Simulated pick and place tasks of objects identified using computer vision in a ROS/Gazebo environment

A semester project in the course of
Robotics and Computer vision

written by

Martin Androvich
marta16@student.sdu.dk

Daniel Tofte Schøn
dscho15@student.sdu.dk

The code for this project is available at
`https://github.com/martinandrovich/rb-rovi`

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

January, 2021

Abstract

A dynamic simulated environment using ROS/Gazebo framework is used to facilitate development of a vision-based pick and place pipeline within a workcell consisting of a UR5 manipulator mounted onto a specialized table with designated pick and place areas, equipped with various perception sensors.

The dynamic simulation environment is established with interfaces that enable interaction with the environment and control of the manipulator and gripper. A reachability analysis is used to determine the optimal base mount location of the manipulator. Motion planning is used to facilitate object relocation via trajectory generation; both interpolation-based and collision-free methods are implemented and evaluated. Two vision-based pose-estimation methods are implemented and evaluated for localization of object within the workcell.

A fully integrated system uses RRT* for collision-free motion planning and a dense stereo algorithm for pose estimation of an object within the designated pick area. This enables generation of trajectories that can be executed using a joint-space effort controller, allowing the gripper to grasp the object and relocate it to the place area without colliding with objects in the workcell.

Evaluation of the collision-free RRT planners showed that trajectories can be generated as long as objects are grasped with a feasible grasping orientation. The dense stereo pose estimation method showed a 99 % success rate for images with $\sigma = 6$ Gaussian noise. A fully integrated system is evaluated by a binary test based on whether the RRT* planner is able to find a solution given an estimated pose; results vary from 55 % to 80 % based on the pick location.

Contents

Contributions

The contributions of the authors to this project are summarized by 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
2.2.1 Model representation	Daniel Tofte Schøn
2.2.2 Objects and senors	Martin Androvich
2.2.3 Robot model and dynamics: Robot description	Martin Androvich
2.2.3 Robot model and dynamics: Gripper description	Daniel Tofte Schøn
2.2.3 Robot model and dynamics: Forward kinematics	Martin Androvich
2.2.3 Robot model and dynamics: Inverse kinematics	Daniel Tofte Schøn
2.2.3 Robot model and dynamics: Dynamics	Daniel Tofte Schøn
2.2.4 Robot control: Intro	Martin Androvich
2.2.4 Robot control: Joint Position Controller	Daniel Tofte Schøn
2.2.4 Robot control: Cartesian Pose Controller	Daniel Tofte Schøn
2.2.4 Robot control: Gripper Controller	Martin Androvich
3.1 Reachability	Martin Androvich
3.2.1 Interpolation-based planning: Linear interpolation	Martin Androvich
3.2.1 Interpolation-based planning: Parabolic interpolation	Daniel Tofte Schøn
3.2.1 Interpolation-based planning: Evaluation	Daniel Tofte Schøn
3.2.2 Collision-free planning: Interface	Martin Androvich
3.2.2 Collision-free planning: Trajectory generation + Evaluation	Daniel Tofte Schøn
3.3 Method selection	Martin Androvich
4.1.1 Assumptions	Daniel Tofte Schøn
4.1.2 Pose estimation	Daniel Tofte Schøn
4.1.3 Monte Carlo simulations	Martin Androvich
4.1.4 Results	Daniel Tofte Schøn
4.1.5 Evaluation	Martin Androvich
4.2.1 Assumptions	Martin Androvich
4.2.2 Pose estimation	Martin Androvich
4.2.3 Monte Carlo	Daniel Tofte Schøn
4.2.4 Results	Martin Androvich
4.2.5 Evaluation	Martin Androvich
4.3 Evaluation	Daniel Tofte Schøn
5 System integration	Martin Androvich
5.1 Evaluation	Daniel Tofte Schøn
5.2 Conclusion and discussion	Daniel Tofte Schøn

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

Acronyms

API application programming interface.
CAD computer-aided design.
COM center of mass.
DH Denavit-Hardenberg.
KDL Orocos Kinematics and Dynamics Library.
OMPL Open Motion Planning Library.
ROI region of interest.
ROS Robot Operating System.
ROVI robotics and computer vision.
SDF Simulation Description Format.
SRDF Semantic Robot Description Format.
TCP tool center point.
URDF Unified Robot Description Format.

Terms

dense stereo Enables for 3D reconstruction of a stereo rectified camera setup, where the matching problem is one-dimensional.

iterative closest point A local alignment algorithm, which minimizes the difference between two point clouds.

middleware Software that connects software components or applications, typically being the layer between the operating system and user applications.

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, $\mathbf{M}(\boldsymbol{q})$, Coriolis matrix, $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, and gravity vector, $\boldsymbol{g}(\boldsymbol{q})$.

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

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

1 Introduction

1.1 Project description

In industrial robotics, pick and place tasks typically require production setups with static pick and place locations, calling for the objects to be placed in specialized holders to prevent any disturbances in their pose (position and orientation) - thus increasing complexity and cost. Using vision-based localization (pose estimation) would allow for more flexible production setups without the need for specialized holders and therefore reduced cost.

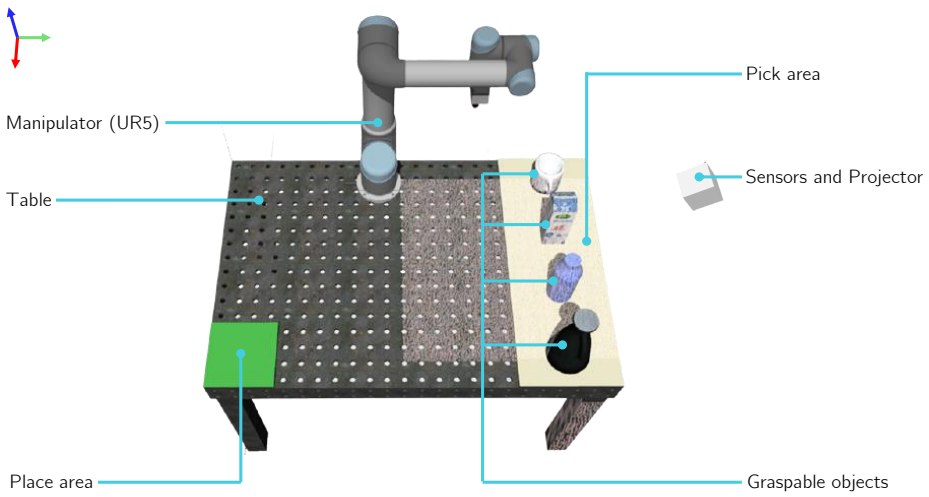


Fig. 1: The robotics and computer vision (ROVI) system workcell in the Gazebo simulation environment, comprised of the UR5 manipulator mounted on a table with designated pick and place areas, graspable objects, and sensors.

In this project, a workcell is established within ROS/Gazebo dynamic simulation environment [1]. The workcell consists of a UR5 manipulator [2] mounted onto a specialized table with designated pick and place areas, furthermore equipped with various perception sensors, as shown in Fig. 1. This setup is used to systematically design and evaluate a vision-based pick and place pipeline, namely the robotics and computer vision (ROVI) system.

1.2 Overview

The ROVI system design process is decomposed into four major components:

- **Establishment of simulation environment and interfaces**
  A simulation environment is established in which the workcell is defined, including the manipulator, graspable objects, perception sensors as well as interfaces for the sensors and robot control.
- **Motion planning of pick and place task**
  A reachability analysis is performed to infer the optimal base position within the workcell. Several approaches are implemented for trajectory generation using motion planning.
- **Vision-based pose estimation**
  Several vision-based pose estimation methods are implemented and statistically evaluated, allowing to estimate the pose of objects within the pick area of the workcell.
- **System integration (pipeline)**
  The optimal planning approach is fused with the optimal vision-based pose estimation approach to establish a complete pipeline in which the manipulator can grasp and relocate an object without any collisions.

2 Simulation environment

A simulated environment and several libraries are used to facilitate the development of the ROVI system, including dynamic simulation of the workcell (manipulator and objects), robot control, perception (cameras and point cloud sensors), motion planning, and computer vision.

There are various middleware solutions [3] that provide a framework for a modular development of robotics systems, one of which is the Robot Operating System (ROS) [4] - it is a state-of-the-art robotics framework and tends to be a standard for robotics application development [5]. ROS facilitates a software framework with reusable components and a common communication pipeline which enables a greatly simplified development with better project manageability. It furthermore leverages a seamless integration with the Gazebo simulator [6], which is designed to accurately reproduce the dynamic environments a robot may encounter.

2.1 Structure

The ROVI system is contained within a single workspace being managed by the `catkin` build system [7] with `rosdep` for dependency management [8]. The functionality of the system is decoupled into several packages, as shown in Table 2, following the separation of concerns principle.

Package	**Description**
`ur5_description`	URDF description of the UR5 robot.
`ur5_dynamics`	Dynamics and kinematics library for the UR5 robot.
`ur5_controllers`	ROS Control controllers for the UR5 robot.
`ur5_moveit_config`	Moveit configuration for the UR5 robot.
`rovi_gazebo`	Integration of ROVI workcell into Gazebo with various interface methods.
`rovi_utils`	Various helper methods for the ROVI system.
`rovi_planner`	Methods relating to motion planning of the UR5 robot in the ROVI system using both KDL and Moveit.
`rovi_pose_estimator`	Vision-based pose estimation methods for objects in the ROVI system workcell.
`rovi_system`	Integration of and testing ground for ROVI system components.

Table 2: Packages in the ROVI system `catkin` workspace.

Packages prefixed with `ur5_` are related to the functionality of the UR5 manipulator, whereas packages prefixed with `rovi_` are related to the design of the ROVI system. The development of packages follows a decoupled development process, where packages are loosely coupled and can thus be developed and tested independently.

2.2 Models and interfaces

In order to establish and interact with the simulated environment within the ROS/Gazebo ecosystem, it is necessary to model and interface: (1) the UR5 manipulator and its attached gripper, and (2) the objects and the sensors of the ROVI workcell.

2.2.1 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 [9], namely Unified Robot Description Format (URDF) and Simulation Description Format (SDF), allowing to define the kinematic chain of an object, alongside the visual, collision and dynamic properties. In the ROS/Gazebo ecosystem, URDF files are used to define the structure of a robot, whereas SDF files are used to model objects and otherwise simulation related entities to be loaded into Gazebo; a URDF model can be augmented to be consumed as a SDF model.

2.2.2 Objects and senors

The graspable objects of the ROVI workcell are modeled in Blender to define their visual 3D appearance, which are then defined as SDF models, allowing these to be loaded into Gazebo.

