



**NTNU – Trondheim**  
Norwegian University of  
Science and Technology

# Vision Based Robotic Control

**Eirik Anfindsen Solberg**

Subsea Technology

Submission date: June 2014

Supervisor: Olav Egeland, IPK

Norwegian University of Science and Technology  
Department of Production and Quality Engineering



## MASTEROPPGAVE

Våren 2014

for stud. techn. Eirik Anfindsen Solberg

### Robotstyring ved bruk av datasyn

To KUKA Agilus industriroboter skal settes opp med datasyn og testes i denne oppgaven. Det er spesielt av interesse å undersøke responstid ved bruk av datasyn for korreksjon av robotens programmerte bevegelser.

1. Sett opp robotene og prøv ut enkel robotstyring.
2. Lag en modell av robotcellen i et grafisk simuleringssystem og prøv ut enkle bevegelsesprogrammer for robotene.
3. Sett opp et system for datasyn som kan kobles til robotenes grensesnitt mot sensorstyring.
4. Lag en demo hvor en robot holder et bevegelig mål og den andre roboten følger dette målet ved bruk av datasyn og vurder ytelsen til systemet.

Oppgaveløsningen skal basere seg på eventuelle standarder og praktiske retningslinjer som foreligger og anbefales. Dette skal skje i nært samarbeid med veiledere og fagansvarlig. For øvrig skal det være et aktivt samspill med veiledere.

Innen tre uker etter at oppgaveteksten er utlevert, skal det leveres en forstudierapport som skal inneholde følgende:

- En analyse av oppgavens problemstillinger.
- En beskrivelse av de arbeidsoppgaver som skal gjennomføres for løsning av oppgaven. Denne beskrivelsen skal kunne ut i en klar definisjon av arbeidsoppgavens innhold og omfang.
- En tidsplan for fremdriften av prosjektet. Planen skal utformes som et Gantt-skjema med angivelse av de enkelte arbeidsoppgavens terminer, samt med angivelse av milepæler i arbeidet.

Forstudierapporten er en del av oppgavebesvarelsen og skal innarbeides i denne. Det samme skal senere fremdrifts- og avviksrappporter. Ved bedømmelsen av arbeidet legges det vekt på at gjennomføringen er godt dokumentert.

Besvarelsen redigeres mest mulig som en forskningsrapport med et sammendrag både på norsk og engelsk, konklusjon, litteraturliste, innholdsfortegnelse etc. Ved utarbeidelsen av teksten skal kandidaten legge vekt på å gjøre teksten oversiktlig og velskrevet. Med henblikk på lesning av besvarelsen er det viktig at de nødvendige henvisninger for korresponderende steder i tekst, tabeller og figurer anføres på begge steder. Ved bedømmelsen legges det stor vekt på at resultatene er grundig bearbeidet, at de oppstilles tabellarisk og/eller grafisk på en oversiktlig måte og diskuteres utførlig.

Materiell som er utviklet i forbindelse med oppgaven, så som programvare eller fysisk utstyr er en del av besvarelsen. Dokumentasjon for korrekt bruk av dette skal så langt som mulig også vedlegges besvarelsen.

Eventuelle reiseutgifter, kopierings- og telefonutgifter må bære av studenten selv med mindre andre avtaler foreligger.

Hvis kandidaten under arbeidet med oppgaven støter på vanskeligheter, som ikke var forutsett ved oppgavens utforming og som eventuelt vil kunne kreve endringer i eller utelatelse av enkelte spørsmål fra oppgaven, skal dette straks tas opp med instituttet.

**Oppgaveteksten skal vedlegges besvarelsen og plasseres umiddelbart etter tittelsiden.**

Innleveringsfrist: 10. juni 2014.

Besvarelsen skal innleveres i 1 elektronisk eksemplar (pdf-format) og 2 eksemplar (innbundet), ref. rutinebeskrivelse i DAIM. Det vises til <http://www.ntnu.no/ivt/master-siv-ing> for ytterligere informasjon om DAIM, uttak, kontrakt, gjennomføring og innlevering.

Ansvarlig faglærer / hovedveileder:

Professor Olav Egeland

E-post: [olav.egeland@ntnu.no](mailto:olav.egeland@ntnu.no)

Telefon: 73 59 71 12

**INSTITUTT FOR PRODUKSJONS-  
OG KVALITETSTEKNIKK**



Per Schjølberg

førsteamanuensis/instituttleder



---

Olav Egeland  
Ansvarlig faglærer

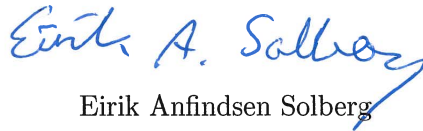


## Preface

This is the concluding Master's thesis in the study program Subsea Technology at NTNU. The work was carried out between January and June 2014.

The interest for the robotics field started with the course "Robotics" lectured by professor Olav Egeland in the spring of 2013. The opportunity to implement computer vision on the brand new robot lab at "Department of Production and Quality Engineering" was very tempting and resulted in the topic "Vision Based Robotic Control" for this final Master's thesis.

Trondheim, 10-06-2014



Eirik Anfindsen Solberg





## Acknowledgment

I would like to give a special thank to PhD candidate Lars Tingelstad for his cooperation and help during the project. His involvement in the work has been very helpful in selecting the right tools and software for the experiment. He also created the framework for the interface between ROS and the KUKA controller which is an essential part of the setup

I would also like to thank professor Olav Egeland for his valuable inputs, supervising and always having the office open for students with questions. He is also responsible for acquisition of the new robotic lab at *Departement of Production and Quality Engineering* and have together with Lars Tingelstad worked hard to get this in order. I am very grateful for the opportunity to work on such an interesting topic and be able to implement the system on actual robots.

The workshop employees at the lab have done a very good job creating the base for the two robots, and mounted them in place. Thank you for this important contribution.

The fellow students in the office and the Subsea Technology program have become my good friends, and i would like to thank them for two great years together.

EAS



## Summary

The main objective of this project was to create a visual system for object tracking, and implement this on the new robotic lab at the *Department of Production and Quality Engineering*.

There are used two identical KUKA Agilus manipulators with six rotating axes. The robot kinematics with the corresponding Denavit-Hartenberg representation are presented. The position based visual servoing method, used for robotic control, is shown with control loop and relations between the different coordinate bases and frames in the system.

Computer vision is a large part of this thesis. The different camera parameters and how to obtain them are explained. The results show the importance of accurate camera calibration, and that one should avoid the temptation to leave out some parameters from the equation. This can cause large errors in the measurements.

The SIFT object detection method is explained, and the performance is compared with another method named SURF. The tests show that while SURF is faster than SIFT, it is outperformed when it comes to the robustness of the algorithm. SIFT was therefore chosen for implementation at the lab.

With the object detected, the manipulators movement needs to be calculated in order to position the camera in the desired position, which is  $300mm$  perpendicular to the object. The algorithm calculates the rotational and translational offset between the current camera position and the desired camera pose. A proportional regulator is then applied to calculate the next small step on the desired trajectory for the Agilus manipulator.

The practical setup of the robot cell is explained with each step needed in order to have a working vision system. The information flow in the system can be chaotic, and a graphic representation is therefore developed and show all steps from image capturing, to robotic movement, and plotting of the trajectories.

Results are presented as plots for both distance calculations and the actual movement of the manipulator. The visual system tracks and follows the object successfully. There are however some issues with variations in the output from the object detection algorithm. This cause variations in the signal used as reference for the robot. A filter was able to reduce these variations, but not eliminate them. Possible solutions are presented and are believed to improve the speed and accuracy of the system if further investigated.



## Sammendrag

Hovedmålet ved denne oppgaven var å lage et system for objekt-deteksjon ved hjelp av datasyn. Dette skulle så implementeres i den nye robotcellen ved *Institutt for Produksjons- og Kvalitetsteknikk*.

