure 6.3: Sequence of the snapshots of the execution with TORO.

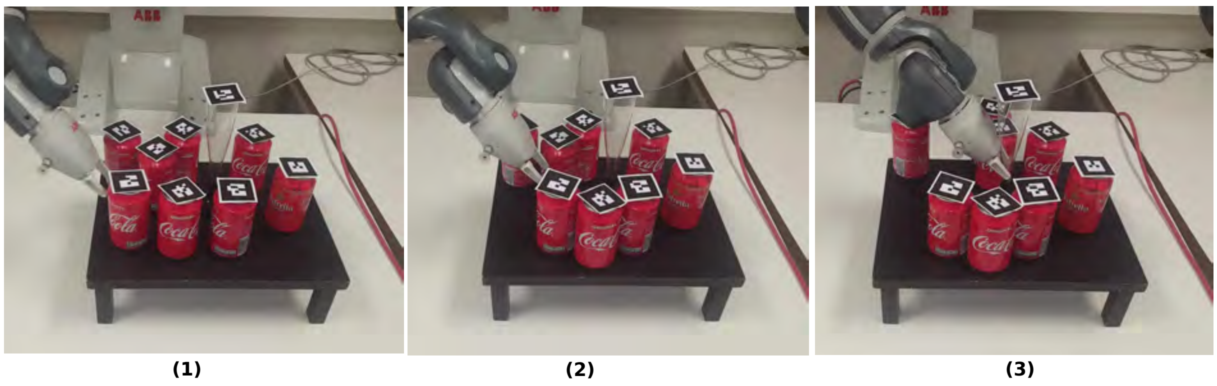


Figure 6.4: Sequence of snapshots of the execution on the real YuMi robot.
Video: https://goo.gl/bHspjZ

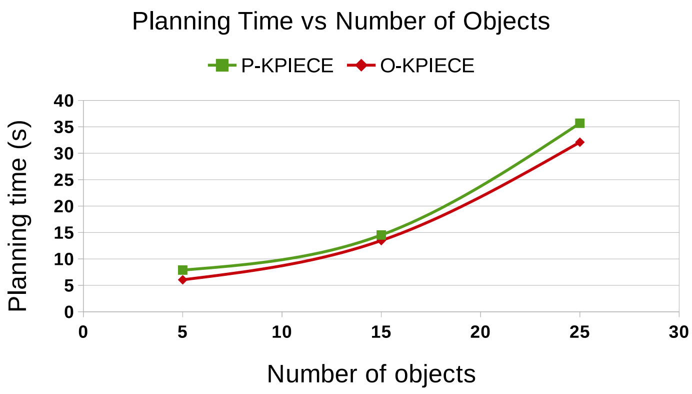


Figure 6.6: Graph of average planning time of 30 runs by varying the clutterness.

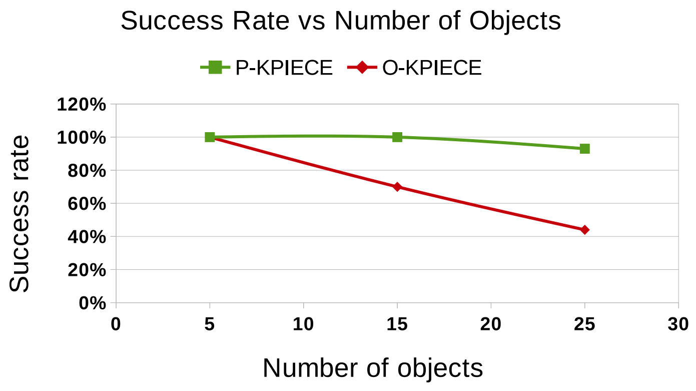


Figure 6.7: Graph of average planning success rate of 30 runs by varying the clutterness.

6.4.1 Simulation setup

The proposed approach has been tested with three different robots: the 14 degree-of-freedoms (DOF) ABB YuMi, the 7 DOF KUKA-LWR, and the 39 DOF TORO Humanoid Robot. The example scenarios are presented in Fig. 6.2. The scene shown in Fig 6.2-a consists of movable objects (red cans) and the target object (wine glass). The goal is to move the robot to a pre-grasp configuration to grasp the wine glass, by pushing away those cans obstructing the path. The scene depicted in Fig 6.2-b consists of movable objects of arbitrary shapes (boxes and bottle) and the target object (cyan cylinder). The scene shown in Fig 6.3 consists of movable objects (bottles), the goal is to move the TORO right arm to the pre-grasp configuration to grasp the middle bottle. Moreover, the approach is also validated with the real YuMi robot, and a sequence of snapshots of the execution is depicted in Fig. 6.4. All experiments were run on an Intel Core i7-4500U 1.80GHz CPU with 16 GB memory.

6.4.2 Benchmarking

The current approach is benchmarked against:

- an ontological physics-based motion planner, introduced in chapter 2, that enhances KPIECE for physics-based motion planning (it will be referred as o-KPIECE);
- task and motion planning approach for grasping in clutter proposed by Srivastava et al. (2014) and Hadfield-Menell et al. (2015). We run 30 simulation for each scenario.

Since p-KPIECE modifies the core planning structure of the KPIECE, in order to investigate the planning behavior after these modifications we compare it with o-KPIECE. The reason to

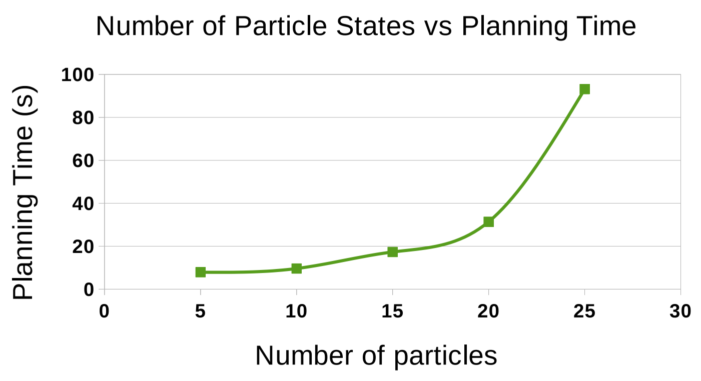


Figure 6.8: Graph of average planning time of 60 runs by varying the number of particle motions.

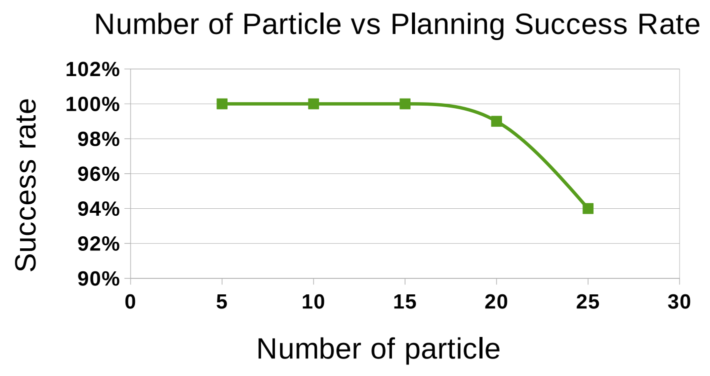


Figure 6.9: Graph of average planning success rate of 60 runs by varying the number of particle motions.

compare p-KPIECE with a task and motion planning approach is the planning domain, i.e., the problem that we are addressing (grasping in clutter) is usually addressed by approaches that combine both planning levels.

Algorithmic parameter evaluation

The current approach always generates more efficient solutions in terms of memory. Extensive tests showed that the number of generated states in p-KPIECE is always less than o-KPIECE. This implies that the number of generated cells is also less. Since the cell parameters are updated after each propagation step, p-KPIECE will update them faster, as compared to o-KPIECE. Fig. **??** shows the histograms of the average number of generated states and cells during planning,

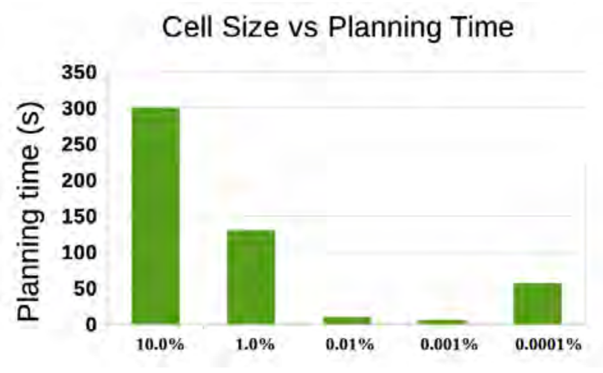


Figure 6.10: Histogram of average planning time of 60 runs by varying the cell size.

using o-KPIECE and p-KPIECE. The comparison between planning time of both planners using different levels of clutterness is shown in Fig. 6.6. Since p-KPIECE considers uncertainty, it performs an additional step that is robustness evaluation of each generated state, which makes it computationally expensive as compared to the o-KPIECE. As a result, for clutter scenes, the planning success rate of p-KPIECE is higher, as shown in Fig. 6.7.

