University of Southern Denmark

MSc. in Engineering Robot Systems

# RoVi Final Project

Robotics and Computer vision


**Authors:**

Steffen Waage Jensen *stjen15@student.sdu.dk*

Emil Reventlov Husted *emhus17@student.sdu.dk*

**Supervisors:**

Aljaz Kramberger *alk@mmmi.sdu.dk*

Dirk Kraft *kraft@mmmi.sdu.dk*

**Deadline: 23-12-2022**

*Characters: 5562 words ∼ 21 pages*

# Table of Content

# 1 Introduction

In this report different methods will be presented on how to perform pick and place in a simulated environment using modern tools and techniques. The simulation is created with RobWork Studio [2], to simulate the system and evaluate on the methods. This project was created in close collaboration in the course Robotics and Vision provided at the University of Southern Denmark. Acknowledgment should be given to all the teams members that has worked on implementation to RobWork, as well as the Teachers and Exercise providers for the course in Robotics and Vision. Different parts of the code are derived mainly from solutions of the lectures and theories provide on by the lectures, any other Acknowledgment of code use can also be found in the code [4] as well on external appendix A. The report will focus on three main components, a robotics part, vision part, and the combination of both developed in a Plugin for the RobWork-Sim package. Pick and Place tasks are common in many productions and therefor a good use-case to real applications on the robotics with combination of vision based estimations, that can make systems more flexible.
Generally this report presents five different methods. The first three methods focus on moving the object from the pick location to the place location, while the last two methods focus on detecting the object and make pose estimation using computer vision. Finally, a combination of these methods is presented to perform a complete pick and place operation. The code used to present the results and shown in the report can be found at the provided repository and the reference section.

## 1.1 Division of work

**Steffen Waage Jensen - stjen15:**

1. **Implementation:**
   - Workspace w. obstacles
   - Parabolic blend
   - RRT
   - Method 1: Dense Stereo
2. **Report sections:**
   - 2.1.2 Workspace w. obstacles
   - 2.4 P2P interpolation with parabolic blend
   - 2.5 RRT
   - 3.1 Method 1: Dense Stereo
   - 5. Discussion (Steffen)
   - 6. Conclusion

**Emil Reventlov Husted - emhus17:**

1. **Implementation:**
   - Workspace
   - Reachability Analysis
   - P2P Interpolation
   - Method 3: Sparse Stereo
   - Integration
2. **Report sections:**
   - 1. Introduction
   - 2.1.1 Workspace
   - 2.2 Reachability analysis
   - 2.3 P2P interpolation
   - 3.2 Method 3: Sparse Stereo
   - 4. System integration
   - 5. Discussion (Emil)

# 2 Robotics - Pick and place

In this section all the methods for the robotics part are introduced. The methods are the workspace environment both with and without obstacle, a reachability analysis of the robot, Point to point interpolation both with and without parabolic blend and rapidly-exploring random tree connect method (RRT).

## 2.1 Provided workcell

### 2.1.1 Workspace

At the beginning of the project a sample plugin was provided where a workcell covering the complete environment for the simulation of the project. The workspace is build upon the RobWork core [2], and have a number of elements used to visualizing. The objects that was chosen to complete the methods was a cylinder, ball and different arbitrary objects. The robot manipulator used was a Universal Robot (UR5) [1] placed on a table, where there was included an initial pick and place area. To grip the object a Weiss Robotics WSG50 gripper was included. The preparation prior analysis included attachment of the gripper at the right orientation and position for the robot tool center point (TCP).

### 2.1.2 Workspace w. obstacles

The setup of the work cell with obstacles is similar to the Workspace described above, except for four large spheres flanking the sides of the table. It was modified as little as possible to keep the standard setup for the testing and development of the RRT, as this cell would not use cameras for detection. Minimal modifications were performed in the form of the gripper WSG50, which was attached to the UR5 robot arm, and the robot was placed on the found reachability point so it would match the sample workspace. Finally, the objects were removed from the cell as only the path planning of the RRT was performed on this cell. Though, during the testing of the RRT, changes to the spheres where done but placed back to match the initially given cell.

## 2.2 Reachability analysis

To find the best robot base position a reachability analysis is needed, to correctly point the best location to manipulating the task objects. From the project description, it is know that the grab object have a certain position, so the work in this project were done on four selected target positions, three on the pick area and one on the place. The task was to analyze two different gripping positions of an object, the implementation are used to move the robot base to validate if the robot has any collision free solutions.

### 2.2.1 Reachability WorkCell

For this analysis the cylinder object were used for several reasons. The example code provided from the Robotics course exercises, created by Kasper Høj Lorenzen, provided a starting point for the analysis. A cylinder was used with the main reason that the movement around the object in combination with collision

checking and the completion of exercise 3 shows good results. The cylinder object also represent a good approximation to a real world scenario to a good shape for pick an place example tasks, and simplistic to calculate because the shape of a cylinder is a fairly simple model. The Workspace can be seen in figure 1, where it shows the cylinder object locations. The location of the cylinder and the movement of that follows the number sequence, seen on figure 1 to the right.

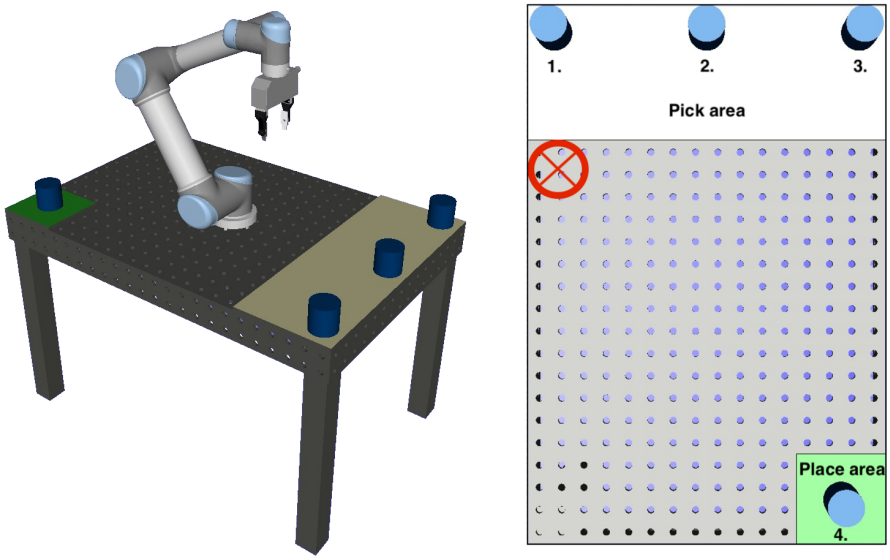



Figure 1: A top view of the workspace for the reachability analysis. 1. First pick reference name *right*, 2. second pick reference name *mid*, 3. third pick reference name *left* and 4. place with reference name *place*

The analysis is starting with reaching the cylinder at the right top corner in the pick area on the table. The robot beginning from the right top corner outside the pick area and base position is then swifted from left to right and then follows a serpentine pattern down towards the place area. A red bullseye shows the starting position and can be seen on figure 1. It can also be seen that this analysis were placing the objects near the corners, out of reach for the robot. The assumption was that the result would be more relevant when the longest distance was chosen. A reference to the code can be found on appendix A, and the code introduce the collision free method to check for any collision with the object around the robot. In this project the avoidance of singularities is essential, and any collision against any obstacle of the robot is excluded from the data pooling. In general it is a subject that it is important to avoid due to loose of accuracy and stability.

### 2.2.2 Side grip analysis

The first of the grip position were the side grip, where the robot loops though different base positions and reaching out for the object and rotating 360 [deg] around the cylinder here for the side. The data has been normalized and correspond to number of solutions when rotating every 1 [deg].

