University of Southern Denmark

Robotics and Computer Vision

Robot Systems 7th semester - Autumn 2019

RoVi Final Project

Bjarke Engsig Larsen
130795
blars16@student.sdu.dk

Mathias Emil Slettemark-Nielsen
230295
masle16@student.sdu.dk

Supervisor: Aljaz Kramberger & Dirk Kraft

Project deadline: 13/12-2019

Word count: 5732 words

Workload for Each Contributor

Mathias Emil Slettemark-Nielsen (masle16):

- Implementations and Experiments:
  - Reachability Analysis
  - Rapidly-exploring Random Trees Connect
  - Method 2: Simulated Depth Sensor
  - Combination of Robotics and Vision
- Sections:
  - The Design of the Workcell
  - Robot Motion Planning: Rapidly-exploring Random Trees Connect
  - Method 2: Simulated Depth Sensor
  - Comparison of the Pose Estimation Methods Implemented
  - Combining Pose Estimation with Pick and Place Execution
  - Discussion (Mathias)
  - Conclusion

Bjarke Engsig Larsen (blars16):

- Implementations and experiments:
  - Point to Point Interpolation
  - Point to Point Interpolaton with Parabolic blend
  - Method 3: Sparse Stereo
- Sections:
  - Introduction
  - Robot Motion Planning: Point to Point Interpolation
  - Robot Motion Planning: Point to Point Interpolation with Parabolic Blend
  - Comparison of the Robot Motion Planning Methods Implemented
  - Method 3: Sparse Stereo
  - Discussion (Bjarke)

Contents

1 Introduction

This report goes through five different methods related to pick and place of objects using a Universal Robots robot arm in a simulated environment using RobWorks [1]. The first three methods are related to moving the object from the pick location to the place location and the last two methods are for detecting the object and determining a pose using computer vision. Lastly, a combination is put together to perform a complete pick and placement.

The code for the project can be found at [2].

Notation

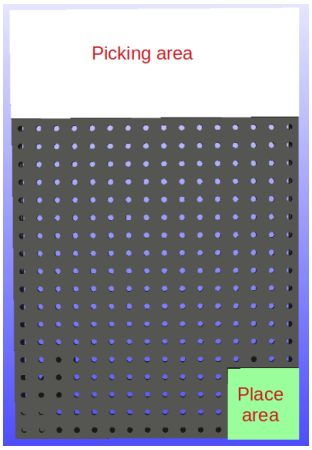
Throughout the report transformation and pose will be used interchangeably, both denote the homogeneous transformation matrix.

2 Pick and Place Execution

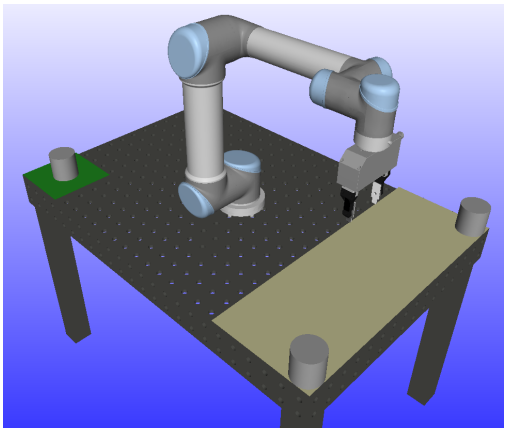
In this section, the robotics part of the pick and place solution is covered. Firstly in Section 2.1, the design of the workcell is described. Secondly in Section 2.2, three algorithms for executing the pick and place action are described. The three algorithms are point to point interpolation, point to point interpolation with parabolic blend and rapidly-exploring random tree connect. The three algorithms will be evaluated and compared, and the best performing algorithm will be chosen to be used in the combination of robotics and vision in Section 4.

2.1 The Design of the Workcell

In this section, the reachability of the manipulator on the robot will be analyzed, by placing the robot base at different positions on the table. The table is shown in Figures 1a and 1b. The manipulator must be able to reach the object at every position in the picking area, and then place the object in the center of the place area with as many collision free kinematic configurations as possible. In Figure 1a the picking and place areas are shown. To perform the reachability analysis the scene shown in Figure 1b is used, the three different placements for picking and placing are shown here and the object used for the analysis. The positions of the cylinders in the picking area are chosen because these are the extremes.



(a) Illustration of the picking and place areas on the table.



(b) Illustration of the placement of the cylinders and the initial placement of the robot base.

Figure 1: Illustrations of the table for the reachability analysis.

In the reachability analysis two grasp configurations are considered: grasping from the side and grasping from the top. The grasp configurations are shown in Figure 2, where the right most figure shows grasping from the top and the other two show grasping from

the side. Figure 2 also shows three bad positions of the robot base, because the robot is close to singularities. Singularities are to be avoided, because of the high joint velocities resulting from passing close to them.

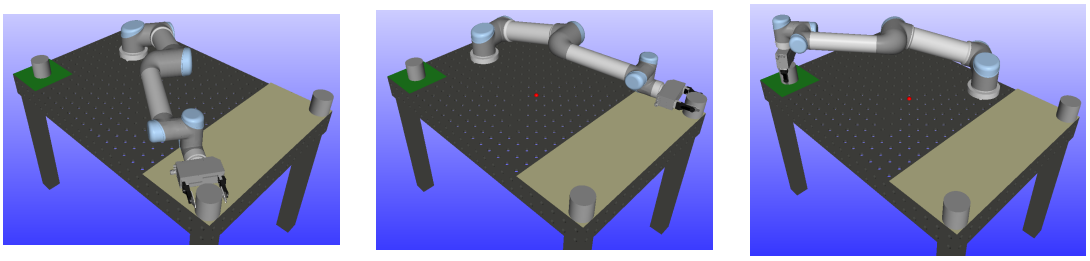


Figure 2: Illustrations of robot base positions which come close to singularities.

The almost singularity in Figure 2 is the elbow singularity, where joint two, three and four are coplanar. Other singularities which can occur are a wrist singularity and a shoulder singularity.

Grasping from the side:

The results from the reachability analysis when grasping from the side are illustrated in Figure 3.

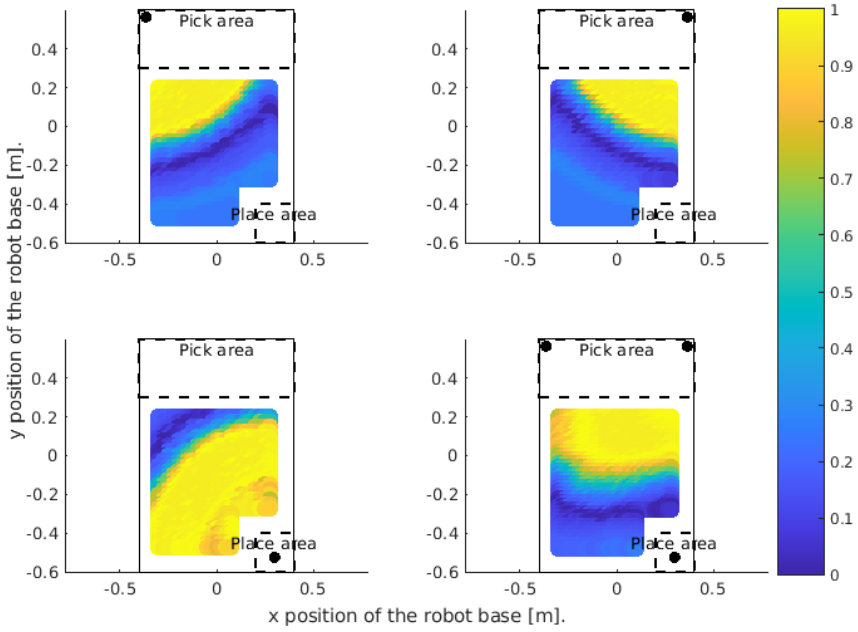


Figure 3: Scatter plots of the robot arm's ability to grasp a cylinder approaching from the side. The black circle indicates the cylinder position. The color scale indicates collision free kinematic solutions for the robot arm.

In Figure 3, the cylinder position is illustrated as a black circle in the pick and place area. The color scale at the right of the figure indicates collision free kinematic solutions, e.g. a value of 1 means the manipulator can reach the cylinder from all 360 degrees. The bottom right figure shows a combination of all cylinder positions, which indicates a good position of the robot base at the top half.

Grasping from the top:

The results from the reachability analysis when grasping from the top are illustrated in Figure 4.

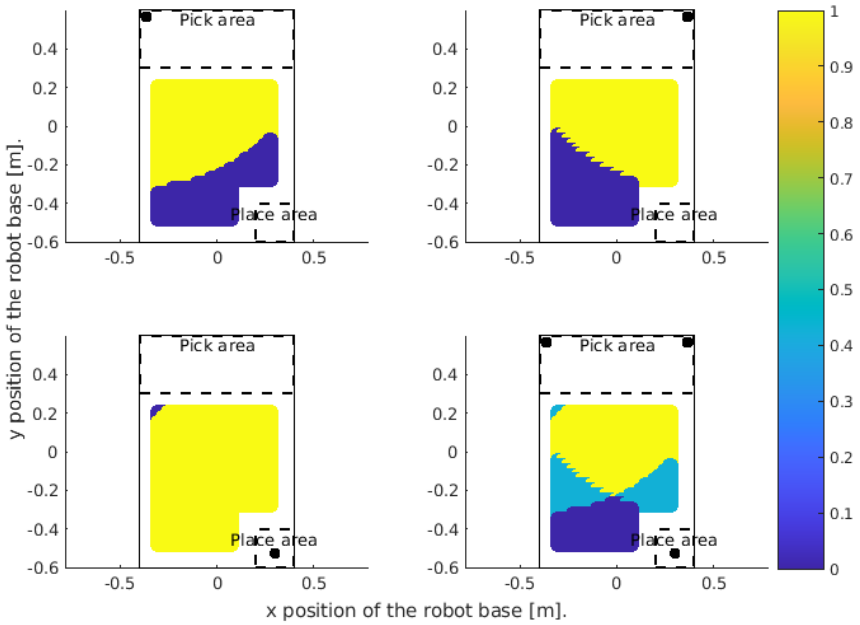


Figure 4: Scatter plots of the robot arm's ability to grasp a cylinder approaching from the top. The black circle indicates the cylinder position. The color scale indicates collision free kinematic solutions for the cylinders.