The robustness evaluation depends on the number of particle motions that are used to compute the belief, the larger the number of particle motions the more robust the plan. However, the computational time increases with the number of particle motions, shown in Fig. 6.8, and this may drop the planning success rate for not being able to find a solution on time, as shown in Fig. 6.9. Another parameter that may affect the planning process is the cell size used by KPIECE, that must be properly set. Big cell sizes do not ease the identification of the relevant regions of the workspace and fail to guide the tree growth, resulting in long unrealistic paths. On the other hand, smaller cell sizes better guide the search, thus enhancing the planning efficiency. Too small cell sizes, however, may increase the computational time required due to the huge number of cells generated. This effect on the planning time is depicted in Fig. 6.10, where cell size is represented as a percentage of the side length with respect to the side of the cube that models the workspace. In this work the state space is projected to the workspace, using the end-effector position.

Comparison with task planning

The current proposal is also compared with the task and motion planning approaches (TMP) for grasping in the clutter without and with uncertainty (Hadfield-Menell et al., 2015; Srivastava et al., 2014), respectively. We have generated a qualitatively similar setup as that in those references, as shown in Fig. 6.11. Since the TMP approach presented in (Srivastava et al., 2014) does not consider uncertainty in the environment, in order to be fair with the comparison, we have assumed that the uncertainty in the environment is almost negligible, and then one particle motion is enough to evaluate the robustness, i.e., $k = 1$. Whereas, to compare with (Hadfield-

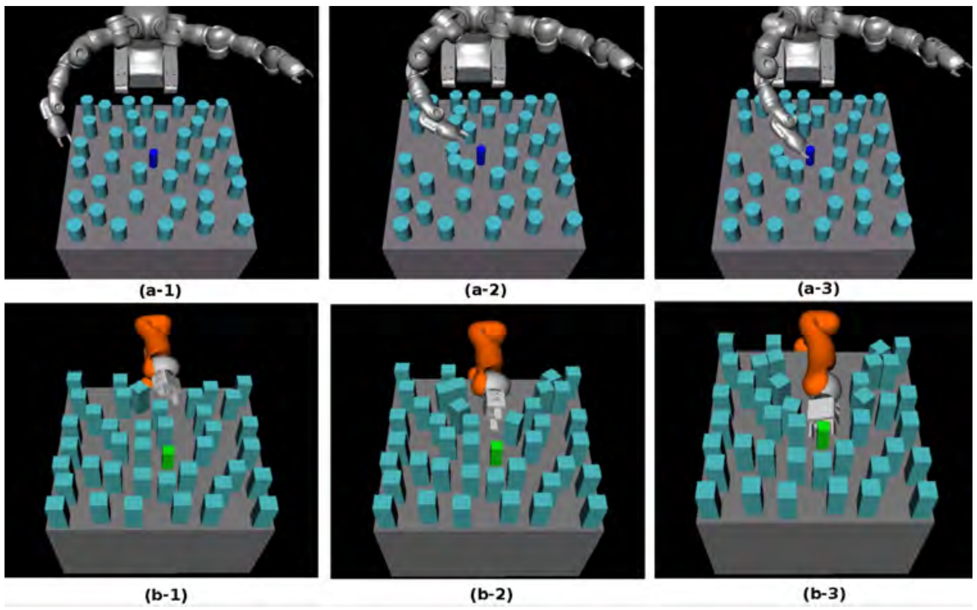


Figure 6.11: Sequence of the snapshot of the executions with the YuMi and the Kuka-LWR, for the comparison with the task and motion planning approach.
Videos: https://goo.gl/ZorqCF and https://goo.gl/F4fZVL

Menell et al., 2015), the value of k is set to 15. The higher the value of k the higher the confidence in the planner exploring safer regions, but the lower the computationally efficiency. The chosen value will depend on the problem; $k = 15$ worked well in the example scenarios presented in this paper. Table 6.1 summarizes the results of the comparison. The data regarding the TMP approaches has been obtained from (Srivastava et al., 2014) and (Hadfield-Menell et al., 2015), being generated with an Intel Core i7-4770K machine with 16 GB RAM (i.e., a faster processor than the one used in the current study). The task and motion planning approach requires explicit reasoning to perform each action, i.e. repeatedly selects an object to pick, computes the collision free trajectory to grasp it, finds and appropriate placement location and moves the object to that location. In contrast, the current proposal does not require explicit reasoning about the complex dynamic multi-body interactions for moving in the clutter. At each step, the objects must satisfy the global set of constraints, that are easy to evaluate. This process makes it easy to compute the robust plan efficiently, even in the presence of uncertainty.

6.4.3 Qualitative analysis

The quality of the solution path computed by p-KPIECE is improved as compared to the o-KPIECE. Since o-KPIECE does not evaluate the post-effect of dynamic interactions and the results of object-object interactions, that leads to the unreasonable results. For cluttered scenes, in most of the cases, it displaces the target object or drops the objects from the table surface,

Obj.	Success rate %				Av. planning time (s)			
	TP	TPU	pKP	pKPU	TP	TPU	pKP	pKPU
5	-	95	100	100	-	89	2.89	9.88
10	-	95	100	100	-	162	5.96	14.52
15	100	83	100	90	32	135	9.15	29.43
20	94	70	100	86	57	229	16.78	47.61
25	90	68	96	73	69	166	26.09	72.28
30	84	48	90	66	77	122	41.21	103.07
35	67	-	73	53	41	-	49.62	134.94
40	63	-	60	40	68	-	71.64	182.19

Table 6.1: Comparison of p-KPIECE with task and motion planning approaches. Obj. represents the number of objects used, TP and TPU represent the approaches presented in Srivastava et al. (2014) and Hadfield-Menell et al. (2015), respectively. pKP and pKPU represent the p-KPIECE with and without uncertainty, respectively.

whereas p-KPIECE carefully evaluates the post-effect of dynamic interactions, interaction velocity and displacement covered by the object in the result of interactions, greatly increasing the planning success rate. Moreover, the incorporation of robustness evaluation phase results in the more stable and robust motion plans, i.e. the integration of the results of robustness evaluation within the planning data structure allows the search to be guided towards the regions that are safer, thus increasing the success in the final execution of the planned paths.

The solution computed by p-KPIECE is also qualitatively different from the TMP solution. The TMP solution is generated by integrating several move, pick and place actions. Each motion planning query is set to move the robot to a particular place (for picking or placing an object); these local queries do not consider the final goal. In contrast, p-KPIECE computes the solution in a more natural way, it launches a single query and moves away the objects obstructing the path.

Currently, no smoothing operation over the trajectory is applied, although it can easily be added by using deterministic control sampling strategies or by applying post processing over the computed path as presented by Van Den Berg et al. (2011).

6.5 Conclusions

This chapter addressed a motion planning problem for grasping in clutter and uncertain environments by using a randomized physics-based motion planner. The developed framework takes into account the uncertainty in the objects' pose and in the contact dynamics. The KPIECE planner has been enhanced by: a) introducing a motion sampler, for the extension of the tree, that samples motions and evaluates the belief about their robustness in the presence of uncertainty; b) biasing the tree exploration strategy based on the computed beliefs. The proposed planner enables single actions to move multiple objects, thereby avoiding the combinatorial explosion in the task planning part of TMP. The work is validated in simulation and in real environment against other approaches, such as ontological physic-based motion planner and task and motion planning approach. The results show significant advantages in terms of planning time, success rate and the quality of the computed solution path.

Part III

Implementation, conclusions and future work

Chapter 7

Implementation framework for knowledge-oriented physics-based planning

7.1 Introduction

Planning and simulation play an important role in robotics research. These are essential tools for the development of strategies and algorithms in various areas of robotics such as motion planning, grasping and manipulation. Moreover, these tools allow to demonstrate the proposed strategies under different environmental conditions and constraints. The increasing complexity in the robotic systems, such as those including collaborative robots (new generation of industrial robots) or humanoid robots, set new challenges for robotic software. These challenges involve the rich semantic description of the environment for the understanding of the scene, the incorporation of dynamics in planning, the capability of reasoning about the performed actions, and computational efficiency. It is difficult to find a software that addresses all these challenging issues.

The chapter presents a simulation framework for knowledge-oriented physics-based motion planning that is developed by extending *The Kautham Project* (Rosell et al., 2014) by integrating the ontological knowledge, reasoning and physics in the planning process. This simulation framework is used to implement and test the planning strategies presented in the previous chapters. This simulation tool is originally presented in (Muhayyuddin et al., 2017c).

