University of Bonn, Germany

Computer Science Department

Motion Planning Strategy For a 6-DOFs Robotic Arm In a Controlled Environment

universitätbonn

Rheinische
Friedrich-Wilhelms-
Universität Bonn

Angeliki Topalidou-Kyniazopoulou

Thesis Committee

Sven Behnke

Maren Bennewitz

Bonn, August 2017

Acknowledgements

After my diploma studies I realised that my dream was to do research in robotics and I wanted to do that at the University of Bonn, Germany. My parents encouraged me to follow my dream and apply for a Master's degree at University of Bonn. I would like to thank them for always being by my side and supporting me through it all.

I would like to thank my supervisor, Professor Sven Behnke, for welcoming me in the Autonomous Intelligent Systems (AIS) group. He gave me the opportunity to make my dream come true and make it even bigger. He has been supporting me from the first to very last day, for which I am and will be eternally grateful. I would also like to thank him for being my Master Thesis supervisor and having faith in my abilities, even when I did not.

During my time in the AIS group I met and worked with innovative, hard-working and kind-hearted people. I would like to thank every one of them for the memories that we made together and for learning from each other. I will always reminisce my journey in the AIS group with a smile and I will always have a fear of missing out on the new, exciting research adventures that Professor Behnke always organizes for the group.

At last I would like to thank all of my friends in Germany for the time we spend together.

Abstract

Time efficiency in autonomous robotic systems is essential; sensor input modules, perception and decision algorithms require a large amount of computational resources and they have high priority in using these resources. However, robotic systems consist of additional modules that are computationally expensive, such as motion planning. This thesis presents a method that can reduce execution time and consequently required computational resources of motion planning for a 6-DoF's robotic arm that performs autonomously pick and place tasks in a stationary industrial environment. The presented method combines the use of motion plans that are computed off-line and short-distance motions plans, that are computed on-line. Pick and place tasks that are performed in a stationary environment allow us to define a motions' workspace and plan multiple motion trajectories off-line. For the purpose of this thesis *ROS* framework and *MoveIt* library are used. *MoveIt* is a free open-source motion planning library that performs only sampling-based planning. We present how we can use features that *MoveIt* offers and plan with constraints in a reasonable time. We choose a 6-DoF robotic arm because it has a large workspace and *MoveIt* can efficiently compute forward and inverse kinematics solutions for chains up to 6-DoF. Our goal is to show how we can use an existing planning algorithm and take advantage of its favourable features and work around its shortcomings.

Statement of Authorship

I hereby declare that I am the sole author of this Master Thesis and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

Bonn, ...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis Outline

This thesis is focusing on designing a strategy of motion planning for a 6-DOF robotic arm in a controlled environment. The main contribution of this thesis is the presentation of a method that fully exploits the characteristics of a controlled environment and the targeted task specific application.

The main focus of this thesis is to create a strategy for motion planning for a robotic arm for a pick and place task. The results of this method have been used in the STAMINA (**S**ustainable and **R**eliable **R**obotics for **P**art **H**andling in **M**anufacturing **A**utomation) [1] project. STAMINA introduces the use of multiple robots in a industrial car warehouse for collecting engine parts and delivering them to the assembly line. The robots should navigate around the warehouse; identify, collect and inspect the requested engine parts and deliver them to the assembly line in order to get assembled into a new car engine. The use of robots in such a scenario makes the production of custom car engines more affordable and easier to get integrated in an mass production assembly line.

In chapter 2 we present the robots on which we applied this method, the middle-ware used for robot-computer communication and the libraries used in our implementation.

[1] `http://stamina-robot.eu/`

In chapter 3 we present the problem that we try to solve and the purpose of this thesis. Furthermore, we introduce some known approaches that have been used for motion planning.

In chapter 4 we present the approach that we chose as our problem solution and how we came to the conclusion that this is the best suited approach. In addition, we present the implementation of our proposed method and the difficulties we managed to overcome during this process.

In chapter 5 we present the results of the proposed method. Additionally we present the conclusions that we made and suggest some future work.

Chapter 2

Background

In this chapter we present to the reader all the essential knowledge in order for him to understand the discussed topics. We present in detail the robotic systems 2.1 that we used for our results. We also present the platform 2.2 that we used during implementation and its libraries 2.2.3.

2.1 Robotic Systems

Nowadays robots have taken a significant place in our everyday life, from automated coffee machines and vacuum cleaning robots to bomb disposal and industrial robots. They help us live an easier and safer life. The term robot is used for describing a system which can interact with a person or the environment and perform a task with little or no guidance. Robots can be autonomous, semi-autonomous or manually controlled. One might support that there are also pre-programmed robots. A vending machine or a production line robot might fall in this category. We can consider that a robot is pre-programmed if it is only capable of executing one task and if every execution is identical to the first one. However, this type of robots are widely considered as automated systems. A manually controlled robot is a robot that does not perform any task by its own, this kind of robot functions only if a user is controlling it. Most robots used in surgery and bomb disposal are manually controlled and require an experienced operator. A semi-autonomous robot is able to perform by its own with a little guidance by an operator. Autonomous robots perform without any guidance. An operator is necessary only for requesting the desired task for execution. The vacuum cleaning robots are considered autonomous, because the user only requests the cleaning task to begin and the robot creates a map of the environment, decides the route, avoids large obstacles and perceives when the cleaning is done.

2.1.1 UR 10 Robot

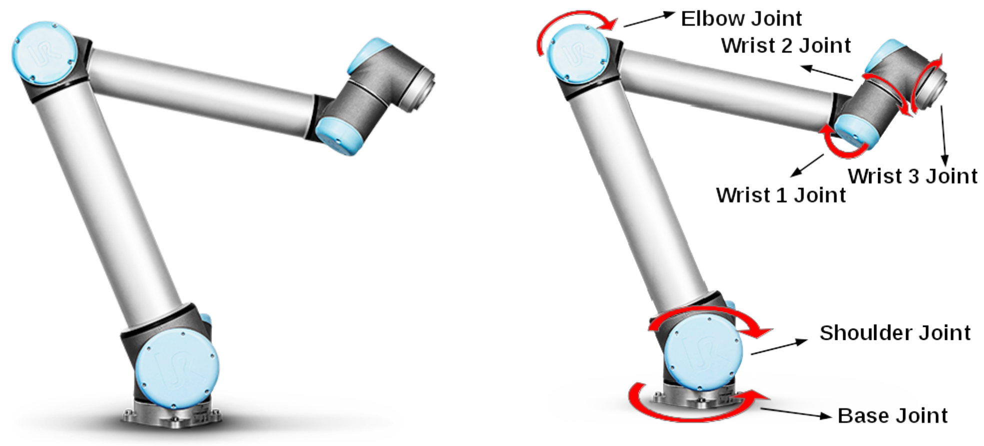


Figure 2.1: Pictures of the *UR 10* robot arm.

The *UR10* [1] robot is produced by *Universal Robots* and it has an on-board controller. It can be controlled manually by a user through an interface or it can run autonomously if desired.

It weighs 28.9kg and has a 190mm diameter footprint. It can reach to 1.3m distance and lift up to 10kg load. The *UR10* has six joints, each joint has 640° rotation range and $120°/s - 180°/s$ speed limit(Table 2.1). It is equipped with force sensors, if any outer force is detected on the arm, it will automatically activate the emergency stop. This feature is available on both modes, autonomous and manual. It does not have any sensors, besides the force sensors, and the tip of the arm does not have any tools attached on it. It is possible to attach a tool, an extension or sensors on the tip due to its design.

When in manual control, the user can move each joint separately or move all joints synchronously by setting a goal position for the end-effector of the arm. A

[1] http://www.universal-robots.com/products/ur10-robot/

Joint	Rotation Range($^\circ$)	Speed Limit($^\circ/s$)
Base Joint	640	120
Shoulder Joint	640	120
Elbow Joint	640	180
Wrist 1 Joint	640	180
Wrist 2 Joint	640	180
Wrist 3 Joint	640	180

Table 2.1: *UR10* joint specification.

teaching mode is also available on the user's interface which allows the user to move the arm by hand to a desired position. However, in this case two users are in need, one to push the teach button on the controller and one to move the arm by hand.

The *UR10* robot can run on automated control as well, the programming can be done at two levels, the *Script Level* and *C-API Level.* In case of *Script Level* programming, the arm is only controlled by a program written in *URScript*, a language developed by *Universal Robots* for the *UR* robot series. The *URScript* language is similar to python, one can use variables, types, functions, flow of control statements etc. It also provides necessary commands for communication and motion control, but it has not extended libraries for motion planning or advance mathematics. If such computations are necessary, it is recommended to write a program in a higher level, the *C-API Level.* At this level all computations can be done in *C* or *C++* and the communication and motion control will be done with a *URscript* program. There is a script available which connects *ROS*(Section 2.2) modules with a *URScript* program. It is available as part of the *ROS-Industrial* repository and more specific the *universal_robot* meta-package[2].