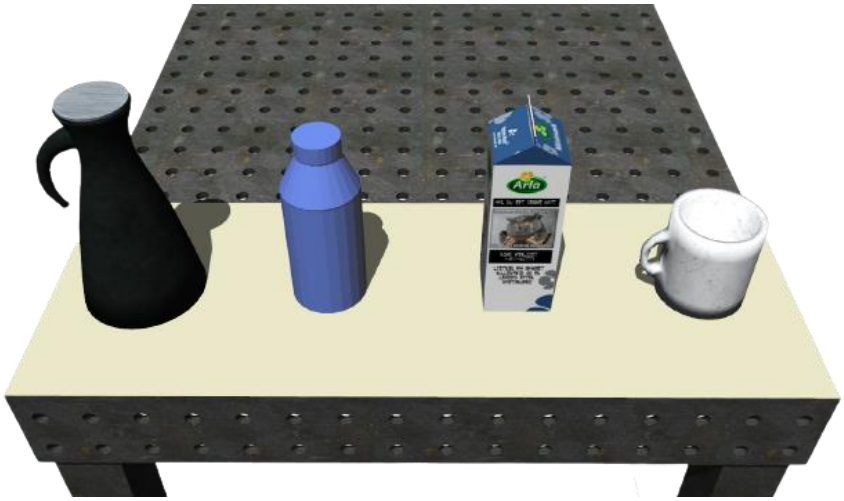


Fig. 2: Example objects of the ROVI system workcell, including (from left to right) a can, a bottle, a milk and a mug.

Some example objects are shown in Fig. 2. The objects are modeled with somewhat realistic inertial properties and friction parameters, enabling a pseudo-realistic dynamic simulation in which the gripper is able to grasp an object without slip or otherwise any peculiar behavior. Dynamic properties are estimated using Meshlab [10] or approximated as simple, rigid bodies (e.g. a homogeneous cylinder). Frictional parameters are manually tuned to provide an empirically well-behaved simulation.

To provide input to the the vision-based pose estimation, several sensors are loaded into the simulation, defined as SDF files using Gazebo plugins, including a camera, a stereo-camera, a Kinect sensor and a structured light projector - all sensors are accessible through the ROS communication interface (via topics). The `rovi_gazebo` package contains any simulation related models and provides an interface to the simulated environment, allowing to: get the pose of a given model, get the tool center point (TCP) pose of the manipulator, get an image from the camera sensors, enable or disable the structured light projector, spawn and remove models, and so forth.

2.2.3 Robot model and dynamics

The UR5 manipulator and gripper of the ROVI system are incorporated into the simulated environment by describing the models of these entities, and defining the forward and inverse kinematics necessary to provide an interface to the manipulator and its TCP.

Robot description

The robot description (URDF) is defined with accordance to its Denavit-Hardenberg (DH) parameters [11] including any joint limits of the UR5; the parameters are shown in Table 3 and visualized in Fig. 3. The dynamic parameters required for dynamic simulation and control are not publically available and are instead approximated as given by [12] with some modifications; these could potentiality be approximated more accurately [13]. Any files related to the model descriptions of the manipulator and gripper are defined in the `ur5_description` package.

Joint	q	d [mm]	α [rad]	a [mm]
1	q_1	89.16	$\frac{\pi}{2}$	0.0
2	q_2	0.0	0.0	−425.0
3	q_3	0.0	0.0	−392.25
4	q_4	109.15	$\frac{\pi}{2}$	q_5
5	q_5	94.65	$-\frac{\pi}{2}$	0.0
6	q_6	82.3	0.0	0.0

Table 3: DH parameters of UR5 manipulator [11].

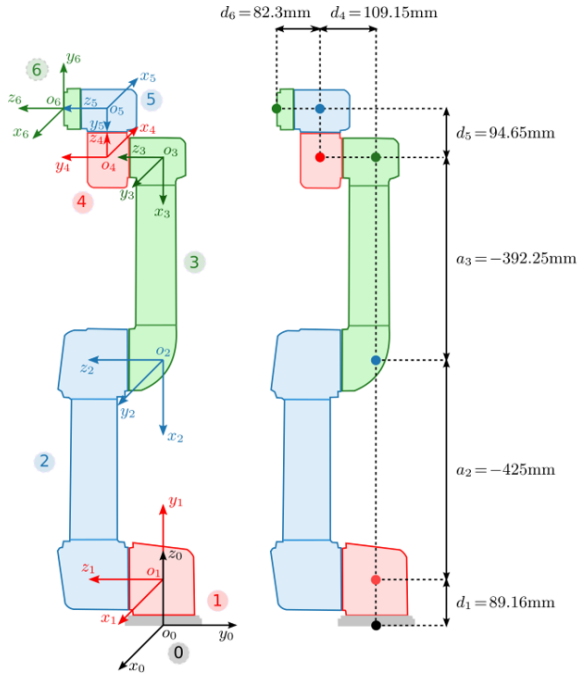


Fig. 3: Illustration of UR5 DH parameters [14].

Gripper description

A WSG-50 servo-electric two-finger parallel gripper from Schunk [15] is attached to the end-effector (flange) of the UR5 manipulator. The gripper, as shown in Fig. 4, is modeled individually, and is contained in the `wsg.xacro` file of the `ur5_description` package.

The fingers of the gripper are custom models with sufficiently high friction parameters to enable grasping. Being 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.

Fig. 4: Schunk WSG-50 parallel gripper.

Forward kinematics

The pose ${}^{b}\mathbf{T}_{n} \in \mathrm{SE}(3)$ of a serial manipulators end-effector is computed by forward kinematic with the joint configuration $\boldsymbol{q} \in \mathbb{R}^{n}$ being known in advance. The computation is performed by multiplying a series of consecutive body transformations ${}^{i}\mathbf{T}_{i+1}(\boldsymbol{q}_i)$, as depicted in (2).

The transformations are rigidly attached virtual coordinate frames, which are attached to the manipulator's links in terms of the standard DH procedure. Arbitrary procedures can be used, but DH supports a minimal representation of the manipulator, i.e., it requires four consecutive rigid body transformations rather than six.

$$ {}^{i-1}\mathbf{T}_{i}(\boldsymbol{q}_i) = \begin{bmatrix} {}^{i-1}\mathbf{R}_{i}(\boldsymbol{q}_i) & {}^{i-1}\boldsymbol{t}_{i}(\boldsymbol{q}_i) \\ \mathbf{0}^{T} & 1 \end{bmatrix}, \tag{1} $$

$$ {}^{0}\mathbf{T}_{6}(\boldsymbol{q}) = \prod_{i=1}^{n=6} {}^{i-1}\mathbf{T}_{i}(\boldsymbol{q}_i). \tag{2} $$

The notation used is slightly abbreviated and excludes the static parameters (d, α, a) from the DH parameters convention.

Inverse kinematics

Determining the joint configurations $\boldsymbol{q}$ for a desired pose ${}^{0}\mathbf{T}_{n}(\boldsymbol{q})$ refers to the inverse kinematics problem; the problem has a finite set of solutions if the number of independent input constraint m is equal to the number of independent joints n, which is the general case for a UR5 manipulator.

The inverse kinematics may be approached in two ways: analytically, by geometry or polynomials [16], or numerically, by gradient descent methods. The analytical approach is preferred if it exists, since it is deterministic and global, whereas numerical methods are quite opposite. The analytical solution is resolved by [17] and utilized. Furthermore, the method is interfaced to return one solution governed by the nearest neighbor algorithm with L2-norm as the metric

$$ \min_{i} ||\boldsymbol{q}_i - \boldsymbol{q}||^2 \quad \text{for} \quad i \in 1, \ldots, 8 \tag{3} $$

where $\boldsymbol{q}_i$ is the i'th solution. Typically, it is desired to determine $\boldsymbol{q}$ for ${}^{0}\mathbf{T}_{tcp}$, which is novel, since ${}^{6}\mathbf{T}_{tcp}$ is a constant transformation matrix - a translation offset on the z-axis - as depicted in Fig. 5.

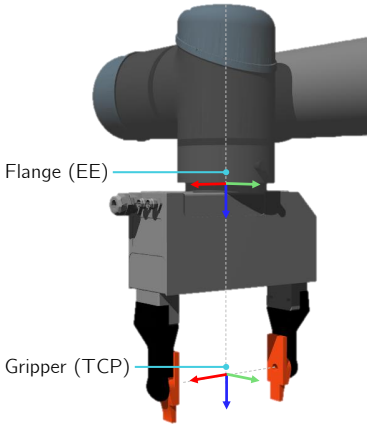


Fig. 5: The UR5 manipulator with an atteached gripper; the frames of the flange (EE) and the gripper (TCP) and co-linear with a constant offset.

Determining $\boldsymbol{q}$ is thus still governed by the aforementioned inverse-kinematic solution, where

$$ {}^{b}\mathbf{T}_{6} = {}^{b}\mathbf{T}_{tcp} \cdot \left({}^{6}\mathbf{T}_{tcp} \right)^{-1}. \tag{4} $$

Dynamics

Control of the manipulator in Gazebo is facilitated by decentralized-control methods [18], which requires the model of the manipulator to be partially or exactly known. The control methods in this domain, provides robust and accurate control of industrial manipulators, given the model and parameters are well-defined, which is the case in simulation.

Differential equations of motion which describe the dynamics of a serial manipulator consisting of n-rigid bodies [19], excluding the gripper attached to the manipulator, are governed by

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

where $\mathbf{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is the symmetric inertia matrix, $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n\times n}$ represents the centrifugal and Coriolis effects, $\mathbf{g}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ 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 [$\mathbf{M}(\boldsymbol{q})$, $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, $\boldsymbol{g}(\boldsymbol{q})$] are computed using Orocos Kinematics and Dynamics Library (KDL) [20] given the kinematic chain of the manipulator for a given robot state [$\boldsymbol{q}$, $\dot{\boldsymbol{q}}$]. An interface to the forward and inverse kinematics, as well as the robot dynamics variables of the UR5 manipulator is contained in the `ur5_dynamics` package.

The gripper dynamics are not modeled since it is simulated as a parallel gripper, which complicates the dynamics and is out of the project's scope. Furthermore, the gripper's gravity is disabled in the simulation environment to avoid gravity compensation in the manipulator controllers.

2.2.4 Robot control

The UR5 manipulator is actuated using ROS Control [21], which is a framework that permits to implement and manage robot controllers in the ROS ecosystem; custom controllers can be implemented [22, 23] by inheriting the classes provided by the framework using one or more of the standard hardware interfaces:

- `PositionJointInterface`
- `VelocityJointInterface`
- `EffortJointInterface`

The `EffortJointInterface` is used to interface the simulated UR5 manipulator, allowing to set the desired joint torques $\boldsymbol{\tau}_d$. The robot state is read using the `JointStateInterface` (available for all hardware interfaces), which retrieves the current robot state of the manipulator in Gazebo.

Controllers and interfaces to these are contained in the `ur5_controllers` package, where a total of two manipulator controllers and a single gripper controller are implemented.

Joint Position controller

Given the model in (5) is applicable, it is possible to realize a decentralized joint position controller as

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

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

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

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

The controller is interfaced by sending `sensor_msgs::JointState` messages to the controller command topic. A joint state trajectory can thus be executed by sequentially transmitting desired joint states; this functionality is facilitated by the `ur5_controllers::ur5::execute_traj(...)` method when invoked with a vector of joint trajectories.