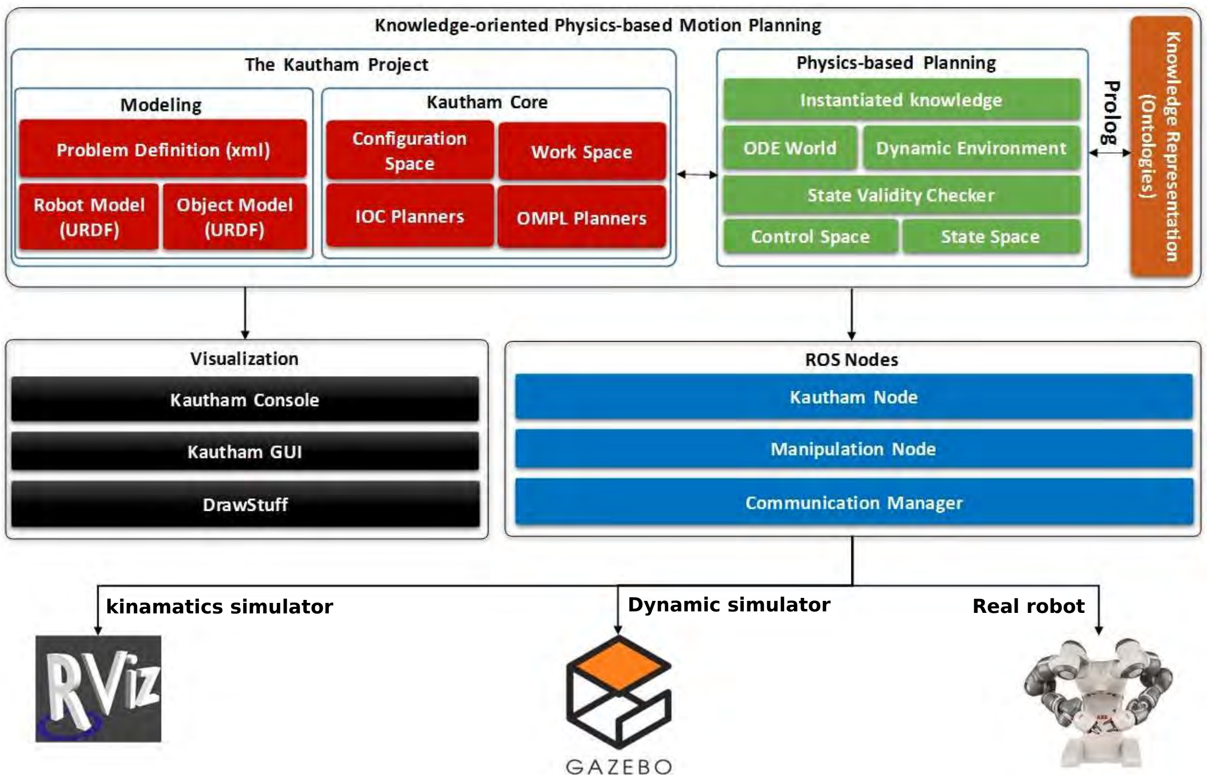


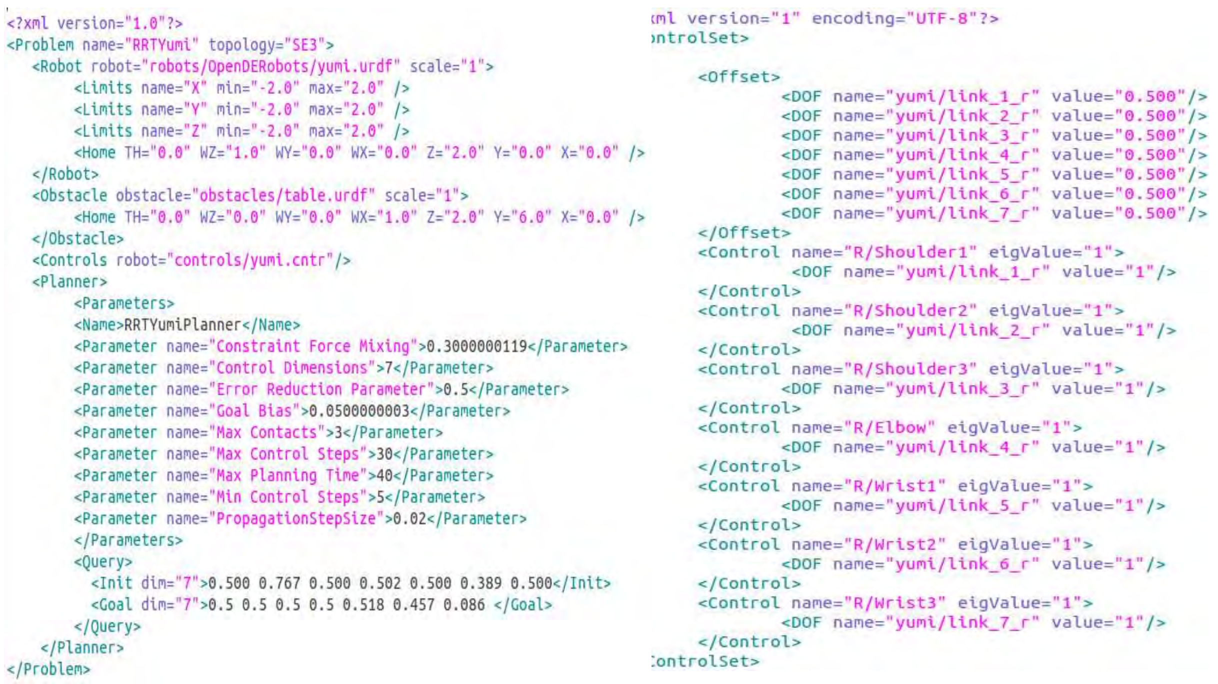
Figure 7.1: Simulation framework for knowledge-oriented physics-based motion planning.

7.2 Implementation framework

The proposed framework (Fig. 7.1) is developed by extending *The Kautham Project*. This section will briefly explain *The Kautham Project* and the implementation details of the proposed extensions for incorporating knowledge, reasoning and physics in planning.

7.2.1 Dependencies

The robotic simulation software depends on several concepts such as robot modeling, 3D rendering and collision checking. It is difficult to develop a standalone application from scratch and therefore, in order to incorporate such features, usually already existing libraries are used. The major dependencies of Knowledge-oriented physics-based planning framework are those of *The Kautham Project*. It is developed using C++ and uses many features of C++11 such as *std* features. It uses the CMake (www.cmake.org) build system and it is available under GIT (git-scm.com) version control system, and can be downloaded from sir.upc.ed/kautham. The GUI is designed in Qt (qt-project.org), 3D rendering is performed using Coin3D (www.coin3d.org). Robots and obstacles are defined using the Unified Robotic Descrip-



```
<?xml version="1.0"?>
<Problem name="RRTYumi" topology="SE3">
   <Robot robot="robots/OpenDERobots/yumi.urdf" scale="1">
        <Limits name="X" min="-2.0" max="2.0" />
        <Limits name="Y" min="-2.0" max="2.0" />
        <Limits name="Z" min="-2.0" max="2.0" />
        <Home TH="0.0" WZ="1.0" WY="0.0" WX="0.0" Z="2.0" Y="0.0" X="0.0" />
   </Robot>
   <Obstacle obstacle="obstacles/table.urdf" scale="1">
        <Home TH="0.0" WZ="0.0" WY="0.0" WX="1.0" Z="2.0" Y="6.0" X="0.0" />
   </Obstacle>
   <Controls robot="controls/yumi.cntr"/>
   <Planner>
        <Parameters>
        <Name>RRTYumiPlanner</Name>
        <Parameter name="Constraint Force Mixing">0.3000000119</Parameter>
        <Parameter name="Control Dimensions">7</Parameter>
        <Parameter name="Error Reduction Parameter">0.5</Parameter>
        <Parameter name="Goal Bias">0.0500000003</Parameter>
        <Parameter name="Max Contacts">3</Parameter>
        <Parameter name="Max Control Steps">30</Parameter>
        <Parameter name="Max Planning Time">40</Parameter>
        <Parameter name="Min Control Steps">5</Parameter>
        <Parameter name="PropagationStepSize">0.02</Parameter>
        </Parameters>
        <Query>
          <Init dim="7">0.500 0.767 0.500 0.502 0.500 0.389 0.500</Init>
          <Goal dim="7">0.5 0.5 0.5 0.5 0.518 0.457 0.086 </Goal>
        </Query>
    </Planner>
</Problem>
```

(a)

```
ml version="1" encoding="UTF-8"?>
ntrolSet>

    <Offset>
            <DOF name="yumi/link_1_r" value="0.500"/>
            <DOF name="yumi/link_2_r" value="0.500"/>
            <DOF name="yumi/link_3_r" value="0.500"/>
            <DOF name="yumi/link_4_r" value="0.500"/>
            <DOF name="yumi/link_5_r" value="0.500"/>
            <DOF name="yumi/link_6_r" value="0.500"/>
            <DOF name="yumi/link_7_r" value="0.500"/>
    </Offset>
    <Control name="R/Shoulder1" eigValue="1">
             <DOF name="yumi/link_1_r" value="1"/>
    </Control>
    <Control name="R/Shoulder2" eigValue="1">
             <DOF name="yumi/link_2_r" value="1"/>
    </Control>
    <Control name="R/Shoulder3" eigValue="1">
            <DOF name="yumi/link_3_r" value="1"/>
    </Control>
    <Control name="R/Elbow" eigValue="1">
            <DOF name="yumi/link_4_r" value="1"/>
    </Control>
    <Control name="R/Wrist1" eigValue="1">
            <DOF name="yumi/link_5_r" value="1"/>
    </Control>
    <Control name="R/Wrist2" eigValue="1">
            <DOF name="yumi/link_6_r" value="1"/>
    </Control>
    <Control name="R/Wrist3" eigValue="1">
            <DOF name="yumi/link_7_r" value="1"/>
    </Control>
ontrolSet>
```