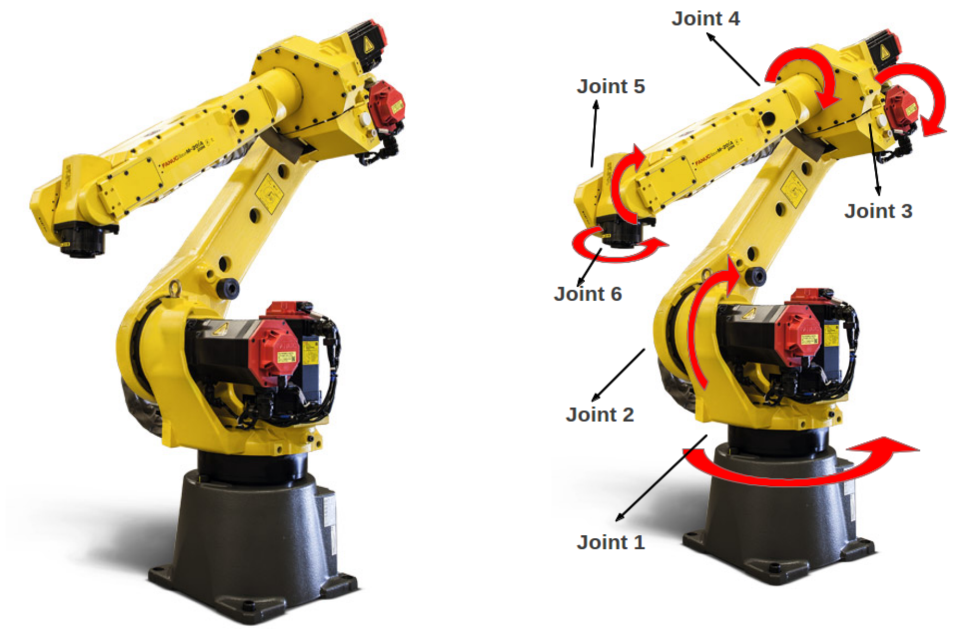


Figure 2.2: The *Fanuc M-20iA/20M* robot arm.

2.1.2 Fanuc M-20iA/20M Robot

The *FANUC M-20iA/20M*[3] robotic arm(Figure 2.2) is manufactured by *FANUC* and is widely used in industrial production lines. It is programmable and manually controlled. It is accompanied by a *R-30iB* controller in a variety of cabinets. *FANUC* provides a large series of robotic arms in a variety of sizes and capabilities.

The *FANUC M-20iA/20M* weighs 250kg and has a 343mm x 343mm footprint. It can lift up to 20kg load and reach to 1.83m distance. It has 6 joints, each joint has a rotation range between $260° - 900°$ and speed limit between $175°/s - 615°/s$(Table 2.2).

The *FANUC M-20iA/20M* arm is controlled by a *R-30iB* controller, which has a user-friendly touch screen, the *iPendant*. The *iPendant* touch screen allows the user to control the robot manually or run an automated program.

[2]https://github.com/ros-industrial/universal_robot
[3]http://www.fanuc.eu/uk/en/robots/robot-filter-page/m-20-series/m-20ia-20m

Joint	Rotation Range(°)	Speed Limit(°/s)
J1	370	195
J2	260	175
J3	460.6	180
J4	400	405
J5	280	405
J6	900	615

Table 2.2: *FANUC M-20iA/20M* joint specification.

In manual control, the user can control each joint independently or move the entire arm at once to a end-effector goal position. The user can define the speed of the arm by setting it as a percentage of the joints' maximum speed. With this method, the motion of the arm retains the design characteristics independently of the chosen speed.

In automated control, the user can choose to run any loaded automated program with the *iPendant*. The *R-30iB* controller supports multiple programs, the user has the ability to run one of them at a time. A step-by-step function is available for all automated programs that are loaded on the controller. The *TU Robotics Institute* provides a program that can be loaded on the *R-30iB* controller and control the *FANUC M-20iA/20M* arm with commands that it receives from a *ROS* system.

2.2 ROS

ROS[4] [1](Figure 2.3) stands for *R*obotic *O*perating *S*ystem and is a free open-source middleware for robotic systems. It provides libraries and software packages for communication, motion control, planning, perception, navigation, mapping, and more. *ROS* is a multi-lingual system with C++ and python as the dominating programming languages. It is heavily supported by the academic and industrial community, which contributes to a reliable and state-of-art system.

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

Any application or algorithm can be easily integrated into a *ROS* system due to its design and communication architecture.

Figure 2.3: The *ROS* logo.

2.2.1 Robot Description Model

The most important part of any robotic system's software is having a realistic representation of the robot's and environment's geometry and structure. In *ROS* systems every geometric characteristic of a robot can be described accurately with the *U*nified *R*obot *D*escription *F*ormat (*URDF*) model, which is an XML format for representing a robot model. With *URDF*, one can describe the dimension and kinematics of every link and joint and any type of sensor that is on the robot. Links can be represented by a simple geometric shape or a more complicated shape with the help of a mesh object.

An accurate *URDF* model contributes to an accurate representation of the environment. Environment representation is very difficult in mobile robotic systems, but good quality sensors in combination with an accurate *URDF* model make it a lot easier. Systematic and unsystematic errors that may occur in perception create a difficulty in representing the environment, thus accuracy in the model and sensors are very important.

2.2.2 Communication

A robotic system, is a large system, which requires multiple calculations and decisions done at once and concurrently. *ROS* solves this problem by having multiple threads and nodes and by providing a communication system among those nodes that run synchronously. A node can run on one or multiple threads. A node may communicate with other nodes by announcing or reading newly obtained information and requesting or stopping an action. Information can be exchanged among nodes in three forms, a message, a service or an action. All messages, services and actions are published on topics (Figure 2.4).

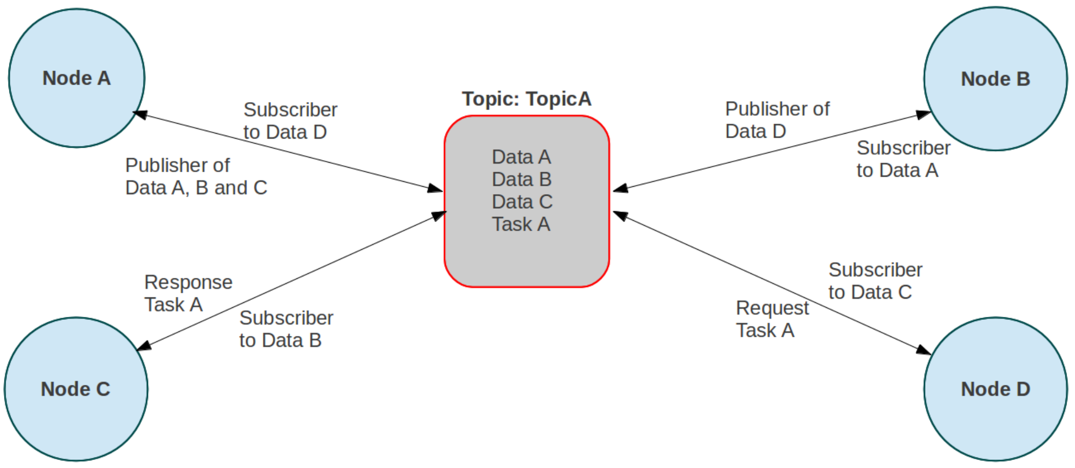


Figure 2.4: *ROS* communication system example.

Topics are named buses used for exchanging messages among nodes. Nodes use topics as announcement boards, they publish data that they need to share and they read information that is helpful to them. They do not know which node is reading their published data or which one published the data that they are interested in. A node may publish or subscribe to multiple topics and topics may have multiple publishers and subscribers. Topics were designed for unidirectional, streaming communication, in case of remote procedure calls a service or an action should be used instead of messages.

2.2.2.1 Messages

A message is published by a publisher and received by a subscriber, nodes may have multiple of both. Each publisher can publish only one type of message. The same rule applies to subscribers as well, one subscriber can read only one type of message. Messages may be of type boolean, float, integer or string. These are the basic message types provided by *ROS*, users may use custom types as well. A large number of *ROS* packages use their own custom, complex, message types for optimized communication. Each message can be timestamped and have a sequence number.

2.2.2.2 Services

A service is a request and reply type of communication. A request is sent by a service client to a service server and a response is sent by a service server back to the client. Nodes may have multiple service clients and servers, each pair uses a unique topic of communication. The communication is considered successfully completed only when the response has reached the service client. Services are used for remote procedure calls and they are blocking, which means that the node will wait for the client to receive a response before continuing to the next command. The request and response part of the service are of type message.

2.2.2.3 Actions

An action is also a request and reply type of communication, but with the option of cancellation and moderation during execution. They require an action client and an action server, as service clients and servers, a node may have multiple of them and each pair uses a unique topic of communication. Actions are used for remote procedure calls, but they are non-blocking, which means that the node may execute commands while its client request is being executed remotely. Actions have 3 parts, goal, feedback and result. An action goal is defined by the action client and it is sent to the action server. The server may send feedback to the client while the task is in execution and it should send a result message as soon as the task is completed. On the client's side, the goal can be cancelled if it necessary, for example, if it takes too long or the feedback is negative.

2.2.3 MoveIt

MoveIt[5] [2](Figure 2.5) is an open-source mobile manipulation software for robots developed at *Willow Garage* by Ioan A. Sucan and Sachin Chitta. It provides solutions for mobile manipulation related problems, such as kinematics, motion

[5]http://moveit.ros.org

planning and control, 3D perception and navigation. The *MoveIt* library is part of the *ROS* packages and it is widely used in robotic systems. Since 2012, it has been used on over 65 robots. What makes *MoveIt* great for developers is that it can be easily configured for any robot. However, it performs best for robotic arms with 6 DOFs(Degrees of freedom) or less.

Figure 2.5: The *MoveIt* logo.

2.2.3.1 *MoveIt* Configuration

There is a graphical interface for configuring *MoveIt* for a new robot, the *MoveIt Setup Assistant*. It uses the *URDF* model of the robot as a base for constructing a *SRDF* (Semantic Robot Description Format) model. The *SRDF* model contains information about pairs of links that will never be in collision, like any adjacent links. This type of information is generated by the *MoveIt Setup Assistant*, but everything else is defined by he user. The user can define one or multiple motion groups. For each group the user can setup an end-effector, a kinematic solver and multiple poses. A link or a joint of the robot may belong to none, one or more motion groups. In case that, a joint does not belong to any of the defined motion groups, it can not be moved with *MoveIt*.