Det er brukt to identiske KUKA Agilus robotmanipulatorer, hver med seks roterende ledd. Robotkinematikken med de tilhørende Denavit-Hartenberg-parametrene er presentert. Den posisjonsbaserte metoden for visuell robotstyring er anvendt og presentert med reguleringsløyfe og forholdet mellom de aktuelle koordinatsystemene.

Datasyn er en stor del av denne oppgaven. De forskjellige kameraparametrene og hvordan disse skal innhentes er forklart. Resultatene viser hvor viktig det er med presis kamerakalibrering, og at man skal være svært forsiktig med å utelate enkelte parametre fra likningen. Dette kan føre til store avvik.

To ulike objekt-deteksjonsmetoder, SIFT og SURF, er testet og ytelsen sammenliknet. Resultatene viser at mens SURF er vesentlig raskere enn SIFT, er den ikke like fleksibel i forhold til forstyrrelser og endringer. SIFT ble derfor valgt for implementering på robotene.

Etter en vellykket objekt-detektering kalkuleres koordinatene som forteller hvor manipulatoren må posisjonere seg for å komme i den ønskede posisjonen, som er en distanse på  $300\text{mm}$  vinkelrett på objektet. Algoritmen kalkulerer posisjons- og rotasjonsavvik mellom nåværende og ønsket kameraposisjon. En P-regulator er benyttet for å kalkulere det neste steget av den ønskede bevegelsen for roboten.

Oppsettet av robotcellen er presentert, og hvert element nødvendig for å oppnå et fungerende system er forklart. Informasjonsflyten i dette systemet kan være uoversiktlig, og en grafisk fremstilling av denne er derfor utformet.

Resultatene blir presentert med grafer som sammenlikner den kalkulerte posisjonen med de faktiske bevegelsene til roboten. Oppsettet fungerer, og manipulatoren klarer å følge etter det aktuelle objektet ved å anvende datasyn. Det er likevel rom for enkelte forbedringer. Algoritmen for objekt-deteksjon er ikke helt stabil, og objektets kalkulerte posisjon varierer derfor med noen millimeter. Ved hjelp av et filter ble disse variasjonene redusert, men ikke fjernet helt. Mulige løsninger er presentert og vil sannsynligvis forbedre både hastighet og presisjon hvis de blir implementert i systemet.



## Glossary and Acronyms

**cv\_bridge** ROS package used for transforming of an image file between ROS and OpenCV format

**Denavit-Hartenberg parameters** Standard convention for representation of robotic axes and links

**fps** Frames per second. The rate at which a camera captures image frames

**Kernel** A matrix which is used in image filtering. This can be sharpening, blurring, edge detection etc.

**Keypoint** An important point (pixel) in a picture used for object detection

**Keypoint descriptors** Neighbouring pixels to a keypoint used to recognize this point

**KR C4 controller** The standard KUKA controller used for control of Agilus

**Node** A program that subscribes and/or publishes from/to a ROS topic

**OpenCV** Open Source Computer Vision Library. Computer vision library with built in algorithms and interfaces for C++, C, Python, Java and Matlab.

**PBVS** Position Based Visual Servoing

**ROS** The Robot Operating System

**ROS package** A collection of code installed as one package on the computer

**ROV** Remotely Operated Vehicle. Unmanned submarine used for work at subsea locations

**RSI** Robot Sensor Interface used for external sensor input by KUKA robots

**SIFT** Scale Invariant Feature Transform. Object detection method

**SIL** Safety Integrity Level

**SURF** Speeded Up Robust Features. Object detection method

**TCP** Tool Center Point

**Topic** An address that ROS nodes can subscribe or publish to in order to receive or distribute information

**UDP** User Datagram Protocol

**X-mas tree** *Assembly of equipment, including tubing-head adapters, valves, tees, crosses, top connectors and chokes attached to the uppermost connection of the tubing head, used to control well production [1]*

**XML** Extensible Markup Language



# Contents

Preface . . . . .	i
Acknowledgment . . . . .	iii
Summary . . . . .	v
Sammendrag . . . . .	vii
Glossary and Acronyms . . . . .	ix
List of Figures . . . . .	xiii
List of Tables . . . . .	xv
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Objectives . . . . .	2
1.3 Limitations . . . . .	2
1.4 Approach . . . . .	2
1.5 Structure of the Report . . . . .	4
<b>2 Robot Kinematics</b>	<b>7</b>
2.1 Denavit-Hartenberg Parameters . . . . .	7
2.2 Position Based Visual Servoing . . . . .	9
<b>3 Computer Vision</b>	<b>13</b>
3.1 Camera Parameters . . . . .	13
3.1.1 Intrinsic Parameters . . . . .	13
3.1.2 Extrinsic Parameters . . . . .	15
3.2 Object Detection . . . . .	17
3.2.1 SIFT . . . . .	18
3.2.2 SURF . . . . .	20
3.2.3 Algorithm Performance . . . . .	21
3.3 Coordinate Calculation . . . . .	23
3.3.1 The Central-Projection Model . . . . .	23
3.3.2 Distance Calculation . . . . .	24

3.3.3	Camera Offset Calculation . . . . .	26
<b>4</b>	<b>Practical Setup</b>	<b>31</b>
4.1	Robot Cell . . . . .	31
4.2	Information Flow . . . . .	32
4.2.1	Image Capturing . . . . .	33
4.2.2	Object Detection and Calculations . . . . .	33
4.2.3	UDP Connection . . . . .	34
4.2.4	Robot Sensor Interface . . . . .	34
4.2.5	World-pose to TCP-pose . . . . .	35
<b>5</b>	<b>Results</b>	<b>37</b>
5.1	Stepper-motor . . . . .	37
5.2	KUKA Agilus . . . . .	39
5.2.1	Regulator Adjustment . . . . .	39
5.2.2	System Performance . . . . .	40
5.2.3	Reference Filter . . . . .	43
5.2.4	Position Lock . . . . .	46
5.2.5	SURF Test . . . . .	47
5.2.6	Maximum Correction . . . . .	48
5.2.7	Video . . . . .	48
<b>6</b>	<b>Concluding Remarks</b>	<b>49</b>
6.1	Discussion . . . . .	49
6.2	Conclusion . . . . .	50
6.3	Recommendations for Further Work . . . . .	51
	<b>References</b>	<b>52</b>
<b>A</b>	<b>Equipment and Software</b>	<b>55</b>
<b>B</b>	<b>Source Code</b>	<b>59</b>
<b>C</b>	<b>Digital Appendix</b>	<b>69</b>
<b>D</b>	<b>Pre-Study Report</b>	<b>71</b>

# List of Figures

2.1	Joint and axis composition of KUKA Agilus manipulator . . . . .	9
2.2	Control loop of the PBVS system . . . . .	10
2.3	World, end-effector, camera and object frame relations . . . . .	10
2.4	Relative pose network for PBVS . . . . .	11
3.1	Camera matrix for Prosilica GC650 . . . . .	14
3.2	Object position when evaluating the importance of distortion calibration . . . . .	15
3.3	Transformation matrix from camera to TCP frame . . . . .	16
3.4	Transformation matrix from end-effector to TCP frame . . . . .	16
3.5	TCP, end-effector and camera frame relation . . . . .	17
3.6	DoG Pyramids extracted from Gaussian Pyramids of different scales . . . . .	19
3.7	Comparison of pixels in the DoG pyramids to select keypoints . . . . .	20
3.8	Keypoint descriptor extracted from image gradients . . . . .	20
3.9	Object highlighted in the scene image after applying SURF . . . . .	21
3.10	Framerate comparison of SIFT and SURF . . . . .	22
3.11	The influence of varying illumination to SIFT and SURF . . . . .	23
3.12	The central-projection model . . . . .	24
3.13	Easy calculation of distance to object . . . . .	25
3.14	Calculation of camera offset when object is rotated around the Y-axis . . . . .	28
4.1	Graphic model of the KUKA Agilus robot cell . . . . .	32
4.2	Information flow of the visual servoing system . . . . .	33
5.1	Setup for 1D tracking with Arduino Uno and stepper-motor . . . . .	38
5.2	Stepper-motor responsiveness . . . . .	38
5.3	Manipulator position feedback with $K_p = K_c = 0.01$ . . . . .	40
5.4	Manipulator response on a three dimensional position step . . . . .	42

