UNIVERSITY OF SOUTHERN DENMARK

# Hand-eye calibration
for 3D modelling

Participants	Frederik Hagelskjær, M.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute
	Kent Stark Olsen, M.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute
	Leon Bonde Larsen, M.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute
	Rudi Hansen, M.Sc. student, Robot Systems Engineering The Maersk Mc-Kinney Moller Institute
Primary supervisor	Kjeld Jensen, Ph.D. student, Field Robotics The Maersk Mc-Kinney Moller Institute
Secondary supervisor	Lars-Peter Ellekilde, Associate professor The Maersk Mc-Kinney Moller Institute
ECTS	10
Period	Spring 2014

Faculty of Engineering
University of Southern Denmark
Niels Bohrs Allé 1
5230 Odense M
Denmark

www.sdu.dk/tek
+45 6550 7303
tek@tek.sdu.dk

Abstract

Preface

Preface my ass...

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Electronic representations of three-dimensional objects are of great importance to many applications. Often the 3D models are hand-build which can be a tedious and time consuming task thus making automatic modelling desirable. In robotics, models of the environment are used as a priori knowledge for collision checking. The problem with hand-build models is that they limit the application of robots to known environments.

One method for recording 3D models is to mount a stereo camera on the end effector of a robot arm (eye-in-hand) and let the robot move the camera to the views needed to generate the model. This way the robot is able to assess unknown objects and update the work cell model accordingly. The precision of the model is not only dependent on the quality of robot arm and camera, but also on the calibration of the system. The camera can be calibrated using a marker plate in different poses thus calculating intrinsic parameters and camera disparity [22]. The robot model can be calibrated in a process called hand-eye calibration solving for the unknown spatial relationships in the kinematic chain.

Another important part of 3D modelling is the reconstruction problem, reasoning about either surface, volume or both. Depending on the application there are very different demands for the final model. If it is used for collision detection a coarse model is more efficient, while a model for grasping needs to be sufficiently detailed to infer grasping points. Modelling is a multi-step process where the gathered data is first transformed, filtered and combined to form a point cloud representing the object. Next an indicator function is approximated to best fit the normals of the inferred solid. Finally a mesh is generated from the indicator function. Using wavelets to approximate the indicator function provides a localised, multi-resolution representation and thus addresses the trade-off between faster coarse models and slower fine models.

In this work an eye-in-hand system for wavelet based surface reconstruction is developed and the effect of hand-eye calibration is investigated. It is hypothesised that a calibrated system can generate significantly better models than the same system without calibration. The calibration is evaluated in a manipulative study, where a known object is modelled before and after calibrating the system. The reconstructed models are visually as well as quantitatively compared to the true 3D model of the object.

The novelties in this report are hand-eye calibration based on point cloud data, an implementation of the wavelet based surface reconstruction proposed in [**Someone wielding the wavelet hammer...**] and implementing the entire system in Robot Operating System(ROS).

In chapter 2 the system is analysed on component level to formulate a requirements specification forming the basis for chapter 3 describing decision making, chapter 4 describing robotics and chapter 5 describing computer vision. In chapter 6 the principles and implementation of wavelet based 3D modelling in ROS are described and chapter 7 is about the point cloud based calibration system. Finally in chapter 8 the evaluation method and findings are presented and discussed, leading to a conclusion in chapter 9.

Chapter 2

Analysis

The system must be capable of performing 3D reconstruction based on the eye-in-hand principle, meaning that a stereo camera is mounted at the end-effector of a robot arm. Implementation of the system must be based on the Robot Operating System (ROS) as this is the controller interface for both robot and sensors. The architecture of the system(Figure 2.1) is a closed loop structure based on the Good Old Fashion Artificial Intelligence(GOFAI) environment interaction model [16]. In the following each part of the system will be analysed with respect to functionality and responsibilities leading to a formulation of a requirements specification.

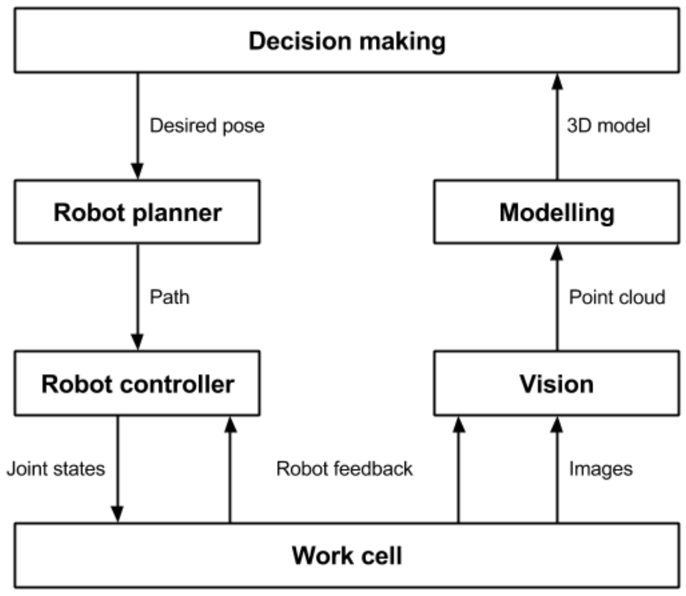


Figure 2.1: Block diagram of the eye-in-hand 3D reconstruction system.

2.0.1 Decision maker

The decision maker is the highest level of abstraction and authority in the system and is responsible for controlling the task, which in this case means moving the robot arm to the right poses and capture images from the sensor. The task can be broken down to starting the process, choosing a pose, executing the pose, capturing the image and ending the task when there are no more poses needed. Any further work like next best view planning, automatic calibration or other alternative tasks will be implemented in the decision maker and it should therefore be general and easy to expand. In the current application, the desired poses will be generated to capture the object from a number of discrete locations on a sphere subject to distance and viewpoint constraints. The captions must be evenly distributed to cover the entire object.

Errors...

2.0.2 Robot planner

The robot planner is the top level abstraction of the robot arm and takes desired pose in 3D space as input generating a desired path in joint space as output. It is responsible for generating the path between the current pose of the robot and the desired pose. The path is subject to a number of physical constraints and must therefore be generated in steps. The first step is to find a collision free path through the workspace thus requiring a kinematic model of the robot as well as a priori 3D models of robot and work cell. When a suitable path has been found, the path must be optimised for length, clearance and undesired movement. In cases where a collision free path cannot be found, the robot planner must relay this back to the decision maker. Implementations in the robot planner must be robust and therefore make use of some of the widely used open source libraries and ROS stacks should be facilitated.

2.0.3 Robot controller

The robot controller executes the path and thus handles communication to low level controllers and feedback sensors on the robot. The path is executed in a closed-loop control system and therefore the joint states realising the path must be converted to actuator velocities. This is done by adding a time dimension to the path, subject to joint velocity constraints. The time tessellated path is further blended to meet acceleration and jerk constraints and is finally interpolated and executed in the control loop. Implementations should be robust and therefore make use of some of the many open source libraries and ROS stacks.

2.0.4 Work cell

The workcell contains a six degrees of freedom RX60 robot arm with a bumblebee stereo camera and a carmine sensor mounted on the end effector(Figure 2.1). The work cell interface is based on a ROS node communicating with the physical robot controller and a node broadcasting data from the camera. Since the RX60 has limited reach, the objects being modelled are hanged from a point over the robot. The a priori model of the work cell contains only the robot arm and simplified bounding boxes representing the immediate surroundings of the robot.

Image missing

2.0.5 Vision

The vision module takes input from the stereo camera mounted on the end effector of the robot and from that generates a 3D point cloud. The images are undistorted and rectified using calibration parameters obtained independently of the system using the ROS calibration node. When the robot is at rest in the desired pose, a signal from the decision maker tells the vision component to capture the two images from the sensor. From the two images a disparity image is formed and by using the obtained projective parameters the point cloud is generated. Implementation should be based on ROS packages and the OpenCV library for image processing.

2.0.6 Modelling

The Modelling part takes point clouds as input and these are transformed to a common frame based on the absolute pose from which they were recorded and a relative pose estimated from matching the points. The combined point cloud is then cropped and filtered to be ready for 3D surface reconstruction. The reconstruction process is performed offline and produces 3D models.

2.0.7 Calibration

The calibration process is responsible for calibrating the system prior to running the task. There are numerous sources of errors in the system, but in a closed-loop system thorough calibration can limit the effects of inaccuracies. Performing a hand-eye calibration can provide the exact transform between camera view and the end-effector, thus in theory removing the need for relative pose estimation in combining the point clouds.

