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

In this report we present our Pick-and-Place project. The goal of this project is to visually find known 3D objects using a pair of stereo cameras and having a robot manipulator interact with said objects. The robot has to move the object from position A to position B. Given the object's position and orientation relative to the base of the robot, the robot should solve for Inverse Kinematics equations to find a suitable interpolation of configuration vectors that facilitates the a translation from point A to point B. Also presented is the calibration of the cameras and robot. In the end, not all goals were met. All individual parts of the projects were sufficiently implemented to produce results and gave good confidence that given more time, the full processing pipeline can be finished.

1 Introduction

This report have been made as a solution for the assignment given in Robotics and Vision 2, at the Southern University of Denmark. The assignment state that a robot needs to pick up an object and place it on a given position. This is what normally is called a pick and place routine. For the assignment a work cell with a robot have been given, both to RobWork and in the real world. Beside the robot itself, two cameras have been given so that an image could be extracted from the work cell. The image from the cameras will then be used in a image processing to find object. This object is then need to be picked up by the robot, which means that a path planner and a picking point need to be compute for the robot. To make sure that the robot and the image are aligned a calibration of the cameras and robot are needed. This makes sure that the coordinates of object and robot are in the same system.

2 The Robot

In this section there will be a look in to the workcell that have been given to make the project to worj with. The Robot that is used in this workcell is know as StaubliRX60. The idea behind looking into this is to minimize error late that can come from badly made Robwork workcells. We also which that the workcell used in the simulation and programing are equal to that of the real world. In figure 1, we have a image of the robot workcell in real life.

Figure 1: workcell in real life

As it can been seen in figure 1 the robot stand in the lower right corner. There i should also be in the workcell for Robwork. In figure 2 is the workcell that have been provided to help simulate the robot. As it can be seen there are different problems that needs to be corrected.

First of all the robot stand in the wrong place of the table, and the robot has the wrong toll implemented on top. Secondly, when using the workcell in Robwork, there where errors do to areas where multiple objects collide or intersect.

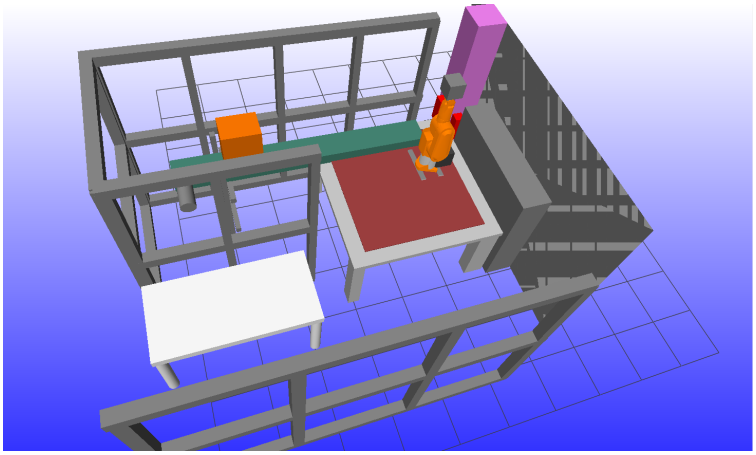


Figure 2: workcell in RobWorkStudio. Be aware that there is some major are that needs to be corrected, f.eks. the position for robot!

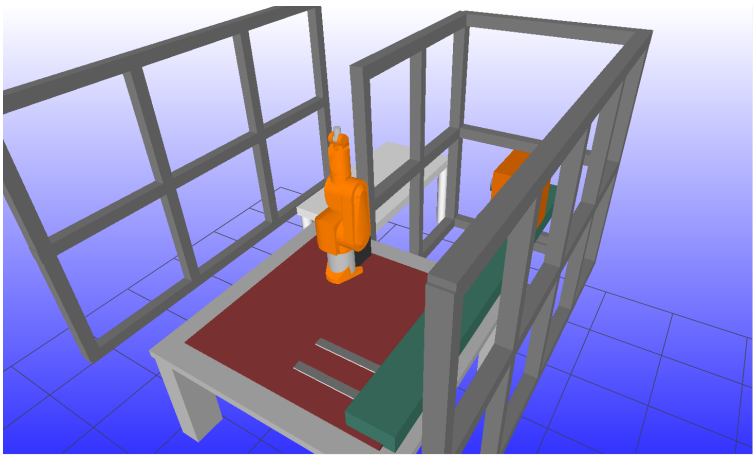


Figure 3: Corrected workcell in RobWorkStudio.