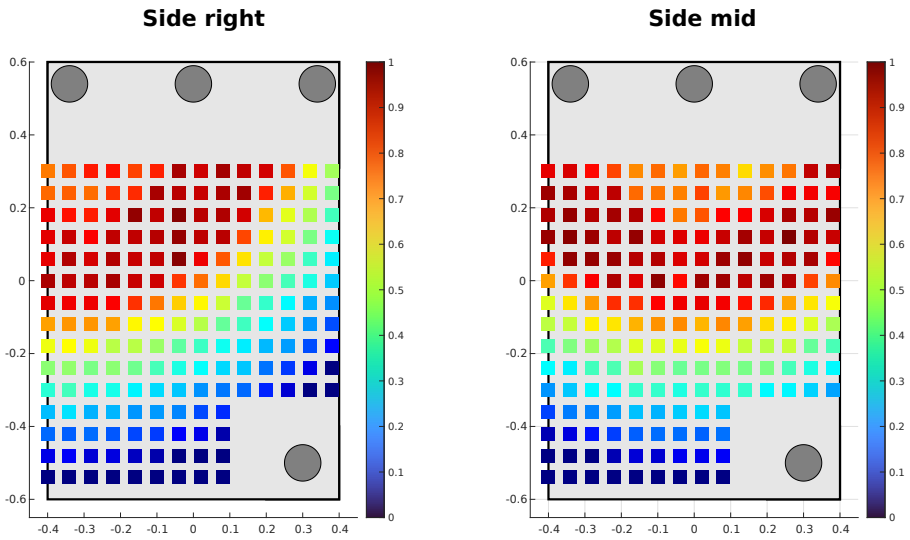



Figure 2: Reachbility analysis of the side grip from the right and mid cylinder

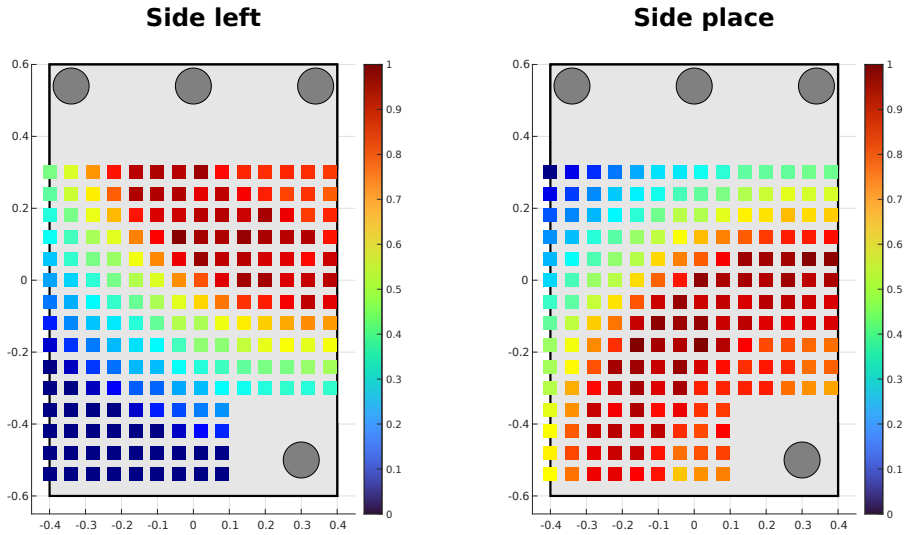
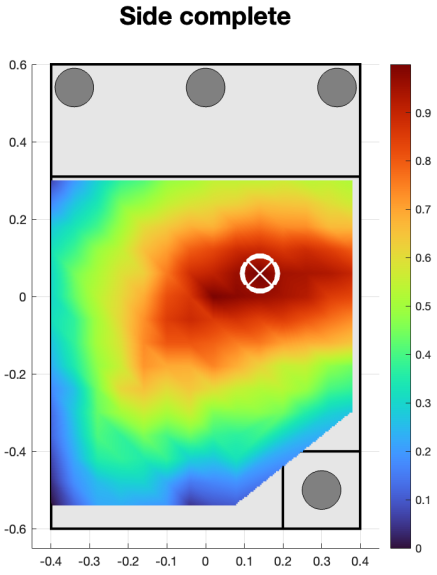
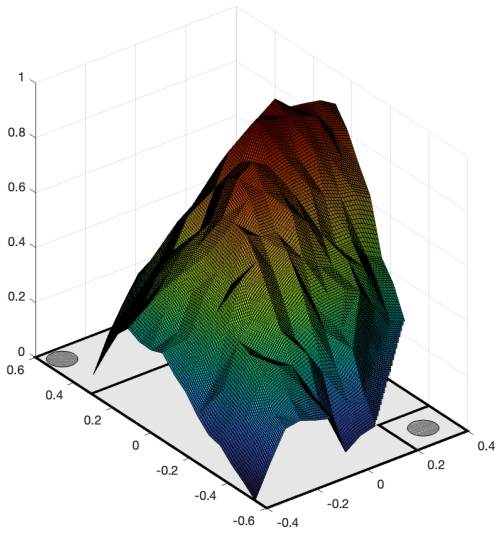



Figure 3: Reachbility analysis of the side grip from the left and place cylinder

The data is conducted on all the objects, and on the figures 2 and 3 the cylinder are illustrated. The reason for during this analysis is for correctly pinpoint a location for the robot so that is has the maximum reachability, therefor the data of the four cylinder placement is integrated into one plot. On figure 4 the combined solution for the side grip can be seen, and a white bullseye is showing the best base position for the robot. The correct position has been calculated to be at $[x, y, z] = [0.14, 0.06, 0.11]$




(a) Complete side grip analysis, showing the best base mounting postion



(b) Complete side grip analysis, with a 3D figure.

Figure 4: Reachability solution for side grip

### 2.2.3 Top grip analysis

For the second grip position, there is the top grip which is located directly above the cylinder. The top analysis shows limited usage of the data, as the object type and method used results in either 0 solutions of 360 solutions which is shown on the figures 6 and 7