Cartesian Pose controller

Given the model in (5) is applicable, it is possible to realize a decentralized Cartesian position controller as

$$\boldsymbol{\tau}_d = \mathbf{M}\left(\boldsymbol{q}\right)\boldsymbol{y} + \mathbf{C}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}\right)\dot{\boldsymbol{q}} + \boldsymbol{g}\left(\boldsymbol{q}\right), \tag{9}$$

$$\boldsymbol{y} = \mathbf{J}^{+}\left(\ddot{\boldsymbol{x}}_d + \mathbf{D}\tilde{\dot{\boldsymbol{x}}} + \mathbf{S}\tilde{\boldsymbol{x}} - \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}}\right), \tag{10}$$

$$\mathbf{J}^{+} = \left(\mathbf{J}^{T}\mathbf{J} + \epsilon\mathbf{I}\right)^{-1}\mathbf{J}^{T} \tag{11}$$

where (9) is the linearization feedback, which cancels the dynamics of the manipulator, and (10) is the commanded acceleration governed by the Cartesian error reference trajectory

$$\tilde{\ddot{\boldsymbol{x}}}_d + \mathbf{K}_d\tilde{\dot{\boldsymbol{x}}} + \mathbf{K}_p\tilde{\boldsymbol{x}} = 0. \tag{12}$$

The controller is interfaced by sending `ur5_controllers::PoseTwist` messages to the controller command topic. A Cartesian reference trajectory is executed by sequentially transmitting desired Cartesian poses and twists; the `ur5_controllers::ur5::execute_traj(traj)` is overloaded to handle Cartesian trajectories.

Gripper controller

Gravity is not activated, which allows for a simple controller, i.e., sending torques directly in a binary manner: low or high torque. The `ur5_controllers::wsg` namespace provides a `::release()` as well as `::grasp()` method which allows to interface the gripper directly, without having to manually specify a desired torque.

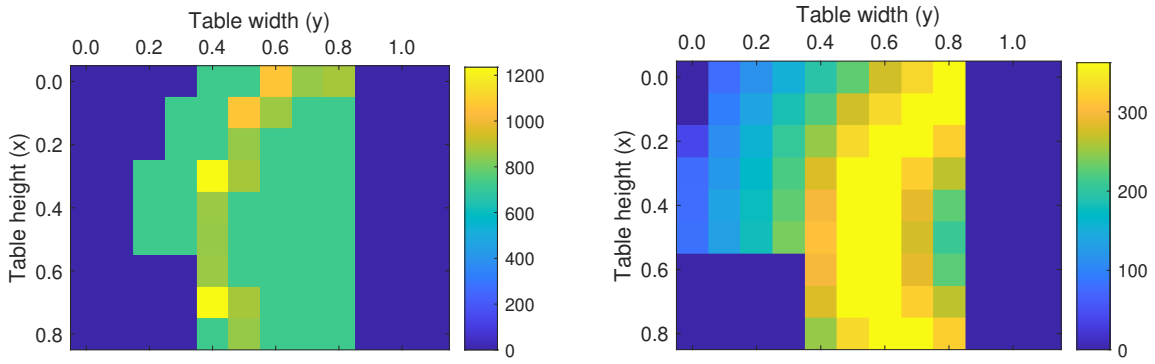
3 Pick and place

Using the configured simulation environment, the pick and place task consists of defining trajectories for the TCP such that the gripper is able to align with the object, grasp it and relocate the object to the place area. This task has several considerations in terms of the manipulator's base position, grasping orientation and motion planning.

3.1 Reachability

The position of the manipulator's base as well as the grasping orientation (from the top or from the side) determines the manipulators capability of optimally performing the pick and place task; this capability is quantified as the manipulator's reachability, defined with respect to the number of collision free kinematic configurations in which the manipulator's TCP can reach both the pick and place poses for a given base position.

To determine the optimal base location, a simplified reachability analysis is performed, which is implemented in the `rovi_planner` package using inverse kinematics from `ur5_dynamics` and collision checking provided by the MoveIt motion planning framework [24]. The table onto which the robot is mounted is discretized into a grid spaced at 0.1 m and a desired pick pose is defined for the TCP at ${}^{w}\boldsymbol{p}_{obj} = [\,0.4\ 1.0\ 0.75\,]$. For each of the two possible grasping orientations (top and side), the manipulator's base is relocated in the xy-plane and the TCP is rotated around the z-axis of the object at a resolution of $\frac{2\pi}{180} = 2°$, checking for collisions at each iteration - the outcome of the study is visualized using heatmaps, as seen in Fig. 6.



(a) Grasp from the top

(b) Grasp from the side

Fig. 6: Reachability heatmaps for different manipulator base positions for the two grasping orientations.

Grasping from the top has a total of 41 445 collision-free joint configurations where grasping from the side has a total of 17 090 collision-free joint configurations. Despite the top-grasping method having a significantly higher number of collision-free joint configurations, grasping from the side provides better grip in the simulated dynamic environment and hence more stable behavior when the gripper is interacting with the graspable objects; the base position is chosen to be the center of the table, providing the optimal reachability when grasping from the side.

Although, when working in a dynamics simulation which seek to simulate the real world, a reachbility metric based on solely the number of collision-free kinematic configurations can be insufficient when seeking to determine the optimal base location for grasping objects, especially when also considering different grasping poses. Instead, the grasping pose for rigid objects could be evaluated with the number of collision-free kinematic configurations being weighted by others metrics, possibly being: (a) how well the gripper can grasp an object with respect to slip or contact points, (b) the manipulability for a single or a set of joint configurations, or (c) the Euclidean distance of a collision-free path between the pick and place poses.

3.2 Motion planning

The realization of pick and place tasks requires a definition of trajectory between the pick pose and the place pose for the robot to follow; such a trajectory must define both the path and the how to traverse such path with respect to time, namely a velocity profile.

3.2.1 Interpolation-based planning

Given the desired pick pose of an object, the objective is to define a trajectory in Cartesian space between the current end-effector 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. 7, with the last two being a pre-pick offset and the pick pose.

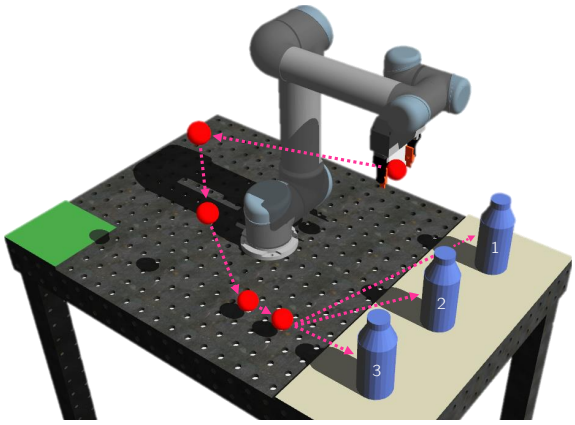


Fig. 7: The ROVI system workcell with the waypoints for linear and parabolic trajectory interpolation methods.

The simplest way of defining a trajectory between two or more Cartesian poses is to interpolate between the poses without the consideration of a collision free path. An interpolation can be performed in both Cartesian space or joint space, with the latter requiring inverse kinematics to compute the desired joint configurations for the waypoints.

Cartesian trajectory generation is achieved using both linear (point-to-point) interpolation and parabolic interpolation, with both methods being implemented in the `rovi_planner` package using KDL. Both trajectory generation methods are invoked with a list of waypoints (poses) given as an array of `geometry_msgs::Pose` objects. The waypoints are iteratively interpolated into trajectory segments and combined into a trajectory composite which is embedded in a `KDL::Trajectory_Composite` object. This trajectory can be invoked with a given timestep $t \geq 0$ to obtain the desired Cartesian position, velocity and acceleration at that timestep.

Linear interpolation

Interpolating between a pair of 3D Cartesian poses is achieved by interpolating the translational and rotational motion separately. The translational interpolation is given by a linear interpolation between an initial position $\boldsymbol{x}_0$ and a final position $\boldsymbol{x}_f$, as

$$\boldsymbol{x}(s) = (1 - s) \cdot \boldsymbol{x}_0 + s \cdot \boldsymbol{x}_f, \tag{13}$$

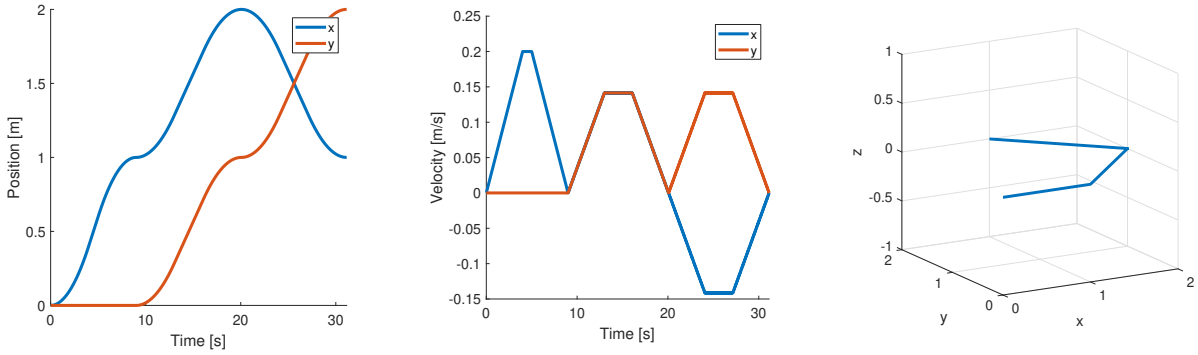
where $\boldsymbol{x}(s) \in \mathbb{R}^3$ is the vector of intermediate positions as a function of $s(t)$, which is a scalar that smoothly varies in the interval $s(t) \in [0, 1]$ as a function of time, e.g., a function governed by a polynomial velocity profile.

Rotational interpolation is given by a single-axis interpolation [25], 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°.

Using linear interpolation between waypoints poses the problem of a discontinuous Cartesian path, which results in infinite velocities between the waypoints. This can be remedied using a Trapezodial velocity profile, which smoothens the Cartesian path, such that end-effector is required to halt at the intersections between waypoints in order to avert infinite velocities.

The linear Cartesian trajectory generation method `rovi_planner::traj_linear(...)` interpolates the path linearly between the waypoints using the `KDL::Path_Line` class, which guarantees that the geometric path in 6D space remains independent of the motion profile parameters by comparing the distances of the rotations and translations, where the shortest in duration is scaled to take as long as the longest [26]. Using the interpolated path, a trajectory segment is computed using a trapezoidal velocity profile with a specified maximum velocity and acceleration. Once all waypoints have been interpolated, a trajectory is composed from the trajectory segments and returned as a `KDL::Trajectory_Composite` object.

A linearly interpolated trajectory is shown in Fig. 8. The maximum velocity is limited to 0.2 $^{m}/_{s}$ and the maximum acceleration is limited to 0.05 $^{m}/_{s^2}$.



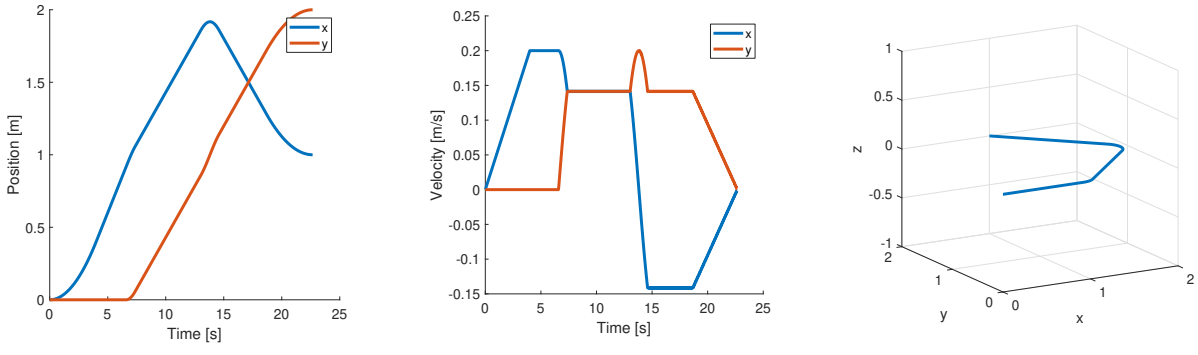
(a) Position **(b)** Velocity **(c)** Trajectory

Fig. 8: Example of Cartesian trajectory generation using linear interpolation defined from four waypoints.