2.2.3.2 Motion Planning

MoveIt uses the Open Motion Planning Library(*OMPL* [6]) [3] which provides a variety of sampling-based planning algorithms. The developer can choose to use either one of *OMPL* algorithms or a custom one, which can be easily integrated into the software. *MoveIt* has only one kinematics solver, the *KDL*, which is a numerical Jacobian-based solver. However, the user can implement his own kinematics solver and integrate it into the software. There is also the option

[6] http://ompl.kavrakilab.org

```
        FollowJointTrajectoryActionGoal action_goal
        FollowJointTrajectoryActionResult action_result
        FollowJointTrajectoryActionFeedback action_feedback
```

Figure 2.6: FollowJointTrajectoryAction Class Description

of generating one with *OpenRave*'s *IKFAST* plug-in[7]. *KDL* and *IKFAST* give better inverse kinematics results for a kinematic chain of 6 or less DOFs. It is recommended to use a custom kinematics solver when having kinematic chains with more than 6-DOFs.

2.2.3.3 Motion Control

MoveIt provides the user with two versions of a motion control manager. The motion control manager can activate a motion, monitor it and cancel it. There is a motion control manager that controls the motors on the robot and there is a fake motion control manager which simulates the motion of the robot's motors. The fake motion control manager can not be used on *Gazebo* (simulator of *ROS*). However, it can be used if only the motion of the robot and not the environment and sensor output are desired to be simulated. When the user wants to simulate the robot motion in a simulated environment, then the non-fake manager and *Gazebo* should be used together. Each *MoveIt* motion group has one dedicated motion controller, since each joint may be part of more than one motion groups, joints may have multiple controllers.

The motion activation, monitoring and cancellation occur with the help of the *ROS* communication system. The desired motion is sent to the control manager, fake or otherwise, in the form of an action. The *FollowJointTrajectoryAction* (Figure 2.6) is defined in the *ROS* package *control_msgs*, which is used to request the execution of a motion from the manager. The action client needs to fill the data for the *action_goal* variable and the motion manager will notify the client during execution with *action_feedback* data and when the action is complete or failed the motion manager will send an *action_result* to the client.

[7] http://docs.ros.org/indigo/api/moveit_ikfast/html/doc/ikfast_tutorial.html

```
Header header
  uint32 seq
  time stamp
  string frame_id
actionlib_msgs/GoalID goal_id
  time stamp
  string id
control_msgs/FollowJointTrajectoryGoal goal
  trajectory_msgs/JointTrajectory trajectory
    Header header
      uint32 seq
      time stamp
      string frame_id
    string[] joint_names
    trajectory_msgs/JointTrajectoryPoint[] points
      float64[] positions
      float64[] velocities
      float64[] accelerations
      duration time_from_start
  control_msgs/JointTolerance[] path_tolerance
    string name
    float64 position
    float64 velocity
    float64 acceleration
  control_msgs/JointTolerance[] goal_tolerance
    string name
    float64 position
    float64 velocity
    float64 acceleration
  duration goal_time_tolerance
```

Figure 2.7: FollowJointTrajectoryActionGoal Class Description

In figures 2.7, 2.8 and 2.9 you will find the class descriptions of FollowJointTrajectoryActionGoal, FollowJointTrajectoryActionFeedback and FollowJointTrajectoryActionResult respectively. Later on, we will focus mostly on how we can compute values for a FollowJointTrajectoryActionGoal variable.

```
Header header
  uint32 seq
  time stamp
  string frame_id
actionlib_msgs/GoalStatus status
  uint8 PENDING=0
  uint8 ACTIVE=1
  uint8 PREEMPTED=2
  uint8 SUCCEEDED=3
  uint8 ABORTED=4
  uint8 REJECTED=5
  uint8 PREEMPTING=6
  uint8 RECALLING=7
  uint8 RECALLED=8
  uint8 LOST=9
  actionlib_msgs/GoalID goal_id
    time stamp
    string id
  uint8 status
  string text
control_msgs/FollowJointTrajectoryFeedback feedback
  Header header
    uint32 seq
    time stamp
    string frame_id
  string[] joint_names
  trajectory_msgs/JointTrajectoryPoint desired
    float64[] positions
    float64[] velocities
    float64[] accelerations
    duration time_from_start
  trajectory_msgs/JointTrajectoryPoint actual
    float64[] positions
    float64[] velocities
    float64[] accelerations
    duration time_from_start
  trajectory_msgs/JointTrajectoryPoint error
    float64[] positions
    float64[] velocities
    float64[] accelerations
    duration time_from_start
```

Figure 2.8: FollowJointTrajectoryActionFeedback Class Description

```
Header header
  uint32 seq
  time stamp
  string frame_id
actionlib_msgs/GoalStatus status
  uint8 PENDING=0
  uint8 ACTIVE=1
  uint8 PREEMPTED=2
  uint8 SUCCEEDED=3
  uint8 ABORTED=4
  uint8 REJECTED=5
  uint8 PREEMPTING=6
  uint8 RECALLING=7
  uint8 RECALLED=8
  uint8 LOST=9
  actionlib_msgs/GoalID goal_id
    time stamp
    string id
  uint8 status
  string text
control_msgs/FollowJointTrajectoryResult result
  int32 SUCCESSFUL=0
  int32 INVALID_GOAL=-1
  int32 INVALID_JOINTS=-2
  int32 OLD_HEADER_TIMESTAMP=-3
  int32 PATH_TOLERANCE_VIOLATED=-4
  int32 GOAL_TOLERANCE_VIOLATED=-5
  int32 error_code
```

Figure 2.9: FollowJointTrajectoryActionResult Class Description

Chapter 3

Problem Statement

In this chapter we present to the reader the problem that we try to solve; the scientific importance of it and the difficulties behind it. Robot motion planning and control is essential in robotics. We are focused on a targeted setup problem. In particular we are interested in presenting a solution of efficient motion planning and control of a robotic arm targeted on performing pick and place tasks on a stationary environment with a high success rate and minimum computing time.

3.1 Sensors and Data Processing

Sensors have a key role in robotic systems. Robots with no sensors in a non-static environment can not move or act safely, thus they can not be autonomous. Sensors collect information about the environment and the status of the robot. It is preferable to use sensors with high frequency, because having the most recent and accurate information is critical. Small time periods allow sensors to collect multiple instances of an area during a very small frame of time. The collected information needs to get processed and evaluated. Usually the data that is collected by the sensors need to be transformed to a convenient format for further information extraction. All the above tasks need a large amount of processing power. This is due to the amount of collected data from a sensor in each period and the fact that most robotic systems have more than one sensor running at time and that small time periods are preferable for getting real time representation of the environment.

Common sensors used in robotic systems are laser scanners, mounted on a fixed or rotating joint, RGB or RGB-D cameras, sonar sensors, force and tactile sensors. Each type of sensors may be targeted for a different use. Force and tactile sensors are used in manipulation of objects. Sonar sensors are used for obstacle avoidance during locomotion. Laser scanners are used for obstacle avoidance, mapping and localization, in combination with RGB sensors laser scanners can be used for object detection and localization as well. RGB-D sensors can be used both for object detection and localization and mapping of small areas. Traditionally, laser scanners have greater measurement range than RGB-D sensors, thus they are a preferable choice for mapping and localization.

Initial processing of sensor data may include elective down-sampling of fields that are out of the main area of interest, data transformation from 3D to 2D or vice versa, neighbourhood extraction according to colour, height, incline or any other feature, obstacle extraction, identification and removal of any robot parts from the data etc.

Initial processing helps in getting a quick first look at the environment and identify the areas of interest that need more in depth processing. It allows to reduce processing time and computing resources by selecting and bounding the areas of interest.

Autonomous robots that can observe the environment and gather accurate data about it, are capable of making safe and optimized decisions regarding their task. Thus, we always try to get more information with more sensors, that are more accurate, faster, can pre-process collected data and adapt to different conditions in lighting, temperature etc.

3.2 Sensor Data Usage in a Robotic System

As mentioned above the amount of sensors used in a robotic system gets larger, as they get cheaper, faster, more accurate and with higher resolution. Having better sensors means that the system is able to collect more accurate data in a shorter period of time. The collected data is mainly used for locomotion and mapping. Locomotion is the module which decides which path the robot is going to follow and finds the current position of the robot on the map. Its name is derived from the words localization and motion, both of them occur at the same time and affect each other. Mapping is the module that creates and maintains the map of the environment. When the robot is in an unknown environment then mapping and localization are happening simultaneously, this is known as SLAM (Simultaneous Localization And Mapping)[4]. If the environment is known then mapping is used for correcting or adjusting the map with newly obtained information.

Both, mapping and locomotion are essential for knowing the current location of the robot and for it to move safely. It is crucially important for locomotion to use a high density map, which means that a large amount of computational resources should be reserved for mapping and locomotion.

3.3 Computational Resources of a Robotic System

Technology is constantly improving and computer processors become more powerful with every new generation. They are capable of making complex calculations with a large amount of data online or at least in very short time. In addition sensors can take more detailed information about the robot's environment which means that the sensor data processing modules need more computational power than before. Having dense data about the environment leads to having more data for mapping and locomotion and consequently needing more computational power for both of them. In order to reserve a large sum of computational power for sensor data processing tasks, mapping and locomotion, we need the rest of the computing modules to be efficient and to require only a few resources.

The software of a robotic system can be divided in groups of smaller programs with regards to the objective task of each program. Each group require a different amount of computational power and run at different time intervals.

As seen at table 3.1 motion planning requires medium amounts of computational power. In addition, it is not used as often as motion control or sensor processing. The medium frequency of execution and the medium amount of computational resources in need may build up to frequent bursts at CPU usage. We can avoid these bursts by using more efficient algorithms of motion planning. It

Programming Module	**Frequency of Execution**	**Computational Resources**
Decision Making	low	low
Motion Control	very high	very low
Communication	high	low
Motion Planning	medium	medium
Locomotion and Mapping	medium	high
Sensor Processing	high	very high

Table 3.1: Robotic system's modules and their characteristics in frequency of execution and computational power requirements.

is easier to reduce the CPU consumption of motion planning rather than of locomotion and mapping, due to their complex algorithms and high density data. Simplifying these algorithms is insufficient and leads to making invalid assumptions about the nature of the environment which means that we end up with unreliable results. On the other hand, motion planning can be efficient in the same way with high or low density sensor data. Sometimes it is feasible to use data with density as low as the width of the robot's smaller movable part. In motion planning all the information that we need to know about the environment is if the robot arm can fit and move safely around a certain area of interest without any collisions.

