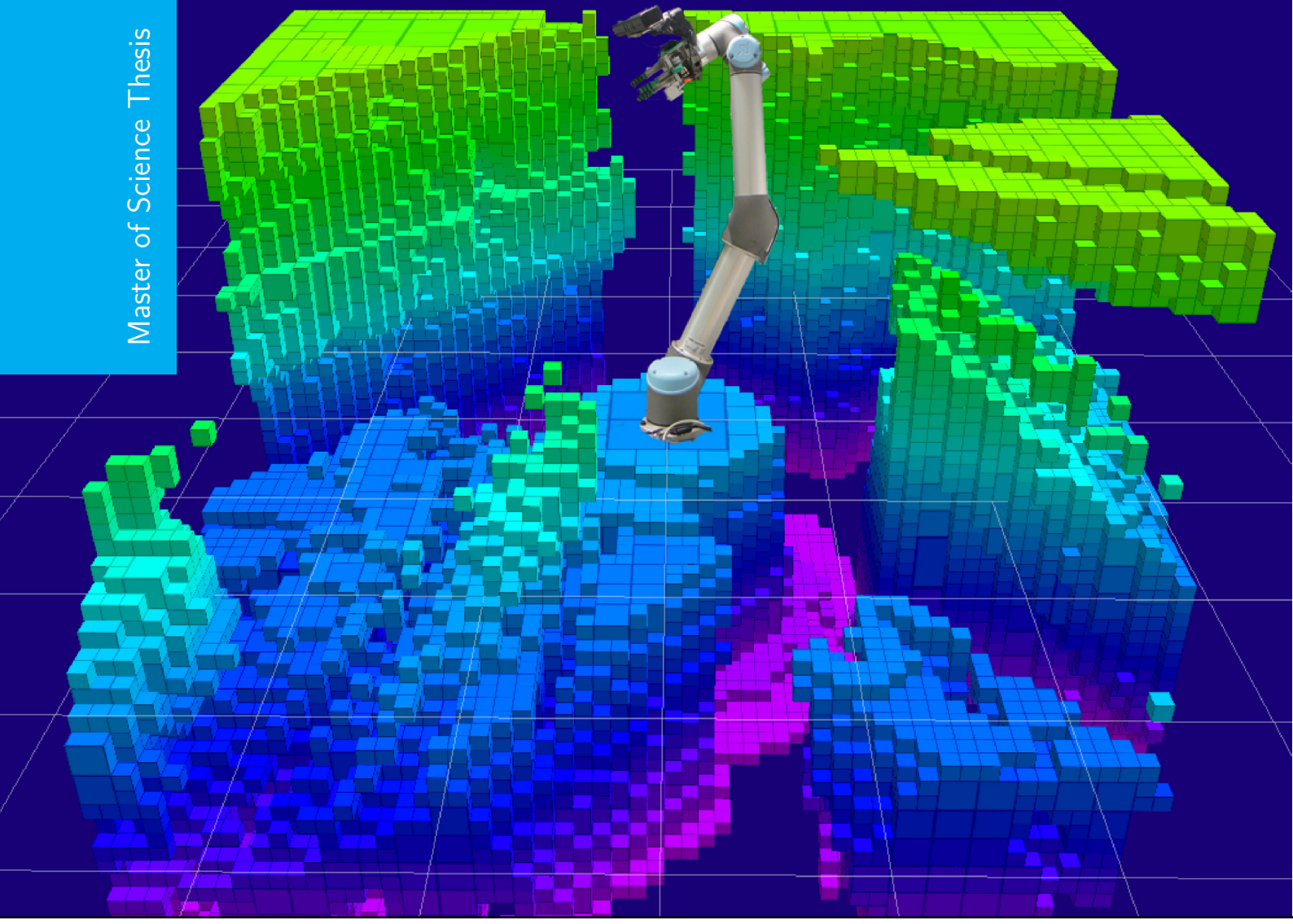
Automatic Extrinsic Calibration and Workspace Mapping

Algorithms to Shorten the Setup time of Camera-guided Industrial Robots

L.N.M. Middelplaats

Master of Science Thesis



TU Delft Delft University of Technology

Mechanical Engineering

Automatic Extrinsic Calibration and Workspace Mapping

Algorithms to Shorten the Setup time of Camera-guided Industrial Robots

Master of Science Thesis

For the degree of Master of Science in BioMechanical Engineering at Delft University of Technology

L.N.M. Middelplaats

June 11, 2014

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
BioMechanical Engineering (bme)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Automatic Extrinsic Calibration and Workspace Mapping

by

L.N.M. Middelplaats

in partial fulfillment of the requirements for the degree of

Master of Science BioMechanical Engineering

Dated: June 11, 2014

Supervisor(s):

Prof.dr.ir. P.P. Jonker

Dr.ir. M. Wisse

Reader(s):

Dr.ir. A.J.J. van den Boom

Abstract

In small to medium enterprises (SME), industrial robot arms are not used very much despite the fact that they offer a large potential to increase the competitiveness. The problem is that to be effective in the SME sector, robot systems should be more flexible. To be economically effective a robot arm should be used with multiple different tasks which are usually in different locations in the company. Therefore, the robot should be able to be reconfigured quickly to the new task and location giving a low robot set-up time. Prevention of collisions with obstacles in the different locations and the extrinsic calibration of sensors on the robot are two important and time consuming tasks during set-up of the robot. Automatic generation of a 3D map of obstacles near the robot would decrease set-up time and therefore increase the flexibility. The calibration of the camera is also a tedious procedure which could be automated. Therefore, in this thesis a system is proposed which extrinsically calibrates a camera on the robot arm, and uses the calibrated camera to scan the environment in a safe way to create a collision map. Several calibration methods are investigated. The method created by Tsai et al. gives the best results and is selected. The calibration system works by estimating the position of the camera relative to a checker-board pattern from multiple points and matching this with the robot orientation. From this the camera position and orientation on the robot are calculated. In this thesis the system is tested in simulation and on a Universal robots UR5 robot arm. To safely map the environment without colliding in to obstacles which are not seen yet a Next Best View (NBV) viewpoint generation system is proposed. The system generates viewpoints for the camera which add the most new information to the map. A virtual wall in the map round the robot represents the unknown space, which prevents the robot to move into unknown space. By viewing the unknown space with the camera the unknown space is cleared. As monocular mapping algorithms are not available yet, a 3D camera is used for data acquisition. Data is stored in an OctoMap system, which is a memory efficient, discretized probabilistic storage system. This system is also implemented on the UR5 robot arm. The calibration system works successfully. In addition, we concluded that noise in the camera pose estimation system is the main limiting factor for achieving high precision. The collision map system has been implemented correctly and also functions successfully. For this part of the system, we conclude that the main limiting factor for the processing speed could be removed if the mapping system would be integrated in the NBV software.

Table of Contents

List of Figures

List of Tables

"Learning never exhausts the mind."
— *Leonardo da Vinci*

Chapter 1

Introduction

Robotics are used a lot in large scale enterprises for production. This enables them to produce at low cost with a constant quality. One would thus expect that given these advantages the smaller enterprises would also use robot systems. However, from the literature review it was found that the Small to Medium Enterprises (SME) are not using very much robotic systems. Usage of robotics could improve the production and help stay completive. The problem is that current robot systems are not sufficiently flexible enough. Although robotic systems have become much more affordable they still represent a large investment. Therefore, to be cost efficient the robot system needs to be usable for different tasks, which are usually at different places in the factory. This requires repeated reconfiguration, which is very time consuming. If this is done often it cuts the effective working time and thus increases the payback time of the robot. In conclusion, the key to increasing the uptake of robotics in SME's is to reduce the time required for reconfiguration.

Vision systems mounted on the robot and in its surroundings can make the system a lot more flexible. If the robot for instance would be able to scan for obstacles in the surroundings itself then the changes in setup and location would not be such a problem. The robot can be moved from location to location for different tasks. Or changes in the setup like different conveyor belts or machines that need to be tended could be automatically detected. If the robot could scan its new environment again and update its collision map this could save time.

Vision systems can also be used to determine the location of the product that needs to be grabbed. With this the exact location of the product does not need to be programmed. This gives less constraints to environment, as it is then not necessary to design a guiding system to put a product precisely in a spot so that the robot can grab it. By using vision systems the product can be in different locations in different orientations. This makes the system a lot more flexible and possibly also more robust.

However for both these tasks the location of the camera needs to be known to get a precise location of the surroundings or objects in it. Therefore it is necessary to calibrate the location of the camera. Manual recalibration is a tedious procedure. So it should also be automatic so that this part can also increase the flexibility of the robot system. It could be beneficial

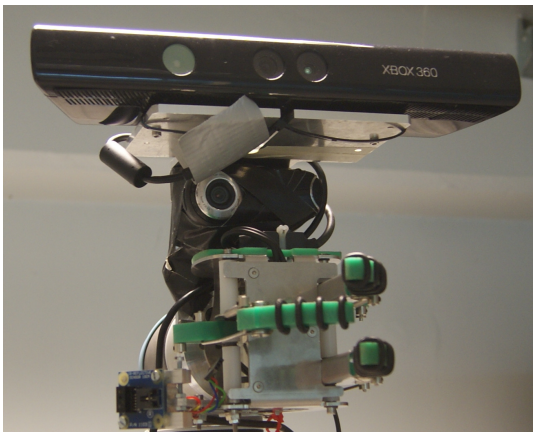


Figure 1-1: The Kinect sensor (top) and a Logitech camera taped to the end effector of the UR5 robot arm. Below it a Fetch Hand from Lacquey.

for some applications that the camera is placed on an other part of the robot or in a different location. For a grasping task for instance the camera is probably placed near the end effector where a gripper is placed. But if the camera is used for collision prevention for instance, then the camera could be placed on the first link (near the base of the robot). For the flexibility the time needed to change between these tasks should be as low as possible. Automatic calibration would also be beneficial in prototyping robot set-ups or in research. Here camera's are sometimes taped to the robot, see for instance figure 1-1.

1-1 Thesis Goals

The goal of this thesis is to contribute to fast reconfigurability of industrial robot arms by automating two parts of the installation process:

- Develop a system that can help with fast (extrinsic) calibration of a camera on a robot arm.
  - Accuracy and precision
    The combined error should be below a centimetre so that the camera can be used to grasp something using the camera with a Lacquey gripper.
- Use the camera to automatically map the surroundings of the robot for obstacles.
  - Mapping the surroundings (workspace) of the robot arm should be done in a safe way, during scanning the robot should not collide with an unknown object.

Here we define a *Camera* as: a monocular Red Green Blue (RGB) camera
And a *Robot arm* as: for instance a 6 Degree Of Freedom (DOF) manipulator like the UR5. Method should be applicable to more or less DOF robot arms.

1-2 Test system

The created system will be implemented on a robot system.

1-2-1 Robot arm

The resulting system is tested on an Universal Robots UR5 robot arm. This is a relatively new robot arm that has a several advantages over traditional robot arms which makes it a good candidate for usage in the SMEs:

- The UR5 does not need safety fences around it as it operates at low speed and has a form of collision detection. This has the advantage that there are no modifications to the workplace necessary. And the robot can also be used among factory workers. Automating the whole production process or a single process step could not be economically feasible. But in this case a part can be automated or a mixture of human workers and robot arms can be used.
- Easy to move around. The robot is light so it can be placed on a moveable table which can be moved by a single person. The robot is powered by normal wall wart connection, it does not require lots of power or a specialized power connection. This makes it possible to move the robot around and use it for a variety of tasks. Traditional robots are fixed to the floor, the whole environment needs to be changed if another tasks or operation has to be done.

1-2-2 Control software

From the literature review it was found that current robot control software is not very flexible. It cannot be changed easily, generally it can only be used with one robot arm brand and cannot easily interface with new sensors. By using an extra software layer, a middleware between the operating system of the user and that of the robot the flexibility can be increased. Open source robot middlewares can interface with different robot brands and support a lot of different sensors. They offer also support multiple robot systems from different brands. From the literature review the Robot Operating System (ROS) was found to be the best candidate for this. Therefore the system will be created on top of this middleware. The UR5 robot has also support in ROS.

1-2-3 Camera system

A monocular RGB camera system would be ideal for SME usage. Compared to other camera systems like depth camera's monocular cameras are relatively cheap and the technology is more standardised and matured. This type of camera can for instance be used for locating objects that need to be moved by the robot.

For the creation of depth maps however monocular camera's are more difficult to use. There is however a system created by Newcombe et al. that can do mapping with a monocular camera using structure from motion[10]. The software for this is however not available yet.

And with structure from motion it would mean that the camera has to move to see depth. But when checking for possible collisions this could mean that the robot could collide with an obstacle which it can not see yet. Therefore it was chosen for the collision mapping part to use a depth camera.

Depth camera

Several depth camera systems exist which can be used for workspace mapping. A short overview of available depth sensors will be given next.

Triangulation based sensors This class of sensors uses a light source that projects a point or a pattern on the object and looks at the reflection with a camera. By using triangulation the distance can be calculated. This approach is similar to stereovison, only instead of using two cameras one camera is replaced with a projector. The usage of this type of sensor has increased a lot after the introduction of the Microsoft Kinect. As this sensor was developed to be used with the Xbox game console the sensor is relatively cheap compared to existing depth sensors. Due to the popularity of the Kinect in research it has good driver support and documentation. Disadvantage of this projection method is that there could be shadows in the image, places where the depth cannot be measured[11]. Because there is an offset between camera and projector it is possible that either the view of the camera or projector is blocked. This creates the shadow in the image. Highly reflective surfaces can also create gaps as this can create over exposure of the image[12].

Time of flight sensors (ToF) This class of sensors consists mainly out of laser scanners and time of flight camera's but radars and ultrasonic sensors also use the same principle. These sensors use the time required for a light pulse to travel to the object and back. Advantage of these sensors is that they are more compact compared to the triangulation based sensors as there is no baseline required between the projector and the camera. The minimum distance at which they can be used is generally also lower that of for instance the Kinect. But like the Kinect they can also have problems with reflections. These sensors were also quite expensive but the costs are decreasing. The new Kinect coming with the Xbox One console will also be a ToF camera.

Stereovision Stereovision works by looking at similarities in the images of two cameras which have fixed distance between them. The different locations of a feature in both images gives a measure for the distance. Stereovision has the problem that enough features or intensity patterns have to present in both images. This gives problems with objects which have don't have much features on them like plates. In a industrial setting with metal machines this could give problems.

Selected sensors

As 'normal' camera a Logitech 9000 webcam will be used. Because of the availability and the good support of the Kinect in ROS this sensor will be used as depth sensor.

It also possible to use the other sensors if the limitations given above are taken into account.

1-3 Thesis outline

This thesis is divided in to three parts. Each part is dived into a literature review, a simulation part and a real world test. The parts are:

- **Camera pose estimation**
  To determine the camera location on the robot the camera location relative to a fixed point needs to be known. Therefore the first part will discuss the different camera pose estimation methods. The different implementations will be discussed, tested in simulation and with a real camera on a robot.
- **Extrinsic calibration**
  Next the methods for estimating the position of the camera on the robot will be discussed. These methods use position of the camera relative to a fixed point as discussed in the previous part. This is combined with the pose of the robot to get an estimate of the location of the camera on the robot. This is also tested in simulation and on a real robot arm.
- **Workspace Mapping**
  The last part discusses the methods that can be used to get a 3D map of the obstacles in the direct environment of the robot. This should be done in a safe way. So that when building this map the robot doesn't collide with an unknown obstacle. The extrinsic calibration of the camera as obtained in the previous part is used for the map building. With this the map building software knowns where the camera is in the map.

Part I

Camera pose estimation

Chapter 2

Camera pose estimation

To get the location of the camera on the robot extrinsic calibration methods will be used, which will be discussed in the next chapter. Extrinsic calibration methods generally require the position of the camera relative to a calibration target to be known. Therefore methods to get the camera pose relative to a calibration target will be studied. In figure 2-1 an example of such a system is given.

For the hand-eye algorithm the motion of the hand and the eye (the camera) need to be known. The hand motion can be determined using the encoders of the robot. Determining the motion of the camera is more difficult.

With a monocular camera displacements in the x and y directions (image plane motions) and rotation round the z-axis can be estimated relatively easily. In plane motions give a large change in the image so this can be detected relatively well compared to the other motions. Movement in the z direction or rotations round the x- and y-axis are more difficult as the change in the image is small. The displacements and rotations have an arbitrary scale. By using a calibration target with known dimensions the rotations and displacements can be quantified.

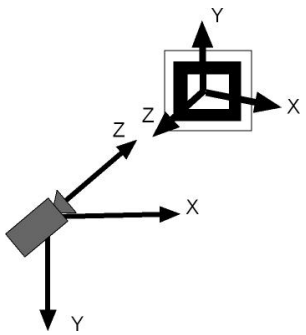


Figure 2-1: Overview of the camera pose estimation system. The system estimates the distance and orientation to the local coordinate system of the marker (the black square). Image from the ARToolKit Website[1].

2-1 Types

Several types of camera calibration targets exist. We will focus here on planar targets as they can be easily used in the workspace of the robot. They can be easily printed with a standard printer and fixed to a surface. Planar targets can be subdivided in to repeated pattern types (like checkerboard patterns) or non repeating patterns like Augmented Reality (AR) markers.

2-1-1 Checkerboard targets

Checkerboard patterns are usually used to calibrate cameras. By getting an image of a checkerboard from multiple viewpoints or checkerboard locations the lens parameters can be estimated. The sharp edges of the checkerboard are used to calibrate the intrinsic parameters of the camera. Other patterns like for instance circles are also used. The checkerboards can also be used to estimate the pose of the checkerboard relative to the camera.

2-1-2 Augmented reality (AR) markers

AR markers are also called fiducial (individually identifiable) markers. Compared to checkerboard patterns they have the advantage that a code or number can be assigned to the different patterns. Or even a text can be encoded in to the pattern. It is also possible to perform calibration using markers[13]. As the markers can have an identification number it is also possible to use multiple markers which can have a specific distance to each other.

2-1-3 Selection

With AR markers it is possible to use multiple markers with a specific distance to each other. This could enhance the precision of the system or it would enable the system to work if one or more markers are occluded. Fiducial markers are used a lot in Augmented reality (AR) to determine the camera movements relative to the marker. As there is a lot of research done is this area the best algorithms for camera pose estimation are probably used in this area. Therefore AR markers will be used.

2-2 AR marker pose estimators

Several systems for marker pose estimation exist. There are for instance:

ARToolKit (2000) [2] This is the most popular system which has been used a lot in research and commercial applications. ARToolKit uses binary thresholding. The markers can be arbitrary images than are framed in a black square. It is available on several platforms and is also ported to for instance cell phones. It is also available as a ROS package. It has a GPLv2[1] and a commercial license.

[1] An open source license that requires that the source code of programs derived from software also has to be released.

ARTag (2005) [14] Is based on ARToolKit. It uses edge detection instead of binary thresholding. Has a GNU General Public License (GPL) license, it is however not downloadable any more.

ARToolKitPlus (2007) [15] A rewritten version of ARToolKit in C++ making it more memory efficient and faster. It also has several improvements like automatic thresholding and it can track multiple markers. It uses a more advanced pose estimation algorithm, the Robust Pose estimator from a Planar target (RPP). It uses binary coded markers, GPLv3 license.

Studierstube (2008) [16] Developed by the same group that developed ARToolKitPlus (Graz University). It is supposed to be faster and less memory intensive than ARToolKitPlus. Features six marker types, two pose estimators and three thresholding algorithms. The software is closed source, support seems to have stopped since 2008.

AprilTag (2010) [17] Uses modified lexicode for markers and uses line detection for marker detection. Has an open source license.

Aruco (2011) [18] OpenCV based. Can track up to 1024 distinct markers and has a BSD license[2]. As it is relatively new there are no reports available which compare it to other systems.

This a small overview based on popularity. An overview of more methods is given in [19], [20] and [21].

The system should be open source. This makes it possible to adapt the system to this specific application. If the code is open source it is also easier to make a Robot Operating System (ROS) package of the created system on which other applications can be created. It should track 6 Degree Of Freedom (DOF) and have a high accuracy. The precision should also be high, although the hand eye algorithm can compensate a bit for this by using multiple measurements for this.

2-2-1 Performance

To determine which is the pose estimator system and what the expected performance will be, we will look into comparative studies. Main criteria will be the robustness to different lighting conditions and the accuracy of the estimated pose.

Lighting conditions In [22] ARTag and ARToolKitPlus were compared on lighting conditions and dealing with occlusion. ARTag could deal better with partial occlusion of the marker. ARTag could worked also better if the light intensity varied over the image.

[2]Open source license which allows that the software may be incorporated in proprietary closed source software.

Accuracy In [23] the pose estimators of ARToolKit and ARToolKitPlus were compared. The first uses Kato et al.'s algorithm while the latter uses Robust Pose estimator from a Planar target (RPP). The RPP method gives the most uniform results and is according to this study the best. In this study the performance was evaluated in a simulation (unfortunately no information is given about the distance, marker size or camera model). The RPP method has a maximum error (Euclidean distance) of about 6 cm while the method from Kato et al. has a maximum error of 1 metre. Distance to the marker is however not specified. From the test it also follows that using more markers and markers that are more away from each other provide a better result.

In [24] an evaluation for ARToolKit was performed. A webcam with a resolution of 640x480 pixels was used with a marker of 5.5 mm. The distance to the marker was 20 to 100 cm. The error increases for the distance, between 20 and 70 cm the error was low according to the study. Although the error was about 7 cm at 70 cm distance. Which relatively high compared to the other studies but the marker used in this study is very small. For a larger marker and a camera with a larger resolution this error could however be lower.

In [25] a (very) basic accuracy test of ARToolKit was performed. Using a 640x480 pixel firewire camera with a 20x20 cm marker the accuracy was evaluated. Only the X and Y measurements were evaluated. At 2.4 meters the error was 12 percent for the X axis while 18 percent for the Y. A maximum error of $\pm$ 27 millimetres at 2.5 meters was given. An upper limit for detection is 2.5 meters. The error also seems to fluctuate with the angle at which the camera is held.

In [20] ARToolKitPlus was tested with an 640x480 pixel grayscale camera with 30x30mm markers at 0.5 meter distance. Here an accuracy for the x,y direction is listed as 5 mm and 20 mm in the z direction. The setup is not calibrated but ARToolKitPlus can undistort the image this using the camera parameters. But according to the author there could be still some distortion present. As the origin is not exactly known he uses relative measurements between the markers. The use of grayscale camera instead of a color camera helps. As is it is not necessary to interpolate the colors in the Bayer filter[26].

On the ARToolKit website[3] and [2] some benchmarks are given although without any measurement conditions or marker size. In figure 2-2 an overview of dependence between the error, the distance and the rotation is given.

In [17] Apriltag and ARToolKitPlus were compared. Tests were done use a 400x400 pixel camera with a focal length of 400 pixels in simulation, the used marker size is however not given. In a distance test the detection rate drops to 50 percent at 25 meters for ARToolKitPlus while Apriltag works till 50 meters and seems more accurate.

2-3 Discussion

According to [19] circular markers offer better pose estimation characteristics than square ones. However according to his literature review: "there are no tried and tested circular marker systems which do not require multiple markers and which are free to use and open-source". But this could be a recommendation for the future. A more through comparison with

[3] http://www.hitl.washington.edu/artoolkit/documentation/benchmark.htm

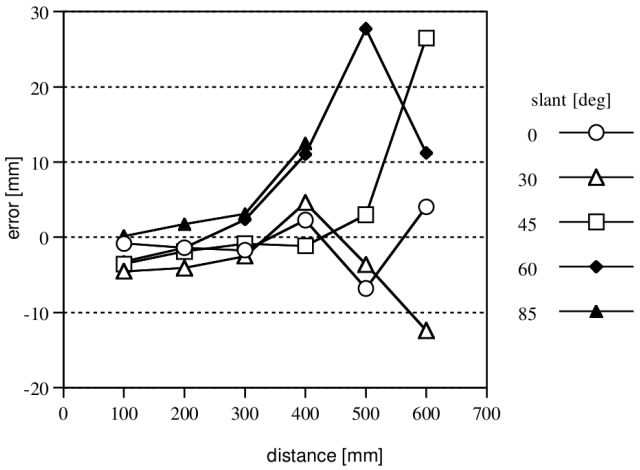


Figure 2-2: Tracking Error versus Range and rotation angle. A 80 mm marker was used incombinatin with a 263x234 pixels camera. Reproduced from [2].

Table 2-1: Overview of the error reported in different studies.

Study	System	Resolution	Marker size [mm]	Distance [m]	Error [mm]
Abawi [24]	ARToolKit	640x480	5.5	0.2-1	70 (at 0.7 m), approx. 100 (at 1 m)
Wayne [25]	ARToolKit	640x480	200	0-2.4	27 (at 2.5m)
Kler [20]	ARToolKitPlus	640x480	30	0.5	X,Y= 5 Z=20
Andersen [23]	ARToolKitPlus	-	-	-	X=50, Y:35, Z=20
Olson [17]	ARToolKitPlus Simulation	400x400	-	50	10.000

experiments of all the algorithms would also be a recommendation. Because several studies do not list essential parameters like marker size or distance from the camera to the marker. A metric that incorporates marker size and the resolution of the camera would make such study more useful general applications.

A checkerboard can contain more edges than a same sized AR marker which could make this more robust. This should also be tested.

2-4 Conclusion

An overview of the studies and their results is given in table 2-1. It can be seen that is difficult to generalize the results as the setups are different and different metrics are used. All the studies use different marker sizes, camera resolutions or the error is not clearly defined. It is thus difficult to get a good estimate of the expected performance of the pose estimation. As ARToolKit is used very much in research and was already available in ROS this was tested first. The accuracy of this algorithm turned out to be limiting the system. As another graduation research showed good results with ARToolKitPlus this was also tested. As this was not available yet in ROS an own implementation was made which could communicate with other ROS nodes.

Chapter 3

Simulation

3-1 Simulation set-up

To test the pose estimator under ideal conditions it was tested in simulation. This makes it possible to test the tracking algorithms without any distortion in the camera image and under perfect lighting conditions. The main target of these tests is to test how precise the camera position estimation can be. And to test under which conditions the optimal results can be achieved. In the simulation it is for instance possible to use a camera with a very high resolution.

The simulator used for this was Gazebo, the simulation package used in Robot Operating System (ROS). With this camera images can be generated for the marker recognition and the calibration can be tested. There was no working implementation of the UR5 robot arm available (the ROS versions Fuerte, Groovy and Hydro were tested, but all had different issues). Therefore a simple robot model was created. In figure 3-1 a screen shot from the simulation is given. It consists of a simple arm with two degrees of freedom and a pole with an image of a marker on it. As Gazebo is a physics simulator it is not possible to just change the orientation of a part. Small changes are possible but large deviation makes the part oscillate. Therefore it was necessary to create a controller to apply a force on a joint. An overview of the system is given in figure 6-1. The simulation consists of several programs, nodes in ROS terminology, that each have a specific task.

3-1-1 Main node

The main node controls all the other nodes. The program contains the measurement procedure for one test. It sends joint angles for the robot arm to the controller, waits on the robot to reach the specified orientation and logs the transformations to a log file. In this part only the transformation between the estimated marker position and the real position is important. But it logs more transformations so that the log file can also be used to determine the camera position on the robot.