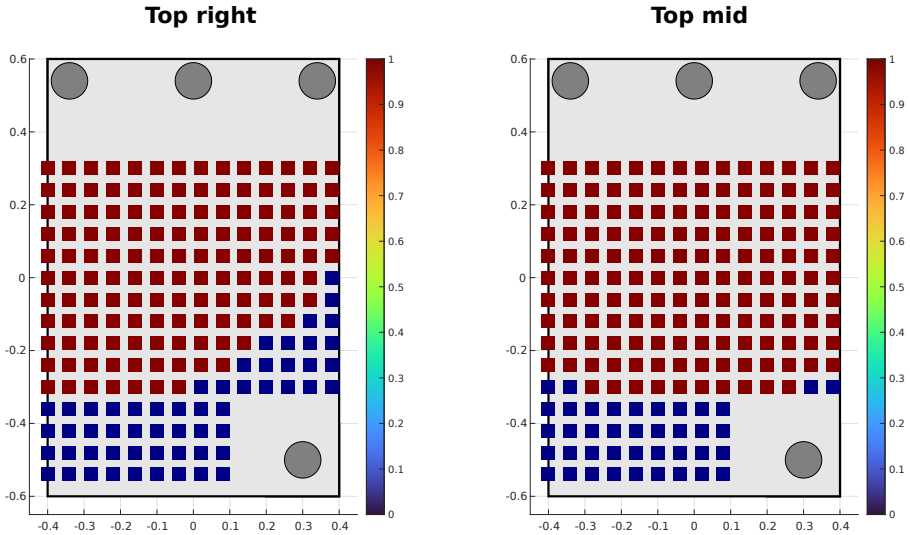



Figure 5: Reachability solution from the Right to the mid-table of the top grip

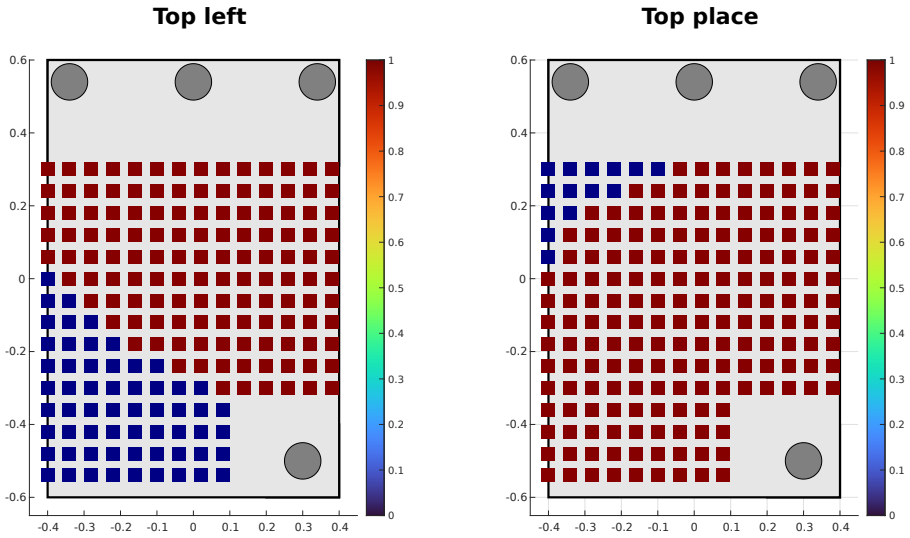
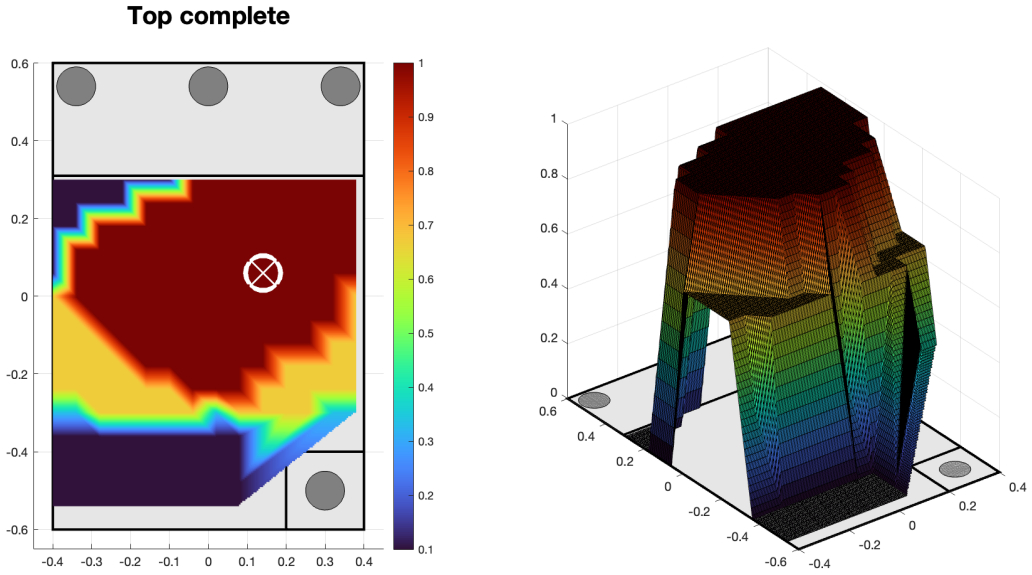



Figure 6: Reachability solution from the left to the place area for the top grip

The best gripping data for choosing the base position would be the side grip, mostly because of the smaller area of full solutions, but also because that the data for the two gripping ways is obviously overlapping on the plot, so it is not that important. But the main gripping option was concluded to be the top grip because of the high amount of solutions.




(a) Complete analysis for the top grip.

(b) 3D representation of the complete solution

Figure 7: Complete top grip solution from the reachability analysis

## 2.3 P2P interpolation

There are different ways to interpolate between points in general pick and place tasks, mainly joint space (JS) interpolation and tool space (TS) interpolation in the cartesion space (CS). From JS interpolation there

are some advantages of not calculating the inverse kinematics, and with known joint angles of the robot the trajectory can be precise. The drawback is that it is difficult to create linear interpolation, as the space does not operate linear. For the TS interpolation between points that define the position and orientation of the end effector, resulting in a well precise trajectory path for the tool end-effector while the rest of the robot's configuration is determined through inverse kinematics. The linear interpolation works by splitting the difference between two points $[A, B]$ into an evenly distributed number of intermediate points and use a velocity profile to calculate the steps between each point. In this project there is used a constant velocity profile instead of ramp up and down velocity between the points. The interpolation is created from the following formula 1, with continues increment of $0.01[s]$.

$$X_{i-1} + (X_i - X_{i-1}) \cdot s(t) \tag{1}$$

The result are a translation Vector3D between the point and a the calculated quaternion rotations of those points. The idea was that the quaternion solution would give a better smooth interpretation of the rotations, but the result of the interpolation still gives a number of singularities. The problems could be that RobWork Studio are the problem, but the tested interpolation still create a jump in joint angles from the derived Transformation matrices of the path points. The problem was not evenly cause, as different changes was made to highlight the issue. Therefor a lot of time was used on correctly defining the starting home position of the robot. The implementation also tries to minimize the errors of the trajectory by eliminating the Gimball lock phenomenon, which can cause the orientation of an object to become no functional when two of Euler angles are equal. A Procedure for the calculation of the interpolation can be seen in algorithm 1, parts of the procures is derived from Robotics lectures, slides and Kasper lecture solutions for example the method for collision free solutions. Additionally there was added a method for linear interpolation used to complete the interpolation between RobWork joint angles.

**Algorithm 1** Collision-free P2P motion for a UR5 robot

```
1:  procedure CollisionFreeP2P(P, T, targetFrame, robot_ur5, wc, state, detector)
2:      path ← [ ]
3:      point ← [ ]
4:      for i ← 1 to P.size() do
5:          for t ← T[i − 1] to T[i] with step size 0.01 do
6:              Pi ← interpolated position at time t
7:              q ← interpolated orientation at time t
8:              point.push_back(convertToTransform3D(Pi, q))
9:              targetFrame.moveTo(point.back(), state)
10:             solutions ← find robot configurations
11:             configuration ← find collision-free solution
12:             path.push_back(configuration)
13:         end for
14:     end for
15:     return (path, point)
16: end procedure
```

To better visualize the interpolation, path plot are provided. See figure 8, where four plots was created.

The transformation matrices was taking from the code to visualizing. Since the trajectory does not operate smoothly the joint angles are a bit off. If the further work was provided on the robotics side a blend method would have been great. Additionally a lot of work was put into conducting the interpolation with a combination of tool space and joint space, since various problems arised by interpolating when the path length between the points was large.

