Robotics and computer vision system for simulated pick-and-place tasks within the ROS/Gazebo/MoveIt framework

A semester project in the course of
Robotics and Computer vision

written by

Martin Androvich
marta16@student.sdu.dk
92411942

Daniel Tofte Schøn
dscho15@student.sdu.dk
84929349

University of Southern Denmark (SDU)
Technical Faculty (Faculty of Engineering)

December, 2021

Abstract

The ROS/Gazebo/MoveIt framework is used to setup a workcell within a simulated environment and develop a vision-based pick-and-place pipeline. The workcell consists of a table with a designated pick and place area, obstacles, graspable objects, perception sensors, and a UR5 manipulator with an attached WSG50 gripper.

A reachability analysis indicates an optimal base pose to be in the center of a table, where objects are grasped from the side. For trajectory generation, both interpolation-based and sampling-based methods are implemented and evaluated. The Probabilistic Roadmap (PRM) planner shows the most promising results with a mean planning time of 0.829 ± 0.255 ms, and mean trajectory duration of 5.73 ± 1.09 s.

For pose estimation, an image-based and a depth-based approach is implemented and evaluated using Monte Carlo simulations. Using a binary classification metric of whether an estimated pose can be kinematically grasped, the depth-based method shows a 65 % to 100 % accuracy, significantly outperforming the image-based method at 52.7 % to 72.6 % accuracy.

The integrated system uses RANSAC Registration with ICP for pose estimation, and a PRM planner for generating collision-free trajectories. Evaluations show that, given a workcell with obstacles and a Milk object, a pick-and-place task is successful $\geq$ 80 % of the time, with an average Euclidean xy-error of 0.0179 ± 0.0262 m with respect to desired and measured place position.

A demonstration is available at `https://www.youtube.com/watch?v=1jAmozyywgk`.

Contents

Contributions

The contributions to this project are summarized in Table 1, where each major section has a primary assignee. However, nearly all sections have been contributed to by both authors to some extent, both in writing and development, since the project has been developed in close cooperation; the summaries given in Table 1 may therefore be misleading and do not reflect the actual workload distribution of this project.

Section	**Primary assignee**
1 Introduction	Martin Androvich
2.1 Structure	Martin Androvich
2.2 Model representation	Daniel Tofte Schøn
2.3 Objects and sensors	Daniel Tofte Schøn
2.4 Manipulator: Robot description	Daniel Tofte Schøn
2.4 Manipulator: Kinematics and dynamics	Martin Androvich
2.4 Manipulator: Forward kinematics	Martin Androvich
2.4 Manipulator: Inverse kinematics	Daniel Tofte Schøn
2.4 Manipulator: Dynamics	Daniel Tofte Schøn
2.4 Manipulator: Robot control	Martin Androvich
2.5 Gripper (end-effector)	Daniel Tofte Schøn
2.5 Gripper (end-effector): Tool center point	Martin Androvich
3.1 Reachability	Martin Androvich
3.2 Interpolation-based trajectory generation: Introduction	Martin Androvich
3.2 Interpolation-based trajectory generation: Linear interpolation	Daniel Tofte Schøn
3.2 Interpolation-based trajectory generation: Parabolic interpolation	Daniel Tofte Schøn
3.2 Interpolation-based trajectory generation: Evaluation	Martin Androvich
3.3 Sampling-based motion planning: Planning interface	Martin Androvich
3.3 Sampling-based motion planning: Evaluation	Daniel Tofte Schøn
3.4 Method selection	Martin Androvich
4 Pose estimation: Introduction	Martin Androvich
4.1.1 Methodology	Daniel Tofte Schøn
4.1.2 Evaluation	Martin Androvich
4.2.1 Methodology	Martin Androvich
4.2.2 Evaluation	Daniel Tofte Schøn
4.3 Method selection	Daniel Tofte Schøn
5.1 Integration	Martin Androvich
5.2 Evaluation	Daniel Tofte Schøn

Table 1: Approximate overview of contributions to the sections of this project.

Acronyms

COM center of mass.
DH Denavit-Hardenberg.
EE end-effector.
EST Expansive Space Trees.
ICP Iterative closest point.
KDL Orocos Kinematics and Dynamics Library.
OMPL Open Motion Planning Library.
PRM Probabilistic Roadmap.
ROS Robot Operating System.
ROVI robotics and computer vision.
SBL Single-query, Bi-directional, Lazy Roadmap.
SDF Simulation Description Format.
TCP tool center point.
URDF Unified Robot Description Format.

Terms

pose The position and orientation of a coordinate frame.

reachability A metric for a manipulator's capability of reaching a desired tool center point (TCP) pose with respect to the number of collision free kinematic configurations for a given Cartesian configuration.

robot dynamics variables The variables of the joint space robot dynamics for a given robot state, including the mass matrix, $\mathbf{M}(\mathbf{q})$, Coriolis matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, and gravity vector, $\mathbf{g}(\mathbf{q})$.

robot state The state of the robot in terms of joint space position, $\mathbf{q}$, and velocity, $\mathbf{q}$.

robotics and computer vision (ROVI) The robotics and computer vision system is an integrated, vision-based pick and place pipeline.

rotatum The derivative of torque with respect to time, i.e., the rate of torque.

1 Introduction

1.1 Project description

Industrial robotics often have setups with pre-defined locations for pick-and-place tasks, in which the objects are contained in specialized holders in order to prevent any disturbances in their pose (position and orientation). Such a setup allows for robust manipulation, but increases the complexity and cost of the system, especially at scale.

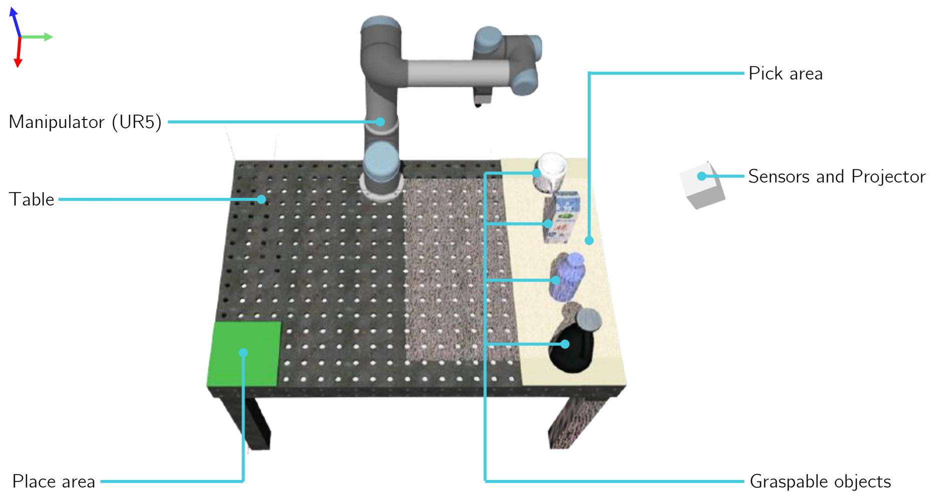


Fig. 1: The robotics and computer vision (ROVI) system workcell in the Gazebo simulation environment, comprised of the UR5 manipulator mounted on a table with designated pick-and-place areas, graspable objects, and sensors.

The robotics and computer vision (ROVI) system makes use of the ROS/Gazebo/MoveIt framework [1, 2, 3] in order to establish a simulated workcell, allowing to systematically design and evaluate a computer vision-based pick-and-place pipeline. A workcell consists of a UR5 manipulator mounted onto a table with designated pick-and-place areas, objects, obstacles, and various perception sensors, as shown in Fig. 1.

1.2 Overview

Design of the ROVI system is composed of four major components:

- **Simulation environment**
  A simulation environment and workcell are established within the ROS/Gazebo/MoveIt framework, defining the UR5 manipulator, graspable objects, obstacles, and perception sensors.

- **Planning**
  A reachability analysis is implemented to determine the optimal base pose within the workcell. Interpolation-based and sampling-based planning is implemented and evaluated.

- **Vision-based pose estimation**
  Image-based and depth-based pose estimation methods are implemented and evaluated, allowing to estimate the pose of objects within the pick area of the workcell.

- **Pick-and-place (integration)**
  A planning approach is combined with a vision-based pose estimation approach to establish a complete pick-and-place pipeline in which the manipulator can grasp and relocate an object without any collisions.

2 Simulation environment

A simulated environment provides the necessary platform for development of the ROVI system. It consists of: a simulated workcell (manipulator and objects/obstacles), robot control, perception (cameras and point cloud sensors), motion planning, and computer vision.

Development of such a system greatly benefits from a modular project structure, as well as access to common tools within the robotics domain (such as libraries for motion planning, computer-vision etc.). Robot Operating System (ROS) [1] is a state-of-the-art robotics framework and tends to be a standard for robotics application development [4]. It provides a modular software framework with reusable components and a common communication pipeline, while also seamlessly integrating with the Gazebo simulator [2].

2.1 Structure

The ROVI system project is structured using ROS packages and meta-packages, all of which is built within a `catkin` workspace [5]. A package encapsulates whatever necessary (code, scripts etc.) to constitute a useful module, whereas a meta-package groups several packages specific to some domain.

For example, the `ur5_ros` meta-package encapsulates the integration of the UR5 manipulator into the ROS/Gazebo ecosystem, in which the `ur5_planner` package provides motion planning functionality for the UR5 manipulator.

The ROVI system consists of three groups of packages, as shown in Fig. 2. The integration of the system is implemented in the `rovi_system` package, in which all other packages are used to constitute a full pick and place pipeline. The package also contains any experiments or examples of the ROVI system.

Package	**Description**
`rovi_system` meta	**Packages related only to the ROVI project.**
`rovi_system`	Integration of packages to provide a full pick and place pipeline.
`rovi_models`	Gazebo models and worlds for the the ROVI workcell.
`rovi_planner`	Interpolation-based trajectory generation.
`rovi_vision`	Computer vision-based pose estimation methods for objects in the ROVI workcell.