Parabolic interpolation

Defining a continuous (smooth) Cartesian path resolves the issue of infinite velocities without requiring the end-effector to halt between waypoints, which can be accomplished by introducing parabolic blends [27] into the path between waypoints, creating a smooth transition - this is referred to as parabolic interpolation. KDL provides the `Path_RoundedComposite` class [28] for this purpose, allowing to iteratively add waypoints to the rounded path composite, creating a `Path_Line` object between to adjecent points and rounding the path between two lines using a `Path_Circle` object with a specified radius.

The `rovi_planner::traj_parabolic(...)` method abstracts the functionality of the `Path_RoundedComposite` class, iterating the given waypoints and computing a Cartesian trajectory with a specified blend radius; this radius affects the accuracy with which the waypoints are reached. An example parbolic interpolated trajectory is shown in Fig. 9. Constructed from the same waypoints as the linearly interpolated trajectory example of Fig. 9, with identical velocity and acceleration limits, but a blend radius of 0.2 m.



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

Fig. 9: Example of Cartesian trajectory generation using interpolation with parabolic blends defined from four waypoints.

Trajectory generation

The waypoints and desired object poses are defined in world frame (${}^{w}\mathbf{T}_{obj}$) but must be given as desired end-effector poses with respect to the manipulator's base frame (${}^{b}\mathbf{T}_{ee}$) in order for the controller to execute the trajectory. Using the relationships

$$ {}^{w}\mathbf{T}_{tcp} = {}^{w}\mathbf{T}_{b} \cdot {}^{b}\mathbf{T}_{tcp} \quad \text{and} \quad {}^{b}\mathbf{T}_{tcp} = {}^{b}\mathbf{T}_{ee} \cdot {}^{ee}\mathbf{T}_{tcp}, \tag{14} $$

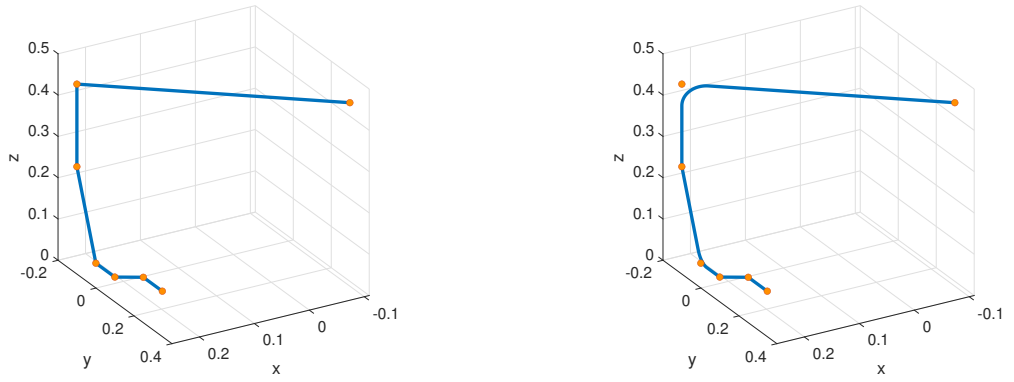
the desired end-effector pose, given an object pose ${}^{w}\mathbf{T}_{obj} = {}^{w}\mathbf{T}_{tcp}$, can be computed as

$$ {}^{b}\mathbf{T}_{ee} = {}^{w}\mathbf{T}_{b}{}^{-1} \cdot {}^{w}\mathbf{T}_{obj} \cdot {}^{obj}\mathbf{T}_{offset} \cdot {}^{ee}\mathbf{T}_{tcp}{}^{-1}. \tag{15} $$

The `rovi_gazebo::get_ee_given_pos(...)` method performs this transformation and allows to specify an optional offset argument ${}^{obj}\mathbf{T}_{offset}$, defining where the object should be grasped. A Cartesian trajectory can thus be generated by defining an array of waypoints for the end-effector. A trajectory then typically requires inverse kinematics to convert the Cartesian poses to joint configurations. The `ur5_controllers::ur5::execute_traj(...)` interface method allows to execute a trajectory using the Cartesian pose controller.

Evaluation

Given the desired waypoints shown in Fig. 7, the `planning_kdl` node of the `rovi_system` package uses the two proposed interpolation-based trajectory generation methods to generate trajectories for three different pick locations of the bottle object in the ROVI workcell. The generated trajectories have the velocity limited to 0.2 $^{m}/_{s}$, the maximum acceleration limited to 0.05 $^{m}/_{s^2}$, and a blend radius of 0.2 m for the parabolic interpolation method - examples of generated paths for the third object pose at ${}^{w}\boldsymbol{p}_{obj} = [\,0.65\ 1.05\ 0.75\,]$ are shown in Fig. 10.



(a) Linear interpolation

(b) Parabolic interpolation.

Fig. 10: Paths of the trajectories generated using the interpolation-based trajectory generation methods for pose 3.

The two methods are evaluated by performing 50 iterations for each pick location per method, logging the planning time and the duration of the generated trajectory, which are shown in Fig. 11 and Table 4, respectively.

Method \ Pose	1	2	3
Linear	22.85 s	20.57 s	19.19 s
Parabolic	17.33 s	15.15 s	13.88 s

Table 4: Trajectory duration of the interpolation methods and different pick poses.

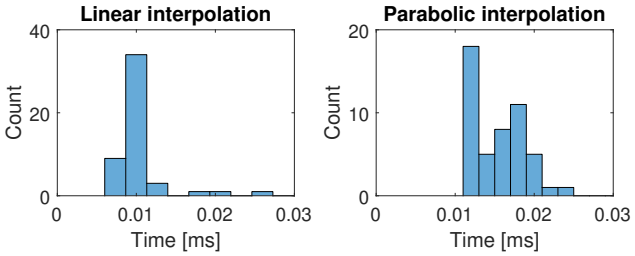


Fig. 11: Planning time histograms of the interpolation methods.

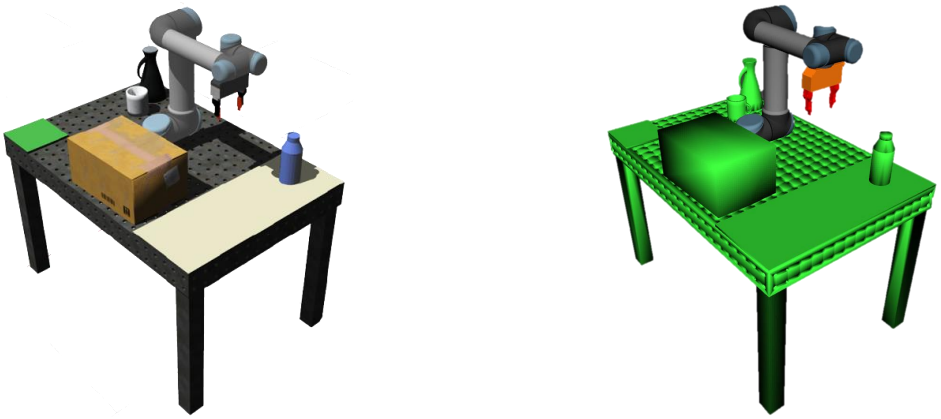
3.2.2 Collision-free planning

Despite the relative simplicity of interpolation-based motion planning, it is not able to guarantee a collision-free path, which is crucial for motion planning of pick and place tasks with non-static object poses. The MoveIt motion planning framework [24] is used to provide collision-free motion planning within the ROS/Gazebo ecosystem. The ROVI system workcell is populated with obstacles as shown in Fig. 12(a).

Planning framework and interface

The MoveIt framework provides numerous capabilities [29], but most importantly it enables motion planning by integrating a robot into the framework and defining a *planning scene*, which is then able to facilitate *motion plan requests* for a given *move group* (e.g. robot arm). The UR5 manipulator is configured using the MoveIt Setup Assistant [30] and is contained within the `ur5_moveit_config` package, including a Semantic Robot Description Format (SRDF) file of the manipulator and other necessary configuration files for use within the MoveIt pipeline.

The planning scene API provided by MoveIt is utilized to define the `rovi_planner::moveit_planner` class, which allows to construct a planning scene for the current state of the ROVI system simulation environment, including the robot state and positions of collision objects, as shown in Fig. 12(b).



(a) Workcell in simulator.

(b) Planning scene representation of collision objects.

Fig. 12: ROVI system workcell with obstacles for collision-free motion planning.

The planner class is able to form motion plan requests using any planner available from the Open Motion Planning Library (OMPL) [31], such as RRT, RRT-Connect and RRT*. A motion plan request is used to specify the desired action for the robot arm move group, such as moving the end-effector of the move group to a new pose. The motion planner checks for collisions (including self-collisions) and allows to attach an object to the robot to account for the object while planning a path. Furthermore, constraints can be defined for the motion plan of the desired action, such as joint-space constraints or orientational constraints for the end-effector.

A MoveIt motion plan is converted into a joint-space trajectory using an Iterative Parabolic Time Parameterization algorithm [32], resulting in an array of joint states. The `ur5_controllers::ur5::execute_traj(...)` interface method provides an overload to execute an array of joint states using the Joint Position Controller.

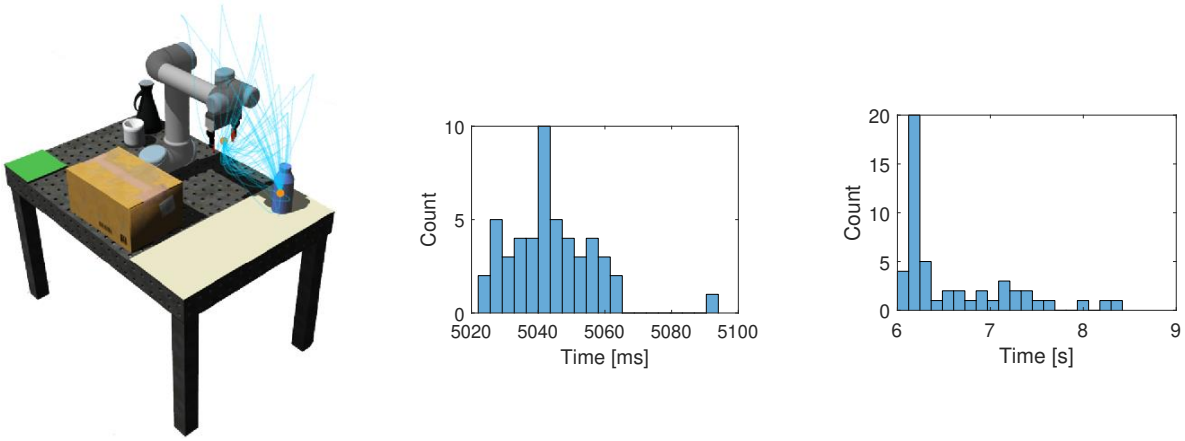
Trajectory generation

Given the planning scene with obstacles, the MoveIt planning interface uses the RRT-Connect OMPL planner to compute a collision-free joint trajectory from the current pose of the end-effector to the desired pick pose; this trajectory is then executed and the gripper is set to grasp the object. With the object grasped, the planning scene is updated and the grasped object is attached to the end-effector; the planning interface computes a trajectory to move the grasped object to the place pose, whereafter the trajectory is executed and the gripper is released, completing the pick and place task.