As it can be seen in figure 3 the errors has been corrected, and the robot is ready for use in the simulation in Robwork and in the calibration. Even though that the corrected workcell do not have a toolmount on it, the group still consider the corrected workcell better than the given for the project.

3 Calibration

In this section will be described the calibration process of the given workcell.It represents an intermediate milestone to reach in order to achieve the "pick and place task".In the following pages can be found a complete description of the methodology and the algorithms implemented.Furthermore,before going through the core description of the process,an overview (highlighting only relevant details) of mathematical models and notions regarding the topic will be outlined with the purpose of making the reading easy to understand.

3.1 Methodology

The calibration of an hand-to-eye robot-camera configuration consists in an estimation of the rigid transformation matrix containing the orientation and position of the stereo vision system frame wtr to the base of the robot.The obtained transformation matrix makes possible to describe a 3D point coordinates in the robot base coordinate system,necessary to plan the configurations of the robot to reach the desired 3D point.First step of the process is the calibration of the camera,thanks to which is possible estimate the different parameters of the camera model,such as the camera matrix,distortion parameters and the projection matrix.A robust hand-to-eye calibration is not a trivial task and is a widely discussed topic,several papers describing different algorithms implementation are available and it could represent a suitable topic for an entire project.Since it is not the main goal of the project,before launching into an implementation from scratch of a robot calibration framework,has been preferred checking for existing out-of-the-box solutions.Unluckily,at the moment the only concrete available solution on ROS is represented by "Industrial Calibration" package (still in experimental stage),that is supported only by robot configuration compatible with ROS-industrial and move it based packages.For the given reason has been decided to develop the "hand2eye" package,with the purpose of obtaining an acceptable transformation matrix estimation without focusing all the available time to achieve the project task on the "Calibration task".As technique to estimate the pose of the camera wtr to the robot base has been choosen to use "the minimization of the errors of correspondencies" (more details will be discussed in the following pages).The final version of the "hand2eye" package is the result of a "trial and error" approach,starting with a first "naive" implementation consisted in reducing the problem to a monocamera case using a coloured marker detection to estimate a 3D pose,the algorithm has been discarded due to wrong implementation choice and unacceptable results.

3.2 Camera Model

The camera model represent a mathematical explanation of how a 3D point (x,y,z) in the real world frame is projected into a corresponding image pixel point (u,v).

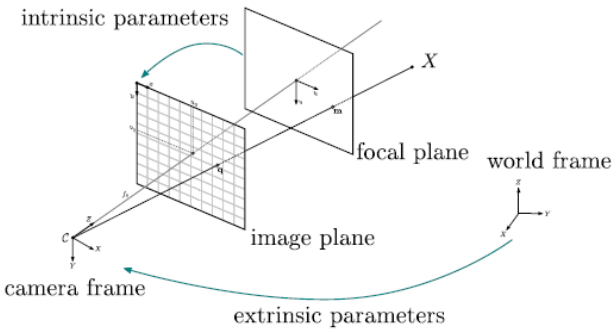


Figure 4: Pinhole Model

Pinhole Model The mathematical relationship between a 3D point and its projection onto the image plane is described by the ideal Pinhole model, see Figure 4,written in matrix forms as $\delta P = Ap$,where:

- δ is an arbitrary scale factor
- p an object in 3D coordinates
- P is the correspondig image projection

- A is camera matrix

The camera matrix elements represent the internal camera parameters,such as the image origin and the focal lenghts of the xy-axis of the camera that are scaling factors to transfer object millimeters into image pixels.The camera matrix represent a simplified model that take count of the "height" and "width" of an object but it does not contain any information regarding the "lenght",i.e. the distance on the optical axis of the camera.

Stereo Vision System Stereo Vision Systems allow to enrich the ideal Pinhole model with the Projection matrix [R,t],in which R and t are the Rotation matrix and the translation vector between the camera frame and the real world frame.The projection matrix represent the extrinsic parameters of the camera.Furthermore,thanks to epipolar geometry, see Figure 5, is possible estimate a perspective projection of a point,allowing the estimation of the distance of a point from the camera frame.