2.0.8 Requirements specification

The above analysis leads to a requirements specification (Table **??**) for the combined system.

Table 2.1: Requirements specification

Block	Component	Requirements
Decision maker	Pose generator	poses on an equidistant sphere
		viewpoint at the object
		evenly distributed
	State machine	choose pose
		ready for capture signal
	Error handler	handle impossible pose
Robot planner	Detector	detect self collision
		detect work cell collision
	Planner	find collision free path
	Optimisation	optimise path length
		optimise clearance
		remove undesired movement
	Model	forward kinematics
		inverse kinematics
		joint limits
Robot controller	Blender	tessellation in time
		handle velocity, acceleration and jerk constraints
	Control	get feedback
		interpolate path
		closed loop control
	Communication	set joint angles
		get joint angles
Work cell	Objects	hang from above the robot
		known model of objects
	Interface	ROS based
Vision	Camera	ROS interface
		calibration for intrinsic parameters
		calibration for projective parameters
	Preprocessor	undistortion
		rectification
	Controller	capture images when signaled
	Processor	generate disparity image
		generate point cloud
Modelling	Filtering	transform points to common frame
		downsample
		combine point clouds
	Reconstruction	generate surface

Chapter 3

Decision making

redundant word?

The decision making component generates 8 poses evenly distributes on an equidistant sphere of radius 0.5 meters always with the viewpoint at the center of the sphere. As the object is not static in the scene, the center of the sphere can be moved by an interactive marker as shown on figure 3.1. The poses are executed by the planner, and when the desired pose has been reached a message will be passed to the sensor nodes.

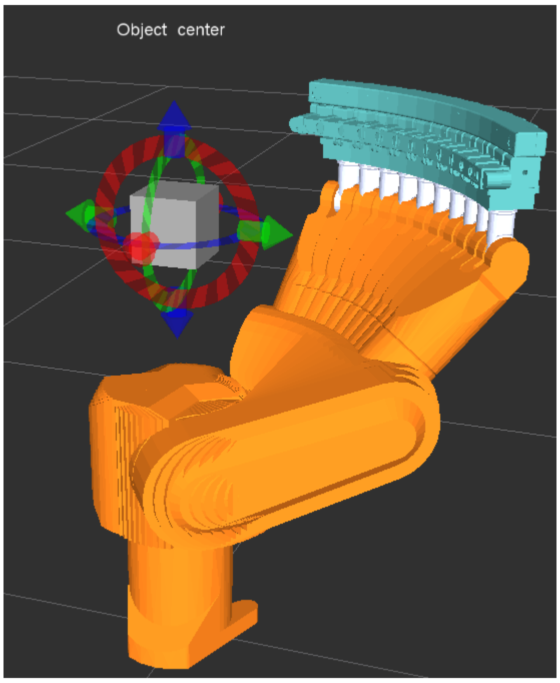


Figure 3.1: The spin center is moveable by an interactive marker. The robot has planned a path to the next desired pose

3.0.9 Special message type

For easy offline manipulation of data a messagetype has been defined to carry the information acquired at each pose. Using this format allows the data to be captured at one point in time and processing at another by using rosbag between the nodes. Initially the message only contains the pose id, max number of poses and poses of the sensors, it is then passed on to the stereo camera node where the captured images are appended. Afterwards the dense_stereo node creates and appends 3D information from the captured images and finally it can be passed to the reconstruction node.

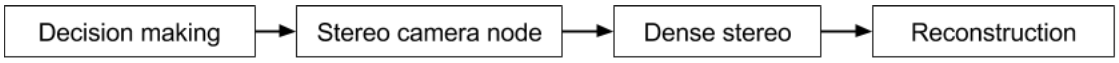


Figure 3.2: The path of the message between the related nodes in the system. In each node new data is appended

Chapter 4

Robotics

Needs to be refering more to the project requirements

.

This chapter describes the setup of the robot system used with the focus of making a quick reference for fast setting up a robot system using tool in the ROS framework. The task of the robot in this project is to be able to move the camera sensor around the object to capture images from different poses. For this purpose a 6 axis Stäubli rx60 has been provided along with a ROS service interface to query and control the configuration. The robot and workcell will be descried in further details in section 4.1. For the robot to be able to move from one pose to another, planners, collision checking and controllers are needed. For the planner to be able to do the collision checking without actually colliding, a model of the robot and the workcell must be present. How to setup this model and available parameters are shortly descriped in section 4.2. Planning a collision free path is done with ROS MoveIt. ROS Control is used to make the interface between MoveIt and the robot. The MoveIt concepts and how to set it up is described in section 4.3, a further look in to the planning algorihtms in section 4.4 and a short introduction to ROS control in section 4.5. A graphical overview of the system is given in figure 4.1

Skipped

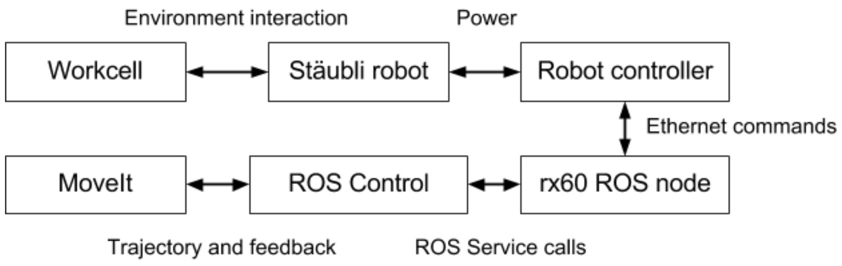


Figure 4.1: Overview of the path from the workcell to the MoveIt stack

4.1 The physical robot and ROS interface

The robot provided for this project is a Stäubli RX60b 6 Axis industrial manipulator. When starting the robot a small rotary switch at the bottom of the controller needs to be turned, and after a few minutes the robot will be booted up and ready. Interfacing the robot is done by a small teach pendant, where it is possible to change application, calibrate, manual control etc. For this project a robot application has been provided such that the robot communicates with a ROS node on a provided machine thus almost no interfacing directly with the robot is necessary. When starting the robot it needs to run the robot application, which is done by releasing the emergency stop, pushing the big green button i the top right corner so it lids up and pressing the (un)pause button. When doing tests with the robot it can be recommended to adjust the moving speed to a minimum to avoid accidents. This is done by the plus and minus buttons on the pendant. The tool mounted on the tool flange is a 3D printet sensor mount with both a Carmine depth camera and a PointGrey BumbleBee stereo camera. Wires have been strip mounted to the robot to avoid them getting stuck in the moving parts. The workcell contains the robot placed on plane big surface, a Troax safety fence and a radiator. Both the fence and the radiator is placed in such distance that the robot will only be able to collide with them in very few configurations. The workcell can be seen in figure 4.2 .

Image placeholder

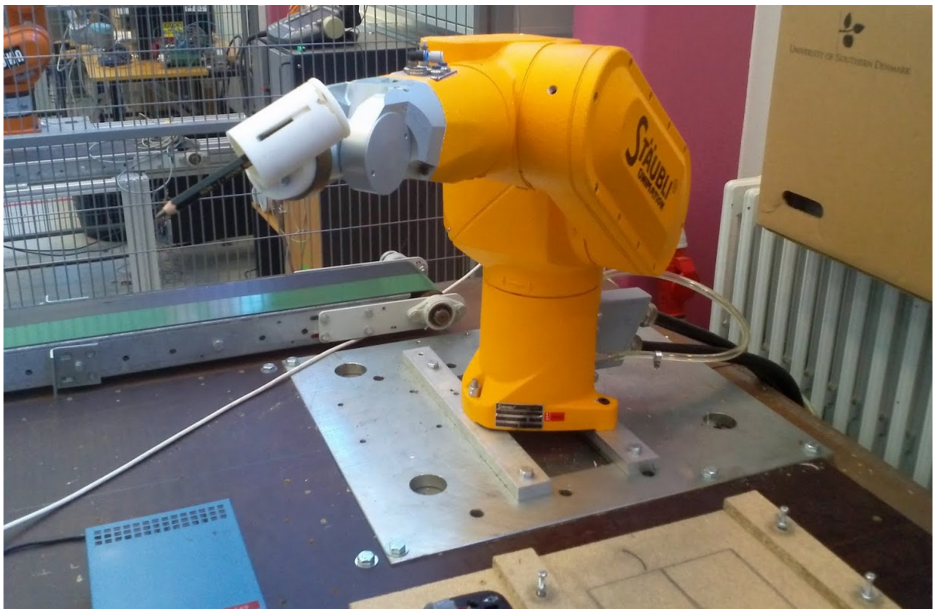


Figure 4.2: Picture of the workcell

The robot axis are controlled by the robot controller which are then connected to a computer with a provided ROS application for communication. To control the robot service calls needs to be made to this node. These service calls provides functionality such as setting various limits, tool position, valve control and most important joint configurations. Furthermore it allows values to be read from the robot, such that feedback to the trajectory controller can be supplied.