5.5	Manual measurement of distance to object after visual servoing translation . . . . .	42
5.6	Manipulator response to a rotational step around axis X and Y .	43
5.7	Unfiltered XYZ reference points from sensor system . . . . .	44
5.8	Filtered XYZ reference points from sensor system . . . . .	44
5.9	XYZ step-test on unfiltered system . . . . .	45
5.10	XYZ step-test on filtered system . . . . .	45
5.11	XYZ step-test on filtered system with position lock . . . . .	47
5.12	XYZ step-test on filtered system with position lock. SURF used for object detection . . . . .	48
A.1	The installed control cabinet for the Agilus robot cell . . . . .	55
A.2	KUKA Agilus KR 6 R900 sixx dimensions . . . . .	56
A.3	Prosilica GC650 used for sensor input . . . . .	57
A.4	The Arduino Uno board . . . . .	57

# List of Tables

- 2.1 Denavit-Hartenberg parameters for KUKA Agilus manipulators . . . . . 8
- 3.1 The influence of image distortion coefficients . . . . . 15
- 5.1 Ziegler-Nichols table for adjustment of regulators . . . . . 39



# Chapter 1

## Introduction

### 1.1 Background

The subsea industry gets more advanced by the minute, and new oil and gas fields are developed in remote locations around the globe. Increasing water depth makes it impossible for divers to perform work at the seabed and remotely operated equipment takes over. ROVs are one example. They are piloted by humans sitting on the surface. Especially during physical operations like opening or closing a valve on a X-mas tree this can be tricky. Sea current variations and movements complicates the work. Since the operator controls the robotic arm movement by looking at a computer screen, his depth vision is lost. If one can use computer vision to detect and measure the distance between the robotic arm and the valve handle, an automatic approach path for the arm can be programmed. This will speed up the process and the challenge with depth vision overcome. The fact that ROVs are already equipped with cameras, makes the implementation on existing equipment easier.

### 1.2 Objectives

The main objectives of this Master's thesis has been as follows:

1. Set up the Agilus robot cell at Valgrinda and test simple movement and control.
2. Create a graphic model of the robot cell in a computer simulation system and test simple movements for the robots in this program.
3. Create a system for computer vision which can be used for sensor control of the robots.
4. Prepare a demo where one robot holds a moving target and the other tracks and follows this target using computer vision. Evaluate the system performance.

### 1.3 Limitations

The practical work has been performed in the lab at the *Department Of Production and Quality Engineering, NTNU*. This is a dry facility and any impact from the marine environment like visibility issues, current and pressure are ignored. The KUKA Agilus manipulators have electrical servos while subsea manipulators typically use hydraulic.

The implemented object detection is for planar objects. Identification of 3D objects will require a different object detection method than those used in this thesis.

### 1.4 Approach

The objectives in this project has been mostly of practical nature with robot cell set-up, programming and testing. The main focus has therefore been on these parts. Literature and articles has been studied in order to solve the challenge at hand.

In connection with the purchase of the six new KUKA robots at Valgrinda, Lars Tingelstad and the undersigned participated in the KUKA college in Göteborg on two different occasions. The basic course in November 2013 and advanced in January 2014. This was an important introduction to the KUKA control system and manual trajectory planning via the KR C4 controller.



### **Objective 1**

The actual mounting of the robots was performed by the workshop employees at Valgrinda.

The safety system used is called ProfiSafe and runs between the KR C4 controller and a Siemens safety PLC on the PROFINET interface. Hard-wiring, control cabinet set-up and ProfiSafe programming was performed in order to get the robots ready for use. Easy robot trajectories were programmed to test the system.

### **Objective 2**

A graphical model has been created in "VisualComponents". Mounting height and distance between the manipulators are the same as in the robot cell at *Department of Production and Quality, NTNU*. Fences are not yet installed due to problems with the delivery.

Robotic movements in VisualComponents are programmed in almost the same way as on the KR C4 controller. This makes the tool great for testing of robot positions and trajectories before implementation on the robot.

### **Objective 3**

A Prosilica GC650 camera has been used for sensor input. This is a grayscale camera with a resolution of 659 x 493 pixels. The choice of camera was mainly done upon availability, but it proved effective and well suited for the task.

Development and implementation of the object detection and offset calculation has been the largest and most demanding part of this project. C++ is the programming language used in the development, with the additional OpenCV library.

Two object detection algorithms has been tested and the performance compared.

### **Objective 4**

The visual servoing system is communicating over the KUKA RSI interface. Two Agilus robots face each other with one holding an object and the other a camera tracking it. The manipulator holding the object follows a predetermined path. The vision system connected to the second manipulator calculates the

real-time offset, and moves the robot accordingly. System performance has been evaluated in chapter 5.

### Literature

Textbooks on the robotics, computer vision and programming topics has been extensively used during the project. The main books are listed below:

- *Robotics: Modelling, Planning and Control* of Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani and Giuseppe Oriolo
- *Robotics, Vision and Control* of Peter Corke
- *Practical OpenCV* of Samarth Brahmhbhatt
- *Absolute C++* of Walter Savitch

In addition the KUKA documentation for RSI, KR C4 controller, ProfiNET and SafeOperation has been used. Relevant scientific articles was also studied, especially on the object detection subject.

### 1.5 Structure of the Report

The report focus on the experiment setup and explains how the robot kinematics, object detection, transformation calculation and information flow is working in this exact experiment. Many features will also work by themself on other systems, but the total arrangement is customized for this equipment.

- Chapter 2 presents the robot kinematics of the Agilus manipulator and the relationships between different base frames used.
- Chapter 3 describes the computer vision with camera parameters, object detection and transformation calculations.
- Chapter 4 shows the practical setup of the robot cell and explains the steps needed in order to have a working visual servoing system.
- Chapter 5 presents the results from the experiments and possible methods for improving the system performance.
- Chapter 6 summarize the work with a discussion, conclusion and recommendations for future improvement and implementation.
- The most important hardware and software used is presented in the appendix. The "object detection and coordinate calculation" source code

can be studied here as well. The Arduino Uno code is also located here.

- The digital appendix includes all source code written for this project. A video of the working visual system is also found here.



## Chapter 2

# Robot Kinematics

This chapter presents the forward kinematics of the KUKA Agilus robots used in this project. The inverse kinematics was unnecessary because the commands sent to the robot was cartesian, not joint corrections. Denavit-Hartenberg convention has been applied in the representation [2]. Transformation matrices are also extensively used. They consist of a 3x3 rotation matrix,  $\mathbf{R}$ , and the 4x1 translation matrix,  $\mathbf{p}$ .

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad (2.1)$$

The resulting 4x4 transformation matrix will then take the form

$$\mathbf{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

### 2.1 Denavit-Hartenberg Parameters

Denavit-Hartenberg (DH) parameters for the KUKA Agilus robots have been created and is shown in Table 2.1. They are based on the rotation direction of the axes in Figure 2.1. KUKA robots start with the Z-axis pointing downwards. This is not a problem, but it results in a somewhat strange DH-table where all  $d_i$  translations are negative.