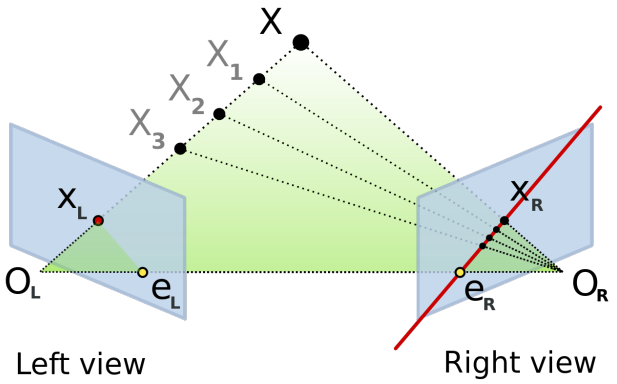


Figure 5: Epipolar Plane

Distortion Coefficients and Rectification Matrix Due to the non-ideality of the camera lens,the Pinhole model is not perfectly followed,this non-ideality arises as distortion of the image,that can be mathematecally modeled traditionally in two main distortion, see Figure 6,radial and tangential. The

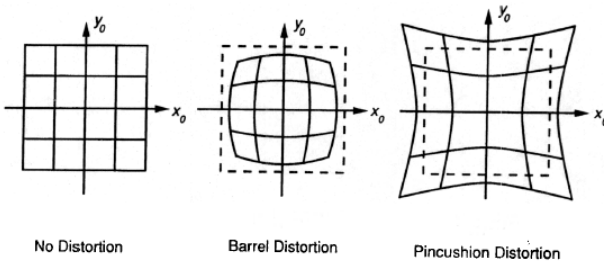
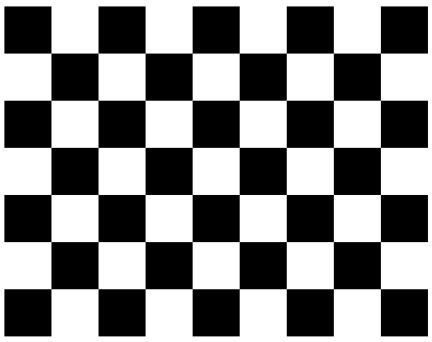


Figure 6: Tangential and radial distortion representation

rectification matrix describes the rotation between two cameras in a stereo vision system,it allows the binocular disparity,that is (as describe on Wikipedia) "the process of relating the depth of an object to its change in position when viewed from a different camera, given the relative position of each camera is known[..]".

Camera Calibration As briefly outlined above,in order to estimate the reprojection of 2D points in the real world,is necessary calibrate the single cameras of the stereo vision system in order to determine the intrinsic and extrinsic parameters of the camera mode.This step has been accomplished using the existing ROS camera calibration package,based on the openCV library.The process consists of capturing different poses of a marker with a particular pattern (usually a black and white chessboard, see Figure 7a) in the camera field, see Figure 7b.



(a) Calibration Chessboard

(b) Camera Calibration Tool ROS

3.3 Hand2eye Calibration Algorithm

Once determined the intrinsic and extrinsic parameters of the stereo vision camera,the camera-robot calibration can be attempted.The implemented method is based on the minimization of the correspondences error.Given two datasets of corresponding 3D points in different reference system is possible,minimizing the least square error,compute the rigid transform that align the two sets of points,that in our case are represented by the centroid of a chessboard respectively wtr to camera frame and wtr to the robot base.The algorithm is summarized in the following pseudo code:

for $N_{points3dA}$ **do**
  compute centroid $\mu_a = (\frac{1}{N}) \sum_{k=1}^{N} P_a$
end
for $N_{points3dB}$ **do**
  compute centroid $\mu_b = (\frac{1}{N}) \sum_{k=1}^{N} P_b$
end
compute covariance matrix H = $\sum_{k=1}^{N} (P_A^i - P_{centroidA})(P_B^i - PcentroidB)$;
decompose H through SVD(H)=[U,S,V];
compute rotation matrix R = VU^T;
compute traslation $t = -R\mu_a + \mu_b$;

Algorithm 0: Minimization of correspondences errors

As explained at[5],to avoid unrelevant results that may be obtained when the determinant of V is negative,an additional check should be included as follow :

if $determinat(R) < 0$ **then**
  Multiply the 3rd column of V by -1;
  Recompute R;
end

Algorithm 1: Determinant(V) signum check

In order to use the algorithm,at least 3 or more points are needed.In this way the error representing the Euclidian distance between the two dataset is minimised,assuring an acceptable estimation of the camera rigid transform.The limit of the method is represented by the fact that in during the capture of the different marker poses the robot has been jogged manually using RobWorkStudio.Particular attention has been paid to the syncronization between the node responsible of broadcasting the marker position vector wtr to base and the one responsible for the position vector of the marker wtr camera frame.in fact,un-paired data describing the same 3D point would affect and weaken the estimation of the rigid transform.