4.2 Robot model

To be able to plan a trajectory for the robot, the planner needs a model of the robot. This model is used both for kinematics but also to avoid self collision. The format for describing a robot in ROS is Unified Robot Description Format (URDF). It is also possible to define the robot in the XACRO macro format and convert it to the macro language URDF, however in this project the robot has been descried directly in URDF. A tool that can be used to generate the URDF file directly from a CAD assembly file is available under the name "SolidWorks to URDF Exporter". This tool was used as a basis for the URDF file for this project, but the quality of the output is somehow questionable. A RobWork workcell containing the description of the RX60 robot was available for the project and has been used as a reference to describe the joints, joints limits and also as source for robot CAD files. It needs to be noticed that CAD format has to be binary STL to work as part of the robot description. The tool on the robot has also been described in the URDF to give the necessary transforms for the camera sensors and a intuitive looking visualization of the robot. In the URDF file it is possible to define both a visual CAD of the joints but also a CAD only used for collision checking. To be sure that the tool does not collide with anything, the collision model is a large box with a least 10mm extra space margin with respect to the visual model. The model can be seen on figure 4.3. To use the URDF file it has to be put into the parameter server which can be done in the launch file .

Needs explaning?

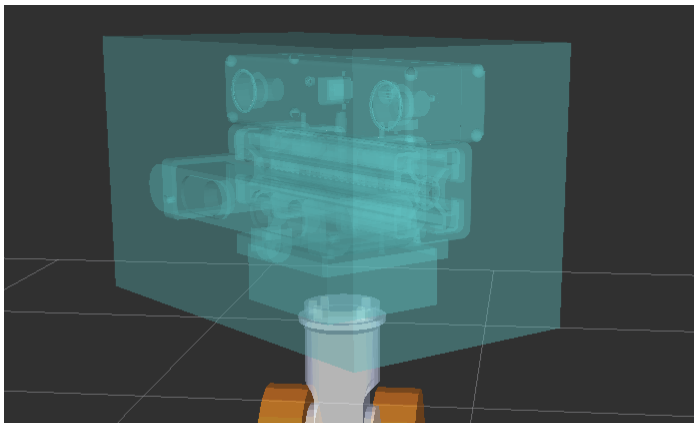


Figure 4.3: The tool mounted on the robot flange and the collision model shown in transparent

4.3 ROS MoveIt

ROS MoveIt is a software planing tool that collect many opensource libraries to a tool that can easily be used for planning with both simple and high dimensionality robots. A great example of a complex robot is the PR2 robot developed at Willow Garage that runs on MoveIt. Controlling a robot with MoveIt can be done by the C++ or python interface, but it also features a planning plugin for the ROS visualizer (Rviz). Combining the programming interface with Rviz gives a great visual feedback, where the planned trajectory can be visualized before and while moving the robot as seen on figure 4.4.

When planning in MoveIt it will by default use the Open Motion Planneing Library (OMPL), however MoveIt provides a plugin interface allowing custom planners to be used. OMPL is a collection of state of art sample based planners that takes a planning request and collision detector as input and if successful returns a planned path. MoveIt then generate a trajectory from the path according to the given robot constraints. The collision checking in MoveIt is done by Flexible Collision Library (FCL). This collision detection allows different input models, such as robot self-collision, workcell model collision and collision with sensed objects. For collision with sensed data, MoveIt has a self-filter, such that if a part of the robot is sensed it can be filtered away from collision map. Without this feature, the sensed robot would collide with the kinematic calculated position of the robot. When configuring a robot with MoveIt a Allowed Collision Matrix (ACM) is generated. This matrix defines joints on the robot that do not need to be checked for collision, such as joints that may never collide in any configuration and succeeding joints. Succeeding joints are of cause able to collide, but this should be avoided by setting prober joint movement constraints instead of the collision detector, as a collision detection calls are expensive in computation. As default MoveIt uses Orocos KDL for solving the kinematics, but for complex robots it is recommended to compile a robot specific solver using OpenRave IKfast. The advantage of a

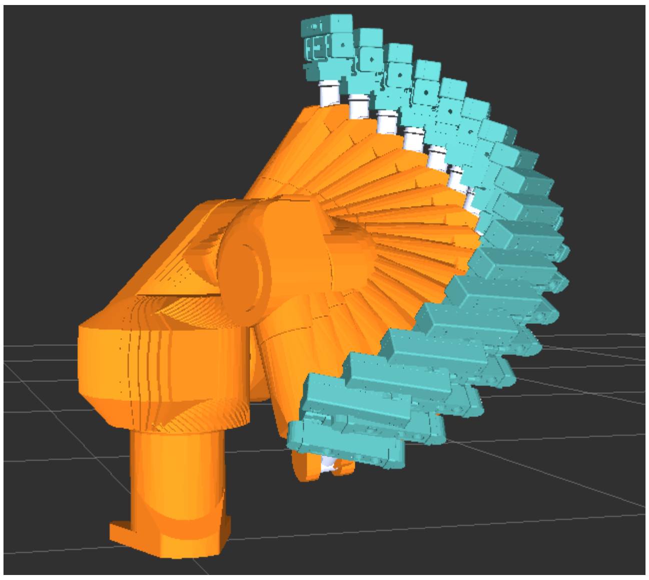


Figure 4.4: Robot trajectory displayed as a series of robot configurations in Rviz

IKfast compiled solver is that it is able to use analytical solvers instead of numerical. When the model of the robot is present, it is easy to start using MoveIt. After installation MoveIt contains a setup tool with a guide trough the whole process. This tool is called MoveIt Setup Assistant and is shown on figure 4.5.

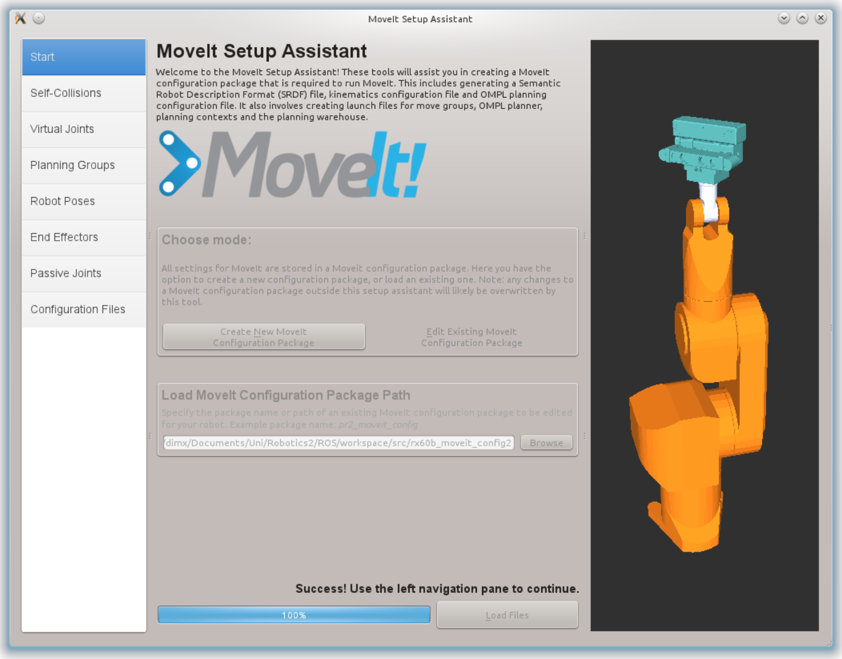
When the MoveIt setup tool has been run, a ROS package has been generated with the prober launch files to start planing with the robot. This package features a demo application which launches Rviz with the motion plugin, so the robot can be seen running. For the real robot to move, at least one controller needs to be implemented. This is described in section 4.5.

4.4 Planning

Tough many planners are available in the OMPL, it has been chosen to use Probabilistic RoadMap (PRM) for this project. PRM is a probabilistic complete multi query planning algorithm with two phases. First the learning phase generates roadmaps in the configuration space by random sampling and collision checking, next the query phase is used to connect the start and goal configuration to the roadmap and within the roadmap find a valid path. The PRM planner has been chosen because the workcell is static and thus the roadmap can be generated beforehand to speed up the planning requests. The definition of which planner to use in MoveIt is defined in the planning request to allow the use of specific planners for certain tasks.