The bottom right figure in Figure 4 indicates a good position of the robot base at the half of the table near the picking area.

Based upon the two reachability analyses, grasping from the side and grasping from the top, the best position of the robot base is at the half closest to the picking area. The robot base is placed at $(0.2, 0.0)$ on the table, which is illustrated in Figure 5.

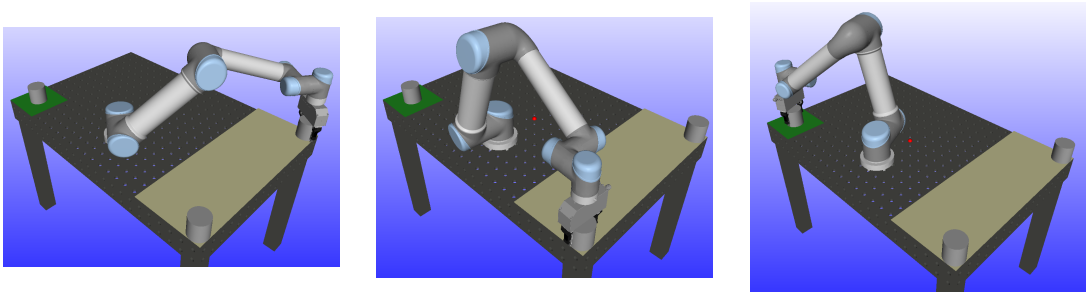
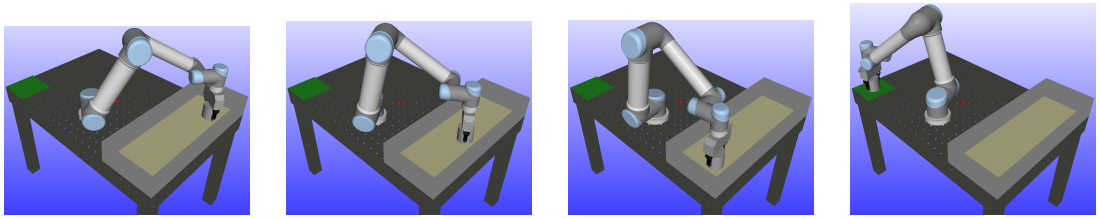


Figure 5: Illustrations of the robot reachabillity at the position $(0.2, 0.0)$.

2.2 Robot Motion Planning

In this section three algorithms for motion planning will be explored. The algorithms are point to point interpolation, point to point interpolation with parabolic blend, and Rapidly-exploring Random Trees Connect, RRT-Connect. The three algorithms will all be analysed on three different configurations, which are shown in Figures 6a, 6b and 6c. The destination of the object is shown in Figure 6d.



(a) Cylinder at $(-0.25, 0.474, 0.15)$ with the corresponding robot arm configuration.

(b) Cylinder at $(0.0, 0.474, 0.15)$ with the corresponding robot arm configuration.

(c) Cylinder at $(0.25, 0.474, 0.15)$ with the corresponding robot arm configuration.

(d) Final position of the cylinders and the corresponding robot arm configuration.

Figure 6: Visualization of the cylinders position on the table and the robot arm configuration to reach the cylinder.

In Figure 6 the picking area is surrounded by walls, this is only done in Rapidly-exploring Random Trees Connect because otherwise, the algorithm would generate a linear path between the source and the destination. Otherwise, the cylinder positions and the workcell is the same when testing the three methods.

2.2.1 Point to Point Interpolation

When doing interpolation between two points in the workspace there are two options — Jointspace interpolation and toolspace interpolation. The simplest version is to use jointspace interpolation since no inverse kinematics are required at each calculated point, whereas the toolspace interpolation requires to solve for the joint configurations at each point along the path using inverse kinematics.

A thing to consider when using jointspace interpolation is that it does not have a well defined path for the position and orientation of the end effector between each point since the linearity of the interpolation is for each joint, and not the position and orientation of the end effector. A different approach is the toolspace interpolation. Since the interpolation happens between points that define the position and orientation of the end effector this gives a well defined path for the end effector while the rest of the robot's configuration is decided by solving inverse kinematics. Another thing to consider when doing linear interpolation is that the trajectory is split up at every defined point. This means the robot will stop at every intersection between two parts of the trajectory. A solution to this problem is presented in Section 2.2.2. For this project, it is chosen to focus on jointspace interpolation.

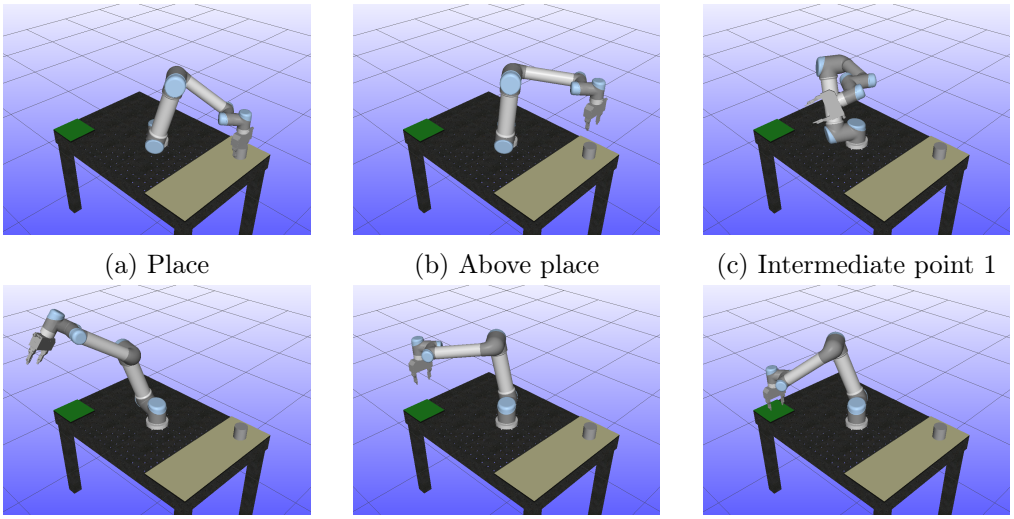
To do linear interpolation between two points the difference is split up into an evenly distributed number of intermediate points. To calculate the fraction of time between each point, resulting in a linear velocity, Equation 1 is used.

$$s(t) = \frac{t - t_{i-1}}{t_i - t_{i-1}} \tag{1}$$

To calculate the next point for the interpolation the difference between the start and end points times the fraction of time is added to the start point. The equation used is from [3] and is seen in Equation 2

$$X(s(t)) = X_{i-1} + (X_i - X_{i-1})s(t) \quad 0 <= s <= 1 \tag{2}$$

The configurations used to interpolate between are seen in Figure 7.

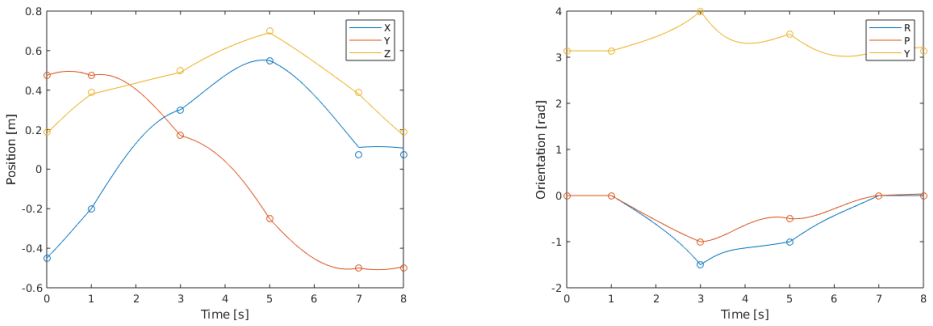


(a) Place (b) Above place (c) Intermediate point 1

(d) Intermediate point 2 (e) Above place (f) Place

Figure 7: Points used to interpolate between.

In Figure 8 the resulting path is seen plotted. The circles indicate the points specified to blend between and the lines are the resulting configurations for the end effector relative to the robot's base frame. Here it is seen that interpolating in jointspace does not result in straight lines for the end effector. If the interpolation was done in toolspace the expected result would be straight lines between the defined configurations.



(a) The position of the end effector in the robot's base frame.

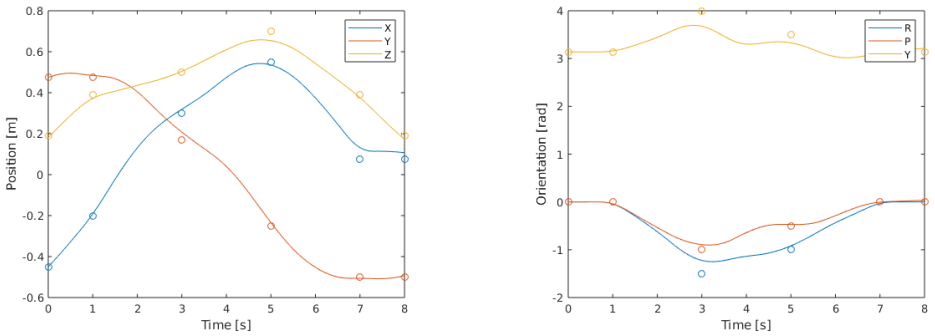
(b) The orientation of the end effector.

Figure 8: The resulting trajectory for the end effector when interpolating in jointspace.