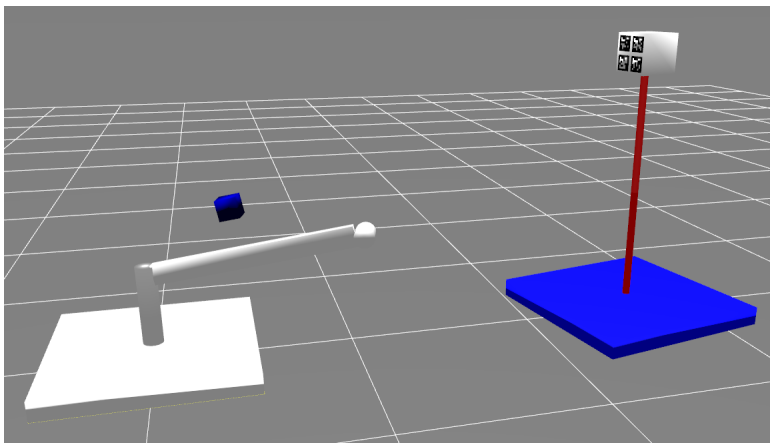


Figure 3-1: Screenshot from the simulation of the robot in Gazebo. The blue box contains the camera. It moves together with the link of the arm beneath it. The pole on the right has a white box on top with four AR markers on it. Different marker configurations were tested. The marker is slightly rotated relative to the base as zero rotation gave numeric problems in the marker pose estimator.

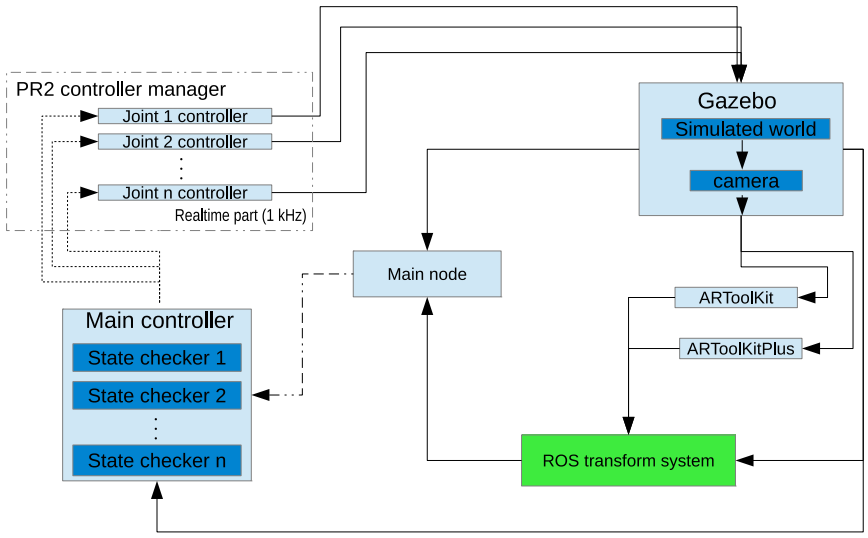


Figure 3-2: Block diagram of the system. Solid lines are information streams (topics in ROS vocabulary), dotted lines symbolize service calls (through which commands are sent). The main node controls the simulated robot arm in Gazebo through the joint controllers. The controllers use the PR2 controller manager structure. Gazebo then generates information like the camera image and transformations between the hand (outer link) and eye (the camera). The block with 'ROS transform system' in it is in green as this a standard ROS node, this node keeps track of all the transformations between parts of the robot and the transform to the marker. The ARToolKit node is from the ROS repository while the ARToolKitPlus node is an own implementation. In appendix C a block diagram with the corresponding file names is given.

3-1-2 Controller

First a custom Proportional-Integral-Derivative (PID) controller was created that applied a torque to the joint directly. Although there was put a lot of effort in tuning the gains it was not possible to control the arm without oscillations. This could have to do with the synchronisation between the simulation in Gazebo and the separate ROS controller node. To solve this problem the custom controller was abandoned and the PR2 controller package was used. These controllers are made for real-time control of the PR2 robot but can also be used for other robots. A controller based on this architecture did not have these problems and was also better configurable. Therefore controllers based on the PR2 real-time controllers are used. Advantage of this architecture is also that the main controller can be configured to control an arbitrary amount of joints. Because of the modular structure it is possible to create extra controllers for more joints with controller gains that are adjustable per controller. The main controller node reads the number of joints and their names from the parameters which are set in the launch file. For each joint a state checker is created which sends the joint state goals to the controllers and checks if these goals are reached. The controllers that actually control the arm run in a separate node per controller. This is because these nodes are run in real time at 1 kHz. In appendix C a block diagram with the corresponding file names of the nodes is given.

3-1-3 Camera

A simulated camera with a resolution of 800x600 pixels is used. There is no distortion added to the image, representing a perfectly intrinsically calibrated camera. The camera parameters are listed in table 3-1. The hfov value is the horizontal field of view in degrees. Nearclip and Farclip determine the drawing range for the simulator, objects outside of this range are not drawn. An overview of all the camera parameters used in ROS is given the documentation[1].

Parameter	Value	Unit
Image size	800x600	pixels
hfov	45	degrees
Nearclip	0.1	
Farclip	100	
Update rate	30	Hz

Table 3-1: Parameters of the simulated camera.

Parameter	Value	Unit
Pattern	Hiro	
Width	500	mm
Threshold	100	(Default)
Use history	True	

Table 3-2: Configuration of ARToolKit.

3-1-4 Pose estimators

ARToolKit and ARToolKitPlus were tested. To prevent both from interfering with each other only one pose estimator was active per test.

[1] http://www.ros.org/reps/rep-0104.html

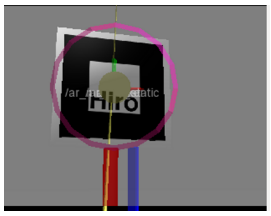


Figure 3-3: Standard ArToolKit marker. Two circles and the frame are projected on it to mark the detection by ARToolKit.

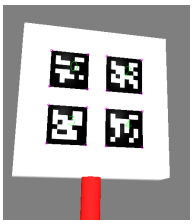


Figure 3-4: Markerboard for use with ArToolKitPlus. Edges are highlighted with circles and the marker number is overlayed.

ARToolKit

To use ARToolKit in combination with ROS a wrapper of ARToolKit in ROS was used[2]. The used parameters are listed in table 3-2. For thresholding the default level is used. The previous position estimate of the marker is used to improve that of the current estimate. The marker which has a 500x500 mm surface with a text ('Hiro', arbitrary text) is used. A threshold of 100 is used for thresholding. The setting 'use history' is used so that the previous marker estimate is used to improve the current estimate. The image of the simulated marker is shown in figure 3-3.

ARToolKitPlus

As there was no well documented wrapper for ARToolKitPlus available in ROS an own implementation of ARToolKitPlus was made. ROS uses a custom image format which cannot be used with ARToolKitPlus. Therefore the CvBridge package was used to convert the image to the OpenCV image format that ARToolKitPlus can use. The camera calibration parameters for ARToolKitPlus are listed in appendix D-3.

ARToolKitPlus can use a single marker but it can also track multiple markers at the same time. Or translate the measurements of multiple markers back to one point. This has the advantage that if one or more markers are occluded the point can still be tracked. The measurements can also be fused to get a better result. The image of the simulated markerboard is given in table 3-4. Again the surface of the marker (or combined markers) is 500x500 mm.

ARToolKitPlus also does thresholding and undistortion of the image. The settings are listed in table 3-3.

Checkerboard pose estimator

A checkerboard with a checkerboard pose estimation algorithm was also used. The system was however not able to detect the checkerboard. This could have been because no distortion parameters were used.

[2] http://www.ros.org/wiki/ar_pose

Parameter	Value	Description
Borderwidth	0.125f	The black border width of the markers is 1/8 of the marker side
Auto threshold	yes	
Threshold retries	3	Number of retries for thresholding
Pixelformat	BGR	BGR color space
Marker type	BCH	Binary coded marker
Pose estimator	RPP	Robust Planar Pose algorithm
Hull mode	Hull full	
Undistortion mode	LUT	Undistortion using lookup tables
Iter	10	number of iterations for distortion compensation

Table 3-3: The settings used in ARToolKitPlus.

3-2 Simulation tests

The target of the simulation tests is to test:

1. Accuracy and precision of the pose estimation algorithm
2. Influence of lighting conditions
3. Influence of camera resolution
4. Accuracy and precision of the resulting hand-eye transformation
5. Influence of the number of obtained poses
6. Influence of the marker size

The first three items will be tested by comparing the known location of the marker with the estimate given by the pose estimation algorithm. The logger node logs this difference to a text file which can be analysed using Matlab. Lighting conditions can be changed by adding extra light in the simulation. The camera resolution can be changed easily in the simulation description files.

The last three items will be tested by calculating the result for a different amount of poses. This will be discussed in a later chapter.

3-2-1 Simulated dimensions

The robot arm consists of a single link with two rotation axes. The camera is placed in the middle of the link at 500 mm and 300 mm height (from the beginning of the link) So that the that link doesn't fully obstruct the view of the camera. The distance that the hand-eye algorithm calculates is the distance between the end of the link (the hand) to the camera. This is 600 mm and 275 mm height (relative to the end of the link). The marker is placed at 2.5 meters away from the robot at 1.45 meters heigh (center of the marker). First the total (corner to corner) dimension of the marker was 0.5 by 0.5 meters. 0.18 by 0.18 metres was also tested, with these dimensions the marker fits on an A4 paper.

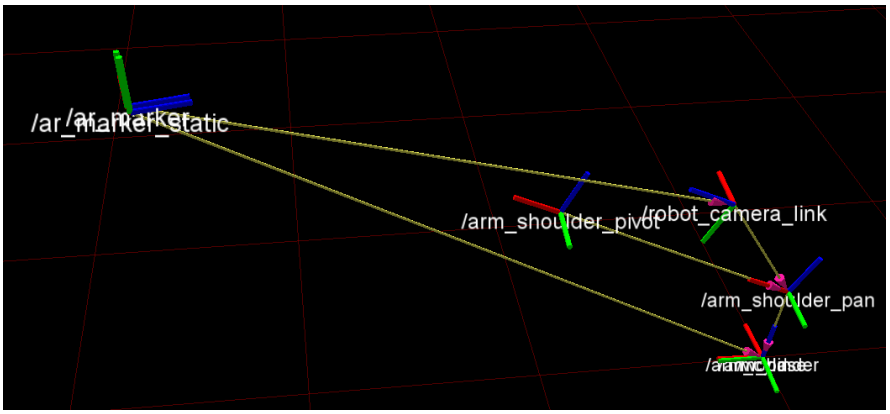


Figure 3-5: Screen shot from the transformation visualization tool (Rviz). On the right the transformations of points on the robot are displayed, with the robot_camera_link the transformation to the camera and armshoulder_pivot the hand of the robot. From the robot base and the robot camera link two transformations are drawn, the first is the correct (static) transformation while the other is the estimated transformation from ArToolKitPlus. It can be seen that there is a small error in the estimation as the lines of axis system do not completely overlap.

3-2-2 ARToolKit, ARToolKitPlus and different marker configurations

The difference in accuracy and precision between ARToolKit and the improved version ARToolKitPlus were tested. This was done by measuring the error between the static and the measured marker position. In figure 3-5 the transforms are displayed. The measured marker transformation is displayed by the line from the robot_camera_link while the line from the base represents the static transformation. The following configurations were compared:

- ARToolKit
  - 0.5x0.5 meter marker
  - 0.18x0.18 meter marker
- ARToolKitPlus
  - 0.5x0.5 meter single marker
  - 0.18x0.18 single marker
  - 4 0.20x0.20 multimarkers
  - 4 0.08x0.08 multimarkers

The simulated camera was placed on the a robot arm with with one link which can rotate over two axes. The arm was moved so that the whole board with the markers stayed in the image but had different positions in the image. Due to the detection of the marker round the edges of the image the number of measurements varies between 185 and 201.

Interesting to note is that ARToolKitPlus needs extra light to properly detect the marker in simulation. It detects the marker at more positions if an source of light is created above the arm. Without the light the image is clearly viewable but to perhaps the extra contrast the light gives a better edge detection. This is odd as ARToolKitPlus has extra dynamic thresholding

features in comparison to ARToolKit, so it it supposed to perform better. ARToolKit on the other hand gives errors with extra light. If the marker is partly in the image and there is extra light a marker is detected very far away.

Results

The results can be seen in figure 3-6 and figure 3-7. A zoomed-in version of the rotation error is given in figure 3-8.

Translation It can be seen that the translation error for all configurations is for the Z axis the largest. This can be explained by the fact that a change in distance on the z-axis does not change the image very much in comparison with movements in the other directions. Interesting is also that the average error is not round zero but it has a bias which is below zero for the X and Y axes while its above zero for the Z axis. If the methods per axis are compared it can seen that the single configurations (methods using one marker) perform slightly better than the multi marker method. For the 0.5x0.5 meter marker ARToolKit performs slightly better than ARToolKitPlus. Which is odd as ARToolKitPlus should be an improvement. For the 0.18x0.18 meter marker the difference is a bit bigger.

Rotation For the rotation error it stands out that ARToolKitPlus has outliers that are lying very far away from the rest of the measured errors. This effect is only present for the smaller markers. Interesting is that the outliers are all lying on the same side. In figure 3-8 a zoomed-in version is given. It can be seen that the errors are the largest for the Roll and the Pitch angles. Again this can be explained by the change of the marker in the image. In plane rotation gives a large deviation of the image while out of plane rotation gives a lesser change in the image. If the large marker configurations are compared it can be seen that ARToolKitPlus works slightly better. The single and multi marker method perform similarly. For the smaller marker it can be seen that ARToolKit has a bias while the average error of ARToolKitPlus is near zero. The minimum and maximum of ARToolKit is lower than that of ARToolKitPlus. Although the outliers are on similar levels with the minimum and maximum of ARToolKitPlus.

3-2-3 Resolution dependence

Using the single marker with ARToolKitPlus the effects of testing a different resolution were tested. A single marker with dimensions of 0.5 by 0.5 meters was used with different camera resolutions (and different calibration files for ARToolKitPlus). Three resolutions were tested: 640 by 480 pixels, 800 by 600 pixels and 1200 by 900 pixels. Higher resolutions were also tested, but the marker detection at higher resolutions than 1200 by 900 pixels did not seem to work.

From the result it can be seen that accuracy and precision of the estimation of the position and orientation of the marker is also determined by the resolution of the camera. In figure 3-9 the translation error between the static marker and the estimated marker is given. It can be seen that especially the z axis very depended of the resolution. It can be seen that precision

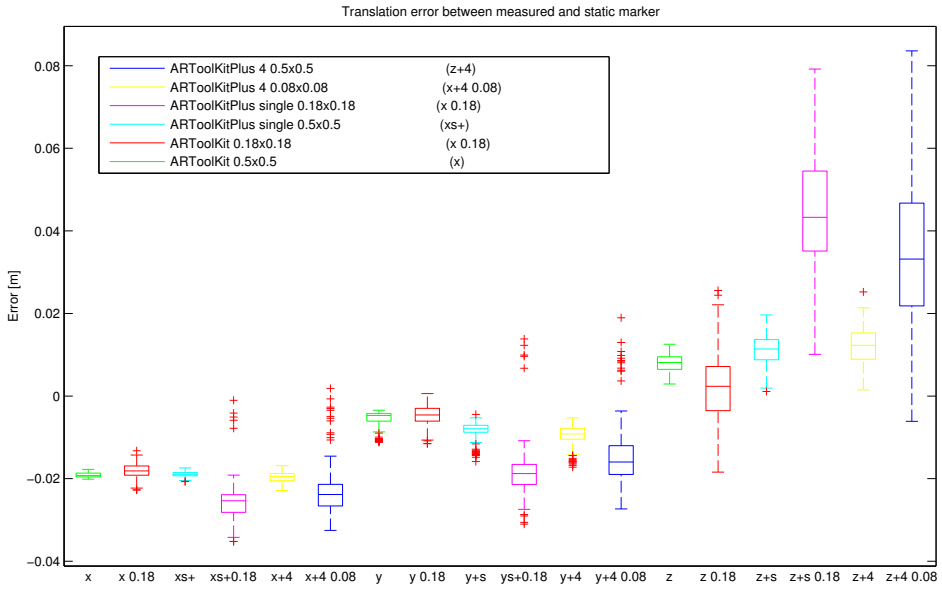


Figure 3-6: Comparison of ARToolKit, ARToolKitPlus with different marker sizes and multi marker versus single marker. It can be seen that with the smaller markers more outliers are present.

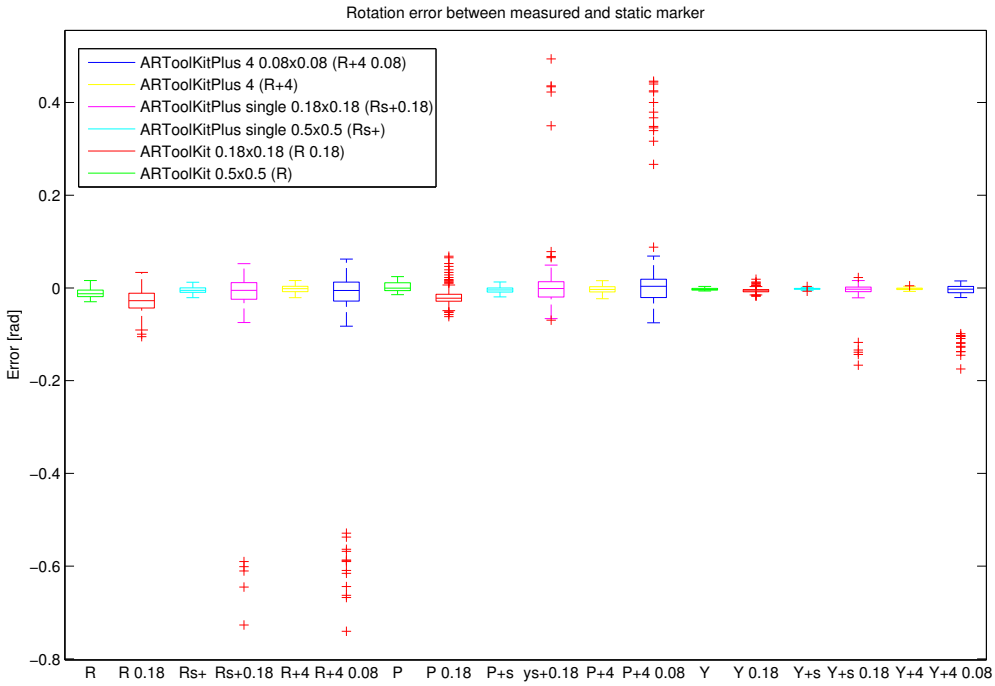


Figure 3-7: Comparison of ARToolKit, ARToolKitPlus with different marker sizes and multi marker versus single marker of the rotation. Interesting to see is that the ARToolKitPlus has in some configurations a lot of outliers. These in the 20x20 centimetres and 18x18 single configurations. But the smaller 8x8 centimetres configuration doesn't have this. Given that the marker size is smaller here more outliers should be present here compared to the larger images.

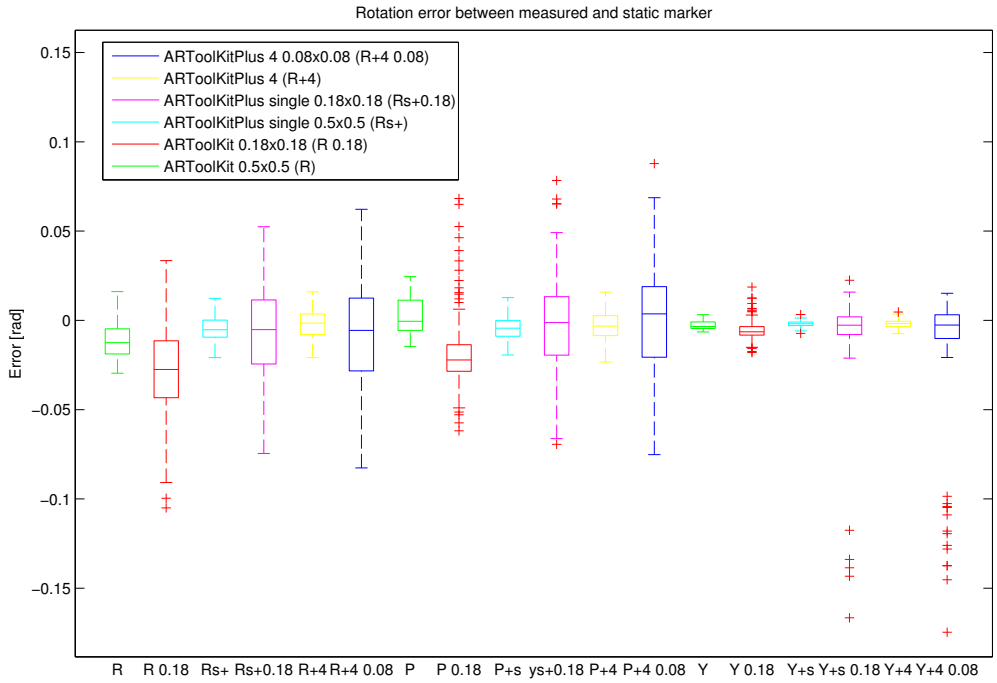


Figure 3-8: Comparison of ARToolKit, ARToolKitPlus with different marker sizes and multi marker versus single marker. Close up of the previous plot, the outliers at 0.4 and -0.6 are here not displayed.

	x	y	z	RMS	R	P	Y	RMS
Lowres	3.535	-4.606	-5.419	4.585	17.5	-0.6267	3.624	10.32
Normal	5.059	-7.555	5.244	6.06	10.27	-5.677	5.46	7.472
Highres	1.479	-7.786	2.955	4.884	6.619	-2.935	-1.187	4.236

Table 3-4: The error for the normal and the high resolution. The translation error is millimetres. The rotation error in Roll, Pitch, Yaw representation in milliradians

improves for all axis but the accuracy only improves for the Y and Z axes. In figure 3-10 the error for the rotation is given. Here the relation between the precision and the resolution is even stronger. The accuracy is however not improved very much here. The effect is the strongest for the yaw axis but the error is not improved very much.

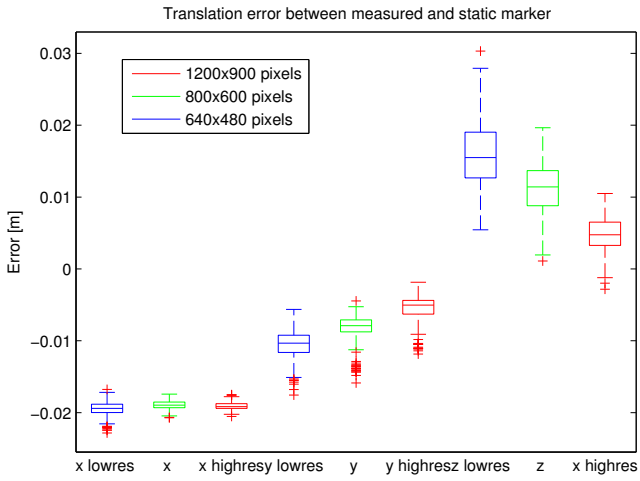


Figure 3-9: The translation error between the static and measured marker for three camera resolutions: 640 by 480 pixels, 800 by 600 pixels and 1200 by 900 pixels.

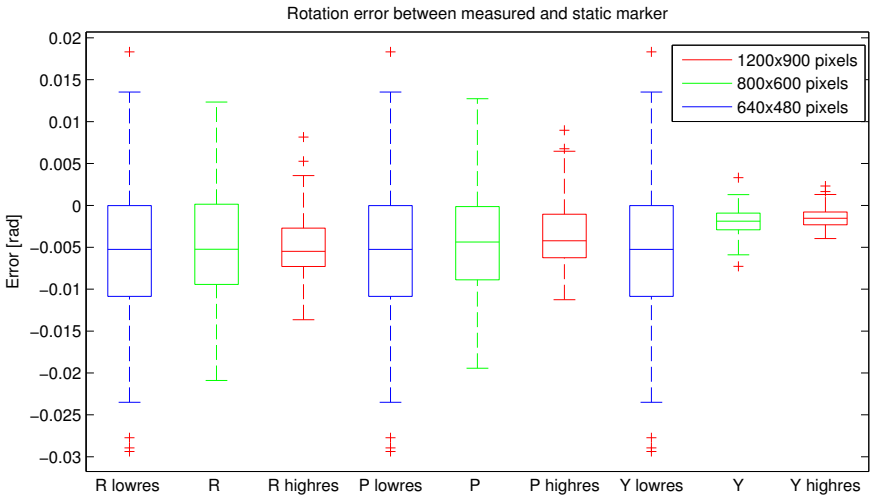


Figure 3-10: The rotation error between the static and measured marker for three camera resolutions: 640 by 480 pixels, 800 by 600 pixels and 1200 by 900 pixels. It can be clearly seen that the error scales with the resolution.

Chapter 4

Validation with a real camera

To be able to estimate the position of the camera on the robot arm the pose of the camera relative to a calibration target needs to be known. This pose estimation system has been tested in simulation in the previous chapter. To validate the simulation results the camera pose estimation will now be tested on a real robot system.

4-1 Test hardware

Robot arm The Universal Robot UR5 robot arm was used. As three of these arms were available in the lab it was logical to use this type of robot arm. This six Degree Of Freedom (DOF) arm can be used without safety fences around it. This makes it easier to use in a normal environment. The robot arm is supported in Robot Operating System (ROS) and can be controlled using a laptop which is connected through ethernet to the robot controller. Full specifications are given in appendix D-1.

Camera system A Logitech Webcam Pro 9000 was used. This camera was taped to a mount which was screwed to the Tool Centre Point (TCP) of the robot.

4-2 Software

4-2-1 Intrinsic calibration

For the extrinsic calibration to work the camera intrinsics need to be known. ROS can store calibration data and send this data along the image stream. To get this data the camera needs to be calibrated. Intrinsic parameters vary per camera model and even within the same model parameters can vary. But if the focus stays fixed the parameters are relatively fixed (they will probably drift over time). These parameters used here are: the principal point, the focal length and the distortion coefficients (5 coefficients using Plumb Bob model[27]).

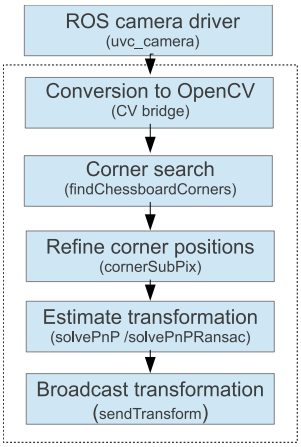


Figure 4-1: Block diagram of the checkerboard pose estimator. The first step (image acquisition) is performed outside the program. Between brackets the ROS package or OpenCV function which is used for the task is given.

The checkerboard used for the extrinsic calibration can also be used to calibrate the camera intrinsically. ROS has a module which was build on the Open Source Computer Vision library (OpenCV) calibration method which can be used to calibrate the camera(which uses an algorithm based on Zhang et al.[28]). By moving the checkerboard around in the image field of the camera several checkerboard poses can be captured. Indicators on the visualization give and indication of enough poses are captured. An optimisation routine then determines the parameters.

The camera intrinsics for the camera's used in this study are listed in appendix D.

4-2-2 Camera pose estimation

ARToolKit and ARToolKitPlus were tested. Results were not very precise. Especially at larger rotations the error increased, there was probably a conversion error somewhere. As there were also doubts about the precision of the Augmented Reality (AR) markers a checkerboard pose estimator was tested.

Checkerboard pose estimator

The checkerboard pose estimator uses functions from OpenCV to detect a checkerboard in the image and estimate its position and orientation. In figure 4-1 a block diagram of the system is given.

Image acquisition is done outside the program through a ROS camera driver. This can be for instance the uvc_camera driver (USB Video Class) or the gscam (Gstreamer based). The camera driver creates an image stream and camera info stream. The camera is assumed to be calibrated. The camera driver loads the calibration information and distributes this over the camera info stream. The Checkerboard pose estimator subscribes to the image stream and the camera info stream on which the camera intrinsics are published. This makes it

possible to use different (calibrated) camera's as the camera intrinsic parameters are loaded automatically. The image data is then converted to an image format that OpenCV can use.

Next corners are searched to detect if there is a checkerboard present in the image or not. The function looks for the internal corners of the chessboard. The number of corners that need to be found are loaded from a configuration file. In this function the histogram of the image is first equalized. With the average brightness as threshold the image is then converted to black and white after which the corners are searched. The function also has a fast check option which speeds up the detection. In tests turned out that this sometimes impedes the detection while a marker is clear in the image. This is therefore not used.

