Pick and Place, using Robwork

Bjørk, Anders; Dinten, Imara van; Setaro, Michelangelo
[anbjo11, imvan13, miset14]@student.sdu.dk
All authors contributed equally

University of Southern Denmark
Faculty of Engineering, Maersk Mc-Kinney Moller Institute

28. Maj 2015

Contents

Abstract

1 Introduction

This report describes the Pick&Place project of group 8. The report is set up to first give some information regarding to the set-up and used workcell, after this the methods for calibration is explained. Followed by a chapter about Vision and Robotics. In the end we'll have a brief discussion about the different aspects of the project, what went right and wrong, followed by a final conclusion. Overall was this an interesting project with quite a lot of challenges.

2 Method

3 The Robot

4 Calibation

5 Vision

The vision part of the project deals with the detection of known objects using mono or stereo cameras.

5.1 Known object

For this project we have two objects, which are created using the 3D design software called Autodesk Inventor 3D CAD. The result of this gave us accurate 3D models which we were able to physically recreate using a 3D printer. The end result was an object the robot is able to pick up, with known dimensions, see Figure 1.

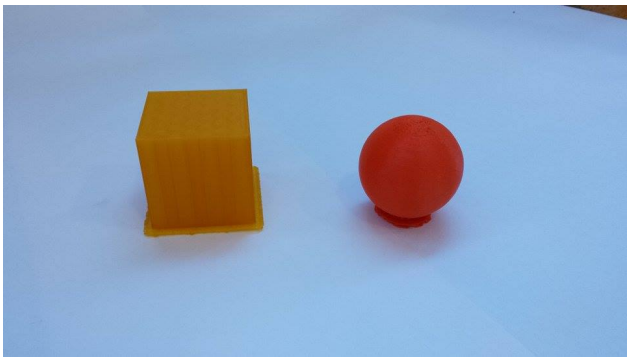

Figure 1: Objects used

5.1.1 3D to 2D

Since the world observed by a camera is 2D, some work has to be done to be able to perceive the 3D object. To do this, the objects were rendered from a multitude of different angles. The simple nature of these objects made it so only a few viewing angles were needed to capture the essence of the 3-dimensional shape. The first object used is a sphere, this shape was chosen as this is the least complicated shape to test with. No matter from which angle the object is observed, the shape will always be the same. See Figure 2. The second

object, a cube, is a bit more complicated shape. When a cube is observed from a different angle, the shape will look different. For this object it was necessary to render images from different angles. A small selection of the resulting images can be seen in Figure 3, Figure 4 and Figure 5.