`ur5_ros` meta	**Integration of UR5 manipulator into ROS/Gazebo/MoveIt ecosystem.**
`ur5_controllers`	ROS Control controllers and interface for the UR5 robot.
`ur5_description`	URDF descriptions of the UR5 robot.
`ur5_dynamics`	Dynamics and kinematics of the UR5 robot.
`ur5_gazebo`	Gazebo interface for the UR5robot.
`ur5_moveit_config`	MoveIt! configuration for the UR5 robot.
`ur5_planner`	Motion planning and reachability analysis using MoveIt for the UR5 robot.
`wsg50`	Model, controller, and interface for the WSG50 gripper.

`ros_utils` pkg	Utilities for interfacing ROS/Gazebo/MoveIt/Eigen etc.

Fig. 2: Packages of the ROVI system.

This approach enables a decoupled development process, where packages are either completely decoupled or only loosely coupled, allowing packages to be developed and tested independently.

2.2 Model representation

Any entities to be dynamically simulated in Gazebo (manipulator, table, graspable objects etc.) must be modeled by describing their kinematic, visual, collision and dynamic (center of mass (COM), inertia, mass) properties.

Both the robot description and objects are modeled using XML-based file formats [6], namely Unified Robot Description Format (URDF) and Simulation Description Format (SDF), allowing to define the kinematic chain of an object, as well as the visual, collision and dynamic properties. In the ROS/Gazebo ecosystem, URDF files define the structure of a robot, whereas SDF files define models and otherwise simulation related entities to be loaded into Gazebo; a URDF model can be augmented to be consumed as a SDF model.

2.3 Objects and sensors

Graspable objects and obstacles of the ROVI workcell are modeled in Blender and then defined as SDF models in the `rovi_models` package, which allows to load these into Gazebo, as shown in Fig. 3.

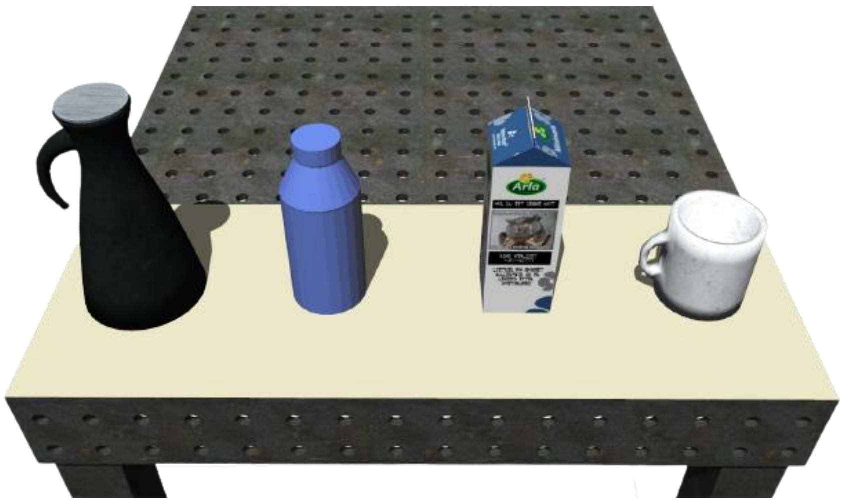


Fig. 3: Example objects of the ROVI system workcell.

The objects are modeled with inertial properties and friction parameters, facilitating a dynamic simulation in which a gripper is able to grasp objects without slip. The dynamic properties are either estimated in MeshLab [7], or approximated as simple rigid bodies (e.g., a cylinder). Frictional parameters are empirically selected provide a somewhat well-behaved simulation.

Input to the vision-based pose estimation is provided by several simulated devices, including a camera, a stereo-camera, a Kinect sensor, as well as a structured light projector. These devices are defined as SDF models, each with a corresponding Gazebo plugin, allowing to interact with the simulated devices using the ROS communication framework (via topics).

The `gazebo.h` module of the `ros_utils` package provides an interface to programmatically interact with the simulation environment, such as using `gazebo::camera().get_img()` to retrieve an image from the camera sensor, or using `gazebo::get_pose()` to read the pose of a link or model in the simulation.

2.4 Manipulator

The integration of the UR5 manipulator into the ROS simulation environment is contained within the `ur5_ros` meta-package. It contains, inter alia, the robot description with approximated dynamic parameters, interface to kinematics and dynamics of the robot, robot control, and a planning interface.

2.4.1 Robot description

The UR5 robot description and its joint limits are based on the Denavit-Hardenberg (DH) parameters [8] shown in Table 2. The model is defined as a dynamic URDF model using the XML macro language "Xacro" [9], including augments that allow to load the model into Gazebo and dynamically attach a Xacro-based end-effector (EE). The model components are defined in the `ur5_description` package, whereas the `ur5_gazebo` package constructs the model, launches Gazebo, and spawns the UR5 manipulator.

The dynamic parameters (mass, inertia etc.) that are required for a dynamic simulation and robot control, are not publically available; these are therefore approximated as given by [10] with some alternations. The dynamic parameters could potentially be estimated more accurately [11].

Joint	$\mathbf{q}$	d [mm]	α [rad]	a [mm]
1	$\mathbf{q}_1$	89.16	$\frac{\pi}{2}$	0.0
2	$\mathbf{q}_2$	0.0	0.0	−425.0
3	$\mathbf{q}_3$	0.0	0.0	−392.25
4	$\mathbf{q}_4$	109.15	$\frac{\pi}{2}$	$\mathbf{q}_5$
5	$\mathbf{q}_5$	94.65	$-\frac{\pi}{2}$	0.0
6	$\mathbf{q}_6$	82.3	0.0	0.0

Table 2: DH parameters of UR5 manipulator [8].

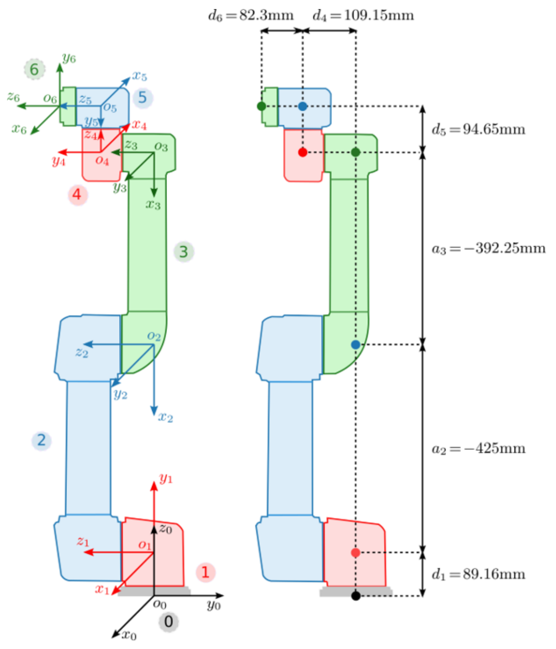


Fig. 4: Illustration of UR5 DH parameters [12].

2.4.2 Kinematics and dynamics

An interface to kinematics and dynamics of the UR5 manipulator is defined in the `ur5_dynamics` package, which contains several methods that map between the robot's joint configuration

$$\mathbf{q} = [q_1 \quad q_2 \quad \cdots \quad q_n]^\top , \tag{1}$$

and pose, which is given as a homogeneous transformation

$${}^{A}\mathbf{T}_{B} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{t}_{B} \\ \mathbf{0}^{1\times3} & 1 \end{bmatrix}, \tag{2}$$

where ${}^{A}\mathbf{R}_{B}$ and ${}^{A}\mathbf{t}_{B}$ is the rotation and translation, respectively, from frame $\{A\}$ to frame $\{B\}$.

Forward kinematics

Forward kinematics is a function $f(\mathbf{q})$, mapping a joint configuration vector $\mathbf{q} \in \mathbb{R}^n$ of n joints, to a pose of the end-effector ${}^{base}\mathbf{T}_{EE} \in \mathbb{R}^{4\times4}$, given in the base frame of the robot.

Given the transformations between n number of links, the full transformation is given as a serial kinematic chain

$${}^{base}\mathbf{T}_{EE} = {}^{base}\mathbf{T}_1 \cdot {}^{1}\mathbf{T}_2 \cdot \ldots \cdot {}^{n-1}\mathbf{T}_n \cdot {}^{n}\mathbf{T}_{EE}, \tag{3}$$

which is implemented as the function `ur5::fwd_kin(q)`.

Inverse kinematics

Determining the joint configurations $\mathbf{q}$ for a desired pose ${}^{base}\mathbf{T}_{EE}$ refers to the inverse kinematics problem, defined as the inverse of the forward kinematics function $\mathbf{q} = f^{-1}(\mathbf{T})$, which has a finite set of solutions (assuming no singularities) for a non-redundant manipulator, being the case for a UR5 manipulator.

The inverse kinematics may be approached in two ways: (a) analytically, by geometry or polynomials [13], or (b) numerically, by gradient descent methods. The analytical approach is preferred, since it is deterministic and global.

The `ur5::inv_kin(pose)` method implements an analytical solution from [14], which returns a set of possible solutions. Furthermore, an overload is implemented, which, given an initial joint configuration $\mathbf{q}_0$, returns the most optimal solution $\mathbf{q}^*$ for a desired pose, governed by the nearest neighbor algorithm with L2-norm as the metric

$$\mathbf{q}^* = \min_i \left\|\mathbf{q}_i - \mathbf{q}_0\right\|^2 \quad \text{for} \quad i \in [1\,..\,8] \tag{4}$$

where $\mathbf{q}_i$ is the i'th solution.

Dynamics

Control of the manipulator is realized by decentralized control methods [15], which provide robust and accurate control of a manipulator, given the model and dynamic parameters of the manipulator are well-defined. The differential equations of motion that describe the dynamics of a serial manipulator consisting of n rigid bodies [16], excluding the attached gripper, are governed by

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q}) \cdot \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \cdot \dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}), \tag{5}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the symmetric inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^{n\times n}$ represents the centrifugal and Coriolis effects, $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the configuration dependent gravity vector, and $\boldsymbol{\tau} \in \mathbb{R}^n$ is the torque commanded by the motors; friction and environmental torques are left unmodeled for simplicity.