2.2.2 Point to Point Interpolation with Parabolic Blend

As seen in Section 2.2.1 one downside to linear point to point interpolation is the lack of smooth motion at each point. To combat the quick changes in velocity and hard changes in direction at each point a parabolic blend can be implemented around each point. This creates a smooth transition between the two interpolated linear segments, both for velocity and position.

The resulting path using point to point interpolation with parabolic blends in jointspace is seen in Figure 9. As for the path without parabolic blends, the circles indicate the specified configurations to move between. Here it is seen that the segments transition smoothly at every defined point rather than having a sharp turn directly on the point.



(a) The position of the end effector in the robot's base frame.

(b) The orientation of the end effector.

Figure 9: The resulting trajectory for the end effector when interpolating with parabolic blends in jointspace.

The parabolic blend uses Equation 3 from [3].

$$P(t-T, X, v_1, v_2, \tau) = \frac{v_2 - v_1}{4\tau}(t - T + \tau)^2 + v_1(t - T) + X \tag{3}$$

Given three points, v_1 and v_2 are the velocities for the segment between the first and second and second and third points respectively. X is the configuration and T is the time at the middle point. τ is the blend time on either side of the middle point — hence the total blend time ends up as $2 \cdot \tau$ and it is centered around T. To generate a path with multiple points to blend between it was chosen to have τ be $\frac{1}{3}$ of the shortest segment near the point. This way it is avoided that any blends overlap and leaves at least $\frac{1}{3}$ linear path in the middle.

2.2.3 Rapidly-exploring Random Trees Connect

In this section, the motion planning algorithm RRT-Connect is explored for performing a pick and place action in the scene shown in Figure 6. The RRT-Connect algorithm is introduced in the paper [4] for motion planning in high-dimensional spaces. RRT-Connect varies from RRT by growing two trees, one from the source and one from the destination, and the trees grow towards each other by the use of a connect heuristic.

To estimate the best step size for RRT-Connect various step sizes are analysed for the 3 positions in Figure 6. The parameters considered are the planning time, the number of

nodes and the path length. In Figures 10, 11 and 12 the results of the analyses are shown.

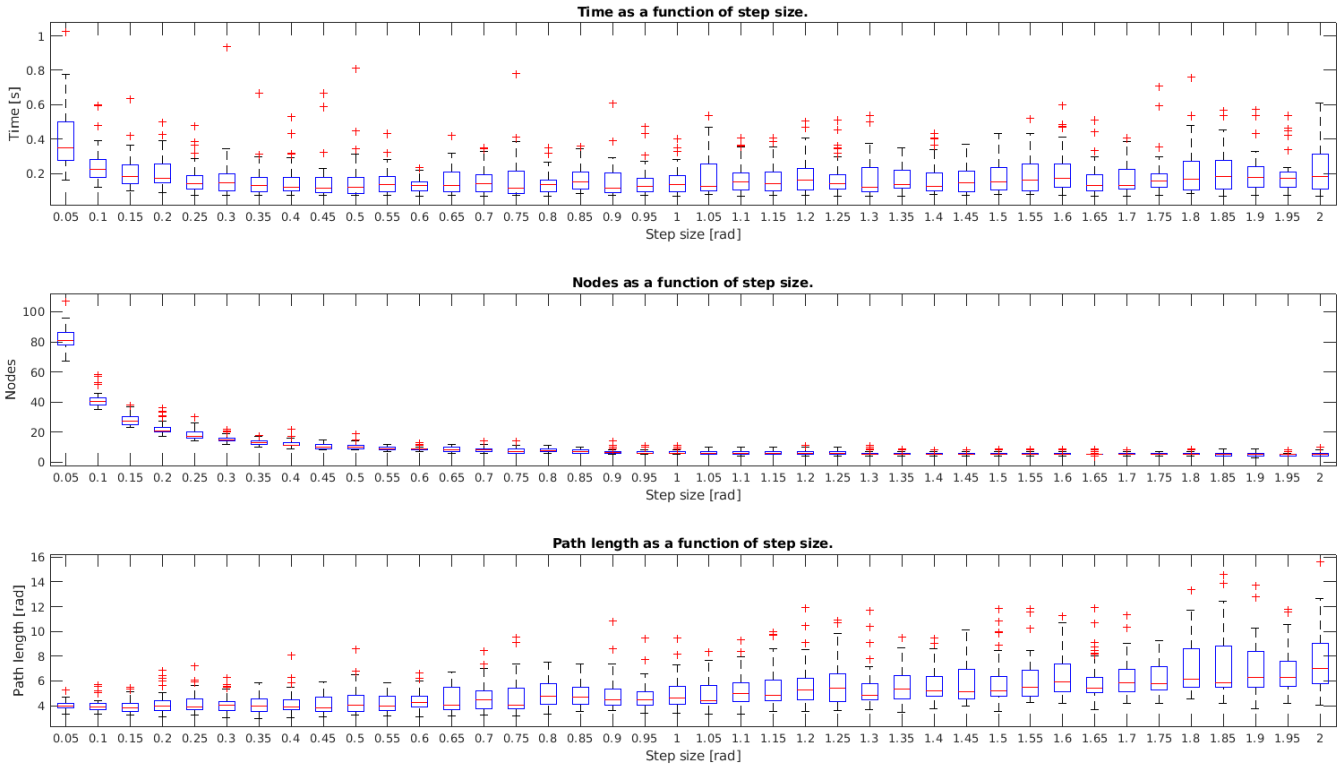


Figure 10: RRT-Connect with the cylinder at (0.25, 0.474, 0.15).

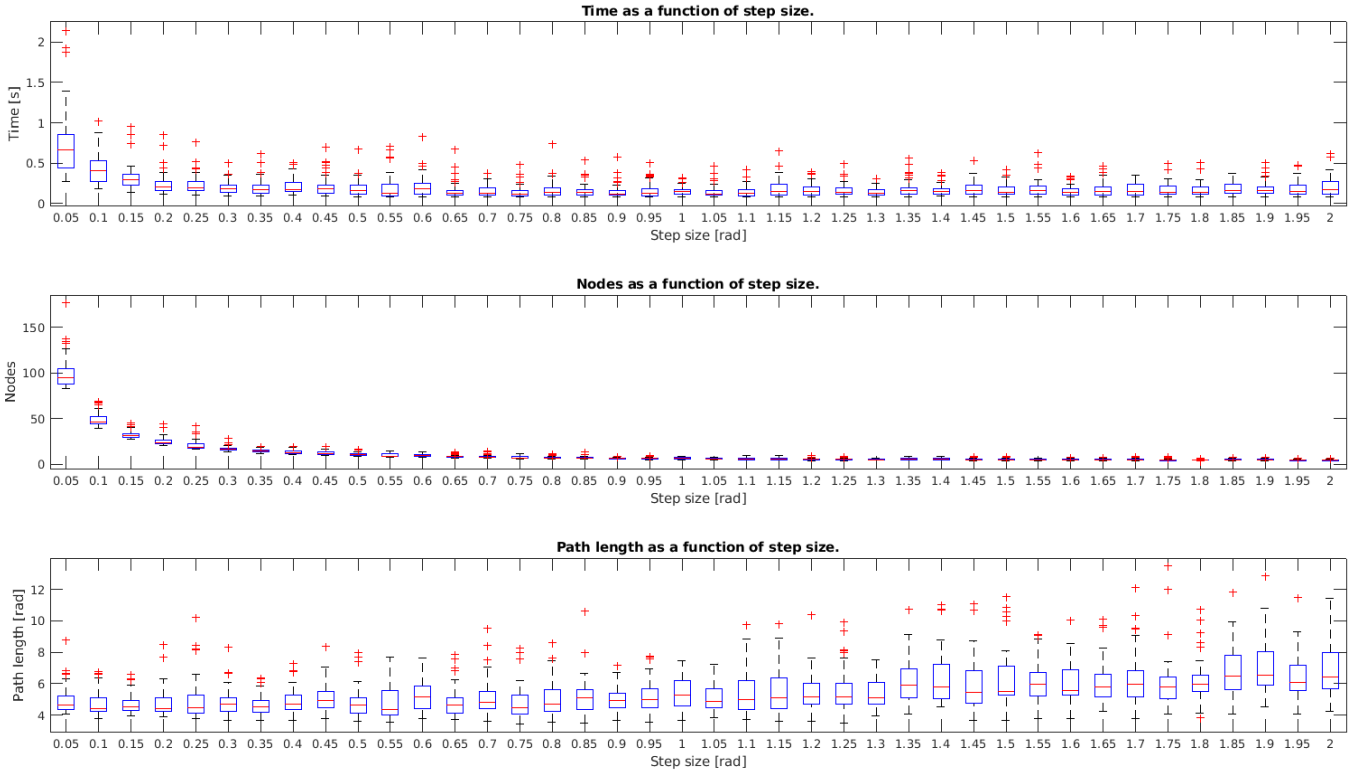


Figure 11: RRT-Connect with the cylinder at (0.0, 0.474, 0.15).