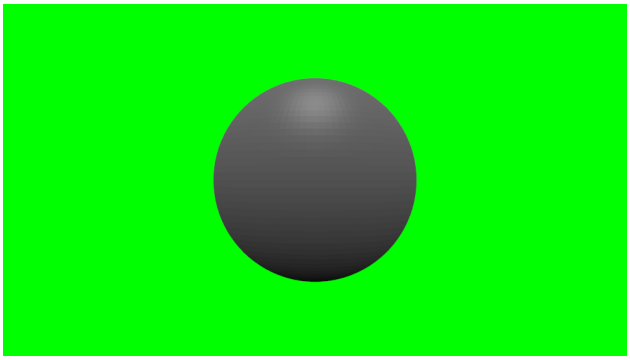

Figure 2: 2D image derived from 3D sphere

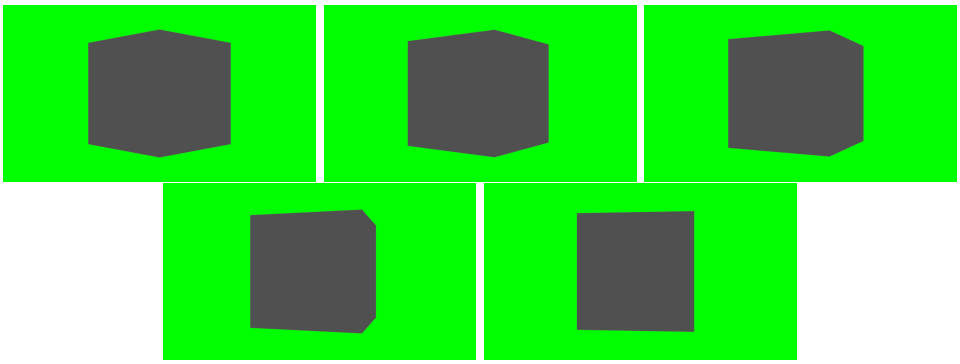

Figure 3: 2D image derived from 3D cube

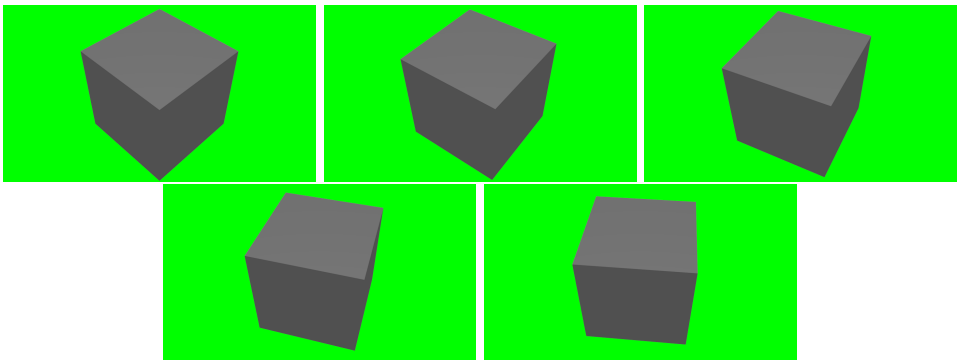

Figure 4: 2D image derived from 3D cube

5.1.2 Contour detection

There are different ways to extract the features of an object. For this project we used contour detection to extract the main features of the object. This way we received enough information from the object to compare

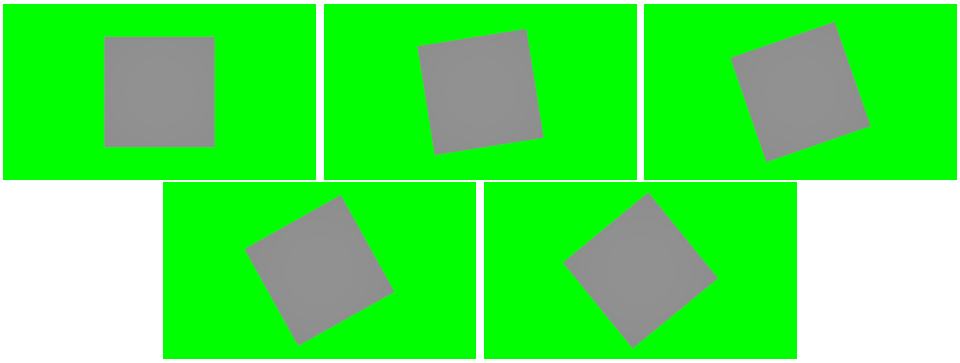

Figure 5: 2D image derived from 3D cube

with, but we didn't overcomplicate the problem. As is described in [1], there are more complicated but robust ways to extract features from an object. Though with the current time limit, we didn't pursue this method. As described later in the report, the used method was crude in comparison but proved to be sufficient to meet our goals. For the contour method we used OpenCV. This library provided us with tested methods and enough flexibility to construct a solution for our problem. The frames from the camera feed are first converted to grayscale images. As is shown in [2], using an RGB image instead of a grayscale image will not increase accuracy by a huge amount. "Almost 90% of edge information in a color image can be found in the corresponding grayscale image." [2] So in compliance with the K.I.S.S. [1] principle, we used a grayscale image. Which is a prerequisite to be able to use the standard contour detection built into OpenCV. Figure 6 shows the end result of the contour detection on a 2D converted image from the sphere.