The variables of the joint-space robot dynamics $[\mathbf{M}(\mathbf{q}),\ \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}),\ \mathbf{g}(\mathbf{q})]$ are computed using Orocos Kinematics and Dynamics Library (KDL) [17] given the kinematic chain of the manipulator for a given robot state $[\mathbf{q}, \dot{\mathbf{q}}]$. An interface is implemented in `ur5_dynamics`, providing methods such as `ur5::mass(q)`.

2.4.3 Robot control

The UR5 manipulator is actuated using ROS Control [18], which is a framework for implementing hardware agnostic controllers in the ROS ecosystem. This allows to define a custom controller by inheriting the classes provided by the framework [19, 20] using the standard hardware interfaces. The `gazebo_ros_control` plugin embedded in the robot description provides a simulated hardware interface [21], which allows to control robots simulated in Gazebo using any ROS control controller that is based one of the standard hardware interfaces.

The `EffortJointInterface` is used to interface the simulated UR5 manipulator, allowing to command desired joint torques $\boldsymbol{\tau}_d$. The robot state is read using the `JointStateInterface`, which retrieves the current state of the manipulator in Gazebo. Controllers and interfaces are contained in the `ur5_controllers` package.

Given the model in (5) is applicable, it is possible to realize a decentralized **Joint Position Controller** as

$$\boldsymbol{\tau}_d = \mathbf{M}(\mathbf{q}) \cdot \mathbf{y} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \cdot \dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}), \tag{6}$$

$$\mathbf{y} = \ddot{\mathbf{q}}_d + \mathbf{K}_d \cdot \dot{\tilde{\mathbf{q}}} + \mathbf{K}_p \cdot \tilde{\mathbf{q}}, \tag{7}$$

where (6) is the linearization feedback, which cancels the dynamics of the manipulator, and (7) is the commanded acceleration, which is governed by the joint error reference trajectory

$$\ddot{\tilde{\mathbf{q}}}_d + \mathbf{K}_d \cdot \dot{\tilde{\mathbf{q}}} + \mathbf{K}_p \cdot \tilde{\mathbf{q}} = 0, \tag{8}$$

where $\sim$ denotes the error between current and desired values, e.g., $\tilde{\mathbf{q}} = \mathbf{q} - \mathbf{q}_d$. The controller is interfaced by sending `trajectory_msgs::JointTrajectoryPoint` messages [22] to the controller command topic; these messages contain the desired joint values $\ddot{\mathbf{q}}_d$, $\dot{\mathbf{q}}_d$ and $\mathbf{q}_d$. A programmatic interface to the controller is available using the methods `ur5::command_setpoint(...)` and `ur5::command_traj(...)`.

2.5 Gripper (end-effector)

A WSG-50 servo-electric two-finger parallel gripper from Schunk [23] is mounted onto the flange of the UR5 manipulator. The gripper, as shown in Fig. 5, is modeled individually in the `wsg50` package, which contains the gripper description (URDF/SDF), controller and interface.

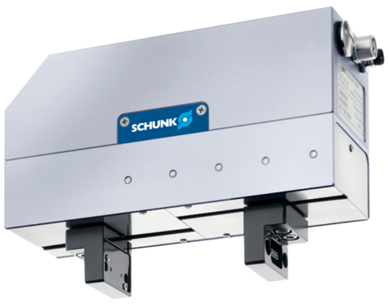


Fig. 5: Schunk WSG-50 parallel gripper.

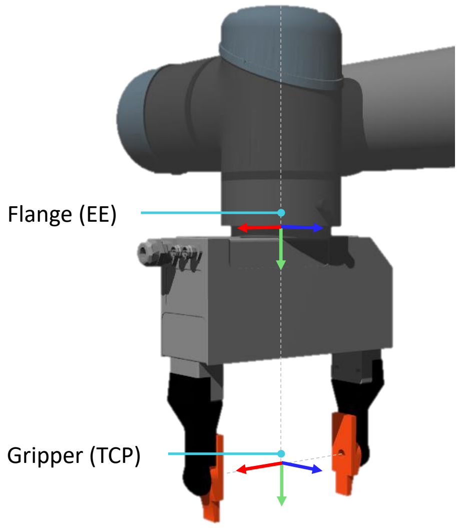


Fig. 6: A gripper attached to the flange of UR5 with TCP at a constant offset.

The fingers of the gripper are modeled with sufficiently high friction parameters to enable grasping in simulation. Since it is a parallel gripper, the joints of the fingers are modeled using the `<mimic>` property, such that the left finger inversely mimics the control signal of the right finger.

Due to the gripper being simulated as a parallel gripper, the dynamics are not modelled, as it complicates the dynamics of the system as a whole, and is out of the scope of the project. Instead, the gravity of the gripper is disabled in the simulation, such that gravity compensation can be avoided in the manipulator controller.

The gripper controller is a simple **Joint Effort Controller** that commands torques to either grasp or release the fingers of the gripper. To provide a more continuous motion, the rotatum (rate of torque) is saturated at $50\,\mathrm{N \cdot m/s}$. The `wsg50` package allows to dynamically attach to gripper to the UR5 manipulator – when done so, it loads the controller and specifies the `ee_T_tcp` parameter, which defines the ${}^{EE}\mathbf{T}_{TCP}$ transformation.

Tool center point

The forward and inverse kinematics are computed with respect to the end-effector (EE) frame (also known as the flange) of the manipulator. However, it is more than often desired to compute the inverse kinematics with respect to the tool center point (TCP), since this allows to align a gripper with an object to be grasped.

In many cases, the transformation ${}^{EE}\mathbf{T}_{TCP}$ from the EE frame to the TCP frame is a constant translation offset along the y-axis, as shown in Fig. 6.

Therefore, the inverse kinematics (IK) for a pose ${}^{base}\mathbf{T}_{TCP}$ are given by computing the IK for the EE pose with an offset, as

$$\mathbf{q} = f^{-1}\left({}^{base}\mathbf{T}_{TCP} \cdot {}^{TCP}\mathbf{T}_{EE}\right) \quad \text{where} \quad {}^{TCP}\mathbf{T}_{EE} = \left({}^{EE}\mathbf{T}_{TCP}\right)^{-1} \tag{9}$$

3 Planning

The pick-and-place task consists of: planning a trajectory to align the TCP of the end-effector with the graspable object in the pick area, grasp the object, and relocate the object onto the place area. Given a known object pose, this task has several considerations: base pose of the manipulator, grasping pose, and trajectory generation.

3.1 Reachability

The capability to optimally execute a pick-and-place task is determined by: the base pose of the manipulator, and the grasp pose (from the top or from the side) of the gripper. This capability can be quantified as the reachability of the manipulator, which, in the simplest case, is defined as the number of collision-free kinematic configurations in which the TCP can reach both the pick and place area for a given base pose.

A reachability analysis is implemented in the `reachability.h` module of the `ur5_planning` package as the `ur5::moveit::reachability(...)` method. It uses the MoveIt framework [3] to setup a planning scene, define the grasping pose, and perform a reachability analysis using collision checking.

The reachability analysis consists of relocating the robot base to a desired location and rotating the EE about the z-axis of an object at a resolution $\Delta\theta$, where for each iteration i, the desired EE pose is computed as

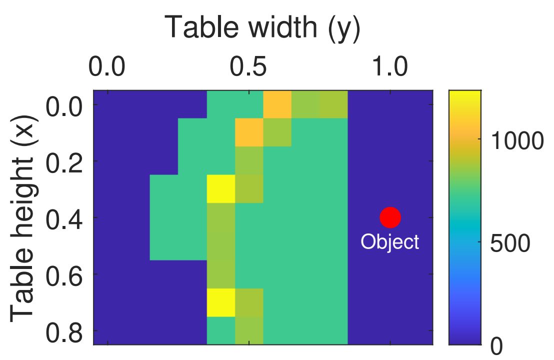
$$ {}^{base}\mathbf{T}_{EE} = {}^{base}\mathbf{T}_{world} \cdot {}^{world}\mathbf{T}_{obj} \cdot \mathbf{R}_z(\theta) \cdot \mathbf{T}_{grasp}, \tag{10} $$

where $\mathbf{T}_{grasp}$ is the grasp pose and $\theta = i \cdot \Delta\theta$. Then, given the pose ${}^{base}\mathbf{T}_{EE}$, the inverse kinematics are used to compute a set of possible joint configurations, where each joint configuration is tested for collisions. At last, the reachability analysis returns the number of iterations, plausible states, and collisions.

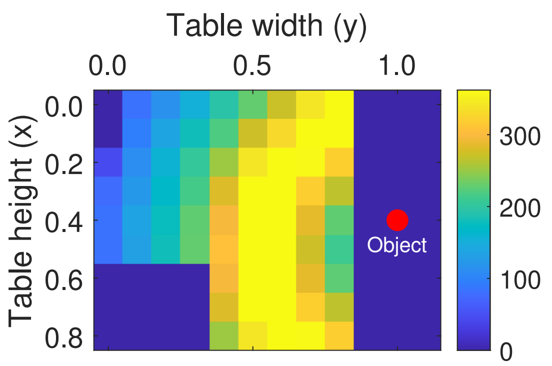
A study is conducted by discretizing the table into an grid spaced at 0.1 m, with a desired pick pose for the TCP at ${}^{world}\mathbf{p}_{obj} = [\, 0.5 \;\, 1.0 \;\, 0.75 \,]$ using the bottle object. For each grasping pose (top and side)

$$ \mathbf{T}_{top} = \left[\mathbf{R}_x(-\pi/2) + \mathbf{t}_z(0.2)\right] \cdot {}^{TCP}\mathbf{T}_{EE} \quad \text{and} \quad \mathbf{T}_{side} = \mathbf{t}_z(0.1) \cdot {}^{TCP}\mathbf{T}_{EE}, $$

the base of the manipulator is relocated in the xy-plane and the reachability analysis is performed at a resolution of $\frac{2\pi}{180} = 2°$. The results are visualized using heatmaps, as shown in Fig. 7.