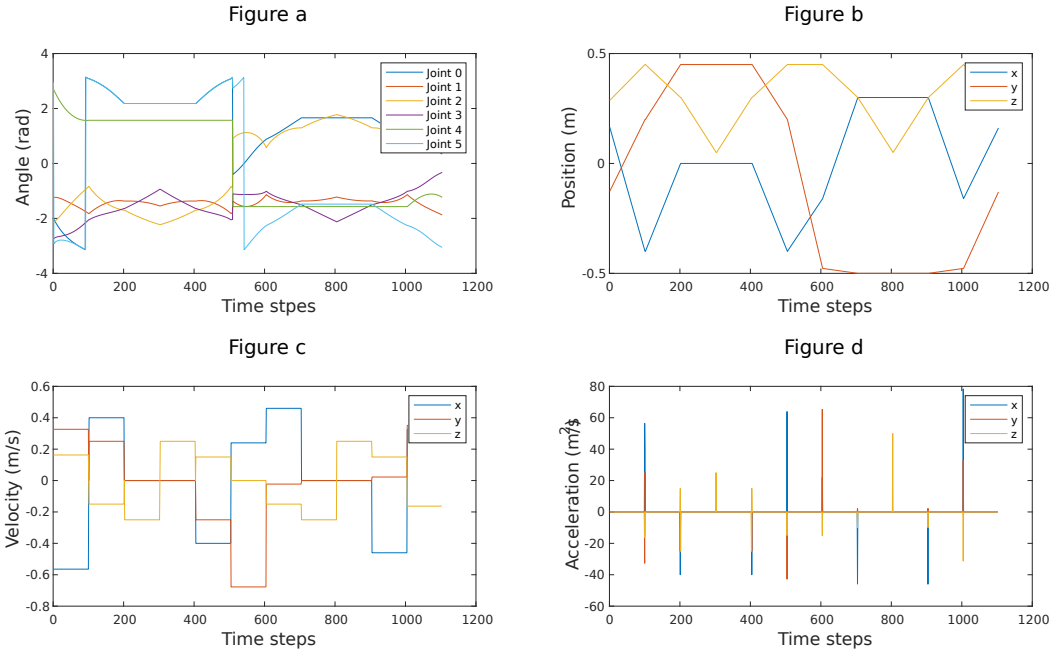
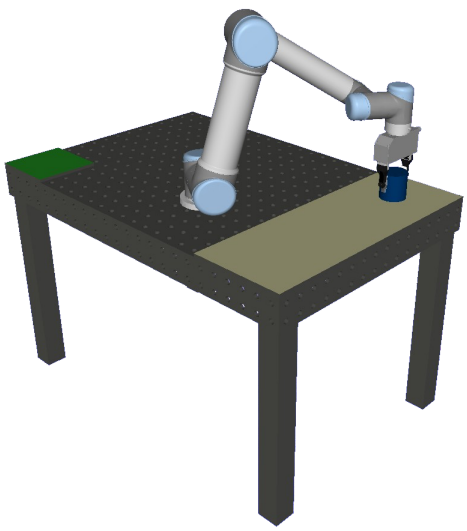
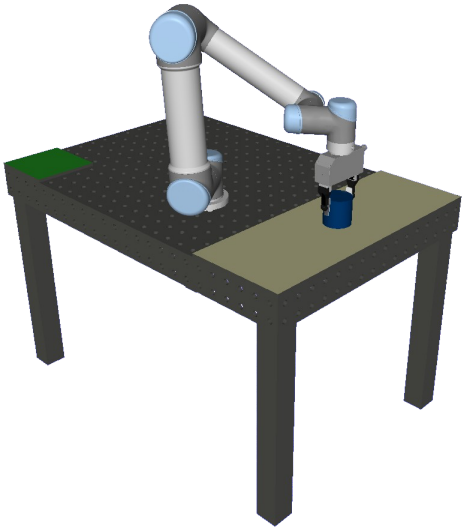



Figure 8: Trajectory plots of the interpolation illustrating: joint angles, position, velocity and acceleration

For the joint angles a plot is provided at the top corner of figure 8*(a)* and shows the angles of all the joints on the robot arm over the time steps. The advantages is to have the ability to look for unexpected movements in the joints or issues with the joints, and to ensure that the manipulator is on the right trajectory. The Cartesian positions plot shows the x, y, and z positions of the end effector and for this plot it can as well be used to visualize the overall trajectory of the arm. At the two bottom subplots shows the linear velocities that is good for identify any sudden changes in velocity, and to ensure that the arm is moving at the desired speed that follows the next plot of acceleration where it can be checked that the manipulator is accelerating and decelerating at the desired goal. The data of the acceleration is also a bit weird, as the idea was not to expecting spikes in the acceleration, and that could indicate that the path is unstable.

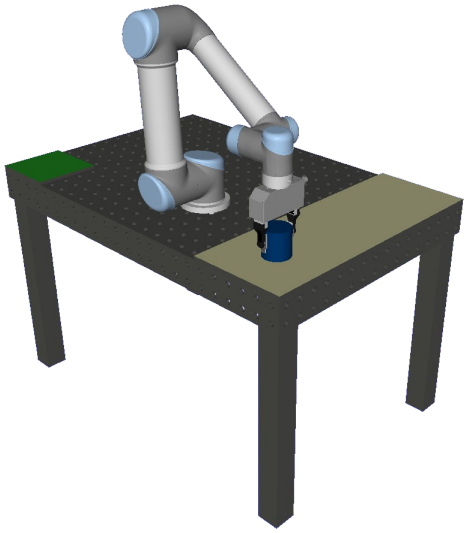


(a) The first pick location at *[-0.25, 0.40, 0.15]*

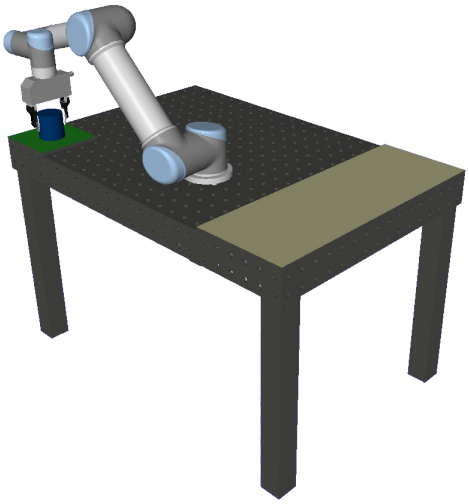


(b) The second pick location at *[0.0, 0.40, 0.15]*

Figure 9: Pick location for right- and mid-cylinder position

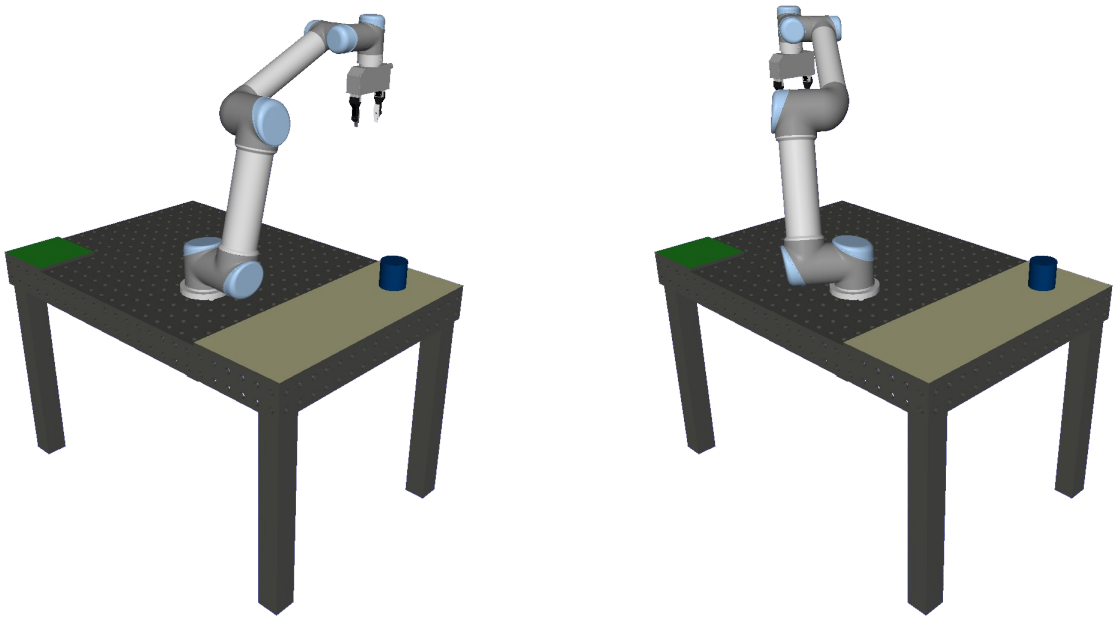


(a) The Third pick location at *[-0.25, 0.40, 0.15]*



(b) The place location at *[0.30, -0.50, 0.15]*

Figure 10: Pick and place location left- and end-cylinder position



(a) The first intermediate point called NearPickPoint in the code

(b) The second intermediate point called NearPlace-Point in the code

Figure 11: The two chosen intermediate point to interpolate between when going between the pick and place area

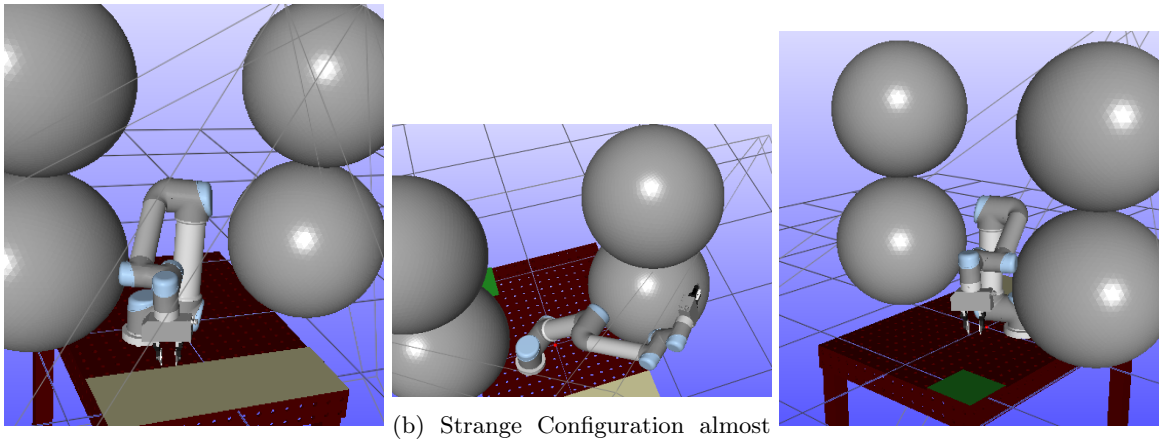
On the figures 9, 10 and 11 the trajectory can be seen. Points was chosen for intermediate, and the rest is calculated from the frame. Most important is figure 10a as it can be sen the rotation that was made on the joint *q[4]* even though that sufficient collision free solutions was taken.