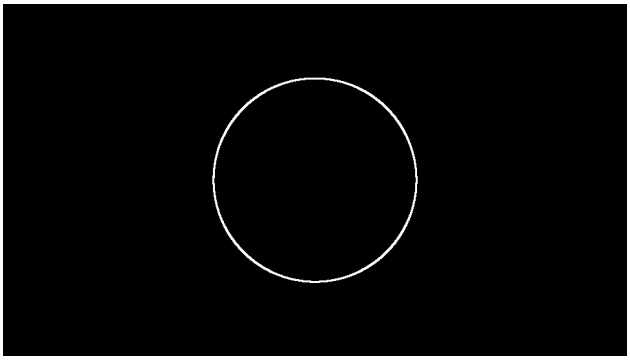

Figure 6: Contour detection from sphere

5.2 Camera

For this project we use a BumbleBee2 and an Asus XTion Pro [2]. Each of these cameras have their own features which are usable in different parts of the project. Figure 7 shows the result of using the Kinect camera. The cameras show us the operational environment of the robot. The Vision part here is to detect the object based on information extracted from the video feed. Detecting the object can be done in several ways. For this part we already have an image of the contours of the object we're looking for. This gives us one way of detecting it.

[1] Stands for "Keep it simple, stupid" or "Keep it short and simple"
[2] Also called a "Kinect", though this is not the original Kinect from Microsoft.

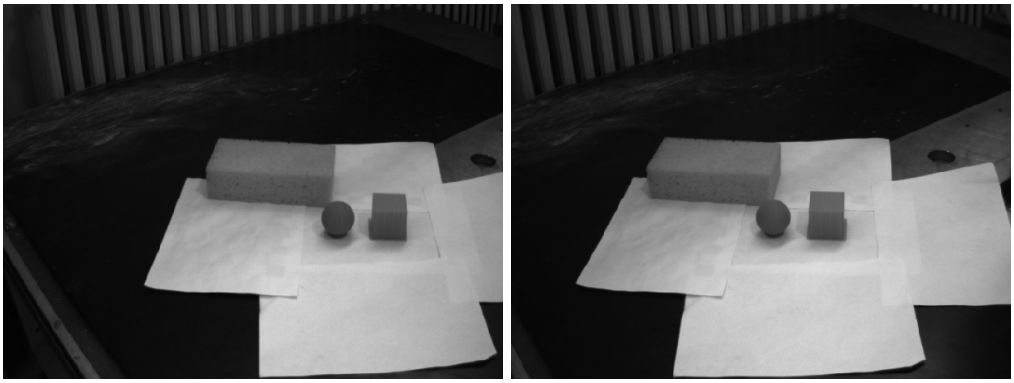

Figure 7: Kinect Camera

5.2.1 Colour segmentation

As the colour of the object is already known, performing colour segmentation on the video feed is a logical step to make. This will narrow the visual search-space by excluding any objects that have nothing to do with the object we're looking for. If the object was built using multiple colours, using a colour segmentation method would still be valuable as long as there are not too much overlapping colours. This object doesn't have any colour patterns, which you could have also used otherwise, but it consists of one colour. This makes this step a bit easier, though detecting an object with multiple colours would have been a nice expansion. For this object, the cube has the same colour, we returned a thresholded image which only contains the colours in a range between H=0-179, S=130-255 and V=80-255. As shadows and different light conditions have an effect on the result, detecting the colour within a certain range is necessary. The resulting image will then be morphologically opened and closed in order to remove any small irregularities. Figure 8 shows the end result.

