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

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 an accurate 3D model which we were able to recreate using a 3D printer. The end result was an object the robot is able to pick up, with known dimensions.

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 1. 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 2, Figure 3 and Figure 4.

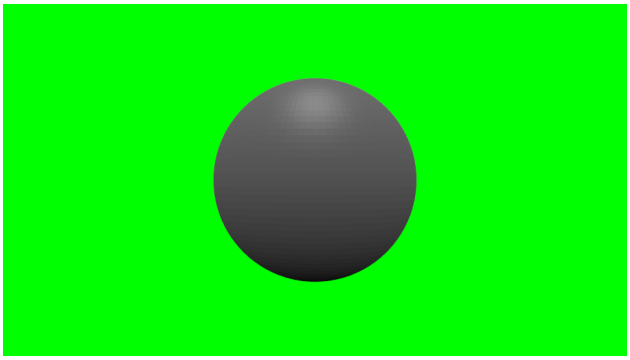

Figure 1: 2D image derived from 3D sphere

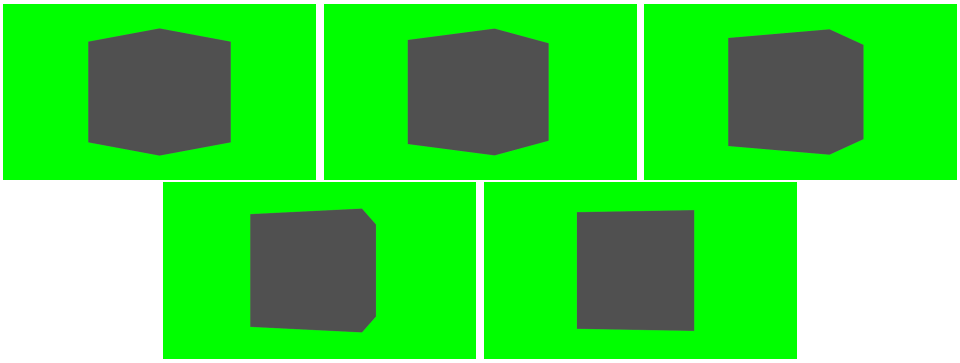

Figure 2: 2D image derived from 3D cube

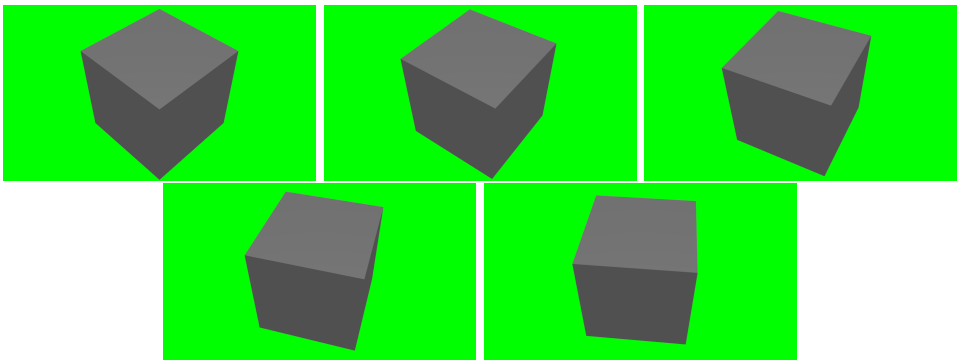

Figure 3: 2D image derived from 3D cube

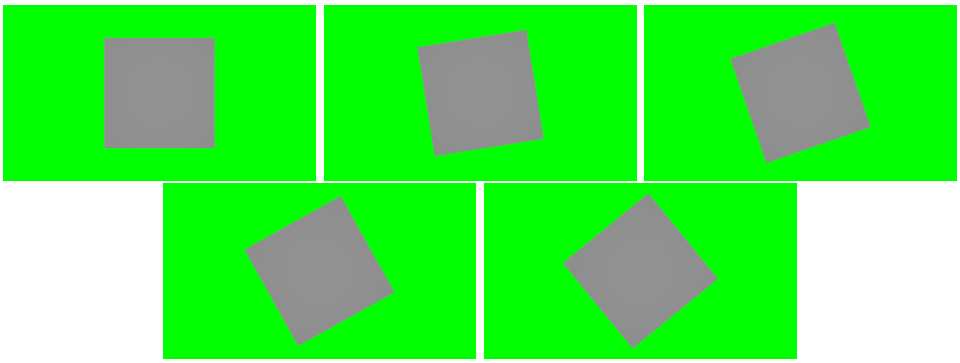

Figure 4: 2D image derived from 3D cube

5.1.2 Contour detection

There are different ways to extract the features of an object. As for this project we used a contour detection to extract the main features from the object. This way we received enough information from the object to compare with, but we didn't overcomplicated the problem. As is described in [1] , there a more complicated but robust ways to better extract features from an object. Though with the current time limit, we didn't pursue this method. As can be seen further in this report, the current method used was, how crude it might be, sufficient for the problem at hand. For the contour method we used OpenCV. This library provided us with tested methods to use and enough flexibility to be tailored for our problem. First the received image is