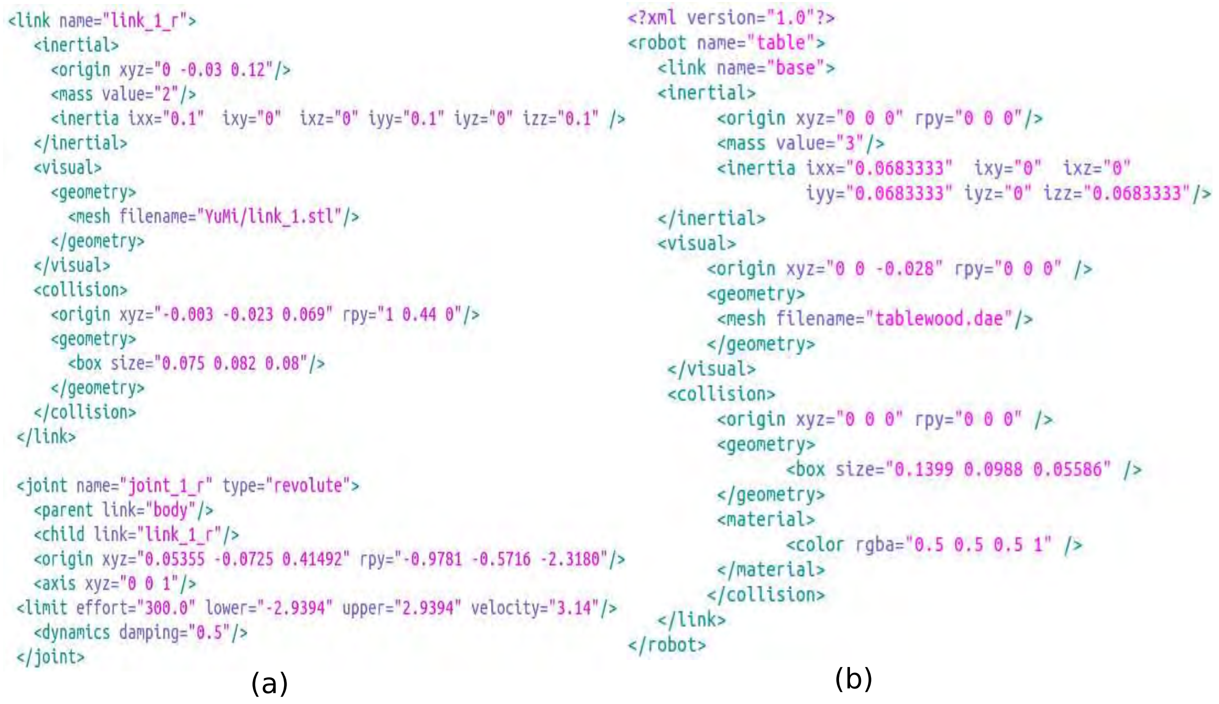
(b)

Figure 7.2: (a) an example of a problem file. (b) an example of a control file.

tion Format (URDF, http://wiki.ros.org/urdf), and the Eigen library (http://eigen.tuxfamily.org/) is used for linear algebra. The Open Motion Planning library (OMPL) (Şucan et al., 2012b) is used as planning core. It provides various sampling-based motion planners such as RRT (LaValle and Kuffner, 2001), PRM (Kavraki et al., 1996) and KPIECE (Şucan and Kavraki, 2012). Moreover, it also provides the capability of planning in state space where ODE is used as state propagator. The current proposal enhances the use of ODE in planning process to incorporate physics using knowledge-based reasoning. The knowledge is represented using Web Ontology Language (OWL) (Antoniou and van Harmelen, 2003) and the Prolog language (http://www.swi-prolog.org/) is used for the reasoning over knowledge.

7.2.2 The Kautham Project

The Kautham Project is a C++ based open-source software for motion planning. It is used for teaching and research purposes at the Institute of Industrial and Control Engineering (IOC-UPC). For research, it is used to develop and demonstrate the motion planning algorithms, particularly for mobile and dexterous manipulators (arms equipped with anthromorphic hands and a mobile base).



(a)

```
<link name="link_1_r">
   <inertial>
     <origin xyz="0 -0.03 0.12"/>
     <mass value="2"/>
     <inertia ixx="0.1"  ixy="0"  ixz="0" iyy="0.1" iyz="0" izz="0.1" />
   </inertial>
   <visual>
     <geometry>
       <mesh filename="YuMi/link_1.stl"/>
     </geometry>
   </visual>
   <collision>
     <origin xyz="-0.003 -0.023 0.069" rpy="1 0.44 0"/>
     <geometry>
       <box size="0.075 0.082 0.08"/>
     </geometry>
   </collision>
 </link>

 <joint name="joint_1_r" type="revolute">
   <parent link="body"/>
   <child link="link_1_r"/>
   <origin xyz="0.05355 -0.0725 0.41492" rpy="-0.9781 -0.5716 -2.3180"/>
   <axis xyz="0 0 1"/>
 <limit effort="300.0" lower="-2.9394" upper="2.9394" velocity="3.14"/>
   <dynamics damping="0.5"/>
 </joint>
```

(b)

```
<?xml version="1.0"?>
<robot name="table">
   <link name="base">
   <inertial>
         <origin xyz="0 0 0" rpy="0 0 0"/>
         <mass value="3"/>
         <inertia ixx="0.0683333"  ixy="0"  ixz="0"
                  iyy="0.0683333" iyz="0" izz="0.0683333"/>
   </inertial>
   <visual>
        <origin xyz="0 0 -0.028" rpy="0 0 0" />
        <geometry>
         <mesh filename="tablewood.dae"/>
        </geometry>
    </visual>
    <collision>
         <origin xyz="0 0 0" rpy="0 0 0" />
         <geometry>
                <box size="0.1399 0.0988 0.05586" />
         </geometry>
         <material>
                <color rgba="0.5 0.5 0.5 1" />
         </material>
        </collision>
   </link>
</robot>
```

Figure 7.3: (a) a part of the URDF file of the robot. (b) the URDF model of the table

7.2.3 Modeling

A motion planning problem is described using an XML file (Fig. 7.2). It consists of four components: robot model, object model, controls and planner. The main parameters that are specified for robots/objects are the path to the corresponding model, translation limits (in case of mobile base) and initial pose with respect to the world frame. Controls are used to define the way how the degree-of-freedoms (DOF) will be actuated. In the simplest case, one control per DOF is considered. Controls can also be specified for obstacles (detailed explanation of controls can be found in (Rosell et al., 2014)). The final part of the problem XML file specifies the name of the planner used (such as RRT), the planning parameters (such as planning time and goal bias) and the query that contains the start and the goal configurations.

A robot is defined as a kinematic tree with optional mobile base, its configuration space is $R = SE(3) \times \mathbb{R}^n$, where n represents the number of joints of the robot. In case of fixed base, the $SE(3)$ part is represented as null. The kinematic structure of the robot is defined using URDF (Fig. 7.3). It contains the visual robot model, the collision model, transformations between the links, joints (along with limits) and dynamic parameters such as damping and masses. The visualization model is defined with triangular meshes that can be represented in *.wrl*, *.stl* and *.dae* formats. The collision model can be represented either by a triangular mesh or by primitive shapes (cylinder, box and sphere). Obstacles are also defined as robot data structures, in case of fixed obstacles, none of its dof are actuated.

7.2.4 Kautham-core

It consists of the workspace, the configuration space and a set of planners. Once a problem is loaded, it fills the data structures of the workspace (that includes the robot/obstacle models, their kinematic limits), and of the configuration space. Moreover, it contains the methods for collision checking using PQP (Gottschalk et al., 1999) or FCL (Pan et al., 2012), and forward kinematics to move the robot to the particular configuration. To sample the configuration space various state samplers (such as random, Gaussian and Halton) are included.

Two families of planning algorithms are implemented in *The Kautham Project*, IOC planners and OMPL planners. The former contains potential field-based planners using navigation functions (Barraquand et al., 1992) and harmonic functions (Connolly and Grupen, 1993). The latter contains the sampling-based geometric and kinodynamic planners (such as RRT, PRM and EST) offered by OMPL. The detailed explanation regarding the implementation of planners can be found here https://sir.upc.edu/projects/kautham/.

7.3 Implementation of physics-based planning

To enable the physics-based planning, ODE is used to handle the rigid body dynamics during propagation. It is an open-source C++ based dynamic engine widely used in the robotics community. Moreover, OMPL provides a flexible way of using ODE as state propagator. From the input files, the robot(s) and object(s) and their properties (such as masses) are read and the ODE bodies are then created using triangular meshes, although in case of simple shapes (such as box, sphere or cylinder), ODE primitive shapes are created. The kinematic tree that represents the robot in the dynamic world is created by adding the joints between the robot bodies. A motor (linear or angular, depending on the joint type) is added to each joint to control the joint velocities and torques. Once the ODE world is created, a dynamic environment class (with the name of the robot, such as *YumiDynamicEnviroment*) is derived from the *OpenDEEnvironment* class provided by OMPL. The derived class reimplementes the functions by defining the control dimensions, control bounds, the way of applying controls, the contact parameters (such as friction, slip, bounce velocity) and the way of evaluating the collisions.