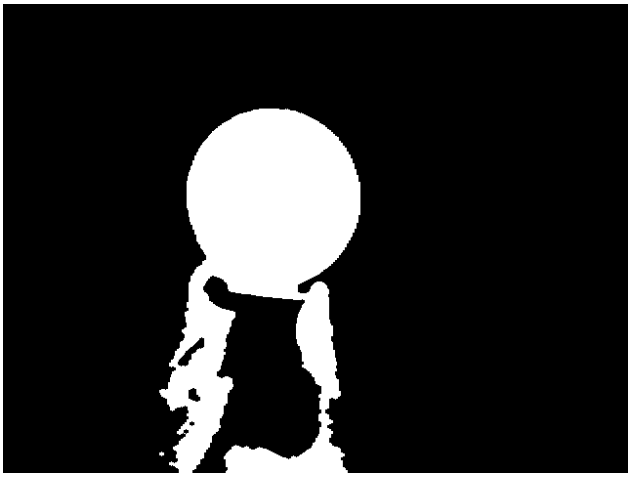

Figure 8: Colour segmentation

5.2.2 Contour detection

The contour detection used on the video feed is the same one used for the 2D image of the object. Only minor changes were necessary in order to change the video feed to the correct colour space. See Figure 9 for the result of the contour detection after a colour segmentation.

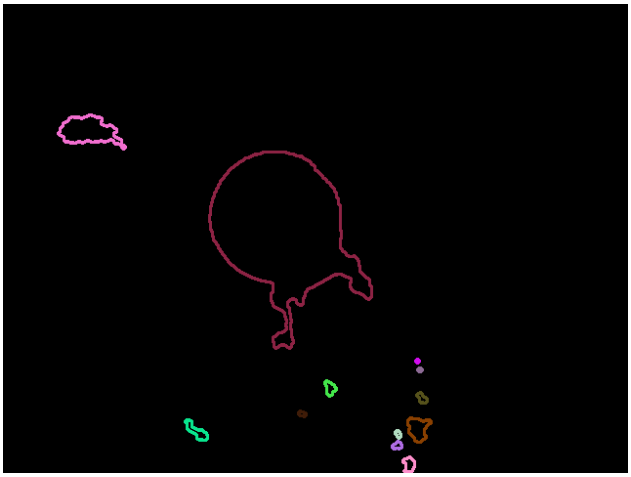

Figure 9: Colour contour detection

5.3 Feature detection

The feature detection uses several different methods in order to compare features from the 2D image to the converted video feed. The feature detection mainly consist of a SURF algorithm combined with a FLANN matcher.

5.3.1 SURF

To detect features from an image, a SURF algorithm is performed on both the object and the scene (i.e. video feed). SURF stands for Speeded Up Robust Features detection method. SURF was inspired by the SIFT algorithm but is several times faster. Because we're using a live feed for feature detection, the speed of the algorithm is a crucial factor. OpenCV has a SURF implementation which we used for this project. We used SURF with a Hessian threshold of 400. This resulted in enough keypoints to use for the comparison.

5.3.2 FLANN matcher

FLANN stands for Fast Approximate Nearest Neighbor Search Library. FLANN is a library for performing fast approximate nearest neighbor searches. The algorithm used to match points from the object and scene is based on this. The FlannBasedMatcher function trains the descriptor collection and finds the best matches [3]. After finding the best matches, a quick calculation is done between the keypoint distances. In the end, only matches which aren't too far away from each other are considered "good" matches and are used. The end result is then drawn and can be seen in Figure 10.

5.4 Vision and ROS

The connection with ROS comes down to receiving the video feed from different cameras and publishing information. After processing the live camera feeds, helped by the parameters extracted from calibration, we can publish the 3D coordinate of the desired object to be handled by the Robotics part of the system. The Robotics part can then calculate the needed movements in order to pick up the object and place it elsewhere. This is discussed in section 6.