In the next step the position of the corners is tried to be refined to a subpixel accurate location by calculating gradients. With these these corners the position of the camera relative to checkerboard are calculated. The parameters which are used are given in appendix D-8.

The pose is calculated with the OpenCV solvepnp() or solvePnPRansac() functions. The pnp part stands for Perspective-N-Point. Both try to match a predefined grid of corner locations to the grid of detected corners in the image. This function also use the camera matrix and the distortion coefficients of the camera. This parameters are obtained through the camera info stream. The function can use the previous guess as an starting point for the next estimation and outputs the translation and rotation of the checkerboard. The solvePnPRansac function uses a RANdom SAmple Consensus (RANSAC) method to minimize the error between both grids. Both methods can use the following methods to solve the PnP problem:

- CV_ITERATIVE (default), Levenberg-Marquardt optimization.
- CV_P3P, a method developed by Goau et al. [29]. Can only use four points.
- CV_EPNP, a method developed by Lepetit et al.[30]

The default is used.

The function outputs a translation and rotation of the object in camera coordinate system. The rotation is given as 3 by 1 (compact) Rodrigues rotation vector. This is converted first to a rotation matrix. Which is then converted to a rotation in quaternions which is the standard rotation representation in ROS. The resulting transformation is broadcasted over the transformation system of ROS so that other nodes can use the obtained position.

Checkerboard_detector2

The ROS package checkerboard_detector2 also uses OpenCV functions to estimate the position of the checkerboard. It also uses the default solvepnp() settings and does not have RANSAC method. It does remove corners and can detect multiple markers. In appendix D-8 an overview of the configure settings of both nodes is given.

4-3 Error of the pose estimator

To test the error of the checkerboard pose estimator the camera was moved parallel to the marker. The camera was held at approximatively 0.77 metres from the wall, parallel to the

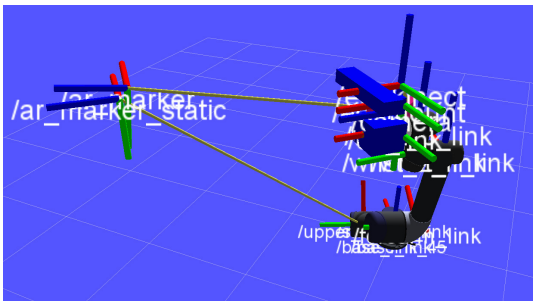


Figure 4-2: Visulatization of the robot model with the transformations. On the left below is a 'static transformation' to the marker position which is used to calculate the error. Above it is the estimated marker position, it can be seen that there is some offset between the two positions.

wall. Three resolutions were tested: 640x480, 800x600 and 1600x1200 pixels. The normal solvePnP solver was used as well as a RANSAC based one. In the ROS transformation system a static transform publisher to marker was used to calculate the error. The distance from the robot base to the marker was taken as the estimated distance when the camera is in front of the marker, which should be the ideal condition.

4-3-1 Results

The results are given in figure 4-3.

For the 640x480 pixels resolution and the 800x600 pixels resolution the test was ran three times, for the 1600x1200 pixels it was ran twice. With the 1600x1200 pixels resolution the chequerboard wasn't detected sometimes. This had probably to do with the larger computational load. Lowering the number of frames per second in the camera to 10 frames per second did not help, the pose estimator did run at an even lower rate. It can be seen that the results are pretty comparable except for the 1600x1200 pixels resolution. If we look at table 4-1 it can seen that the standard deviation even increases a little bit. This is in contrast with the simulation in which there was a clear improvement when using higher resolutions. This could be due to distortion in the lens and a limiting optical resolution of the lens. The algorithm uses the distortion parameters but this could still be a limiting factor. The optical resolution of the lens could also influence the results. It could be that the image is not becoming much sharper when using a higher resolution due to the lens resolution.

It can be seen that the error for the Z axis is very constant. This is probably because there is no movement and rotation in this direction, the camera stays at the same distance and orientation to marker. Rotation is also in plane, a small rotation will give a large number of pixels that change. This in contrast to the out of plane rotation were a small rotation changes much less pixels in the image. The change in error for the X and Y translation is more difficult to explain. This could be due to distortion. The image of the checkerboard is moved over the sensor, it does not occupy the whole sensor. So distortion round the edges of the frame could be of influence, although the algorithm should compensate for this.

If we look at the table 4-1 it can be seen that standard deviation for the translation on the x and z axes is in the order of millimetres. While it is in the order of centimetres for the y

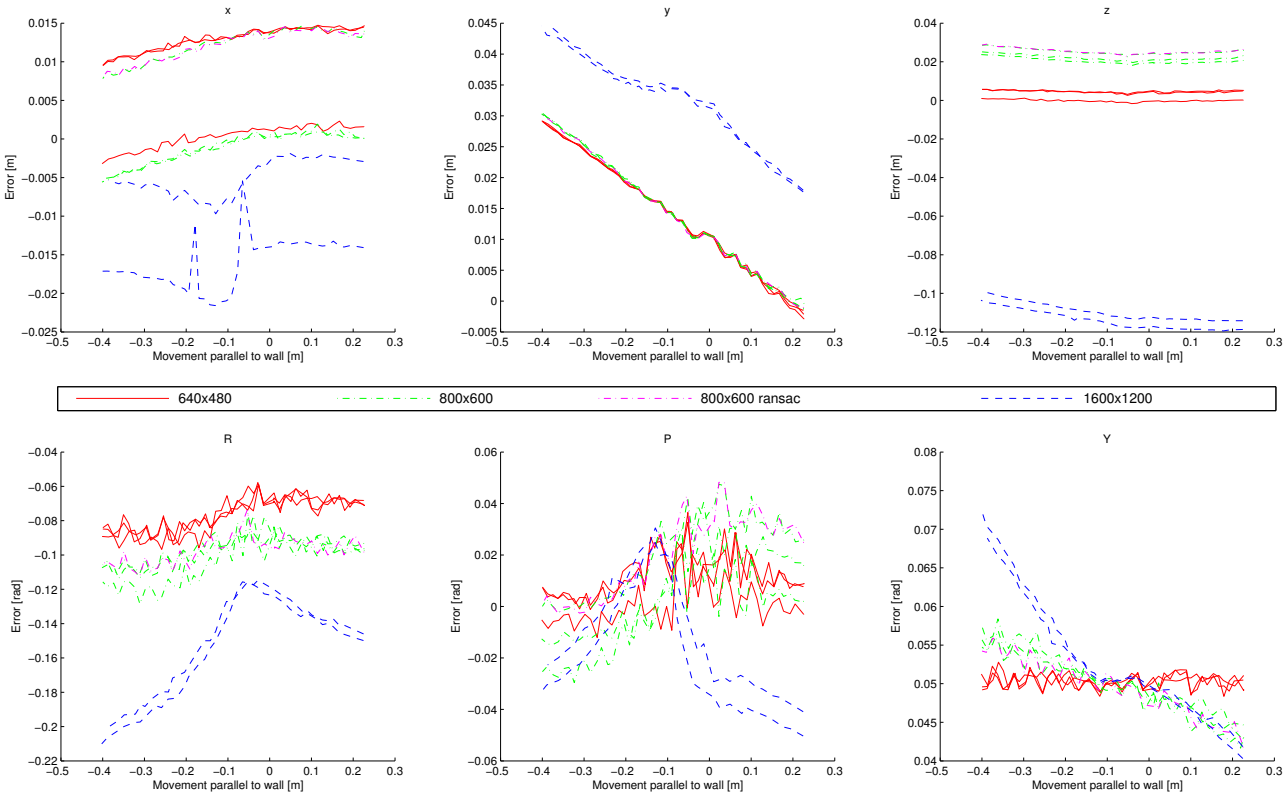


Figure 4-3: Error of the Logitech camera with checkerboard pose estimation. First row the translation error, second row the rotation error. On the X axis the movement parallel to the wall. The zero point on the x axis approximately 0.2 metres of the marker center. Interesting to see is that for the translation on the x and z axis the error is relatively constant. While for the y axis the error decreases for the movement in x direction.

	640x480	800x600	800x600 RANSAC	1600x1200
range x	5.294	6.3865	6.665	11.984
std x	1.4627	2.0078	1.996	3.5158
range y	31.326	31.554	31.349	26.893
std y	9.3172	9.4692	9.4266	7.065
range z	2.8482	4.992	6.2209	15.271
std z	0.616 14	1.445	1.4498	4.7785
range R	35.59	37.698	38.458	90.18
std R	9.896	9.8469	7.8219	29.382
range P	35.925	48.91	51.056	73.332
std P	8.2749	16.17	15.397	24.154
range Y	3.6707	11.54	14.17	29.713
std Y	0.863	3.8146	3.7258	7.9038

Table 4-1: The range (maximum minus minimum value) of the measurement per axis for the translation (x,y,z) and rotation (in Roll, Pitch, Yaw system) sand the average standard deviation of the measurements. Translations in millimetres and rotation error in milliradians.

axis. For the rotation it is round 35 millirads for the Roll and the Pitch, while is a factor ten lower for the Yaw. For higher resolutions the deviation increases.

4-3-2 Discussion

In this experiment the rotation was tested but the distance to the marker also varied. A test with constant distance to the marker i.e. a curved trajectory would have been better. The checkerboard pose estimator had problems with coupling in the rotations and translations. Rotations on one axis resulted in translations and rotations. This problem originated somewhere in the conversion of the estimated position in the program and with the coupling to position of the robot. This was solved for the checkerboard detector. It could be possible that a similar problem was limiting the performance of the AR marker systems. Due to time constraint this was not tested if this was the case.

4-3-3 Conclusion

From this test it was found that the standard deviation of the checkerboard pose estimator is in the order of millimetres on the x and z axes. For the y axis there is linear relation with the translation of the camera. A higher resolution does not contribute to a better result.

Part II

Extrinsic calibration

Chapter 5

Extrinsic calibration of the camera in the workspace

In the previous chapter methods to estimate the position of the camera relative to a calibration target have been studied and tested. This position data will now be used with the pose of the robot to calculate the position of the camera on the robot. After this methods for mapping the workspace will be given.

The problem of determining where a camera is relative to a marker or to its environment has been studied extensively. The more specific problem of determining where a camera is mounted relative to a gripper is called Hand-Eye calibration. The first solutions of this problem date back to 1987. First by Shiu and Ahmad [31] and later by Tsai-Lenz [32]. Although a lot of different solutions have been presented since then the method by Tsai and Lenz still remains very popular.

The hand eye calibration problem has been approached with several sensor systems. There are implementations that use stereo cameras[33], lasers, Kinect senor[34], IMU's combined with indoor GPS[35] and several others. As in this study a monocular camera is used the discussion here will be limited to this sensor system. If we limit the set of methods to the methods that use monocular cameras the methods can be grouped in two categories. These categories are robot hand calibration and simultaneous robot hand and robot world calibration. In the first category the pose of a camera on a robot arm is calibrated relative to the gripper (the hand) of the robot. In the second category the robot pose relative to the world coordinate system is also calculated. According to [36] both approaches can be subdivided into:

- Separable closed form solutions (two stage)
- Simultaneous closed form solutions
- Iterative closed form solutions

Most approaches work by moving the camera round a reference object like a checkerboard. By using multiple viewpoints the camera position can be determined. In figure 5-1 an example

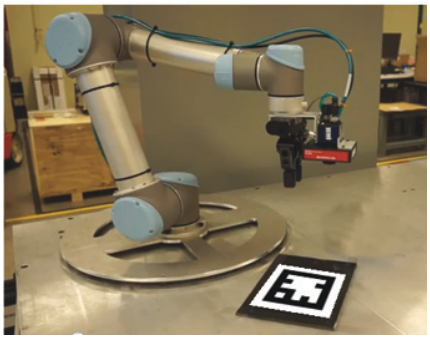


Figure 5-1: Example of an UR5 robot arm with a gripper and camera system fitted on the tool center point. The camera is pointed at an AR marker.

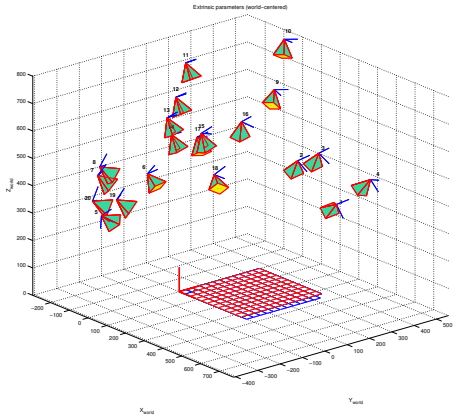


Figure 5-2: Visualization of the different poses in Matlab using the Camera Calibration Toolbox. Each pyramid represents a camera pose and the square the marker.

setup is given of a robot arm with a gripper and camera on the tool center point and an Augmented Reality (AR) marker beneath it. The robot arm can move the camera round the marker and capture images of the marker from different viewpoints. The viewpoints are visualized in figure 5-2.

An overview of the different methods will be given. The studies listed in bold are studies that are compared in the 'Selection of Method' section. After this section a method is selected.

5-1 Robot hand calibration

Using this approach the camera position and orientation of the camera relative to the gripper of the robot are determined.

5-1-1 Separable closed form solutions (two stage)

The first approach is relating the hand movement to the perceived motion of the image of the camera:

$$AX = XB \tag{5-1}$$

In this formulation A is the Tool Centre Point (TCP) (the 'hand') motion T_{t1}^{t2}, B the perceived camera motion T_{c1}^{c2} and X the transformation between the hand and the camera. The motion of the TCP is determined using the encoders of the robot. And the motion of the camera is extracted from the sensor. This can be for instance a camera with a calibration target or a 3D camera. The motions between these points can be sub-dived into a rotation and a translation. This is usually noted by the homogeneous matrix:

$$A = \begin{bmatrix} R_A & t_A \\ 0 & 1 \end{bmatrix} \tag{5-2}$$

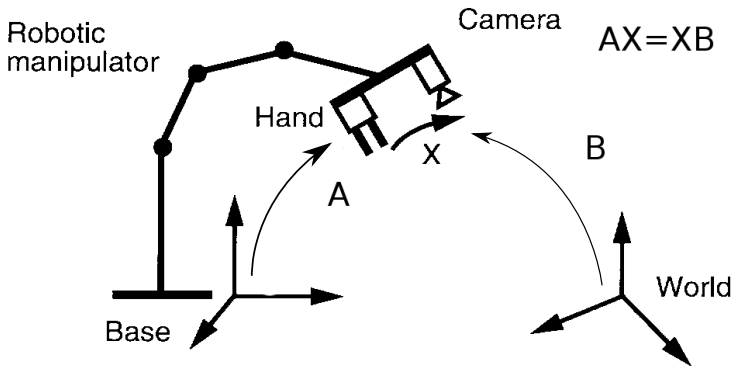


Figure 5-3: Visualization of the different poses in Matlab using the Camera Calibration Toolbox. Adapted from [3].

The whole equation can be formulated as:

$$AX = XB \tag{5-3}$$

$$\begin{pmatrix} R_A & t_A \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_X & t_X \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_X & t_X \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_B & t_B \\ 0 & 1 \end{pmatrix} \tag{5-4}$$

The separated rotation:

$$R_A R_x = R_x R_B \tag{5-5}$$

And translation:

$$R_A t_x + t_A = Rxt_B + t_x \tag{5-6}$$

Shiu and Ahmad [31] were the first to formulate the problem like this. An angle-axis formulation is used to solve the rotational part.

$$R_x = Rot(k_{A_i}, \beta_i) R_{x_{P_i}} \tag{5-7}$$

With:

$$\begin{aligned} R_{x_{P_i}} &= Rot(v, \omega) \\ v &= k_{B_i} \times k_{A_i} \\ \omega &= atan2(|k_{B_i} \times k_{A_i}|, k_{B_i} \cdot k_{A_i}) \end{aligned} \tag{5-8}$$

With R_x known the translation can be calculated using linear algebra:

$$\begin{pmatrix} R_{A_1} - I \\ \vdots \\ R_{A_n} - I \end{pmatrix} t_x = \begin{pmatrix} R_X t_{B_1} - t_{A_1} \\ \vdots \\ R_X t_{B_n} - t_{A_n} \end{pmatrix} \tag{5-9}$$

Zhuang et al.[37] reformulated the solution of Shiu and Ahmad with quaternions making it possible to also use rotations round 180°. Problem with the solution of Shiu and Ahmad is

that every time that a frame is added to the system the size of the linear system doubles. In a later formulation by Tsai and Lenz [32] a fixed size system is used. Again an angle axis method is used to solve the rotation in 5-5 and then using linear least squares the translation can be calculated.

$$Sk(k_{R_{A_1}} + k_{R_{B_1}})k'_{R_x} = k_{R_{A_1}} - k_{R_{B_1}} \tag{5-10}$$

$$k_{R_x} = \frac{2k'_{R_x}}{\sqrt{1 + |k'_{R_x}|^2}} \tag{5-11}$$

$$Sk(x) = \begin{pmatrix} 0 & -x(3) & x(2) \\ x(3) & 0 & -x(1) \\ -x(2) & x(1) & 0 \end{pmatrix} \tag{5-12}$$

Angle of rotation:

$$\theta = 2arctan(|k'_{R_x}|) \tag{5-13}$$

With the rotation R_x known linear least squares is used to get the translation from 5-6.

The camera needs to make at least two movements with non-parallel rotation axis for a solution[32],[38] but more movements increase precision of the result. **Wang[39]** also used an angle-axis formulation. Three methods are presented which are all closed form. The first method requires a reference object at a pre-calibrated location and uses just transformations. The second method also requires a reference frame and is comparable to Tsai-Lenz and Shiu-Ahmad. In the third method no calibration target is necessary but there are special motions of the robot arm necessary.

Other approaches are based on quaternion algebra by Chou et al.[40], [41] and Horaud and Dornika(referenced here as **Horaud-Dornika'95)[42]**, screw motion analysis[38], properties of the Euclidean group[43], **Li et al.[44]** use canonical matrix representation and non-linear equations.

5-1-2 Simultaneous solutions

The two stage methods presented before have the advantage that they are fast. But the error in the rotations propagates through in the translations. Therefore there are also other approaches which simultaneously determine rotation and translation. For this formulation there are several implementations available. **Daniilidis et al.** use dual quaternions to describe the system[45]. **Andreff et al.[46]** use the Kronecker product to create a linear system. The method has as problem that the resulting R_x rotation is not necessarily orthogonal which could lead to errors in the resulting transformation.

5-1-3 Iterative solutions for rotation and translation

It is also possible to use an iterative solution. By minimizing $||AX - XB||$. This approach also solves the problem with the propagating orientation errors. But this method can be computationally expensive because of the complex optimizations. If the number of equations

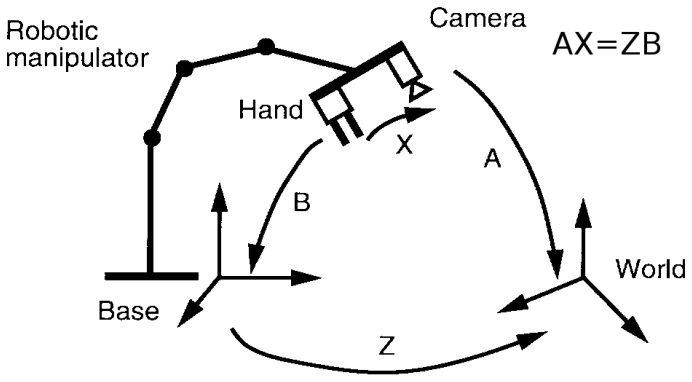


Figure 5-4: Simultaneous robot hand and robot world calibration. Usually formulated as AX=YB or AX=ZB. Adapted from [3].

get larger the differences between closed form and iterative solutions becomes smaller[36]. **Wei et al.[47]** developed a method which does not use a calibration point but can just use points in the image. To do this however stereovision is used. The method can also estimate the internal parameters of the cameras. In the study also the movements which create an optimal result are studied.

In [48] the sensors of the PR2 robot are calibrated using a bundle adjustment approach and Levenberg-Marquardt optimization.

5-2 Simultaneous robot hand and robot world calibration

The previous methods determined the pose of the camera on the robot. But it is also possible to simultaneously determine the pose of the base of the robot relative to world (usually the calibration target). This is usually called simultaneous robot hand and robot world calibration. This class of methods can also be divided in to separable closed form solutions, simultaneous solutions and iterative solutions.

The simultaneous determination of the hand eye transformation and the robot world transformation is usually formulated as:

$$A_i X = Y B_i \tag{5-14}$$

In this formulation A is the pose of the camera frame relative to the calibration frame. B is the pose of the TCP relative to the robot base. And X and Z respectively are the hand eye and world-base transformations. This formulation can also be divided in to separable, simultaneous and iterative solutions. This was first formulated by Zhuang et al[49] in a linear approach. A closed form and a non-linear approach were later created by Dorniaka et al[3]. **Shah et al. [50]** combined the quaternion methods of [3] and [51], with the Kronecker methods of [52] and [53].

Ernst et al. [54], [6] argue that industrial robots are not calibrated optimally and that deviations of 2-3 millimetre can arise. Also optimally calibrated optical tracking systems can also give Root Mean Square (RMS) errors in the order of 0.2-0.3 millimetres. This can lead

Method	Advantages	Disadvantages
Separable	Simple, fast	Errors in rotational part propagate to translational part.
Simultaneous	No propagation of rotation errors	Results depend on scaling.
Iterative	Possibly more accurate	No guarantee optimal solution is found, depended on starting criteria.

Table 5-1: Overview of the advantages and disadvantages of each method. Applicable for AX=XB and AX=YB methods.

to non-orthogonal rotation matrices which can give computational problems. They developed therefore a method which allows the X and Y matrices to be non-orthogonal. This method is called the QR24 calibration algorithm, which simultaneously solves for the rotation and translation using least squares with QR factorisation (hence the name). A variant of this, OR15 also works without or with incomplete rotational data input. If there is no complete six Degree Of Freedom (DOF) tracking data available this method can give a partial solution as opposed to other methods which would not work then. The number after the QR stands for the number of parameters that are estimated.

5-2-1 Overview of methods

In table 5-1 a very general overview of the advantages and disadvantages of the various calculation methods is given[36]. Both the robot hand (AX=XB) and the simultaneous robot hand and robot world calibration (AX=YB) can be solved using the same calculation method.

5-3 Structure from motion

In most studies a calibration target is used. This can be a point in the image or for instance a checker board or AR marker. From the 2D images the position and orientation is then estimated using the camera intrinsics. It is also possible to use structure from motion to determine the camera movement. Structure from motion tracks features from image to image and estimates the camera motion from this. An advantage of using this method is that can be used in dirty environments where the calibration target would get dirty. A good example of this is given in **Schmidt et al.[55]** where the calibration is used to calibrate a camera which is attached to an endoscope. In this case a marker is impractical in a sterile environment. In this case the 'hand' is at the end of the endoscope which is outside the body. A marker is attached to this to track the 'hand' position. While the 'eye' at the other end is inside the body of a person undergoing surgery. A disadvantage of this method is that the translation is estimated with an unknown scale factor which also needs to be estimated.

$$A_i = \begin{pmatrix} R_{ai} & \lambda u_{ai} \\ 000 & 1 \end{pmatrix} \tag{5-15}$$

With R_{ai} the rotation between image i-1 and i and u the translation which is related to the camera translation as $t_{ai} = \lambda u_{ai}$ [53]. By using pure translations and pure rotations it

is possible to calculate the scale factor λ. According to [55] Structure from motion is less accurate than methods that use a calibration target. In the method presented by Andreff et al. in [46] the points needed for structure from motion also need to be tracked over the whole robot movement which is difficult for a long trajectory.

5-4 Comparative studies

Although there are a lot of different algorithms there are few comparative studies available comparing several methods at the same time. Although the method presented by Tsai-Lenz[32] is relativity old the method still performs reasonably well compared to newer algorithms. An overview of the studies that compare different algorithms will be given next. In table 5-3 an overview of the studies is given and which methods they compare. In 5-4 the results per algorithm are given.

Wang in [39] compares Tsai-Lenz and Shiu-Ahmad with his own methods. For the input data a monocular camera (510x490 pixels) with a calibration grid is used. The method of Tsai-Lenz has the lowest standard deviation and is selected as best.

In [44] Li and Betsis compare Tsai-Lenz, Horaud-Dornika'95[42] and there own methods which include a least squares solution, geometric and a non-linear optimization with quaternions. Test are done in simulation and on the KTH head-eye system. They concluded that their own non-linear and the Tsai-Lentz algorithm give the best results.

In Wei et al.[47] an active motion approach is used to calibrate both the intrinsic and extrinsic parameters. In this study a stereovision set-up is used with cameras with a resolution of 460x420 (TV lines). The method is compared with an extended version of the Tsai-Lenz method, which can also determine the intrinsic parameters. In the experiments their own method performs better than the modified Tsai-Lenz method.

In [56] a method was developed that does not need markers and gave average values of 2×10^{-3} rad for rotation and 2 mm for translation. For this a camera was used with a resolution of 1260x960 pixels. Although the authors claim that the new method is well suited in reference to the method of Tsai-Lenz no comparative data is given.

In [53] a comparison between the Algorithm of Tsai-Lenz, Daniilids et al. [57], Horaud[42] and two new algorithms was made. A new linear formulation is introduced which can handle small rotations. When the robustness to noise is tested the new method and the method of Tsai-Lenz are the best. For higher noise levels is the translation error lower for the dual Quaternion method of Daniilidis et al. The effect of the number of motions is also tested. The new formulation proved better although the Tsai-Lenz algorithm was also very close.

In [6] hand-eye calibration was investigated for application in Transcranial Magnetic Stimulation (TMS). TMS is performed on human heads which needs high precision. The Tsai-Lenz, Daniilidis' dual quaternion method and three own methods (QR24, QR15 and QR24m) were tested. The influence of different distance sensor systems was also compared: an optical system, magnetic, laser and synthetic data were used. Poses were tested for n=5 to 250. The translation error is for the QR methods the lowest. But for the rotation error the Dual Quaternion method works the best on real data. On synthetic data all methods perform similar. In table 5-2 the translation and rotation errors for the different methods are given.

In this table the results obtained with laser data are also included. The errors of all the algorithms with this sensor method are higher indicating that this data source probably has more noise in it. This gives and indication of the effect of noise on the algorithms. It can be seen that for the rotation the methods of Tsai-Lenz and Daniilidis et. al perform also good. The number of pairs that gave the optimal result was also tested. The optimal number varied for the different sensor types. For instance for the optical tracking the new methods and the Daniilidis method stabilized very quickly for the translation while the error of the Tsai-Lenz method increased after about 25 pairs. For the rotation all lines have about the same pattern, they all have a minimum at about 90 pairs. For the synthetic data the results are different. The new methods again converge quickly. Both the Tsai-Lenz and Daniilidis method have a minimum below 100 pairs for the error which increases again until 150 pairs on which gets to roughly the same error again. The rotation error of all methods decreases very quickly. At about 140 pairs all algorithms get even a small negative error. Note that is study was done with relatively small workspaces. The effect of increasing the workspace was also tested. Although the maximal workspace with radius 500 mm and a maximal rotation angle of 30 degrees is still small for industrial robots. The calibration error slightly increased. For the synthetic tests all methods had errors below 0.25 mm and 0.1 degrees.