(a) Grasp from the top



(b) Grasp from the side

Fig. 7: Reachability heatmaps for different manipulator base positions for the two grasping orientations.

The most optimal base pose when grasping from the side has 362 collision-free joint configurations, whereas grasping from the side has 1236 collision-free joint configurations – both located near the center of the table.

Despite the top grasping orientation having a higher reachability score, grasping from the side provides better grip in the simulation due to a greater number of contact points, thus providing more stable behavior. The base pose is chosen to be the center of the table, providing the optimal reachability when grasping from the side.

3.2 Interpolation-based trajectory generation

A trajectory $\mathcal{T}(t)$ is a path with an applied time scaling. Interpolation-based trajectory generation uses a set of waypoints to generate a trajectory subject to kinematic constraints (velocity/acceleration/joint limits) by interpolating between the waypoints, but without considering obstacle avoidance.

Given the desired pick pose of an object, in this case one of three locations, the objective is to define a trajectory between the initial EE pose and the pick pose via. four intermediate waypoints. The final path contains a total of seven waypoints; the first five are shown in Fig. 8, with the last two being a pre-pick offset and the actual pick pose.

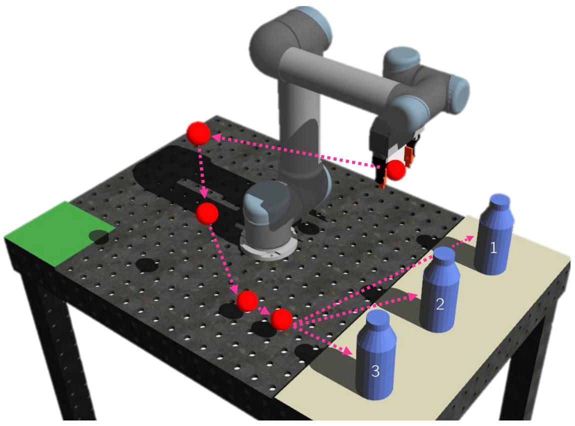


Fig. 8: Object locations and waypoints (🟥) for the interpolation-based trajectory generation.

The `rovi_planner` package implements trajectory generation in Cartesian space, which requires inverse kinematics to be executed by a joint-position controller. It uses Orocos Kinematics and Dynamics Library (KDL) [17] to implement both a point-to-point (linear) trajectory, and a linear segment with parabolic blends (LSPB) trajectory.

In order to define a trajectory $\mathcal{T}(t)$ as a function of time $t \in [0; T]$ of duration T, it is necessary to interpolate between the initial pose $\mathbf{T}_0$ and final pose $\mathbf{T}_f$, decoupled into translational interpolation and rotational interpolation.

Interpolation can be generalized by a scalar $s \in [0; 1]$, which is a function of time that maps the duration of the trajectory T to a normalized range, as

$$s(t) : [0; T] \to [0; 1],$$

which is known as time scaling (or timing law / velocity profile), since the function $s(t)$ defines how the trajectory is followed with respect to time. A trajectory is then defined as

$$\mathcal{T}(s(t)) : [0; 1] \to [\mathbf{T}_0; \mathbf{T}_f] \quad \text{where} \quad t \in [0; T],$$

returning an intermediate pose $\mathbf{T}_t$ for each time step t as a function of the time scaling function $s(t)$.

3.2.1 Linear interpolation (P2P)

Given a set of desired waypoints, a trajectory can be composed from separately interpolated point-to-point segments, where the EE comes to a full stop at the intermediate points.

The translational interpolation is given by a linear interpolation

$$\mathbf{x}(s) = \mathbf{x}_0 + s \cdot (\mathbf{x}_f - \mathbf{x}_0), \tag{11}$$

where $\mathbf{x}_0, \mathbf{x}_f \in \mathbb{R}^3$ are the initial and final positions respectively, and $s(t) \in [0; 1]$ is the time scaling.

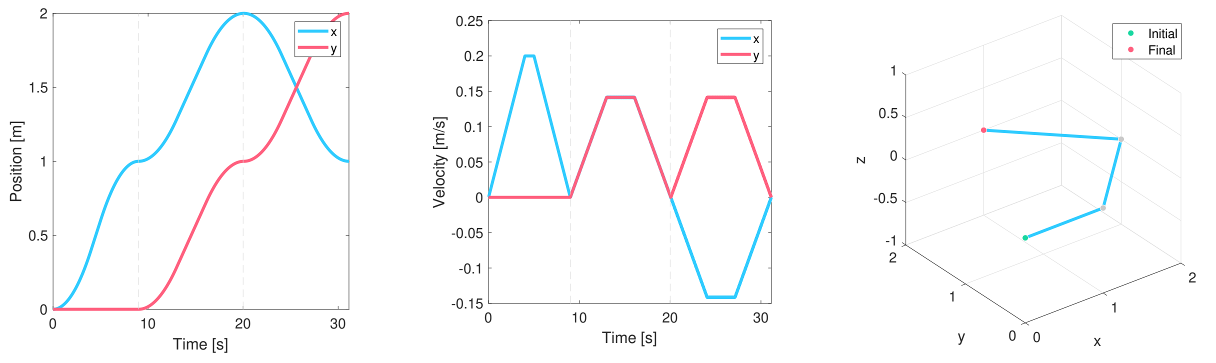
Rotational interpolation is given by a single-axis interpolation [24], which rotates a frame over the existing single rotation axis formed by a given start and desired end rotation. If more than a single rotational axis exist, an arbitrary is chosen; therefore it is not recommended to interpolate 180°.

A smooth interpolation must have no discontinuities in position, velocity, acceleration or sometimes even jerk. If the time scaling $s(t)$ is a smooth function, then $\mathbf{x}(s(t))$ will also be smooth, since it is linearly proportional to $s(t)$.

Using trapezoidal time-scaling [25] will guarantee no discontinuities in position and velocity, but cannot avoid jumps in acceleration. The discontinuity in acceleration may cause problems for some robots, but is most often smoothed out by the bandwidth of motion control systems.

The `rovi_planner::traj_lin()` method utilizes the `KDL::Path_Line` class to compose a trajectory of linearly interpolated paths from a list of waypoints. Each path is governed by a trapezoidal velocity time scaling, constrained by a maximum velocity and maximum acceleration. The trajectory ensures equitemporal motion by scaling all components down to the slowest axis (both in translation and rotation).

Figure 9 shows an example of a linearly interpolated trajectory with a maximum velocity of $0.2\,{}^{\mathrm{m}}/_{\mathrm{s}}$ and a maximum acceleration of $0.05\,{}^{\mathrm{m}}/_{\mathrm{s}^2}$.



(a) Position **(b)** Velocity **(c)** Path

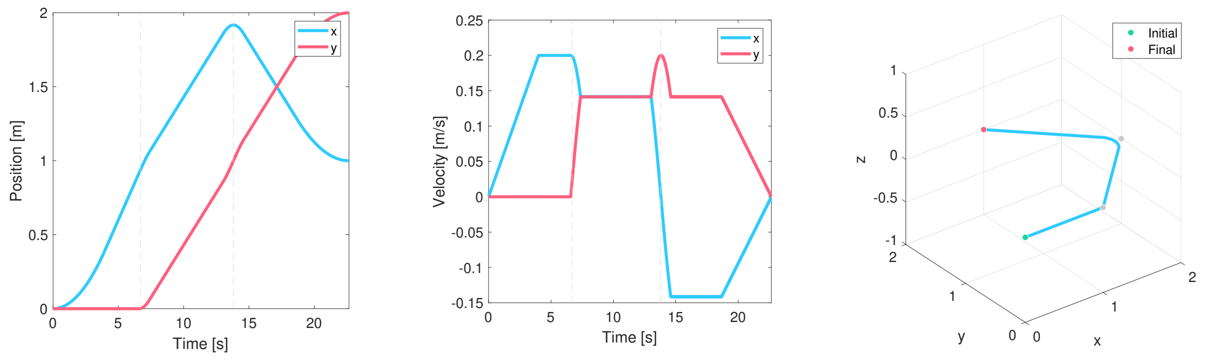
Fig. 9: Example of Cartesian trajectory generation using linear interpolation defined from four waypoints.

3.2.2 Parabolic interpolation (LSPB)

A continuous motion between waypoints can be achieved by introducing blends (rounded corners) at the intermediate waypoints, ensuring a smooth path in which the EE does not halt at intermediate points. The Linear Segment with Parabolic Blends (LSPB) method [25] achieves this by defining blends in the spatial domain, not to be confused with linear interpolation with a trapezoidal profile, which introduces blends in the temporal domain.

The `rovi_planner::traj_par()` utilizes the `KDL::Path_RoundedComposite` class to compose a parabolic trajectory from a list of waypoints with a specified corner radius (blend). It should be noted that the corner radius affects the accuracy at which the waypoints are reached.

Figure 9 shows an example of a parabolically interpolated trajectory with the same maximum velocity and maximum acceleration as in the linear interpolation of Fig. 9, but with a corner radius of 0.2 m.



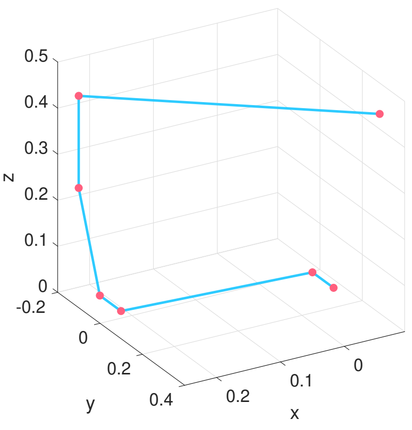
(a) Position **(b)** Velocity **(c)** Path

Fig. 10: Example of Cartesian trajectory generation using parabolic interpolation defined from four waypoints.