3.4 Motion Planning

Motion planning is a problem that can be solved with a variety of methods. Sampling-based methods [5] are the ones that are less demanding in computational resources and some of them are also very time efficient. The *OMPL* library(section 2.2.3.2) provides a large variety of sampling-based algorithms. Despite of all of them being sampling-based, they have different characteristics and approaches to the problem of going from position A to position B. Sampling-based planning is time efficient, especially if there is a clear and wide path to the target position. However, it does not always give an optimal solution. In the presence of multiple obstacles between the initial and target position, large delays may occur depending on which sampling-based algorithm is running. It does not work well when having multiple obstacles and in this case it is possible to get a repetitive motion as a solution, which is not desirable. A repetitive motion is a motion that has one or more parts in repetition, i.e. going repeatedly back and forth to an obstacle. This is a deficit that sampling-based planning can not overcome due to its nature. In this thesis we will concentrate on how we can solve this problem without increasing the execution time and efficiency of the motion planning algorithms.

3.5 Motion Planning Improvements for an Autonomous Robot in a Controlled Environment

In this thesis we are going to examine the improvements that we can make on existing motion planning algorithms in order to get better results faster and without increasing the algorithm's computational needs. We are going to work with a 6-DOFs robotic arm that is designed for industrial purposes. We are going to use *ROS* as our operating system. *ROS* provides us with a motion planning library, *MoveIt*, in which we can integrate our own custom programms. We are using *ROS* and *MoveIt*, because both of them are well designed and reliable systems, while allowing developers to easily integrate custom modules. Our goal is to maintain planning time at the lowest possible value without causing large bursts in CPU consumption. Our robotic system uses three to four RGB-D cameras, both short and wide range, which means that most our processing power is reserved for the sensor processing modules. We need to produce feasible, safe, short and stable motion plans for our robotic arm with the use of *MoveIt*'s sampling-based planners.

Sampling-based planners have the disadvantage of not giving optimal solutions and sometimes not giving any solution at all due to bad initializations. We are going try to minimize the effects of this algorithmic behaviour with simple techniques that can be used when the robotic arm is moving in a stationary environment. A stationary environment is an environment that follows some regulations and rules and has none or minimum unexpected elements in it. It allows us to safely make assumptions about it that contribute in simplifying our problem.

Having a stationary environment in an industrial set-up besides the allowing simplifications, it creates some extra specifications that the motion planning module should be fitted to. In particular, the arm's end-effector should always face downwards. The arm is designed for carrying breakable, expensive car parts, which should stay in the arm's gripper and should not fall on the ground. An unexpected flip of the arm's end-effector during a motion is undesired, because it may lead to unsafe conditions. In addition industrial safety rules apply, because the robot will be in a room where people work and their safety around the robot is our first priority.

In most industries robots are widely used, they might be teleoperated or autonomous. In any case there is user that either controls the robot or assists the robot if something goes wrong. We will introduce a system that is designed for autonomous robots and it reduces the need of an assisting human.

3.6 Related Work

Several efforts have been made in order to efficiently reduce planning time for a constrained motion. Sampling-based planning algorithms can not handle planning with constraints in a reasonable time frame, or they can not generate solutions at all. In this section we briefly discuss methods for applying task constraints on known sampling-based motion planning algorithms.

In [6], M. Stilman presents two methods that explore alternative solutions in joint space planning for robotic arms that follow task or workspace constraints. He proposes *Tangent Space Sampling*, in which each RRT (Rapidly-exploring Random Tree) sample is projected into the linear tangent space of its nearest neighbour. The linear tangent space describes the task constraints that the motion plan should follow. Only the samples that follow the task constraints are accepted as valid solutions. In addition he proposes a First-Order Retraction method, in which joint space samples are displaced towards a direction where the constraint violation error is decreased. Displaced joint space samples are accepted only if their constraint error is under a defined threshold.

In [7], a randomized roadmap method is presented for motion planning that produces candidates points that follow task constraints. The proposed method requires preprocessing, in which a roadmap is generated in the configuration space of the robot. The configuration space includes obstacles that the robot should avoid, candidates are are generated on the surface of the obstacle and not inside the obstacle. This stationary configuration can be transformed to dynamic by updating the obstacles, removing the candidates that are currently parts of the obstacles and generate new candidates for the space previous obstacles occupied.

In [8], *C*onstrained *Bi*-directional *R*apidly *E*xploring *R*andom *T*ree (CBiRRT) algorithm for motion planning in configuration space with constraints is presented. This is algorithm is an extension of BiRRT planning algorithm by introducing exploration of the configuration space manifolds that correspond to constraints. In this approach, sampling is performed in configuration space as usual BiRRT planners do, and with the use of projection operation samples are moved onto constrained manifolds, if necessary. In case of multiple constraints the connectivity feature of RRT is utilized for connecting manifolds of different constraints. This method has been used for planning motion trajectories for tasks such as opening a sliding door and moving a dumbbell without making any noise. This method can handle end-effector pose and torque constraints as long as they can be evaluated by a function of the robots configuration,

Chapter 4

Approach and Implementation

In this chapter we present the chosen approach of motion planning for a robotic arm based on the motion plans of the *MoveIt* library for pick[9] and place tasks in a controlled environment.

4.1 Chosen Method

Having a robotic arm in a controlled environment, executing only pick and place tasks helps us reduce the planning time needed for any arm motion. Our set-up is presented in Figure 4.1. As seen, objects for picking can be either in large boxes or flat surfaces. Placing is performed on a on-robot box.

By robot design, it is known that the robot will approach the surfaces and boxes that contain the objects with its left or right side and it will place the picked object on the front portion of the platform. In addition, based on the used sensors, we know that the robotic arm should move closer to the object for the verification of the detection with data provided by a short-distance RGBD sensor mounted on its end-effector. The specifications of the task and robot design create conditions for simplifying the motion planning problem for picking. The workspace for the pick and place task is fully defined in terms of space which helps us design carefully our strategy.

Motion planning can be time consuming for large distance motion paths with a large number of constraints. Our goal is to perform safely a pick task for objects that may be heavy, fragile or have a shape that does not allow very firm

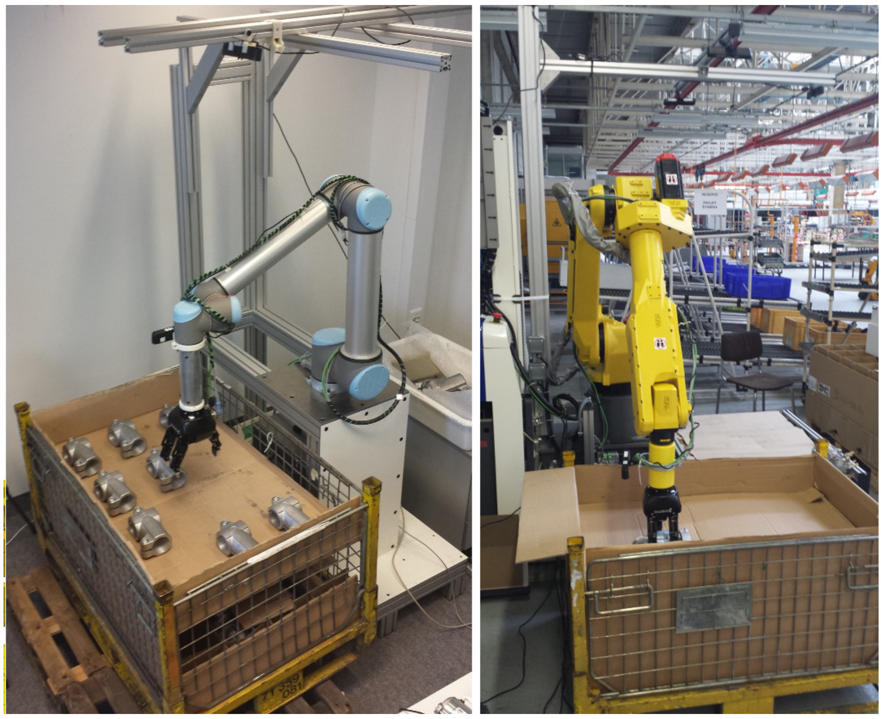


Figure 4.1: Testing environment in the lab and storage environment.

gripping, which creates a lot of constraints in planning motions. The task by itself, constrains the motion of the end-effector, which is order to move safely after grasping an object should move with a stable vertical orientation. Humans try to carry something safely by holding it from the bottom up, in contrast robots do the opposite. Robotic grippers can apply a firm gripping force, which can ensure the stability of the object in hand. However, this stability can change if the pose relation between gripper and object changes during a movement, which can happen if the end-effector changes it orientation with respect to the ground. Gravity affects the applied forces on the object, which means that either the gripper should adjust the applied forces on the object with respect to its orientation and shift of the object's center of mass or the robotic arm should move with the same orientation to the ground until the placement of the object. Hence, we can achieve our goal by constraining the orientation of the end-effector during motion on two axes (x and y), as shown in figure 4.2. In addition, the arm in bounded by the bars above it that support the cameras, which are RGBD