3D Reprojection of the Marker Position in the Camera Frame The algorithm to create the dataset of 3D points of the marker centroid wtr to camera frame,has been completely implemented using OpenCV functions for the most from the Calib3d[7] header,it could have been implemented also in Matlab,but OpenCV has been preferred due to familiarity with the library.Theoretically,known all the cameras parameters (i.e. intrinsic and extrinsic parameters,distorsion and rectification matrix) performing a 3D reprojection of a detected point consists in calling the CV function triangulatepoints(),feeding in the vector of points from the two cameras and the know parameters from calibration.Practically,due to bad optimization of the implemented function on OpenCV/Calib3d,the pipeline of the algorithm has unexpectedly increased,introducing redundacy in the whole process.In fact,after different research in OpenCV user guide and community[6],has emerged that to obtain correct value from the triangulation,is necessary performing again the stereo vision system calibration in order to determine all the camera parameters to feed in triangulatepoints().Apperently,this represents the correct usage of the function.Morover,along the process the different vector of points have to be converted several times between the usage of the fuctions.The encountered issues have considerably incresed the time frame dedicated to the development of the algorithm,due to debugging and evaluation of the 3D reconstruction of points.A qualitative evaluation of the genuineness of the reprojected 3D points,has been performed comparing the obtained normalized coordinates,with an estimated distance along the axis(using a ruler) of the marker center from the origin of the camera frame.Using this criterion,the results of the final version of the algorithm have been considered reliable, see Figure 8.In summary,the pipeline of the process:

- Detection of the marker feature using cv::findchessboardcorner()
- stereo calibration to determine Rotation,Translation,Fundamental and Essential matrices using cv::stereoCalibrate()
- rectification of the image using cv::stereoRectify()
- computation of the ideal point coordinates from the observed points using cv::undistortPoints()
- refining of the points using cv::correctMatches()
- triangulation of the points using cv::triangulatePoints()
- normalization of the humogeneous coordinates and computation of the marker centroid

In the image8 can be noticed that as marker has been choosen a $8x6$ chessboard,due to findchessboardcorner() performance in feature detection in combination with it.During different the algorithm has been able to detect an average of 7 out of 10 marker poses,making the algorithm suitable to generic hand-to-eye configurations calibration,without the necessity of large adjustment to the source code.