converted to a gray scaled image. As can be seen in [2] , using a RGB image over Gray scaled image will not increase the accuracy by much. "Almost 90% of edge information in a colour image can be found it the corresponding gray scaled image." [2] So in compliance with the K.I.S.S. [1] principle, we used a gray scaled image which is also a prerequisite to be able to use the standard contour detection build in OpenCV. Figure 5 shows the end result of the contour detection on a 2D converted image from the sphere.

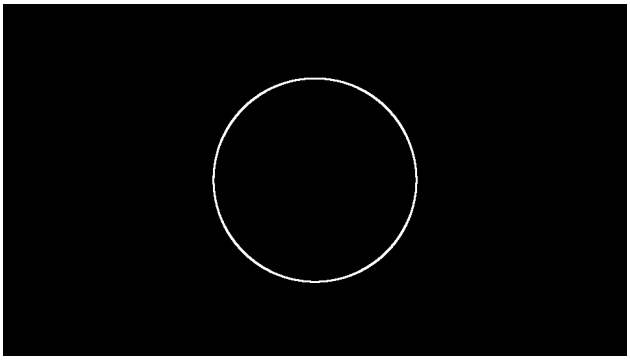

Figure 5: Contour detection from sphere

5.2 Camera

For this project we use a BubleBee2 and an Asus XTion Pro (also known as a "Kinect"). Each of these cameras have their own features which are usable in different parts of the project. Figure 6 shows the result of using the Kinect camera. The cameras shows us the workplace of the robot. The vision part here is to detect the object derived from the video feed. Detecting the object can be done in several ways. For this project we already have a image of the contours of the object we're searching for. This gives us one way of searching for it.

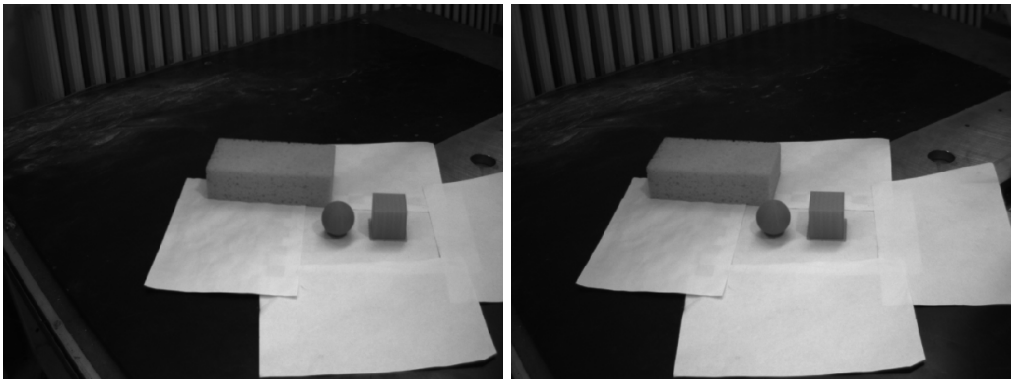

Figure 6: Kinect Camera

5.2.1 Colour segmentation

As the colour of the object is already known, to perform a colour segmentation on the video feed is a logical step to take. This will exclude any other objects that has nothing to do with the object we're searching for.