## 2.4 P2P interpolation w. parabolic blend

The parabolic blend was not implemented as the time ran out. Some coding was done, but nothing could be shown to be satisfactory levels.

## 2.5 RRT

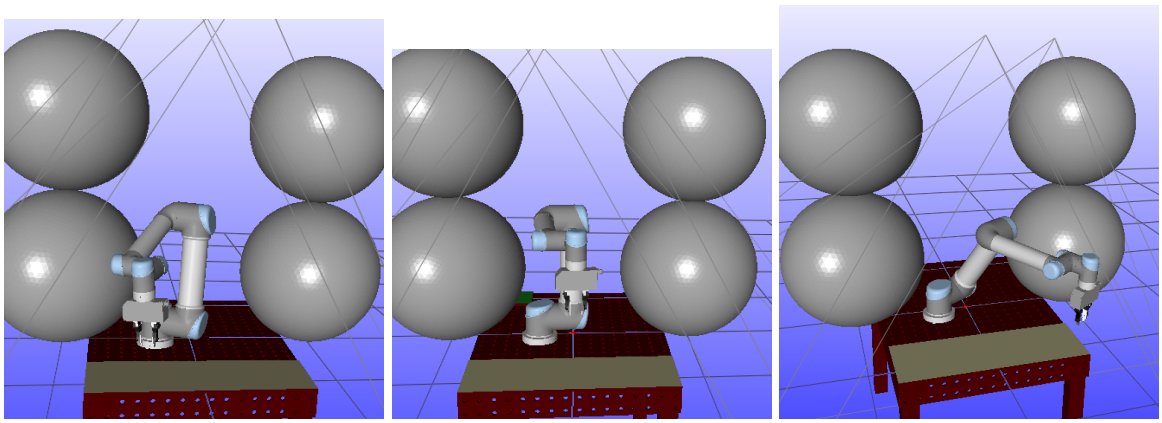
Rapidly-exploring Random Trees (RRT) work by randomly creating nodes spreading out from start and end positions; these nodes are connected to their own tree, so you would have a start tree and end tree spreading out creating nodes. As they spread, the nodes will check if they are close enough to another node from the other tree forming and if yes, they will connect, and we have our path. This path should be optimized as nodes can be removed to create a more practical approach to going from point to point. The advantage of using an RRT compared to a point-to-point method is that it can handle more challenging environments without needing to be prompted, as the path will be created as part of the tree spread, but this also means it will be slower and more cumbersome to find a path. In addition, it will sometimes have some strange configurations. See figure 12 for a view of an unconventional path. The robot arm starts at the "low" start position, see 12a, and goes close to one of the spheres as seen 12b, but stops before it collides and then moves to the goal see



(a) Start position (b) Strange Configuration almost collision (c) Arrives at goal

Figure 12: Strange configurations

12c. So while it will have some strange configurations, it will arrive at the destination if there is a possible path. We used three positions during testing with the same goal see figure 13. We looked into the generation



(a) Low start position (b) Mid start position (c) Top start position

Figure 13: Top position

time of the RRT compared to the step size of the nodes and the step size compared to how many steps the path has. In figure 14 you can see the step size in relation to the path size.

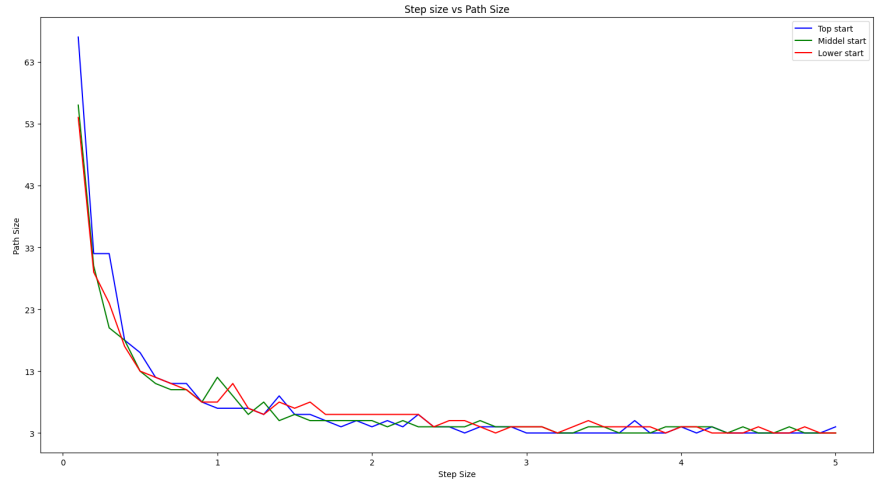



Figure 14: Comparison of step size with how long the path is

We can see that all three positions give around the same performance when generating their path to reach their goal, with the top starting point (blue line) performing slightly worse but nothing significant. In figure 15, you can see the step size compared to the generation time.

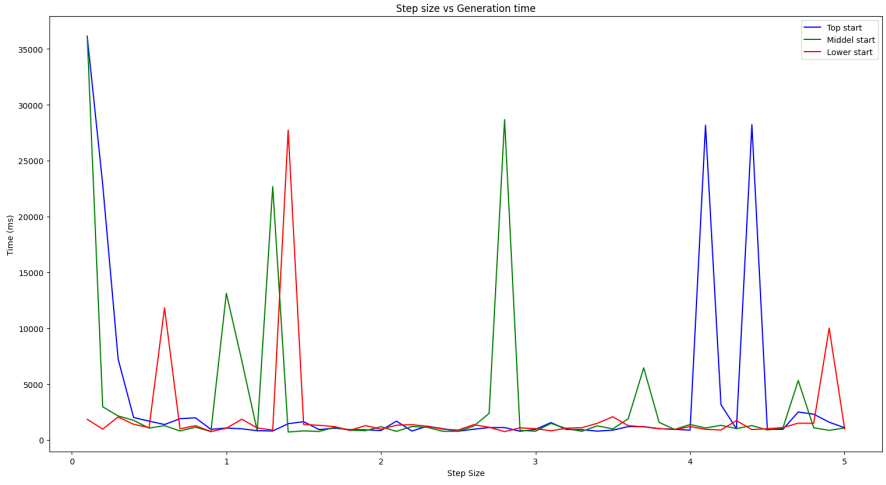



Figure 15: Comparison of step size with how long it took to generate a path

We can see a far more significant difference in the generation speed when we compare the different start positions. The performance is very spread out, with the Top start position (blue) performing worse at the large step size and the middle point (green) in of in the middle and red being slower at the start (red). However, this also shows that overall, the RRT's performance is not dictated by its position in the workspace. Instead, a too-low step size will slow it down.