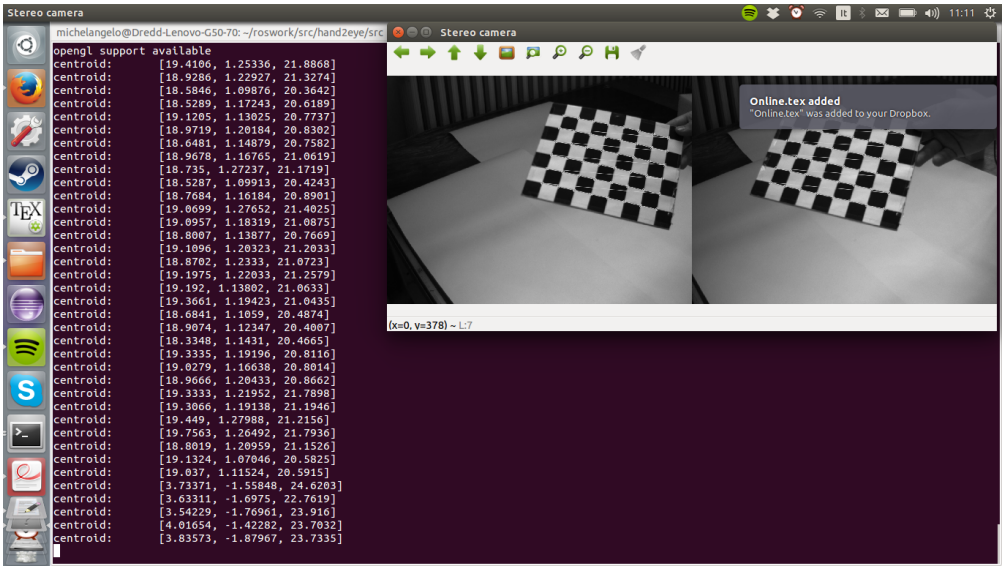


Figure 8: ROS node performing 3D reprojection of the chessboard center

Marker Position in Robot Base Frame The second node of the "Hand2eye" calibration package,consist in an easy algorithm implemented using RobWork library.For this reason few lines are needed to describe it.It uses the provided workcell and a custom ROS client,that broadcast the continuesly (until the node is killed) the joint configuration of the robot.The workcell then is loaded to allow the computation of the trasformation matrix of the tool wtr to the base frame by setting a new state of the simulated workcell every time a new joint configuration is available.

3.4 Hand2eye ROS Package

Structure As briefly outlined in the previous paragraphs,the current Hand2eye calibration package,consists in two nodes: the getP-camera_node,that once subscribed and syncronized the "/stereo_camera/left/image_raw" and "/stereo_camera/left/image_raw",it converts them to opencv image and performs the 3D reprojection of the points and saving the coordinates on txt file for an offline processing of the datasets.The "getP_target" is a ROS client that,until is killed,continuously sends the service request "GET_JOINT_CONFIGURATION" and uses the service response,containing the joint configuration,to compute the position vector of the marker wtr to the base frame,likewise the first node it saves on txt file the marker coordinates.The Rigid transform estimation is performed using a c++ program that implement the error minimization algorithm outlined at the beginning of the section.

Practical Issues The issues regarding the Calib3d header of OpenCV has absorbed most of the the available time,delaying the development and the schedule.In fact it has not been possible making concrete tests on the entire package and evaluate the genineness of the final results.The beginner level of knowledge on ROS has slightly affected the development process.

Improvements Even if it has not been possible to complete and test the Hand2eye calibration,a final version of the package has been designed but not implemented:the main idea is to publish the points datasets from the existing node as custom messages on new topics that could be subscribed and syncronized by a ROS node implementation of the rigid transform estimation program which,in turn,would make a new estimation

every time a new point in the datasets is available and publishing it on a new topic making it at disposal for further uses.

3.5 Conclusions

The encountered issues,as expected,have shown the non triviality of an hand-to-eye camera-robot calibration.But considering the availability of few calibration packages,most of them in experimental stage,could be an opportunity to complete and optimize the designed Hand2eye package that,even if in early stage,could be an easy and adaptive solution to generic hand-to-eye set-up,that could be shared as entry-level calibration package for amateur applications.

4 Vision

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

4.1 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 9.

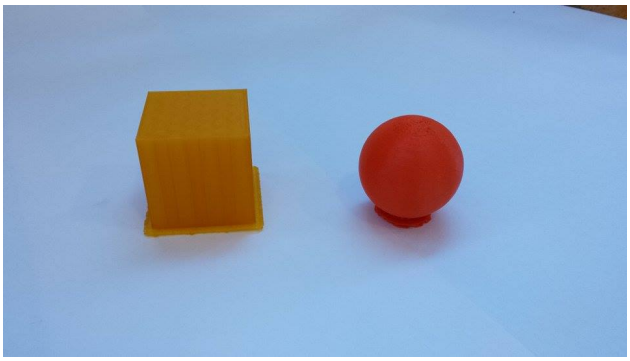


Figure 9: Objects used

4.1.1 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 10. 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 11, Figure 12 and Figure 13.