Link	$\theta_i$ [rad]	$d_i$ [mm]	$a_{i-1}$ [mm]	$\alpha_{i-1}$ [rad]
1	$\theta_1^*$	-400	-25	$-\frac{\pi}{2}$
2	$\theta_2^*$	0	315	0
3	$\theta_3^*$	0	35	$\frac{\pi}{2}$
4	$\theta_4^*$	-365	0	$-\frac{\pi}{2}$
5	$\theta_5^*$	0	0	$\frac{\pi}{2}$
6	$\theta_6^*$	-80	0	$\pi$

Table 2.1: Denavit-Hartenberg parameters for KUKA Agilus manipulators

The DH-parameters can be used to calculate the manipulators forward kinematics as seen in equation 2.3. The joint variable  $q$  is denoted as  $\theta$  in a revolute joint and  $d$  in a prismatic joint. This calculation is done for every link, and the transformation matrix from the manipulator base to end-effector,  $\mathbf{T}_E^W$ , is obtained by applying equation 2.4. The result is a 4x4 matrix that describes position and rotation of the end-effector as a function of all six joint variables,  $q_{1-6}$ . [2]

$$\mathbf{A}_i^{i-1}(q_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

$$\mathbf{T}_E^W = T_6^0 = A_1^0 A_2^1 A_3^2 A_4^3 A_5^4 A_6^5 \quad (2.4)$$

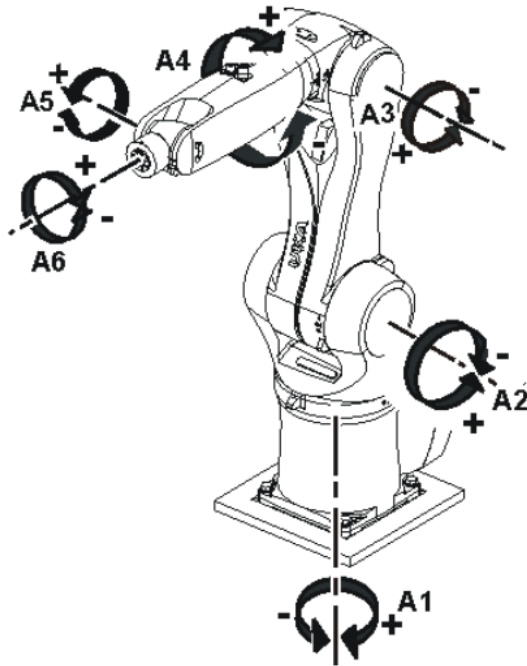


Figure 2.1: Joint and axis composition of the KUKA Agilus manipulator [3]

## 2.2 Position Based Visual Servoing

The method *Position Based Visual Servoing* (PBVS) has been applied in this project. The object pose in relation to the camera frame is used to control the manipulator movements. The control loop is seen in Figure 2.2 and the relationship between relative poses is shown in Figure 2.4. The different frame locations in Figure 2.3. The reference is the transformation matrix  $T_O^{C*}$  which describes the desired pose of the camera in relation to the object. It is compared to the real-time pose  $T_O^C$ , estimated by the visual system, to determine the required motion  $\Delta T_C$ . In the experiment described in section 5.2, the operation of the *joint controller* is executed by the *KUKA KR C4 Controller*. *Feature extraction*, *Pose estimation* and *PBVS control*, is calculated in ROS with C++ on a Linux computer connected over Ethernet.

The "eye-in-hand" configuration with the camera attached to the robot manipulator has been applied.

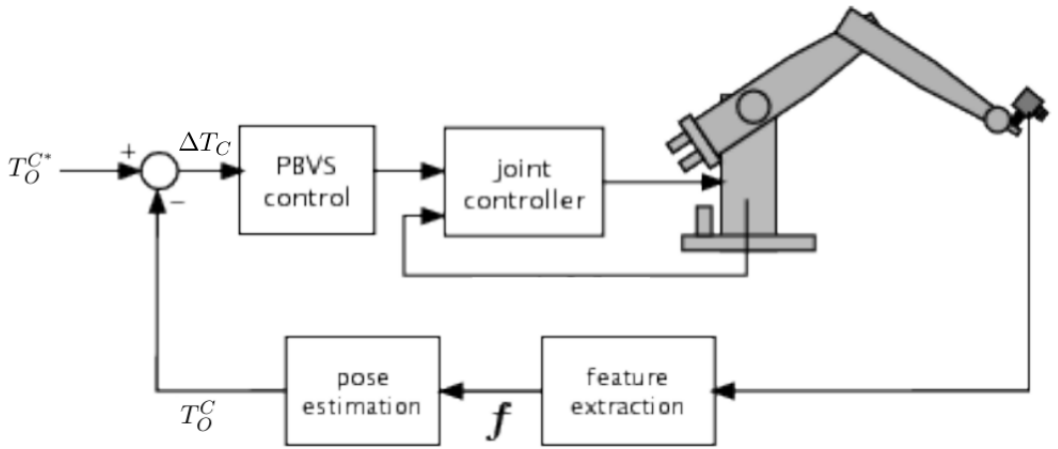


Figure 2.2: Control loop of the PBVS system [4]

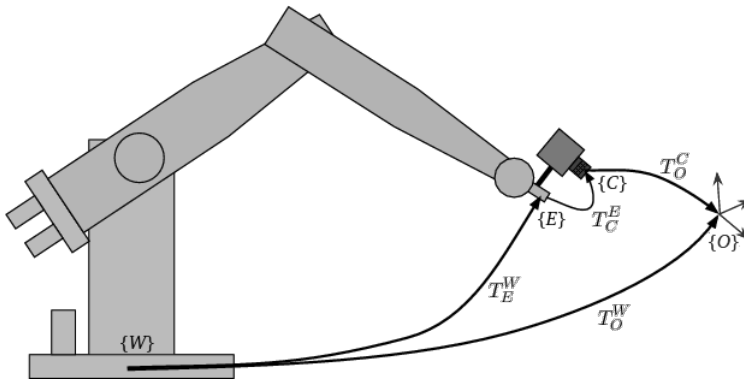


Figure 2.3: World, end-effector, camera and object frame with associated transformation matrices of the "eye-in-hand" configuration applied [4]



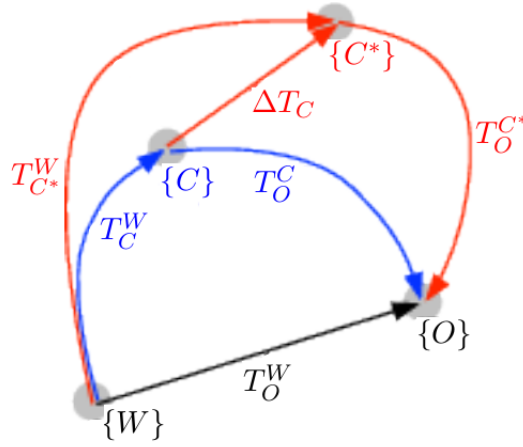


Figure 2.4: Relative pose network for PBVS [4]



## Chapter 3

# Computer Vision

The visual sense is important in almost any activity or work being performed by humans. Decisions are made on the background of information received by visual input. In order for robots to perform humanoid tasks and take its own decisions based on the surrounding environment, it is vital that it can use vision to make the necessary adjustments.

Today there is a wide range of relative easy-to-use cameras that can be used as visual receivers. The trick for the robot is to use this information of bits and bytes to make the right decisions [5]. Control based on feedback of visual measurements is termed *visual servoing* [2].

The computer vision methods applied are mainly based on Peter Corke's *Robotics, Vision and Control* [4] and the features of OpenCV [6].

### 3.1 Camera Parameters

Camera parameters defines the location, orientation and the relation between observed objects and their position on the cameras image plane.

#### 3.1.1 Intrinsic Parameters

The intrinsic parameters describes the physical size and relation between some important camera specifications. These are focal length ( $f$ ), pixel size ( $s$ ), and the principal point ( $c$ ). In addition real cameras have some distortion due to the lens curvature. The pixel size is usually found in the camera documentation, while focal length, principal point and distortion is found by performing a camera calibration.

$$\mathbf{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 818 & 0 & 361 \\ 0 & 821 & 225 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 3.1: Camera matrix for Prosilica GC650. Values are given in pixels

Calibration for the Prosilica GC650 was done with a ROS package named *image\_pipeline* [7] which takes multiple pictures of a chessboard in various positions and orientations. It then calculates the focal lengths  $(f_x, f_y)$ , focal center  $(c_x, c_y)$  and the distortion coefficients. The focal length and focal center forms the camera matrix and is seen in Figure 3.1.