Test	Optical			Laser		
Translation	Min	Median	Max	Min	Median	Max
Algorithm						
Tsai-Lenz	0.0350	0.2950	0.9289	1.2812	7.6678	18.1265
Daniilidis et. al	**0.0156**	0.2239	0.8305	2.0395	6.7426	13.3181
$QR24_m$	0.0388	0.2255	0.6833	4.4768	10.6338	28.5564
QR24	0.0223	0.1317	0.3095	0.8984	1.3517	**3.3947**
QR15	0.0223	**0.1314**	**0.3089**	**0.8848**	**1.3216**	3.4044

Test	Optical			Laser		
Rotation	Min	Median	Max	Min	Median	Max
Algorithm						
Tsai-Lenz	**0.0042**	0.0979	**0.3758**	0.2883	**0.8088**	1.4628
Daniilidis et. al	0.0072	**0.0857**	0.3832	0.4873	0.9475	1.4691
$QR24_m$	0.0117	0.0961	0.3860	**0.2442**	0.8193	**1.3946**
QR24	0.0130	0.0940	0.4069	0.4121	0.8574	1.4730
QR15	-	-	-	-	-	-

Table 5-2: Translation (first box) and rotation errors (second box) for the different methods. Matrices that result in the lowest RMS error have been used. Dimension for the translation is millimetres and degrees for the rotation, lowest values bold. The QR24m method is a preconditioned version of QR24. Adapted from[6].

In [55] hand-eye calibration was used for a less common usage. It was used to calibrate a camera mounted on an endoscope used for surgery. Hand data was acquired using an infra-red optical system (outside the body) while the camera (inside) used structure from motion to track its position. As no ground truth was available the calculated hand-eye transform was used estimate the eye movement from the hand movement and compare this with the actual movement. The method was compared to that of Daniilidis et. al, Horaud-Dornika'95[42] and that of Andreff et al. All methods gave similar results, although the method of Horaud-

Dornika'95 gave the lowest total error (rotation and translation combined).

Study	Year	Tsai-Lenz[32]	Shiu et al.	Daniilidis.	Ernst	Li	Dornaika, Horaud	Shah[50]	Wei[47]	Wang	Andreff	Best Method
Wang[39]	1992	•	•							•		Tsai-Lenz, non-lineair method comparable
Li and Betsis[44]	1995	•				[44]	•					Tsai-Lenz
Shah [50]	2013					[52]	[3]	•				Shah
Ernst et al. [6]	2012	•		•	•							Ernst
Wei[47]	1998	•							•			Wei
Andreff et al. [53]	2001	•		•			[42]				•	Andreff, Tsai-Lenz second
Schmidt [55]	2005			•			[42]				•	Dornaika, Andreff comparable

Table 5-3: Overview of studies(horizontal) that compare different methods (vertical). Different methods with the same author are listed with a reference instead of a dot. The result column lists what that according to that study is the best method among the compared methods.

[1]Standard deviations after 10 measurements [mm] and [degrees], no error provided.

[2]No unit listed for translation, presumably [mm], rotation in degrees.

[3]Error only represented in graphs.

[4]No rotation error provided.

[5]No units are given for translation or rotation.

[6]No ground truth was available as in this experiment as an endoscope was used inside a body. The hand was a marker which was tracked outside the body while the eye was the camera of the endoscope. Using structure from motion the camera movement was tracked. By using the calculated transform the hand and eye movement could be related. The percentage is a measure of how good these relate.

Study	Tsai-Lenz	Shiu et al.	Daniilidis.	Ernst	Li	Dornaika, Horaud	Shah[50]	Wei[47]	Wang	Andreff	Schmidt	Result
Wang[39][1]	3.39 0.59	7.70 2.41							3.64 0.97			Tsai-Lenz
Li and Betsis[44] [2]	0.011 0.3278				0.011 0.3279	0.011 0.3279						Tsai-Lenz, non-linear method comparable
Shah [50] [3]					>20		>5					Shah
Ernst et al. [6]	0.2950 0.0979		0.2239 0.0857	0.1317 0.0940								Ernst
Wei [47] [4]	3.25							2.11				Wei
Andreff et al. [53] [5]	0.018 0.000011		0.096 0.0000161			0.149 0.0000977				0.023 0.0000006		Andreff, Tsai-Lenz second
Schmidt [55] [6]			21.5% 5.5%			15.85% 5.36%				16.45% 5.35%	18.10% 5.36%	Dornaika, Andreff comparable

Table 5-4: Overview of studies that compare methods with results. The first line of the error is the translation error, the second the rotation error. Different methods with the same author are listed with a reference instead of a dot.

5-4-1 Selection of method

A lot of methods exist for determining the pose of a camera. The pose of the robot base relative to the world is not as important as the pose of the camera on the robot. Therefore the robot hand calibration methods are preferred over the simultaneous robot and robot world calibration methods. There are relatively not very much comparative studies. But from the studies it can be seen that a lot of methods perform similarly. If we look at table 5-3 it can be seen that the method by Tsai-Lenz is selected as the best or second best the most often. In the case of table 5-2 we consider rotation errors more of influence than the translation errors. As the rotation errors can be of more influence than translation errors at a larger distance. If we look at the rotation error in table 5-2 then the minimum and maximum errors of the Tsai-Lenz are the lowest. Therefore this algorithm is chosen.

5-4-2 Discussion

The influence of noise on the algorithms should be investigated. This could have a large influence of on the result.

Chapter 6

Simulation

On a real robot system it is difficult to get a ground truth as it is difficult to measure correctly. In simulation it is easier to check the results and exclude the effects of noise. Therefore the system was first tested in simulation.

6-1 Simulation setup

The same simulation setup as used in chapter 3 was used. Only now a transform broadcaster and a hand-eye calculation node were added.

6-1-1 Transform broadcaster

In the first implementation positions of the marker and the gripper were sent continuously to the hand eye calculation node. This resulted however in very large datasets and very long computation times. From previous researches it was found that after a certain threshold the result not necessarily improves if more points are added [6]. Therefore a node that can regulate the data was put in between. The main node sends a request to send the positions to the hand eye node. Then the logging node requests the positions of the gripper and marker from the central transformation node. These positions are then broadcasted to the hand eye node which stores them.

6-1-2 Calculation node

The calculation node calculates the hand eye transform. The hand-eye algorithm used was that of Tsai et al. implemented in a package called ViSP hand eye calibration[1]. This node stores the transforms from the base to the hand and the transforms between base and eye that are send from the logger node. If the calculation service is called it calculates the distance from the hand to the eye using all the recorded transforms.

[1] ros.org/wiki/visp_hand2eye_calibration

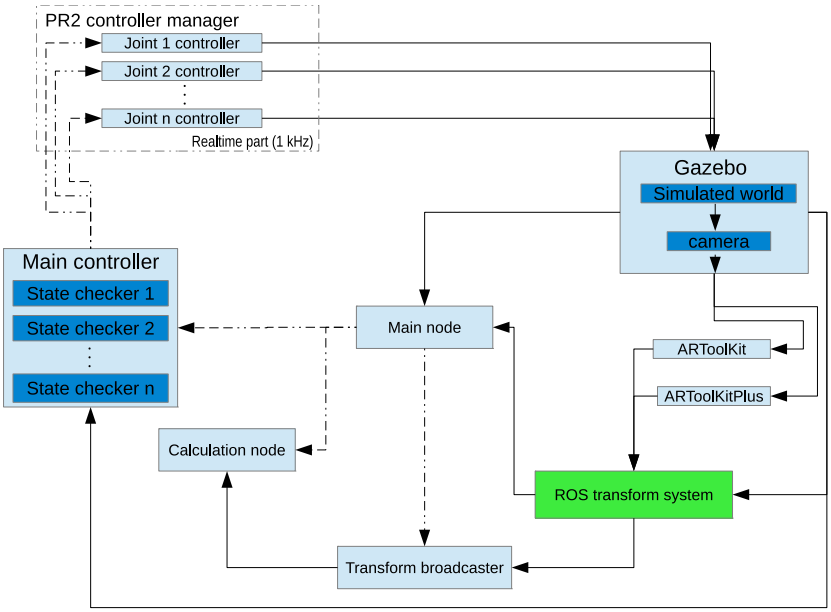


Figure 6-1: Block diagram of the system. Solid lines are information streams (topics in ROS vocabulary), dotted lines symbolize service calls. The main node controls the simulated robot arm in Gazebo through the joint controllers. Gazebo then generates information like the camera image and transformations between the hand (outer link) and eye (the camera). The green 'ROS transform system' block keeps track of all the transformations between parts of the robot and the transforms between camera and marker. In appendix C a block diagram with the corresponding file names is given.

6-1-3 Main node

The main node moves the robot arm through a predefined set of joint angles. With these angles the image of the marker moves over the sensor and reaches the edges of the sensor. It is also possible to let the system look for the marker. In this mode the base link is rotated around until a marker is found. If the marker is found all the links are rotated separately to detect on which link the camera is. First the outer link is rotated after which the inner links are rotated. If the marker position changes than it means that the camera is on the link that is moved.

When the link the camera is on is known the system starts measuring where the camera is on the link exactly and what its orientation is. This is done by moving the link with the camera some more and recording the transformations to the marker and transformations to the outer part of the link for several poses.

When a number of poses is stored the calculation procedure is called which determines the position and orientation.

6-2 Hand-Eye algorithm

The pose estimation algorithm introduces noise in the measurements. To test the Hand-Eye algorithm without this noise the pose estimation algorithm was replaced with the correct

Number	Error x	Error y	Error z	Error R	Error P	Error Y
5	-3.8800	367.1460	-22.5520	-0.2855	0.4755	-10.5591
10	-1.0420	-5.0751	0.4220	0.6721	0.1434	1.9140
15	-0.7070	-1.7275	0.0910	0.6940	0.1255	1.1680
30	-0.3010	-0.5192	-0.5192	0.7400	0.0656	0.9020
50	-0.1960	-0.1843	-0.0340	0.7540	0.0497	0.8320
100	-0.0790	-0.0317	-0.0260	0.7750	0.0227	0.8030
200	-0.0260	-0.0094	-0.0170	0.7860	0.0117	0.79800

Table 6-1: Error of VisP package for synthetic data. Dimensions for x,y,z in millimetres and R,P,Y in milliradians.

transformations. This can be done in ROS using a static transformation broadcaster, which sends a transformation similar to the pose estimation algorithm. With this the hand-Eye transformation can be calculated without the noise.

In table 6-1 the error for different numbers of poses is given. The algorithm needs at least three poses to work but it can be seen that the error at five poses is still very large. At ten poses however the error has dropped to the millimetre range and milliradians range for the rotation. After about 50 poses the rate in which error changes decreases.

The calculation time increases as more position pairs are added. On a relatively slow Core 2 Duo processor the calculation in the beginning of the simulation takes a few seconds, this increases to over a minute with 180 pairs. Real calculation times will be much shorter as the computer is already overloaded with the simulation. The 'top' command lists a load average of over 12, which means that the processor is about 600% overloaded.

6-2-1 Comparison with other algorithms in Matlab

By using a node in the simulation that publishes the correct transformation from camera to the marker it is possible to get noise free input data for the algorithms. Combined with the transformations from the base to the camera link it is possible to test several algorithms in Matlab. A Matlab implementation of Tsai-Lenz was used[2] and several algorithms from the calibration-toolbox[3]. This toolbox contains algorithms from Tsai-Lenz[32], Daniilidis et al[45], Park et al.[43] and Horaud-Dornika'95 [42]. The Tsai-Lenz implementation from the toolbox will be listed as Tsai-Lenz Matlab 2.

All algorithms use the transformation from base to the gripper. The single Tsai-Lenz implementation uses the local coordinate system, the transformations from marker to camera. While the in the toolbox all implementations use the inverse, the transformation from camera to marker. 144 poses are used for the calculation.

Results

The results can be seen in table 6-2 for the translation and in table 6-3 for the rotation. It can be seen that the Robot Operating System (ROS) implementation of Tsai-Lenz gives the

[2] http://lazax.com/www.cs.columbia.edu/~laza/html/Stewart/matlab/handEye.m
[3] https://www.vision.ee.ethz.ch/software/calibration_toolbox//calibration_toolbox.php

Axis	Tsai-Lenz ROS	Tsai-Lenz Matlab	Tsai-Lenz Matlab 2	Horaud-Dornika'95 [42]	Park [43]	Daniilidis [45]
δ x	-0.02	0.08791	0.2946	0.6018	0.1721	0.6793
δ y	-0.007011	0.1069	1.348	1.298	1.302	2.227
δ z	-0.02	0.1278	-0.1071	-0.621	0.09642	-0.6046
RMS δ	**0.01682**	*0.1088*	0.7992	0.9003	0.7601	1.389

Table 6-2: The error of several algorithms in Matlab with the synthetic (correct) data, all in millimetres. 144 poses are used for the calculation. Lowest error is in bold, lowest Matlab algorithm error italic.

Axis	Tsai-Lenz ROS	Tsai-Lenz Matlab	Tsai-Lenz Matlab 2	Horaud-Dornika'95 [42]	Park [43]	Daniilidis [45]
δ R	-0.01032	-0.02391	-0.418	-1.445	-0.009008	-1.244
δ P	0.009686	-0.1361	-0.1797	0.01573	-0.00648	-0.1261
δ Y	0.003677	0.1658	0.1777	-0.007118	0.00089	0.1493
RMS δ	0.008444	0.1246	0.282	0.8343	**0.006427**	1.389

Table 6-3: The errors of several algorithms in Matlab for the rotation with the synthetic (correct) data (all in milliradians).144 poses are used for the calculation. Lowest value in bold.

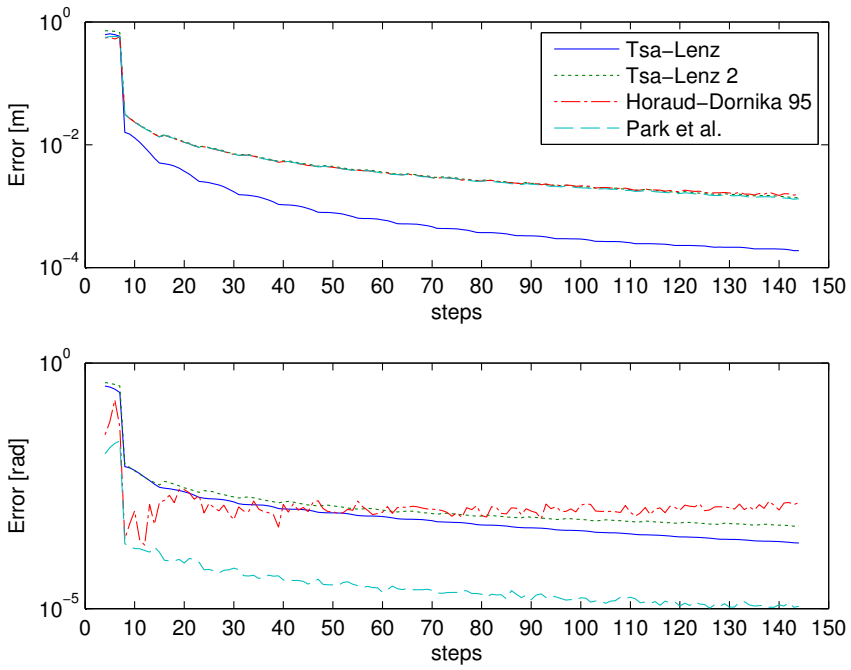


Figure 6-2: Overview of the error of different algorithms for a different number of steps.

measurement	x	y	z
ArToolKit	1.212	-35.41	-2.789
ArToolKit+	4.502	-5.808	5.859
ArToolKit+4	2.857	-1.022	-0.286

Table 6-4: The error for the translation of the hand-eye calibration for different camera pose estimation algorithm. Dimension in millimetres.

best results followed by the Matlab implementation. For the rotation error the algorithm of Park et al. performs best although the Tsai-Lenz implementation in ROS is also very near this value. The ROS implementation of Tsai-Lenz gives a better result than the Matlab implementations of the same algorithm. This could be due to conversion errors, although both Matlab implementations also give different results. In figure 6-2 the error for the number of steps is given.

With wrong translation matrices the algorithms still work reasonably. Accidentally the base to hand transformations were build using the correct rotation matrices but using the marker to static marker translation vector. Although the order of the resulting translations were wrong the magnitude of the translations was still near the correct value.

6-2-2 Sensitivity to noise

To test the sensitivity to noise for the different algorithms noise was applied to the correct transformations. Noise (uniformly distributed pseudo random) was applied on the transformations from the camera to the marker (and the inverse). To simulate noise in the encoder readings a little noise was applied to the base to gripper transformations. The translation noise on the camera marker transformations was varied between -0.04 and 0.04 meters. The rotation noise varied between -0.04 and 0.04 radians. Analysis of the error showed that the actual translation error varies between -0.02 and 0.02 (approximately) and -0.02 and 0.01 rad for the rotation. This noise was applied by converting the rotation matrices back to Roll, Pitch and Yaw (RPY) angles, adding the noise and convert the angles back to a rotation matrix. Using this approach the rotation matrix properties stayed correct.

The results are given in figure 6-3 for the translation and figure 6-4 for the rotation. The Method of Daniilidis et al. is omitted as for certain data set the results became complex. It can also be seen that the methods of Horaud-Dornika and Park et al. look the same for the X and Y axes. There is however a minimal difference.

6-3 Combined system

The resulting hand eye transformation for the measurements is given in table 6-6 for the translation and the rotation in table 6-7. The pose estimation error for this dataset is given was discussed in chapter 3, in figure 3-6 (translation) and 3-7 (rotation). It can be seen that the error of ArToolKitPlus in the single configuration is the smallest. But for a smaller size the error increases a lot. For the rotation ARToolKitPlus with four markers performs best but the single configuration is also very near.

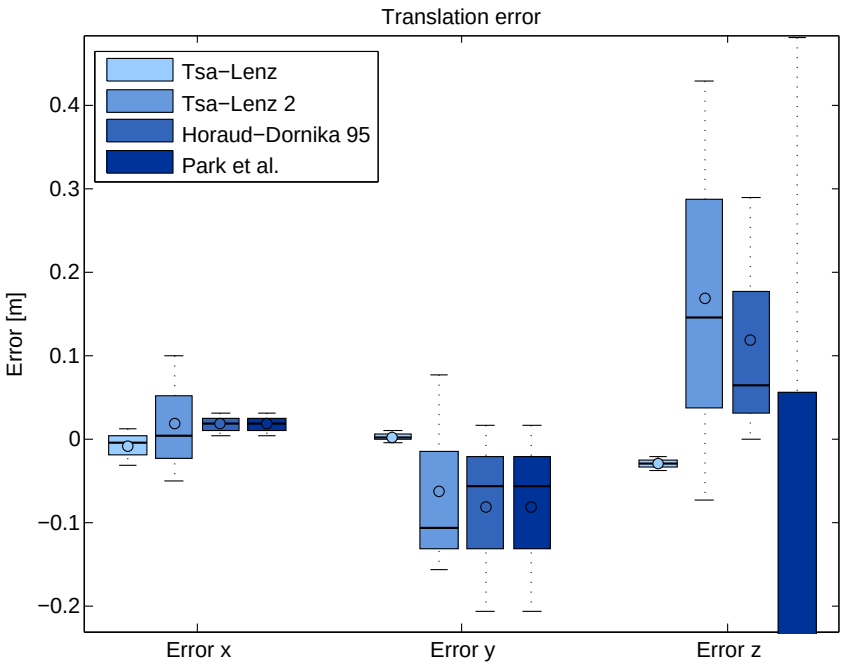


Figure 6-3: Overview of the errors of the different algorithms in Matlab with noise. 144 poses are used. Both rotation and translation noise is varied between -0.02 and 0.02.

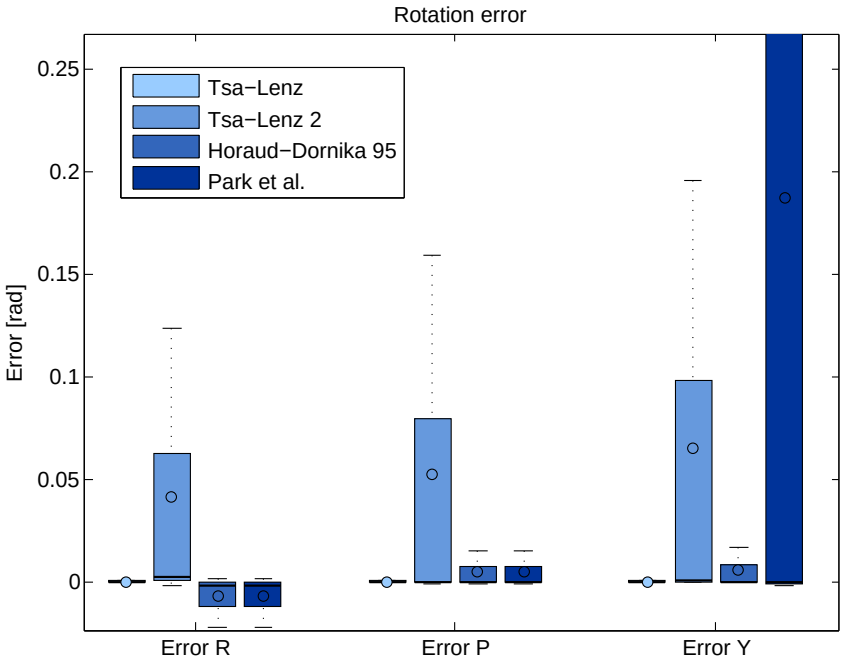


Figure 6-4: Overview of the errors of the different algorithms in Matlab with noise. 144 poses are used. Both rotation and translation noise is varied between -0.02 and 0.02.

Measurement	R	P	Y
ArToolKit	-3.154	29.13	-3.278
ArToolKit+ single	-2.665	-0.9484	10.22
ArToolKit+4	-2.684	-6.966	4.685

Table 6-5: The error of the rotation for different algorithms in milli rad.

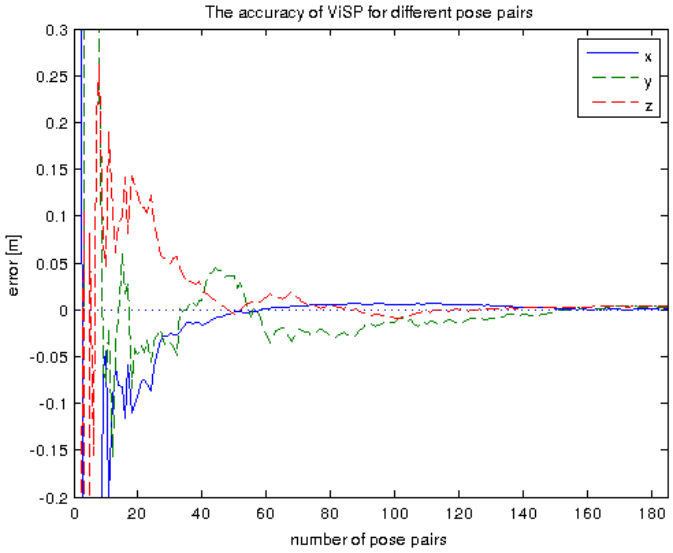


Figure 6-5: The error between the result of ViSP and the correct translation. The dotted line is the zero line.

In figure 6-5 the translations and in figure 6-6 rotation errors for a different number of pose pairs have been determined. The input data are the transformations give by ArToolKitPlus (with light). It can be seen that the error for the translation is very large until about 60 pose pairs. The y and z error remain large and approach asymptotically the zero line. While the x error reaches this much faster. For the rotation the steady state is reached after about 100 pose pairs.

6-4 Discussion

In the tests the camera was fixed on a dual axes arm. This means that the possible movements are limited. An arm with more links has more possible positions a larger displacements for the end effector. This could improve the results.

6-5 Extrinsic calibration using the PR2 robot

To test with a robot arm with more degrees of freedom the extrinsic calibration was also tested with a simulated PR2 robot. A Kinect camera was fixed to the right gripper palm link (r_gripper_palm_link in the robot description). The Red Green Blue (RGB) camera of

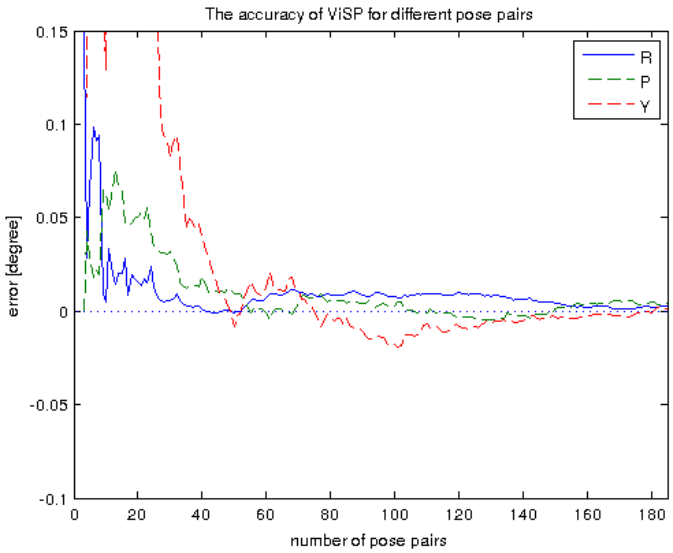


Figure 6-6: The error between the result of ViSP and the correct translation. The dotted line is the zero line.

Axis	ARToolKit	ARToolKit	ARToolKit+	ARToolKit+	ARToolKit+4	ARToolKit+
Config.			single	single	+4	+4
Surface [m^2]	0.5^2	0.18^2	0.5^2	0.18^2	0.2^2	0.08^2
x	3.986	-2.513	4.517	69.23	4.874	128.5
y	-33.88	-60.2	-7.65	405.2	17.83	698.8
z	10.1	-1.128	4.743	-295.1	-1.598	-909
RMS	20.54	**34.79**	**5.815**	292.1	10.71	666.1

Table 6-6: The error of the final value of the Tsai algorithm for the translation for the various pose estimators. Average of the last 5 values. Fourth value is the RMS error of x, y and z together, lowest value per system in bold. In millimetres.

Axis	ARToolKit	ARToolKit	ARToolKit+	ARToolKit+	ARToolKit+4	ARToolKit+
Config.			single	single	+4	+4
Surface [m^2]	0.5^2	0.18^2	0.5^2	0.18^2	0.2^2	0.08^2
R	1.825	30.66	9.919	613.5	-0.6962	1173
P	-31.76	-53.46	-5.905	191.2	9.67	118.1
Y	5.018	-5.311	4.481	286.5	7.328	588.8
RMS	18.59	**35.71**	7.149	406.2	**7.016**	761.1

Table 6-7: The error of the final value of the Tsai algorithm for the rotation for the various pose estimators. Last five results averaged, in milliradians. RMS of the R, P and Y value. Lowest value per system in bold.

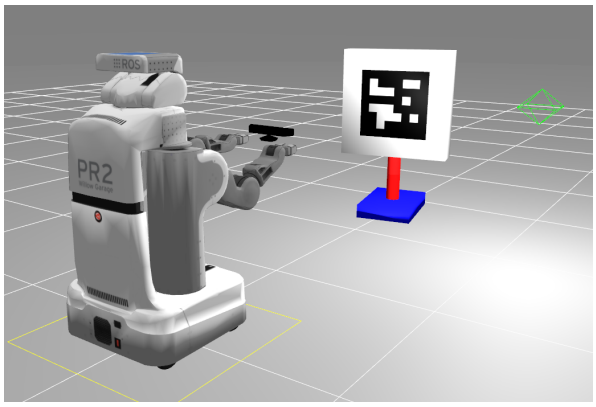


Figure 6-7: Screenshot from the simulation of the PR2 robot in Gazebo. The black bar above the gripper represents the Kinect.

the Kinect was used, camera parameters in appendix D-2. A 0.5 by 0.5 meters marker was placed 2.5 meters away from the PR2 with the center of the marker 0.85 meters high. This means the marker is 1.73 meters away from the Kinect frame (r_gripper_kinect_frame). The calibration was tested for 15 poses. The x error was about 4 centimetre, y error 46 centimetre while the z error was about 1 cm. Rotation errors are in the order of 0.01 degree. The translation results is thus worse than previous results with the 2 Degree Of Freedom (DOF) robot arm. The rotation result is a lot better. It could be that this bad translation result was due to small amount of poses. The result for more poses was however not tested.