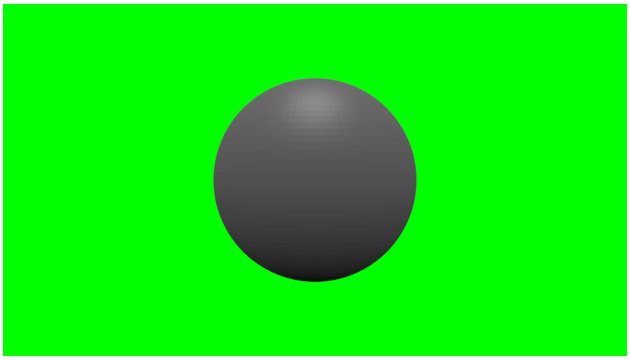


Figure 10: 2D image derived from 3D sphere

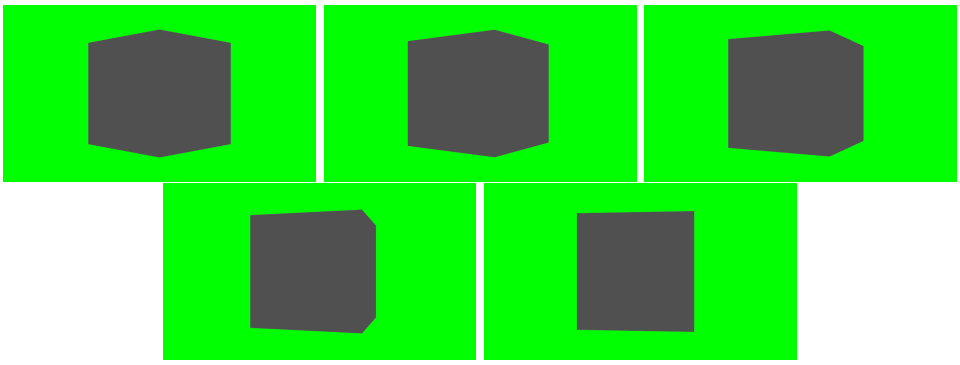


Figure 11: 2D image derived from 3D cube

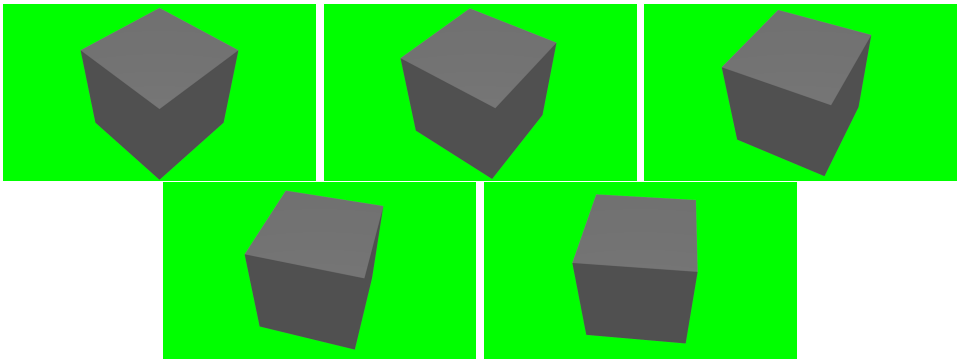


Figure 12: 2D image derived from 3D cube

4.1.2 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 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

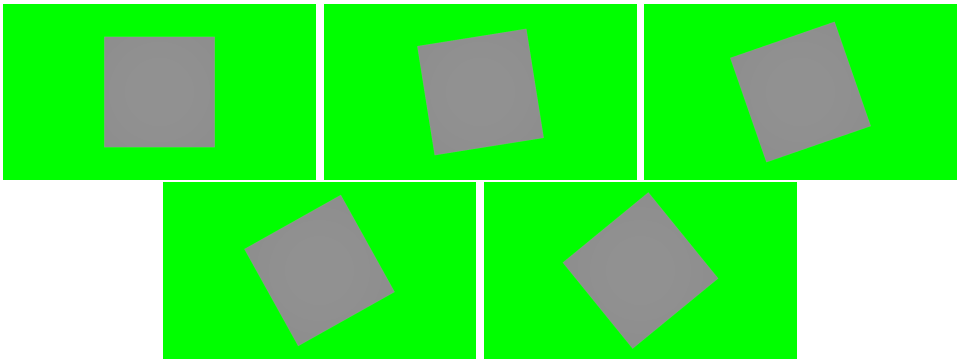


Figure 13: 2D image derived from 3D cube

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 14 shows the end result of the contour detection on a 2D converted image from the sphere.

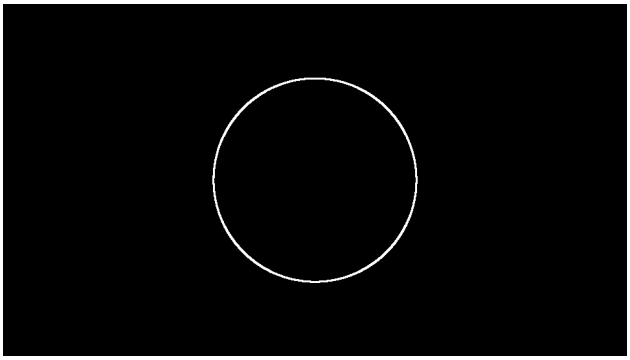


Figure 14: Contour detection from sphere

4.2 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 15 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.