The control dimensions are set equal to the number of actuated degree-of-freedoms and the control bounds define the control (velocity or torque) limits for each joint. A method is provided to define the way of applying the controls. The controls can be joint velocities with maximum allowed torque limits (that the motor can exert to achieve the desired velocity). Contact parameters are defined between each pair of bodies in contact, describing the interactions. For instance, when an interaction takes place between two bodies, these parameters define how many contact points must be considered, what is the value of the friction coefficient, what is the bounce velocity, what is the constraint force mixing (CFM) and the error reduction parameter (ERP). CFM and ERP are ODE parameters that model the damping and spring behavior of the contact. These contact parameters must be defined carefully because inappropriate values may results in unstable behaviors.

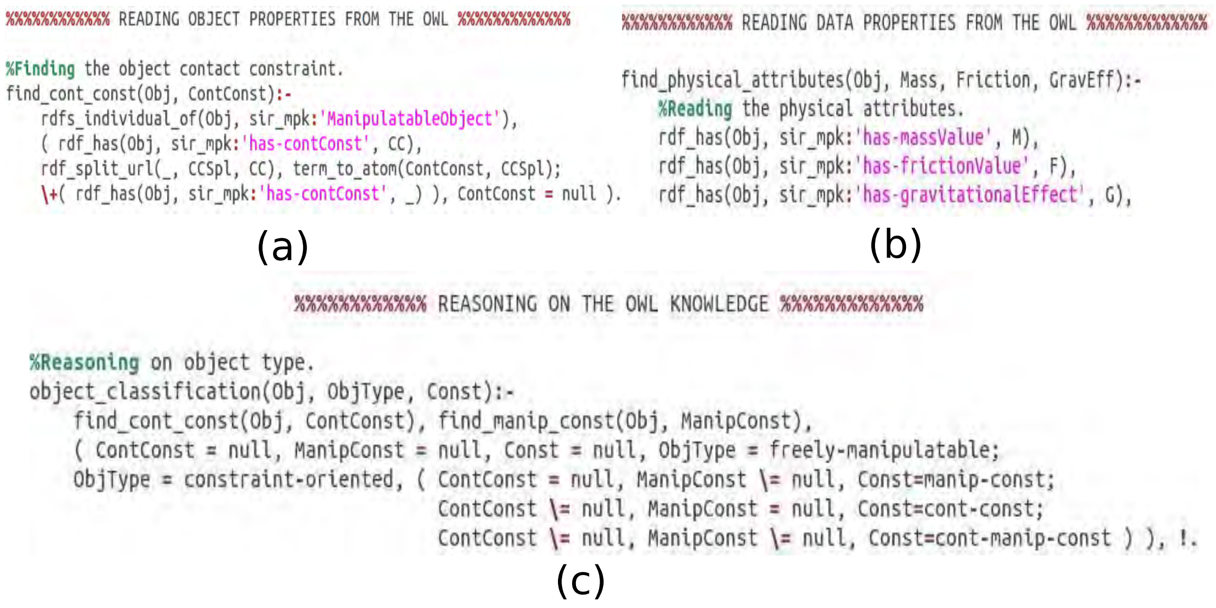
Since physics-based planning allows the dynamic interactions in planning, the way of evaluating collision needs to be modified. Collisions with fixed objects will be forbidden, but collision with movable objects will be allowed, although collision with some movable object may be allowed only from certain parts. The differentiation of the objects according to their collision properties is a challenging issue. It is handled by incorporating the contact constraints in the knowledge as explained in Sec. 7.4. The state validity checker will evaluate the satisfaction of the constraints that are imposed by the knowledge.

The state space of each body (robot link or obstacle) in a dynamic environment is 12 dimensional (3 for position, 3 for orientation, 3 for linear velocity and 3 for angular velocity). It is represented as an OMPL *OpenDEStatepace* that is a compound state space with 3 real vector spaces and one SO(3) space for orientation. The state space implements the distance function to measure the distance between two states. The proposed framework provides two implementations of distance function that measure the distance in the workspace and in the configuration space. To measure the former, the position of the TCP is projected in the workspace and the Cartesian distance is measured there. Whereas the latter measures the distance between two configurations, according to the topology of the configuration space.

Moreover, the state space generator also provides the pointer to the projection used. Currently one method of projection is implemented, it projects the position of end effector in the workspace, besides the random projections offered by OMPL.

The implementation of the control space includes different sampling methods. Since physics-based planning is computationally intensive, the complexity can be reduced by implementing robust control sampling strategies. The current implementation provides a random control sampler, an heuristic-based control sampler and a power-efficient control sampler. The heuristic-based control sampler samples n controls and select the one that results in a state closer to the goal state. The power-efficient control sampler adapts the control sampling strategy according to the region of the state space, i.e. if the robot is in contact with an object the sampling strategy computes the minimum force that is required to push the target object and sample the controls accordingly, as detailed in chapter 3.

All control-based planners offered by OMPL can be used for knowledge-oriented physics-based planning. For every planner we need to set a pointer to the defined dynamic environment, state space and control space. The planners such as RRT, KPIECE, EST, SyCLoP are already available for planning. Other planners such as SST can be incorporated easily.



```
%%%%%%%%%%%% READING OBJECT PROPERTIES FROM THE OWL %%%%%%%%%%%%%

%Finding the object contact constraint.
find_cont_const(Obj, ContConst):-
    rdfs_individual_of(Obj, sir_mpk:'ManipulatableObject'),
    ( rdf_has(Obj, sir_mpk:'has-contConst', CC),
    rdf_split_url(_, CCSpl, CC), term_to_atom(ContConst, CCSpl);
    \+( rdf_has(Obj, sir_mpk:'has-contConst', _) ), ContConst = null ).
```

```
%%%%%%%%%%%% READING DATA PROPERTIES FROM THE OWL %%%%%%%%%%%%%

find_physical_attributes(Obj, Mass, Friction, GravEff):-
    %Reading the physical attributes.
    rdf_has(Obj, sir_mpk:'has-massValue', M),
    rdf_has(Obj, sir_mpk:'has-frictionValue', F),
    rdf_has(Obj, sir_mpk:'has-gravitationalEffect', G),
```

```
%%%%%%%%%%%% REASONING ON THE OWL KNOWLEDGE %%%%%%%%%%%%%

%Reasoning on object type.
object_classification(Obj, ObjType, Const):-
    find_cont_const(Obj, ContConst), find_manip_const(Obj, ManipConst),
    ( ContConst = null, ManipConst = null, Const = null, ObjType = freely-manipulatable;
    ObjType = constraint-oriented, ( ContConst = null, ManipConst \= null, Const=manip-const;
                                     ContConst \= null, ManipConst = null, Const=cont-const;
                                     ContConst \= null, ManipConst \= null, Const=cont-manip-const ) ), !.
```

Figure 7.4: Examples of Prolog predicates, (a) represents the predicate for the object properties, (b) describes the predicate for the object data properties. (c) describes the predicate for reasoning over the object types.

7.4 Implementation of knowledge representation and reasoning

Knowledge is represented with ontologies using OWL, which is a formal way of representing knowledge in term of classes. These classes contain information about the robot (robot kinematic and dynamic properties) and about the environment (the objects and their relationship with each other). The relation among classes is defined based on axioms. The axioms are facts that are used for conceptual understanding. We used the Protégé editor (http://protege.stanford.edu/) to formulate ontologies (they can be found at https://sir.upc.edu/projects/ontologies/). Domain specific ontologies can be easily defined to handle other planning domain problems, such as task planning.

To enhance the planning process with knowledge, the knowledge is fetched from the ontologies and stored in *instantiated knowledge*. The *instantiated knowledge* is a low-level representation of knowledge that contains the type of the objects, such as manipulatable or fixed, and their contact constraints. These constraints are modeled by specifying regions around the objects, such that the robot can interact with the objects only from these regions. Moreover, other types of constraints can be introduced easily such as manipulation constraints (constraints over orientation that robot has to maintain during manipulation). The detailed explanation of instantiated knowledge can be found in chapter 3.