Chapter 7

Usage

7-1 Usage

The camera that is calibrated using the hand eye calibration system can be used for several applications. Examples are:

- Workspace mapping for obstacle avoidance
- active collision checking
- Visual servoing: use the camera to grasp something

For the first two options relatively large safety margins are used. For safety the robot should stay away relatively far from the obstacle. Therefore some error in the camera calibration can be tolerated if the boundaries of the possible error are known. For the third option, grasping, the calibration should be as accurate as possible. The calibration error will be probably too high for high precision applications. But usually these systems are position programmed and don't use a camera system. In the Small to Medium Enterprises (SME) handling of goods is an application that has a high interest to be automated. By using a camera system this could be made more flexible. A robot handling system that handles food will therefore be taken as an example application here. A gripper from the company Lacquey will here be taken as an example. This gripper can handle position inaccuracies of about 1 cm.

To calculate the maximal distance on which the object can be we take the error of ARToolKitPlus+4 with 0.5 meter marker, given in table 7-1 and table 7-2.

For a worst case scenario the maximal distance at which the camera can be used to grasp an object will be calculated. The gripper can compensate for a positioning error of 1 centimetre at the object that needs to be grasped. First we take the translation error:

$$(0.01 - \frac{2.857}{1000} - \frac{1.022}{1000} - \frac{0.286}{1000}) = 0.0058 \tag{7-1}$$

Measurement	x	y	z
ArToolKit	1.212	-35.41	-2.789
ArToolKit+	4.502	-5.808	5.859
ArToolKit+4	2.857	-1.022	-0.286

Table 7-1: The error for the translation of the hand-eye calibration for different camera pose estimation algorithm. Dimension in millimetres.

Measurement	R	P	Y
ArToolKit	-3.154	29.13	-3.278
ArToolKit+ single	-2.665	-0.9484	10.22
ArToolKit+4	-2.684	-6.966	4.685

Table 7-2: The error of the rotation for different algorithms in milli rad.

This means that after compensating for the translation error the gripper can still compensate for half a centimetre. If we also take the rotation errors in to account the maximum distance at which the gripper can be used (more steps in appendix A-1) is:

$$\sqrt{\frac{(0.01 - \frac{2.857}{1000} - \frac{1.022}{1000} - \frac{0.286}{1000})^2}{tan(-6.966/1000)^2 + tan(4.685/1000)^2}} = 0.6909[m] \tag{7-2}$$

This means that the camera should be no more than 69 cm away from the object to reliably grasp the object. For this the roll error is not taken in to account as this depends on the size of the object.

Chapter 8

Extrinsic calibration on the UR5 robot

The extrinsic calibration method has been tested in simulation in the previous chapter. In this chapter it will be tested how good the system works in a real robot setup. After this chapter the resulting hand-eye calibration will be used for workspace mapping with the robot.

8-1 Test setup and tests

Target of this chapter is to determine and validate the location of two camera's on the robot arm:

- A Logitech webcam
- A Kinect

The created extrinsic calibration system will be tested on the UR5 robot arm. During the test the system will give the location of the camera. The data required for the calibration will also be saved and used to do a offline extrinsic calibration. By using different subsets of the data in random order the repeatability of calibration result will be tested. The repeatability of the results of other algorithms will also be tested by using their Matlab implementation.

After this the calibration result will be validated by estimating the camera position in two different ways.

8-1-1 Hardware

Robot The Universal Robots UR5 was used again. The Arm Navigation package was used to move the end effector to various positions.

The UR5 robot arm has according to the specifications a repeatability of ± 0.1 millimetres.

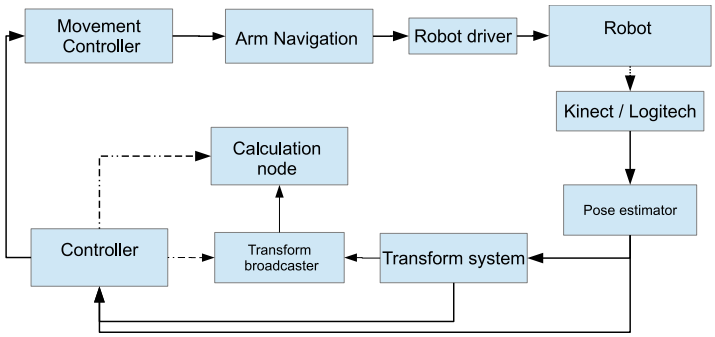


Figure 8-1: Block diagram of the extrinsic calibration system used with the UR5 robot arm. Images from the Kinect or the Logitech webcam are processed by the Pose estimator which estimates the position of the checkerboard. This is then send to the ROS transform system where the position of the checkerboard can be used by the calculation node and the Controller node. The Controller node is also responsible for logging all the transformations to a text file.

Camera system In separate tests the Logitech 9000 camera and a Microsoft Kinect were used. The Logitech camera was mounted approximately 7 cm above the Tool Centre Point (TCP). While the Kinect was mounted approximately 14 centimetres above the TCP. Image acquisition was done at 640x480 pixels.

8-1-2 Software

A similair software system as used in the simulation was used. In figure 8-1 a block diagram is given.

Controller A ROS node which commanded the arm to a set of predefined TCP positions was used. This node sent the TCP position to the Arm Navigation software. After a point was reached the transformations between various points were collected using the tf node and logged to a file. To cancel the effects of possible vibrations from movement the program paused for several seconds before taking the measurements so that the vibrations could dampen out.

8-2 Calibration results

The following hand-eye transform was obtained for the Logitech webcam:

$$T_{TCP}^{cam} = \begin{bmatrix} 0.0052 & -0.0523 & 0.9986 & 0.0266 \\ -0.9994 & 0.0328 & 0.0069 & -0.0162 \\ -0.0331 & -0.9981 & -0.0521 & 0.0777 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8-1}$$

For the Kinect the average result of three tests was taken:

$$T_{TCP}^{cam} = \begin{bmatrix} 0.99425 & -0.0066764 & -0.10684 & 0.1035 \\ 0.0065052 & 0.99998 & -0.0019509 & 0.0418 \\ 0.10686 & 0.0012447 & 0.99427 & 0.1447 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8-2}$$

For the Logitech webcam 48 points were used, although the translation result is after 10 poses already in the millimetre range near the final value. For the Kinect 3 sets of 22 measurements were averaged.

8-3 Repeatability test

To test the repeatability of the calculated hand-eye transforms the hand-eye calculation has been done a 100th times with a random 70% subset of the measurement data from the Logitech camera. Two Tsai implementations, Horaud et al. '95, Ernst et al. (OR24e) and Daniilidis et al. in Matlab were tested. The same data was also send to the Tsai ROS implementation. For this a new node was created which reads in the same logfiles as the Matlab uses and publishes this over the ROS topics used by the calculation node.

8-3-1 Results

In figure 8-2 and in figure 8-3 the calibration results are given. It can be seen that the three Tsai implementations give slightly different results. The ROS implementation gives the lowest spread. The Horaud et al. '95 implementation has the largest spread. If we look a the ROS implementation the the difference between the minimum and maximum are: for x approx. 1 cm, y approx. 0.5 cm and for z approx. 0.5 cm. This gives and indication of how much the calculated value can be off in the most extreme case.

8-4 Validation of the extrinsic calibration

To validate the extrinsic calibration of the camera ground truth is necessary. If the exact location and orientation of the camera is known the accuracy of the calibration result can be validated. But it was proven very difficult to measure the exact location and orientation of the camera. The housing of the camera is rounded which makes it difficult to mount it parallel or orthogonal to a link of the robot. And there is no translation known from the mounting point to sensor. The rotation of the sensor in the housing is also not known. Therefore it would difficult to get a good estimate of the transformation from TCP to camera. It would be easier to use an industrial camera with known transformations. This was however not available.

To get an estimate of the accuracy it is possible to use the estimated position of the checkerboard. This can be measured in two ways relative to the robot base:

- Using the camera and the calculated transform of the checkerboard pose estimator.
- By placing the robot TCP to the checkerboard and using the encoders of the robot arm to measure the position of the TCP.

With these two transformations to the same point an error can be calculated. See figure 8-4 for the transformations. The error was determined at the camera coordinate system and not at the checkerboard. This location is chosen as a slight error in the rotation of the

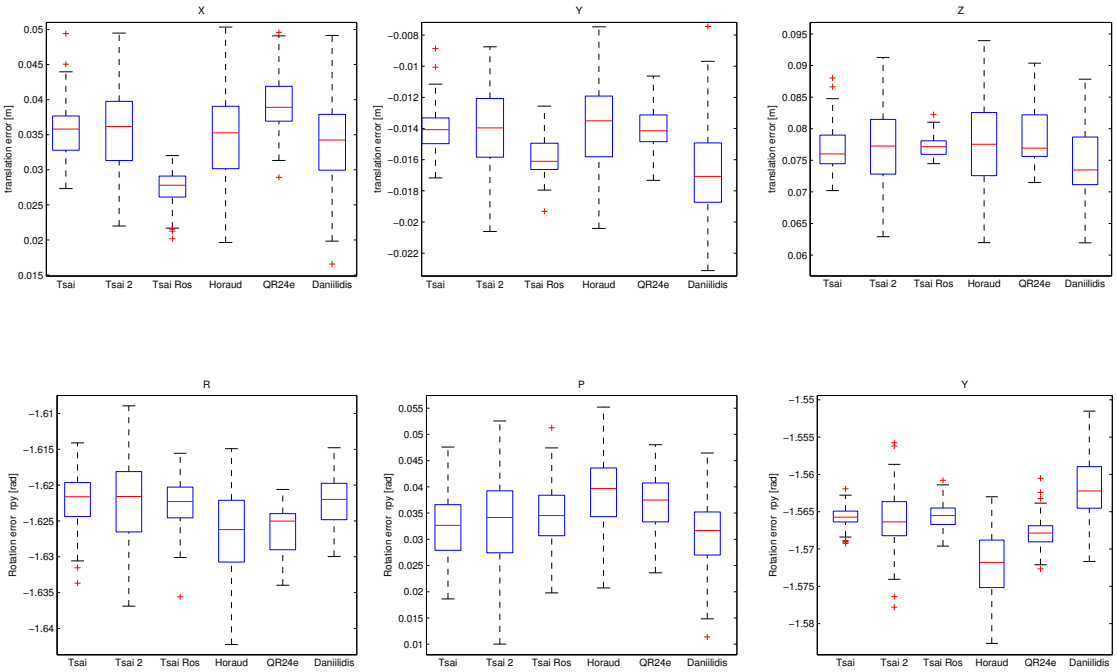


Figure 8-2: Translation and rotation results for the four algorithms with different random subsets (70%) of the data. Translation on top, rotation below (in Roll Pitch Yaw notation). It can be seen that the Tsai et al. algorithm has a low spread compared to the other algorithms.

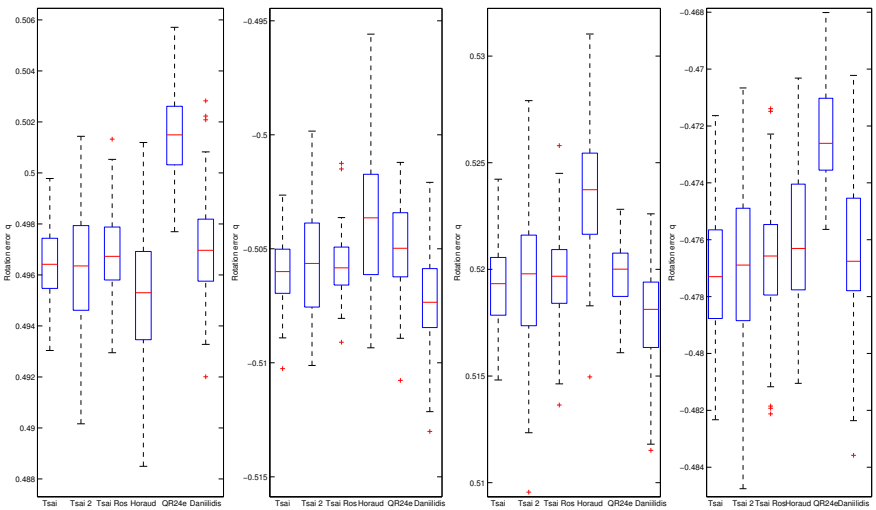


Figure 8-3: The calibration rotation results in quaternions. It can be seen that the ROS implementation of the Tsai et al. algorithm has a relatively low spread. The QR24e algorithm deviates a bit from the rest of the algorithms.

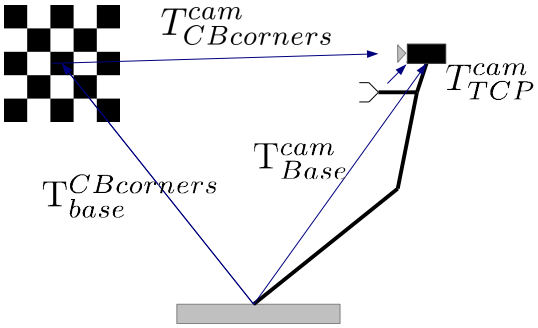


Figure 8-4: Overview of the transforms on used to calculate the error. The position of the camera is calculated by calculating the position using the base to CB and CB to cam transforms and the base to cam transform.

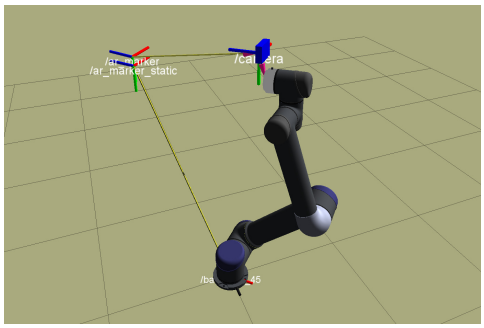


Figure 8-5: The same visualization but now in the visualization tool of ROS visualizing the real estimated checkerboard position and a transformation from base to checkerboard to check this. There is some offset between the estimated and 'correct' transform.

Figure 8-6: The TCP of the UR5 with a bolt fitted in the top left corner thread. The tip of the bolt is pointed at the centre of the checkerboard. The positions of crossings of the checkers are also measured in this way.

calculated hand-eye transform gives a large deviation on the checkerboard position. If the camera location is used the translation and rotation error are not coupled.

To determine the exact location of the checkerboard a bolt with cut off bolt head was inserted in top left thread on the TCP. Using a sharp point on the bolt tip it could be positioned on the centre of the checkerboard and the crossings of the checkers. In figure 8-6 an image of this is given.

Using manual positioning the tip was positioned on the centre of the checkerboard, the outer corners and a part of the inner corners of the checkerboard. By translating the TCP point to the end of the bolt the position of the checkerboard outer corners in robot coordinates ($T^{CBcorners}_{base}$) are found.

$$T^{CBcorners}_{base} = T^{ee\_link}_{base} T^{be}_{ee\_link} \tag{8-3}$$

Where $T^{ee\_link}_{base}$ is the position of the end effector and $T^{be}_{ee\_link}$ the translation of the TCP to

the end of the bolt (all homogeneous coordinates).

The center of the checkerboard is found by using the Matlab ABSOR package[1] which uses Horn's quaternion-based method. It can calculate the translation and rotation of the checkerboard frame in base coordinates using least squares. This is calculated with the translations of the checkerboard corners in base coordinates and the corners in checkerboard frame coordinates. The result of this is displayed in figure 8-7. The blue dots represent the TCP coordinates in the base frame and the green dots the corner coordinates using the translation. The axis system through it gives the transformation of the checkerboard in base coordinates (T_{base}^{CB}).

By using the measured transform from checkerboard to camera (the inverse of which was measured with the camera pose estimator) the position of the camera can be calculated:

$$T_{base}^{camCB} = T_{base}^{CB} T_{CB}^{cam} \tag{8-4}$$

The position can also be calculated by transforming through the robot links and using the calculated hand-eye transform:

$$T_{base}^{camrobot} = T_{base}^{ee\_link} T_{TCP}^{cam} \tag{8-5}$$

The difference between the two transformations is then the error:

$$T_{camrobot}^{camCB} = T_{base}^{camrobot} - T_{base}^{camCB} \tag{8-6}$$

8-4-1 Discussion

There are a few limitations in the measurements which limit the accuracy of the result:

- The checkerboard corners are not measured very precise as the robot is manually positioned to the corners. The board can also deform a bit. If we look for instance at the height difference between the top and bottom corners this has an error of 4 mm on the left and 2 mm on the right.
- Errors in the estimate of the rotation of the checkerboard give a large offset at the camera position because of the distance.
- The camera pose estimator is not very precise.
- The UR5 accuracy is not specified by the manufacturer but the repeatability is specified at ±0.1 mm. This error occurs both in the measurement of the location of the checkerboard using the robot and in the measurement using the camera.

To limit the influence of sources of error given above multiple measurements have been taken. For the checkerboard position and orientation 20 corners have been measured. For calculation of this the least squares method is used which gives a maximum error of 1.2 millimetres for the second set.

[1] http://www.mathworks.com/matlabcentral/fileexchange/26186-absolute-orientation-horns-method

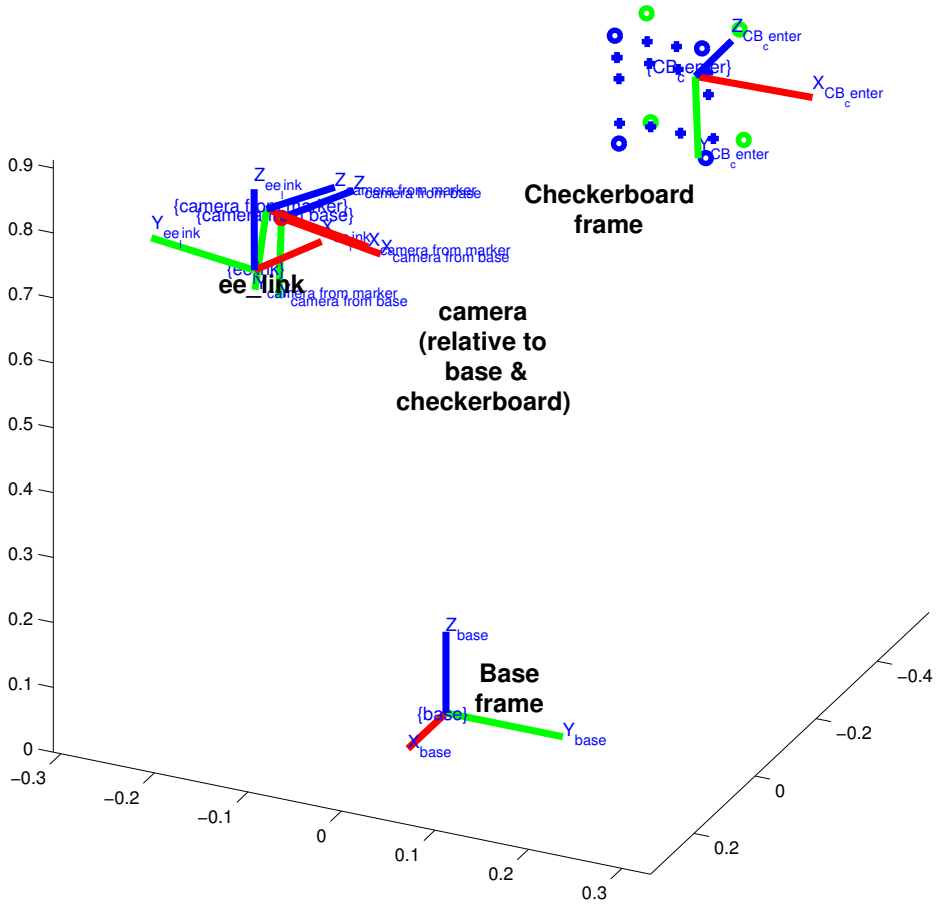


Figure 8-7: Visualisation of the coordinate frames of the robot with the Logitech camera. On the left top the TCP frame (ee_link) with the two camera frames (relative to the TCP and relative to the checkerboard) next to it. On the top right the checkerboard with in blue circles the TCP positions of outer corners in circles and crosses for the inner. In green the checkerboard positions. Dimensions in meters.

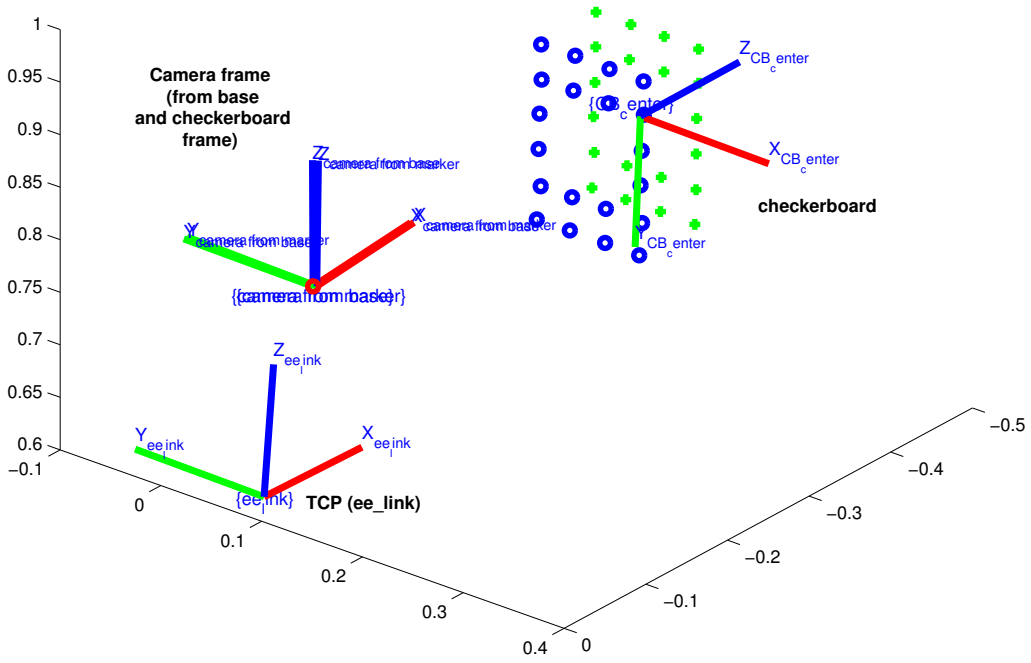


Figure 8-8: Visualisation of the coordinate frames of the robot with the Kinect. On the left top the TCP frame (ee_link) with the two camera frames (relative to the TCP and relative to the checkerboard) next to it. It can be seen that the frames almost overlap, better than with of the Logitech camera. On the top right the TCP positions of the corners of the checkerboard in blue circles. In green crosses the checkerboard corners. Dimensions in meters.

	Translation Error [mm]			Rotation Error [degrees]			
Camera	x	y	z	x	y	z	Measurements
Logitech webcam	−14.381	7.974	−16.768	4.607	−1.839	−1.500	7
Kinect	−12.854	−10.254	0.414	−0.798	0.558	0.258	11
Logitech Webcam (test II)	−13.864	4.915	−14.053	4.3282	−2.236	−2.3629	19
Kinect (test II)	−6.383	−4.858	−2.115	−0.910	0.221	0.125	31

Table 8-1: Translation and rotation error (difference between the axis systems from figure 8-7. For the Logitech webcam an average of three measurements on different positions was taken. For the Kinect 11 measurements were taken.

By taking 10 measurements from the same position an estimate for the standard deviation of the error from the pose estimator can be made. If we look at table 8-2 it can be seen that in the worst case this is approximatly 0.6 millimetres for the Logitech camera and 0.2 millimetres for the Kinect. This is however from one pose, the error could vary from position to position. If we look at the standard deviation in chapter 4 it can be seen that the error could also be in the order of a centimetre.

The error can be calculated for multiple poses round the checkerboard. For the second test the error has been calculated 19 times for Logitech webcam and 21 times for the Kinect. With this a measure for the overall precision can be made. If we look at table 8-2 it can be seen that for the Logitech webcam the standard deviation for the translation is approximately half a centimetre for the x and y axes while it is almost a centimetre for the z axis. For the Kinect the standard deviation is much less, 3 millimetre for the x and round 1 for the other axes. The rotation standard deviation is also a less than that of the Logitech webcam.

8-4-2 Conclusion

Using equation 8-6 the difference between the two axes systems can be calculated which forms the error. This error can be calculated for different viewpoints. In table 8-1 and in table 8-2 the results are given. It can be seen that the calibration result is correct in the order of centimetres per axis. For the rotation the error is for the Kinect less than a degree. For the Logitech webcam the error is larger, the largest error is 4.6 degrees.

		Logitech Webcam	Kinect RGB	Logitech Webcam static	Kinect RGB static
Translation [mm]	x	−13.864	−4.2605	−16.16	−10.841
	std	6.6539	3.3922	0.061 578	0.0951
	y	4.9159	−5.3612	4.2203	−3.8038
	std	4.1217	1.1585	0.606 53	0.2307
	z	−14.053	−2.6687	−14.476	−0.9525
	std	8.9131	1.3441	0.125 37	0.2218
Rotation [deg]	X	4.3282	−0.9082	4.2298	−0.9147
	std	0.338 54	0.0687	0.0138	0.002 24
	Y	−2.236	0.1143	−2.2198	0.4465
	std	0.131 61	0.2006	0.0011	0.0238
	Z	−2.3629	0.0051	−2.5439	0.3797
	std	0.238 14	0.1624	0.0583	0.0258

Table 8-2: Translation and rotation errors and their standard deviations for the Logitech webcam and the Kinect. For the Logitech camera 19 poses have been used. For the Kinect 21 poses have been used. The static measurements use 10 measurements from the same position.

Part III

Workspace mapping

Chapter 9

Workspace mapping

In the previous chapters methods for calibrating the camera extrinsically were discussed and tested. This calibration will now be used to create a 3D map of the environment of the robot. Using the calibrated position the 3D map can be created relative to the robot base. And the camera can be moved to a position relative to the base to scan an unknown area.

9-1 Target

With the extrinsic calibration, as discussed in the previous chapters, the robot can now be easily equipped with a camera system. This is the first step in increasing the flexibility of the robot system. Next step will be the automatic creation of a 3D obstacle map of the surroundings of the robot. With this the robot arm can be easily used for different tasks in different places in the factory. Normally when the robot arm is put in a new environment the location of the obstacles have to be programmed in the motion planning system. With this mapping system this isn't necessary any more, the robot can be easily moved and started. Which decreases the required setup time. The robot should create a 3D map of its direct surroundings. This map should contain all obstacles so that the robot can be safely used after the mapping and can plan around all the obstacles. While creating the map the robot does not know the locations of possible obstacles yet. Therefore it should not move in to unknown space. The idea is that the robot has a small 'safe zone' which is clear of obstacles. From this zone the robot can scan the surroundings and map all the unknown space.

9-2 Problem type

Determining the placement for a sensor to explore unknown areas is known as Next Best View (NBV) planning. The Next Best View approach was first discussed in [58]. Classical problems related to this are 3D modelling of objects, the art gallery problem and pursuit and evasion. The problem discussed here is related to the 3D scanning of objects. Only with this

problem the robot is limited in the space it can move in, as it can not move in unknown space. And the whole environment is to be modelled. Compared to other exploration approaches like SLAM (Simultaneous localization and mapping) this problem is different as the robot is fixed and the location of the sensor is known.

9-3 Related Work

In 'Hierarchies of octrees for efficient 3D mapping' [59] a hierarchy of octree maps is used to model the environment. It is not clear if the current OctoMap implementation also uses this. OctoMap can subdivide the voxels but it is not clear if really separate maps are used in this paper. A NBV approach is used for testing with a PR2 robot. The robot can drive round the table to obtain better views. The hierarchy octree is less memory intensive than the monolithic octree.

In 'Sensor-based Exploration for general robotic systems' [60] and the following paper 'An exploration method for general robotic systems equipped with multiple sensors' [61] exploration is done using a expanding boundary. For this octrees are used with sensors like stereo cameras or lasers. It seems that their current implementation does not deal with uncertainty of the measurements. They do suggest how this can be done. It was not tested on a real robot.