4.2.1 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

[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.

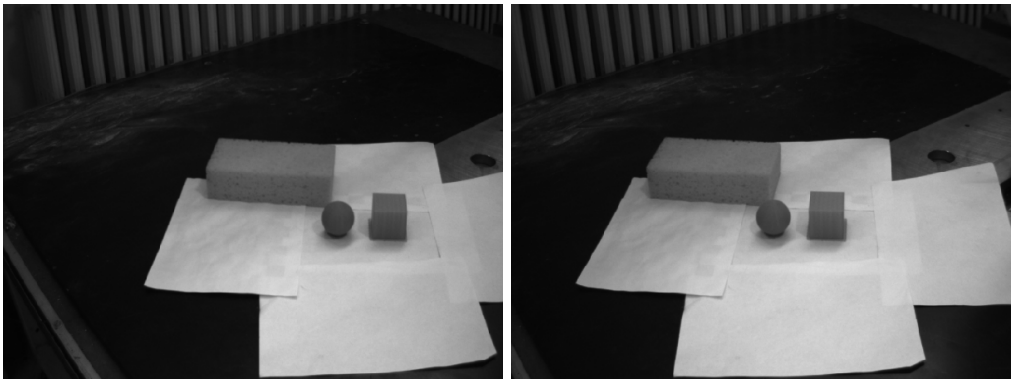


Figure 15: Kinect Camera

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 16 shows the end result.

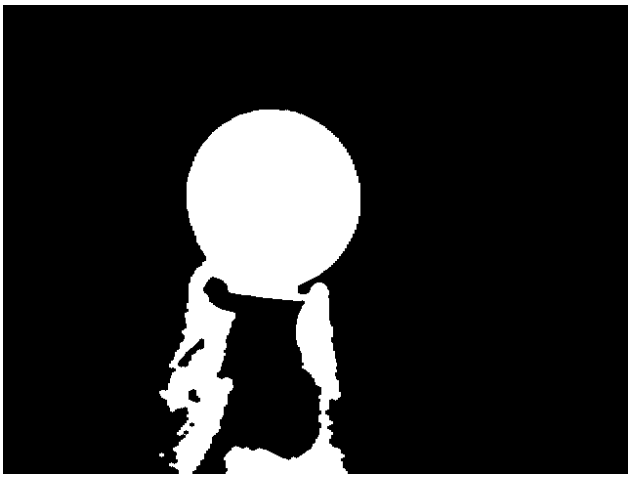


Figure 16: Colour segmentation

4.2.2 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 17 for the result of the contour detection after a colour segmentation.

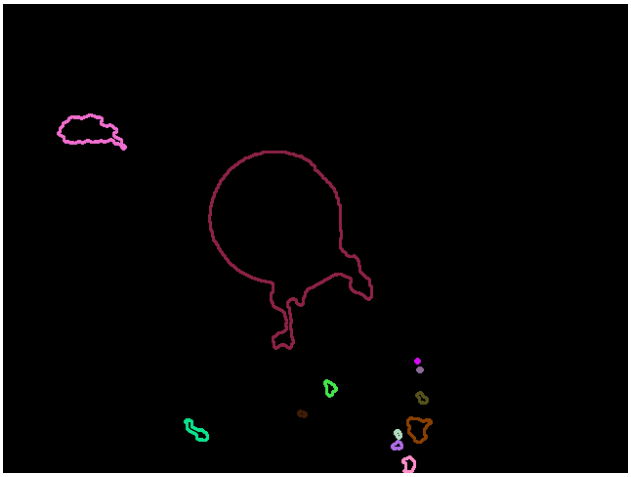


Figure 17: Colour contour detection

4.3 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.

4.3.1 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.

4.3.2 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 18.

4.4 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 5.