The distortion parameters was found to be important in order to get accurate measurements. There are five distortion parameters calculated for use in OpenCV. Three for radial and two for tangential distortion. *Radial distortion is a deformation of the image along the direction from a point called the center of distortion to the considered image point, and tangential distortion is a deformation perpendicular to this direction. The center of distortion is invariant under both transformations [8].*

Distortion parameters are presented as a matrix of five elements. Radial factors are denoted with  $k$  and tangential  $p$ . Distortion matrix for the Prosilica camera was found to be:

$$Distortion_{coefficients} = [k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3] = \begin{bmatrix} -0.187112 \\ 0.046056 \\ -0.002983 \\ 0.001150 \\ 0.000000 \end{bmatrix}^T \quad (3.1)$$

After experiencing some distance deviation, a position measurement was performed both with and without distortion parameters in order to evaluate the importance of this calibration. The object was located at the edge of the cameras field of view where the image distortion is largest. Figure 3.2 show the object location. The position measurements for both calibrated and uncalibrated setup is shown in Table 3.1. It is obvious that a measurement error (only due to distortion) of almost 50% in the X-direction can be problematic. This is further discussed in section 5.2.2.



Figure 3.2: Object position when evaluating the importance of distortion calibration

Direction	Uncalibrated	Calibrated
$X$	$65mm$	$45mm$
$Y$	$180mm$	$135mm$
$Z$	$110mm$	$95mm$

Table 3.1: The influence of image distortion coefficients. Distance to object measured with camera calibrated and uncalibrated for distortion. Values are taken from the TCP base frame in Figure 3.5

### 3.1.2 Extrinsic Parameters

The extrinsic parameters explains how the camera is oriented and located in relation to a known coordinate system. In this project, the *Tool Center Point* (TCP) has been used. This is the fixed location of the camera in relation to the Agilus end-effector. It is also the origin of the TCP coordinate system seen in Figure 3.5. The transformation  $\mathbf{T}_{TCP}^E$  is extracted from the KUKA controller after calibration of the camera tool frame.  $\mathbf{T}_{TCP}^C$  is the transformation from the cameras coordinate system to TCP.

$$\mathbf{T}_{TCP}^C = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 15 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.3: Transformation matrix from camera to TCP frame. Translations in [mm]

$$\mathbf{T}_{TCP}^E = \begin{bmatrix} -0.3890 & 0.9209 & -0.0255 & -38.2 \\ -0.9212 & -0.3889 & 0.0083 & -94.7 \\ -0.0023 & 0.0267 & 0.9996 & 163.8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3.4: Transformation matrix from end-effector to TCP frame. Translations in [mm]

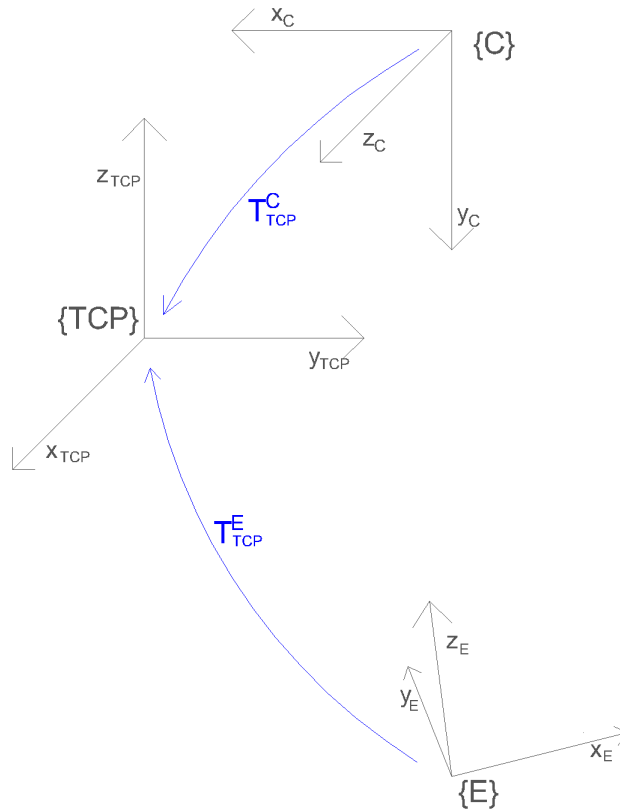


Figure 3.5: Relation between intrinsic, TCP and end-effector coordinate system

## 3.2 Object Detection

Object detection is all about finding an object or area of interest in a field of view. The fastest way is to use a color-based algorithm. This method looks for predefined color or grayscale intensities that will identify the object in an image. The speed of this algorithm makes it very good for use in a controlled environment where disturbances of objects with similar color is not present. In a chaotic environment like an X-mas tree on the sea floor surrounded by marine life, multiple objects and varying light conditions, it will be useless.

Another method is to use *keypoint* detection. The idea here is to locate important points from a test picture of the object to be recognized. This is done by storing information about the neighbouring pixels called *keypoint descriptors*, and match these with similar points in the incoming image stream from the camera. This is a more comprehensive method than the color detection discussed earlier, but it is less influenced by disturbances in the environment or object itself.

Because of the robustness and the fact that detection with extreme speed was unnecessary in this project, keypoint detection has been used for object recognition and tracking. There are multiple methods for keypoint detection. Two of the most common have been tested and is presented in the following sections. Since these algorithms detect a number of keypoints, they do not need to see the whole object in order to make a match.

### 3.2.1 SIFT

SIFT is the most famous and used keypoint detection and description algorithm today [5]. This approach was introduced by professor David G. Lowe in the paper at the *University of British Columbia*, Vancouver in 2004. It is both scale and rotation invariant, which means that it can recognize objects with different size (often due to varying distance) and rotation than that of the test picture. It is also to a certain degree invariant to change in illumination and some other disturbing elements [9].

In order to extract keypoints that are scale invariant, a Gaussian kernel which blurs the image, is ran on samples with different scales multiple times, producing a Gaussian Pyramid of each scale. Then samples of the Gaussian Pyramid is subtracted from each other, and a Difference of Gaussian (DoG) Pyramid is created. This is illustrated in Figure 3.6



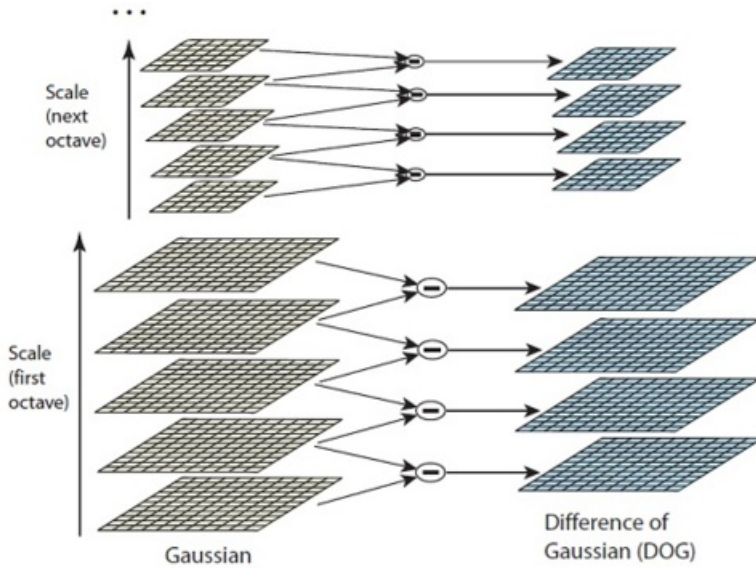


Figure 3.6: DoG Pyramids extracted from Gaussian Pyramids of different scales [9]

A keypoint is selected if it has higher or lower value than all of its 26 neighbours in the DoG pyramid. This process is seen in Figure 3.7. A control algorithm that checks if these keypoints have sufficient contrast and are not part of an edge is then applied.