In [62] extra information is added to the voxel, information about by which sensor a cell is seen is added. They also store information about if the neighbour cell is occupied or not.

In [63] a NBV algorithm is used with Kinect mounted on a PR2 robot. This is done to deal with occlusions in a cluttered environment. Not only the point with the highest information gain is selected but also the probability of seeing the point is taken into account. A Probabilistic Roadmap (PRM) planner is used to optimize the route to the point for the highest information gain. This study does not go in to safe scanning.

In [64] the known 2D frontier-based exploration methods are extended towards 3D environments. This is done by calculating voids, which are unknown areas which are not occluded or enclosed. These area's are combined with the frontiers (free cells with a unknown cell next to it) to get the best viewpoint. The algorithm is tested on a Mesa Robotics matilda robot platform with a Schunk custom-built 5-DOF manipulator on it. The author notes that the system needs to be improved to be used in real time.

In [65] a workspace mapping approach with multiple sensors is researched. A combination of the DLR Laser-Range Scanner, stereo vision and a single and dual laser stripe profile projector with camera are used. The approach is tested on a Kuka KR16 in a Small to Medium Enterprises (SME) usage scenario.

9-3-1 Depth sensor

The Kinect has been chosen as depth sensor in paragraph 1-2-3. It can measure depth using an InfraRed (IR) projector and an IR camera and also take normal images with an Red Green Blue (RGB) camera. The RGB camera was used to determine the location of the camera using the extrinsic calibration. As the IR camera and the RGB camera are fixed with a known

relative distance to each other the location of the IR camera is then also known. But it is also possible to use the IR camera for the extrinsic calibration directly (for accurate results the IR projector should then be covered and an external IR light source should be used[1]). Specifications for the depth camera are given in table 9-1. To understand how the Kinect can be used best with workspace mapping we also look in to the accuracy and range of the Kinect. This also important to determine the mapping resolution.

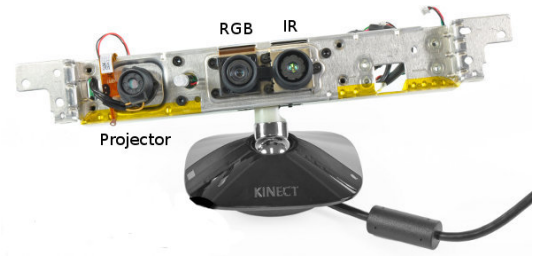


Figure 9-1: The Kinect sensor with its cover removed. From left to right: the infrared projector, the RGB camera and the IR camera. The IR projector and camera are approximately 7.5 centimetres apart. Source: iFixit[2]

Accuracy and precision According to [12] the resolution of the depth measurements decreases quadratically. It is about 2 millimetres at one meter and at 3 meters 2.5 centimetres. While at 5 meter it is increased to about 7 centimetres. This is because of the IR grid the Kinect uses to calculate the depth. The features in the grid lay further apart at larger distances thus the resolution decreases. The random error in the depth measurements is a few millimetres at half a meter which increases quadratically to 4 centimetres at 5 meters. For mapping applications the range they are recommending is therefore 1-3 meters.

For the Openni driver of ROS the precision was evaluated[3], see figure 9-2. This graph was created by pointing the Kinect at a wall and fit a square using RANdom SAmple Consensus (RANSAC) on the point cloud. The fitted wall gives then the average distance. The graph shows how far off the measurements are from this average.

Range The Kinect depth sensor range is according to Microsoft[5] minimum 0.8m and maximum 4m. The Kinect for Windows Hardware can however be switched to Near Mode which provides a range of 0.5 m to 3m instead of the Default range. The OpenKinect page specifies 0.8 to 3.5 meters[6]. In [67] a minimum distance of 0.6 is given.

A small own test with the Kinect pointing at a wall indicated that the minimum range is 0.45 metres. If an object is placed before the wall at this range a hole with the contour of the object is created in the depth data. This should be taken into account when implementing the NBV algorithm.

[1] http://wiki.ros.org/action/show/openni_launch/Tutorials/IntrinsicCalibration?action=show&redirect=openni_camera%2Fcalibration
[2] http://www.ifixit.com/Teardown/Microsoft+Kinect+Teardown/4066
[3] http://wiki.ros.org/openni_kinect/kinect_accuracy
[4] http://wiki.ros.org/openni_kinect/kinect_accuracy
[5] http://msdn.microsoft.com/en-us/library/hh438998.aspx
[6] http://openkinect.org/wiki/Imaging_Information

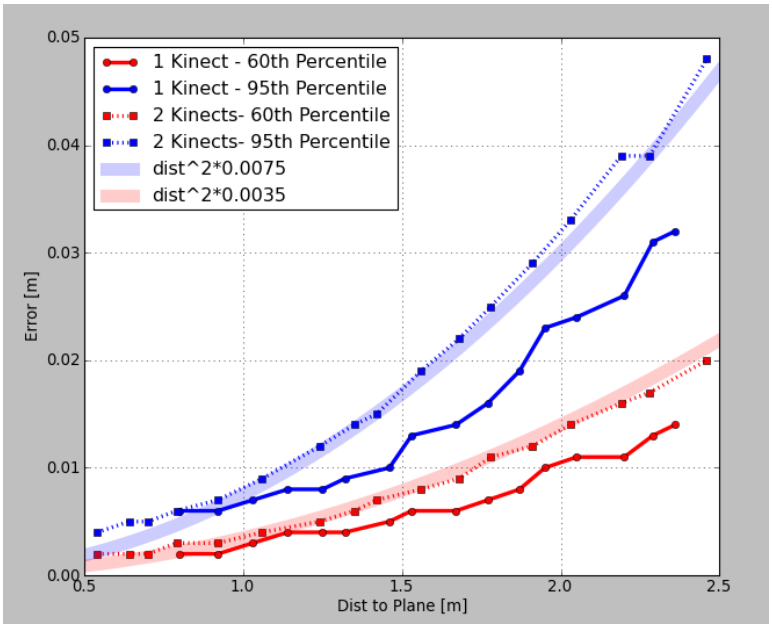


Figure 9-2: Kinect Precision for two Kinects[4]. Tested by pointing the Kinect at a wall for various distances and fitting a plane on the pointcloud. This provides the average distance to which the distance to the other points is measured. The used resolution is not specified.

Parameter	Value	Value Microsoft	Primesense	Unit
Viewing angle vertical	43	45.6f	45	°
Viewing angle horizontal	57	58.5	58	°
Frame rate (Gazebo)	30			fps
Frame rate (max)	60			Hz
Farclip (Gazebo)	5			
Resolution	1280x1024@15Hz, 640x480@30Hz (default)			
Min. (Gazebo)	0.8 ([66] cite 0.5)			meters
Max. distance real Kinect	5(3 on near range)	0.8m - 4.0m	0.8m - 3.5m	m

Table 9-1: Kinect parameters from [7],[8]. According to the Kinect for Windows documentation the nominal viewing angles are 58.5°and 45.6°[9].

It possible to add a lens before the Kinect lenses to decrease the minimum range. The Nyko Zoom lens (misleading name, it is actually a wide angle lens) can do this for instance[7]. This makes the Kinect lose depth information at the edges however[67].

9-3-2 Map storage methods

Several map storage methods exist. A small comparison between map storage methods was made. The map should represent free, occupied and unknown areas.

In table 9-2 an overview of different map storage methods is given with their advantages and disadvantages.

The OctoMap system is chosen as it can deal with unknown space, is very efficient and it has good support in Robot Operating System (ROS). Due to its probabilistic storage it can deal with noise from the sensor. With the ROS implementation it is also possible to use multiple depth sensors that add data to the map.

9-3-3 Octrees

As the OctoMap system is an important part of the mapping system a short overview of the system will be given (based on [4], [59] and [5]).

Octrees are hierarchical data structures which describe the state of a volume of space. Each node in a Octree represents the space contained in a cubic volume which is usually called a voxel[4]. This volume can be subdivided in to eight sub volumes, hence the name. The sub volumes can be subdivided again until the minimum voxel size is reached. As the tree is an hierarchical structure the readout can be stopped at a higher level to get a coarser subdivision of a volume. This makes it possible to use different resolutions for different applications. For instance a the planner of robot platform doesn't need as high resolution as the robot manipulator on top of it for navigation. In figure 9-3 an example of the different resolutions of the same map is given.

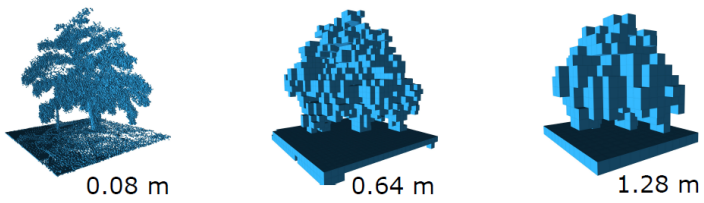


Figure 9-3: By changing the depth of query multiple resolutions of the same map can be generated at any time. Reproduced from [4]. In the first picture the resolution (size of the nodes) is 0.08 meters, the second 0.64 meters and in the third 1.28 meters.

Voxels mark the space as free or occupied. Unknown areas are marked as uninitialized spaces in the Octree. The known nodes are probabilistic, each node encodes a probability of how certain it is that a node is occupied or free. Using a probabilistic approach instead of a binary

[7] http://www.nyko.com/products/product-detail/?name=Zoom

Type	Advantages	Disadvantages
Point Clouds	• No discretizaton of data • Map area not limited	• Unbounded memory usage • No representation of free or unknown space • Cannot easily cope with sensor noise • No Unknown state, but it can be added with a custom type[8]
3D vixel grids	• Probabilistic update • Constant access time	• Memory requirement – Complete map is allocated in memory – Extent of map has to be known
2.5D (elevation) maps	• memory efficient	• Memory requirement • Not completely probabilistic • No distinction between free and unknown space • Cannot model underpasses • Cannot store free or unknown areas in a volumetric way
Octrees	• Encodes free, occupied and unknown space by default • Probabilistic, helps with sensor noise • Flexible multi resolution • Memory efficient	• Data is discretized, information is lost • Loop closure can be difficult (less of a problem with a fixed robot) [68] • traversability estimate not very precise

Table 9-2: Different map building methods.

representation helps dealing with false data which is added to the map due to noise in the sensor readings. The more an occupied space is seen the higher the occupancy probability of the voxel will be.

The probability $P(n|z_{1:t})$ of a leaf node n to be occupied given the sensor measurements $z_{1:t}$ is estimated according to:

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{9-1}$$

Where $P(n|z_t)$ the probability of the node n being occupied given measurement z_t is. $P(n|z_{1:t-1})$ the probability based on the previous measurements. And $P(n)$ the prior probability.

Using the log odds notation equation 9-1 can be rewritten (in appendix A-2 the steps required are described) as:

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{9-2}$$

With:

$$L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{9-3}$$

This formulation is used as it provides faster updates as multiplications are replaced by additions. If the sensor models are pre computed than the logarithms do not need to be recalculated during the update of the probabilities.

A threshold is used to discriminate between the free and the occupied state. Each observation that a voxel is occupied increases this probability. And then needs the same amount of observations that the voxel is empty to get it back to the same state again. This makes the map however less dynamic, a voxel which is observed as occupied several times will not change back to free fast. This can be problematic however in dynamic environments. Therefore clamping thresholds are introduced. These prevent the value from growing to much in one direction by clamping the value to a minimum or maximum value. This gives a different update rule:

$$L(n|z_{1:t}) = max\Big(min\Big(L(n|z_{1:t-1}) + L(n|z_t), l_{max}\Big), l_{min}\Big) \tag{9-4}$$

Where l_{min} and l_{max} the minimum and maximum log-odds values are. This has the advantage that the map can change faster. Due to the clamping neighbouring nodes get the same value which makes it possible to compress these voxels with pruning.

With pruning nodes with the same probability can be merged to achieve compression.

Sensor model OctoMap can use any distance sensor which can sense the distance to the detected endpoint (so discrete infra red collision detection sensors cannot be used). It uses a beam based inverse sensor model to determine which voxels need to be updated. To do this a 'virtual ray' is cast between the sensor origin and the endpoint (see figure 9-4). The volumes

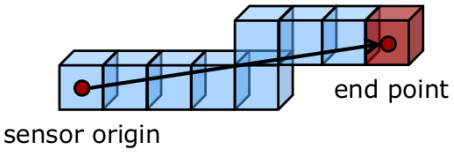


Figure 9-4: The voxels through which the ray traces from sensor origin to its endpoint. Adapted from [5].

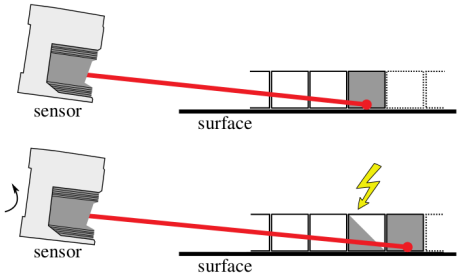


Figure 9-5: A sweeping laser scanner first lists a voxel as occupied (gray) but in the next scan line the voxel is updated to free again. Adapted from [5].

through which the ray travels are updated with the free probability and the endpoint with the occupied probability. In

$$L(n|z_t) = \begin{cases} l_{occ}, & \text{if the beam is reflected in the volume} \\ l_{free}, & \text{if the beam is reflected} \end{cases} \tag{9-5}$$

In log-odds the occupied value l_{occ} is positive and the free value l_{free} is negative. For instance $l_{occ} = 0.85$ and $l_{free} = -0.4$, corresponding to probabilities of 0.7 and 0.4 for occupied and free volumes.

This sensor model can however lead to holes in the scan when a sweeping sensor is used. An example of this is given in figure 9-5. This can be prevented by treating a set of scan lines as one update.

Extensions

In [69] a method is presented to incrementally update the map based on detected changes which is incorporated in OctoMap version 1.5. In [59] hierarchies for OctoMaps are used, creating a hierarchical data structure of sub maps which can be updated individually and can have different resolutions. This means that for instance for a mobile robot a coarse resolution can be used for the overall map while sub maps, the top of a table for instance, can have a much finer resolution. With this approach the whole map doesn't need to use the same finer resolution.

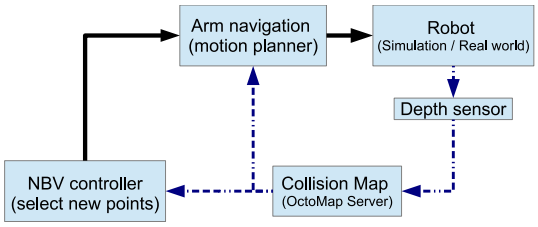


Figure 9-6: Block diagram of the Next Best View system. The same system (with different parameters and models) can be used for the simulated PR2 robot or the real UR5 robot arm. The lines from the Kinect to Information gain and Arm navigation nodes represent a data stream containing a 3D map of the environment. The thick solid lines represent service calls.

9-4 Proposed design

To create a map of the surroundings an empty OctoMap is used at the start. The robot cannot move through unknown space. By using a depth camera the robot scans the room and checks if an area is free or occupied with an obstacle. The camera will be mounted on the end effector because this has the most dexterity. This means however that the robot cannot clear the room around the robot. Therefore there should be a small 'safe' zone free of obstacles round the robot so that it has some movement freedom in the beginning. A NBV algorithm will be used to generate new points (viewpoints). A number of points are generated and for each point it is tested in the existing map how much new information about the environment is obtained from this point. The point with the greatest information gain is selected and the end effector is moved to this point. While planning the trajectory to this point the planner uses the collision map to plan around occupied and unknown space. It cannot move through unknown space as there could be an obstacle present there. The extrinsic calibration will be used to get the position of the sensor on the robot.

In figure 9-6 an overview is given of the implementation of workspace mapping system. The working of the system will be discussed by following the arrows from the 'Robot' block.

Robot A robot arm with a depth sensor somehere on a moveable link. Preferably an outer link (near the Tool Center Point, TCP) as this link has more dexterity.

Depth sensor The depth sensor is mounted somewhere at on a robot on a link which can move. In the robot description the position of the sensor on the link is given. This can be obtained using the extrinsic calibration. The 3D sensor produces a 3D point cloud of perceived points in the environment of the sensor.

Collision map An Octomap map server which records the 3D data from the depth sensor and stores this in the map.

NBV controller The NBV controller selects a number of points. With the map data the information gain (how much unknown spaces are seen from the viewpoint) per point is calculated. The point with the highest gain is selected and send to the Arm Navigation Node.

Arm navigation This node receives a point from the NBV controller and plans a route from the current camera position to the given camera position. The Arm navigation then checks the route against a map from the Collision map server. By using the robot description file (which tells how big all the links are) the route is checked for collisions with objects and traversal through unknown space.

9-5 Conclusion

In this chapter a method for scanning the workspace has been selected. Also a way to store the data from the scans has been selected. In the next chapters these will be implemented, first in simulation and after this on a real robot.

Chapter 10

Workspace mapping: Simulation

In the previous chapter a design for a 3D workspace mapping system has been proposed. The best way to implement this will be tested in simulation first. After this it will be tested on a real robot.

10-1 Goal

Goal of this chapter is to investigate how the proposed design can be implemented and how this will perform.

10-2 Related software

Some software was already available in ROS so this was tested first.

10-2-1 Next best view in ROS

The next best view package in ROS[1] was reviewed. It seems to be made for a mobile robot with a laser scanner. As this project has not been updated for a long time it had several problems with changed code interfaces. It uses a combination of OctoMap and the Point Cloud Library (PCL). Because of these (practical) problems and the limited documentation it was chosen not to use this package.

10-3 Simulation with the PR2

It was not possible to get the UR5 robot working in simulation using the Gazebo package. Three ROS versions (Fuerte, Groovy and Hydro) with different versions of Gazebo were

[1] http://www.ros.org/wiki/next_best_view

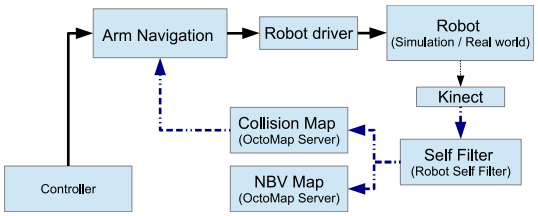


Figure 10-1: Block diagram of the simulation set-up.

tested but all had problems in some subsystem. Therefore the usage of other robot arms available in simulation was investigated. The robot arm needed to have at least 5 Degree Of Freedom (DOF) to be able to attain a different poses and orientations. The PR2 robot was found much better supported and also has arm which can be used for workspace mapping. The arm has seven DOF so this complies with requirement. The PR2 is however a large moveable robot with a torso, head and two arms. This meant that all the room around the robot can not be scanned as the torso is blocking the view.

10-3-1 Simulation set-up

In figure 10-1 the overview of the simulation is given. Switching back from the simulated PR2 to the real UR5 later would take some time. Therefore it was chosen to test only using a predefined set of point and test the NBV algorithm only on the real robot.

Octomap server

The robot should not move through occupied space but also not through unknown space. The standard motion planning software of ROS, the arm navigation package, can however not be configured to plan around unknown space. The flexible collision library (FCL) might offer support for this in the future. But this whole package is still under development and not ready for release[2].

The arm navigation package will plan around occupied space in the OctoMap. To deal with unknown space it was chosen to initialize the map with occupied space and load this map at the start. In this way the robot can not move through unknown space. It can 'clear' the occupied space again by looking at the occupied area. See figure 10-3.

This does give problems with generating the Next Best View (NBV). To determine the best new viewpoint the NBV algorithm will look for the viewpoint in which the most unknown space will be seen. This will be then the view that sees the most occupied space as this also represents unknown space. But it can then not differentiate between occupied space which represents an object and unknown space. The NBV algorithm will thus keep looking at the same occupied area thinking it has not been seen yet.

Therefore it is necessary to use two OctoMaps:

- One OctoMap which is used for the collision checking which is initialized to occupied. This will noted as the collision map.

[2] http://gamma.cs.unc.edu/FCL/fcl_docs/webpage/generated/index.html

- One OctoMap that is used for generating the next best view. This map will not be initialized. This map will be noted as the NBV map.

The collision map will not be fully initialized with occupied space. Only the space reachable by the robot needs to be initialized to prevent movement in to unknown space. It is possible to initialize the whole space but this will require extra memory while it has no added functionality.

Arm navigation

The arm navigation package was used to plan trajectories between points in Cartesian space. The arm navigation package used the (occupied) collision map to check for collisions.

Self Filter

The 3D data obtained by the sensor is filtered through a filtering node. This node uses the robot shape (from the Universal Robotic Description Format (URDF) description on the parameter server) to filter out parts of the robot that could be blocking the view in the image. This prevents that parts of the robot are added to the 3D map. Although these parts could be indeed an obstacle it is not necessary to add them to the map as the motion planning node (the Arm navigation node) can also check for self collisions without the map. And parts of the robot can also move so the collision map is not a reliable way to check for self collisions. For the UR5 robot arm self occlusions are not very common in the current setup. But for the PR2 robot for instance it is quite common that the other arm is blocking the view. For self filtering several packages are available in ROS package system:

- Robot Self Filter[3]
- RGBD Self Filter[4]
- Camera Self Filter[5]

The Robot Self Filter is part of the Arm Navigation Stack. Camera Self Filter (from the Bosch stack) seems to be for RGB camera's only. The RGBD self filter package claims to need 50-70 ms to filter and to be significantly faster than the Robot Self Filter. This probably because it is GPU based, unfortunately it is not downloadable any more. Therefore Robot Self Filter is used. In appendix C-1 the used parameter are listed.

10-3-2 Results

In figure 10-3a the occupied map with the PR2 robot is given. The wall around the robot limits its movements. Next to it in figure 10-4a the NBV map is given. It can be seen that is

[3] http://www.ros.org/wiki/robot_self_filter
[4] http://www.ros.org/wiki/rgbd_self_filter
[5] http://www.ros.org/wiki/camera_self_filter

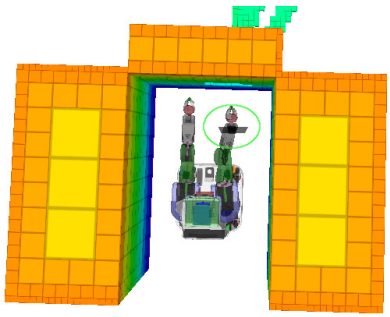
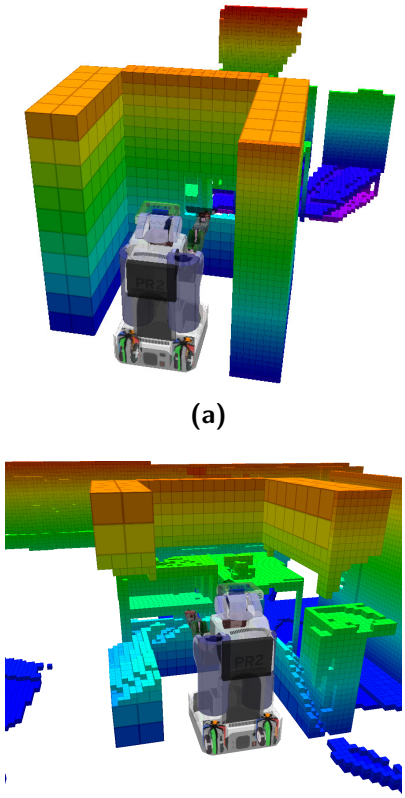


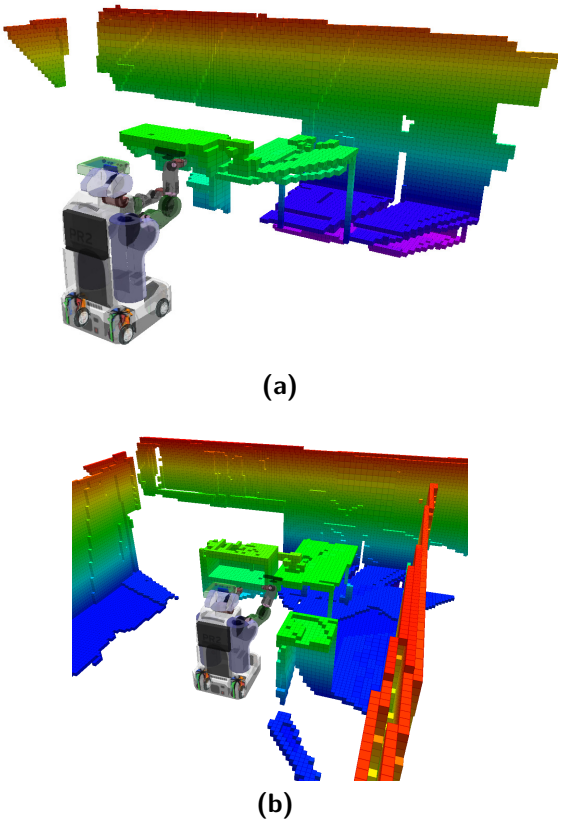
Figure 10-2: The collision OctoMap with a model of the PR2 robot in it. The hand with the Kinect on it is highlighted with a circle.



(a)

(b)

Figure 10-3: At the start (a) of the simulation the space round the PR2 is initialized as occupied. The Kinect mounted at the gripper of the PR2 starts registering obstacles and free space. Occupied space which turns out to be free is cleared again (the small hole in the 'wall'). After moving the Kinect arround more space is cleared (b).The octomap in these pictures has a 0.05 meter resolution.



(a)

(b)

Figure 10-4: The Next Best View map at the beginning when a few movements have been made (a). And after several movements (b) when a larger portion of the room has been seen.

partly empty, although the Kinect has already seen a part of the scene. In the figure below it (figure 10-3b) can be seen that after a few movements the 'wall' around the robot has be cleared for a large part. In the figure next to it (figure 10-4b) can be seen that NBV map contains now a larger part of the environment of the robot.

From this test it was found that the initial NBV map also needed to be initialized. Not with occupied space like in the collison map but the free space in the 'safe zone' should be initialized to free. Otherwise the area round the camera will be unknown. Then the NBV algorithm would not work efficiently at the start. As the first voxel in front of the camera would be unknown the raytrace that used to determine new viewpoints would stop there. If the direct area round the robot is initialized as free in the NBV OctoMap then it is possible to start raytraces from more locations. This is important for the NBV algorithm. Because if the area is not initialized as free it means that new viewpoints can only be generated in the area that is cleared from the first viewpoint which limits the set of usable points very much. This will make the scan take more time as in the beginning of the scan more viewpoints which have less information gain have to be used to clear the direct area.

10-4 Conclusion

It was not possible to simulate the UR5 therefore the PR2 robot was used. As switching back from the PR2 to the real UR5 would take some time it was chosen not test the whole design in simulation.

From part that was simulated it was found that the proposed design with one OctoMap is not possible due to limitations in the Arm Navigation Package. Two OctoMap servers need to be used, one for collision checking and one for determining the next best view. It was also found that is better initialize a small area around the robot as free in the next best view map. As the set of possible viewpoints in the beginning is then much larger the scan will take less time.

Chapter 11

Workspace mapping with the UR5

In the second part extrinsic calibration methods have been studied and tested to get the location of a Kinect camera on the robot. With this location known the camera will now be used to scan the surroundings of the robot for obstacles in a safe manner. In the previous chapters it was found that two OctoMaps should be used. In this chapter the NBV algorithm will be implemented and tested on a UR5 robot arm.

11-1 Hardware

The Universal Robot arm was used with a Kinect mounted on the TCP above the gripper. The calibration result from the previous chapter was used to get the transformation from the end effector to the Kinect. A laptop was used to control the robot and process the Kinect data.

11-2 Software

In figure 11-1 an overview of the system is given. The blocks indicate the ROS nodes (separate programs) and their interaction.