# 3 Vision - Pose estimation

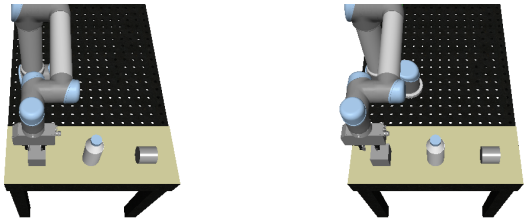
## 3.1 Method 1: Dense Stereo

The Dense stereo method uses two images of the same object with only a slight variation to the left 16a and right 16b and for a full overview see figure 16. The pixel is then matched in both images, and the disparity

(a) View of the bottle from the left camera (b) View of the bottle from the right camera

Figure 16: View of the bottles from the camera

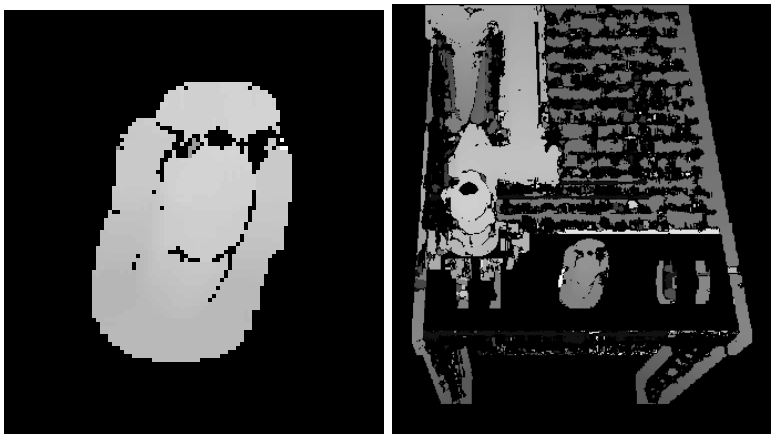
is calculated. This can then be used to define the depth of every pixel, giving us a depth map of every pixel. With this, each pixel would have an x,y,z coordinate that can be used as a point cloud for the object, a similar point cloud will be created for the scene see figure 17 With this point cloud, it should be possible to match



(a) Scene in the left camera (b) Scene in the right camera

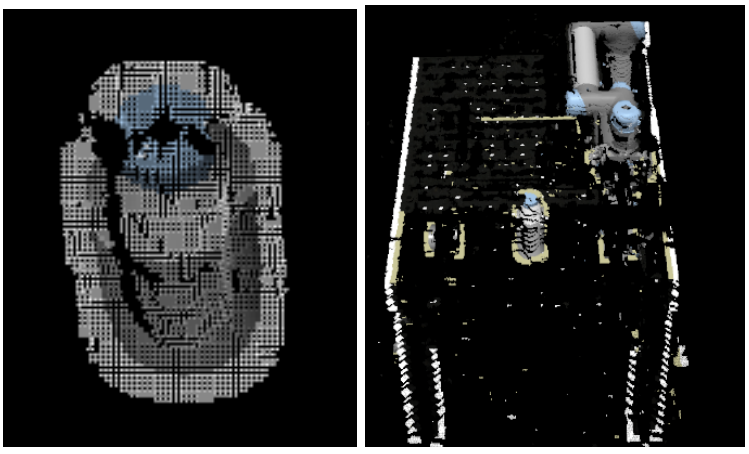
Figure 17: View of the scene in the camera

an object's point cloud with a scenes point cloud which the object is part of, and calculate the object's pose within the local and global frame. This code is based on assignments 4,5 from the vision part of the Robotics and vision course. We will use one example to give a general overview of how the process works. First, we have the two images of our scene with the object, in this case, a bottle. When we have the Bottle by itself in an empty scene, the cameras are still in the same position as their configuration in the first scene see fig 16a and 16b. We use open cv to calculate the disparity in the image and reproject it, giving us a normally distributed image that we can use for visual conformation see 18 where the left image 18a is the bottle and 18b is the scene with the bottle the middle. This depth map we created from the distribution can now be saved as a point cloud. In the end, we will have two point clouds, one for the scene with the object in it see 19b and one for the object 19a. The pre-processing of the point could not work as intended, but the theory



(a) Disparity Bottle (b) Disparity Scenea

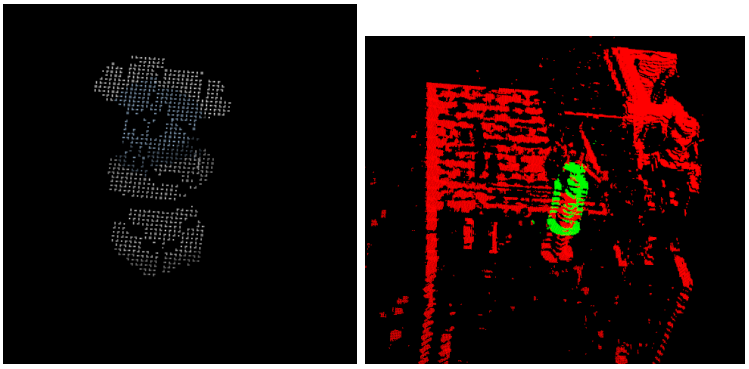
Figure 18: Normalised depth maps



(a) point cloud of the Bottle (b) point cloud of the Scene

Figure 19: Point clouds

behind it is that by using filters, we can improve the point cloud and thereby improve the potential pose estimation of the object. It would be using three different filters. First, the voxel grid sets a 3-dimensional grind over the entire cloud, and the pixels inside one of the boxes would be centralized and the centroid. Then the outlier removed would remove any points that did not have enough supporting points, thereby removing any outlying points. The spatial filter uses an upper and lower limit and only keeps any points where data is within the threshold. See figure 20a for an attempt at filtering the bottle. We can see a very large data loss. If it had worked, we should be able to do the global and local pose estimation on the dense cloud to try and calculate our pose estimation. As the pre-processing did not work as intended, the global and regional pose estimation could not be performed. In figure 20b we can see the start of the un-filtered bottle as seen Suppose a proper point could have been created. In that case, testing with especially salt and pepper noise could have shown some interesting effect on the point cloud and the pose estimation, as the extra pixel data could make it hard for the system to detect the correct pixel pairings to create the disparity calculations.



(a) Badly filtered bottle (b) Local pose estimation

Figure 20: Trying to pose estimate

## 3.2 Method 3: Sparse Stereo

Here the second vision method is presented and is about detecting the object by retrieving the feature location of the object in the scene. The setup is stereo where the key is to detect features in the object and triangulate between them to compute the pose of the object. The sensors are simulated in RobWork and are differentiated with the left and right camera. The first technique used to locate the position was on a green ball, the color were easy to filter out and a circular ball had fewer features to extract and a better representation of the center of the object. In figure 21 the steps is shown to obtain the 3D pose of the ball.


Figure 21: Flow diagram of the steps in sparse stereo detection of a ball

After the image collection, the green ball was color filtered to the range the green color. Then the circles was located in the scene with Houghcircles from OpenCV and the pose estimation was found with help of OpenCV triangulation method [6]. The importance is features that can be recognized in both the left and right image, here the feature are the center of the circle that is found. In general the real world scenarios of a ball would not be the best object to use with sparse stereo as it would be best to use rigid objects with some key features, but the ball have the key feature and the performance shows great results when testing in the simulated environment. The advantages to use the ball is the precise pose estimation as the center feature is on the object frame. The ball method is different from the second feature extraction that was conducted with this vision method, where a more complex object were used to test the functionality of the sparse data used for the M3. To make use of the features from the green ball the center points was used in by triangulate the features. This is done so that the 3D position can be calculated from a point from the 2D projections of the two images.

### 3.2.1 M3 with feature detection

Sparse stereo is a method for locating points in 3D space that correspond to sets of matched points on an object, therefor unique features for an rigid object is the best choice. In figure 22 it shows the second extracted object from the workspace, instead of the green ball, there is used a T-Rex head with a high number of features. To better color segment the object the green color filtration was reused, and then filtering out everything else from the scene, so only the object is used to detect the features. there is used FAST and ORB methods from OpenCV [5]. After a completion of the matched features the same OpenCV triangulation method was used to estimate the position. This method is effective for finding points on the surface of the object closest to the cameras, but may not be suitable for irregular shapes when a complete pose estimation are needed. Some problems arrive when matching the found features to the transformation to the table. The output 3D point has a large deviation to the expected pose relative to the table, therefor no further work was done the the object for pose estimation with this method. The object does not match the method used, and further work needs to be implemented to correctly map the detected features to the origin of the object frame. The code for this can be found in external appendix A.