Evaluation

The trajectory generation using the RRT-Connect OMPL planner is evaluated in the `planning_rrt` node of the `rovi_system` package, which constructs a workcell with obstacles as shown in Fig. 12(a) and spawns the bottle object in one of the three predefined pick poses. A trajectory is attempted to be generated for each of the three pick poses, where each pose is evaluated 50 times with the maximum number of planning iterations limited to 10 000 and the maximum planning time limited to 5 s.

For each planning attempt in which the planner is successfully generates a trajectory, the system logs the generated trajectory in Cartesian space, the planning time, and the duration of the resulting Cartesian trajectory. For the pick task of the first object pose, Fig. 13(a) shows the translational motion of the generated trajectories, where a histogram of planning times is shown in Fig. 13(b) and a histogram of trajectory durations is shown in Fig. 13(c).



(a) Trajectory **(b)** Planning time **(c)** Trajectory duration

Fig. 13: Trajectories generated for the pick task of the first object pose using RRT-Connect planner.

The planner was only able to generate trajectories for the first pick pose of the three that were evaluated, although successfully able to generate a collision-free trajectory in each planning attempt for that pose. The inability to plan for the other two poses is presumably due to the lack of orientational freedom in the z-axis of the grasping pose, which is currently set to a predefined offset from the object pose with a locked orientation, causing a collision with the cardboard box in the workcell. A potential solution is to iteratively attempt different orientations in case that the planner fails, essentially rotating the end-effector around the object at a set resolution, similar to the reachability test. A more advanced approach would facilitate grasp planning capabilities of MoveIt [33].

The place task is not evaluated due to the instabilities of the dynamic simulation and the controller when an object is to be grasped, since this might taint the evaluation of the planner. However, a videographic demonstration of the pick and place task using the RRT* planner [34] exhibits the planners capability of relocating an object with a known pose from the pick area to the place area whilst avoiding collisions.

3.3 Method selection

A statistical comparison of the proposed motion planning methods is erroneous due to the inherit differences in their methodologies. The interpolation-based planning methods use a predefined path resulting in a deterministic execution time with minimal deviation, where as the collision-free planning methods use a stochastic approach to generate a path, making the execution time between the deterministic and stochastic methods incomparable.

Both the interpolation-based and collision-free motion planning methods allow to successfully relocate an object from the pick area to the place area. However, the interpolation-based methods are only suitable for static environments with known poses of possible collision objects, allowing to define waypoints prior to planning in order to avoid collision; and even then, problems with self collisions may occur when using inverse kinematics for trajectory execution.

The collision-free based method using MoveIt, on the other hand, provides a reliable way to generate a collision-free trajectory between the pick and place poses, taking in consideration both the collisions with any objects in the workcell and collisions between the manipulator and grasped object (self-collisions). However, this method comes at the expense of a stochastically varying planning time, where the trajectory generation may occasionally fail and require re-planning. This method is preferred for the ROVI system pick and place tasks, primarily due to its consideration for self-collisions.

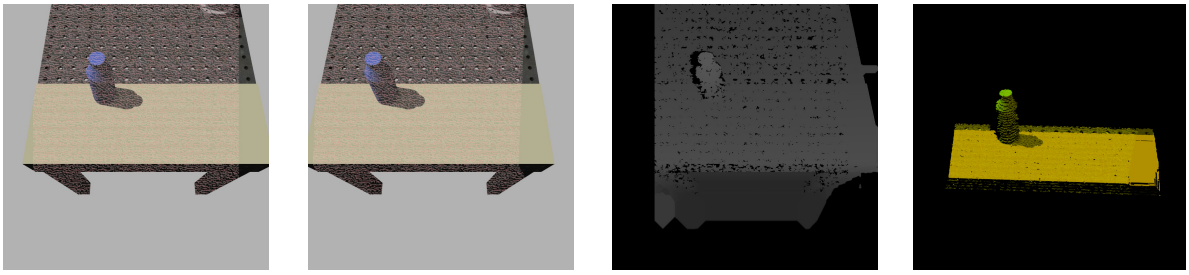
However, as evidenced by the videographic demonstration of the pick and place task, the trajectory between the pick and place location is non-optimal, since, due to the unconstrained motion planning request, the end effector may rotate around its z-axis while transporting the grasped object, which introduces unnecessary motion and may be problematic for certain types of objects; such an issue is resolvable by constraining the orientation of the end-effector, but may increase the planning time.

4 Pose estimation

Pose estimation is defined as localizing the position and orientation of an object, which is required in pick and place applications without hard constrained environments, i.e., the pose may not be statically determined. Two pose estimation methods are implemented to locate objects within the ROVI system, contained in the `rovi_pose_estimator` package - these are then tested and evaluated.

4.1 Dense stereo (M1)

Pose estimation with dense stereo is pursued on the blue bottle depicted in Fig. 14(a). A static structured red light pattern is used to add inhomogeneity to the scene, since the performance of disparity map estimation degrades on uniform, texture-less surfaces.



(a) Left camera **(b)** Right camera **(c)** Disparity map **(d)** Point cloud

Fig. 14: Representations of the the blue bottle object from the ROVI system simulated environment.

4.1.1 Assumptions

Several assumptions and prior knowledge of the scene are utilized to simplify the pose estimation process:

1. The bottle is assumed to be orientated as depicted in Fig. 14 (a). This restriction minimizes the number of Monte Carlo simulations needed to evaluate the algorithm with statistics, since the movement of the bottle is restricted to movement in the xy-plane.
2. The height of the bottle with respect to the world frame is by definition the first assumption deterministic, since the table and camera are both static.
3. The body frame of the bottle is attached to the center of mass (COM) with the z-axis pointing towards the bottle cap. Furthermore, the z-axis is parallel with the z-axis of the table, which provides adequate information to describe the orientation of the bottle for grasping purposes.

4.1.2 Pose estimation

The rectified stereo image pair is retrieved from the simulated environment and converted to 8-bit grayscale. These images are then pre-processed with a Gaussian kernel to suppress high-frequency noise.

The disparity is estimated by using StereoSGBM [35] followed by weighted-least squares filtering as suggested in [36]. An alternative method is to use StereoBM, which increases the uncertainty, but is real-time applicable with less hyperparameters to tune. However, the temporal advantages of this method would be negated by the global pose estimation method, which is the main performance bottleneck.

The pixels are re-projected to 3D by applying dense triangulation, which constructs a point cloud as depicted in Fig. 14(d). Before the point cloud is used for pose estimation is initialized, the point cloud is pre-processed to reduce the number of redundant point using the following operations:

1. A Boxfilter with a conservative choice of parameters is applied to extract the region of interest (ROI).
2. Statistical outlier removal is applied.
3. Plane fitting to remove the table points.
4. Density reduction by voxel grid re-sampling.
5. Normal estimation with left camera as view point.

Spin image features are estimated for the query point cloud (i.e. the point cloud of interest given by the computer-aided design (CAD) model) and the scene. The correspondences between the two clouds are determined by brute-force matching the features with a nearest neighbor search utilizing L1-norm as metric.

The complete pose estimation consists of a global and local stage. The global alignment technique is based on RANSAC model-fitting. The method iterates for a fixed number of iterations, empirically determined to be 10 000 iterations, with each iteration sampling three pairs of feature matches followed by a Kabsch transformation [37]. The determined transformation is applied to the original query point cloud, using inlier points to classify the fitness of the pose; an inlier point is determined by a nearest neighbor search, which is within a threshold gap with L2-norm as metric. The pose with the highest fitness score is considered to be the best pose.

The local pose estimation method, iterative closest point [38], is then applied to minimize the correspondence error between the transformed query point cloud and scene by minimizing the Euclidean distance of the inliers. The method reduces the pose estimation variance (assuming the estimated pose is close to the expected), since the global alignment method is stochastic. The final estimated pose is the transformation provided by the two consecutive pose estimation stages.

4.1.3 Monte Carlo simulations

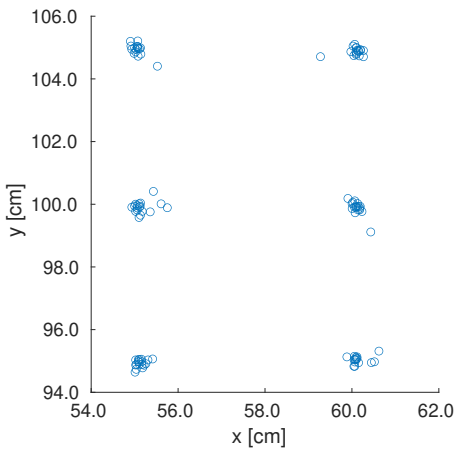
To examine the robustness of the pose estimation pipeline, six known object poses, denoted by $\boldsymbol{T}_i \in \mathbb{R}^{4\times4}$, are estimated with an increasing independent zero-mean normal distributed image noise. The noise is incrementally increased with $\sigma = 3$, starting at zero, until the algorithm starts degrading and is no longer able to provide a robust pose estimate. Twenty simulations are conducted for each pose with a given noise; the corresponding poses are stated in Table 5.

	$\boldsymbol{T}_1$	$\boldsymbol{T}_2$	$\boldsymbol{T}_3$	$\boldsymbol{T}_4$	$\boldsymbol{T}_5$	$\boldsymbol{T}_6$
x [m]	0.55	0.55	0.55	0.60	0.60	0.60
y [m]	0.95	1.00	1.05	0.95	1.00	1.05

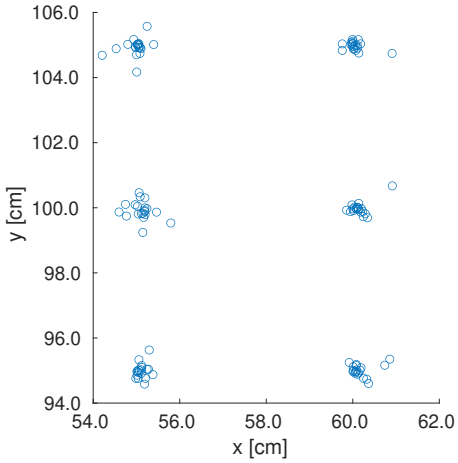
Table 5: The varying variables in the Monte Carlo simulations.

4.1.4 Results

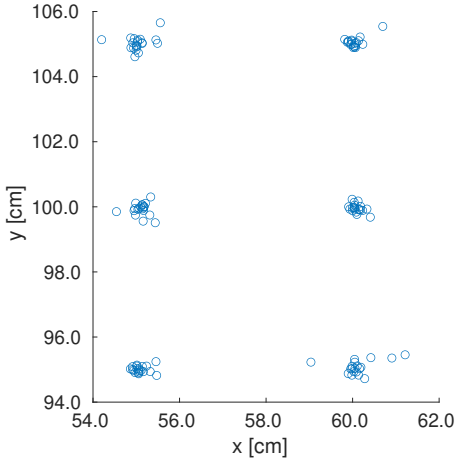
After six experiments, a significant dispersion in the clusters was observed and the experiments were terminated; the experiment results are depicted in Fig. 15.