4.5 Robot control

When MoveIt has generated a trajectory for the robot, a controller needs to take care of the physical movement. In the package generated by MoveIt the control is done by fake_controller which does not actually control anything. First of all the MoveItSimple-ControllerManger needs to be loaded to provide a service interface for the actual controller. For actual control there is a package called ros_control which provides a framework and tools for implementing a robot controller. The ros_control package consists of a controller



MoveIt Setup Assistant

Welcome to the MoveIt Setup Assistant! These tools will assist you in creating a MoveIt configuration package that is required to run MoveIt. This includes generating a Semantic Robot Description Format (SRDF) file, kinematics configuration file and OMPL planning configuration file. It also involves creating launch files for move groups, OMPL planner, planning contexts and the planning warehouse.

Choose mode:

All settings for MoveIt are stored in a MoveIt configuration package. Here you have the option to create a new configuration package, or load an existing one. Note: any changes to a MoveIt configuration package outside this setup assistant will likely be overwritten by this tool.

Load MoveIt Configuration Package Path

Specify the package name or path of an existing MoveIt configuration package to be edited for your robot. Example package name: *pr2_moveit_config*

/dimx/Documents/Uni/Robotics2/ROS/workspace/src/rx60b_moveit_config2

Success! Use the left navigation pane to continue.

Figure 4.5: The MoveIt setup tool

manager, a hardware_interface and different types of controllers. To get the robot running, a new package with a node for the controller has to be made. In the node, the framework is included and a description of the robot inputs and outputs are defined. The control loop of the node fetches the state from the robot and passes this to the controller. The controller then outputs the desired movement for the robot which has to be passed on to the robot. As described in section 4.1 the interfacing to the physical robot is done by service calls. In the project it has been chosen that the controller node also is responsible of publishing the current robot state. The robot state is used by MoveIt and robot_state_publisher. The robot_state_publisher is the node responsible for updating the TF tree.

Chapter 5

Vision

This chapter covers the transformation of 2D stereo images into a useful 3D point cloud. Using only a stereo camera as passive sensor.

5.1 Theory - Dense Stereo

Dense stereo uses two 2D images and converts them to a dense 3D point cloud. That is the 3D postion of every point in the 2D images is desired. To obtain this the disparity between all features are required, and the following stages are done with that in mind. The stages are as following.

- Undistortion
- Rectification
- Correspondence
- Reprojection

5.1.1 Undistortion and Rectification

The undistortion and rectification is performed automatically and therefore only a short description will be included. By knowledge of these parameters one could make functions take them into account, but it is much more effective to transform the images.

As a lens is used for the cameras the normal camera model cannot be used, i.e. a 3D point, X, is mapped to the image plane, x, by $x = fX/Z$, where f is the focal length and Z is the distance to the camera. Both radial and tangential distortion exist respectively creating the circular and elliptical distortion. By correcting the image for these parameters all pixels now follow the camera model and their projection line can be found.

The basic idea of rectification is based on epipolar lines. Epipolar lines are created by knowing the transform between the cameras. For every single point in one camera the line on which it can appear on another camera can be found. That is the epipolar line is a mapping of a pixels projection line in one camera to onto the image plane of another camera.

By knowing the transformation between the two images, epipolar lines can be created for every point. An epipolar line is the line on which any point in one image must be positioned in the other image. Thus making the search for a feature much faster as it only occurs in one dimension.

Rectification takes this one step further. By rectifying the two images the camera planes are made to be parallel. Making the search for correspondence a one dimensional, horizontal task, drastically increasing the speed.

Thus undistortion is a manipulation done independently for the cameras and the rectification is performed for the two cameras together, respectively an intrinsic and extrinsic parameter.

5.1.2 Correspondence

Correspondence is the matching between position of pixels in the two images. This creates a disparity map for all pixels in the two images. Though there is no guarantee that the disparity map is actually correct.

Many methods exist the main goal is dense stereo is to minimize the global error.

Global matching tries to minimize the overall error in the disparity image. That is creating the optimal disparity image, though this is an NP-hard problem. Most of the top-ranked algorithms for the Middlesbury Stereo vision comparison **??** and thus more local approaches are implemented.

$$E(D) = \sum_p (C(p, D_p) + \sum_{q \in N_p} P_1 T[|D_p - D_q| = 1] + \sum_{q \in N_p} P_2 T[|D_p - D_q| > 1]$$

The rectification of the images guarantees horizontal epipolar lines, and thus the most simple approach would be a 1D scan. Though this leads to high vertical

The method used is the Semi-Global block matching.

Mutual Information - handling radiometric distances

Smoothness constraint - vertical smoothness

The matching can be done with either a single pixel or larger pixel patches.

5.1.3 Reprojection

During the calibration of the cameras, projections matrices were obtained for both cameras. The projection matrices are based on the left camera lens, meaning the baseline, distance to the left camera, is 0 for the left matrix. By multiplying the projection matrix with a 3D point in the left cameras compared to the left camera, the points position in the image can be found. That is:

$$\begin{pmatrix} x \\ y \\ \omega \end{pmatrix} = \begin{pmatrix} F_x & 0 & C_x & -F_x T_x \\ 0 & F_y & C_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Here F_x and F_y are the focal length of the camera, C_x and C_y are the middle position of the image.

T_x is the translation to the left camera. That is for the right camera it is the baseline and for the left it is 0. The baseline is measured in millimetres.

But goal is to obtain the opposite, the transform from 2D to 3D. That is the action of converting the obtained disparity map into a 3D point cloud. To do that the x and y positions are combined with the disparity. Thus multiplying with the Q matrix the 3D map position can be found.

$$Q[x\ y\ d\ 1]^T = [X\ Y\ Z\ W]^T$$

Where the Q matrix is defined by the right projection matrix, except for the C'_x.

$$Q = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 0 & F_x \\ 0 & 0 & -1/T_x & (C_x - C'_x)/T_x \end{pmatrix}$$

Thus multiplying with the above mentioned vector, and scaling W to one the 3D position can be obtained.

$$\begin{pmatrix} x - C_x \\ y - C_y \\ F_x \\ d - 1/T_x \end{pmatrix} \Rightarrow \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \frac{x - C_x}{d(-1/T_x)} \\ \frac{y - C_y}{d(-1/T_x)} \\ \frac{F_x}{d(-1/T_x)} \\ 1 \end{pmatrix}$$

5.2 Method - Images to point cloud

As the project uses ROS for communication between nodes as much as possible ROS have been used for the stereo operations. Though it wasn't possible to use generic ROS nodes for every operation, the reason for this is explained in section 5.4.

5.2.1 Calibration

To perform the undistortion and rectification the intrinsic and extrinsic parameters of the cameras are required. These were obtained by the ROS node *stereo_camera_calibrate* by capturing images of a known chessboard pattern in different positions and rotations. When enough images have been captured the ROS node have enough input to solve for the camera parameters. It thus returns the camera matrix along with distortion parameters and the distance between the cameras.

5.2.2 Undistortion and rectification

Undistortion and rectification is done with a simple ROS node called image *image_proc*. By setting giving the raw images together with a camera calibration file, created beforehand in section 5.2.1. It uses these and returns rectified and undistorted images ready to be used by block matching algorithms.

5.2.3 Blockmatching and reprojection

The blockmatching is done with the OpenCV function SGBM described in section 5.1.2. Time stamps are checked for arriving images to make sure that they are recorded at the same time. The SGBM then uses the images and a certain set of parameters, discussed in section 5.4 to make a disparity map.

A ROS node is created takes the rectified images, creates a disparity map and reprojects this to 3D using OpenCV functions.

5.3 Results

Here results from the different operations are shown, these results are qualitative to show that the concept works not quantitative for a measurement of how well.