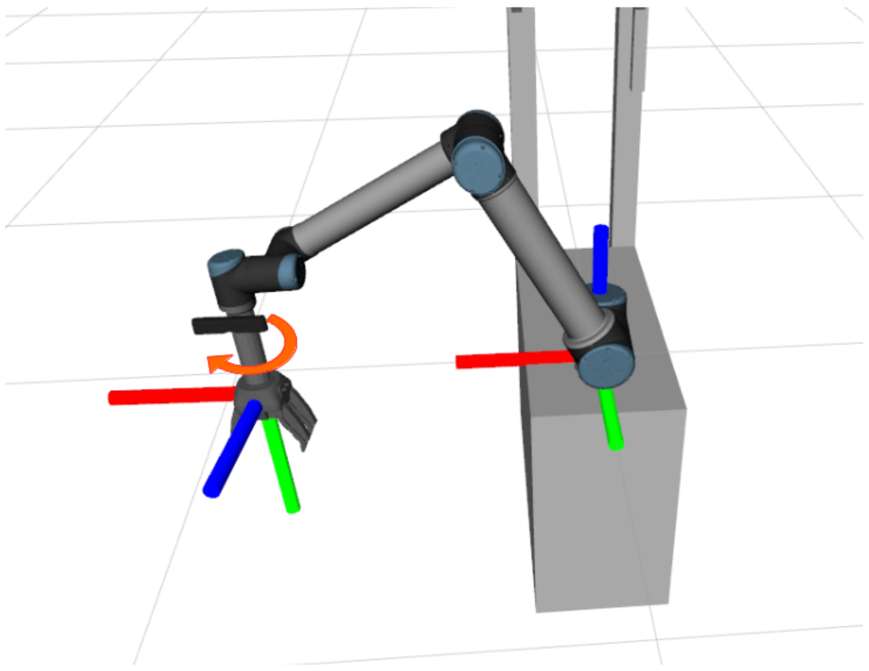


Figure 4.2: Orientation constraint of the end-effector on x and y axes of the global frame.

sensors for indoor environments and can measure depth up to 3.5m. In summary, the workspace of the robotic arm is constrained by its kinematics, placement of the camera supporting bars and end-effector rotation in just axis(z).

Having that many constraints creates large delays in planning with the *MoveIt* library. In particular, having an orientation constraint for the entire motion path requires to get the inverse kinematics solution for every step of the path and confirm that there is no violation of the constraint. Inverse kinematics for a 6-DoF chain is time expensive by itself and having that computed for all points of the motion trajectory is inconvenient and reduces the chance of getting a motion plan for the requested pose transition. The probability of getting successfully a motion plan is reduced to the fact that the motion planning algorithm in use is sample based and that the inverse kinematics solver does not always return the solution closer to the current configuration. Not having the closest IK(Inverse Kinematics) solution also suggests that the path could get very large or the transition between the two configuration is not feasible due to the orientation constraint.

The above mentioned factors helped us come to the conclusion that it would be more efficient to create a grid of poses for the end-effector that describes the workspace of the pick task[9] and plan off-line the respective motions. This strategy allows us to save time and resources, and move safely after grasping an object.

In summary, we suggest to create a grid of end-effector poses that possibly correspond to the necessary poses for object detection verification. Define an initial and final pose for the pick task that are used for every execution and use this trio of poses for determining the pairs of motion plans for off-line computation.

A controlled environment allows us to use pre-computed motions, but this is not sufficient for completing the pick task. Thus, we will concatenate these pre-computed motions with short-distance motions that are computed on-line for achieving maximum accuracy and time efficiency.

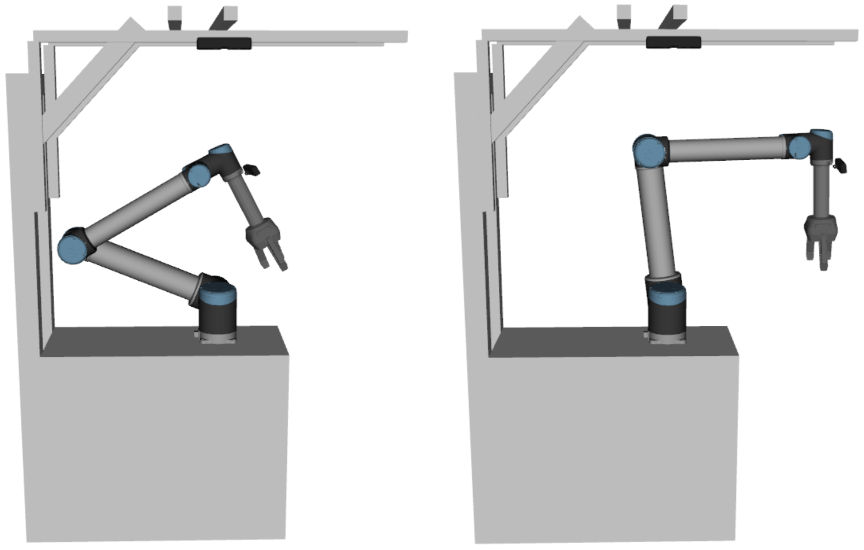
In the following sections of this chapter, we present the implementation of the proposed method in detail.

4.2 Off-line Computation of Motion Plans

The off-line computation of motions for a pick task requires to define the initial and final pose of the task and the coordinates that describe the workspace for the robotic arm and the sampling step of its poses. Once all trios of poses is available, we should compute all feasible motion plans and then we can define the motion pipeline that should be used in a pick task. The motion pipeline is constructed by pre-computed and on-line computed motions.

4.2.1 Initial and Final Pick Task Poses

The selection of initial and final poses is done by complying with some criteria. The initial pose is also the default pose of the robotic arm. Thus, it should be convenient for the locomotion of the robot and it should not occlude with the FoV of the sensors that oversee the pick and place workspace. Convenient for locomotion pose, is a pose that positions the centre of mass of the robotic arm close to the centre of mass of the fixed parts of the robot. Thus, a fully extended arm pose is highly discouraged, because it moves the centre of mass of the robot towards the direction of the extension. In addition, having an extended arm is unsafe during locomotion, due to possible undetected collisions. As shown in Figures 4.3a and 4.4a, a folded pose has been chosen for each robot model.



(a) Initial Pose (b) Final Pose

Figure 4.3: Initial and final pose of the *UR10* arm for the pick task.

The final pose should comply with the end-effector orientation constraint mentioned in the previous section and be convenient for the place task. The picked objects are placed on a box that is the front part of the robot platform. As shown in Figures 4.3b and 4.4b, the end-effector is perpendicular to the ground and close to the place task area.

Having a stable starting and ending point for picking creates a predictable and consisted behaviour, which helps the user or observer to easily recognize any mistakes or mishaps that might occur. It is also common in humans who work in the assembly line of a factory to repeat similar, if not the same, motions for the same task. Thus, the above design is not far from human behaviour.

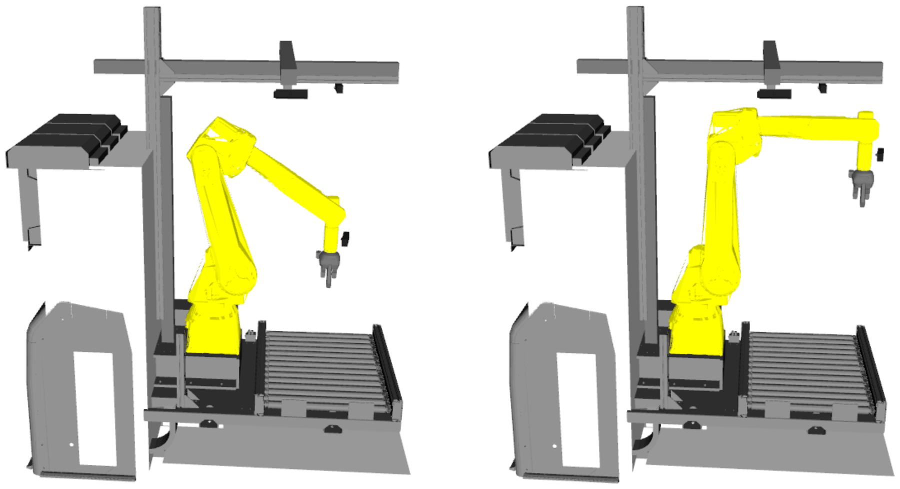

(a) Initial Pose (b) Final Pose

Figure 4.4: Initial and final pose of the *Fanuc* arm for the pick task.

4.2.2 Robot's Workspace as a Pose Grid

As mentioned in Section 4.1 the robot's workspace is constrained by the characteristics of the robotic arm, the design of the robot and the FoV of the sensors. The robot's workspace is defined by the intersection of its visible and reachable area, where it should be able to move safely by identifying possible collisions. Besides the limitations derived by the robot's design, there is one more limitation derived by the storage environment(Figure 4.1), where the picking is performed. The maximum and minimum height, length and width of the boxes and surfaces used, should be consider for the robot's workspace definition. Once the area is defined, we can define the density of pose sampling with regards to the density of objects placed on each surface and the size of the objects. For example, if there are at most 50 objects on a surface of $2m^2$ it is redundant and not optimal to plan 200 motions for that surface. The density of pose samples of the grid is also affected by the size of the gripper in use and the average size of the objects. The gripper in use has approximately $15cm^2$ gripping surface and the objects have an average diameter of $10cm$. Having that in mind, the distance between two samples in the horizontal plane should not be less $5cm$. We can use a different sampling step in the vertical plane, the average height of the objects is $10cm$, so we could use this value as the sampling step or a value slightly larger.

Ideally we want to have one grid pose for one candidate object position for pick up. When the storage surface is at most capacity this is feasible, but when we have fewer objects there are more than one grid poses per object. In any case we are going to choose the most suitable one.

The criteria for the most suitable pose is that it should be at the shortest distance to the target pose and its corresponding motion is collision free for the current instance of the environment. The target pose is defined by the position of the object for picking.