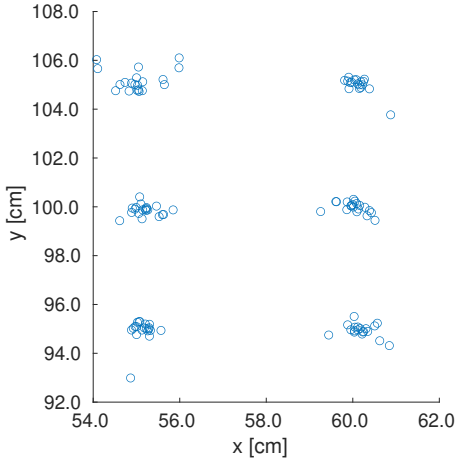
(a) Noise is $\mathcal{N}\left(\mu = 0,\, \sigma^2 = 0^2\right)$



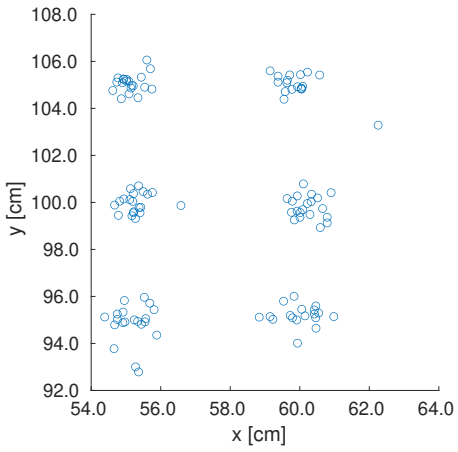
(b) Noise is $\mathcal{N}\left(\mu = 0,\, \sigma^2 = 3^2\right)$



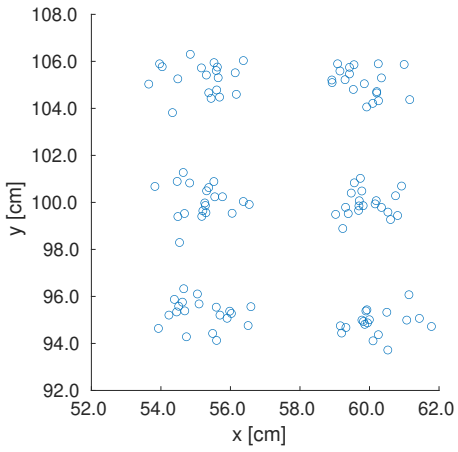
(c) Noise is $\mathcal{N}\left(\mu = 0,\, \sigma^2 = 6^2\right)$



(d) Noise is $\mathcal{N}\left(\mu = 0,\, \sigma^2 = 9^2\right)$



(e) Noise is $\mathcal{N}\left(\mu = 0,\, \sigma^2 = 12^2\right)$



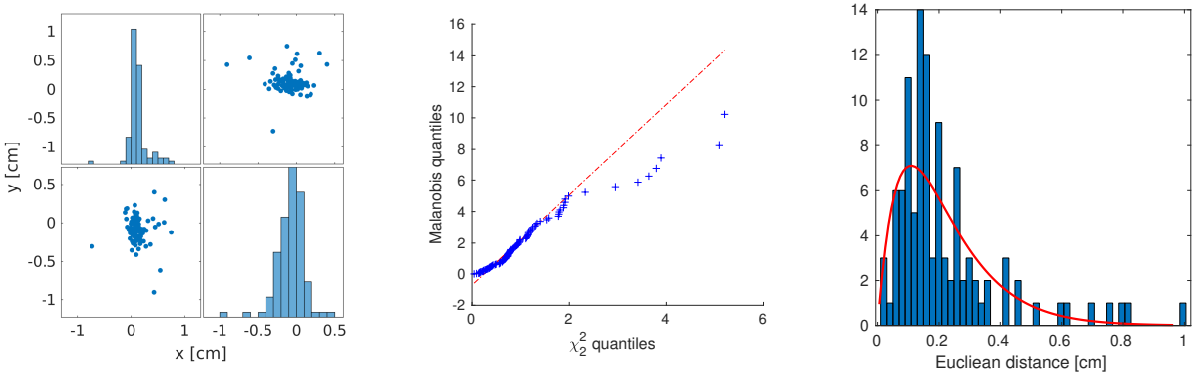
(f) Noise is $\mathcal{N}\left(\mu = 0,\, \sigma^2 = 15^2\right)$

Fig. 15: Six experiments are depicted, the noise is ranging from $\sigma = 0$ to 15 with intervals of three.

4.1.5 Evaluation

A success criterion for grasping the bottle from the side is optimistically a Euclidean distance less than 1.1 cm from the actual position. The threshold value is determined from the radius of the bottle and maximum grasp width of the gripper. Given the criterion and results, a binomial metric can be computed to quantify the success rate of the pose estimation method.

To verify whether descriptive statistics are applicable, a model check is used to assess the assumption of normality on the experiment with zero noise. The histograms of Fig. 16(a) are seemingly normally distributed when excluding outliers. The Q-Q plot in Fig. 16(b) supports the assumption of normality with a few outlier points straying away from the line.



(a) Deviation scatter plot.

(b) Q-Q plot.

(c) Error histogram.

Fig. 16: Normality evaluation of experiment 1.

The normality analysis is repeated for the remaining experiments, with similar results; none of the experiments show coherent signs of abnormality besides outliers, allowing to assume normality on all the experiments. With the assumption of normality, descriptive statistics along with Euclidean distance error histograms and binomial error ratios are used to evaluate the pose estimation algorithm. The results are shown in Table 6 and Fig. 17.

σ	$\|\|\boldsymbol{\mu}\|\|_2^{0.5}$ $[cm]$	$\|\boldsymbol{\Sigma}\|$ $[cm^2]$	error[%]
0	0.1418	0.0008	0
3	0.1093	0.0024	0
6	0.1006	0.0021	1
9	0.0977	0.0119	4
12	0.1993	0.0627	11
15	0.0786	0.2872	34

Table 6: Descriptive statistics and binomial error ratio.

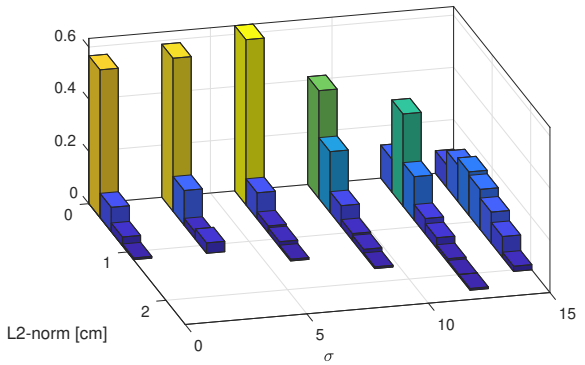


Fig. 17: L2-norm histograms for the different experiments.

The squared mean error $||\boldsymbol{\mu}||_2^{0.5}$ is approximately zero mean for all experiments, and the total variance and error increases incrementally with σ as expected. The bivariate histogram shown in Fig. 17 gives a visual intuition of the change in Euclidean error rate as a function of the increasing variance. For $\sigma = \{0, 3, 6, 9\}$, the low error rate is presumably optimistic since the global alignment method is stochastic with 10 000 RANSAC iterations.

The algorithm is neither examined with other types of objects or in multiple scenarios (e.g., the bottle with various orientations), making it difficult to deduce whether the deduced error rate is trustworthy; in any case, a wider range of experiments should therefore be conducted to negate the stochasticity of RANSAC.

4.2 Sparse stereo (M3)

For objects with identifiable features (using e.g. SIFT), the pose can be estimated using the sparse stereo method, which is based on feature extraction followed by a matching of those features from different views. In the ROVI system, this approach is applied to a simple cube object - an example of an estimated pose is shown in Fig. 18, utilizing the corners of the cube as features.

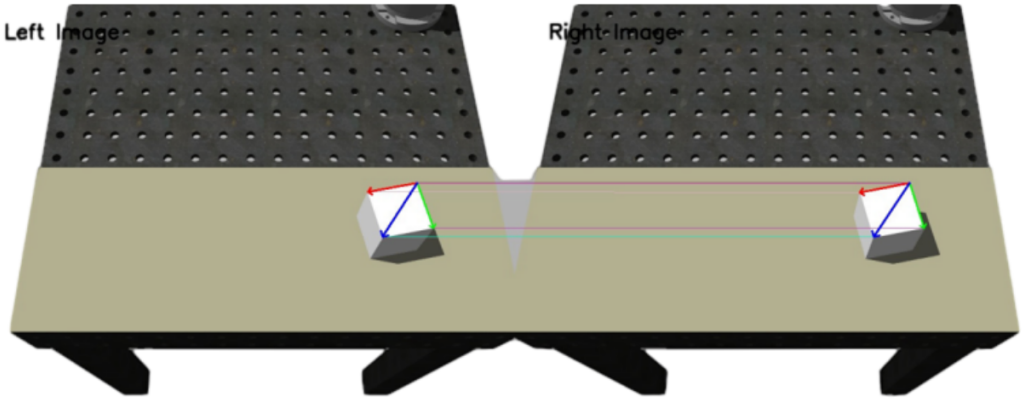


Fig. 18: The sparse stereo algorithm generates a right-hand sided coordinate frame in a random corner of the cube, which functions as a pose with respect to the world coordinate system. The blue vector should not be confused with the z-axis; it is the vector to the diagonal corner.

4.2.1 Assumptions

Several assumptions and constraints are utilized to estimate the pose of the cube:

1. The z-position of the cube is deterministic by virtue of the table and camera are assumed motionless, and the cube is resting flat, surface to surface, with the table.
2. The body frame of the cube is rigidly attached to an arbitrary corner, with the positive z-axis remaining parallel to the positive z-axis of the world frame. With this assumption, it is possible to describe the orientation of the cube with a single parameter.

4.2.2 Pose estimation

The rectified stereo image pair of the scene is retrieved from the simulated environment. Both images are pre-processed with the following methods in order to extract the corners of the cube in each of the stereo images:

1. The region of interest (ROI) (pick area) is extracted.
2. Gaussian filtering and morphology openings are applied to suppress high-frequency noise.
3. Canny edge detection is applied.
4. Contours are extracted [39], keeping contours with an area above a threshold, expecting only a single contour.
5. Detecting corners (from the filled contours) using Shi-Tomasi Corner Detector [40].

The corner correspondences in the image pair are then determined by L1-norm brute-force. The spatial knowledge of the pixels in a rectified stereo camera allows for a constrained search: the corners in the left image will appear as vertically right-shifted corners in the right image, which ideally lie on the epipolar line. Stereo triangulation is then employed on the corner pairs to enable pose estimation.

To determine the estimated pose, one of the four 3D points is selected, then searching for the two nearest 3D points with respect to the Euclidean distance. With the three points, it is possible to constitute a right-handed coordinate system, from which the estimated pose is extracted.