The image orientation is now assigned by calculating the gradient of all samples in a square region around the keypoint. These are grouped together  $4 \times 4$  where the sum of all gradients are found. This is known as the *keypoint descriptor* and is seen in 3.8. This set of vectors is now normalized in order to obtain illumination invariance [9].

After all keypoints from the image stream has been extracted, they are matched with the test image to find the corresponding locations. If enough keypoints are matched between the two samples, the next step is to calculate the object pose in relation to the camera frame. This is explained in section 3.3.

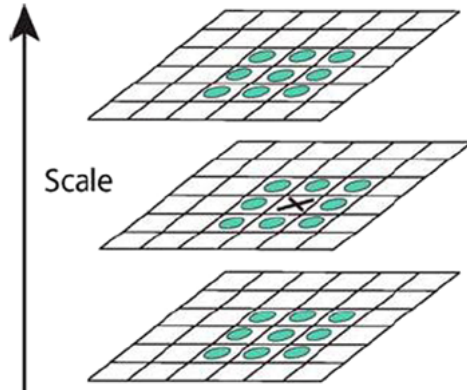


Figure 3.7: Comparison of pixels in the DoG pyramids to select keypoints [9]

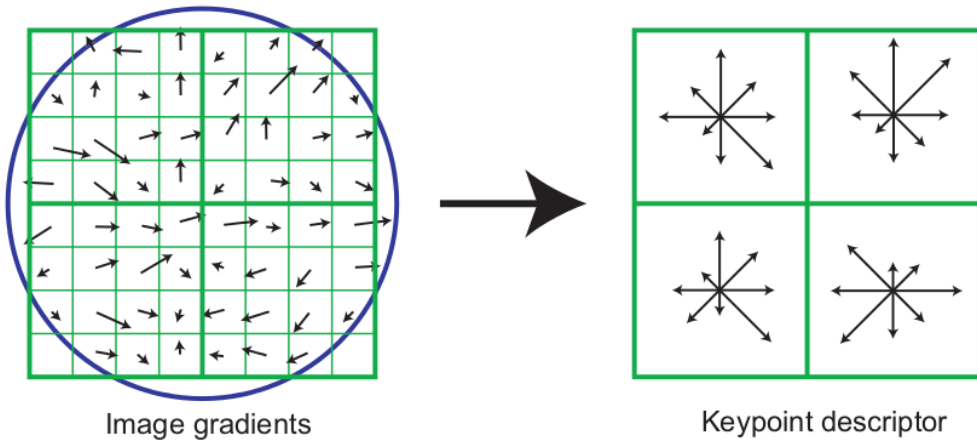


Figure 3.8: Keypoint descriptor extracted from image gradients [9]

### 3.2.2 SURF

SURF is another keypoint detector and descriptor, first described in the article *SURF: Speeded Up Robust Features* [10]. According to the same article it *approximates or even outperforms previously proposed schemes with respect to repeatability, distinctiveness, and robustness, yet can be computed and compared much faster.*

Figure 3.9 shows the SURF algorithm that recognizes the test image in the scenery. It is seen that this algorithm is both scale and rotation invariant

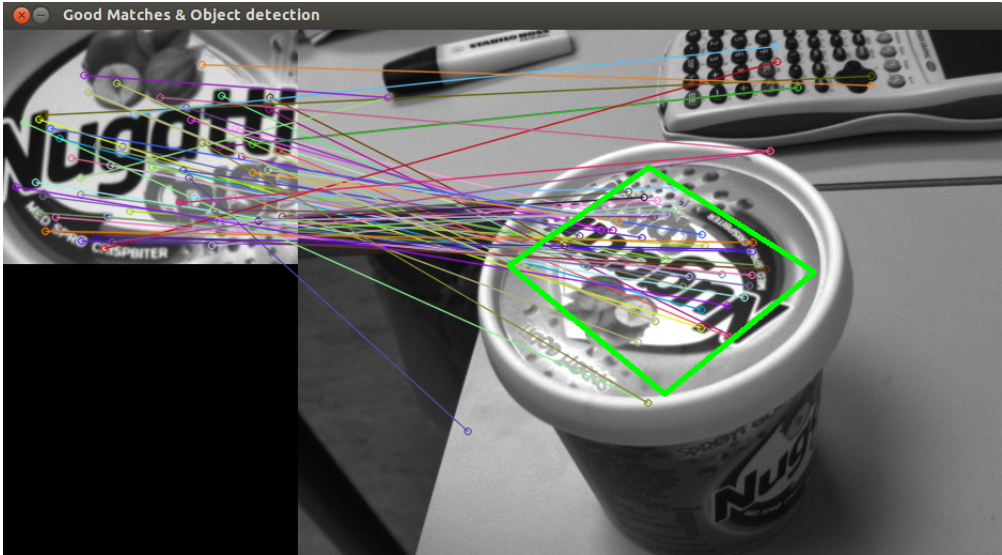


Figure 3.9: Object highlighted in the scene image after applying SURF

from this example. The lines of different colors indicates the identified and matched keypoints in each image. Not all keypoints match correctly, and keypoint descriptors are identified in the wrong location. This does not influence the result since the distance towards adjacent keypoints are also compared, and mismatches are sorted out.

### 3.2.3 Algorithm Performance

Both SIFT and SURF have been tested in order to find the best suited object detection algorithm for this project. The speed and robustness of the tracking was of special interest.

In Figure 3.10 the frequency of the SURF and SIFT algorithms is compared. SIFT runs with about four iterations per second. SURF runs significantly faster at between six and seven iterations per second. An improvement of over 50%.

When comparing the robustness of the algorithms it was observed that a given object with no rotation was trackable with SURF up to 40 cm, while SIFT recognized the object at a distance of 60 cm. Both methods performed fairly equal with regards to rotation of the object.

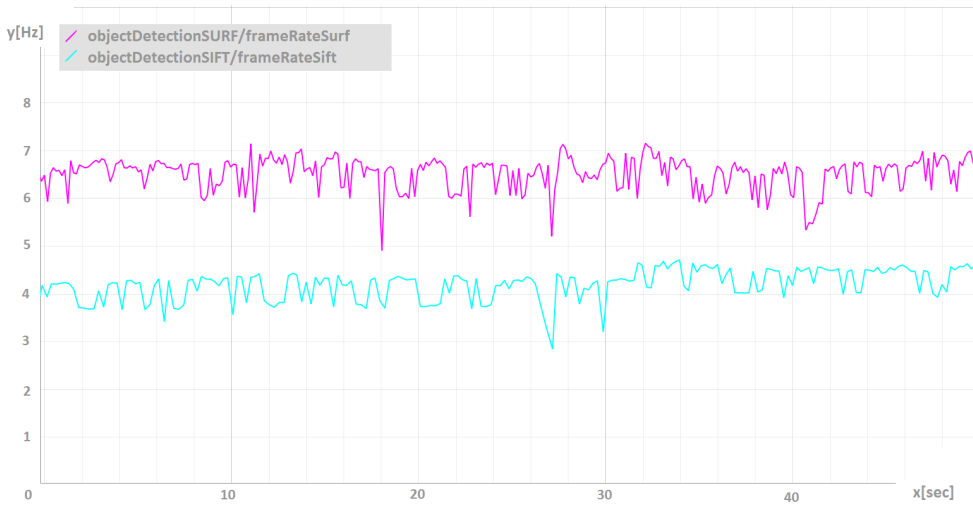


Figure 3.10: Framerate of SURF and SIFT

Varying illumination of an object is also of interest, especially when used in a subsea environment. In Figure 3.11 the behavior of the different algorithms is shown. Both algorithms is illumination invariant to a certain degree. This test shows which method that is most robust on objects with lower and higher illumination than the test picture. It is observed that both algorithms tracks well in normal conditions, although SURF show some larger disturbances. SIFT performs better when it comes to light variations as the plot clearly illustrates larger variations in the SURF measurements in both dimmed and extra illuminated objects.