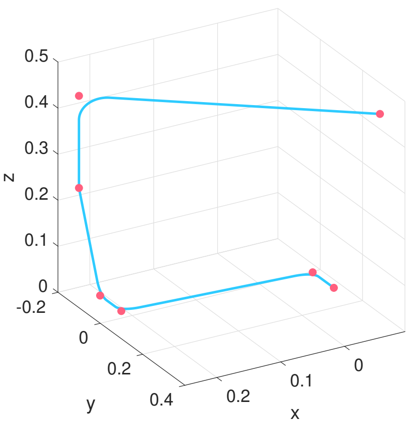
3.2.3 Evaluation

Given the desired waypoints and object locations shown in Fig. 8, the `planning_interpolation` test uses the interpolation-based methods to generate trajectories with a maximum velocity of 0.2 m/s, maximum acceleration of 0.05 m/s², and a corner radius of 0.2 m for the parabolic interpolation method.

Each waypoint is a pose ${}^{base}\mathbf{T}_{EE}$ in the base reference frame of the UR5 manipulator, since the resulting Cartesian trajectory is executed using inverse kinematics (which takes a pose ${}^{base}\mathbf{T}_{EE}$ as input). Examples of generated paths for the second (middle) object pose at ${}^{world}\mathbf{p}_{obj} = [\,0.40\ 1.05\ 0.75\,]$ are shown in Fig. 11.



(a) Linear interpolation



(b) Parabolic interpolation

Fig. 11: Paths (in base frame) generated using the interpolation-based methods for pose 2.

Each method is evaluated by generating 50 trajectories for each pick location and logging the planning time and trajectory duration of each iteration. The trajectory durations are shown in Table 3, and the average planning times are shown in Fig. 12.

Method \ Pose	1	2	3
Linear	18.97 s	20.35 s	22.85 s
Parabolic	13.66 s	14.93 s	17.33 s

Table 3: Trajectory durations of the interpolation-based trajectory generation methods.

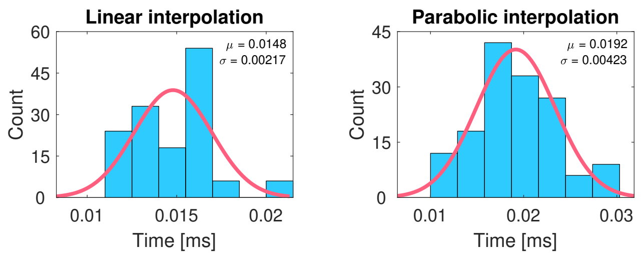


Fig. 12: Average planning time for each of the interpolation-based trajectory generation methods.

The trajectory durations are constant within their group, e.g., the linearly interpolated trajectory of the first pose is always 18.97 sec. The planning times are averaged across the pick locations for each method; they are normally distributed, with a mean planning time of $0.014\,80 \pm 0.002\,17$ ms for the linear interpolation method, and a mean planning time of $0.019\,20 \pm 0.004\,23$ ms for the parabolic interpolation method.

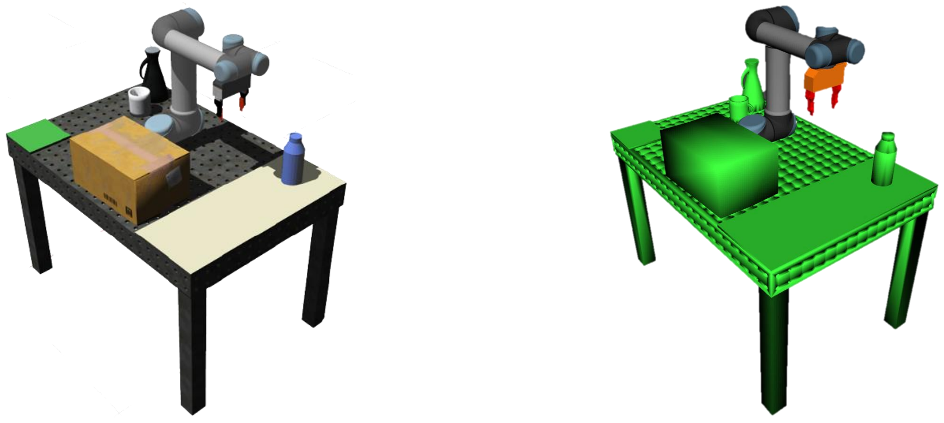
In general, the parabolic interpolation method provides a shorter trajectory duration, due to the blends and non-halting motion, however, at the cost of a slightly longer planning time. The parabolic interpolation method is only applicable if the accuracy at which the waypoints are visited is not of priority.

3.3 Sampling-based motion planning

3.3.1 Planning interface

Sampling-based motion planning constructs a probabilistic roadmap by probing the configuration space using a local planner [25]. The `moveit.h` module of the `ur5_planner` package implements an interface to the MoveIt motion planning framework, allowing to dynamically setup a *planning scene* and perform *motion plan requests* for a given *move group*, such as the robot arm with an attached end-effector (EE).

A planning scene is constructed from the current state of the workcell, including the state of the manipulator and its attached end-effector, as well as positions of collision objects in Gazebo, as shown in Fig. 13.



(a) Workcell in simulator. **(b)** Planning scene representation of collision objects.

Fig. 13: ROVI system workcell with obstacles for collision-free motion planning.

The planning interface can form planning requests using planners available in the Open Motion Planning Library (OMPL) [26, 27], such as RRT, PRM, SBL, etc. A request is formed for a given desired pose ${}^{base}\mathbf{T}_l$ and link l of a move group, constrained by positional and orientational tolerances. The planner checks for collisions (including self-collisions) and allows to attach an object to the end-effector, accounting for the object during planning.

A successful MoveIt motion plan is converted into a joint trajectory using an Iterative Parabolic Time Parameterization algorithm [28]. This trajectory can then be executed using the Joint Position Controller.

Given a planning scene with obstacles and pose ${}^{world}\mathbf{T}_{obj}$ of a graspable object, the pick-and-place task is realized by using the planning interface to compute collision-free joint trajectories. Planning is performed with respect to the TCP of the manipulator with specified tolerances; these give orientational freedom about the z-axis, which is important, as it is not always possible to align the TCP frame with the frame of an object due to obstacles.

For a pose ${}^{world}\mathbf{T}_{obj}$ of an object in the world frame, the `ur5::get_tcp_given_pose(pose, tf)` method uses a specified grasp pose $\mathbf{T}_{grasp}$ to compute the desired TCP pose ${}^{base}\mathbf{T}_{TCP}$ in the robot base frame as

$$ {}^{base}\mathbf{T}_{TCP} = {}^{base}\mathbf{T}_{world} \cdot {}^{world}\mathbf{T}_{obj} \cdot \mathbf{T}_{grasp}, \tag{12}$$

where ${}^{base}\mathbf{T}_{world}$ is extracted from the pose of the manipulator in the workcell simulation. For the place task, the object is attached to the end-effector after being grasped.

3.3.2 Evaluation

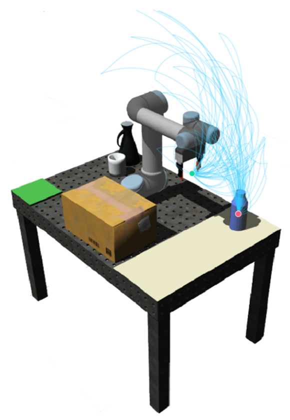
The `planning_moveit` test compares the Probabilistic Roadmap (PRM) [29] and Single-query, Bi-directional, Lazy Roadmap (SBL) [30] planners, available in OMPL within the MoveIt framework.

The workcell is populated with obstacles as shown in Fig. 13(a), and the bottle object is spawned in one of the three pick locations. For each pick location, the planner attempts to generate a trajectory, logging the success, and, given a successful plan, the planning time, and the resulting trajectory with its duration.

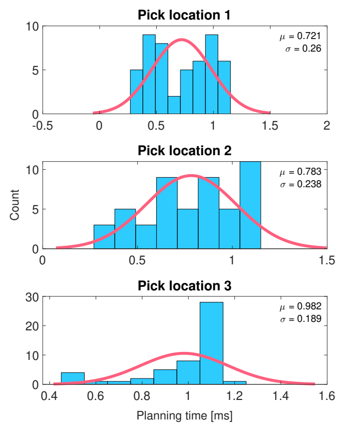
Each planner is evaluated 50 iterations for each pick location, with a maximum planning time of 1 s and the maximum number of planning iterations limited to 10. The positional tolerances are set to 0.001 m, and the orientation tolerances are set to $\begin{pmatrix} 0.001 & 0.001 & 3.14 \end{pmatrix}$ rad for roll, path and yaw, respectively.

Probabilistic Roadmap (PRM)

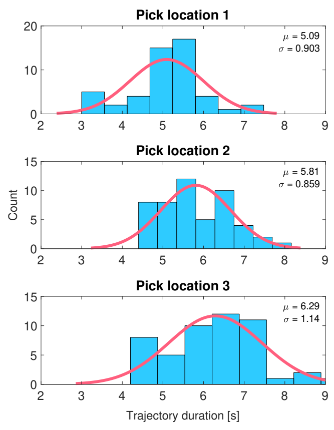
The PRM planner samples the configuration space of the robot, testing configurations for whether they are in the free space, using a local planner to attempt to connect these to other nearby configurations.



(a) Trajectory



(b) Planning time



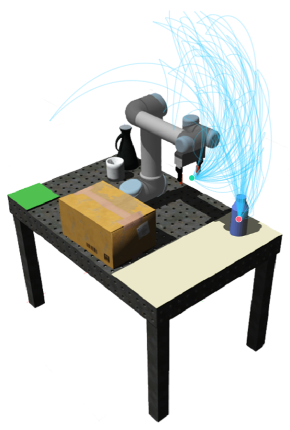
(c) Trajectory duration

Fig. 14: Results of the pick task study for the Probabilistic Roadmap planner.