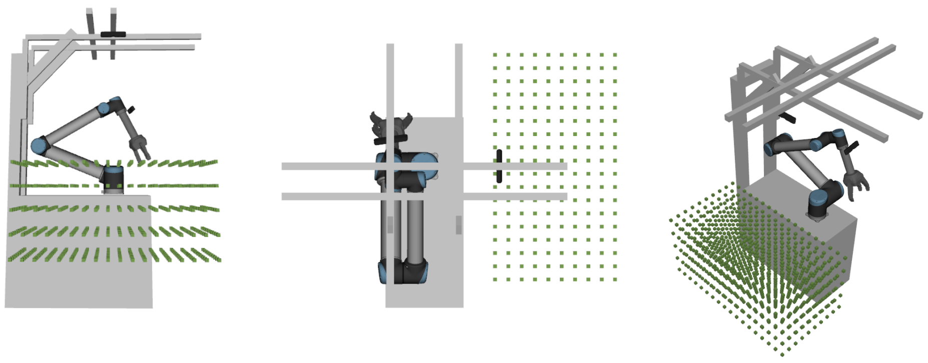


Figure 4.5: The produced workspace grid for the right side of the robot. Each green dot represents one or more poses in the 6-D space.

The program, which computes the sample poses of the workspace, can be used for any *ROS* compatible robotic arm, which is configured for the *MoveIt* library. The user can define maximum and minimum values and sampling step for the position of the pose in the a 3D Cartesian space and a sampling step for orientation sampling in one axis. We sample in a 6D space, because an arm pose is described by its position in a 3D Cartesian space and the orientation(again 3D) of the arm's end effector. Two out the three orientation axes are locked to a certain value, due to the end-effector's orientation constraint. Thus, we define only the rotation of one axes when we sample positions. The user can also define the reference link of the Cartesian coordinates, the name of the generated grid file, and the distance step between the samples both in vertical and horizontal plane. The user can choose whether the workspace will have a rectangular shape or a circular one on the horizontal plane. The generated workspace is stored in a file as a grid of poses, the given name of the grid is stored in the file, and used as its file name as well. The poses are stored as a `geometry_msgs::PoseArray` variable and each pose has type `geometry_msgs::Pose`. The end result is visualised in *RViz* (Figure 4.5).

When choosing the attributes of the workspace, we should consider the size of the gripper, the average size and height of the objects, the size of the robot's workspace and the method of object storage. For every sampled pose the programme requests an inverse kinematics solution(IK), if one is found then the pose is added to the grid. We should also consider that not all sample poses will be reachable by the robot for any motion. The initial pose of a motion plan in combination with the IK of a sample may not produce a successful motion due to the motion constraints. All taken sample poses belong to the workspace of the arm, which means that there is at least one IK solution that can satisfy every one of them. However, it is not possible to plan from any valid initial pose to any valid target pose.

Taking all of the above in consideration in our study, we used a rectangular shape for better approximation of the shape of the boxes. We chose 2 different orientations for sampling because all objects are packed in the boxes the same way. We chose a $7cm$ step on the horizontal plane and we doubled it for the vertical plane, due to the size of the objects and the gripper.

4.2.3 Motion Grid Computation

Once we have defined the robot's workspace, we have all potential poses that will be the arm's target poses. In addition the initial and final poses are used for defining the two planned motions for each pose.

4.2.3.1 Motion Definition

For all potential grid poses we have two motions, one that approaches the grid pose and one that retracts from it. We use two additional poses that serve as an initial pose during the approach and as a target pose during the retraction of the arm. These two poses are used for all planned motions and are presented in section 4.2.1. If a different robot design and sensor layout allow to use the same configuration for the initial and final pose, it is encouraged to do so. This approach can reduce off-line planning time and motion workspace data storage significantly. If we assume that the approach and retract motion take the same amount of time and need the same amount of data storage, we will need only half for each process.

4.2.3.2 Motion Planning with Constraints and Restrictions

Virtual collision objects are used as the representation of storage surfaces during off-line motion planning. It is essential to design motion trajectories with the end-effector higher than the height of the target pose. Due to the nature of the task, we know that there will be an object and a hard surface bellow the target pose. We use virtual collision planes to represent them during planning. During on-line motion planning this is not necessary, because all visible collisions are represented in a 3D collision map that is constructed by RGBD sensors' input.

The process of computing all motions of the generated pose grid takes a large amount of time due to the large number of sample poses. If we consider that for a successful motion plan, the planning algorithm requires in average 0.4 seconds and the cell grid contains 1-3 thousand poses that each requires two plans. In addition, for each successful plan on a pose the file is updated, which is time consuming. We choose to repeat each successful plan at least two times and keep the shortest motion path. In addition when the motion planning module fails we attempt to get a solution 5 times in total. A sample-based planning algorithm can fail to find a solution even if a motion plan is feasible. It has been observed that after the third failed planning attempt, the chances of getting a successfully planned motion are very slim. The most time consuming stage of this method is the failing plans. *MoveIt* allows the user to set a planning time threshold. We have set this threshold to 2.0 seconds, which means that if the planner is not able to find a solution for a pose in our grid it will spend at least 5 times this time trying, in total 10 seconds. Through observation we came to the conclusion that 2.0 seconds allowed planning time was suitable for our set-up.

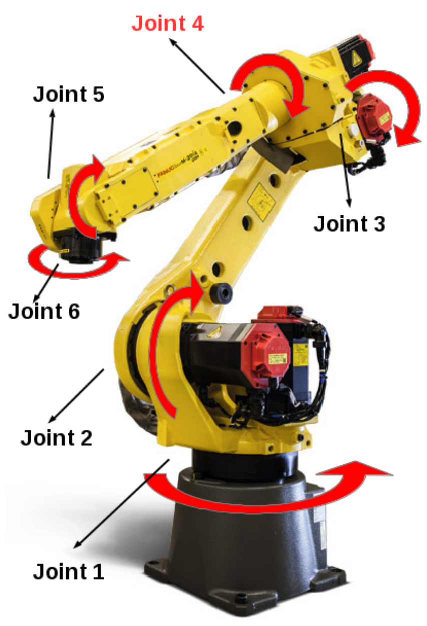
Motion planning for picking tasks is constrained by a variety of factors. The most essential and time consuming constraint, is the end-effector orientation constraint. It is very dangerous to flip the end effector during movement, because the object in hand might fall or move to an unsafe position. If the picked object falls, it might injure a person and in addition the object might get defected or destroyed and possibly unsuitable for further use.

MoveIt gives us the ability of constraining the motion in multiple ways. However, the orientation constraint delays the planning very noticeably and gives us a very low success rate in motion planning, which in consequence makes this feature unusable. Practically we can not use an extra feature that drops the success rate to 12% and increases the planning time by 60%.

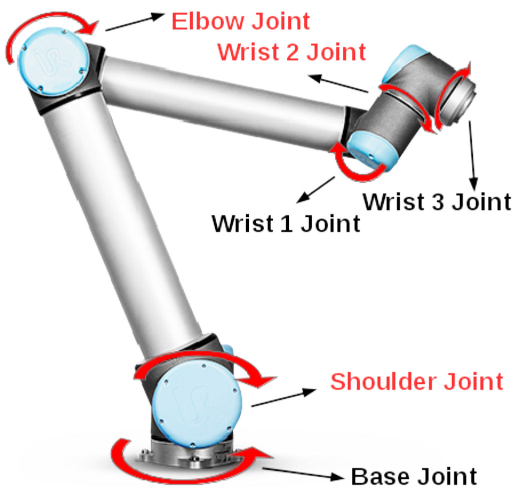
In order to constrain the orientation of the end effector during the motion we use the joint constraints that *MoveIt* offers. We can constrain the motion of each joint separately during the motion, if necessary. In our study we used two robotic arms with 6-DOFs, due to different kinematics and joint range we used different constraints on each arm. The joints with constrained motion range are presented in Figure 4.6

The selection of which joints to constrain on each robotic arm has been made based on observation. Trials have been made, during which we have observed the behaviour of *MoveIt*'s planning solutions. Our decision has been made based on the desired target poses and the failed attempts of the planning algorithm. For the *Fanuc* arm, the planning algorithm failed every time it tried to move *Joint 4* more than 180^o. For the *UR10* arm the planning algorithm failed every time it tried to move the *Shoulder* and *Elbow* joint more than 90^o. In addition, to the *Shoulder* and *Elbow* joints, we noticed that it was essential to constrain *Wrist 2* joint as well, because a large motion on this joint violated the orientation constraint that we aim to apply on the end-effector.

By constraining the motion range of certain joints we achieve the effect of the end-effector orientation constraint with lower time cost and produce motion plans that fit our needs. Restrictions in joint space are significantly cheaper in computation than constraints in Cartesian space. Cartesian space constraints



(a) *Fanuc* Arm



(b) *UR10* Arm

Figure 4.6: The constrained joints are in red font.

require to compute IK solutions for every computed point of the motion path, which is highly time consuming and has low success rate. The success rate is low because, the motion planning algorithm can not accept any IK solution, it requires a solution that is very similar to the one of the previous point of the path.

4.2.3.3 Valid Motion-Pair of Motion Workspace

For every grid pose we need a pair of motion trajectories. Both of these motions should be successfully planned in order to consider that the grid pose is reachable. In the case that, the approaching motion could not be planned, we abort the retraction motion planning. If the approaching motion is planned successfully, but the retraction motion is not, then this grid pose is considered unreachable. We plan in pairs because it serves better the robot behaviour of the picking task.