[1] Stands for "Keep it simple, stupid" or "Keep it short and simple"

If the object was build using multiple colours, using a colour segmentation method would still be valuable as long as there are not too much interference of overlapping colours. This object doesn't have any colour patterns, which you could have also used otherwise, but it consist of one colour. This makes this step a bit easier, though detecting an object with multiple colour 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. This result will then be (morphological) opened and closed in order to remove any small irregularities. Figure 7 shows the end result.

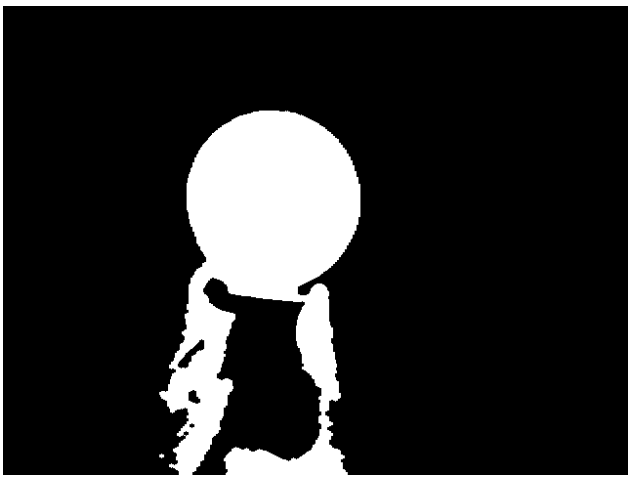

Figure 7: Colour detection

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 where necessary in order to change the video feed to the correct colour space. See Figure 8 for the result of the contour detection after a colour segmentation.

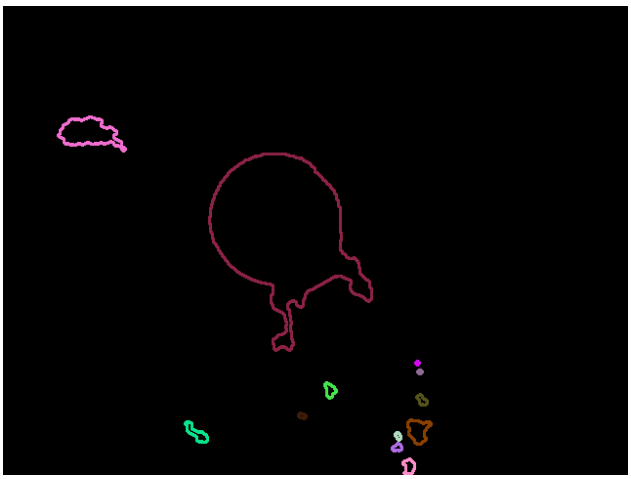

Figure 8: Colour contour detection

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 with a FLANN matcher.

5.3.1 SURF

To detect features from an image, a SURF algorithm is performed on both the object as the scene (i.e. video feed). SURF stands for Speeded Up Robust Features detection method. SURF was inspired from the SIFT algorithm but is several times faster. As for this part we're using a live feed, the speed of the algorithm is crucial. OpenCV has a SURF implementation which we used for this project. We used SURF with an 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 used FlannBasedMatcher function trains the descriptor collection and find the best matches [3] . After finding the best matches, a quick calculation is done between the keypoint distances. In the end only matches who 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 9.