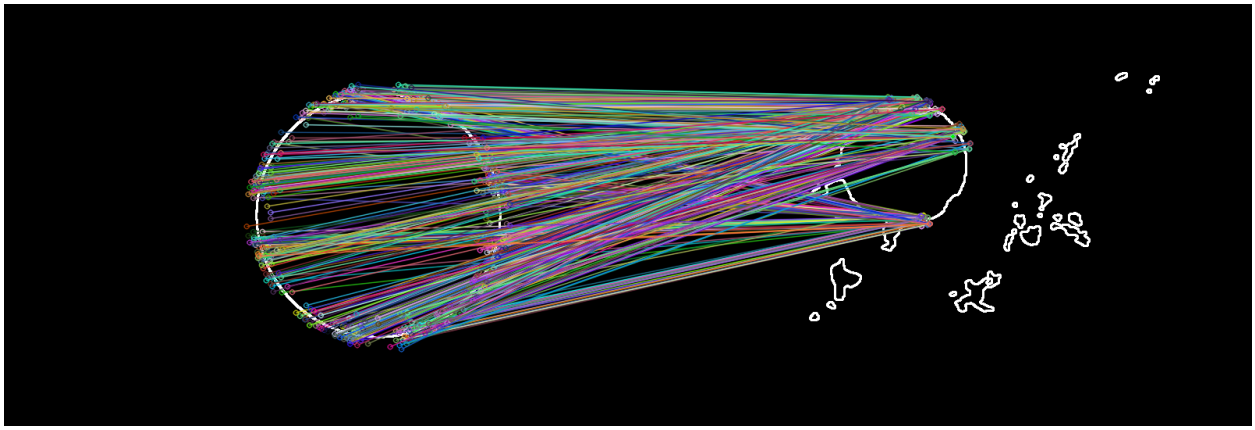

Figure 10: Colour contour detection

5.4.1 PCL

ROS currently has a message-type implemented for Point Clouds. The second revision, which we were planning to use, is also the de facto standard in PCL [4]. We chose the ready-made ROS implementation over an OpenCV implementation because it was fully implemented and quite robust.

The left image in Figure 11 shows a point cloud as rendered in RViz, using PCL in ROS. The middle image zooms in on that particular cloud. As a trial, we used a larger (rectangular) object in the scene as this gave better visual feedback. The right image in the figure highlights this rectangular object. The object in the backdrop is the radiator that is mounted on the wall.

Point clouds can be used to give quite accurate estimations of an object's position in 3D space viewed from the camera. Unfortunately, we were not able to implement this as a final stage of the visual processing pipeline. By utilizing both cameras we would be able to extract Point Cloud information based on regions found in earlier stages of the Vision pipeline. This would add the necessity of calculating transformations between the two cameras. By finally combining this information with the transformation matrices applicable to the Robotics system, we would be able to publish the coordinates of the object as seen from the robot.

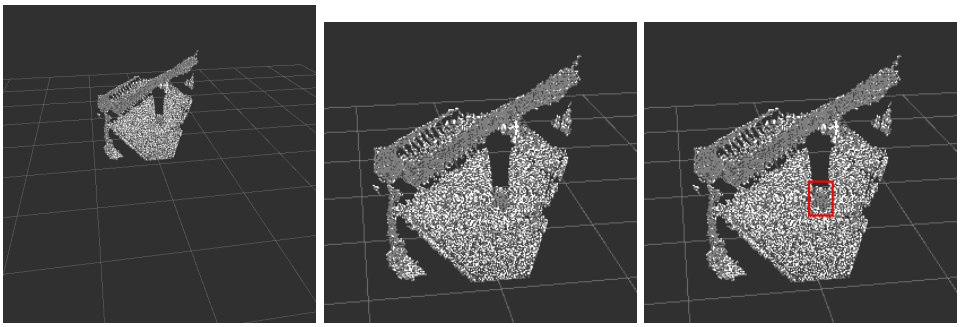

Figure 11: PCL

6 Robotics

In this section, we will consider two important things. The first deals with finding a good path for the robot to follow. This means of course that the joint configurations change. Another thing entirely is how the joint configurations avoid going out of limits and avoid hitting obstacles in the way. To do this a path planner needs to be made to take these matters into account. The group has decided to use a Probabilistic Roadmap Planner, PRM for short. The second important matter is to find the correct joint configuration to grasp the object. There are a few things that we know about the object, that is the point where the robot can grasp it as well as the orientation. To get possible joint configurations and interpolations thereof matching the desired endpoint, there is a set of inverse kinematics equations that need to be solved. The constraints for these are such that any found joint configuration does not exceed joint limits or provokes a collision with anything in the workcell.