4.2.3 Monte Carlo

To examine the robustness of the pose estimation pipeline, six known object poses, denoted by $\boldsymbol{T}_i \in \mathbb{R}^{4\times4}$, are estimated with an increasing independent zero-mean normal distributed image noise. The noise is incrementally increased by $\sigma = 3$, with initial start at zero, and the experiments are stopped once performance degeneration occurs. Twenty simulations, with the same noise level, are conducted for each pose $\boldsymbol{T}_i$. The varying poses are shown in Table 7.

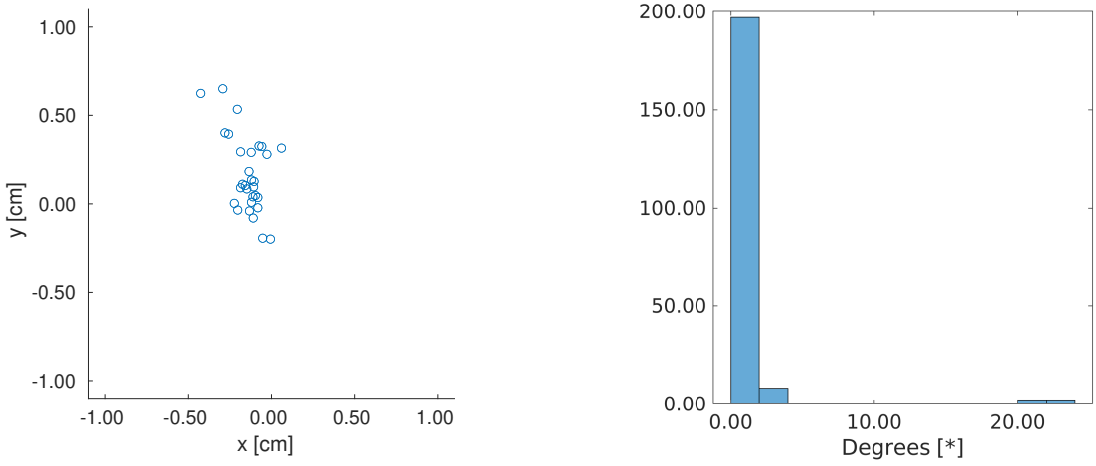
	$\boldsymbol{T}_1$	$\boldsymbol{T}_2$	$\boldsymbol{T}_3$	$\boldsymbol{T}_4$	$\boldsymbol{T}_5$	$\boldsymbol{T}_6$
x [m]	0.10	0.20	0.30	0.40	0.50	0.60
y [m]	1.00	1.03	1.05	1.03	0.98	1.01
roll [rad]	0.1	0.2	0.3	0.4	0.5	0.6

Table 7: The varying variables in the Monte Carlo simulations. Roll is defined as a rotation about the world frames positive z-axis.

The algorithm is deterministic when $\sigma = 0$, this is treated as a special case, where all combinations of poses are examined.

4.2.4 Results

The fourth experiment, $\sigma = 12$, shows significant degeneration, which led to an termination of the experiments. The results of the first experiment are summarized in Fig. 19 with a scatter plot of deviation and a orientation error histogram. The results do not include all outliers (± 1.1 cm and ±10°), hence they were intractable to plot.

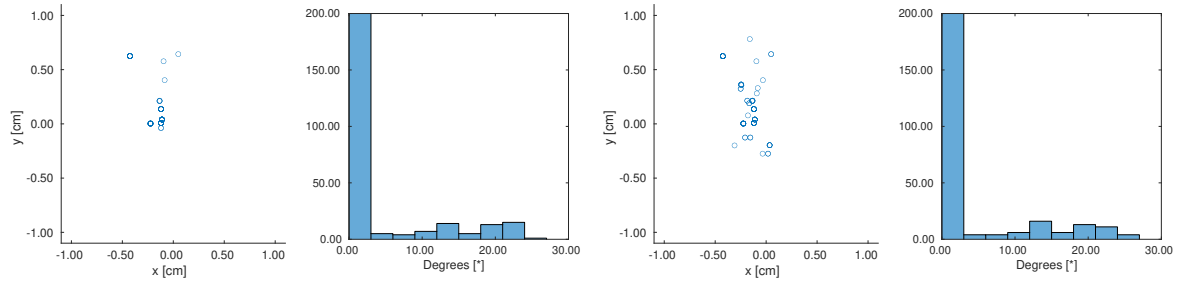


(a) Scatter plot of deviation.

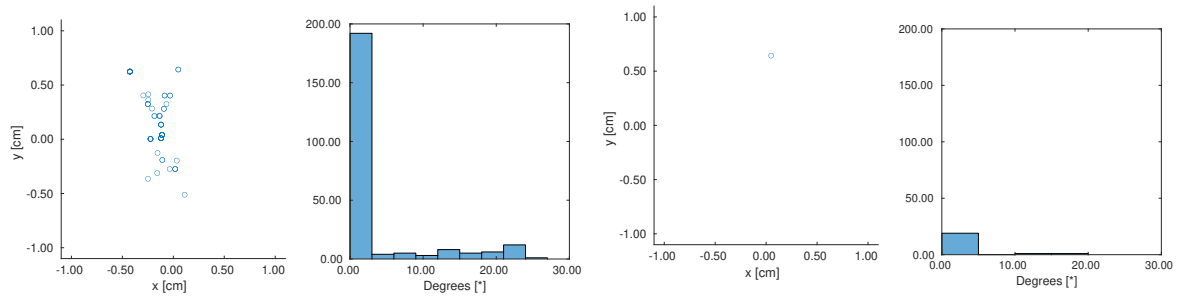
(b) Histogram of orientation error.

Fig. 19: This depicts the results of the noiseless experiment.

The remaining experiment results are depicted in Fig. 20.



(a) Scatter plot of deviation, $\sigma = 3$. **(b)** Orientation error $\sigma = 3$. **(c)** Scatter plot of deviation, $\sigma = 6$. **(d)** Orientation error for $\sigma = 6$.



(e) Scatter plot of deviation, $\sigma = 9$. **(f)** Orientation error for $\sigma = 9$. **(g)** Scatter plot of deviation, $\sigma = 12$. **(h)** Orientation error for $\sigma = 12$.

Fig. 20: Four experiments are depicted, the noise is ranging from $\sigma = 3$ to 12 with intervals of three.

4.2.5 Evaluation

A binary metric is utilized to evaluate the effectiveness of the algorithm, since a continuous metric, such as assuming normality and utilizing descriptive statistics, is inappropriate due to the number of outliers; the shape of the graphs and the rapid performance degeneration in the last experiment.

A successful experiment is defined as being within the L2-norm of 1.1 cm and orientation deviation of 10°. The results are shown in Table 8, showing both the individual L2-norm and orientation error, as well as the total error.

σ	L2-error [%]	Orientation error [%]	Total [%]
0	10	10	17
3	17	17	17
6	17	17	17
9	34	20	38
12	96	88	98

Table 8: The error rates from the experiments.

A systematic error-rate is observed in the first three experiments, which all occur at roll of 0.1 rad and are caused by a mismatch of correspondences, which is shown in Fig. 21.

In the fourth experiment with $\sigma = 9$, the error rate increases rapidly; nevertheless, the algorithm can still extract the corners of the cube. The fifth experiment breaks the algorithm, since it relies on finding a single contour, but instead detects numerous, as shown in Fig. 22.

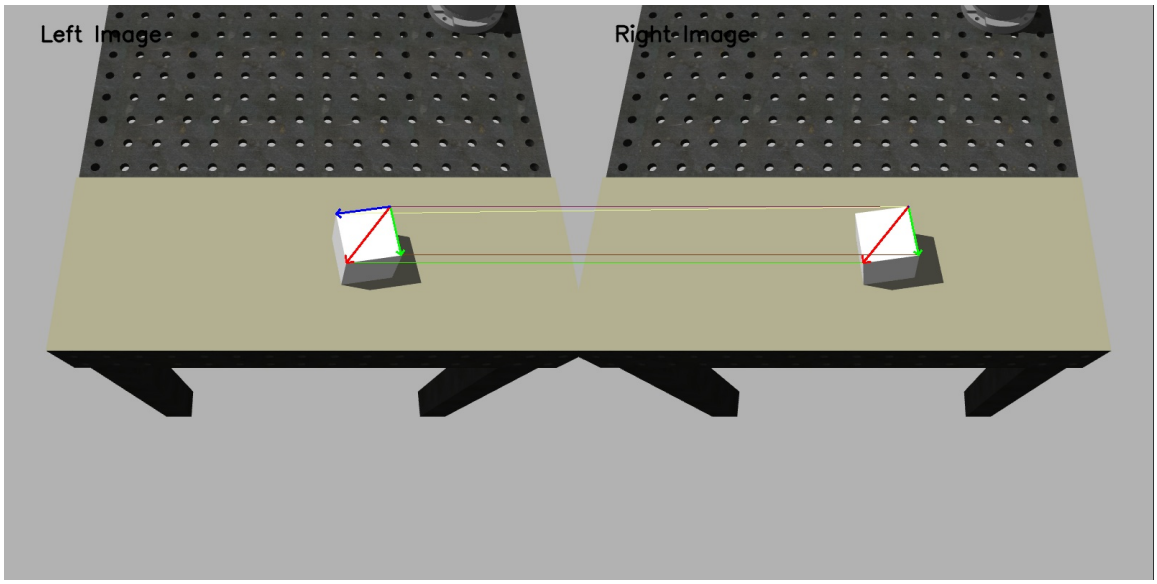


Fig. 21: Wrong correspondences, roll = 0.1 rad.

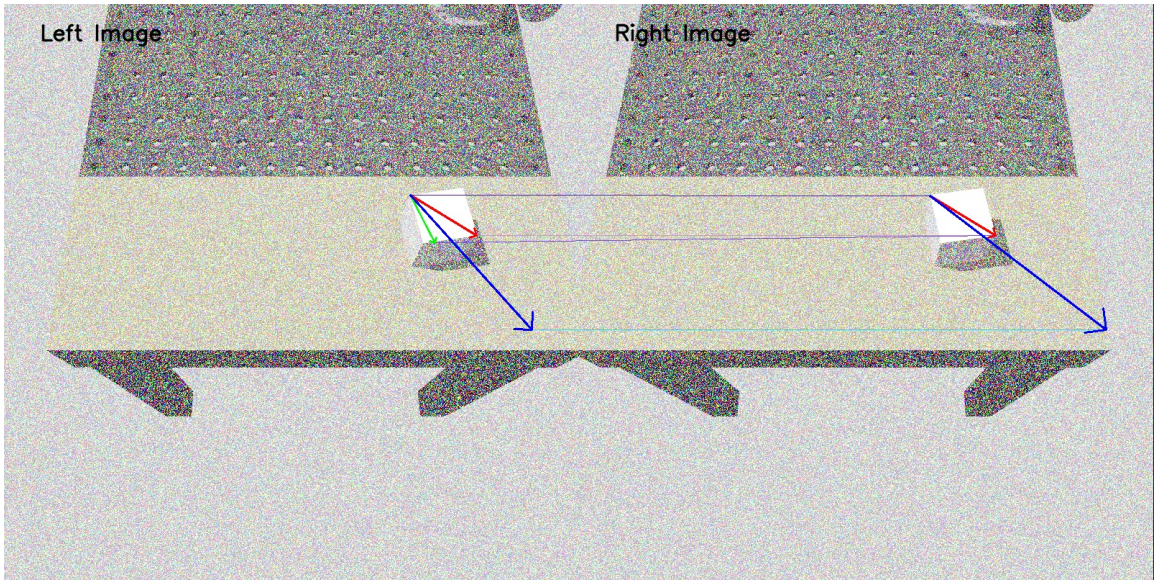


Fig. 22: Numerous contours are detected, so "random" corners are utilized and thus breaks the algorithm.