5.3.1 Calibration

From the calibration, the both the distortion parameters and the projection matrixes are returned. The distortion parameters can be seen in equation 5.2. A more in depth description of the undistorion can be seen in the OpenCV book

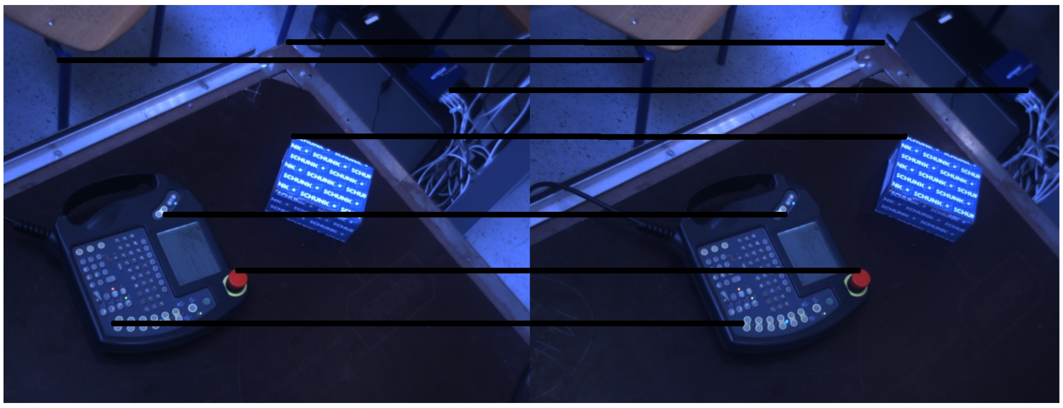
$$[k_1, k_2, p_1, p_2, k_3] = [\ -0.422868,\ 0.192589,\ -0.000004,\ 0.002985,\ 0.000000\ ] \tag{5.1}$$

The rotation matrix for rectification are also returned from the calibration. It can be seen that the rotation needed is small but still needed for the 1D correspondence to work.

$$R = \begin{pmatrix} 0.998701 & -0.002701 & -0.050892 \\ 0.002645 & 0.999996 & -0.001179 \\ 0.050895 & 0.001043 & 0.998703 \end{pmatrix} \tag{5.2}$$

Figure 5.1 shows and example of two output images from *image_proc*. These images are both undistorted and rectified by the function. The image have included horizontal lines showing the effectiveness of the image to align features.

Figure 5.1: Result of undistortion and rectification on an image pair. The horizontal lines shows corresponding pixels in the two images.



5.3.2 Disparity map

From two rectified images the disparity map can be created. Figure 5.2 is an example of such a disparity map. The overall structure of the scene is recognizable, but it can be seen that algorithm does not necessarily guarantees complete coverage with standard parameters. Thus section 5.4 will include a discussion of the selection of these parameters.

5.3.3 Reprojected point clouds

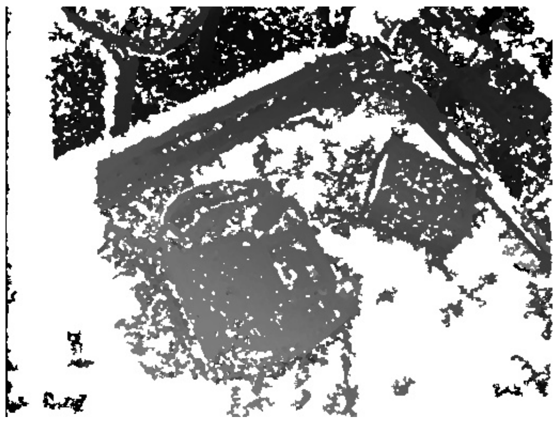
The only thing left to do is reprojecting the disparity map into 3D coordinates. From the calibration the parameters in 5.3 were given for the right camera.

$$\begin{aligned} C_x &= 582.152313 \\ C_y &= 393.397633 \\ F_x &= 1321.556521 \\ -F_xT_x &= 158.751707 \end{aligned} \tag{5.3}$$

Given $-F_xT_x$ the baseline between the cameras calculated as in equation 5.4.

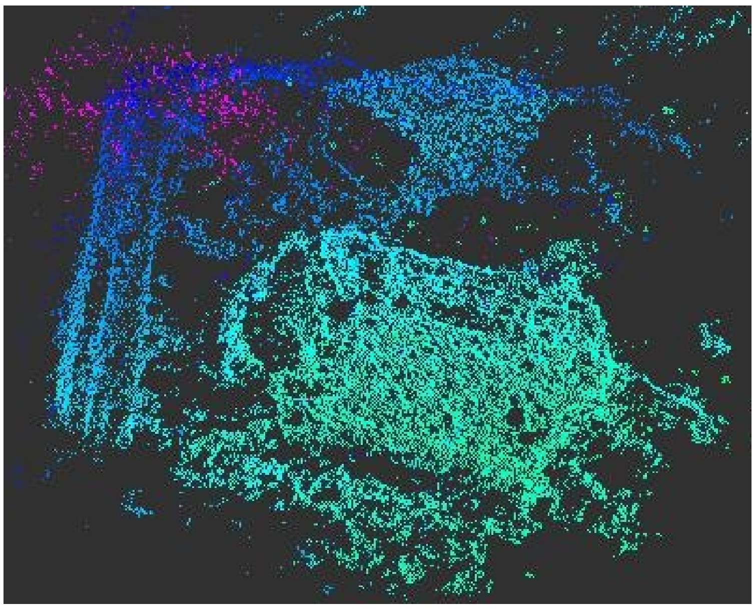
$$Tx = -F_xT_x/(-Fx) = Tx = 158.8/-1321.6 = 0.1201 meters \tag{5.4}$$

Figure 5.2: Disparity image for the two images shown in figure 5.1.



Using these parameters for the Q-matrix, the disparity image is reprojected to 3D. An example of the reprojection for the disparity image in figure 5.2 can be seen in figure 5.3

Figure 5.3: Point cloud reprojected using the parameters shown in section 5.3.3 performed on the disparity image in figure 5.2.



5.4 Optimizing parameters towards the task

In most of the functions the input were non-adjustable values, but for the SGBM there are certain parameters were no single answers exist. The most important being the disparity range i.e. how far the algorithm will search for

When the 3D positions were extracted in section 5.1.3 the connection in equation 5.5 between distance and disparity were found.

$$Z = \frac{F_x}{d(-1/T_x)} \tag{5.5}$$

When solving for disparity the connection in equation 5.6 is given.

$$d = \frac{F_x T_x}{Z} \tag{5.6}$$

That is with a known distance to the setup the one knows the search area, and thus the disparity range can also be calculated. Thus the distance is around 0.5 meters and the focal length and baseline is known from the calibration and shown in section 5.3.3. Thus the expected disparity can be calculated as in equation 5.7.

$$d = \frac{1321.55pixel * 0.1201m}{0.5m} = 317.4pixel \tag{5.7}$$

Thus the minimum and maximum disparity both have to range around this number. It is because of this that the point cloud generated from the ROS *image_proc* cannot be used. The node is only able to search with a maximum disparity range of 128 pixel. This gives a minimum distance of 1.24 meters. This wouldn't fit the desired setup at all and therefore the automatically generated point cloud is rejected. The exact disparity range can be found by looking at the maximum size of the object. This is an important aspect as setting the correct disparity range will drastically increase the search time and potentially remove mismatches.

The algorithm is only able to scale the disparity in ranges of 16.

From these equations the minimum and maximum disparity are respectively chosen to be 16*16=256 and 16*24=384, resulting in a range from 0.41 to 0.61 meters. Thus if the distance to object should move outside this range it will not be found.

Speckle settings are another important parameter, though adjusting this depends entirely on the use of the point cloud. If the reconstruction doesn't take care of noise the speckle-size should be very low. The speckle range should also accommodate the point cloud so as to keep the signal/noise as high as possible, but still fitting the system.

Chapter 6

Modelling

The task of reconstructing an unknown surface from a point cloud is not a trivial task. Especially not when this point cloud is obtained through a camera mounted as the tool on a robot arm. To be able to reconstruct the full surface several views of the object are required, which mean that several point clouds are required to be aligned with each other and stitched together. The modelling component of this system is divided into two sub-components, the first component described in this chapter is filtering and the second component described is the reconstruction component.

6.1 Filtering

Filtering of raw point clouds is required because of several different reasons. The number of points delivered to the modelling component from the raw point clouds is huge, so filtering non-interesting points away creating a region-of-interest (ROI) in the raw point clouds. Lowering the number points in each cloud delivered to the modelling component is required such the workload can be kept within an acceptable range. The number of points can be further reduced by down-sampling the points left in the ROI by the cut-off filter. A voxel-grid filter utilised for down-sampling also creates the advantage of equal sampling density, but the disadvantage is that the down-sampling means loss of information, and therefore there is a trade off between speed and level of detail of the reconstruction process.

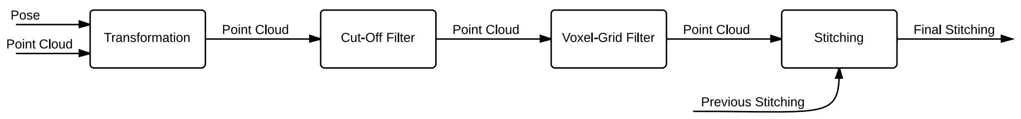


Figure 6.1: Illustrates the flow through the filter.