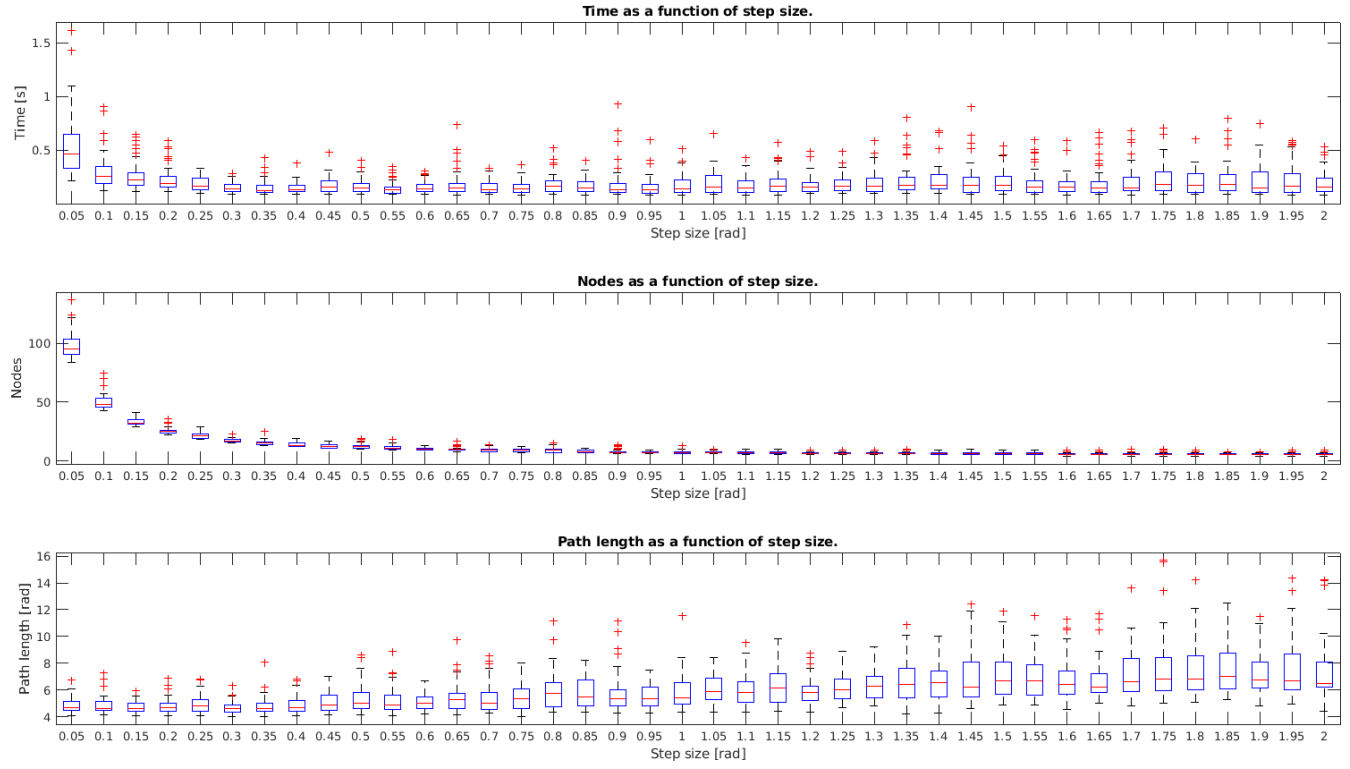


Figure 12: RRT-Connect with the cylinder at (-0.25, 0.474, 0.15).

From the Figures 10, 11 and 12, the step size of RRT-Connect is chosen to 0.6. This step size is deemed acceptable in time, number of nodes and path length for all 3 positions of the cylinder. The results of RRT-Connect with a step size of 0.6 are shown in Table 1.

RRT-Connect	**Planning time [s]**	**Path length [rad]**	**Number of nodes**
Cylinder in position (0.25, 0.474)	0.1307(±0.0387)	4.4025(±0.7935)	8.9(±1.2817)
Cylinder in position (0, 0.474)	0.2126(±0.1304)	5.1726(±0.8953)	10.08(±1.5497)
Cylinder in position (-0.25, 0.474)	0.1578(±0.0537)	5.1016(±0.5578)	10.04(±0.9889)

Table 1: The results of the RRT-Connect with a step size of 0.6

2.2.4 Comparison of the Robot Motion Planning Methods Implemented

To compare the three robot motion planning methods the planning time and the path length of each method is considered. Table 2 shows the planning time and path length for each method with the cylinder at different positions.

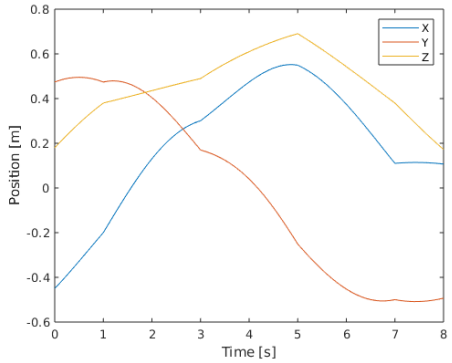
Method — Cylinder in position (0.25, 0.474)	**Planning time [s]**	**Path length [rad]**
Point to Point Interpolation	$2 \cdot 10^{-5}(\pm 1.4142 \cdot 10^{-4})$	$8.1819(\pm 1.7944 \cdot 10^{-15})$
Point to Point Interpolation with Parabolic Blend	$4 \cdot 10^{-5}(\pm 1.9795 \cdot 10^{-4})$	$7.6840(\pm 8.9720 \cdot 10^{-15})$
Rapidly-exploring Random Trees Connect	0.1307 (± 0.0387)	4.4025 (± 0.7935)
Method — Cylinder in position (0.0, 0.474)	**Planning time [s]**	**Path length [rad]**
Point to Point Interpolation	$2 \cdot 10^{-5}(\pm 1.4142 \cdot 10^{-4})$	$8.0742(\pm 7.1776 \cdot 10^{-15})$
Point to Point Interpolation with Parabolic Blend	$6 \cdot 10^{-5}(\pm 2.3990 \cdot 10^{-4})$	$7.5544(\pm 3.5888 \cdot 10^{-15})$
Rapidly-exploring Random Trees Connect	0.2126 (± 0.1304)	5.1726 (± 0.8953)
Method — Cylinder in position (-0.25, 0.474)	**Planning time [s]**	**Path length [rad]**
Point to Point Interpolation	$2 \cdot 10^{-5}(\pm 1.4142 \cdot 10^{-4})$	$8.4105(\pm 7.1776 \cdot 10^{-15})$
Point to Point Interpolation with Parabolic Blend	$6 \cdot 10^{-5}(\pm 2.3990 \cdot 10^{-4})$	$7.8404(\pm 8.9720 \cdot 10^{-15})$
Rapidly-exploring Random Trees Connect	0.1578 (± 0.0537)	5.1016 (± 0.5578)

Table 2: Summary of the planning time and path length for the three implemented methods.

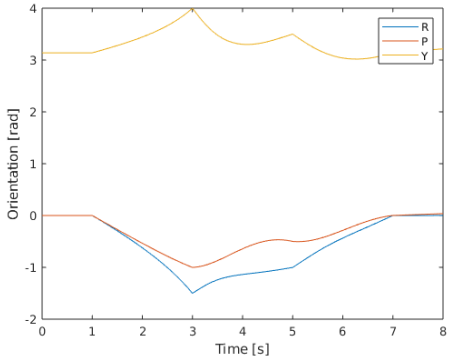
From Table 2, the RRT-Connect method has the longest planning time and the shortest path length. However, the standard deviation is much higher than the other methods, which is because the method is random. The downside of RRT-Connect, when compared to the other methods, is that point to point interpolation with and without parabolic blend has some predefined path configurations. The predefined configurations are protected from singularities by the developer, thus making these methods more secure.

For further comparison of the three methods the TCP displacement is shown in Figure 13, when executing from the object configurations to the goal configurations with the object

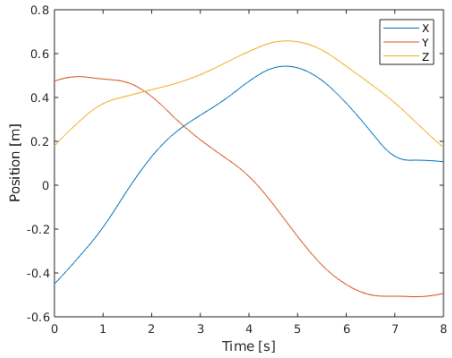
attached to the TCP.



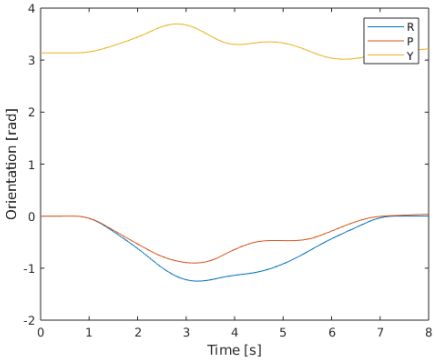
(a) Point to point interpolation x, y and z displacement of the TCP from object to goal.



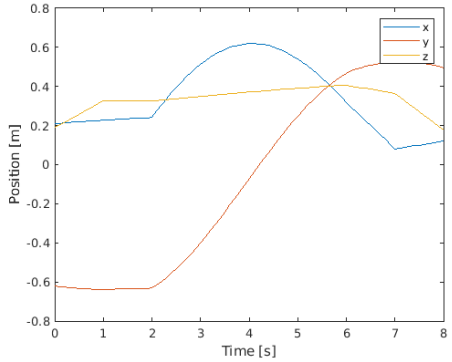
(b) Point to point interpolation roll, pitch and yaw displacement of the TCP from object to goal.



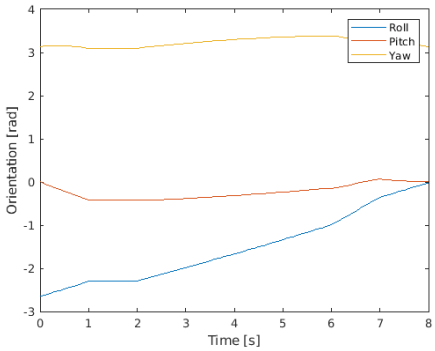
(c) Point to point interpolation with parabolic blend x, y and z displacement of the TCP from object to goal.



(d) Point to point interpolation with parabolic blend roll, pitch and yaw displacement of the TCP from object to goal.



(e) Rapidly-exploring Random Trees Connect x, y and z displacement of the TCP from object to goal.