The reasoning module is defined in Prolog, which is a language of facts and rules that de-

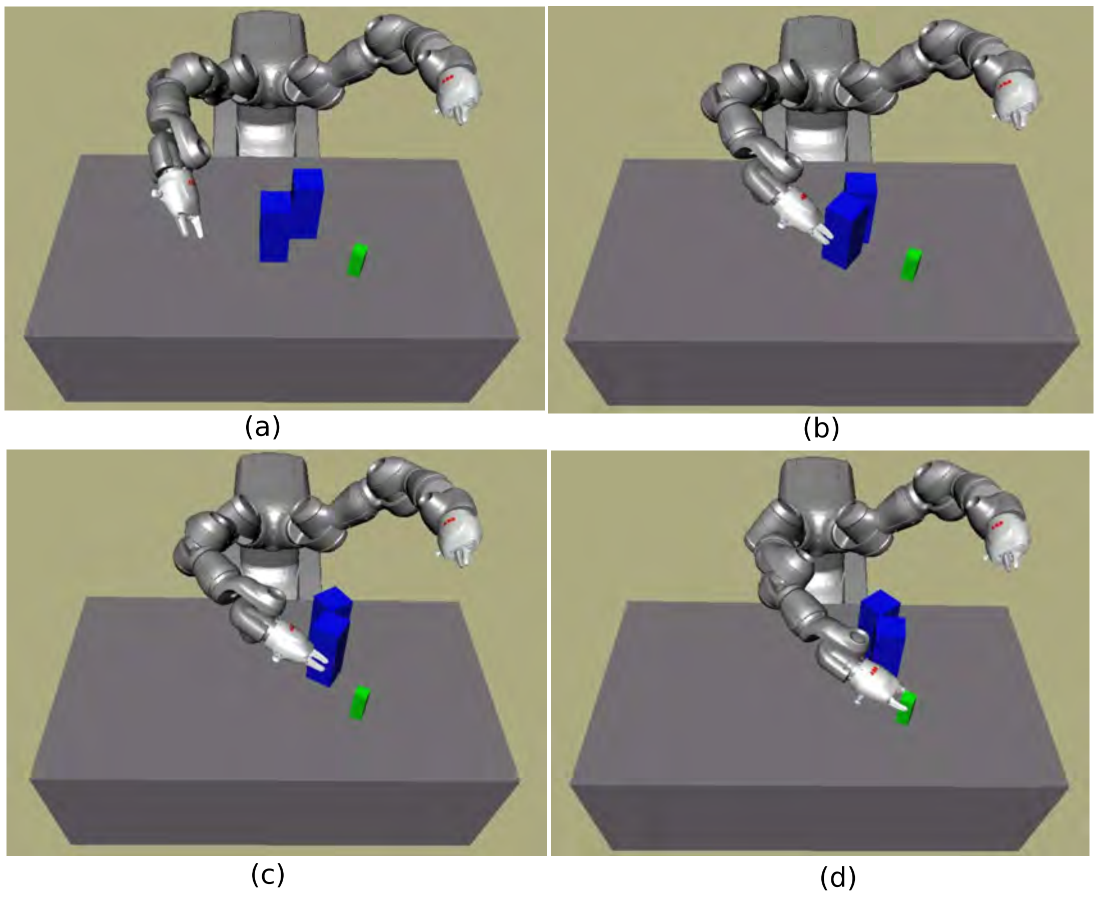


Figure 7.5: Visualization of the robot motion with the *Kautham-GUI*

fines predicates for the knowledge-based reasoning. The predicates are defined in a file (with extension .pl). While creating the ODE world, the Prolog environment is initialized and reads the knowledge from the OWL using predefined predicates. Some examples of the Prolog predicates are shown in Fig. 7.4. The predicates to access object and data properties are shown in Fig. 7.4-a and Fig. 7.4-b respectively. The predicate *find_cont_const(obj, ContConst)* takes the name of ODE body (from ODE world) as input and returns the associated contact constraints. *find_physical_attribute(obj,Mass.Friction,GravEff)* reads the physical properties of the bodies in the ODE world. The predicate described in Fig. 7.4-c is an example of the reasoning over OWL for the object classification. The predicate *object_classification(obj,objectType,Const)* reasons about the types of the object and classify them accordingly into the manipulatable and fixed objects, along with their constraints (contact and manipulation). The Prolog predicates fetch the knowledge from the ontologies and fills the data structures of the *instantiated knowledge* that is used by the motion planner. According to the problem domain, more predicates can be easily defined.

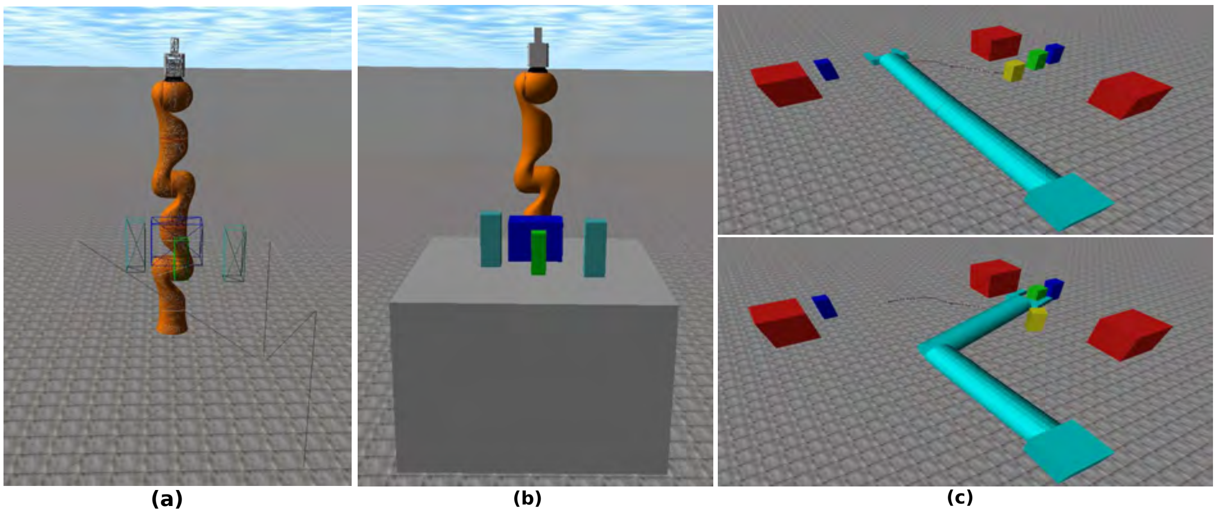


Figure 7.6: Example scenes of the visualization of the triangular mesh view and the actual scene view for different robots using *DrawStuff*

7.5 Visualization

The proposed framework uses the *Kautham-GUI* tool for the visualization of the scene. It provides the visualization of the robot model, the collision model and the visualization of the configuration space (or a projection of it when its dimension is greater than three). The scene can also be visualized with *DrawStuff* (OpenGL based ODE viewer). It provides the visualization of the collision model, the actual robot/object model and the mesh views. Fig. 7.5 and Fig. 7.6 depict the visualization using *Kautham-GUI* and *DrawStuff* respectively. The numerical results of a query (such as a list of configurations of the solution path) can also be viewed using *Kautham-Console*, that is a console-based interface of *The Kautham Project*.

7.5.1 Implemented ROS Nodes

The Kautham Project provides a ROS based interface through a node called *Kautham-rosnode*. It provides the services such as *OpenProblem*, *SetQuery*, *Sovle* and *GetPath*. The current proposal implements two further nodes, *manipulation node* and *communication manager* for handling the manipulation queries and communicating with the real robot or a third party simulation environment, such as Gazebo.

7.5.2 Manipulation node

The manipulation node extends the functionality of the Kautham node. It is capable of handling the manipulation queries such as push or pull queries. A manipulation query is defined by specifying a target object (that will be pushed or pulled by the robot), the type of manipulation action (such as push, pull or move) and other planning parameters (such as planning time). In response it returns the controls and durations that are to be applied to move the robot from the start to the goal state by satisfying the constraints.

7.5.3 Communication manager

The communication manager is another ROS node that manages the communication between a client application and the real robot. It provides the services to set the query for the manipulation node and sends the computed path to the real robot via ROS/ROS Industrial and receives the feedback from the real robot, such as joint states. Moreover, it also provides the communication between the planning framework and Gazebo or Rviz (http://wiki.ros.org/rviz) to visualize the computed path.

7.6 Conclusions

This chapter has described a simulation tool for knowledge-oriented physics-based motion planning by extending *The Kautham Project*. It provides an easy and reliable way to incorporate the rigid- body dynamics and the knowledge-based reasoning (about the action effects) in planning process. It also provides the manipulation node to easily handle the manipulation queries (such as push/pull). The proposed simulation tool also provides an easy way to communicate with real robot through the ROS based communication manager that manages the communication between the proposed tool and the real robot.

Chapter 8

Conclusions and future work

8.1 Conclusions

This thesis developed a series of physics-based motion planning algorithms for grasping and manipulation in cluttered and uncertain environments. Physics-based motion planning in computationally intensive and it is quite challenging to determine which is the right way to interact with the objects. We investigated various strategies to determine the solutions that overcome these challenging issues and find the more robust and realistic motion plans. In the first part of the thesis we tried to explore the possibility of the integration of the knowledge (related to the robot and the objects in the workspace) with the planning process. The role of this knowledge was to facilitate the planning process by describing the right way of interacting with the moveable objects in the environments.

Then we extended our investigation by developing the argument that humans adapt the interaction forces depending on the properties of the target object e.g., we never exert the same force when moving our arm freely, when pushing a table or when pushing a plate on the table. To develop such interaction capabilities we integrate the knowledge-based reasoning process that evaluate the physical properties of the objects to determine the appropriate control force bounds that the robot should obey while manipulating the objects. Then the knowledge-based reasoning is further extended to handle the temporal constraints within physics-based motion planning by identifying the compatibilities between temporal and manipulation constraints. The reasoning process tries to simplify the temporal constrains when possible, according to manipulation constraints.