In figure 6.1 an illustration of the flow through the filter can be seen. The vision layer is delivering a point cloud for each individual view, along with this point cloud a pose of the frame in which the points are recorded is delivered. These two sets of data is delivered to the ROS node which handles filtering, this node filters each individual cloud and stitches each of them together in a common cloud. This common cloud, when finished, is sent to the reconstruction layer. In the sections below is a brief description of the components shown in figure 6.1.

6.1.1 Point cloud library

The Point Cloud Library (PCL) utilised in this project is a library which provide functionality for working on 3D point clouds. PCL delivers a variety of functionality such as filters, segmentation, surface reconstruction, kd- and oc-trees, visualisation, etc. PCL can be found at http://www.pointclouds.org/, along with documentation and tutorials.

6.1.2 Point cloud transformation

Messages received from the vision layer needs to be processed before filtering. This is because the messages delivered to the modelling component contain a point cloud and a pose of the current camera view. The coordinates of the individual points in the cloud are related to the camera frame, but this frame is moving around the object so a transformation of points is needed such they can be related a common static frame.

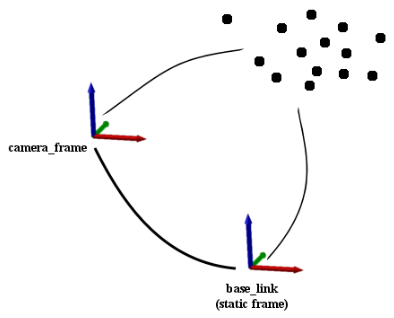


Figure 6.2: Illustrates points in different frames.

6.1.3 Cut-off filter

The cut-off filter utilised is the implementation from PCL, `pcl::PassThrough< ...>`. A cut-off filter is utilised to create a ROI in the transformed point cloud, partly because the number of points needs to be reduced with respect to processing time and then because the region in which the object resides is fairly small to the region which is recorded by the camera.

6.1.4 Voxel-grid filter

The voxel-grid down-samples the left overs from the cut-off filtering. The down-sampling causes loss of information which mean that surface details are lost in the process, so the amount of down-sampling should be chosen with respect to processing time versus level of detail to be reconstructed. The PCL library luckily have such functionality (`pcl::VoxelGrid< ...>`) which is utilised in the filtering sub-component.

The voxel-grid filter works by splitting down the ROI into smaller regions (voxels) of certain resolution in which each of the voxels are analysed. Figure 6.3 show the principle of the voxel-grid. A new point is approximated for each voxel, the new point is approximated by the points centroid which is contained in the voxel. This method is a little slower compared to just placing the new point in the center of the voxel, but it helps save some more detailed information about the surface curvature.

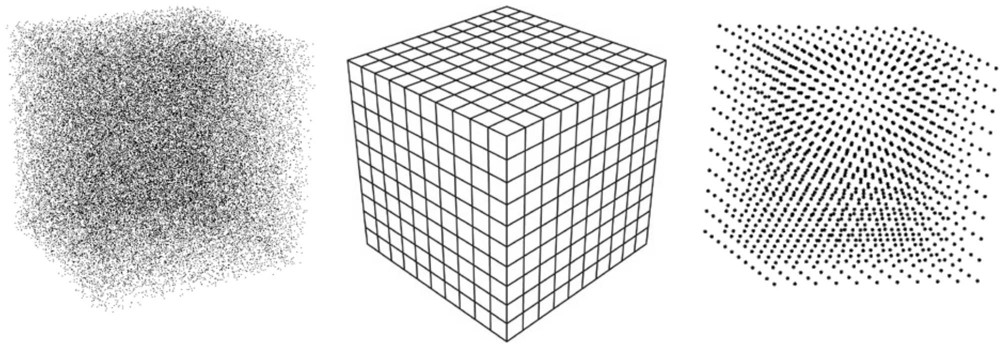


Figure 6.3: Illustrates the down sampling principle of the voxel-grid filter. The left point cloud is down sampled through a set of virtual cubes, which leaves the right cloud after processing the input. A uniform distribution of points is obtained this way for instance.

6.1.5 Point cloud stitching

Subsection not finished

The individual point clouds from the individual views of the object are passed through the filter, the individual cloud is then stitched into another common cloud which is assembled from a number of individual clouds, predetermined by the decision making layer. The stitching of each cloud happens in the static frame, due to the transformation of the input to the filter node. The point clouds are stitched in the order at which they are received, thus the first frame becomes a sort of reference frame for the following frames. This is because the algorithm used for stitching the clouds together actually calculates a transform at which new incoming point clouds are transformed. The algorithm utilised for stitching is Iterative Closest Point (ICP). This is also implemented in in PCL and therefore this implementation (`pcl::IterativeClosestPointNonLinear< ..., ...>`) of the algorithm utilised. The ICP algorithm has it downsides. For example if a cloud is not aligned correctly, this will cause an error to propagate thought to the following clouds which could lead to a rather distorted model of the object of interest[3]. The algorithm will be described in further details in chapter `ref??`.

Ref til calibration chapter

6.1.6 Results

Missing section content

6.2 Surface reconstruction

Surface reconstruction from a large set of unstructured points obtained through laser scanning techniques or stereopsis is not a trivial task but nonetheless it is useful in many applications. For example a surface reconstruction could aid a robots end effector grasping some unknown object [20]. Many methods for surface reconstruction exist in different domains of science, some algorithms are based on neural network approaches [21], others are based on sculpting or region growing algorithms (e.g [2] and [1]) used in computational geometry, some utilise an implicit method framework to represent incoming data as a surface [11] and [5].

In 1999 F. Bernardini et al. proposed in a fairly simple algorithm (Ball Pivoting Algorithm (BPA)) for reconstruction of surfaces from point clouds sampled over smooth objects [2]. BPA is pivoting a sphere of a certain diameter around an edge of a seed triangle. Pivoting the sphere around all the edges is connecting three points to form a new triangle and so on. BPA is a part of the region growing algorithms as this algorithm uses a seed triangle builds the surface around this and outward. The algorithm is fairly easy to work with as it has one parameter which decides the radius of the sphere and that is it. N. Amenta et al. proposed in 2001 [1] the power crust algorithm, which essentially is a three dimensional Voronoi approach [12]. The power crust algorithm is well defined and proven which makes it one of the most known algorithm regarding surface reconstruction. This algorithm is a sculpting algorithm in computational geometry. Common for those algorithms mentioned is that these are explicit methods which requires neighboring information which leads to high computational time consumption. Therefore methods mentioned above are not well suited for reconstruction of large data sets.

In 2006 M. Kazhdan et al. proposed in [11] an algorithm which resides in the implicit method framework, this algorithm is called Poisson Surface Reconstruction (PSR). PSR is a fitting scheme that allow solving for the indicator function of the surface. It is shown in [11] this approach of fitting a surface resembles the Fast Fourier Transform (FFT). In [11] it is shown that the FFT approach requires five times as much space but is approximately double as fast while creating approximately the same number of triangles. The real advantage of the Poisson approach is that it is scalable and therefore does not rely on a uniform distribution of points and thus a higher degree of details can be reconstructed in areas where the point density is higher.
J. Manson et al. did in 2008 propose a wavelet approach of the problem of reconstructing surfaces of large sets of unstructured points [14]. The method is robust to noise in data because it is relying on implicit methods as PSR, and is therefore well-suited for reconstruction of real data which is overlayed with some random noise. The wavelet approach an advantages over PSR, the wavelet approach is able to reconstruct non-closed surfaces in opposition to PSR [11]. Using the Haar wavelet synthesise an acceptable surface if the points are well-aligned, else more smooth wavelets can be used such as Daubechies-2. Using octrees and small-support wavelets makes the synthesisation very fast, smaller support equals less computational time and space. Using Daubechies-2 compared to Haar increases computational time by a scale of four, and the space consumption is upscale by approximately four as well. As the methods for reconstruction mentioned first does not guarantee watertight mesh and does not work very well with large point sets, mean those methods are not of interest for further studies. The PSR and the wavelet approach both does estimate an indicator function of the surface, but each handles this indicator function differently. As these method is based on implicit methods they are more robust to noise in the input and approximates the surfaces better compared to the methods mentioned first.