These tests show that SURF is fast, but outperformed or matched by SIFT when it comes to reliability and robustness. For use in in-homogeneous areas like subsea, SIFT will probably be the best solution given that very high speed is not required. The SIFT algorithm was therefore used in the practical experiments in chapter 4.

It is important to remember that these are just brief tests of the algorithm performances. A more thorough investigation could influence the result.

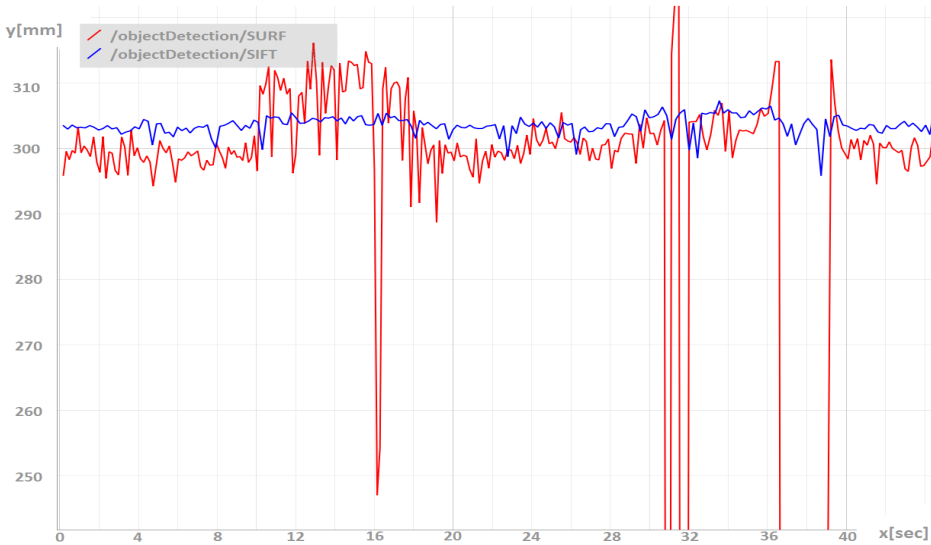


Figure 3.11: The influence of varying illumination to SIFT and SURF. SIFT in blue and SURF in red. The y-axis is distance to object and x-axis time. At approximately 10 seconds the light on the object is dimmed, and at 18 seconds it returns to normal. At 30 seconds the object is illuminated and at 40 seconds it return to normal conditions once again.

### 3.3 Coordinate Calculation

The coordinate calculation has been an essential part of this project. The object pose in relation to the camera is calculated in order to find the desired camera position. In order to estimate the location of an object, it is vital to know the camera parameters found in section 3.1.

#### 3.3.1 The Central-Projection Model

The central-projection model is a popular method for transforming a point in the 3D world to a point on the 2D image plane. In Figure 3.12 the point  $P$  is transferred to pixel values on the image plane with equation 3.2 [8].

$$x_c = f \frac{X}{Z}, y_c = f \frac{Y}{Z} \quad (3.2)$$

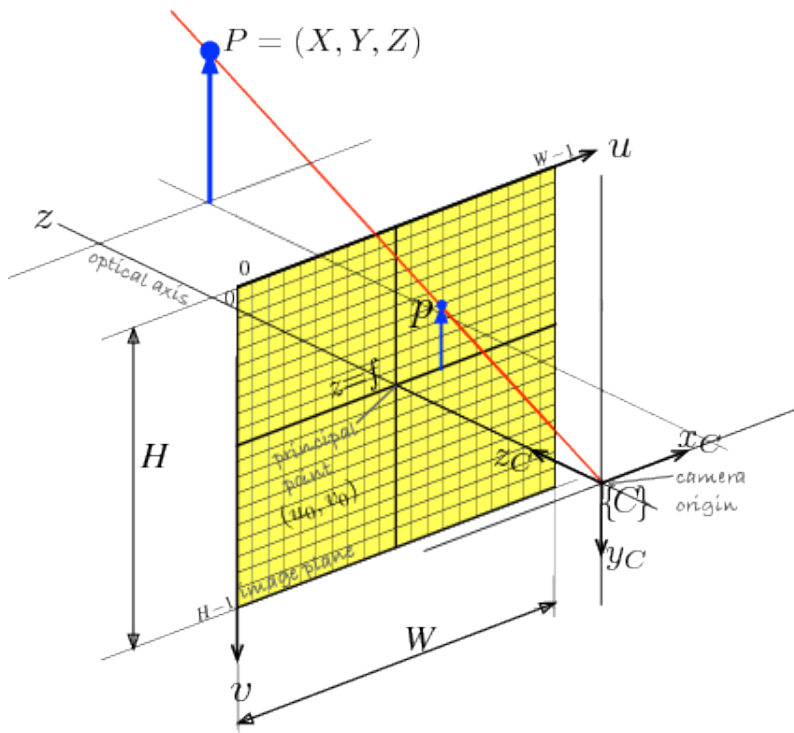


Figure 3.12: The central-projection model with the image plane in front of the camera origin [4]

### 3.3.2 Distance Calculation

The objective in this project was to determine the location and orientation of a recognized object and follow this with a robot manipulator. From 3.2 it is clear that the distance  $Z$  is needed in order to calculate  $X$  and  $Y$  coordinates of  $P$ . The distance can be calculated if information about the size of the observed object is known. The relation

$$Z = f \frac{W}{x_1} \quad (3.3)$$

can be applied, where  $f$  and  $x_1$  are in pixels.  $W$  and  $Z$  are in desired length units (see Figure 3.13).

Another method, is to take a test image at a known distance from the camera. This is compared to the captured image stream and the relation between these

makes it possible to calculate distance. Figure 3.13 shows this method applied on the x-axis. Distance  $Z_2$  is found in equation 3.5 after the object size is calculated in 3.4.

The experiment in section 5.2 use this method with a built in OpenCV function called *solvePnP*. This use the Levenberg-Marquardt algorithm [11] to estimate the object pose, and returns the rotation and translation vector in relation to the camera frame  $\{C\}$ .

The translation vector returned is in pixel size, and needs to be transformed into metric units. The Z-value is dependent on the focus length. If the object is located at the same distance as the object from the test image, the Z-value returned from *solvePNP* is equal to the camera focus length. The distance to the object is then calculated by equation 3.6.