The first part of the thesis concluded that the knowledge-oriented planning (that includes integration of knowledge with physics-based motion planning and a reasoning process over the knowledge) significantly enhances the robot capabilities to perform the given scenarios in complex environments. The particular conclusions are listed below:

- The use of knowledge in physics-based motion planning to define the set of manipulation constraints (that describe how to interact with the objects in the environment) enhances the planning efficiency of the motion planner. Furthermore it modifies the planning behavior of sampling based motion planner by biasing the tree growing towards the valid interaction regions.
- Knowledge-based reasoning within the planning process results in an adaptive way of interaction between the robot and the objects. The computed motion plans are always robust and power-efficient.
- Knowledge-oriented physics-based motion planing provides an easy way to handle the complex temporal goals. It applies the knowledge-based reasoning to evaluate the compatibility between temporal and interaction constraints, and a temporal goal simplification process, in case of incompatibility between them.

The second part of the thesis investigated the possibilities of incorporating (in the planning process) the object pose uncertainties that arise due to the initial sensing of the environment and to complex multi-body dynamic interactions, assuming that no sensory feedback will be provided during the execution. The key challenge was how to handle the uncertainty propagation into the future states due to the robot-object or object-object interactions and how to integrate the information regarding the robustness of the state with the tree-growing process. The study concluded that:

- To compute robust motion plans in the presence of uncertainty requires an evaluation step before adding the controls into the planning tree. The robustness can be evaluated by repeatedly applying the sampled controls by varying the object poses and computing the probability of valid resultant states.
- Knowledge-based reasoning can be used to identify the target region i.e., the part of the workspace where effort should be focused to compute the robust plan. Then biasing the sampling in this region enhances the planning efficiency.
- The uncertainty region of the moved objects could be recomputed by fitting the state distribution (generated during the robustness evaluation phase) with Gaussian-like distributions such as Gaussian mixture models.
- The belief of the controls that are computed during the robustness evaluation phase are integrated with the planning tree, in order to bias the growing towards the states that are more robust.
- The computed motion plans are always robust enough to handle the potential deviation in the pose due to sensing and due to the interactions (robot-object and object-object).

The final part of the thesis presented a knowledge oriented physics-based motion planning tool that is used to evaluate the developed planning strategies. It provides an easy way to incorporate new robots and planning scenes. We have incorporated KUKA LWR, YuMi ABB, TORO DLR and TIAGo PAL.

Along with the conclusions stated above, this thesis also opens new research problems that require further exploration, in order to achieve the ultimate objective of the physics-based planning that is to develop human-like understanding of the physics for robotic manipulation systems. These research directions are described below.

8.2 Future work

In this thesis we investigated the use of knowledge mainly to determine the state validation constraints and appropriate control bounds to interact with the objects. The state validity checker evaluates the set of validity constraints after each propagation step, the state is said to be valid if it satisfies all the given constraints. Similarly, the control bound are also updated at each iteration.

1. One of the possible future research direction could be to replace the validity checker and the control sampler with a learning technique that:
    - Instead of evaluating the validity constraints and control bounds at each propagation step, robot can learn which are the valid interactions and what control force should be applied. This could be done by replacing the state validity checker and the control sampler with deep neural networks. These networks take the current state as input and return whether the state is valid/invalid and what controls are appropriate from this state.
    - Another possibility could be to replace the validity checker and control sampler with cognitive modules of a memory model. Cognitive modules try to synthetically replicate particular aspects of the human brain. A memory model usually consist of a perceptual associative memory that perceives the environment through sensing and develops the association (for the objects in the environment) by coordinating with a long term memory that consists of episodic and semantic memory. The robot will keep storing its experiences in the long term memory and will evolve its behavior with time.
2. To handle uncertainties in the planning process, a future research direction could be to close the loop at execution level for the more robust estimation of the updated uncertainty regions. Moreover to use this sensory feedback to refine the control path using the techniques like linear quadratic regulator.
3. The physics-based temporal logic planning could be extended for the manipulation skill planning in which the robot use knowledge-based reasoning to analyze the situation of the

environment such as *clutter* and *swapping*. Corresponding to different situations, task semantic templates (in terms of LTL formula) can be defined in knowledge. While planning, the robot identifies the situation and executes the corresponding task semantic template to perform the task. This manipulation skills planner could be used as part of high-level symbolic task planner (such as fast forward) in such a way that the task planner focuses on complex high level symbolic plans and the minor low-level details (such as how to swap the objects) should be automatically handled by the manipulation skill planner.

Bibliography

Akbari, A., Gillani, M., and Rosell, J. (2016a). Reasoning-based evaluation of manipulation actions for efficient task planning. In *Proceeding of the ROBOT: Second Iberian Robotics Conference*, pages 69–80. Springer.

Akbari, A., Muhayyuddin, and Rosell, J. (2016b). Task planning using physics-based heuristics on manipulation actions. In *Proceeding of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8.

Akbari, A., Muhayyuddin, and Rosell, J. (2016c). Task planning using physics-based heuristics on manipulation actions. In *Proceeding of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8.

Akbari, A., Muhayyudin, and Rosell, J. (2015). Task and motion planning using physics-based reasoning. In *Proceeding of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*.

Alterovitz, R., Siméon, T., and Goldberg, K. Y. (2007). The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and systems*, pages 233–241. MIT Press, Cambridge, Massachusetts.

Andre, G. (2011). A software architecture for robot control and its application to social robotics. Proceeding of the IEEE International Conference on Robotics and Automation: Workshop on Open Source Software in Robotics.

Antoniou, G. and van Harmelen, F. (2003). Web Ontology Language: OWL. In Staab, S. and Studer, R., editors, *Handbook on Ontologies in Information Systems*, pages 67–92. Springer-Verlag.

Barraquand, J., Langlois, B., and Latombe, J.-C. (1992). Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):224–241.

Berenson, D., Srinivasa, S., and Kuffner, J. (2011). Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460.

Bhatia, A., Kavraki, L. E., and Vardi, M. Y. (2010). Sampling-based motion planning with temporal goals. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 2689–2696.

Bhatia, A., Maly, M. R., Kavraki, L. E., and Vardi, M. Y. (2011). Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, 18(3):55–64.

Bruyninckx, H., Soetens, P., and Koninckx, B. (2003). The real-time motion control core of the orocos project. In *Proceeding of the IEEE International Conference on Robotics and Automation*, volume 2, pages 2766–2771.

Bry, A. and Roy, N. (2011). Rapidly-exploring random belief trees for motion planning under uncertainty. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 723–730.

Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

Connolly, C. I. and Grupen, R. A. (1993). The applications of harmonic functions to robotics. *Journal of Robotic Systems*, 10(7):931–946.

Cosgun, A., Hermans, T., Emeli, V., and Stilman, M. (2011). Push planning for object placement on cluttered table surfaces. In *Proceeding of the IEEE International Conference on Intelligent Robots and Systems*, pages 4627–4632.

Şucan, I. and Kavraki, L. E. (2012). A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics*, 28(1):116–131.

Şucan, I. A. and Chitta, S. (2013). MoveIt! http://moveit.ros.org.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach. In *Proceeding of the Robotics: Science and Systems*, pages 1–6. Ann Arbor, USA.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Diab, M., Muhayyuddin, Akbari, A., and Rosell, J. (2017). An ontology framework for physics-based manipulation planning. In *Proceeding of the ROBOT: Third Iberian Robotics Conference*, pages 452–464. Springer.

Diankov, R. and Kuffner, J. (2008). Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA.

Dogar, M., Hsiao, K., Ciocarlie, M., and Srinivasa, S. (2012). Physics-based grasp planning through clutter. In *Robotics: Science and Systems (RSS)*.

Dogar, M. and Srinivasa, S. (2010). Push-grasping with dexterous hands: Mechanics and a method. In *Proceedings of 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Dogar, M. and Srinivasa, S. (2011). A framework for push-grasping in clutter. In *Robotics: Science and Systems (RSS)*.

Donald, B., Xavier, P., Canny, J., and Reif, J. (1993). Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066.

Du Toit, N. E. and Burdick, J. W. (2012). Robot motion planning in dynamic, uncertain environments. *IEEE Transactions on Robotics*, 28(1):101–115.

Evan, D., John, H., Nathan, K., and Dylan, S. (2010). Extending open dynamics engine for robotics simulation. In *Simulation, Modeling, and Programming for Autonomous Robots*, volume 6472 of *Lecture Notes in Computer Science*, pages 38–50. Springer Berlin Heidelberg.

Fainekos, G. E., Girard, A., Kress-Gazit, H., and Pappas, G. J. (2009). Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352.

Freese, M., Singh, S. P. N., Ozaki, F., and Matsuhira, N. (2010). Virtual Robot Experimentation Platform V-REP: A Versatile 3D Robot Simulator. In Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., and von Stryk, O., editors, *Proceeding of the International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, volume 6472 of *Lecture Notes in Computer Science*, pages 51–62. Springer.

Garrido, S., Moreno, L., Blanco, D., and Monar, F. M. (2010). Robotic motion using harmonic functions and finite elements. *Journal of Intelligent and Robotic Systems*, 59(1):57–73.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

Gottschalk, S., Lin, M., Manocha, D., and Larsen, E. (1999). Pqp–the proximity query package. http://gamma.cs.unc.edu/SSV/.