6.2.1 Wavelets

6.2.2 Indicator function

6.2.3 Mesh generation

6.2.4 Implementation

6.2.5 Results

Chapter 7

Calibration

In the process of combining data from several view points, a model of the system is needed as to align the measurements correctly. An idealised model can be formulated based on motion theory, but in inherently imperfect hence the need for calibration. In the context of eye-in-hand based surface reconstruction, especially calibrating the kinematic chain from object robot base is important in order to combine data from several views. Calibration of robot systems has received considerable attention and continues to be an active field of research. A solution for the unknown transforms from camera to end effector and from object to robot base can be obtained relatively easy by solving a homogeneous transform [17]. Several algorithms for more or less autonomous calibration has been proposed [18, 19] often simultaneously calibrating camera and hand-eye [10, 13, 23]. The calibration process presented here is based on the linear method [19] with some application specific modifications.

7.0.6 Notation

In the formulation of hand-eye calibration a number of homogeneous transforms between frames are be described. Most literature uses a mathematically founded notation of Ai, Bi, A, B, X and Y leading to nice and clean equations. In this work a more explicit and engineering friendly notation will be used stating for each transform the state index (since there are several different robot states involved to allow different views of the object), the name of the frame described and the name of the reference frame. Examples of the notation are explained in table 7.1. The transform is a a 4x4 matrix containing a 3x3 rotation matrix and a 1x3 translation vector. The terms end-effector and Tool Center Point(TCP) will be used interchangeably.

Table 7.1: Examples of the notation used to describe rigid transformation.

$[T_i]_{camera}^{object}$	The transform from object to camera at the ith robot state
$[T_i]_{base}^{TCP}$	The transform from end-effector to base at the ith robot state
$[T_i]_{TCP}^{camera}$	The transform from camera to end-effector at the ith robot state
$[T_{i-1,i}]_{camera}$	The camera frame at state i relative to the camera frame at state *i-1*

7.0.7 Linear hand-eye calibration

Looking at figure 7.1 (left and right) it is clear that a kinematic chain from the object to the robot base can be formulated.

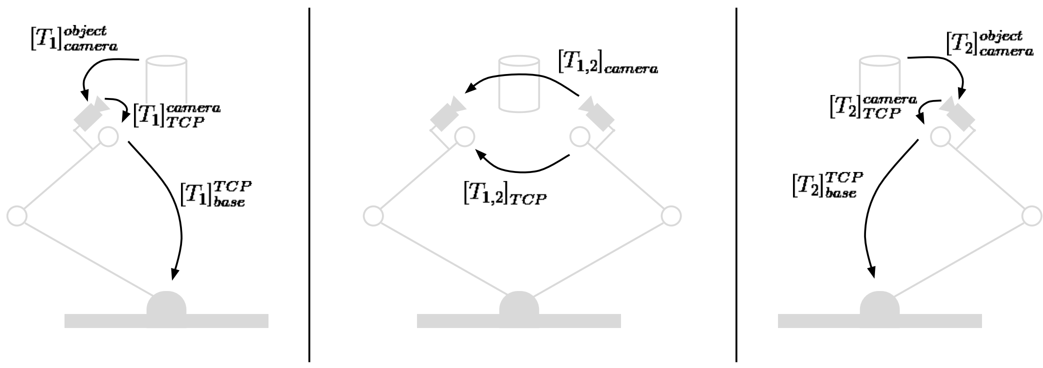


Figure 7.1: Sketch of a robot with a camera mounted on the end-effector in two different states capturing a cylinder object. Shows the robot in state 1 (left), state 2 (right) and both robot states in same image (center).

$$[T_i]_{base}^{object} = [T_i]_{base}^{TCP} \cdot [T_i]_{TCP}^{camera} \cdot [T_i]_{camera}^{object} \tag{7.1}$$

The transform from end-effector to robot base can be found from the joint states of the robot and the transform from object to camera can, assuming the camera is calibrated, be found from projection geometry. This leaves only the transform from camera to end-effector as unknown. Assuming that the object is static and captured from different robot states, it is possible to formulate the transform between the two camera poses (Figure 7.1, center).

$$\begin{gathered} [T_{1,2}]_{camera} \cdot [T_1]_{camera}^{object} = [T_2]_{camera}^{object} \\ \Updownarrow \\ [T_{1,2}]_{camera} = [T_2]_{camera}^{object} \cdot ([T_1]_{camera}^{object})^{-1} \end{gathered} \tag{7.2}$$

Similarly the transform between the two poses of the end-effector can be formulated(7.1, center).

$$\begin{gathered} [T_1]_{base}^{TCP} \cdot [T_{1,2}]_{TCP} = [T_2]_{base}^{TCP} \\ \Updownarrow \\ [T_{1,2}]_{TCP} = ([T_2]_{base}^{TCP})^{-1} \cdot [T_1]_{base}^{TCP} \end{gathered} \tag{7.3}$$

Assuming that the transform from camera to end-effector is static it is clear that.

$$[T_1]_{TCP}^{camera} = [T_2]_{TCP}^{camera} = [T]_{TCP}^{camera} \tag{7.4}$$

This makes it possible to formulate a closed loop kinematic chain (7.1, center). This equation is in most literature denoted $AX = XB$.

$$[T_{1,2}]_{camera} \cdot [T]_{TCP}^{camera} = [T]_{TCP}^{camera} \cdot [T_{1,2}]_{TCP} \tag{7.5}$$

The missing transform can thus be found by solving a set of $n-1$ linear equations for n robot states

$$\begin{cases} [T_{1,2}]_{camera} \cdot [T]_{TCP}^{camera} = [T]_{TCP}^{camera} \cdot [T_{1,2}]_{TCP} \\ \vdots \\ [T_{i-1,i}]_{camera} \cdot [T]_{TCP}^{camera} = [T]_{TCP}^{camera} \cdot [T_{i-1,i}]_{TCP} \\ \vdots \\ [T_{n-1,n}]_{camera} \cdot [T]_{TCP}^{camera} = [T]_{TCP}^{camera} \cdot [T_{n-1,n}]_{TCP} \end{cases} \tag{7.6}$$

Normally the calibration is made by capturing views of a chessboard marker from several robot states. There are however two rather implicit assumptions [8] not justified by this approach. First the calibration of the camera assumes a perfect pinhole model and that the optical axis of the camera is perpendicular to the object, but equation 7.5 assumes that the object is static, which for most practical objects are contradictions. There are several approaches to solving this, but those are outside the scope of this discussion.

Refs

7.0.8 Application specific method

In this work an alternative approach is taken to include more of the possible uncertainties in the hand-eye calibration and to account for the unjustified assumptions mentioned above. Uncertainties in the vision process generating the transform from object to camera could stem from inaccurate parameters, the pinhole model assumption or pixel quantisation errors to mention a few. Uncertainties in the transform from end-effector to base can stem from mechanical inaccuracies, wear and tear, quantisation errors in encoders or from dynamics, also just to mention a few.

Solving the linear system from equation 7.6 is equivalent to making a least-squares fit and thus after calibration a fixed transform minimising the error over all robot states is applied to each state. The problem with this is that a linear method is used to estimate errors that are by no means linear. Furthermore the method is vulnerable to outliers, since one faulty view will influence all others. Instead a method capable of fitting non-linear functions is desirable, but to use such, individual errors at each robot state must be known. Furthermore it is desired to include as much of the reconstruction process as possible inside the the calibration loop, to account for as many errors as possible. Therefore it is desirable to base the calibration on the point clouds instead of the camera output and furthermore it is desirable to evaluate each data pair instead of pooling them for all views.

Estimating a rigid transform between two point clouds is considered an absolute orientation problem [9] and can be solved with an Iterative Closest Point(ICP) algorithm. The ICP algorithm tries to estimate a transform to align the source cloud to the target cloud by alternating between matching points that are close, estimating the transform and transforming the remaining points. Normally the transform is estimated using least-squares, but an alternative implementation using damped least-squares[6] and thus not assuming linearity is found in PCL (`pcl::IterativeClosestPointNonLinear< ..., ...>`). This is already implemented as part of the point cloud registration process is the modelling component where it is used for pairwise alignment of the point clouds as part of the registration process. Knowing the pairwise pose estimate means that another closed loop formulation can be expressed (Figure 7.2).