We will discuss the high level working of the system here first, in the next parts a more detailed description will be given. The 'Robot' block is the biggest subsystem in this diagram. This block represents a real robot or a simulated one. From this the robot pose and the Kinect pose is generated using values from the encoders . The Kinect scans the world and generates a 3D image from it (a pointcloud). This data is then streamed to two OctoMap servers: The Collision OctoMap server and the NBV OctoMap server. Difference between these servers is that the Collision Map server uses a map which contains a virtual wall round the robot while the NBV is started empty. The flow of 3D images to the servers is controlled by the 3D data switch. The Map from the NBV OctoMap server is used by the Information Gain node to determine where to move to. The Collision Map OctoMap server is used to check

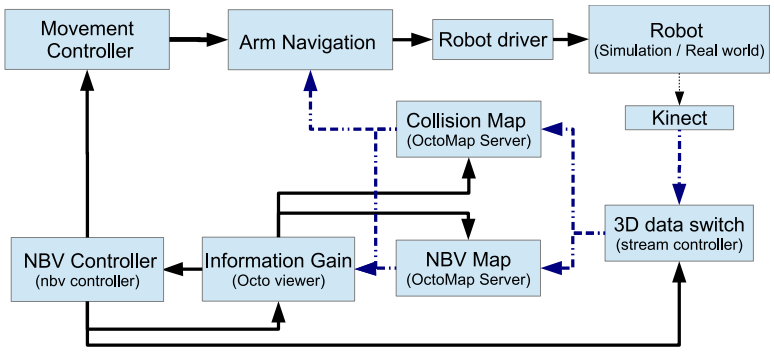


Figure 11-1: Block diagram of the Next Best View system implemented on the UR5 robot arm. The lines from the Kinect to Information gain and Arm navigation nodes represent a data stream containing a 3D map of the environment. The thick solid lines represent service calls. The self filter has been replaced with a 'switch' which switches the data stream on and off.

for collisions both by the Arm Navigation Node and the Information Gain node. The NBV Controller node is the central node which controls all the nodes and generates points for the robot to move to. Movement commands are processed by the Movement Controller, the Arm navigation and the Robot driver.

11-2-1 Performance

The raytracing operations in the OctoMap server nodes use a lot of processing power. The planning of the trajectories for the robot is also a very processing power demanding task. During tests it was found that the processor of the laptop (a bit older dual core Intel Core2Duo processor) had difficulties keeping up with the processing of incoming data. Because of this the OctoMap servers sometimes missed data, areas that were scanned by the Kinect were not cleared in the map. Requests for data from the map servers also took several minutes. Therefore some adjustments to the programs had to be made which will be explained later.

A more modern quad core processor with hyper-threading (for instance a Intel Core i7) would increase the performance of the system probably a lot, as then each node can then run in a separate thread. The programs could also be be optimized more, in the future work section in chapter 12 possible solutions are given.

11-2-2 Kinect

The Microsoft Kinect is used to get 3D positions of the obstacles. The Kinect driver is set to drop 8 of the 10 frames. This done to lower the rate at which the 3D images are sent to the OctoMap servers. This a trade off between data acquisition and performance. A higher rate would mean that a point is seen multiple times which increases the certainty of a measurement. But it also means that the OctoMap server can have trouble keeping up and starts dropping incoming data. The robot is not moving very fast so it possible to set the rate lower.

11-2-3 3D data switch

This is a simple node which receives data from the Kinect and publishes the data again on another topic. With service call other nodes can stop the publication on the second topic, switching the stream 'off'. This again done for performance reasons. During testing it was found that the time necessary to send a map from the OctoMap servers to the Information gain node decreased significantly when the OctoMap servers were not receiving any data at this moment. Therefore this node stops the 3D data to the OctoMaps when a map update request is sent.

The switching node does create extra overhead, therefore other options to disable the stream were also investigated. The Kinect driver has no option of disabling it with software. It is possible to use a bug in the driver by enabling hardware registration. This stops the stream but increases the CPU usage of the Kinect driver. The driver for the Asus Xtion does have a option to temporary disable it.

11-2-4 Collision Map and NBV map

The 3D images are added to both the Collision and the NBV map. For the UR5 setup a collision map which is cylindrically shaped is used. It has a free zone of 0.5 meters around the robot. The occupied zone is from 0.5 to 1.1 meters, which is maximum of the distance the robot can reach. The height is 1.15 metres high, which is above the maximum height the robot can reach at the beginning of the occupied space. See figure 11-2 for the collision map. The NBV map only has the free zone, see figure 11-3.

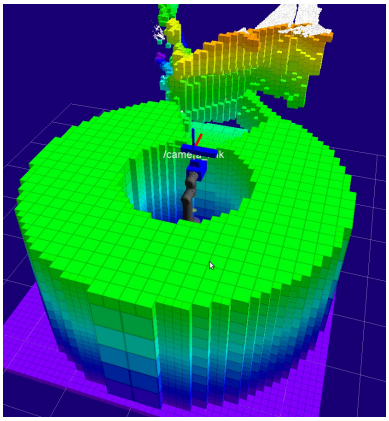


Figure 11-2: The virtual wall in the Collision OctoMap at the start of a scan. The robot is in the center of the cylinder. It can be seen that the Kinect has already scanned the wall behind the robot and some voxels of the virtual wall have been cleared already.

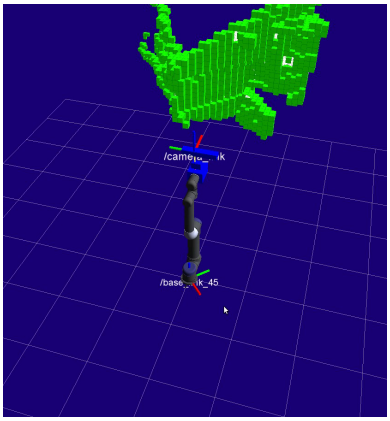


Figure 11-3: The NBV OctoMap with the robot in it. This Map is initialized empty. It can be seen that the same area as in the previous image has been added to the map.

Both maps have a resolution of 5 centimetres. This means that the smallest feature in the map is a 5x5x5 centimetre voxel. Lower resolutions are possible but this decreases the performance.

For collision checking 5 centimetres is enough as the robot should stay away even further from obstacles. The resolution should also be matched to the precision and the accuracy of the used sensor and the actuator it is fixed on. If we look at the accuracy of the Kinect as discussed in paragraph 9-3-1 it can be seen that the accuracy in workspace of the robot is below this resolution.

11-2-5 Information gain

The information gain node gets the collision map and NBV map through a service call from the OctoMap servers. The NBV map is used to test how much unknown space can be seen from a certain viewpoint. This is done by ray casting in the 3D OctoMap, from the generated viewpoint 'rays' are sent to an endpoint. This is done for multiple rays so that an 'image' is generated from that viewpoint. This image gives an indication how much unknown cells can be seen from the generated viewpoint. In figure 11-4 an example of the image that is made is given.

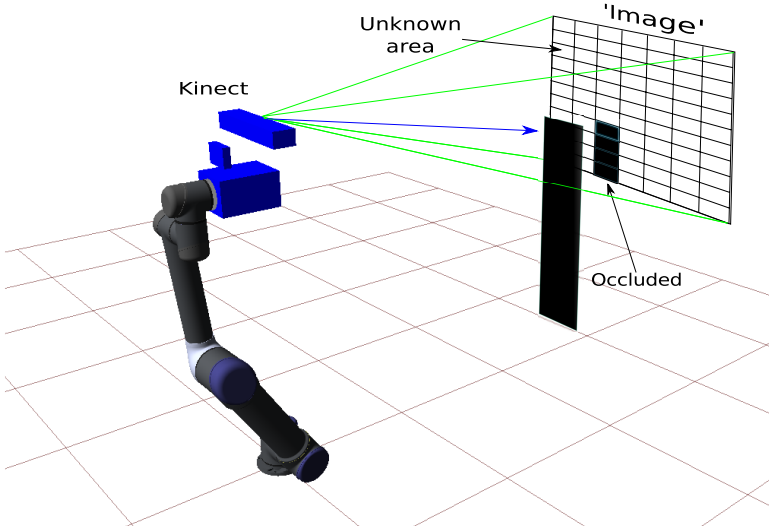


Figure 11-4: The grid on the right represents the 'image' that is made. To the center of each cell in the grid a ray is traced. The blue line represents such a ray. This ray will stop at the black obstacle and not reach the 'image'. The occluded area means that less unknown area is seen from this viewpoint. Another viewpoint in which parts of the image are not occluded will have more unknown area which can be seen and added to the map. The distance in this figure is not in scale with the robot.

This 'image' represents the area that the camera would see if the view is not blocked. The size of the 'image' is determined by the field of view of the Kinect(57 degrees horizontal and 43 degrees vertical) and its maximum range (5 metres). This gives a square of 5.4296 (width) by 3.9391 (height) meters at 5 meters from the camera (the maximum range of the Kinect).

The rays are casted from the camera to this area. The number of rays needed to check the area depend on the resolution. With a resolution of 5 centimetres for the OctoMap voxels this means that 108x78 rays need to be traced. The rays 'fly' from the camera in the direction of the image until they hit an obstacle or unknown space. The number of rays to the image that hit unknown space are counted. If occupied cells are hit it means this part of the area has

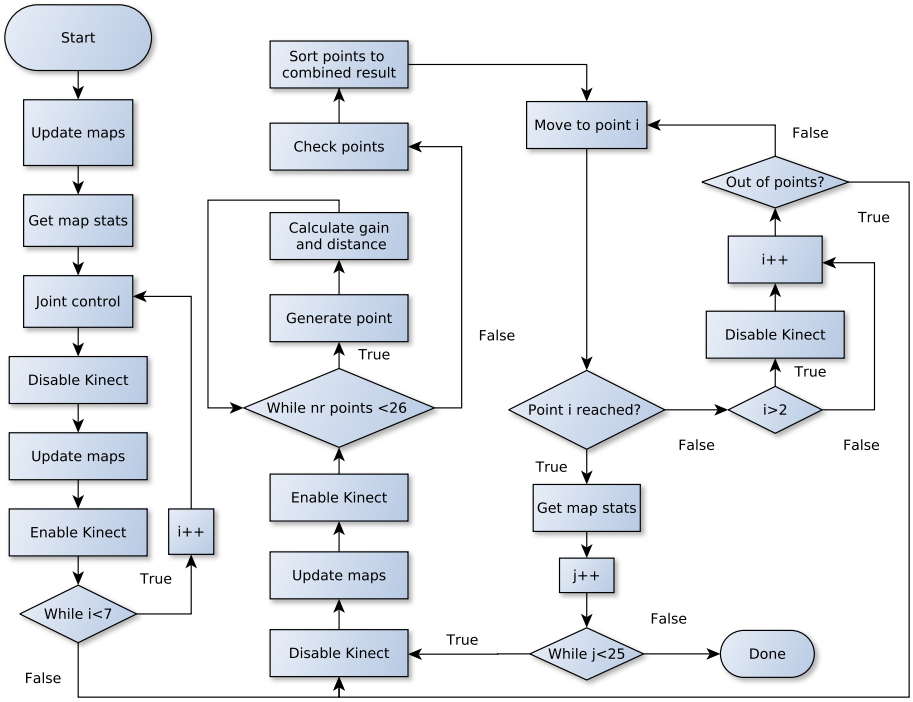


Figure 11-5: Program flow of the mapping system. The left column is the joint control part, the right the NBV part.

already been seen at least once. The more unknown cells in the 'image' the more interesting this viewpoint is.

There is also a weighted variant of this function implemented in this node which can subtract the distance to the number of unknown cells. In this way it possible to favour unknown cells that are close to the robot over unknown cells that are further away.

Another function this node implements is counting the states of all voxels and generate statistic's about the map. The node has also functions to check the status of a voxel at a certain coordinate and a function to save the map.

11-2-6 NBV controller

The Next Best View (NBV) controller is the node that controls the whole system, it controls the robot and can also switch of the data stream from the Kinect. In figure 11-5 the flow of the program is given.

At the start first the maps are updated, this means that the information node asks new maps from the OctoMap servers. With the maps loaded in memory statistics from the voxel states are generated (number of occupied, free and unknown voxels). As the collision map contains a virtual wall the robot cannot easily move yet. This gave problems with the path planner, very odd trajectories were generated because of the limited space. Therefore for the initial scan joint control is used instead of Cartesian control. With joint control a first scan is made

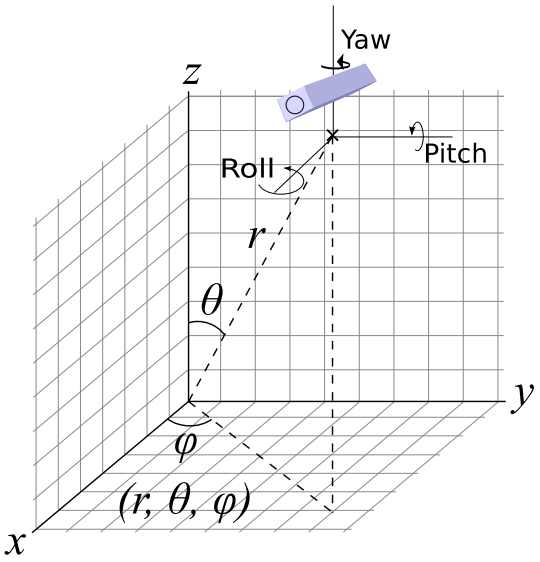


Figure 11-6: Representation of the coordinate systems used for the random point generation. The box represents the camera which is above the the end effector. By taking a random value for φ and θ an end effector position is generated. Using the Roll, Pitch and Yaw angles an orientation is generated separate.

of the direct environment of the robot. This is done by rotating the base link and the y-axis of the end effector. After each step statistics are generated.

After this part the Next Best View part is started. As the direct environment of the robot is now cleared there is more room to move and Cartesian control can now be used. With the NBV algorithm unknown area's that are missed from the initial scan are scanned. This done by moving to different viewpoints from which unknown area's can be scanned.

For each step the map is first updated by commanding the Information gain node to request a new map from the OctoMap server. Before the map update the Kinect is disabled to increase the map loading speed. Then the maps are requested, after which the Kinect is enabled again.

Next 20 random end effector poses (position and orientation) are generated. This is done with spherical coordinates, see figure 11-6. For each point the information gain is calculated by the information gain node. The distance from the current position to the generated point is also calculated. With weighting factors the gain and distance are subtracted from each other. Viewpoints with a lot of unknown voxels in the 'image' get a high score. While the distance is subtracted from this score, penalizing viewpoints that are far away. This gives the combined value value which is stored. Each point is then checked if it is not in an occupied or unknown area. If this is the case the point is removed.

Next the points are sorted to the total combined value. The motion planner is then ordered to plan to the point with the highest score (the combination of the largest number of unknown cells and the lowest travel distance). If the motion planning fails because for instance the trajectory is in collision or not reachable, the second point is taken. This is done until a point is reached. If the system is out of points before a point is reached a new set of points is generated.

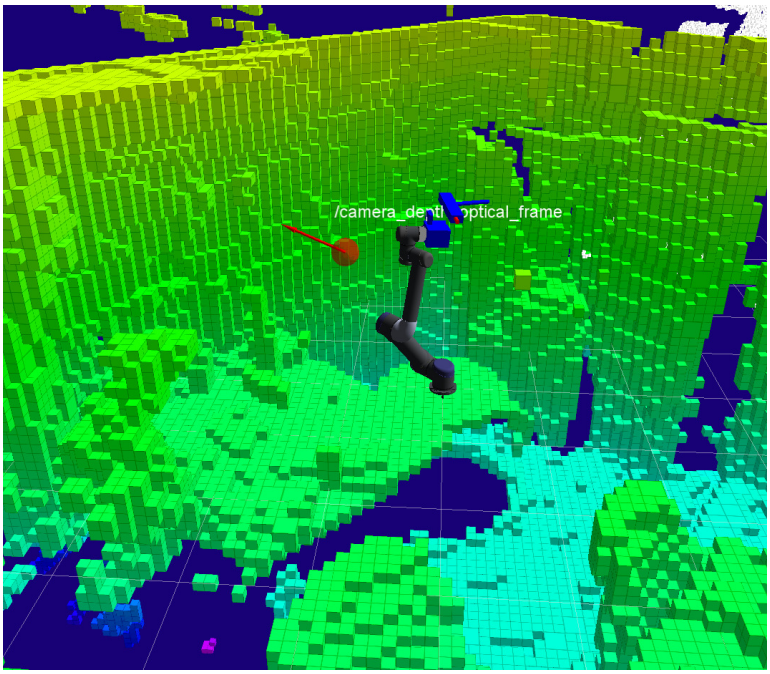


Figure 11-7: Screenshot from the visualization tool RViZ of the robot arm with occupied voxels in green around the robot. The red sphere with the arrow visualizes a new scan point and the direction of the camera. The table where the robot is standing on is clearly visible as well as the table next to it.

11-2-7 Movement controller and Arm Navigation

During tests it was found that the planning was aborted often because the allowed planning time was exceeded. This was solved by increasing the allowed time and disable the Kinect after three tries. Disabling the Kinect gives the planner more processing power thus making it possible that more trajectories can be tested. But if a satisfactory trajectory is found it is executed immediately so there is no time to enable the Kinect again. Disabling has thus the disadvantage that no scans can be made during movement.

11-3 Results

The scanning system works but due to the computational load it takes some time for a scan to complete. This a trade-off between operating speed and data loss due to the processing speed. The system can scan faster but then area's are not cleared although they have been scanned with the Kinect. This probably because of that the OctoMap servers can then not cope with the amount of incoming data and are dropping data.

If the speed is kept at a moderate level the system can keep up. In figure 11-8 the scan statistics are plotted. It can be seen that in the first seven scan points a lot of the surroundings are seen. This is during the first scans which use joint control. After this the NBV algorithm scans the places that could not be seen using joint control. It can be seen that after 20 scan points the map doesn't change much any more. In figure 11-11 to figure 11-13 the NBV map

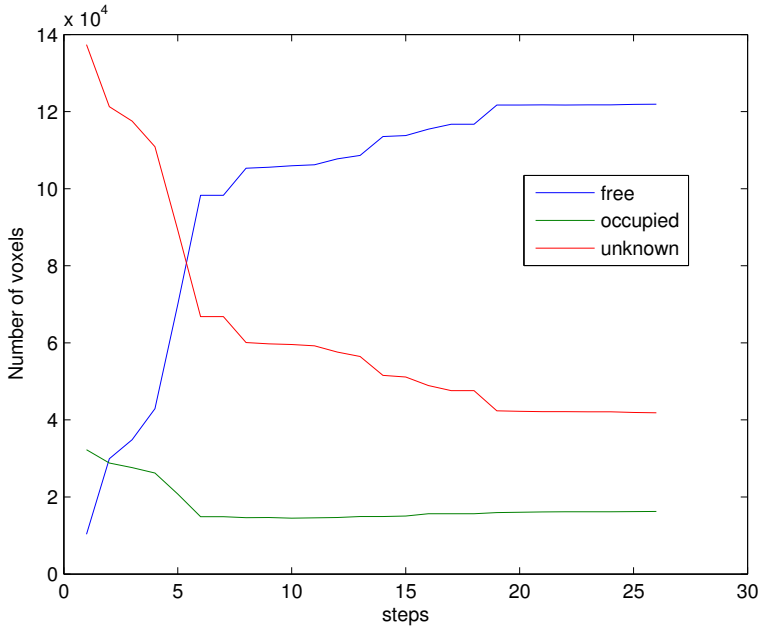


Figure 11-8: State of the voxels of the map.

is given. First it is empty after which obstacles are added. Below it is the same map but now inverted, unknown space is displayed as occupied. In figure 11-14 it can be seen that most space is still unknown. In figure 11-15 it can be seen that more space is known now. There is still a lot of unknown space visible but this behind occupied space (the green voxels) so these are spaces which never can be seen. In figure 11-17 the occupied collision map is given. It can be seen that the cylinder shaps is completly cleared in figure 11-18.

During tests it can be seen that the NBV algorithm really has some added value to the scan process. For instance in figure 11-9 and figure 11-10 it can be seen that the NBV algorithm makes the robot look behind obstacles. The NBV algorithm also made the robot look through a cut-out of a box which simulated a machine opening. If only a scan is made by rotating the robot this opening isn't properly scanned. With the NBV algorithm it is scanned better. Although as the NBV algorithm uses random poses there is no guarantee it will always do this.

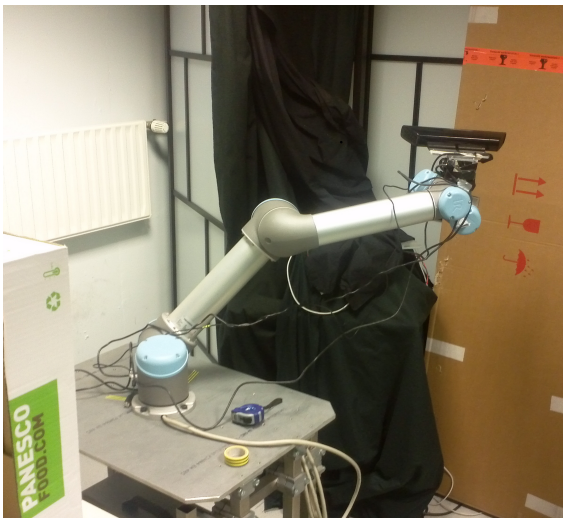


Figure 11-9: Example of the NBV controller making the robot look to the space behind the movable screens.

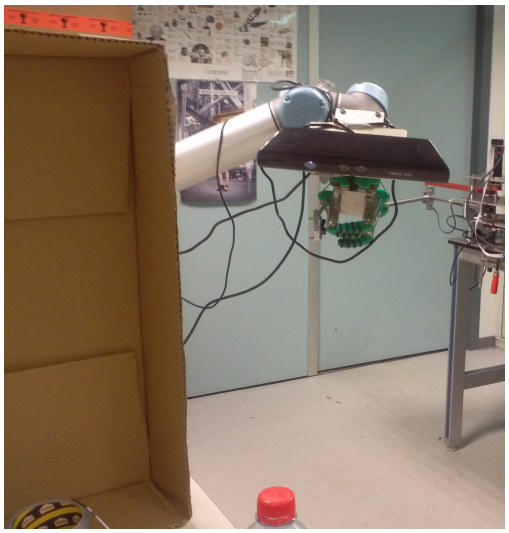


Figure 11-10: Example of the NBV controller making the robot look to the space behind the box.

Figure 11-11: The NBV map after 3 scans.

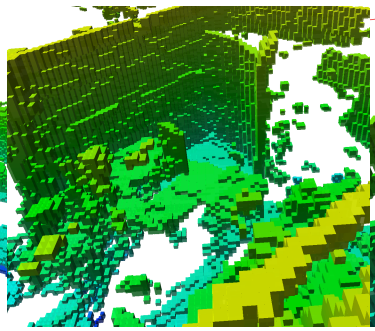


Figure 11-12: The NBV map after 23 scans.

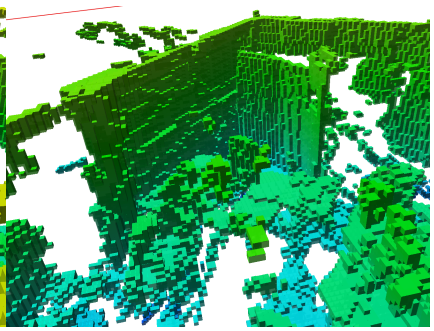


Figure 11-13: The NBV map after 38 scans.

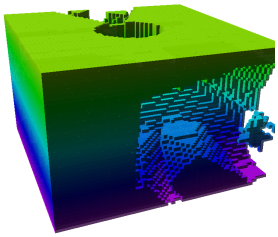


Figure 11-14: The Inverted NBV map, unknown space is now displayed as occupied.

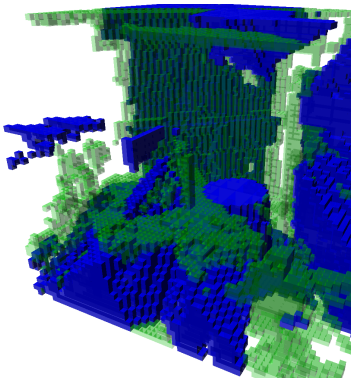


Figure 11-15: The inverted map after 23 scans. The blue voxels are unknown space and the green voxels are occupied.

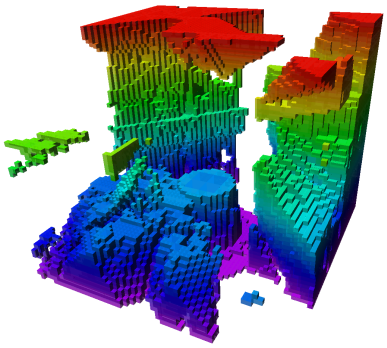


Figure 11-16: The inverted map after 38 scans.

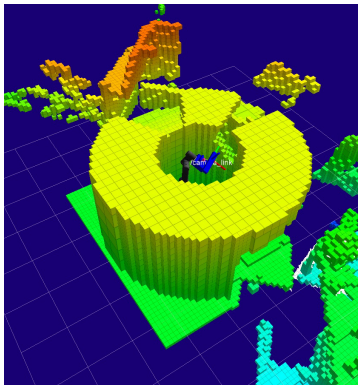


Figure 11-17: The occupied map after 3 scans.

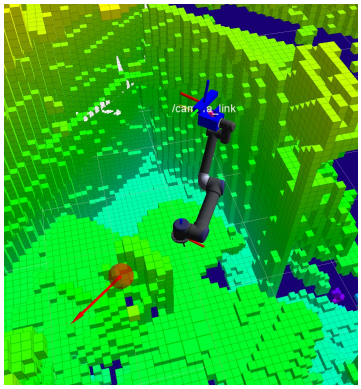


Figure 11-18: The occupied map after 23 scans.

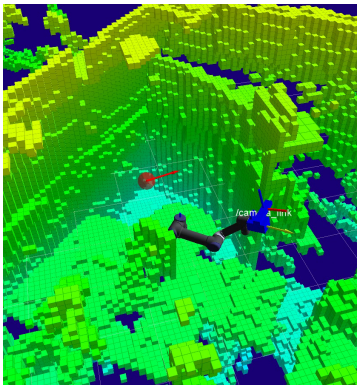


Figure 11-19: The occupied map after 38 scans.

Chapter 12

Conclusion and future work

12-1 Conclusion

The goal of this thesis, as defined in the introduction, was to contribute to fast reconfigurability of industrial robot arms by automating two parts of the installation process:

1. Develop a system that can help with fast (extrinsic) calibration of a camera on a robot arm.
    - Accuracy and precision
      The combined error should be below a centimetre so that the camera can be used to grasp something using the camera with a Laquey gripper.
2. Use the camera to automatically map the surroundings of the robot for obstacles.
    - Mapping the surroundings (workspace) of the robot arm should be done in a safe way, during scanning the robot should not collide with an unknown object.

For the first goal, the extrinsic calibration, it can be concluded that this works good. It was difficult to validate the result by measuring the camera pose directly. Therefore the position of the checkerboard was measured by placing the Tool Centre Point (TCP) at the checkerboard. The error was then calculated by determining the camera location in two ways. The first way is to use the robots encoders and the calculated hand-eye transform. The second is by using the estimated camera position relative to the checkerboard. The error of the calibration result for the Logitech webcam is in the order of a centimetre for the x axis, for the other axes it is less. The rotation error is approximately 4 degrees for the x axis and also less for the other axes. For the Kinect the calibration result is better, the largest translation error is approximately half a centimetre. The rotation error is also lower, the largest error is in the order of a degree and the errors for the other axes are lower.

For the second goal, the workspace mapping system, it can be concluded that this also works good. The Next Best View (NBV) algorithm looks with the camera behind objects and sees

unknown space that is otherwise not seen. The robot cannot plan movements in unknown space and will thus not collide with objects it has not seen yet. Doing an initial scan by rotating the joints first solved the problem of the complex trajectories which were generated by a lack of movement space.

12-2 Discussion

Interesting to note is that in simulation the camera pose estimation result was proportional with the resolution of the camera. In practice this was however not the case, changing the resolution did not have much effect on the error. This could be due to the distortion and the optical resolution of the used webcam. In simulation the image improves when using a higher camera resolution, giving a better pose estimate. But in the practice the lens could be limiting the resolution increase on the sensor of the camera. Calibration helps compensate the distortion but probably the resolution of the image is still affected.