Gupta, M., MÃijller, J., and Sukhatme, G. S. (2015). Using manipulation primitives for object sorting in cluttered environments. *IEEE Transactions on Automation Science and Engineering*, 12(2):608–614.

Hadfield-Menell, D., Groshev, E., Chitnis, R., and Abbeel, P. (2015). Modular task and motion planning in belief space. In *Proceeding of the IEEE International Conference on Intelligent Robots and Systems*, pages 4991–4998.

Haustein, J. A., King, J., Srinivasa, S. S., and Asfour, T. (2015a). Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3075–3082.

Haustein, J. A., King, J., Srinivasa, S. S., and Asfour, T. (2015b). Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3075–3082.

He, K., Lahijanian, M., Kavraki, L. E., and Vardi, M. Y. (2015). Towards manipulation planning with temporal logic specifications. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 346–352.

Hogan, N. (1984). Adaptive control of mechanical impedance by coactivation of antagonist muscles. *IEEE Transactions on Automatic Control*, 29(8):681–690.

Hsu, D., Kindel, R., Latombe, J.-C., and Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255.

Hsu, D., Latombe, J.-C., and Motwani, R. (1997). Path planning in expansive configuration spaces. In *Proceeding of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2719–2726.

Johnson, A. M., King, J., and Srinivasa, S. (2016). Convergent planning. *IEEE Robotics and Automation Letters*, 1(2):1044–1051.

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

Kim, J. and Khosla, P. (1991). Real-time obstacle avoidance using harmonic potential functions. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 790–796 vol.1.

King, J. E., Cognetti, M., and Srinivasa, S. S. (2016). Rearrangement planning using object-centric and robot-centric action spaces. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 3940–3947.

King, J. E., Ranganeni, V., and Srinivasa, S. S. (2017). Unobservable monte carlo planning for nonprehensile rearrangement tasks. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 4681–4688.

Kupferman, O. and Vardi, M. Y. (2001). Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314.

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., and Karlsson, L. (2014). Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research*, 33(14):1726–1747.

Laskey, M., Lee, J., Chuck, C., Gealy, D., Hsieh, W., Pokorny, F. T., Dragan, A. D., and Goldberg, K. (2016). Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In *IEEE International Conference on Automation Science and Engineering*, pages 827–834.

Latombe, J.-C. (1991). *Robot Motion Planning*. The Springer International Series in Engineering and Computer Science.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

LaValle, S. M. and Kuffner, J. J. (2001). Randomized Kinodynamic Planning. *International Journal of Robotic Research.*, 20(5):378âĂŞ400.

Lavalle, S. M. and Kuffner, J. J. (2001). Rapidly-Exploring Random Trees: Progress and Prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308.

León, B., Felip, J., Marti, H., and Morales, A. (2012). Simulation of robot dynamics for grasping and manipulation tasks. In *Proceeding of the IEEE International Conference on Humanoids*, pages 291–296.

León, B., Ulbrich, S., Diankov, R., Puche, G., Przybylski, M., and Morales, A. (2010). OpenGRASP: A Toolkit for Robot Grasping Simulation. In *Simulation, Modeling, and Programming for Autonomous Robots*, volume 6472 of *Lecture Notes in Computer Science*, page 109âĂŞ120. Springer Berlin / Heidelberg.

Li, Y., Littlefield, Z., and Bekris, K. E. (2016). Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564.

Lim, G. H., Suh, I. H., and Suh, H. (2011). Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(3):492–509.

Lozano-Pérez, T. (1983). Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*, 32(2):108âĂŞ120.

Luders, B., Kothari, M., and How, J. (2010). Chance constrained RRT for probabilistic robustness to environmental uncertainty. In *AIAA guidance, navigation, and control conference*, page 8160.

Mahler, J. and Goldberg, K. (2017). Learning deep policies for robot bin picking by simulating robust grasping sequences. In *1st Conference on Robot Learning*.

Maly, M. R., Lahijanian, M., Kavraki, L. E., Kress-Gazit, H., and Vardi, M. Y. (2013). Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 353–362. ACM.

McMahon, J. and Plaku, E. (2014). Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In *Proceeding of the IEEE International Conference on Intelligent Robots and Systems*, pages 3726–3733.

Melchior, N. A. and Simmons, R. (2007). Particle RRT for path planning with uncertainty. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 1617–1624.

Muhayyuddin, Akbari, A., and Rosell, J. (2015). Ontological physics-based motion planning for manipulation. In *Proceeding of the IEEE International Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–7.

Muhayyuddin, Akbari, A., and Rosell, J. (2016). Physics-based motion planning: Evaluation criteria and benchmarking. In *Proceeding of the ROBOT: Second Iberian Robotics Conference*, pages 43–55. Springer.

Muhayyuddin, Akbari, A., and Rosell, J. (2017a). κ-PMP: Enhancing physics-based motion planners with knowledge-based reasoning. *Journal of Intelligent & Robotic Systems*.

Muhayyuddin, Akbari, A., and Rosell, J. (2017b). Physics-based motion planning with temporal logic specifications. *International Federation of Automatic Control (IFAC)-PapersOnLine*, 50(1):8993–8999.

Muhayyuddin, Akbari, A., and Rosell, J. (2017c). A tool for knowledge-oriented physics-based motion planning and simulation. In *Proceeding of the EAI International Conference on Future Intelligent Vehicular Technologies*. Springer.

Muhayyuddin, Akbari, A., and Rosell, J. (2018a). Knowledge-oriented physics-based motion planning for grasping under uncertainty. In Ollero, A., Sanfeliu, A., Montano, L., Lau, N., and Cardeira, C., editors, *Proceeding of the ROBOT: Third Iberian Robotics Conference*, pages 502–515. Springer.

Muhayyuddin, Moll, M., Kavraki, L., and Rosell, J. (2018b). Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters*, 3(2):712–719.

Pan, J., Chitta, S., and Manocha, D. (2012). Fcl: A general purpose library for collision and proximity queries. In *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3859–3866.

Pfaff, P., Plagemann, C., and Burgard, W. (2008). Gaussian mixture models for probabilistic localization. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 467–472.

Pilania, V. and Gupta, K. (2017). Localization aware sampling and connection strategies for incremental motion planning under uncertainty. *Autonomous Robots*, 41(1):111–132.

Plaku, E., Kavraki, L., and Vardi, M. (2010). Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics*, 26(3):469–482.

Plaku, E., Kavraki, L. E., and Vardi, M. Y. (2013). Falsification of ltl safety properties in hybrid systems. *International Journal on Software Tools for Technology Transfer*, 15(4):305–320.

Prestes, E., Carbonera, J. L., Fiorini, S. R., Jorge, V. A. M., Abel, M., Madhavan, R., Locoro, A., Goncalves, P., Barreto, M. E., Habib, M., Chibani, A., GÃĺrard, S., Amirat, Y., and Schlenoff, C. (2013). Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193 – 1204. Ubiquitous Robotics.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, page 5.

Rosell, J., Alexander, P., Akbari, A., Muhayyuddin, Leopold, P., and Néstor, G. (2014). The kautham project: A teaching and research tool for robot motion planning. In *Proceeding of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*.

Rosell, J. and Iñiguez, P. (2005). Path planning using harmonic functions and probabilistic cell decomposition. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 1803–1808.

Sánchez-Miralles, Á. and Sanz-Bobi, M. Á. (2004). Global path planning in gaussian probabilistic maps. *Journal of Intelligent and Robotic Systems*, 40:89–102.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 639–646.

Stilman, M. and Kuffner, J. J. (2005). Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503.

Stilman, M., Schamburek, J.-U., Kuffner, J., and Asfour, T. (2007). Manipulation planning among movable obstacles. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 3327–3332.

Şucan, I. A. and Kavraki, L. E. (2009). Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer.

Şucan, I. A., Moll, M., and Kavraki, L. E. (2012a). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82.

Şucan, I. A., Moll, M., and Kavraki, L. E. (2012b). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19:72–82. `http://ompl.kavrakilab.org`.

Tenorth, M. and Beetz, M. (2009). Knowrob knowledge processing for autonomous personal robots. In *Proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4261–4266.

Tosello, E., Fan, Z., Castro, A. G., and Pagello, E. (2017). Cloud-based task planning for smart robots. In Chen, W., Hosoda, K., Menegatti, E., Shimizu, M., and Wang, H., editors, *Intelligent Autonomous Systems 14*, pages 285–300. Springer International Publishing.

Tsianos, K. I., Şucan, I. A., and Kavraki, L. E. (2007). Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1(1):2–11.

Vahrenkamp, N., KrÃűhnert, M., Ulbrich, S., Asfour, T., Metta, G., Dillmann, R., and Sandini, G. (2013). Simox: A robotics toolbox for simulation, motion and grasp planning. In *Intelligent Autonomous Systems 12*, volume 193, pages 585–594. Springer Berlin Heidelberg.

Van Den Berg, J., Abbeel, P., and Goldberg, K. (2011). LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913.

Zickler, S. and Veloso, M. (2009). Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *Proceeding of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 27–33.