(f) Rapidly-exploring Random Trees Connect roll, pitch and yaw displacement of the TCP from object to goal.

Figure 13: Comparison of TCP displacement when executing from object at position $(0.25, 0.474)$ to goal with the object attached.

From Figures 13c and 13d, it is seen that the point to point interpolation with parabolic blend moves more smoothly than point to point interpolation, shown in Figures 13a and

13b, which is expected.

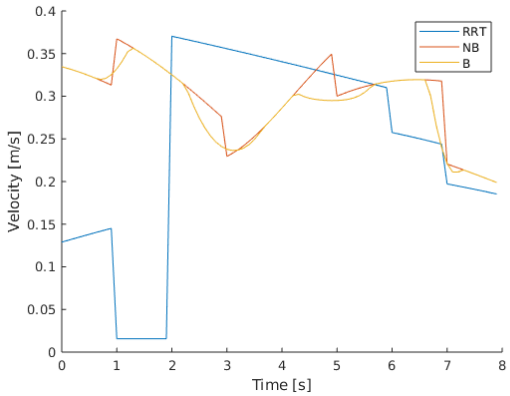


Figure 14: Velocity in 3D space for the tool point. *RRT:* RRT-Connect, *NB:* Point to point interpolation, *B:* Point to point interpolation with parabolic blend.

Figure 14, shows the velocity profile for the tool point of the robot when moving from the object to the goal with the object attached. The velocity profile for point to point interpolation and point to point interpolation with blend are both based on the same set of points and hence give the same profile with the blended trajectory having smooth transitions in velocity. The RRT-Connect velocity profile is based on a path found using RRT-Connect and then linear interpolation between each segment. The reason for the very low velocity between one and two seconds in Figure 14 is that RRT-Connect has found two points relatively close together and the implementation of the linear interpolation, for now, only supports integer values for the time for each segment.

3 Pose Estimation of Objects

In this section, two simple pose estimation methods are outlined to estimate the pose of an object in a scene. The same object is used in both of the methods. In Section 3.1, a method for pose estimation is described which uses a simulated depth sensor in RobWorks. In Section 3.2, a method for position estimation by the use of two cameras in RobWorks is described. The two implemented methods are compared in Section 3.3.

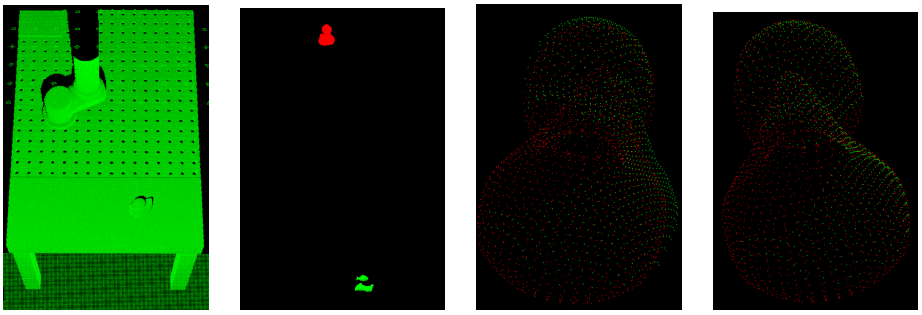
3.1 Method 2: Simulated Depth Sensor

This pose estimation method uses the simulated depth sensor in RobWorks and computes the object pose in the scene by the use of point clouds [5]. The goal is to estimate a pose that has the minimum sum of distances between the object points and the scene points. The pipeline for the simulated depth sensor method is illustrated in Figure 15.

Figure 15: Illustration of the pipeline for the simulated depth sensor method.

The pipeline of the simulated depth sensor, shown in Figure 15, is described further below.

1. A point cloud of the scene is loaded from the simulated depth sensor, this is shown in Figure 16a.
2. The point cloud is processed, thus irrelevant information is discarded. An illustration of the processed point cloud and the object point cloud is shown in Figure 16b, where green points are the input point cloud and the red points are the object point cloud.
3. The object is globally aligned in the scene, which is illustrated in Figure 16c.
4. The object is locally aligned in the scene, this is illustrated in Figure 16d.
5. The estimated pose is found.



(a) Input cloud from depth sensor. (b) Processed input cloud. (c) After global alignment. (d) After local alignment.

Figure 16: Example of the simulated depth sensor pipeline. Green points are scene points and red points are object points.

3.1.1 Point Cloud Processing

Before the pose of the object is found the scene is processed, thereby making the alignment of the object in the scene easier both computationally and pose estimation wise. Figure 17 shows the implemented point cloud processing pipeline.

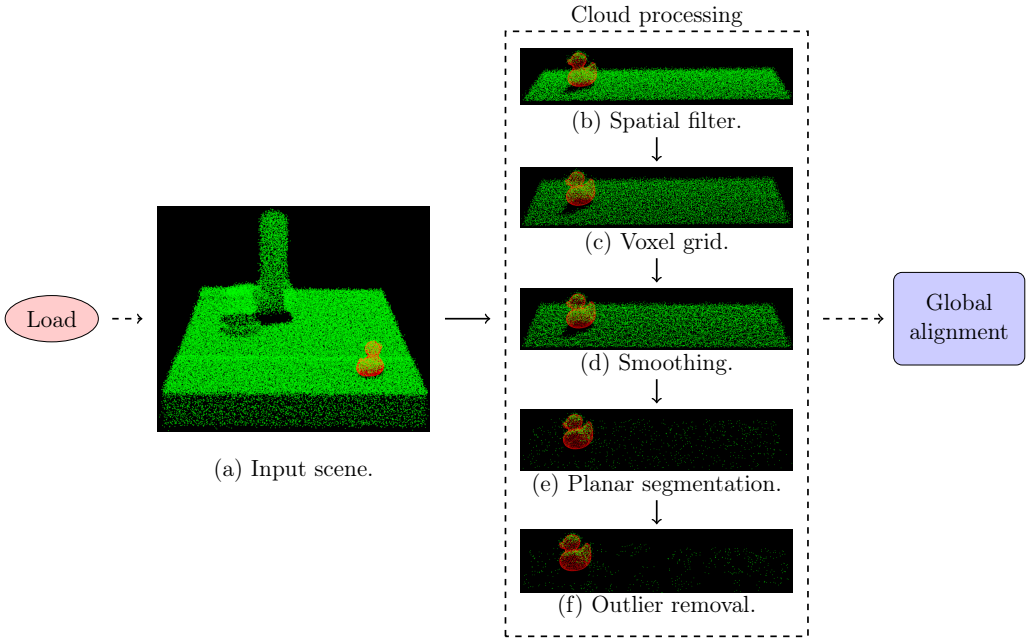


Figure 17: Illustration of the point cloud processing pipeline with noise from a normal distribution with a standard deviation of 0.005. Green points are the input points from the simulated depth sensor, and red points are the true pose of the object.

In Figure 17a, the scene is loaded from the scanner and noise with a standard deviation of 0.005 is added to the scene. Then, in Figure 17b, the scene is passed through a spatial filter that removes points outside intervals in x, y, and z. This filtering is based on the scene

and the placement of the scanner, thus everything outside of the picking area is removed. Afterwards in Figure 17c, the scene point cloud is downsampled using a voxelized grid approach with a leaf size of 0.005 meter. Then in Figure 17d, the scene is resampled by moving least squares with a polynomial order of 2 and a sphere search radius of 0.01 meter. In Figure 17e, a simple plane segmentation is performed which removes points in the point cloud that support a plane model. The distance, which determines how close a point must be to be considered an inlier of the plane, is set to 0.01 meter. Lastly in Figure 17e outliers are removed, by the use of statistics of the whole point cloud to determine which points are too far from the other points. The implemented outlier removal looks at 100 neighbors for each point, and if the distance is larger than 0.1 of the mean distance to the query point the point will be removed. After the filter process, the final input point cloud is passed on to the global alignment block, which is described below.

3.1.2 Global Alignment

This method finds the pose of a 3D object, which brings the object into optimal alignment in a given 3D scene. The pipeline of global alignment is illustrated in Figure 18.

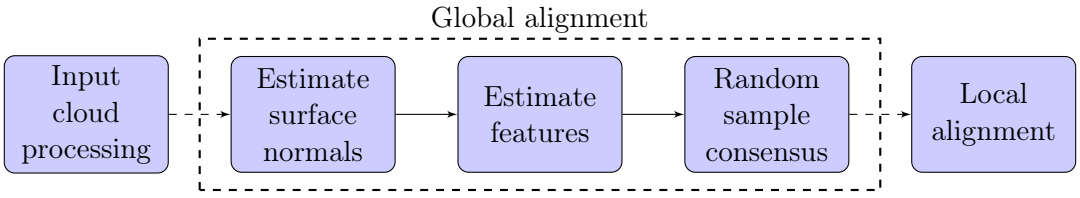
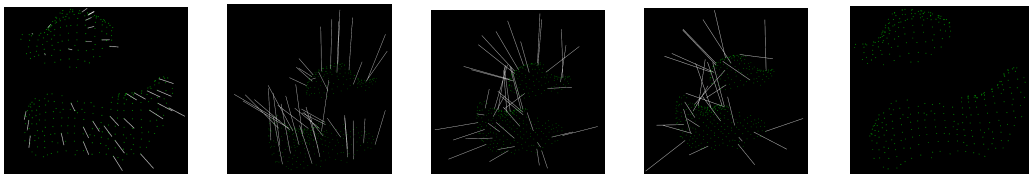


Figure 18: Illustration of the global alignment pipeline.

Estimate surface normals:

The surface normal is computed as the normal on the plane fitted to the points within a certain radius. The surface normals are used for computing the feature descriptors used for matching between the object and scene. In Figure 19, the estimated surface normals with different radii are shown. The radius must be larger than the voxel grid leaf size, e. g. in Figure 19e no surface normals are found when the radius is 0.001 meter, however the radius must not be too big either as illustrated in Figure 19a. Through several observations and experiments the radius was chosen to 0.01 meter.