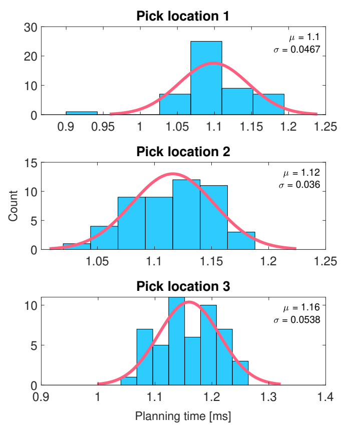
There were no failed planning attempts for the PRM planner; the planning times are generally below 1 ms for all pick locations, but do have a somewhat high variance. The trajectory durations increase as the object moves further from the end-effector's starting pose.

Single-query, Bi-directional, Lazy Roadmap (SBL)

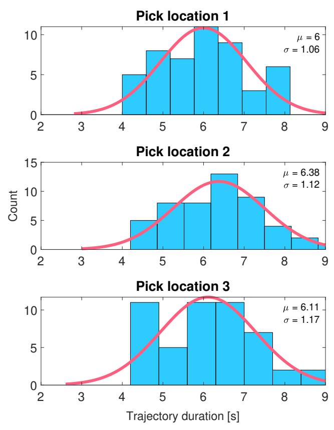
The SBL planner is essentially a bidirectional version of Expansive Space Trees planner with lazy state validity checking. The results of the study are shown in Fig. 15.



(a) Trajectory



(b) Planning time



(c) Trajectory duration

Fig. 15: Results of the pick task study for the Single-query, Bi-directional, Lazy Roadmap planner.

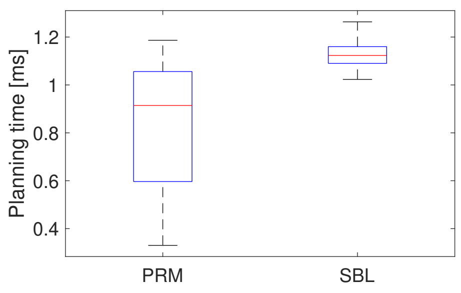
The SBL planner has similar planning times for each pick location with low variance. There were no failed planning attempts for any of the pick locations. The trajectory durations are also similar.

Comparison

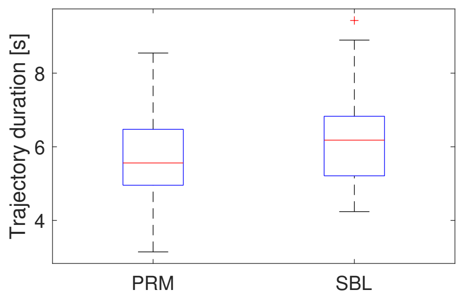
A visual model check (histogram, Q-Q plot) of the results indicate that most of the results are normally distributed; for each method, the data is accumulated across the pick locations for comparison.

The mean planning time of the PRM planner is at 0.829 ± 0.255 ms, which has a relatively high dispersion. In comparison, the SBL planner has a much more predictable (less dispersion), yet higher, mean planning time at 1.1200 ± 0.0577 ms, as shown in Fig. 16(a).

Both methods show visually similar trajectories, as evidenced by Fig. 14(a) and Fig. 15(a), with durations approximately in the range of 5 s to 6 s. The PRM planner has an average trajectory duration of 5.73 ± 1.09 s, whereas the SBL planner has an average trajectory duration of 6.24 ± 1.33 s, as shown in Fig. 16(b).



(a) Planning time



(b) Trajectory duration

Fig. 16: Box plots of accumulated (over pick locations) planning time and trajectory duration.

Despite the larger dispersion, the PRM planner is able to provide shorter planning times, which on average are below 1 ms. It also provides less varying and shorter trajectory durations. Furthermore, the PRM planner can be setup to re-use the probabilistic roadmap it generates, meaning that the planning time will decrease over time, as long as the obstacles stay static within the workcell – it is chosen as the preferred sampling-based planning method.

3.4 Method selection

A comparison of the interpolation-based trajectory generation and the sampling-based motion planning is inherently erroneous. Interpolation-based trajectory generation may provide deterministic and optimal solutions in static environments, but does not consider obstacle avoidance or self-collisions. On the other hand, the sampling-based motion planning approach is well suited for dynamic environments with obstacles, but is only probabilistic complete and will often return far from optimal solutions.

Both methods can be used to relocate a graspable object from the pick area to the place area. However, the interpolation-based method requires beforehand knowledge of obstacles, where waypoints must be defined manually; and even then, self-collisions and singularities can occur during trajectory execution. Using it for an arbitrary pose in the pick area would require a robust implementation, since at the very least, collisions with the table, object and self-collisions must be avoided. An algorithm could be devised for such endeavor; it would provide deterministic planning times and trajectory durations, but will require modifications if the workcell changes.

The sampling-based method provides a more flexible approach at the cost of a varying planning time and trajectory duration; in the worst case, the trajectory generation might fail and require re-planning. The method takes into consideration both self-collisions and collisions with obstacles; given a workcell with dynamic obstacles and objects, the sampling-based motion planning seems as the obvious choice, for which the PRM planner has best performance in terms of both planning time and trajectory duration.

4 Pose estimation

The pose of graspable objects in the pick area must be known in order to perform pick-and-place tasks. Since the objects are not static within the workcell, 6D pose estimation is utilized in order to estimate the pose (position and orientation) of an object. Pose estimation is categorized as either depth-based (geometric), image-based, or a hybrid of these. The ROVI system implements and evaluates two approaches: RANSAC Registration with ICP (depth-based), and RGB Template Matching (image-based).

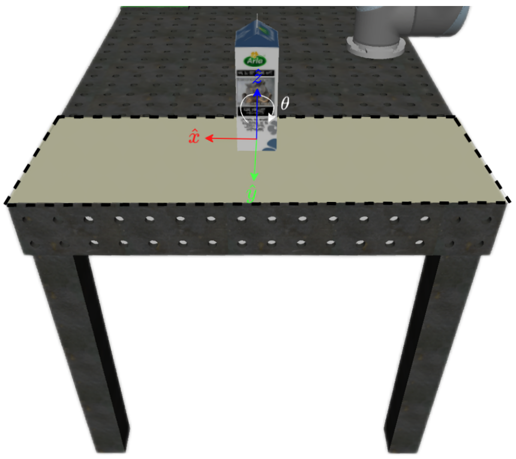


Fig. 17: The Milk object placed within the pick area (dotted lines) of the ROVI workcell.

Both methods are based on a set of **scene assumptions**. The pose estimation is done with respect to the Milk object, as shown in Fig. 17. The object is assumed to be standing upright (flat with the table), where only one, non-occluded instance of the object is present in the outlined pick area of the workcell at all times. Thus, the pose is fully characterized by three parameters: (x, y, θ), where x and y refer to the position in the table plane, and θ is the relative orientation about the z-axis with respect to the world frame.

4.1 RANSAC Registration with ICP

Point cloud registration is the process of determining the rotation and translation that aligns two point clouds. The `RANSACRegistrationWithICP.h` module of the `rovi_vision` package implements depth-based pose estimation using the Kabsch algorithm with RANSAC and Iterative closest point (ICP), allowing to estimate the pose of a Milk object in the workcell, given a captured 2.5D point cloud of the scene and a 3D model of the object as inputs.

4.1.1 Methodology

A 2.5D point cloud of the scene is retrieved from the Kinect sensor within the simulated environment; a sequence of pre-processing steps are applied before pose estimation:

1. Points that lies outside a pre-defined workspace are treated as outliers and are removed.
2. Statistical outlier removal is utilized to remove irregularities.
3. Plane fitting is used to remove the table points.
4. Voxel downsampling to leaf size of one-centimeter ($1 \times 1 \times 1$ cm).

The point cloud of the object (`milk.pcd`) is loaded and downsampled to the same voxel grid size as the scene point cloud. Surface normals (with respect to the camera direction from the scene) and Spin image keypoint descriptors [31] are estimated for both point clouds. The keypoint descriptors are then bruteforce-matched using the Euclidean norm as a similarity metric, storing most similar correspondence pairs.

For point cloud registration, RANSAC is performed for 1 000 000 iterations, parallelized on n threads. In each iteration, three keypoint descriptor correspondences (pairs of 3D points from the two point clouds) are randomly sampled. The three samples must pass a pruning step, which seeks to reject inadmissible correspondences by comparing the triangle area ratio spanned by the three object points and scene points, respectively. A candidate transformation should ideally have a ratio of 1.0, but due to noise, the ratio is relaxed to be in the range of 0.7 to 1.3 If the three correspondences pass the pruning test, a transformation (from scene to object) is determined using Kabsch algorithm. The scene is then transformed, and the number of inliers is found by utilizing a nearest neighbor search with a fixed distance rejection criteria corresponding to the width of a voxel. The transformation with the largest number of inliers is considered to be the best candidate.

Iterative closest point (ICP) [32] is then utilized to refine the alignment between scene and object, by minimizing the Euclidean distance between the inliers. An inlier is defined similarly to RANSAC, but with a distance rejection criteria of three voxel widths. The algorithm is stopped after ten iterations, or if the Euclidean error between two consecutive steps exceeds a threshold (in order to avoid divergence). The estimated pose ${}^{world}\mathbf{T}_{obj}$ of the object in the world frame is given by multiplying the transformation from ICP and RANSAC, and inverting it.

4.1.2 Evaluation

Error propagation of the RANSAC Registration with ICP pose estimation method is evaluated by Monte Carlo simulations with gradually increasing zero-mean Gaussian noise on the captured scene point cloud. The standard deviation is incremented by 4 mm until significant dispersion occurs. Each increment of Monte Carlo simulation uses twenty random object poses with twenty iterations per pose.

Three metrics are used to evaluate the performance: (a) a binary metric that classifies whether the estimated pose of the object is graspable, (b) the dispersion of the translation- and orientation-error, and (c) the time complexity.

Graspable is defined solely by a kinematic analysis, where the Milk object should be within the allowable clearance of the WSG50 finger-tips, which is approximated to be within a 3 cm translational error. The orientation of the object can be represented by two orientations due to two-sided symmetry, which is accounted for in metric (b).