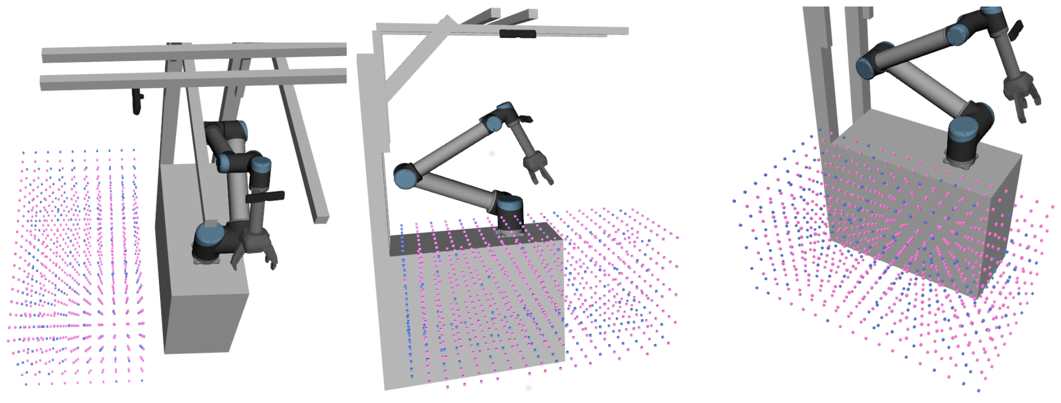
Having one or more kinematic solutions for one pose does not guarantee that there will be a path to and from that pose. The necessity of a pair of motions increases the probability of failing to plan for a pose. One solution for that problem could have been to use the same pose for the beginning and the completion of the task. In our case this is not advisable due to the design of the platform, arm, objects and task itself. However, if it is possible to use the same pose as the initial pose during the approach and the final pose during the retraction, it is advised to do so. Using only one reference pose simplifies the problem by planning one motion, instead of two as we can revert the first motion and produce the retraction motion. It also decreases the possibility of failing plans and the planning time.

The initial pose of the arm should be a pose that is safe when the platform is moving around the factory. The arm has to be folded in a position close the centre of mass of the platform and it should not occlude the view of the cameras that are monitoring the environment for new objects and obstacles. All of the above requirements of the initial pose contradict the requirements of the intermediate pose. In which the main focus is the collision free state of the object-on-hand, satisfying all potential objects in hand, and the object's safe transfer during the place task.

4.2.4 Motion workspace Representation

We designed a grid of cells in order to store the pairs of planned motions with their respected poses. A grid of cells has a name and an array of cells. Each cell consists of a pose and two trajectories, one to approach the pose and one to retract from it. Both of the cell trajectories have been planned by following the constraints mentioned in Section 4.2.3.2.

In Figure 4.7 is shown the cell grid that represents the motions' workspace. In pink we can see the poses, for which two motion were successfully planned and in blue the poses for which it not possible to plan one of the two necessary motions. For all poses that are presented in Figure 4.7 the robotic arm has at least one IK solution, however not all of them can be reached due to the constraints that we apply during motion planning as discussed in Section 4.2.3.2. All visualized



(a) Front View (b) Side View (c) Angled View

Figure 4.7: Pink dots represent the poses that got a successfully planned pair of motions and blue dots represent the poses that failed to do so.

poses belong to the grid pose that represents possible approach poses. The cell grid is stored with *YAML* in a file and it is loaded once when the pick and place task runs for the first time.

4.2.5 Verification of Precomputed Motions

It is essential to verify that all computed and stored motions are valid and they can be executed without causing any problems. Thus, we have added an extra verification step to this procedure. *MoveIt* does not produce motion plans that are in self-collision, but it might be in collision with the virtual collision objects if they were not placed correctly. or there was a synchronization problem. It is possible to have synchronization issues when using *ROS*. If we publish on the same cycle the addition of a collision object and request a motion plan, it is uncertain which one will be executed first or if multiple objects are published then one of them may get discarded, thus extra verification is reasonable to performed.

As an extra verification step, we check the motion plans of all poses of the generated grid for collisions with virtual collision objects. For each pose in the grid we place a virtual collision object where the object for picking would have

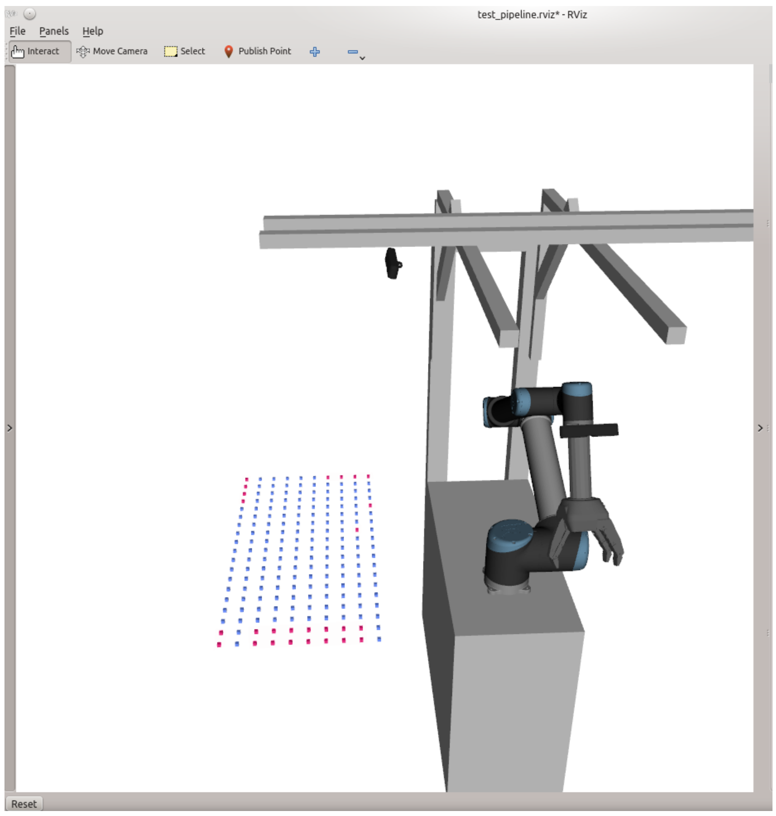

test_pipeline.rviz* - RViz
File Panels Help
Interact Move Camera Select Publish Point
Reset

Figure 4.8: The poses with successfully planned motions are presented in blue colour and the ones that failed in red. Representation of one layer.

been and simulate the picking task. If a collision is detected or simulation fails, then this pose is removed from the grid.

4.3 System and User tools

A number of Graphical Interfaces have been developed in order to configure the predefined motions' workspace and compute the motions themselves. In particular, a Trajectories Editor, a Motion Grid Editor and a Motion Design Verification interface have been developed.

4.3.1 Trajectories Editor

An *RViz* plugin has been developed for creating and saving a motion for a robotic arm. This editor is useful for storing motions that are performed multiple times in a run, so instead of designing them every time we only check for collisions before execution. In our controlled environment in particular, the robotic arm follows a number of pre-determined points in its workspace, which means that some motions between these pre-determined points can be planned once. In particular, the robotic arm after the completion of the place task returns to the final pose of the pick task. As mentioned above, the robotic arm should be in its initial pose, thus we used this tool for designing and storing the transition motion for the final to the initial pose. This motion is used once per pick and place task.

4.3.2 Grid Visualization Plugin

The Pose Grid Visualization Plugin that has been implemented allows the user to define the size, density and shape of the grid. We offer a rectangular box or a sphere as the grid shapes. The density of the grid depends on the distance sampling between the samples on all 3 axes. The plugin computes and visualizes the workspace of the pose grid and the samples that can be reached by the robotic arm. This tool is used for creating the pose grid, the visualization is necessary only for user verification of the given parameters.

4.4 Integration of Precomputed Motions in the System

In this section we present how the pre-computed motion plans can be utilized with maximum gain in a large and complex robotic system. Our method has been integrated in the system presented in [10]. However, the system does not fully rely on the pre-computed motion plans, but it can plan a full motion plan if our method fails to find a suitable plan.

4.4.1 Pick and Place Task

The pre-computed plans have been integrated in the pick and place task. This task can be divided in 6 stages, object detection, object approach and verification, object grasping, object picking, placing position detection and object placing. The pre-computed plans have been used at the stages of object approach and verification and object picking.

The algorithm chooses an object for picking, the robotic arm should approach the object so it can use a close range RGB-D camera and verify that the selected object is the correct object. The motion plan used for the approach is a combination of two plans if possible. If there is a pose in the cell grid in a distance of 10cm from the approach pose and the pre-computed motion path does not collide with any obstacles in the current scene then this precomputed motion path will be used. In addition, a motion plan is computed on-line for reaching the final approach destination and we concatenate the two motions in one before execution. Object grasping is performed by using only on-line planning, because grasp poses depend on the pose and type of the object. The pick and place task is performed on a variety of objects thus, it is not possible to compute off-line motion plans for grasping. Object picking is concluded with on-line planned motion from the pose to the chosen pose of the grid and its respective retract off-line planned motion. Again the two motions are concatenated in one before execution. After the completion of the pick task, the place task is performed. In the place task the pre-computed motions are not used, because the placing position of the object depends on the type of object and free space of the on-robot placing box.

Figure 4.9 shows how the pre-computed and on-line computed trajectories are used together in one motion.

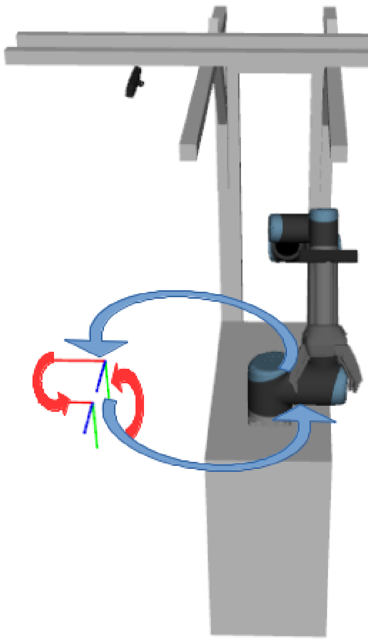


Figure 4.9: the on-line planned trajectories are shown in red and the pre-computed plans are shown in blue.

Chapter 5

Conclusions and Future Work

In this chapter we present the results of our method, the conclusions that we arrived to and any future work that could optimize our method.