12-3 Future work

12-3-1 Extrinsic calibration

The system should be tested on different locations of the robot. Further away from the TCP seems to improve the result.

Augmented Reality (AR) markers were discarded as the results in practice were not good enough. But this was probably due to a conversion error somewhere. AR Markers could still be interesting. With AR markers it is possible to cover large parts of the surroundings of the robot with different markers. If this is done it is possible to use a subset of the markers giving a larger set of possible viewpoints. With a single checkerboard the number of possible viewpoints is more limited. Using more viewpoints could improve the of the calibration. Disadvantage is that the locations of the markers relative to each other does need to be known very precise with this.

It would also be interesting to refine the results of the output of the Tsai-Lenz algorithm with an iterative algorithm. Using an iterative algorithm directly could be computationally taxing but if a good estimate is already known then this could be less of a problem.

12-3-2 Workspace mapping

The current implementation is adapted to deal with the slow computer used in the research. If a more recent quad core computer would be used the system could scan a lot faster. The current implementation proves that the method works, but it could be implemented a lot more a efficient. The current setup consists of two OctoMap servers which could be integrated in one custom made OctoMap server. If a custom OctoMap server is implemented the NBV algorithm can be implemented inside this system. This will increase the speed a lot as shared memory can then be used instead of slow socket communication. The operating speed will also

increase as there is then only one computationally intensive raytracing operation necessary when adding new data to the map.

With a custom implementation it will also be possible to implement a maximum range method. With this the space can also be cleared if nothing is hit, in appendix B-2 methods for this are given.

It is also possible to implement labelled voxels which can store additional information like timestamps or color information. With timestamps for instance, occupied area's which have not been seen for a long time can be set to unknown again. Another improvement would be to use the Point Cloud Library (PCL) to filter the data and segment objects with this (see appendix B-2).

Appendix A

Formula derivations

A-1 Usable distance

Goal is to calculate the r in figure A-1 which represents the distance at which the error (err) at the object is below 0.0058 metres.

Using Pythagoras:

$$err^2 = a^2 + b^2 \tag{A-1}$$

The error depends on the calibration error per axis:

$$\begin{aligned} a =& tan(\varphi)r \\ b =& tan(\theta)r \end{aligned} \tag{A-2}$$

The usable distance r is then:

$$r^2 = \frac{\text{err}^2}{tan(\varphi)^2 + tan(\theta)^2} \tag{A-3}$$

$$r = \sqrt{\frac{0.0058^2}{tan(-6.966/1000)^2 + tan(4.685/1000)^2}} = 0.6909[m] \tag{A-4}$$

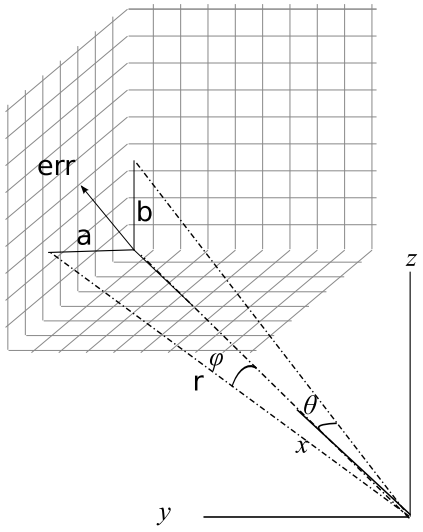


Figure A-1: Coordinate system used to calculate the grasp error. On the top left the object is given and on the right the camera coordinate system.

A-2 OctoMap formula

In section 9-3-3 formula 9-1 (from [5]) was rewritten. In this appendix the steps required to get from A-5 to A-12 will be given.

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{A-5}$$

If the multiply the Right Hand Side (RHS) by the denominator, we can express $P(n|z_{1:t})$ as

$$P(n|z_{1:t}) = \frac{P(n|z_t)P(n|z_{1:t-1})(1 - P(n))}{P(n|z_t)P(n|z_{1:t-1})(1 - P(n)) + (1 - P(n|z_t))(1 - P(n|z_{1:t-1}))P(n)} \tag{A-6}$$

Multiplying by minus one and adding one at both sides:

$$1 - P(n|z_{1:t}) = \frac{(1 - P(n|z_t))(1 - P(n|z_{1:t-1}))P(n)}{P(n|z_t)P(n|z_{1:t-1})(1 - P(n)) + (1 - P(n|z_t))(1 - P(n|z_{1:t-1}))P(n)} \tag{A-7}$$

Subtracting and taking the inverse of both sides, we get:

$$\frac{P(n|z_{1:t})}{1 - P(n|z_{1:t})} = \frac{P(n|z_t)P(n|z_{1:t-1})(1 - P(n))}{(1 - P(n|z_t))(1 - P(n|z_{1:t-1}))P(n)} \tag{A-8}$$

Taking the logarithm of both sides leads to:

$$\log\left[\frac{P(n|z_{1:t})}{1 - P(n|z_{1:t})}\right] = \log\left[\frac{P(n|z_{1:t-1})}{1 - P(n|z_{1:t-1})}\right] + \log\left[\frac{P(n|z_t)}{1 - P(n|z_t)}\right] - \log\left[\frac{P(n)}{1 - P(n)}\right] \tag{A-9}$$

In the paper[5] the following definition was made:

$$L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{A-10}$$

Using this notation $L(n)$ we have

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) - L(n) \tag{A-11}$$

A uniform prior probability is assumed where $P(n) = 0.5$, leading to $L(n) = 0$, giving the compact update equation:

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{A-12}$$

Appendix B

Custom OctoMap server

In this study two OctoMap servers have been used. But it is also possible to create an own implementation of the OctoMap server as there are templates of the OctoMap server available. If a custom version is implemented it is possible to add other improvements.

B-1 Improvements using the Point Cloud Library (PCL) and labelled octrees

- PCL methods can be used to segment data before adding it to the map. Surfaces can be first fitted for instance. This could improve the quality of the map.
- Use PCL VoxelGrid filtering[1] to filter and limit data to usable range and downsample it to the leaf size
- Use semantic mapping, label objects. So that the context can be used. If a machine or conveyor is moved the whole object could be moved in a interface.
- With the labelled octree type[2] one OctoMap server could be used. Using the label extra information can be stored. It also possible to use one map in the server but send different maps to other nodes using one basis map. The arm navigation could for instance receive a map where unknown space is also set to occupied. Other Octree types are also possible: color information, timestamps or storing with which sensor a voxel has been seen is also possible.

[1] http://wiki.ros.org/pcl_ros/Tutorials/VoxelGridfiltering

[2] http://code.google.com/p/alufr-ros-pkg/source/browse/branches/octomap_mapping-experimental/octomap/include/octomap/OcTreeNodeLabeled.h

B-2 Clearing space without obstacles

The point cloud created by the depth sensor describes occupied space by giving the coordinates to it. Obstacles far away are used to clear the empty space between the camera and the obstacle. However if there are no objects present within the maximum range of the sensor nothing is returned. A solution can be to scan downwards so that the floor is always seen.

Another, better solution would be to modify the OctoMap server. This could be done in two ways[3]

- Using 'fake points' beyond the maximum range of the sensor. This will fill the rays with free space up to the maximum range.
- Using a custom update in the OctoMap which sets all the elements along the ray to free. This is similar to the ground plane segmentation system (if the floor should not added to the map). With this rays to the floor are only used for clearing space not setting occupied space.

[3] https://groups.google.com/#!topic/octomap/00YA_xBML_Q

Appendix C

Simulation documentation

C-1 Simulation block diagram

The detailed block diagram of the structure of the simulation can be seen in figure C-1. The corresponding source files are listed next to the blocks. If the package is outside the ar_joint_controller package than then package name is listed in front.

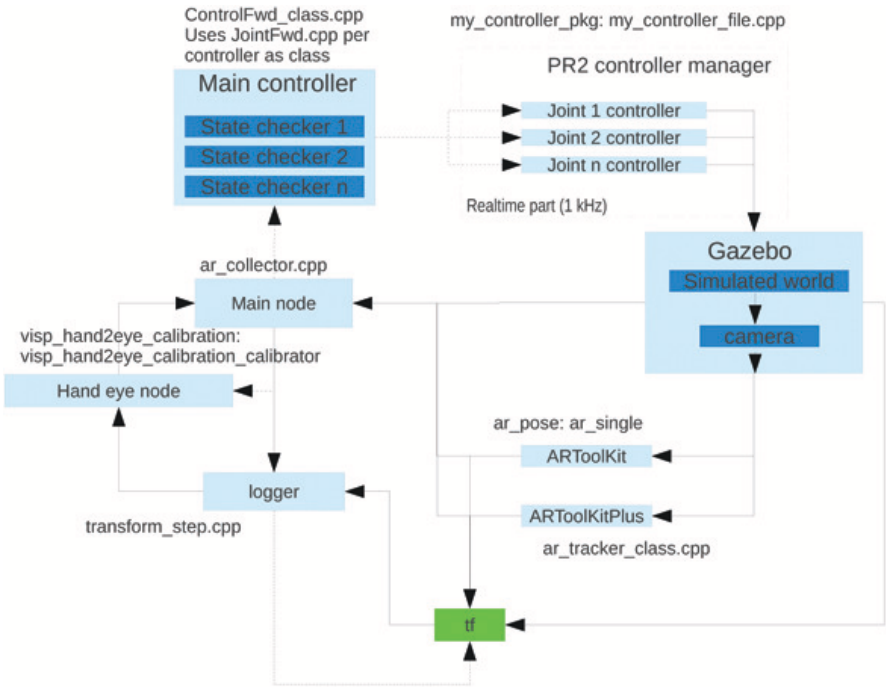


Figure C-1: The block diagram with the file names of the source files of the nodes.

C-2 Custom material

A checkerboard pattern was not available in the Gazebo Simulator. Therefore a custom material script was created.

```
<visual name='arm_base_checkerboard_bch0'>
  <origin pose='-0.0 -0.00 0.85 0.0 0 0'/>
        <geometry>
    <!-- x y z 0.5 0.12 0.5 -->
          <box size='0.21565 0.22 0.27915'/>

        </geometry>
        <material script='Gazebo/checkerboard'/>
      </visual>
```

In

`/simulator_gazebo/gazebo/gazebo/share/gazebo-1.0.2/Media/materials/scripts`

Checkerboard material:

```
material Gazebo/checkerboard
{
  technique
  {
    pass
    {
      ambient 1.0 1.0 1.0 1.0
      diffuse 1.0 1.0 1.0 1.0
      specular 0.2 0.2 0.2 1.0 12.5

      texture_unit
      {
        texture checkerboard_freiburg2.png
        filtering trilinear
      }
    }
  }
}
```

C-3 Self Filter

Parameter	Value
min_sensor_dist	0.01
self_see_padd	0.02
elf_see_scale	1

Table C-1: Configuration of the self occlusion filter.

Appendix D

Robot specifications and camera calibration data

D-1 Universal Robots UR5 Specifications

Specification	Value	Unit
Reach	850	mm
Movement	+/- 360	degrees
Repeatability	+/- 0.1	mm
DOF	6	
Payload	5	kg
Speed	Joint max. 180 degrees/s tool approx 1 m/s	

Table D-1: Specifications for the UR5 robot arm. Source: Universal Robots specification sheet.

D-2 Simulated Camera

Parameter	Value	Description
x size	640	pixels in horizontal direction
y size	480	pixels in vertical direction
cc_x	320.5	principal point in pixels x
cc_y	240.5	principal point in pixels y
fc_x	589.36645488527788	focal length in pixels x
fc_y	589.3664548852778	focal length in pixels y
kc_1 - kc_4	0.0	distortion coefficients
Iter	10	number of iterations for distortion compensation

Table D-2: ARToolKitPlus parameters for the Kinect on the PR2.

Parameter	Value	Description
x size	800	pixels in horizontal direction
y size	600	pixels in vertical direction
cc_x	400.5	principal point in pixels x
cc_y	300.5	principal point in pixels y
fc_x	965.68563990468	focal length in pixels x
fc_y	965.68563990468	focal length in pixels y
kc_1 - kc_4	0.0	distortion coefficients
Iter	10	number of iterations for distortion compensation

Table D-3: ARToolKitPlus parameters used in the simulation.

D-3 Logitech Pro 9000

Parameter	Value	Description
Hardware id	046d:0809	USB hardware id
x size	640	pixels in horizontal direction
y size	480	pixels in vertical direction
cc_x	308.629039	principal point in pixels x
cc_y	226.921375	principal point in pixels y
fc_x	534.782032	focal length in pixels x
fc_y	534.103333	focal length in pixels y
kc_1	0.045860	distortion coefficient
kc_2	-0.130100	distortion coefficient
kc_3	0.002500	distortion coefficient
kc_4	-0.000597	distortion coefficient
kc_5	0.000000	distortion coefficient

Table D-4: Calibration data for the Logitech Pro 9000 for a 640 by 480 pixels resolution.

Parameter	Value	Description
Hardware id	046d:0809	USB hardware id
x size	800	pixels in horizontal direction
y size	600	pixels in vertical direction
cc_x	381.662765	principal point in pixels x
cc_y	285.042717	principal point in pixels y
fc_x	644.891113	focal length in pixels x
fc_y	650.425720	focal length in pixels y
kc_1	0.022619	distortion coefficient
kc_2	-0.093668	distortion coefficient
kc_3	0.008655	distortion coefficient
kc_4	-0.002706	distortion coefficient
kc_5	0.000000	distortion coefficient

Table D-5: Calibration data for the Logitech Pro 9000 for a 800 by 600 pixels resolution.

Parameter	Value	Description
Hardware id	046d:0809	USB hardware id
x size	1600	pixels in horizontal direction
y size	1200	pixels in vertical direction
cc_x	723.615497	principal point in pixels x
cc_y	649.802808	principal point in pixels y
fc_x	1524.010254	focal length in pixels x
fc_y	1541.401733	focal length in pixels y
kc_1	0.051928	distortion coefficient
kc_2	-0.238112	distortion coefficient
kc_3	0.022684	distortion coefficient
kc_4	-0.010708	distortion coefficient
kc_5	0.000000	distortion coefficient

Table D-6: Calibration data for the Logitech Pro 9000 for a 1600 by 1200 pixels resolution.

D-4 Calibration Kinect

Parameter	Value	Description
Hardware id		USB hardware id
x size	640	pixels in horizontal direction
y size	480	pixels in vertical direction
cc_x	321.610538	principal point in pixels x
cc_y	252.175642	principal point in pixels y
fc_x	537.327026	focal length in pixels x
fc_y	538.345520	focal length in pixels y
kc_1	0.179381	distortion coefficient
kc_2	-0.318182	distortion coefficient
kc_3	-0.002684	distortion coefficient
kc_4	0.002093	distortion coefficient
kc_5	0.000000	distortion coefficient

Table D-7: Calibration data for the Kinect RGB camera for a 640 by 480 pixels resolution.

D-5 Checkerboard pose estimator settings

Parameter	Package		
	Checkerboard_detector2	find_extrinsics	
	Solvepnp()	Solvepnp()	SolvepnpRansac()
Iterations	20	30	30
Threshold	1e-2	0.1	0.1
Search window size	(5,5)	(11, 11)	(11, 11)
Zero zone	None	None	None
Iterations			100
reprojectionError			8.0
minInliersCount			10
useExtrinsicGuess	False	False	False

Table D-8: Configuration settings for the pose estimation algorithms.

Appendix E

Software

The commands necessary to run the various parts of the designed software.

E-1 Error of the pose estimator

```
ur5_real_find_extrinsics_only2.launch
rosrun ar_joint_controller ur_collector
```

E-2 PR2 extrinsic calibration

```
roslaunch ar_joint_controller pr2_extrinsic.launch

rosrun ar_joint_controller ar_tracker
 /usb_cam/image_raw:=/r_gripper_kinect/r_gripper_kinect/rgb/image_raw

rosrun ar_joint_controller move2pose_extrinsic
```

E-3 Validation of the extrinsic calibration

```
roslaunch ur5_real_validation.launch
rosrun ar_joint_controller ur_collector
```

E-4 Workspace Mapping

```
roslaunch ar_joint_controller ur5_real_nbv_cylinder.launch
rosrun ar_joint_controller nbv_controller
```

Bibliography

[1] P. Lamb, "Tutorial 2: Camera and Marker Relationships."

[2] H. Kato and M. Billinghurst, "Marker tracking and HMD calibration for a video-based augmented reality conferencing system," in *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pp. 85–94, IEEE Comput. Soc, 1999.

[3] F. Dornaika and R. Horaud, "Simultaneous robot-world and hand-eye calibration," *Robotics and Automation, IEEE ...*, vol. 14, no. 4, pp. 617–622, 1998.

[4] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," *... practice in 3D ...*, 2010.

[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189–206, Feb. 2013.

[6] F. Ernst, L. Richter, and L. Matthäus, "Non-orthogonal tool/flange and robot/world calibration," *... Journal of Medical ...*, no. January, pp. 407–420, 2012.

[7] Microsoft, "Kinect for Windows Sensor Components and Specifications."

[8] Microsoft, "Coordinate Spaces."

[9] Microsoft, "Natural User Interface (NUI), constants."

[10] R. a. Newcombe and A. J. Davison, "Live dense reconstruction with a single moving camera," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1498–1505, June 2010.

[11] M. Andersen, T. Jensen, and P. Lisouski, "Kinect depth sensor evaluation for computer vision applications," 2012.

[12] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications.," *Sensors (Basel, Switzerland)*, vol. 12, pp. 1437–54, Jan. 2012.

[13] B. Atcheson, F. Heide, and W. Heidrich, "CALTag: High Precision Fiducial Markers for Camera Calibration.," in *VMV 2010: Vision, Modeling & Visualization*, 2010.

[14] M. Fiala, "Artag, an improved marker system based on artoolkit," 2004.

[15] D. Wagner and D. Schmalstieg, "Artoolkitplus for pose tracking on mobile devices," in *Computer Vision Winter Workshop 2007*, p. 9, 2007.

[16] D. Schmalstieg and A. Fuhrmann, "The studierstube augmented reality project," *Presence: ...*, vol. 11, no. 1, pp. 33–54, 2002.

[17] E. Olson, "AprilTag: A robust and flexible visual fiducial system," *2011 IEEE International Conference on Robotics and Automation*, pp. 3400–3407, May 2011.

[18] S. Garrido-Jurado, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, 2014.

[19] J. Stork, *Camera pose estimation with circular markers.* Thesis, University of Amsterdam (UvA), 2012.

[20] T. D. Kler, "Integration of the ARToolKitPlus optical tracker into the Personal Space Station," *Section: Computer Science, University of Amsterdam, ...*, 2007.

[21] J. Matoušek, *Depth map for augmented reality improvement.* Thesis, Czech Technical University in Prague, 2011.

[22] M. Fiala, "Comparing ARTag and ARToolkit plus fiducial marker systems," *IREE International Worksho on Haptic Audio Visual Environments and their Applications, 2005.*, pp. 147–152, 2005.

[23] R. Andersen, "Systematic test of pose estimation algorithms for use in a specific scene.," *covil.sdu.dk*, vol. 2, no. 1, pp. 1–11.

[24] D. F. Abawi, J. Bienwald, J. W. G.-u. Frankfurt, and R. Dörner, "Accuracy in Optical Tracking with Fiducial Markers : An Accuracy Function for ARToolKit," no. Ismar, pp. 0–1, 2004.

[25] P. Wayne, W. Piekarski, and B. Thomas, "Measuring artoolkit accuracy in long distance tracking experiments," *In 1st Int'l Augmented Reality ...*, no. C, pp. 1–2, 2002.

[26] E. Olson, "EECS568 Mobile Robotics: Methods and Principles," 2011.

[27] D. Brown, "Decentering Distortion of Lenses," *Photometric Engineering*, vol. 32, no. 3, pp. 444–462, 1966.

[28] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE ...*, vol. 1998, no. 11, pp. 1330–1334, 2000.

[29] X. Gao and X. Hou, "Complete solution classification for the perspective-three-point problem," *Pattern Analysis and ...*, vol. 25, pp. 930–943, Aug. 2003.

[30] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *International Journal of Computer Vision*, vol. 81, pp. 155–166, July 2008.

[31] Y. Shiu and S. Ahmad, "Finding the mounting position of a sensor by solving a homogeneous transform equation of the form AX= XB," *Robotics and Automation. Proceedings. ...*, pp. 1666–1671, 1987.

[32] R. Tsai and R. Lenz, "A new technique for fully autonomous and efficient 3D robotics hand/eye calibration," *Robotics and Automation, IEEE ...*, vol. 5, no. 3, 1989.

[33] T. Kiesche and N. Krüger, *Hand-eye calibration with a stereo camera*. PhD thesis, 2006.

[34] D. W. Kim and J. E. Ha, "Hand/Eye Calibration Using 3D-3D Correspondences," *Applied Mechanics and Materials*, vol. 319, pp. 532–535, May 2013.

[35] R. Schmitt, Y. Cai, and P. Jatzkowski, "Estimation of the absolute camera pose for environment recognition of industrial robotics," *Production Engineering*, vol. 7, pp. 91–100, Dec. 2012.

[36] M. Shah, R. D. Eastman, and T. Hong, "An overview of robot-sensor calibration methods for evaluation of perception systems," *Proceedings of the Workshop on Performance Metrics for Intelligent Systems - PerMIS '12*, p. 15, 2012.

[37] H. Zhuang and Z. Roth, "Comments on "Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form AX=XB" [with reply]," 1991.

[38] H. Chen, "A screw motion approach to uniqueness analysis of head-eye geometry," *Computer Vision and Pattern Recognition, 1991. ...*, pp. 145–151, 1991.

[39] C. Wang, "Extrinsic calibration of a vision sensor mounted on a robot," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 2, 1992.

[40] J. C. K. Chou and M. Kamel, "Finding the Position and Orientation of a Sensor on a Robot Manipulator Using Quaternions," *The International Journal of Robotics Research*, vol. 10, pp. 240–254, June 1991.

[41] J. Chou and M. Kamel, "Quaternions Approach to Solve the Kinematic Equation of Rotation AaAx= AxAb of a Sensor Mounted Robotic Manipulator," *IEEE International Conference on Robotics and ...*, no. 2, pp. 656–662, 1988.

[42] R. Horaud and F. Dornaika, "Hand-eye calibration," *The International Journal of Robotics Research*, vol. 3, no. 6769, 1995.

[43] F. Park and B. Martin, "Robot sensor calibration: solving AX=XB on the Euclidean group," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 717–721, 1994.

[44] M. Li and D. Betsis, "Head-eye calibration," *Computer Vision, 1995. Proceedings., Fifth ...*, vol. 2, no. 7, pp. 40–45, 1995.

[45] K. Daniilidis, "Hand-Eye Calibration Using Dual Quaternions," *The International Journal of Robotics Research*, vol. 18, pp. 286–298, Mar. 1999.

[46] N. Andreff, R. Horaud, and B. Espiau, "On-line hand-eye calibration," in *Second International Conference on 3-D Digital Imaging and Modeling*, 1999.

[47] Q. Wei, K. Arbter, and G. Hirzinger, "Active self-calibration of robotic eyes and hand-eye relationships with model identification," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 158–166, 1998.

[48] V. Pradeep, K. Konolige, and E. Berger, "Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach," *Experimental Robotics*, 2014.

[49] H. Z. H. Zhuang, Z. Roth, and R. Sudhakar, "Simultaneous robot/world and tool/flange calibration by solving homogeneous transformation equations of the form AX=YB," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 4, 1994.

[50] M. Shah, "Solving the Robot-World/Hand-Eye Calibration Problem Using the Kronecker Product," *Journal of Mechanisms and Robotics*, vol. 5, p. 041009, July 2013.

[51] R. Hirsh, G. DeSouza, and a.C. Kak, "An iterative approach to the hand-eye and base-world calibration problem," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, pp. 2171–2176, 2001.

[52] A. Li, L. Wang, and D. Wu, "Simultaneous robot-world and hand-eye calibration using dual-quaternions and Kronecker product," *International Journal of the Physical ...*, vol. 5, no. 10, pp. 1530–1536, 2010.

[53] N. Andreff, "Robot Hand-Eye Calibration Using Structure-from-Motion," *The International Journal of Robotics Research*, vol. 20, pp. 228–248, Mar. 2001.

[54] L. Richter, "Robotized Transcranial Magnetic Stimulation," in *Robotized Transcranial Magnetic Stimulation*, ch. 4, New York, NY: Springer New York, 2013.

[55] J. Schmidt, F. Vogt, and H. Niemann, "Calibration-free hand-eye calibration: a structure-from-motion approach," in *Pattern Recognition*, p. 526, 2005.

[56] T. Ruland, T. Pajdla, and L. Kruger, "Globally optimal hand-eye calibration," *Computer Vision and Pattern ...*, no. 2, pp. 1035–1042, 2012.

[57] K. Daniilidis and E. Bayro-Corrochano, "The dual quaternion approach to hand-eye calibration," *Proceedings of 13th International Conference on Pattern Recognition*, pp. 318–322 vol.1, 1996.

[58] C. Connolly, "The determination of next best views," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 432–435, 1985.

[59] K. M. Wurm, D. Hennes, D. Holz, R. B. Rusu, C. Stachniss, K. Konolige, and W. Burgard, "Hierarchies of octrees for efficient 3D mapping," *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4249–4255, Sept. 2011.

[60] L. Freda, G. Oriolo, and F. Vecchioli, "Sensor-based Exploration for general robotic systems," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2157–2164, Sept. 2008.

[61] L. Freda, G. Oriolo, and F. Vecchioli, "An exploration method for general robotic systems equipped with multiple sensors," *Intelligent Robots and Systems, ...*, no. 045359, pp. 5076–5082, 2009.

[62] S. Kohlbrecher, K. Petersen, G. Steinbauer, J. Maurer, P. Lepej, S. Uran, R. Ventura, C. Dornhege, A. Hertle, R. Sheh, and J. Pellenz, "Community-driven development of standard software modules for search and rescue robots," *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–2, Nov. 2012.

[63] C. Potthast and G. S. Sukhatme, "Seeing with your hands: A better way to obtain perception capabilities with a personal robot," *Advanced Robotics and its Social Impacts*, pp. 50–53, Oct. 2011.

[64] C. Dornhege and A. Kleiner, "A frontier-void-based approach for autonomous exploration in 3D," *Advanced Robotics*, vol. 27, pp. 459–468, Apr. 2013.

[65] M. Suppa, *Autonomous robot work cell exploration using multisensory eye-in-hand systems*. PhD thesis, Gottfried Wilhelm Leibniz University of Hannover, 2008.

[66] J. Smisek, M. Jancosek, and T. Pajdla, "3D with Kinect," *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 1154–1160, Nov. 2011.

[67] M. T. Draelos, *The Kinect Up Close: Modifications for Short-Range Depth Imaging*. PhD thesis, University of Colorado at Boulder, 2012.

[68] M. Fallon, "Efficient scene simulation for robust Monte Carlo localization using an RGB-D camera," *Robotics and Automation ...*, 2012.

[69] B. Lau, C. Sprunk, and W. Burgard, "Efficient grid-based spatial representations for robot navigation in dynamic environments," *Robotics and Autonomous Systems*, vol. 61, pp. 1116–1130, Oct. 2013.

Glossary

List of Acronyms

DOF	Degree Of Freedom
AR	Augmented Reality
IR	InfraRed
GPL	GNU General Public License
TCP	Tool Centre Point
NBV	Next Best View
OpenCV	Open Source Computer Vision library
PID	Proportional-Integral-Derivative
PCL	Point Cloud Library
RGB	Red Green Blue
RHS	Right Hand Side
RMS	Root Mean Square
ROS	Robot Operating System
RANSAC	RANdom SAmple Consensus
SME	Small to Medium Enterprises
URDF	Universal Robotic Description Format