Simulations are stopped after four increments of added noise. Figure 18 shows the translational error (error in ground truth and estimated xy−position). Metric (b) requires an assumption of normality, for which a visual model check (Q-Q plots, scatter plots) is performed. In general, it is reasonable to assume that the xy-error can be approximated by marginal normal distributions when removing 0 % to 20 % outliers. The resulting orientation error ranges between a normal distribution and a uniform distribution when noise is increased.

Results of the three descriptive metrics are shown in Table 4, where σ_x, σ_y and σ_θ measure the dispersion without outlier removal. The mean run time μ_t is decreasing as noise increases due to pruning in RANSAC.

σ Metric	0 mm	4 mm	8 mm	12 mm
Binary	100 %	96.50 %	74.75 %	65.00 %
σ_x	2.34 mm	5.40 mm	13.5 mm	21.5 mm
σ_y	2.29 mm	13.8 mm	15.7 mm	15.5 mm
σ_θ	14.9°	34.6°	59.4°	67.4°
μ_t	5.03 s	6.4 s	4.90 s	3.43 s

Table 4: Results of Monte Carlo simulations after four increments.

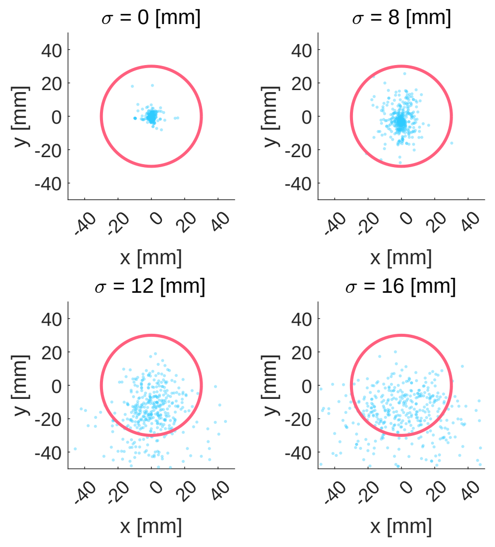


Fig. 18: Scatter plots of the translation error.

4.2 RGB Template Matching

Image-based pose estimation uses images of the scene in order estimate the pose of a known object. The `RGBTemplateMatching.h` module of the `rovi_vision` package implements RGB Template Matching. The approach uses a database of relevant object poses and their respective RGB images (templates), where LineMod [33] is used to determine the stored image that best matches the scene image, thus inferring the pose of the object.

4.2.1 Methodology

Template generation

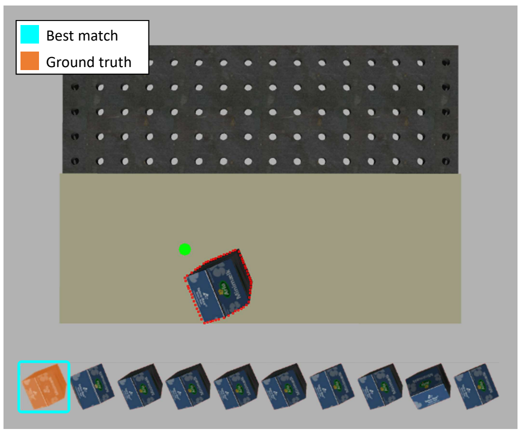
The simulated camera is placed in the workcell (scene) with a top-view of the pick area; the intrinsic parameters are used to render and generate image templates, which can be done in one of two ways: offline and online.

Offline template generation centers the bounding box of the object at the origin of an empty 3D world coordinate system. A set of virtual camera poses are generated on view spheres with the cameras pointing towards the centroid of the object. The view spheres are parameterized by their radius, longitude and latitude – however, only a certain range of those parameters have relevancy. One of the difficulties in generating offline templates is determining the relevant range of hyperparameters that avoids rendering templates outside of the region of interest.

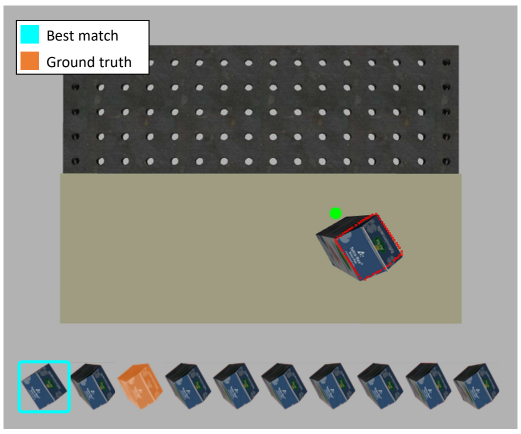
Online template generation utilizes the simulation environment, since the ground truth pose of the object within the pick area is known. This allows to generate templates using the simulated camera, by simply moving the object around in the pick area at some discretization. However, this limits the generalization of the templates, as a new set of templates must be generated if the camera is moved. This method is well suited for the pick-and-place task with static camera poses, which is the case for the ROVI system.

Pose estimation using LineMod

For pose estimation, a LineMod instance is configured with RGB modality and a coarse-to-fine grained pyramid structure of $\{8, 4, 2, 1\}$. After the templates are loaded, the pose estimation pipeline is ready; given an input RGB image, the instance returns a sorted list of best matches, where the matched template with the largest similarity score is mapped to a pose of the object using a lookup-table, as shown in Fig. 19.



(a) Correct match



(b) Wrong match

Fig. 19: Two example matches using LineMod, each with a list of the top ten matched templates (sorted left-to-right), with outlined best match (🟦) and highlighted ground truth (🟧).

It may not be appropriate to exclusively rely on the similarity metric of LineMod for pose estimation, since it can return false positives, as shown in Fig. 19(b). Post-processing could be employed to ensure robustness, but is ignored in this pipeline in favor of reduced complexity.

4.2.2 Evaluation

The RGB Template Matching pose estimation method is evaluated by Monte Carlo simulations with gradually increasing zero-mean Gaussian noise of the scene image. The standard deviation is increased by 30 (channel-wise RGB intensity) until significant dispersion occurs. Each increment of Monte Carlo simulation uses 100 random object poses with twenty iterations per pose.

The same three metrics from 4.1.2 Evaluation are used for evaluating the performance of the method: (a) binary classification, (b) translation and orientation error, and (c) time complexity. However, the orientation error is no longer constrained by symmetric properties of the Milk object, since the method is able to utilize textural information.

The simulations are stopped after four increments of added noise. Table 5 shows the resulting metrics and the translation error is shown in Fig. 20.

Metric \ σ	0	60	90	120
Binary	54.0 %	72.6 %	66.6 %	52.7 %
σ_x	28.5 mm	23.7 mm	26.5 mm	31.5 mm
σ_y	23.6 mm	19.8 mm	22.0 mm	20.7 mm
σ_θ	4.12°	8.25°	9.41°	6.4°
μ_t	0.099 s	0.106 s	0.120 s	0.110 s

Table 5: Results of Monte Carlo simulations after four increments.

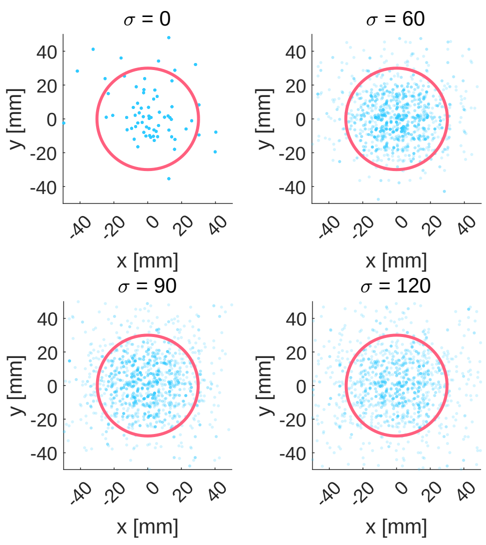


Fig. 20: Scatter plots of translational errors.

It is reasonable to assume that the translational error is Gaussian distributed for the three last experiments since isocontours are apparent. The orientation error is almost zero mean Gaussian distributed with minimum noise, meaning, the noise is caused by the discretization error from template generation.

The binary classification rate increases at the second simulation, which may imply that the candidate matches should be further processed before inferring on the pose. The mean run time μ_t is nominally constant throughout the experiments.

4.3 Method selection

Both pose estimation methods are applicable for pick-and-place tasks within the ROVI system. However, when comparing the binary classification scores, a key metric for pick-and-place tasks, the depth-based method at 65 % to 100 % accuracy significantly outperforms the image-based method at 52.7 % to 72.6 % accuracy. The image-based method does provide a significantly better run time, but is irrelevant if the pose estimation is erroneous.

It should be noted that RGB Template Matching, despite its low baseline accuracy, is impressively resilient to noise; it looks promising for industrial pick-and-place tasks if proper post-processing were to be employed.

5 Pick-and-place

5.1 Integration

The ROVI system integrates the vision-based pick-and-place pipeline, which is primarily comprised of three modules that interact with the simulation environment, namely: (a) vision-based pose estimation, (b) sampling-based motion planning, and (c) robot control – an overview of the system is shown in Fig. 21.

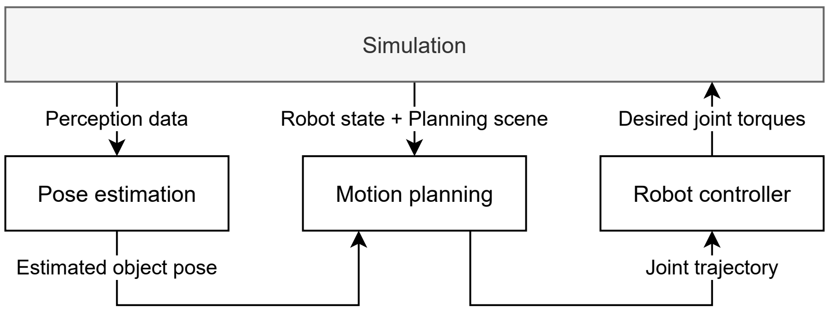


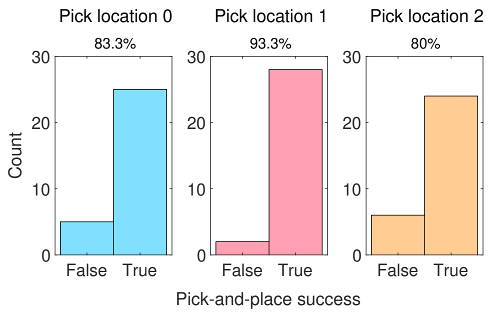
Fig. 21: Overview of integration of ROVI system.