5.1 Results

We performed 50 repetitions of planning a motion using one initial configuration and one target pose. We chose four different configurations for the same motion. First configuration uses the proposed method with the pre-computed motion plans in a cell grid. The second configuration uses *MoveIt* with joint constraints, as we used for computing the pre-planned motions. The third configuration uses *MoveIt* without introducing any motion constraints on the planner. Finally, the fourth configuration uses *MoveIt* and applies an orientation constraint on the end-effector. We present the planning time of each method and their success rate.

For the purpose of this experiment we used the UR10 arm in simulation, we did not include any obstacles in the workspace of the robot and chose an arbitrary target pose that follows the applied orientation constraint. For our proposed method, we used a pre-computed cell grid that has motion plans for 2.645 poses, and none of them matches the target pose. The chosen target pose and the initial configuration is shown in figure 5.1.

In Table 5.1, we show the number of successful plans for each configuration. We should mention that for all configurations we used the same sampling-

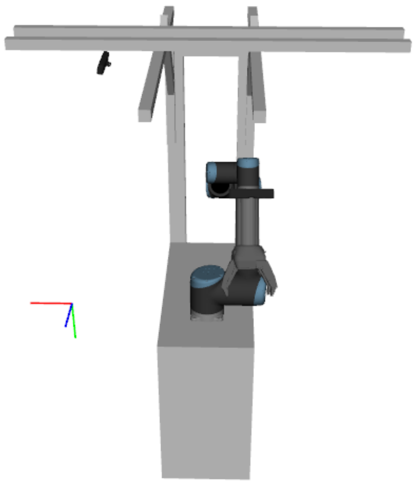


Figure 5.1: The target pose is shown as an axis and the current configuration of the robot is the initial configuration of the motion plan.

based planner, RRTkConfigDefault (Rapidly-exploring Random Tree), we allowed *MoveIt* to plan each motion for maximum 2.0$secs$, and do only one planning attempt per trial. In addition, the same IK solver has been used.

Planning Method	Successfull Plans	Failed plans
Pre-computed Plans	47	3
MoveIt with joint constraints	48	2
MoveIt without constraints	46	4
MoveIt with orientation constraints	6	44

Table 5.1: Motion Plan success rate for all three configurations used in the experiment.

As shown in Table 5.1, the least successful approach is to use *MoveIt* with an applied orientation constraint on the end-effector. We managed to get only 6 out of 50 motion plans, and all of them required the maximum allowed planning time (2.0*secs*). In case of larger planning time the results may be more successful for planning with orientation constraints. Using *MoveIt* without any planning constraints appears to be more successful than the above configuration. Four out of 50 motion plans failed, which is acceptable, since we performed only one planning attempt per trial. We should take in consideration that the configuration for the target pose affects the result of motion plans, since there might be more than one configuration for the target pose. We can expect that one or more of them may not be reachable from the initial configuration. We get similar number of failed motion plans for pre-computed plans and planning with joint constraints, as well. The later configuration has been the most successful with two failed motion plans out of 50 trials, it has only one less failed trial than our proposed method with pre-computed plans, which had three failed planning attempts.

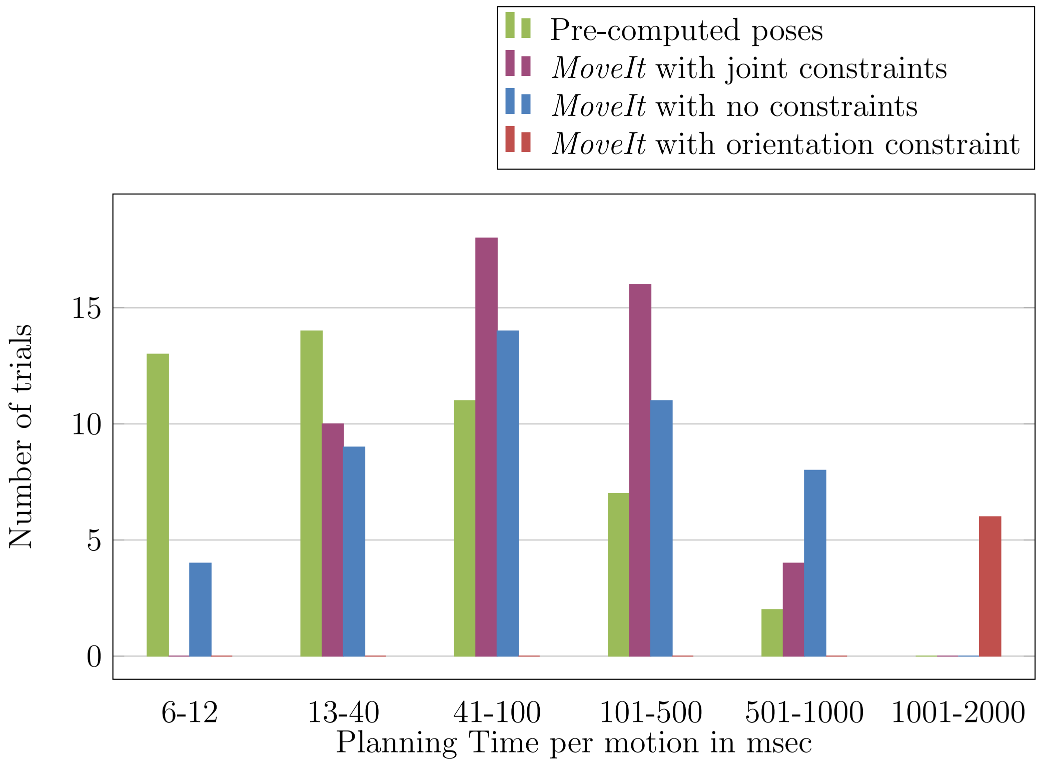


Figure 5.2: Planning time of a motion plan for 3 different planning configurations.

Figure 5.2 presents the planning times of all successful motion plans per configuration. The method that uses pre-computed plans requires significantly less planning time than the other three methods. This is explained fairly easy, the planning distance on this experiment for the other three methods is more than 13 times larger than the minimum distance for this method.

5.2 Conclusions

Based on the results presented in section 5.1 we can conclude that the proposed method reduces planning time for a specific manipulation task.

We managed to use an existing motion planning library (*MoveIt*) and exploit all of its desirable features and compensate for its shortcomings. We used *MoveIt* collision space computation module, which allows to add and remove collision objects and define allowed collisions with obstacles if necessary. In addition, *MoveIt* can detect and avoid self-collision for any robot and plan collision free motion trajectories. We managed to use joint constraints that are computationally inexpensive in contrast to orientation constraints and generate motions that mimic the effect of an orientation constraint in a reasonable time frame.

Our proposed method works very well for a defined task, but it can not be used for any manipulation task as is. In case of having to perform several manipulation tasks that require different constraints or different task workspace, separate configurations are required for joint constraints and motion workspace during pre-processing and on-line motion planning. However, that does not make our method non-transferable, but more complicated to use if several tasks are required to be performed by one autonomous robot.

We introduced an alternative use of an known and widely used motion planning library that makes motion planning less demanding during task execution.

5.3 Future Work

We propose as future work, a GUI that will guide the developer to do all necessary steps to get the pre-computed plans and generate the motion plans. In particular, the user would be able to load a robot model, define its joint constraints, initial pose and final pose if necessary; and motion workspace that corresponds to a specific task. This is part of the workspace and motion planner configuration. After the completion of the configuration the user would be able to automatically generate the grid of target poses and the modified planner that uses all defined joint constraints. It would be useful if the user could test the modified planner before enabling the generation of the motion plans. As the last step we suggest the generation of the task's cell grid, the grid with the target poses and corresponding motion plans to go to this pose and retract from it.

This kind of interface would make the proposed method easily transferable to different robots and different tasks. In addition, it would be possible to use this method for multiple tasks with very little effort by the user/developer.

References

[1] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. Volume 3. (2009) 5 18

[2] Chitta, S., Sucan, I., Cousins, S.: Moveit![ros topics]. Robotics & Automation Magazine, IEEE **19**(1) (2012) 18–19 21

[3] Sucan, I.A., Moll, M., Kavraki, L.E.: The open motion planning library. IEEE Robotics Automation Magazine **19**(4) (Dec 2012) 72–82 22

[4] Dissanayake, M.W.M.G., Newman, P., Clark, S., Durrant-Whyte, H.F., Csorba, M.: A solution to the simultaneous localization and map building (slam) problem. IEEE Transactions on Robotics and Automation **17**(3) (Jun 2001) 229–241 29

[5] Tsianos, K.I., Şucan, I.A., Kavraki, L.E.: Sampling-based robot motion planning: Towards realistic applications. Computer Science Review **1**(1) (August 2007) 2–11 30

[6] Stilman, M.: Task constrained motion planning in robot joint space. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. (Oct 2007) 3074–3081 32

[7] Amato, N.M., Wu, Y.: A randomized roadmap method for path and manipulation planning. In: Proceedings of IEEE International Conference on Robotics and Automation. Volume 1. (Apr 1996) 113–120 vol.1 33

[8] Berenson, D., Srinivasa, S.S., Ferguson, D., Kuffner, J.J.: Manipulation planning on constraint manifolds. In: 2009 IEEE International Conference on Robotics and Automation. (May 2009) 625–632 33

[9] Holz, D., Topalidou-Kyniazopoulou, A., Stuckler, J., Behnke, S.: Real-time object detection, localization and verification for fast robotic depalletizing. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. (Sept 2015) 1459–1466 34, 37

[10] Holz, D., Topalidou-Kyniazopoulou, A., Rovida, F., Pedersen, M.R., Kruger, V., Behnke, S.: A skill-based system for object perception and manipulation for automating kitting tasks. In: Emerging Technologies Factory Automation (ETFA), 2015 IEEE 20th Conference on. (Sept 2015) 1–9 48