Taking as example the camera pose at state 1 ($[T_1]_{base}^{camera}$) and following the pairwise alignment transforms all the way around ending up back at state 1 the result must be equal to the original pose and the error at that pose.

$$\begin{gathered}[T_1]_{base}^{camera} \cdot [T_{12}]_{PC} \cdot [T_{23}]_{PC} \cdot [T_{34}]_{PC} \cdot [T_{45}]_{PC} \cdot [T_{51}]_{PC} = [T_1]_{base}^{camera} \cdot [T_1]_{camera}^{error} \\ \Updownarrow \\ [T_{12}]_{PC} \cdot [T_{23}]_{PC} \cdot [T_{34}]_{PC} \cdot [T_{45}]_{PC} \cdot [T_{51}]_{PC} = [T_1]_{camera}^{error}\end{gathered} \tag{7.7}$$

Looking at the closed loop in figure 7.2 if say the view at state 3 had a large error, that error would in the evaluation of all other states result in two contributions (T_{23} and T_{34}) of equal magnitude and opposite sign. This means the method is robust to outliers. Now having estimated the error at each pose, a non-linear method can be used to estimate the calibrated transform. The ideal method would be one eliminating the error in each state, while interpolating in between. However such a method cannot be made with a static transform.

Her er der som jeg ser det to muligheder. Enten følger vi Kents idé om at publicere et dynamisk tf som interpolerer mellem fejlene eller også benytter vi en damped least-squares eller lignende til at bestemme et statisk tf...

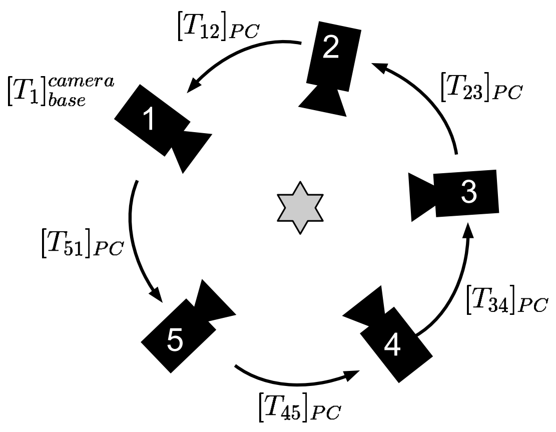


Figure 7.2: The closed loop formed by the point cloud transformations.

Chapter 8

Results and Discussion

To evaluate the performance of the system, a series of paired experiments were made before and after calibration. In the experiments three known (true 3D model is present) objects with different level of detail were used (Figure **??**). In each experiment the robot goes through ??? the 24 poses, capturing the object from 24 viewpoints. The system is then calibrated based on the results and the test is repeated. From each experiment two 3D models are built one before and one after calibration.

8.0.9 Evaluating the calibration

To evaluate the models built before calibration are visually compared to the models built after calibration to assess the improvement. More quantitative measures could be made, but if the difference is not visible to the naked eye, it is not worth doing the calibration.

8.0.10 Evaluating the reconstruction

To evaluate the combined reconstruction process the 3D models from the experiments are compared to the known model in a quantitative way...

Chapter 9

Conclusion

RoVi is the best course ever .... blablabla

Bibliography

[1] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. "The power crust, unions of balls, and the medial axis transform". In: *Computational Geometry* 19.2-3 (July 2001), pp. 127–153. ISSN: 09257721. DOI: `10.1016/S0925-7721(01)00017-7`. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0925772101000177`.

[2] F. Bernardini et al. "The ball-pivoting algorithm for surface reconstruction". In: *IEEE Transactions on Visualization and Computer Graphics* 5.4 (1999), pp. 349–359. ISSN: 10772626. DOI: `10.1109/2945.817351`. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=817351`.

[3] T.E. Choe and University of Southern California. Computer Science: Doctor of Philosophy. *Registration and Three-dimensional Reconstruction of a Retinal Fundus from a Fluorescein Images Sequence.* University of Southern California, 2007. ISBN: 9780549237112. URL: `http://books.google.dk/books?id=vBbOFccdtj8C`.

[4] K. Daniilidis. "Hand-eye calibration using dual quaternions". In: *The International Journal of Robotics Research* 18.3 (Mar. 1999), pp. 286–298. ISSN: 0278-3649. DOI: `10.1177/02783649922066213`. URL: `http://ijr.sagepub.com/cgi/doi/10.1177/02783649922066213http://ijr.sagepub.com/content/18/3/286.short`.

[5] Bin Dong and Zuowei Shen. "Wavelet frame based surface reconstruction from unorganized points". In: *Journal of Computational Physics* 230.22 (Sept. 2011), pp. 8247–8255. ISSN: 00219991. DOI: `10.1016/j.jcp.2011.07.022`. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0021999111004505`.

[6] H Gavin. "The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems". In: *Department of Civil and Environmental Engineering, ...* (2011), pp. 1–17. URL: `http://scholar.google.com/scholar?hl=en\&btnG=Search\&q=intitle:The+Levenberg-Marquardt+method+for+nonlinear+least+squares+curve-fitting+problems\#0`.

[7] Heiko Hirschmüller. "Stereo processing by semiglobal matching and mutual information." In: *IEEE transactions on pattern analysis and machine intelligence* 30.2 (2008), pp. 328–341. ISSN: 0162-8828. DOI: `10.1109/TPAMI.2007.1166`.

[8] Radu Horaud and F. Dornaika. "Hand-Eye Calibration". In: *The International Journal of Robotics Research* 14.3 (June 1995), pp. 195–210. ISSN: 0278-3649. DOI: `10.1177/027836499501400301`. URL: `http://ijr.sagepub.com/cgi/doi/10.1177/027836499501400301`.

[9] BKP Horn. "Closed-form solution of absolute orientation using unit quaternions". In: *JOSA A* 4.April (1987). URL: `http://www.opticsinfobase.org/abstract.cfm?uri=josaa-4-4-629`.

[10] Andreas Jordt, Nils T Siebel, and Gerald Sommer. "Automatic High-Precision Self-Calibration of Camera-Robot Systems". In: (2009), pp. 1244–1249.

[11] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson Surface Reconstruction". In: (2006).

[12] Hugo Ledoux. “Computing the 3D Voronoi Diagram Robustly: An Easy Explanation”. In: *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)* Isvd (July 2007), pp. 117–129. DOI: 10.1109/ISVD.2007.10. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4276112.

[13] Henrik Malm and Anders Heyden. “Simplified intrinsic camera calibration and hand-eye calibration for robot vision”. In: *Intelligent Robots and Systems, 2003.(...* October (2003). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1250764.

[14] J. Manson, G. Petrova, and S. Schaefer. “Streaming Surface Reconstruction Using Wavelets”. In: *Computer Graphics Forum* 27.5 (2008), pp. 1411–1420. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2008.01281.x. URL: http://doi.wiley.com/10.1111/j.1467-8659.2008.01281.x.

[15] W E N Peizhi et al. “Non-Closed Surface Reconstruction Based on Wavelet Trans-Form and Maximum Blurred Entropy”. In: (2013), pp. 8088–8091.

[16] Rolf Pfeifer, Max Lungarella, and Fumiya Iida. “Self-organization, embodiment, and biologically inspired robotics.” In: *Science (New York, N.Y.)* 318.5853 (Nov. 2007), pp. 1088–93. ISSN: 1095-9203. DOI: 10.1126/science.1145803. URL: http://www.ncbi.nlm.nih.gov/pubmed/18006736.

[17] YC Shiu and S Ahmad. “Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form AX= XB”. In: *Robotics and Automation, IEEE ...* 5 (1989), pp. 16–29. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=88014.

[18] RY Tsai and RK Lenz. “A new technique for fully autonomous and efficient 3D robotics hand/eye calibration”. In: *Robotics and Automation, IEEE ...* 5.3 (1989). URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=34770.

[19] RY Tsai and RK Lenz. “Real time versatile robotics hand/eye calibration using 3D machine vision”. In: *Robotics and Automation, 1988. ...* (1988), pp. 554–561. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=12110.

[20] B. Wang et al. “Grasping unknown objects based on 3d model reconstruction”. In: *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.* (2005), pp. 461–466. DOI: 10.1109/AIM.2005.1511025. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1511025.

[21] Xue-mei Wu and Gui-xian Li. “Sparseness Points Cloud Data Surface Reconstruction based on Radial Basis Function Neural Network (RBFNN) and Simulated Annealing Arithmetic *”. In: (2007), pp. 881–884. DOI: 10.1109/CIS.Workshops.2007.110.

[22] Zhengyou Zhang. “A flexible new technique for camera calibration”. In: *Pattern Analysis and Machine Intelligence, IEEE ...* 1998.11 (2000), pp. 1330–1334. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=888718.

[23] Zijian Zhao and Yuncai Liu. “Integrating Camera Calibration and Hand-Eye calibration into robot vision”. In: *7th World Congress on Intelligent Control and Automation.* Chongqing, China, 2008, pp. 5721–5727. ISBN: 9781424421145.

Appendix A

Sample appendix

RoVi is the best course ever .... blablabla