(a) Estimated surface normals with a radius of 0.1 meter.

(b) Estimated surface normals with a radius of 0.05 meter.

(c) Estimated surface normals with a radius of 0.01 meter.

(d) Estimated surface normals with a radius of 0.005 meter.

(e) Estimated surface normals with a radius of 0.001 meter.

Figure 19: Illustration of estimated surface normals of the scene with different radii after filtering.

Estimate Features Descriptor:

A very important element in pose estimation is to find the correct corresponding points between the object and scene, thus to match points a local feature descriptor is used. The used feature descriptor method is Fast Point Feature Histogram, FPFH, which is introduced in [6] and also used in [7]. Based on the results in [6] and [7] it was chosen to use FPFH. FPFH constructs a 33-dimensional histogram by:

1. Finding all oriented points in a spherical neighborhood of a certain radius around each point.
2. Computing relative angles using the estimated surface normals from before, and the direction vector from the source point to each neighbor point.

The radius used for the neighbor search must be bigger or equal to the radius used for estimating the surface normals. The implemented radius is set to 0.01 meter, which was found to be reasonable. Figure 20 shows the feature matches between the input point cloud, the green points, and the object point cloud, the red points.

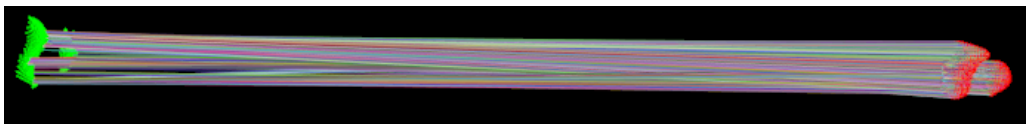


Figure 20: Illustrations of the feature matching between the input point cloud, the green points, and the object, the red points. The image is rotated 90 degrees to the right.

Random Sample Consensus:

To estimate the transformation, T, which transforms the object points, p, into the scene points, q, the Kabsch algorithm [8] is used to solve the equation $q = T \cdot p$. However, because of the uncertainty of whether the found correspondences are the true correspondences between the object and the scene a RANdom SAmple Consensus [9], RANSAC, method

is implemented. The implemented RANSAC method is based on the paper [7], which outlines a simple and robust estimation algorithm that performs 15 times faster than standard RANSAC. The steps of the algorithm are:

1. Find 3 random object points and their correspondences in the scene, by nearest neighbor matching of the feature descriptors.
2. Calculate the ratio between the edge lengths of the polygons formed by the 3 points on both the object and the scene. If the similarity is below a certain threshold go to step 1.
3. Estimate the transformation using the 3 point correspondences from step 1, and apply it to the object.
4. Save the transformation if the number of inliers is the highest so far and above the required number of inlier points.
5. Stop the algorithm after a certain number of iterations otherwise go to step 1.

The implemented RANSAC method samples 3 points in step 1, which is the minimum requirement for 6 Degrees of Freedom, DoF. The inlier fraction of inlier points required for accepting a transformation is set to 0.05 as in [7], this was observed to have the best performance. The similarity between the polygon edges is set to 0.9, thus creating a quite strict rejector but making the execution time of the RANSAC much faster. The maximum distance between two corresponding points in object and scene is set to 0.0075 meter, which is based on the voxel grid leaf size and experience. The maximum number of iterations, k, given a desired success probability, $p = 0.99$, an expected inlier fraction, $w = 0.05$, and sample points, $n = 3$ is [9]:

$$Iterations = \frac{log(1-p)}{log(1-w^n)} = \frac{log(1-0.99)}{log(1-0.05^3)} = 36839 \tag{4}$$

However, to ensure a more precise pose estimation the iterations are set to 80000.

3.1.3 Local Alignment

The local alignment pose estimation finds the optimal alignment of two 3D models, which are almost aligned. The implemented local alignment is based on the algorithm Iterative Closest Point, ICP, which is introduced in [10]. The steps implemented are:

1. Find nearest neighbors between the transformed object and the scene, and threshold neighbors by euclidean distance.
2. Estimate the transformation using the neighbor correspondences.
3. Apply the transformation to the object points.
4. Stop the algorithm after a certain number of iterations otherwise go to step 1.

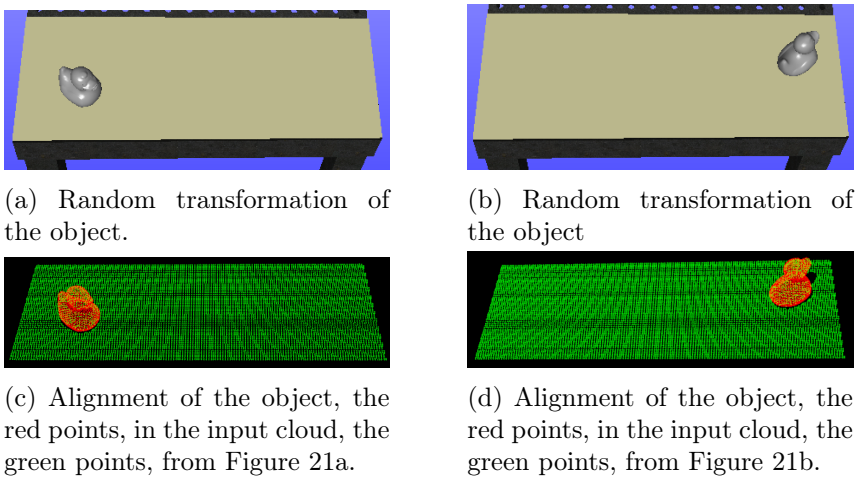
The implemented ICP method is set to run for 500 iterations, which deemed acceptable in execution time versus pose estimation precision. Furthermore, the euclidean distance threshold for accepting neighbors was set to 0.0001. Thereby, ensuring the method not to use point correspondences which are most likely not true correspondences.

3.1.4 Evaluation of the Method

To evaluate the method 30 random positions with a random rotation around the z-axis are generated, and noise from a normal distribution is added to the simulated depth sensor input point cloud. The real transformation is generated from RobWorks, which is described as $T_{Table}^{Realpose}$. However, the estimated transformation is described as $T_{Scanner}^{Estimatedpose}$, thus to compare the two transformations the following equation is used:

$$\begin{aligned} T_{World}^{Table} \cdot T_{Table}^{Real\ pose} &= T_{World}^{Scanner} \cdot T_{Scanner}^{Estimated\ pose} \Rightarrow \\ T_{Table}^{Real\ pose} &= \left(T_{World}^{Table}\right)^{-1} \cdot T_{World}^{Scanner} \cdot T_{Scanner}^{Estimated\ pose} \end{aligned} \tag{5}$$

Two illustrations of the random transformation of the object and the found alignment is shown in Figure 21.



(a) Random transformation of the object.

(b) Random transformation of the object

(c) Alignment of the object, the red points, in the input cloud, the green points, from Figure 21a.

(d) Alignment of the object, the red points, in the input cloud, the green points, from Figure 21b.

Figure 21: Illustration of two of the random transformations of the object, which are used for the evaluation of method 2.

The method is evaluated on three parameters: execution time, difference in angle between real transformation and estimated transformation and the euclidean distance between the real transformation and the estimated transformation. To compare the rotations the intrinsic notion of distance, which is introduced in [11], is used. In Figure 22 the results from the evaluation are shown.

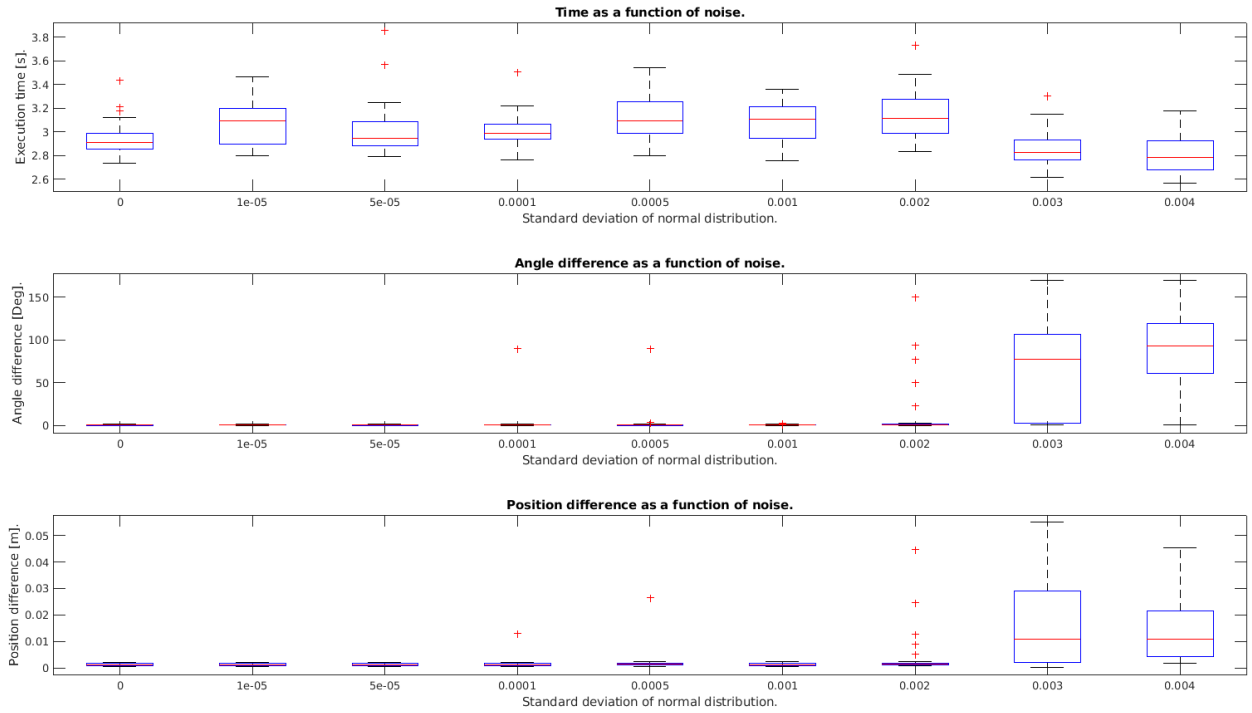


Figure 22: Boxplot of the evaluation of method 2. The top figure shows the execution time versus the added noise. The middle figure shows the difference in angle versus the added noise. The bottom figure shows the difference in position versus the added noise.