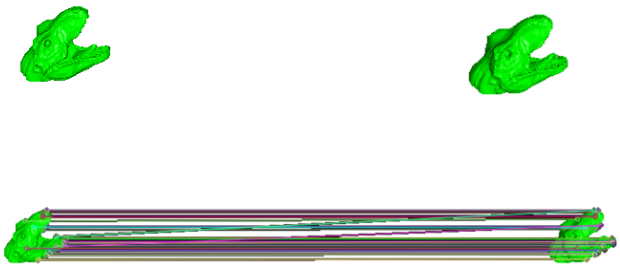
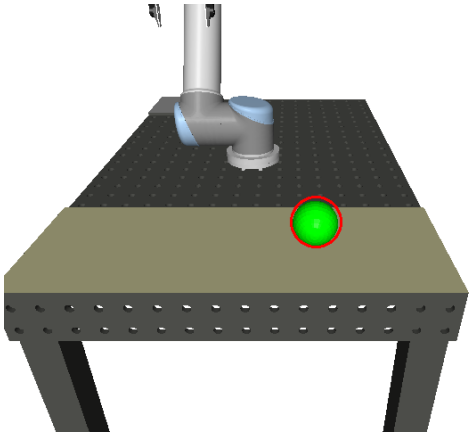


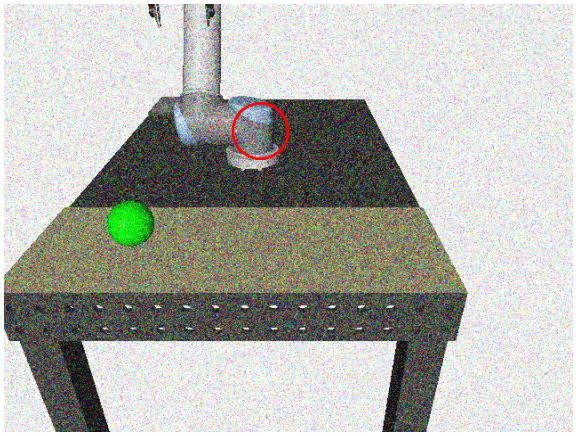
Figure 22: images derived from the DinoSparseStereo package illustration the feature matched from the T-Rex object.

### 3.2.2 M3 evaluation

To evaluate the pose estimation of the M3 method, it was suggested to use Gaussian noise distribution. After the exclusion of the second object, etc. the T-Rex head, it was chosen to test on the green ball it outline the performance of pose estimation. The reason for introducing Gaussian noise to the images is to create a good evaluation of the robustness, accuracy and show if the system can meet expectation under various levels of noise. Real images from a non-simulated environment are typically full of noise from sensor input and light, and this noise can affect the accuracy of the depth estimates. Therefor it is needed to stress test the system, and see when the features in the image are not detecting the ball position. There where used consistent and controlled error levels to introducing noise, the chosen standard deviation was [0,1,5,20.50.100,200,255]. The noise is also good for evaluating between methods, but the time was limited so the overall comparison of the methods from M1 and M3 was not made unfortunately.

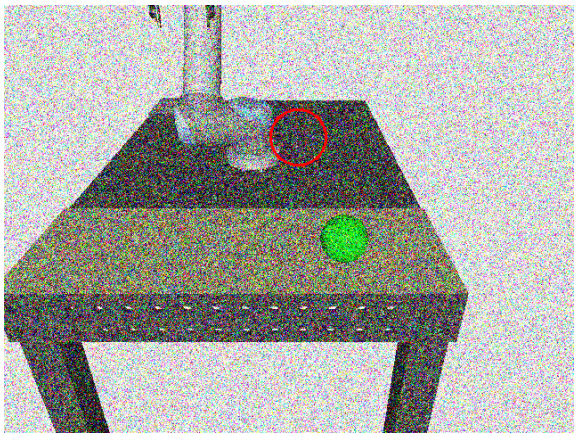


(a) Image with Std[0] from noise evaluation

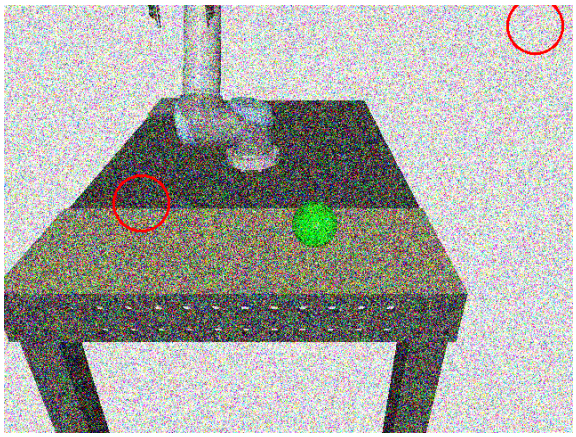


(b) Image with Std[50] from noise evaluation

Figure 23: Test images for noise with strange preliminary results compared to the larger data pool



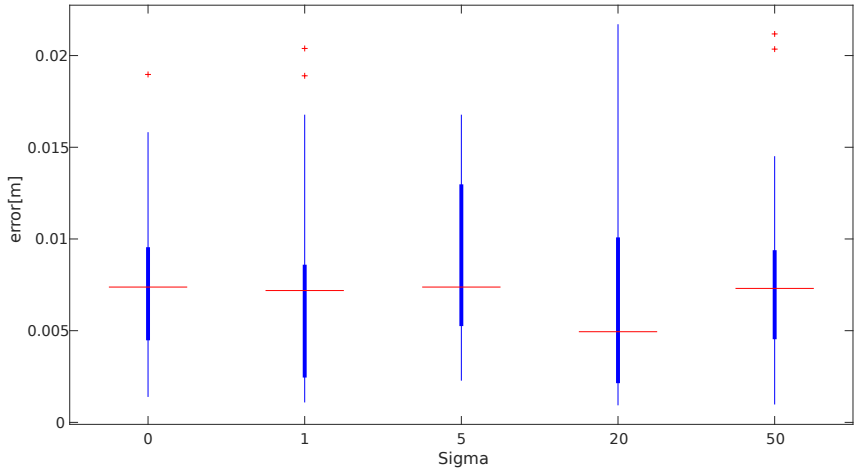
(a) The first image with Std[100] from noise evaluation with significant errors



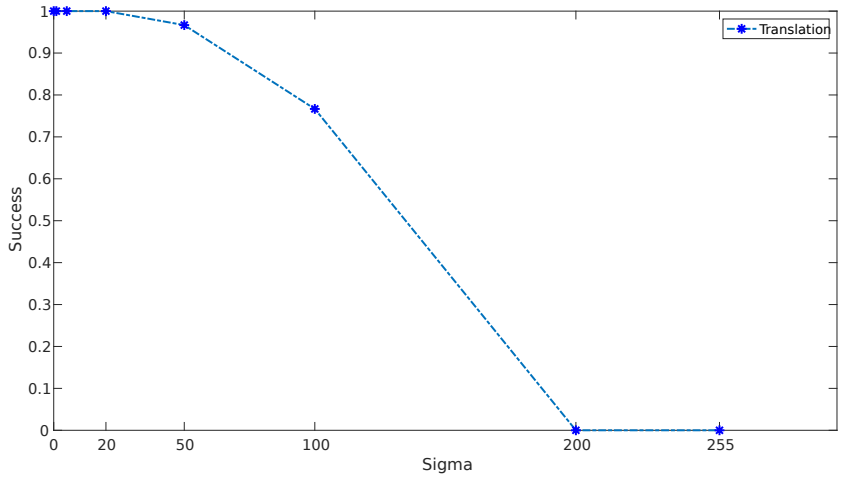
(b) The second image with Std[100] from noise evaluation with significant errors

Figure 24: Test images for the noise where the system breaks

Output of the image data can be seen on the figures 23 and 25b, the system break within a standard deviation of 100 in this experiment. The test was made with 30 images, that was loading in one by one and evaluated on both cameras. The figures 23 and 25b shows some of the data, where the different performance issue is shown. The first issues are the circle detection that is not fitting correctly so the system needs to be improved to better deal with uncertainties. Second the performance is quickly downgrading and already from a standard deviation of 50 the system get mismatch representation of circles. After the noise experiment, various changes was made from change in experiment values of HoughCircle detection, the color was changed to a full green scale value in the RGBA color space, and additional changes to HSV color filtering.