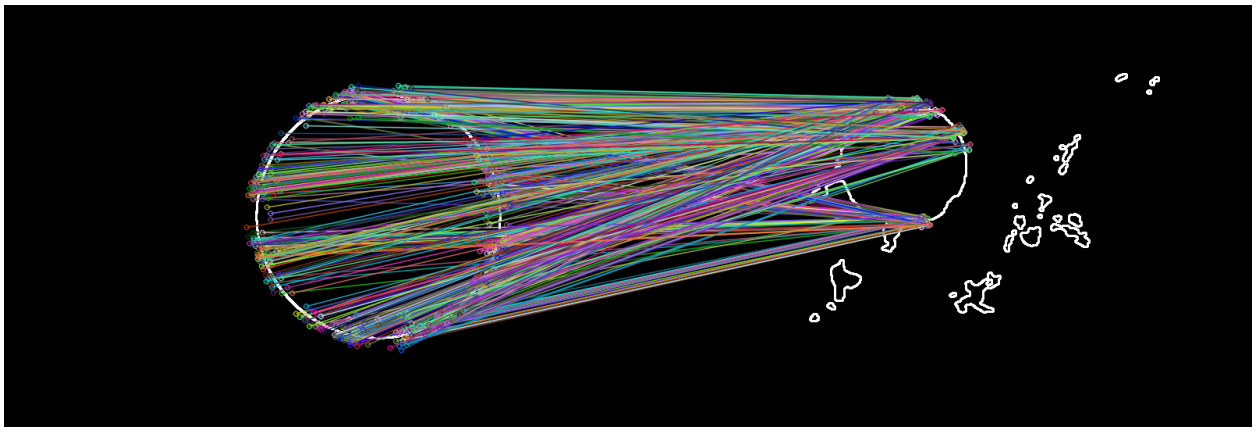

Figure 9: Colour contour detection

5.4 Vision and ROS

The connection with ROS comes down to receiving the video feed from different cameras and publishing information. After this Vision part, combined with the information received from calibrating the robot and camera, we can give the Robotics side coordinate of the wanted object. The Robotics part then can calculate how it needs to move in order to pick the object and place it somewhere else. This is further discussed in the Chapter Calibration and the Chapter Robotics.

5.4.1 PCL

ROS currently has a message-type implemented for Point Clouds. The second revision, which we use, is also the de facto standard in PCL [4]. Using this instead of implementing PCL with use of OpenCV was chosen as this option was already fully implemented and up to standard. On the left in Figure 10 the original image returned from using PCL in ROS, the middle is zoomed-in. For the sake of this document we used a larger (also rectangular) object, as this gave a more visible result back. In image on the right of Figure 10 we added a box to show exactly where the rectangular object was placed. The object in the back is the radiator that is standing against the wall. The PCL can give vital information back regarding the position of the object regarding to the camera. Unfortunately for our project we could not implemented this as a final stage for the vision part. The necessary calculations would be to calibrate the the video feed from the BumbleBee2 with the video feed from the Kinect, as to know how the object in one frame stands regarding to the other. Then only the points within the frame of the found object after the previous Vision part would be returned. Using these combined with the previous calibration of the camera and robot, it is possible where in the world frame the object stands and thus sending the correct coordinates to the Robotics part.

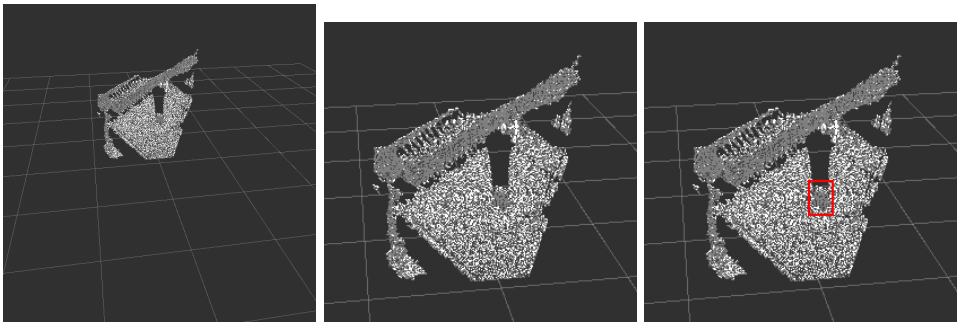

Figure 10: PCL

6 Robotics

In this section there will be a look at two important things. First how to find a good path for the robot to move by. The robot is of course moved by changing the joint configuration, but to avoid hitting object like a wall or the table a path planner need to be made so that it can avoid obstetrical. For that the group have decide to use Probabilist Road Map planner, or PRM planner for short. Secondly the robot need to find the right joint configuration to grasp the known object. The thing there are know of the object is point of where the robot can grasp, and the orientation of the point. There for the use of Inverse kinematics is to be used to decide multiple joint configuration for doing that. The list of different joint configuration then needs to be sorted to find a joint configuration that do not exceeds joint limits, or collided with other object like the wall or table.

6.1 Path planning

As told before, the method used for path planning has been chosen to be a RPM planner. The reason why is that the RPM planner often give a better route than that of the more f.eks. the RRT planner.

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