6.1 Path planning

As mentioned before, the method used for path planning has been chosen to be a PRM planner. The reason for this is that the PRM makes a roadmap where the nodes have a uniform displacement. Given the fact that these roadmaps are generated based on the workcell, these can be generated offline to have better runtime performance. The uniformity of the nodes will guarantee that there are paths to follow to get to the desired endpoint. Alternatives, like the Rapidly exploring Random Tree (RRT) do not share the same characteristics, which means that the runtime-complexity of the algorithm can not be guaranteed. For each different endpoint in RRT, new paths would have to be generated. In section 6.2 there is a comparison to show the difference in calculation time between RRT and PRM.

6.1.1 Implementation

The PRM planner is generated by first giving the planner a work cell and information about the devices used. After this an initial phase is started where a number of nodes are being randomly placed in the work cell. While they are placed, a local planner makes a check to see if the position of the node does not intersect with an object's geometry. The generated nodes are then connected so they form the start of the graph used in the roadmap. While connecting the nodes, the connecting edges are being checked to ensure that they do not run through any objects. After this, each node is expanded further by a number of new nodes, thus growing the graph further. Some number of iterations later, there is a complete roadmap spanning the entirety of the workcell. By adding the start and goal configurations, a simple Dijkstra search will yield the result by returning a vector of joint configurations as determined by the roadmap.

6.2 Test of the planner

To test the consistency with which the planner creates a graph and plots a route, the algorithm has been used to generate a roadmap for the same workcell 100 times. In doing so, the runtime and number of intermediate nodes on the path have been measured. To be able to compare the efficiency of the PRM algorithm with the RRT algorithm, the same tests have been repeated with a RRT planner. The RRT planner implementation used in these tests is that of RobWork. The results of the comparison can be found in table 1.

6.3 Inverse Kinematics for grasping Objects

With the path planner out of the way, it's time to look at grasping the desired object. As mentioned before, the robot is moved by changing its joint configuration vector. An object does not contain any information

Planner	Creation time	Time used searching the map	Number of joint configurations used
PRM	∼5sec.	∼0.9sec.	∼4
RRT	∼0.06sec.	—	∼30

Table 1: Result of test. Creation time, includes the time used to find a route. Time used searching the map, only applies for the RPM, and represents the time used to find a new route using the previous graph. The number of joint configurations used, is a number on how many configurations are used to get the robot from A to B.

at all. In accordance with section **??**, the vision system can detect the object and create a so-called point cloud. If we consider a random point in that set, we get a three-dimensional position given by X-, Y- and Z-coordinates and an orientation in RPY format. With this information, a homogeneous transformation for the desired position can be calculated. In addition, a similar transformation can be made relative to the starting configuration of the robot. All the transformations are calculated from the base to the toolmount of the robot. By feeding all this data into a Jacobian-based Inverse Kinematics solver, new joint configurations can be found with a small displacement. Thus, giving an interpolation between current and desired joint configurations.

6.4 Conclusion on Robotics

There have been several attempts at implementations of various algorithms. However, only the PRM planner has been successfully implemented. Even though the tests have shown that it will take some time to generate a roadmap, it still outperforms the RRT planner in various aspects. These aspects mostly cover the number of intermediate configurations needed for the interpolation. This is due to the nature of RRT, where small equal steps are made in configuration space. The last part that we have tried to implement is the Jacobian-based Inverse Kinematics solver. Due to the lack of time and several issues with the code, it has not been successfully implemented. If it had been successfully worked out, the robot would have been able to move to an object that it could grasp.

7 Discussion

8 Conclusion

References

[1] Stefan Lanser and Olaf Munkelt and Christoph Zierl, *Robust Video-Based Object Recognition Using CAD Models*, Intelligent Autonomous Systems IAS-4, 1995, 529–536.

[2] Dutta, S. and Chaudhuri, B.B., *A Color Edge Detection Algorithm in RGB Color Space*, Advances in Recent Technologies in Communication and Computing, 2009. ARTCom '09. International Conference on, Oct 2009, 337–340, 10.1109/ARTCom.2009.72.

[3] OpenCV, *FlannBasedMatcher*, http://docs.opencv.org/, 29-05-2015.

[4] Point Cloud Library, http://pointclouds.org/, 29-05-2015