(a) Showing the error estimation from one data sample of best solutions




(b) The translating of success in finding the right pose in correlation to how noise the image are

Figure 25: Figures of the M3 vision performance

The new change experiment can be interpreted on figure 25 where the goal was achieve with changes to the system improved the system performance by 100% and the noise stability test breaks around a standard deviation of 200. The further data is based upon the rewritten experiment explained above. The two plots simply plots the data precision and success for pose estimation.

# 4 System integration

A combination of the system was implemented for displaying simple functionality and there was introduced basic UI elements to a RobWork Plugin. Because of the problems that arrived with the methods the result became a simplistic implementation, where object used for integration was the green ball. Without significant noise to the system 3.2, the M3 pose estimation technique gave a good estimation. The interpolation was included to create the path, and the performance can be visualized directly inside RobWork Studio, with

limitation such as problems with singularities explained in section 2.3. A video of the complete system can be found at reference [3] but also a part of the external appendix C, where a simple video was created to show the work for the combined vision detection and robot interpolation. The SamplePlugin integration was fairly successfully but often the time was used on troubleshooting the unknown errors resulted from mismatch in the scene.xml and device.xml files.

# 5 Discussion

In section 2.2 the reachability of the robot was introduced. The code was mainly conducted from the robotics lecture 3, where Kaspers gave a useful initial solution to the reachability. This help to conducted the two analysis of the top and side grip. The implementation were done as a plug and play solution, where the initial values needs to be put in and the analysis loops through the base positions according to the chosen step size of incremented the x and y values for the robot base on the table. The plots are derived in Matlab see external appendix B, and was chosen to take more samples to analyze then the necessary 60 base position. The amount of positions included in the experiment was 190 different positions, to include enough data to used on the graphs. The interpretation of the top grip was difficult, as the solution either derived to 0 or 360 solutions. The robot base position was calculated to be $[x, y] = [0.14, 0.06]$, and initial that was not provided any problems but mostly to problems were arised from this. Problems in regards to RobWork environment, made it difficult to move the robot base position when before during interpolation. The option was to move the base from within the code instead of updating the scene files.

(*Emil*)

For the linear interpolation the section 2.3 was implemented with points used for interpolating between two points in a trajectory path, the work can most certainly be improved upon. To optimize the velocity profile for interpolation, a ramp up and down could be added to the trajectory but instead a constant velocity profile was used. The improvement on the work in the robotics code for the P2P could also be to derived a rearrange of the code to do both interpolation in tool space and joint space with int the same pick and place operation.

(*Emil*)

The M3 sparse stereo method that was used in this project are explained in Section 3.2, and shows the result of the performance of the pose estimation. The disadvantage was that the method were not providing an orientation of the object. The method are used by finding correspondences between points in two images by triangulation taken from slightly different viewpoints, and then find correspondences by calculating the depth of each found point. The method only uses the position of points in the images to calculate depth, and does not consider the orientation of these points. One way to obtain this information was to use feature detection algorithms such

as FAST and ORB, which can identify distinctive features in the images and track them as the viewpoint changes. These features can then be used to estimate the pose of the object, including its orientation. Despite of the low accuracy of the feature method in M3, the method with a green ball gave a decent solution for pose estimation before adding any noise, the position was relative precise with a deviation of $0.01[m]$. To improve on the ball detection, the future work was to used the developed Sobel edge detection to cross check the method for locating the ball. Some test was made with the inclusion of the edge-detector and it initial showed that it increased the performance under the noise experiment. The implementation was to add a inner layer that looked for two instances of drawn circles, and after compared the radius difference between the two solutions with a certain threshold.

(*Emil*)

In section 2.5 we showed the RRT how it has minor performance fluctuations with the differences in start positions and how the step size should not be too small, or it will harm the performance. In a more complex environment, the performance could be much more varied. RRT also showed that while it had some strange configurations, as expected of a random node performance. The main optimization would be an effective planning optimizer like RRT* that could improve the pathing. However, the RRT performed well ever without it and was robust to changes in step size.

(*Steffen*)

The dense stereo 3.1 could have been implemented better. There were massive problems in getting the point cloud of the scene and the objects alone. Therefore modifications to parameters in the dense stereo code were needed to get it to work. This meant a lot of trial and error, and a lot of time was spent trying to get the point cloud. When finally then, the problem of the pre-processing showed itself after. The correct parameter to process the point cloud (PCD) were hard to find, and the data loss was much higher than one could have wanted, leading to global and local pose estimation not working. So the issue was getting images of high enough quality and creating a quality point cloud. However, point clouds were generated by the dense stereo implementation. With more time, a good pose estimation could be done using our implementation.

(*Steffen*)

# 6 Conclusion

In this report, we examined various techniques for designing the workspace in RobWork, robot motion planning, and pose estimation of an object. The reachability analysis of the best position for the robot base, presented in Section 2.2, revealed that the placed the robot base at (0.14, 0.06, 0.11) on the table was the best position and had the highest number of collision free solutions. We tested two different algorithms for robot motion planning to execute the pick and place tasks. The Rapidly-exploring Random Trees Connect algorithm and point to point interpolation. Two methods for pose estimation of an object were also presented. The sparse stereo method was only able to estimate the position with a shorter execution time but was more reliable. The dense stereo was partially implemented but never provided a pose estimation that could be compared to sparse, we believe the sparse would be faster and better on simple object pose estimations, and the dense would work better on objects with complex features, but Dense stereo is also slower. Finally, a combination of the pose estimation and pick and place execution techniques was demonstrated, which utilized the sparse stereo method for pose estimation due to its increase in reliable accuracy poses and point to point interpolation for pick and place to avoid singularities. This combination was capable of accurately determining the pose of an object in the pick-up area and then successfully picking it up and placing it in the place area.

# References

[1] Universal Robots A/S. *Universal Robot UR5e*. URL: https://www.universal-robots.com/products/ur5-robot/. (accessed: 09-12-2022).

[2] Lars-Peter Ellekilde and Jimmy A. Jorgensen. "RobWork: A Flexible Toolbox for Robotics Research and Education". In: *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)* (2010), pp. 1 –7.

[3] Emil Reventlov Husted. *Presenting the Pose Estimation with Pick and Place Execution in combination within RobWork Studio*. 2022. URL: https://youtu.be/Vbl9PtCqroc. (accessed: 22-12-2022).

[4] Emil Reventlov Husted and Steffen Waage Jensen. *RoVi project*. 2019. URL: https://github.com/stackovercode/RoVi_Project2022.git. (accessed: 22-12-2022).

[5] OpenCV. *OpenCV Homepage*. URL: https://opencv.org/. (accessed: 11-12-2022).

[6] OpenCV. *triangulatePoints()*. URL: https://docs.opencv.org/4.x/d0/dbd/group__triangulation.html. (accessed: 22-12-2022).

# External Appendices

**Overview of external appendices**

1. Appendix A - Plugin.zip
    (a) SamplePlugin.zip
    (b) ReachabilityAnalysis.zip
    (c) Robotics.zip
    (d) RRT.zip
    (e) DenseStereo.zip
    (f) SparseStereo.zip
    (g) DinoSparseStereo.zip
    (h) camereaImageMaker.zip
    (i) WorkCell.zip
    (j) WorkCellObstacle.zip
2. Appendix B - Matlab.zip
    (a) FinalM3Performance.mlx
    (b) M3Experiment folder
    (c) FinalPathPlots.mlx
    (d) PathExperiment folder
3. Appendix C - Miscellaneous.zip
    (a) RoviProject.mov
    (b) codeExecutionReadme.txt