From Figure 22, it can be seen that the method performs acceptably until a noise with a standard deviation of 0.002. The average execution time for which the method is deemed acceptable is 3.043($\pm$1.865) seconds, the average intrinsic notion distance is 1.708($\pm$9.380) degrees and the average euclidean distance between the real position and the estimated position is 0.0015($\pm$0.00211) meter.

The implemented method has a limitation in the choice of object because validation of the alignment is based on the number of inliers. For example, if the object had been a box the number of inliers when the box is aligned on a table, would be high, thus the method would have a hard time estimation the true pose of the box. However, this was compensated by the use of a non-uniform object, a rubber duck [12] which can be seen in Figure 21.

3.2 Method 3: Sparse Stereo

Sparse stereo is used to find points in 3D space corresponding to sets of matched points on the object. The images are obtained with the two simulated cameras placed in the given workcell in RobWorks. Figure 23 shows the steps for retrieving 3D points for a set of images.

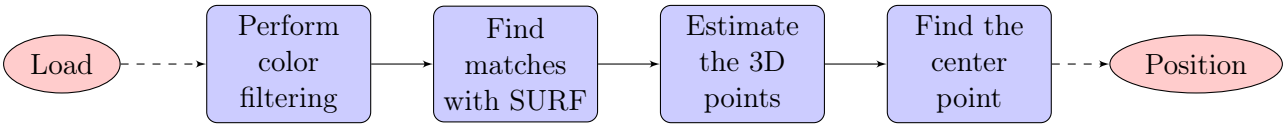
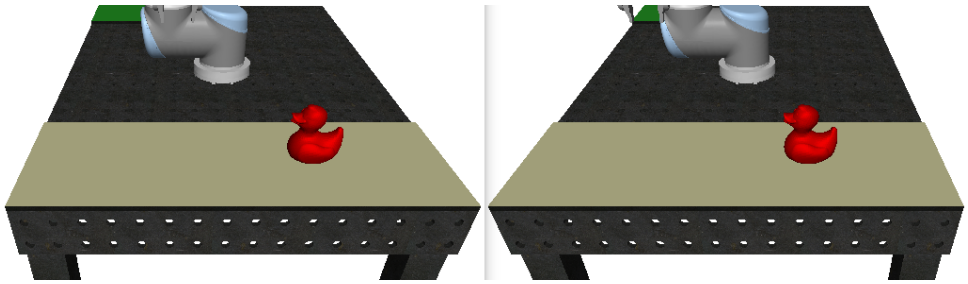
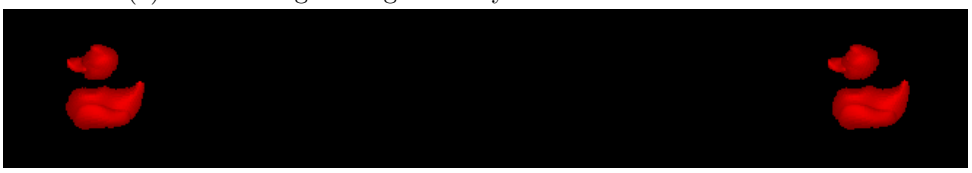


Figure 23: Illustration of the pipeline for the sparse stereo method.

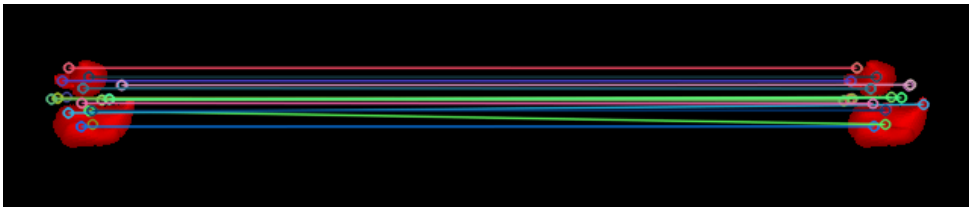
To make sure the object is isolated in the images the object is assumed to have a certain color. In this case, the object is red for a high contrast to the rest of the scene. This way, as seen in Figure 24b, the object can be isolated by filtering the color in the images. To find features in the images the Speeded-Up Robust Features, SURF, algorithm was implemented using OpenCV [13]. The features matched are seen in Figure 24c.



(a) Left and right image directly from the simulated cameras.



(b) Left and right image after color filtering.



(c) Left and right image with matched points.

Figure 24: The process for finding matching points on the object.

After finding matched features in the two images, triangulation is used to find the estimated position in 3D space for each point. Then the average distance to all other points is calculated for each point, and the point with the lowest value is the point chosen to represent the detected object's position. With this method, the points found are on the surface of the object towards the cameras. Hence, it is not the most well suited for irregular shapes if desiring a complete pose estimation. For the rubber duck used in this example the default error between the detected point and center of the object changes with the

orientation of the object.

3.2.1 Evaluation of the Method

The robustness of the method is tested by adding Gaussian noise, with a mean of 0 and varying standard deviation, to the images. The standard deviation was tested from 0 to 25 in steps of 0.5. The random positions of the object are the same as in 3.1.4. The result is seen in Figure 25.

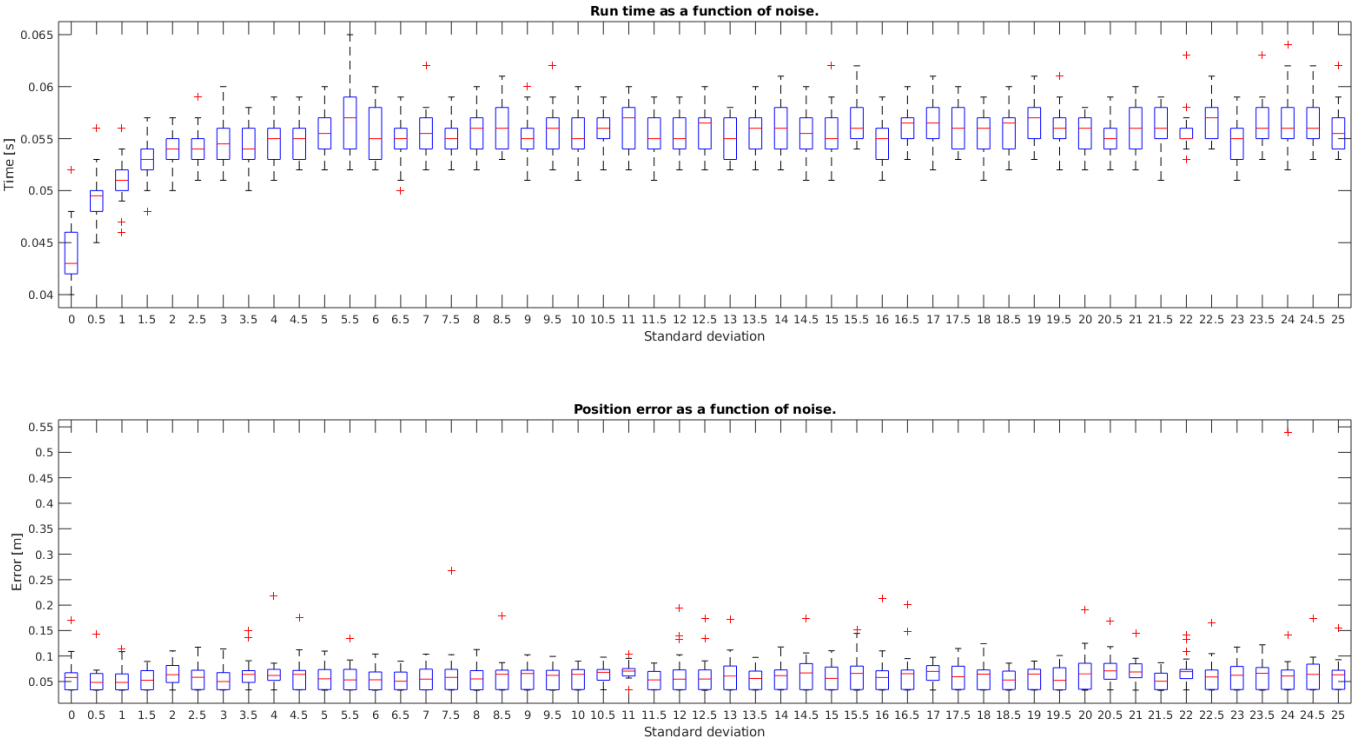


Figure 25: Execution time and error in estimated position with increasing levels of noise added to the picture.

From Figure 25, it can be seen that the method is very robust and can manage noise from a normal distribution with a standard deviation of up to 25 without failing. However, the euclidean distance from the real position to the estimated position has an average error of 0.0552($\pm$0.0029) meters and average execution time of 0.0606($\pm$0.0287) seconds. The position error occurs because the found 3D point is on the surface of the duck, which has a non-uniform structure, and the real position is at the center of the duck. This method does therefore not suit non-uniform objects and would perform better if the object was uniform, e.g. a ball.

3.3 Comparison of the Pose Estimation Methods Implemented

To compare the two implemented pose estimation methods the execution time, the angle error between the real rotation and the estimated rotation and the euclidean distance between the real position and the estimated position are considered. Furthermore, the robustness of the method is also considered, meaning how much noise from a normal distribution the method can tolerate. Table 3 shows a summary of the results of the pose estimation methods.