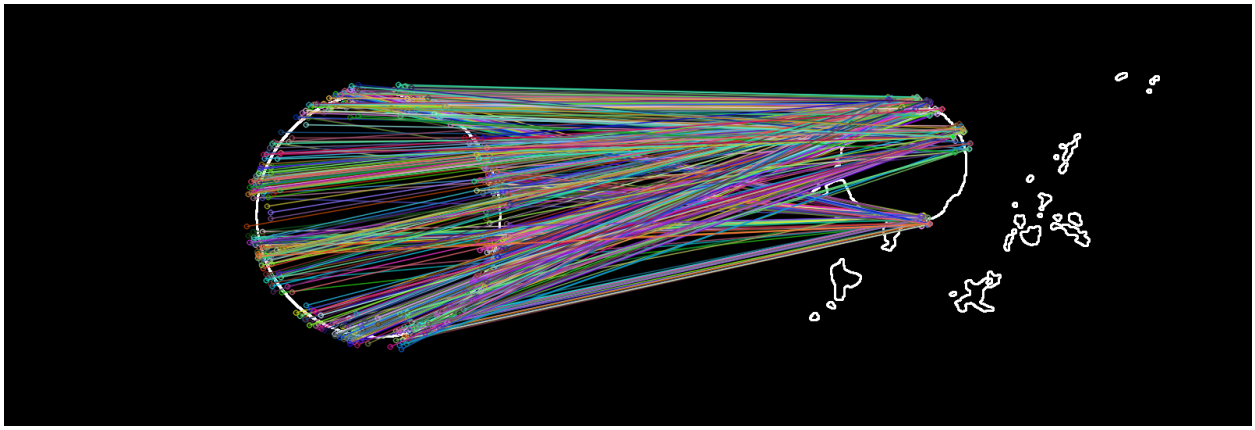


Figure 18: Colour contour detection

4.4.1 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 19 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.

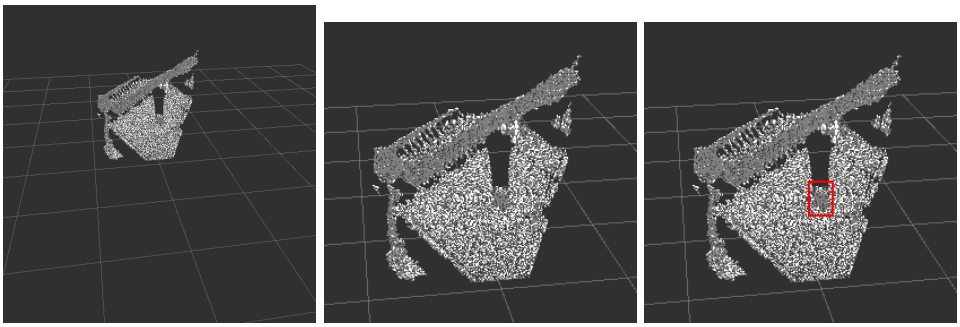


Figure 19: PCL

5 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.

5.1 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 5.2 there is a comparison to show the difference in calculation time between RRT and PRM.

5.1.1 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.

5.2 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.

5.3 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.

5.4 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.

6 Conclusion

For the vision part of the project, a lot of the implementation goals have been met. The system is defined by its base necessities in that it can segment live camera feeds based on colour and detect known object contours within the camera frames. Given the time, PCL support could be implemented as well as general robustness improvements. This would fully complete the whole visual processing pipeline which would feed data into the Robotics part of the system.

The calibration system, as presented was met with a few issues in implementation. However, the path that is taken shows great promise to be a user-friendly tool for camera calibrations. Work could be done to make the solution more generic, this could be the task for a continued research project.

The Robotics part of the system has met one of its goals. The PRM path planner has been implemented and tested. Tests have proven it to be more efficient compared to a pre-made RRT implementation. The Inverse Kinematics solver was not successfully implemented, this means that the robotics pipeline stops at generating roadmaps that span the possible configurations of the robot within the workcell. Given more time, the Inverse Kinematics problems could be solved in a variety of ways to ultimately reach the goal of grasping known objects.

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

[5] Finding optimal rotation and translation between orresponding 3D points. `http://nghiaho.com/?page_id=671` 23/5-2015

[6] Answered question about triangulatepoints() on OpenCV community `http://answers.opencv.org/question/3275/is-triangulatepoints-outputing-rubish/` 28/5-2015

[7] OpenCV user guide `http://docs.opencv.org/doc/user_guide/user_guide.html` 28/5-2015