The pipeline integrates RANSAC Registration with ICP pose estimation and PRM motion planning. The robot is mounted in table center and objects are grasped from the side. Given a workcell with obstacles and a graspable object, a pick-and-place task is realized by estimating the pose of the object, planning a trajectory to relocate it to the place area, and executing the motion on the UR5 manipulator using a Joint Position Controller – a demonstration is shown in [34], which is contained in the `demo_pick_and_place` executable of the `rovi_system` package.

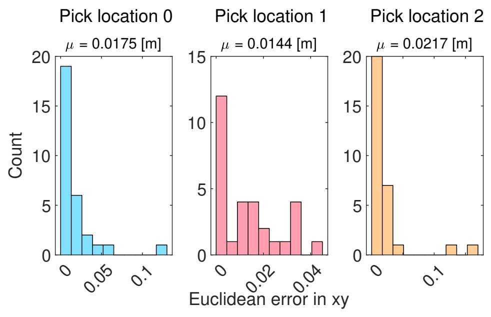
5.2 Evaluation

The pick-and-place pipeline is evaluated as a binomial experiment in the `pick_and_place` test. For three different pick locations in a workcell with obstacles, the estimated pose of a Milk object is used to plan and execute a trajectory in order to grasp and relocate the object to the place area. A binary success classification is based on the Euclidean xy-error in the desired and measured place pose being ≤ 0.05 m.

The RANSAC Registration with ICP pose estimation is configured to run for 1 000 000 iterations with a leaf size of 0.01. For the PRM planner, the maximum planning time is set to 1 s, and the maximum number of planning iterations is limited to 10. The positional tolerances are set to 0.001 m, and the orientation tolerances are set to $\begin{pmatrix} 0.001 & 0.001 & 3.14 \end{pmatrix}$ rad. The experiment is repeated 30 times for each pick location; the results are shown in Fig. 22.



(a) Histograms of binary classification of pick-and-place success.



(b) Euclidean xy-error of relocated object after pick-and-place task.

Fig. 22: Results of integrated pick-and-place test.

The success rate of the pick-and-place task is 80 % to 93.3 % with a mean Euclidean xy-error of 0.0179 ± 0.0262 m across all three pick locations. Some of the failed attempts are due to controller inaccuracies (gripper getting stuck on object), or if the object slips out during relocation. The xy-error is due to non-symmetrical gripper control.

5.3 Conclusion and discussion

By integrating RANSAC Registration with ICP depth-based pose estimation and PRM sampling-based motion planning within a ROS/Gazebo/MoveIt framework, the ROVI system is able to perform pick-and-place tasks on a Milk object with an accuracy of ≥ 80 % using a UR5 manipulator with an attached WSG50 gripper. We assume that the pick-and-place pipeline will show a similar accuracy for other graspable objects, as long as a 3D model is provided for pose estimation, and a proper grasp pose is specified for the object.

There are, however, improvements to be made for a more robust pick-and-place system; especially if the pick-and-place pipeline is to be implemented in a physical setting.

Minor inaccuracies in pose-estimation and/or controller execution occasionally lead to the gripper colliding its fingertips with the edge of an object, completely impairing the simulation. A pre-pick pose could alleviate this issue by grasping an object from the side, in which a "slide-in" motion is likely to nudge the object into place, instead of the gripper getting stuck.

The system can only handle identical objects (e.g., only Milk) in the pick area, since the planner interface must know what object to attach to the EE during planning. Object identification could be used to determine the object being grasped, which would allow to perform pick-and-place tasks with different objects being simultaneously present in the pick area. Furthermore, the pick operation can also be improved by defining planning tolerances and a grasp pose per unique object, storing these in a look-up table.

References

[1] Quigley, Morgan et al. “ROS: an open-source Robot Operating System”. In: *ICRA Workshop on Open Source Software* 3 (Jan. 2009).

[2] Koenig, N. and Howard, A. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: 3 (2004), 2149–2154 vol.3. DOI: `10.1109/IROS.2004.1389727`.

[3] Coleman, David et al. “Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study”. In: (Apr. 2014).

[4] Ivaldi, Serena et al. “Tools for simulating humanoid robot dynamics: A survey based on user feedback”. In: 2015 (Nov. 2014). DOI: `10.1109/HUMANOIDS.2014.7041462`.

[5] Straszheim, Troy et al. *Conceptual overview of ROS catkin*. URL: `http://wiki.ros.org/catkin/conceptual_overview`.

[6] Open Source Robotics Foundation. *Using a URDF in Gazebo*. URL: `http://gazebosim.org/tutorials/?tut=ros_urdf`.

[7] Open Source Robotics Foundation. *Inertial parameters of triangle meshes*. URL: `http://gazebosim.org/tutorials?tut=inertia`.

[8] Universal Robots. *DH-parameters for calculations of kinematics and dynamics of UR robots*. URL: `https://www.universal-robots.com/articles/ur/parameters-for-calculations-of-kinematics-and-dynamics/`.

[9] Glaser, Stuart, Woodall, William, and Haschke, Robert. *Xacro (XML Macros)*. URL: `https://wiki.ros.org/xacro`.

[10] ROS-Industrial. *URDF description of UR5 with approximated dynamic parameters*. URL: `https://github.com/ros-industrial/universal_robot/blob/kinetic-devel/ur_description/urdf/ur5.urdf.xacro`.

[11] Kovincic, Nemanja et al. “Dynamic parameter identification of the Universal Robots UR5”. In: (May 2019). DOI: `10.3217/978-3-85125-663-5-07`.

[12] Kufieta, Katharina. “Force estimation in robotic manipulators: Modeling, simulation and experiments”. In: *Department of Engineering Cybernetics NTNU Norwegian University of Science and Technology* (2014). URL: `http://folk.ntnu.no/tomgra/Diplomer/Kufieta.pdf`.

[13] Diankov, Rosen. “Automated Construction of Robotic Manipulation Programs”. PhD the-

sis. Carnegie Mellon University, Robotics Institute, Aug. 2010. URL: http://www.programmingvision.com/rosen_diankov_thesis.pdf.

[14] Messmer, Felix et al. *universal_robot ROS*. URL: http://wiki.ros.org/action/show/universal_robots.

[15] Siciliano, Bruno et al. *Robotics*. Springer London, 2009. DOI: 10.1007/978-1-84628-642-1. URL: https://doi.org/10.1007%2F978-1-84628-642-1.

[16] Spong, M.W., Hutchinson, S., and Vidyasagar, M. *Robot Modeling and Control*. Wiley, 2005. ISBN: 9780471649908. URL: https://books.google.dk/books?id%20=%20jyD3xQEACAAJ.

[17] Smits, R. *KDL: Kinematics and Dynamics Library*. URL: http://www.orocos.org/kdl.

[18] Chitta, Sachin et al. "ros_control: A generic and simple control framework for ROS". In: *The Journal of Open Source Software* (2017). DOI: 10.21105/joss.00456. URL: http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf.

[19] *ros_control wiki on GitHub*. URL: https://github.com/ros-controls/ros_control/wiki.

[20] Slate Robotics. *How to implement ros_control on a custom robot*. URL: https://medium.com/@slaterobotics/how-to-implement-ros-control-on-a-custom-robot-748b52751f2e.

[21] Open Source Robotics Foundation. *ros_control in Gazebo*. URL: http://gazebosim.org/tutorials/?tut=ros_control.

[22] Glaser, Stuart and Foote, Tully. *trajector_msgs*. URL: http://wiki.ros.org/trajectory_msgs.

[23] Schunk. *WSG-50 servo-electric 2-finger parallel gripper*. URL: https://schunk.com/fileadmin/pim/docs/IM0004935.PDF.

[24] The Orocos Project. *KDL: Single Axis Interpolation*. URL: http://docs.ros.org/en/melodic/api/orocos_kdl/html/classKDL_1_1RotationalInterpolation__SingleAxis.html#details.

[25] Lynch, Kevin M. and Park, Frank C. *Modern Robotics: Mechanics, Planning, and Control*. 1st. Cambridge University Press, 2017. ISBN: 1107156300.

[26] Şucan, Ioan A., Moll, Mark, and Kavraki, Lydia E. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). https://ompl.kavrakilab.org, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[27] *OMPL Available Planners*. URL: https://ompl.kavrakilab.org/planners.html.

[28] MoveIt. *MoveIt Iterative Parabolic Time Parameterization*. URL: http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/time_parameterization/time_parameterization_tutorial.html.

[29] Kavraki, L.E. et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: vol. 12. 4. 1996, pp. 566–580. DOI: 10.1109/70.508439.

[30] Sánchez, Gildardo and Latombe, Jean-Claude. "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking". In: *Robotics Research*. Springer Berlin Heidelberg, 2003, pp. 403–417. ISBN: 978-3-540-36460-3.

[31] Siebert, Jan. "SIFT Keypoint Descriptors for Range Image Analysis". In: *Annals of the BMVA* (Jan. 2009).

[32] PCL. *Iterative Closest Point*. 2020. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/interactive_icp.html.

[33] Hinterstoißer, Stefan et al. "Gradient Response Maps for Real-Time Detection of Textureless Objects". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (2012), pp. 876–888.

[34] Androvich, Martin and Schøn, Daniel Tofte. *Pick-and-place with UR5 using RANSAC/ICP pose estimation and PRM planning (ROS/Gazebo/-MoveIt)*. Youtube. 2021. URL: https://www.youtube.com/watch?v=1jAmozyywgk.