	Method 2: Simulated depth sensor	**Method 3: Sparse stereo**
Robustness [std. deviation]	0.001	25
Execution time [s]	3.043(±1.865)	0.0606(±0.0287)
Rotation error [Deg]	1.708(±9.380)	÷
Position error [m]	0.0015(±0.00211)	0.0552(±0.0029)

Table 3: Summary of the implemented pose estimation methods.

Table 3 shows the strengths and weaknesses of the two pose estimation methods.

The simulated depth sensor method has a small position error and angle error. However, it can only tolerate noise from a normal distribution with a standard deviation of max 0.001, and the execution time is higher than the sparse stereo method.

The sparse stereo method does not calculate a rotation, thus if the object to grasp has to be grasped from a certain angle this method would not work. The sparse stereo method would work well for a uniform object, such as a ball. Furthermore, the method can sustain more noise than the simulated depth sensor method.

Thus, the sparse stereo method would be chosen for simple uniform objects and the simulated depth sensor method for non-uniform objects.

4 Combining Pose Estimation with Pick and Place Execution

In this section, a combination of the developed pose estimation method and robot motion planning method is described.

Based on the results in section 2.2.4, the robot motion planning method: point to point interpolation with parabolic blend, was deemed most suitable for this task. This is because the method has some predefined points, which are constructed so that the robot avoids singularities. Even though the Rapidly-exploring Random Trees Connect method calculated a shorter path, the path was generated randomly which could potentially lead the robot close to singularities.

Based on the results in section 3.3, the simulated depth sensor method is integrated into the combined system. The simulated depth sensor showed the most precise results when considering rotation and position error.

The pipeline for the combination is illustrated in figure 26.

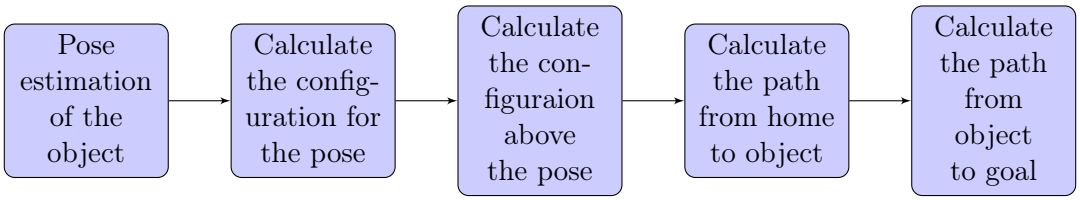
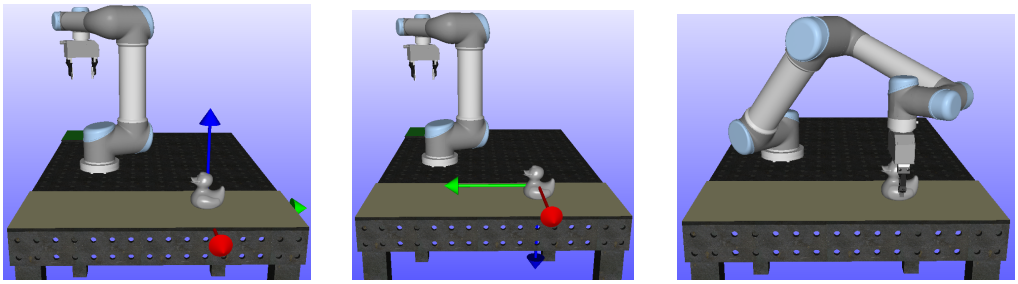


Figure 26: Illustration of the pipeline for the combination of pose estimation and pick and place execution.

In Figure 27a, the coordinate frame for the object to be picked is shown. The z-axis of the object frame points in the opposite direction to the TCP, thus when calculating the configuration of the estimated pose it is rotated 180 degrees around the x-axis. Furthermore, the coordinate frame of the object is at the bottom, thereby the TCP comes in collision with the object when trying to grasp it. Therefore, the z position is adjusted so the TCP does not collide with the object when grasping is performed. The grasp target coordinate frame is shown in figure 27b, which makes the robot able to grasp the object as shown in figure 27c. The object used is the duck [12] that is also used in Section 3 for the pose estimation.



(a) The coordinate frame for the object.

(b) The coordinate frame for the grasp target of object.

(c) The robot grasping the object.

Figure 27: Illustration of how the object is grasped.

A video of the combined pose estimation and pick and place can be seen at [14], which shows the task completed at three different object locations.

5 Discussion

In Section 2.2.3, the robot motion planning algorithm RRT-Connect is outlined. This algorithm could be optimized by adding post processing, where each node in the path is checked for whether or not it can be skipped. Thereby, eliminating unnecessary nodes and optimizing the generated path.

(*Mathias*)

In Section 3.1, the scene only contains one object when performing pose estimation. However, if more objects were in the scene the method could be extended by doing euclidean clustering. Thus, dividing the point cloud into clusters and removing the clusters which have low similarity to the object cluster. Then, for each remaining cluster, the best transformation is found for the object and the transformation with the most inliers is considered correct.

(*Mathias*)

In Section 3.1, the simulated depth sensor is used to obtain the scene point cloud. However, only one sensor is used, thus to optimize the number of points for the object a second depth sensor could be implemented in the scene. To ensure that the two input clouds are aligned ICP could be used, which aligns to almost aligned point clouds.

(*Mathias*)

In Section 2.2.1 the points used to interpolate between could easily be optimized simply by observation if efficiency for the task is the goal. The way that the time intervals for each interpolated segment are calculated could also be optimized to at least accept floating point time intervals. To optimize the velocity profile for the interpolation a ramp could be added at the beginning and end of the trajectory.

(*Bjarke*)

In Section 3.2, the sparse stereo method is outlined. One of the downsides to the method is that it does not find an orientation for the object. This could be improved either by estimating a pose based on aligning multiple of the detected matched points with a known model of the object or by using it only for uniform known objects. Furthermore, by implementing the RANSAC method described

in 3.1.2 the uncertainty of the matching features from SURF could have had a smaller impact on the estimated pose of the object.

(*Bjarke*)

6 Conclusion

Throughout the report, methods for work cell design, robot motion planning and pose estimation of an object were explored.

The best placement of the robot base position was analyzed, in Section 2.1, by the reachability of the robot when grasping from the top and side. The analysis showed that the side closest to the pick area of the table had most collision free solutions for each grasp position, thus the robot base was placed at (0.2, 0.0) on the table.

Three different robot motion planning algorithms were explored to perform a pick and place execution. It was shown that Rapidly-exploring Random Trees Connect generated the shortest path, but had the longest planning time. Furthermore, when considering singularities point to point interpolation with and without parabolic blend performed better than Rapidly-exploring Random Trees Connect. Point to point interpolation with parabolic blend had a shorter path than point to point interpolation, with only a small difference in planning time.

Two methods for pose estimation of an object were presented, one which used a simulated depth sensor and one that used two aligned cameras. It was shown that the two implemented methods had different strong sides, the simulated depth sensor method was good at pose estimation with both rotation and position, whereas the sparse stereo method was not as good at pose estimation and could only estimate the position. However, the sparse stereo method was more robust and had a shorter execution time.

A combination of the pose estimation and pick and place execution was created, which was shown in [14]. The combination used the simulated depth sensor method for pose estimation since it showed more precise pose estimations. Furthermore, for pick and place the combination used point to point interpolation with parabolic blend, thus ensuring that singularities were avoided. The final combination can estimate a pose of the object in the picking area, pick the object from the picking area and place the object in the place area.

References

[1] Lars-Peter Ellekilde and Jimmy A Jorgensen. "Robwork: A flexible toolbox for robotics research and education". In: *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. VDE. 2010, pp. 1–7.

[2] Mathias Emil Nielsen and Bjarke Engsig Larsen. *rovi project*. 2019. URL: `https://github.com/Masle16/rovi_project` (visited on 12/12/2019).

[3] Aljaz Kramberger Henrik Gordon Petersen. "Robotics and Computer Vision(RoVi) - Lecture Notes". Lecture notes for the class Robotics and Computer Vision at University of Southern Denmark. 2019.

[4] J. J. Kuffner and S. M. LaValle. "RRT-connect: An efficient approach to single-query path planning". In: 2 (Apr. 2000), 995–1001 vol.2. DOI: `10.1109/ROBOT.2000.844730`.

[5] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[6] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast point feature histograms (FPFH) for 3D registration". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3212–3217.

[7] Anders Glent Buch et al. "Pose estimation using local structure-specific shape and appearance context". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 2080–2087.

[8] Wolfgang Kabsch. "A solution for the best rotation to relate two sets of vectors". In: *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 32.5 (1976), pp. 922–923.

[9] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: `10.1145/358669.358692`. URL: `http://doi.acm.org/10.1145/358669.358692`.

[10] Paul J. Besl and Neil D. McKay. "Method for registration of 3-D shapes". In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Ed. by Paul S. Schenker. Vol. 1611. International Society for Optics and Photonics. SPIE, 1992, pp. 586–606. DOI: `10.1117/12.57955`. URL: `https://doi.org/10.1117/12.57955`.

[11] Du Q Huynh. "Metrics for 3D rotations: Comparison and analysis". In: *Journal of Mathematical Imaging and Vision* 35.2 (2009), pp. 155–164.

[12] Willie. *Rubber Duck*. 2013. URL: `https://www.thingiverse.com/thing:139894` (visited on 12/12/2019).

[13] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[14] Mathias Emil Slettemark-Nielsen. *Illustration of the Combination of Pose Estimation with Pick and Place Execution.* 2019. URL: `https://raw.githubusercontent.com/Masle16/rovi_project/master/combination/combination_video.webm` (visited on 12/12/2019).