4.3 Evaluation

The implemented dense stereo algorithm allows for arbitrary pose estimation given a CAD model of the object of interest. In contrast, the sparse method is specialized for cubes and does not require prior knowledge of the dimensions. The dense stereo algorithm performs slightly better in terms of the added Gaussian noise. However, with the expense of long computational time; parallelization can presumably make the method suited for real-time applications, but may require expensive hardware for sufficient results.

The sparse method is real-time applicable, with further improvement on noise handling and error handling (on the two before mentioned errors), it is anticipated to provide for robust pose estimation of cubes - it is therefore chosen for pose estimation within the ROVI system.

5 System integration

The ROVI system integrates the vision-based pick and place system pipeline, which is comprised of several components that all interact with the simulation environment, namely: (a) vision-based pose estimation, (b) motion planning using the MoveIt planning interface with any of the RRT planners, and (c) robot control using a Joint Position Controller - an overview of the system is shown in Fig. 23.

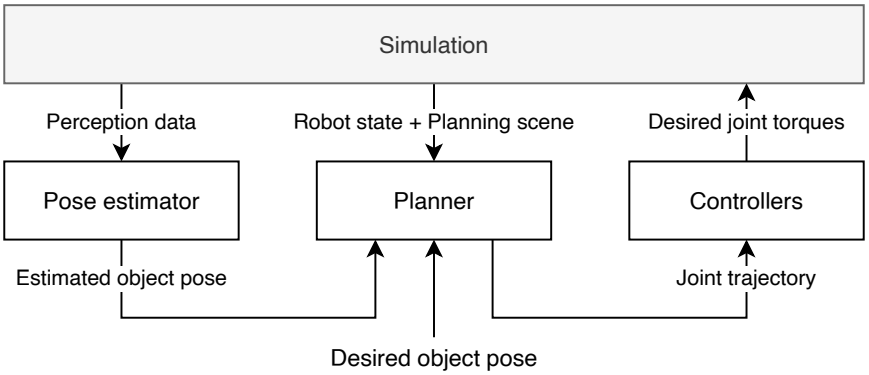


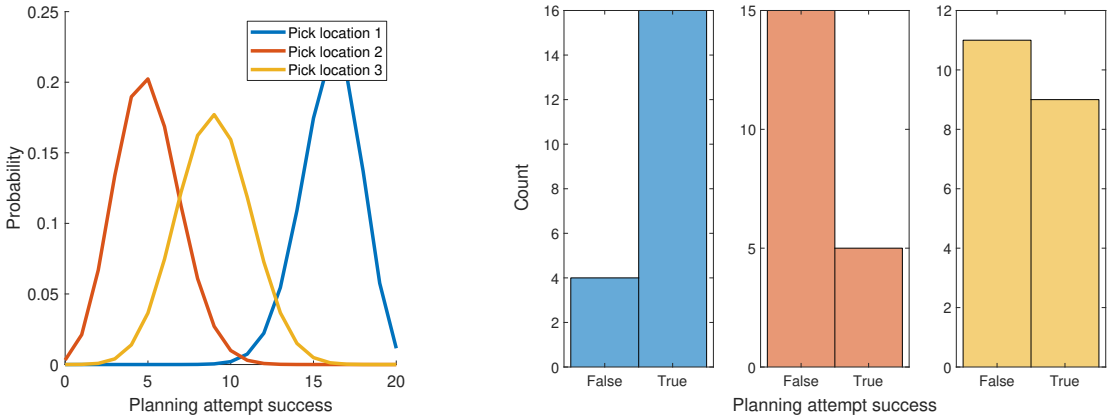
Fig. 23: Overview of the ROVI system integration pipeline containing pose estimation, motion planning and robot control.

The pipeline of the integrated system enables a dynamic relocation of objects from the pick area to the place area, despite any collision objects within the workcell, as evidenced by the videographic demonstration [41]. The integration of the system is contained within the `planning_integrated` node of the `rovi_system` package. This node is similar to the `planning_rrt` node, where the hard-coded pick locations are replaced with estimated pose locations, in which the poses of objects are identified using the M1 dense-stereo pose estimation method.

5.1 Evaluation

The integration of pose estimation into the pick task is evaluated using a binomial experiment based on whether the planner is able to use an estimated pose to find a solution using RRT* within the specified planning and pose estimation constraints of 5 s maximum planning time, 10 000 maximum number of planning attempts, 10 000 RANSAC iterations and a zero-mean Gaussian image noise with a standard deviation of $\sigma = 6$.

The experiment is conducted using three object pick locations (for which a plan is plausible), where for each location, the pose is repeatedly estimated and the planner attempts to generate a trajectory - this is repeated 20 times per pick location and the boolean result is logged for each planning attempt. The results are shown in Fig. 24.



(a) Binomial probability density functions

(b) Histograms

Fig. 24: Results of integrated planning attempts using RRT* and estimated object poses.

For the first pick location, the system integration test boast a success rate of 80 %, which is also evidenced by the binomial probability density function as seen in Fig. 24(a). The second and third pick locations boast a lower success rate of respectively 75 % and 55 %. However, in the evaluation of the collision-free planner, the second and third locations were unplannable due to the grasping orientation. Since the pose estimation method provides an arbitrary orientation revolving around the z-axis of an object, there is now a probability that a plan is found for these pick locations.

To provide a more robust system integration, the orientation of the estimated pose for simple object shapes (e.g. cylinders) could be overridden by a predefined grasping orientation; this would also to manually vary the orientation around the z-axis of an object, in case a collision hinders planning. Considerably lowering the planning and pose estimation constraints would allow for faster overall planning times, since the values used for values are not optimized and chosen somewhat arbitrarily.

5.2 Conclusion and discussion

By integrating motion planning and vision-based pose estimation, the ROVI system is able to successfully execute a pick and place task for the bottle object in the dynamically simulated ROS/Gazebo workcell. However, a more rigid evaluation is desirable in order to fully determine the capabilities of both the integrated system and the individual methods; an improved evaluation would include different objects, randomized object poses, multiple environments with different obstacles and an increased number of repetitions per experiment.

The dynamic simulation environment has been configured with parameters that enable a well-behaved simulation and does not focus on a realistic simulation. If the pick and place task pipeline is to be carried over and implemented in a physical setting, it is necessary to first ensure a proper simulation by appropriately modeling the dynamics of e.g. the gripper, graspable objects and so forth. Furthermore, the controllers must also be tuned to optimally function with a more realistic simulation environment.

References

[1] Qian, Wei et al. "Manipulation Task Simulation using ROS and Gazebo". In: *2014 IEEE International Conference on Robotics and Biomimetics, IEEE ROBIO 2014* (Dec. 2014). DOI: 10.1109/ROBIO.2014.7090732.

[2] Universal Robots. *The UR5 collaborative robot arm of the CB3 family*. URL: https://www.universal-robots.com/cb3/.

[3] Tsardoulias, Emmanouil G. and Mitkas, Pericles A. "Robotic frameworks, architectures and middleware comparison". In: *CoRR* abs/1711.06842 (Nov. 2017). arXiv: 1711.06842. URL: http://arxiv.org/abs/1711.06842.

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

[5] 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.

[6] 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.

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

[8] Garage, Willow. *rosdep: command-line tool for installing system dependencies*. URL: http://docs.ros.org/en/independent/api/rosdep/html/.

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

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

[11] 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/.

[12] 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.

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

[14] 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.

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

[16] Diankov, Rosen. "Automated Construction of Robotic Manipulation Programs". PhD thesis. Carnegie Mellon University, Robotics Institute, Aug. 2010. URL: http://www.programmingvision.com/rosen_diankov_thesis.pdf.

[17] Robot, Universal. https://github.com/ros-industrial/universal_robot. 2020.

[18] 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.

[19] 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.

[20] Ruben Smits Erwin Aertbelien, Orocos Developers. *DIFFERENCE BETWEEN DENAVIT - HARTENBERG (D-H) CLASSICAL AND MODIFIED CONVENTIONS FOR FORWARD KINEMATICS OF ROBOTS WITH CASE STUDY*. http://www.orocos.org/kdl. [Online; accessed 21-may-2020]. 2014.

[21] 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.

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

[23] 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.

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

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

[26] The Orocos Project. *KDL: PathLine Class*. URL: http://docs.ros.org/en/melodic/api/orocos_kdl/html/classKDL_1_1Path__Line.html#a1ea3f21f577aee2a4252c5a802b6a7f2.

[27] Kunz, Tobias and Stilman, Mike. *Turning paths into trajectories using parabolic blends*. Tech. rep. Georgia Institute of Technology, 2011.

[28] The Orocos Project. *KDL: PathLine Class*. URL: http://docs.ros.org/en/melodic/api/orocos_kdl/html/classKDL_1_1Path__RoundedComposite.html.

[29] MoveIt. *MoveIt Concepts*. URL: https://moveit.ros.org/documentation/concepts/.

[30] MoveIt. *MoveIt Setup Assistant*. URL: http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html.

[31] 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.

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

[33] MoveIt. *MoveIt Grasps*. URL: https://ros-planning.github.io/moveit_tutorials/doc/moveit_grasps/moveit_grasps_tutorial.html.

[34] Androvich, Martin and Schøn, Daniel Tofte. *Demonstration of ROVI Pick and Place task using RRT* (ROS/Gazebo)*. Youtube. 2021. URL: https://www.youtube.com/watch?v=VVeNA2eW8yw.

[35] Hirschmuller, H. "Stereo Processing by Semiglobal Matching and Mutual Information". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (2008), pp. 328–341. DOI: 10.1109/TPAMI.2007.1166.

[36] OpenCV. *Disparity map post-filtering*. 2020. URL: https://docs.opencv.org/master/d3/d14/tutorial_ximgproc_disparity_filtering.html (visited on 12/08/2020).

[37] Eggert, D.W., Lorusso, Alice, and Fisher, Robert. "Estimating 3-D Rigid Body Transformations: A Comparison of Four Major Algorithms". In: *Machine Vision and Applications* 9 (Mar. 1997), pp. 272–290. DOI: 10.1007/s001380050048.

[38] PCL. *Iterative Closest Point*. 2020. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/interactive_icp.html (visited on 12/08/2020).

[39] Suzuki, Satoshi and be, KeiichiA. "Topological structural analysis of digitized binary images by border following". In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985), pp. 32–46. ISSN: 0734-189X. DOI: https://doi.org/10.1016/0734-189X(85)90016-7. URL: http://www.sciencedirect.com/science/article/pii/0734189X85900167.

[40] Jianbo Shi and Tomasi. "Good features to track". In: (1994), pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[41] Androvich, Martin and Schøn, Daniel Tofte. *Integrated ROVI Pick and Place task using RRT* and dense stereo pose estimation (ROS/Gazebo)*. Youtube. 2021. URL: https://www.youtube.com/watch?v=7yV93-zxQeA.