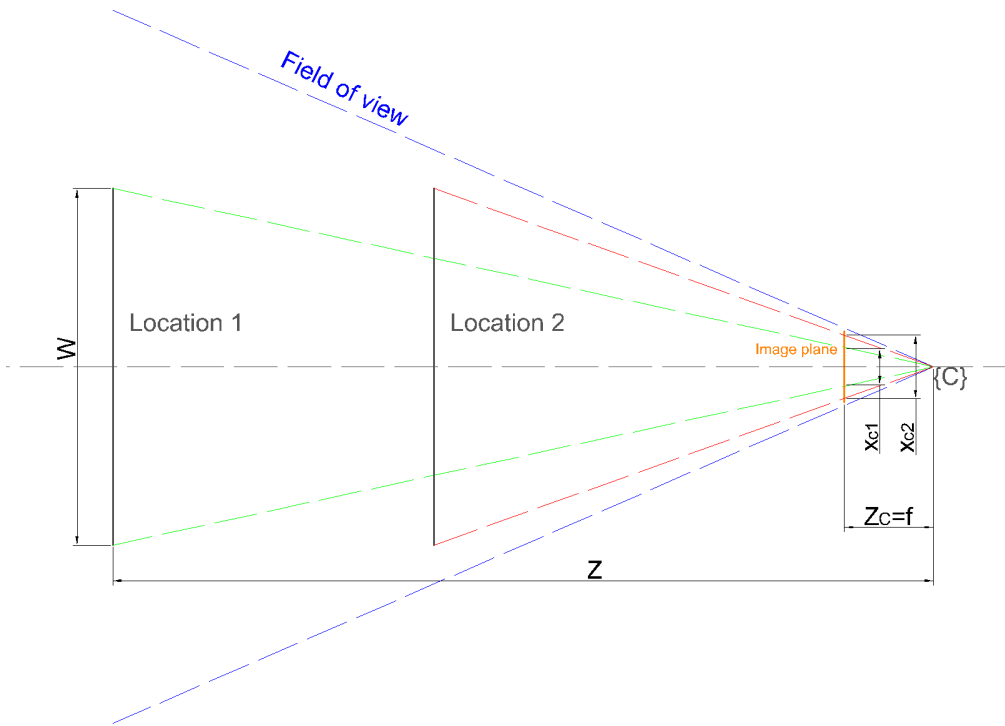


Figure 3.13: Easy calculation of distance to object

$$\frac{W}{Z_1} = \frac{X_1}{f} \Rightarrow W = Z_1 \frac{X_1}{f} \quad (3.4)$$

$$Z_2 = f \frac{L}{X_2} \quad (3.5)$$

$$Z_{mm} = \frac{Z_{px}}{f} Z_{test} \quad (3.6)$$

### 3.3.3 Camera Offset Calculation

The desired camera position,  $C^*$ , is located at a distance of 300 mm perpendicular to the object. The transformation offset,  $\Delta \mathbf{T}_C$ , consists of two independent relations,  $\Delta \mathbf{p}_{trans}$  and  $\Delta \mathbf{T}_{rot}$ .  $\Delta \mathbf{p}_{trans}$  relates to the physical offset between the object center and the camera frame.  $\Delta \mathbf{T}_{rot}$  is the necessary movement of the camera in order to stay perpendicular to the object, and is related to the object rotation. Figure 3.14 shows an overview of the positions with the rotation  $\beta$  around the  $Y$  axis.

Distance to object,  $Z$ , was found in section 3.3.2 and is used in the following calculations.

The position offset found by *solvePnP* needs to be converted to metric units. The relation between  $X$ ,  $Y$  and  $Z$  direction will be the same, and the metric offset of  $\Delta \mathbf{p}_{trans}$  is found in equation 3.7, 3.8 and 3.9.

$$\phi = \arctan\left(\frac{X_{px}}{Z_{px}}\right) \quad (3.7)$$

$$\Delta X_{trans} = Z * \tan(\phi)$$

$$\psi = \arctan\left(\frac{Y_{px}}{Z_{px}}\right) \quad (3.8)$$

$$\Delta Y_{trans} = Z * \tan(\psi)$$

$$\Delta Z_{trans} = Z - Z_{desired} \quad (3.9)$$



A known value is  $\beta$  which is extracted from the *solvePnP* function in OpenCV. The two angles  $\varphi$  is equal to  $\frac{\pi-|\beta|}{2}$  since they are part of the isosceles triangle with the two sides  $Z_{desired}$ .

The length of  $\Delta p_{rotY}$ , which is the translational offset due to rotation around the Y-axis, is found with the law of cosines.

$$\Delta p_{rotY}^2 = 2Z_{desired}^2(1 - \cos(\beta)) \quad (3.10)$$

The offset in  $X$  and  $Z$  direction is now found by equation 3.11 and 3.12. The Y-axis is pointing downwards, and a negative rotation will therefore give a positive offset in the X-direction. If the rotation is positive, the offset will be negative, and equation 3.11 has to be multiplied by (-1).

$$\begin{aligned} \Delta X_{rotY} &= \sin(\varphi)\Delta p_{rotY} \\ \Delta X_{rotY} &= \sin(\varphi)\sqrt{2Z_{desired}^2(1 - \cos(\beta))} \end{aligned} \quad (3.11)$$

$$\begin{aligned} \Delta Z_{rotY} &= \cos(\varphi)\Delta p_{rotY} \\ \Delta Z_{rotY} &= \cos(\varphi)\sqrt{2Z_{desired}^2(1 - \cos(\beta))} \end{aligned} \quad (3.12)$$

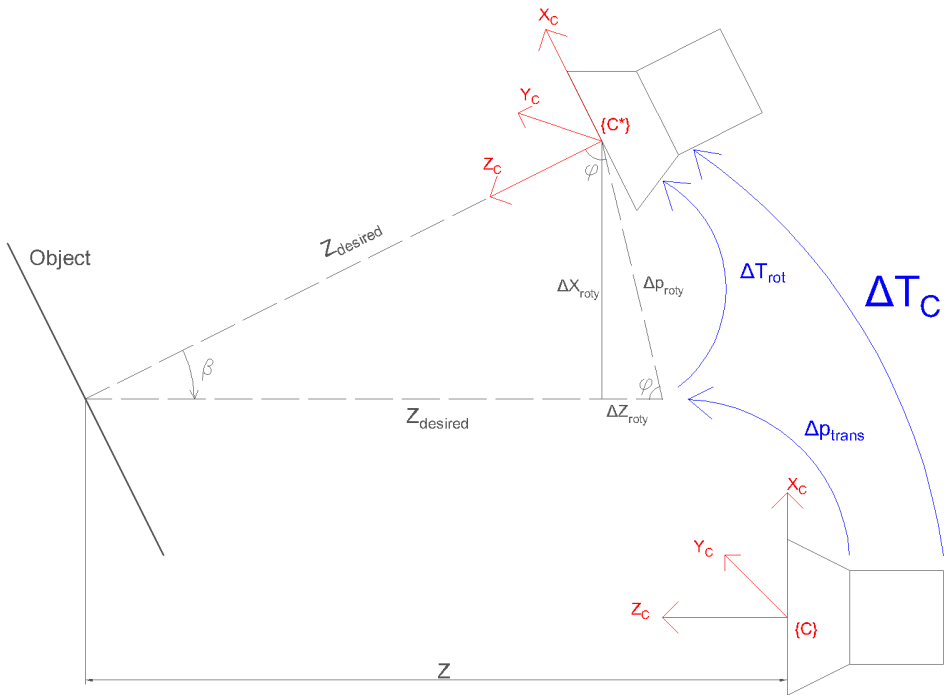


Figure 3.14: Calculation of camera offset when object is rotated with  $\beta$  around the Y-axis

Translational offset due to rotation around the X-axis,  $\Delta p_{rotx}$ , is calculated by the same method. Equation 3.13 has to be multiplied by  $(-1)$  if the rotation,  $\gamma$ , around the X-axis is positive.  $\Phi$  is the equivalent angle to  $\varphi$  in Figure 3.14, but related to the  $\gamma$  rotation around the X-axis.

$$\begin{aligned} \Delta Y_{rotx} &= \sin(\Phi) \Delta p_{roty} \\ \Delta Y_{rotx} &= \sin(\Phi) \sqrt{2Z_{desired}^2 (1 - \cos(\gamma))} \end{aligned} \quad (3.13)$$

$$\begin{aligned} \Delta Z_{rotx} &= \cos(\Phi) \Delta p_{roty} \\ \Delta Z_{rotx} &= \cos(\Phi) \sqrt{2Z_{desired}^2 (1 - \cos(\gamma))} \end{aligned} \quad (3.14)$$

The total position offset is found by adding the two vectors  $\Delta\mathbf{p}_{trans}$  and  $\Delta\mathbf{p}_{rot}$ .

$$\Delta\mathbf{p}_c = \Delta\mathbf{p}_{trans} + \Delta\mathbf{p}_{rot} = \begin{bmatrix} \Delta X_{trans} \\ \Delta Y_{trans} \\ \Delta Z_{trans} \end{bmatrix} + \begin{bmatrix} \Delta X_{roty} \\ \Delta Y_{rotx} \\ \Delta Z_{rotx} + \Delta Z_{roty} \end{bmatrix} \quad (3.15)$$

Rotational offset is extracted directly from the *solvePnP* function. These are *Roll-Pitch-Yaw Euler angles* and are converted to rotation matrix with equation 3.16 [2].

$$\Delta\mathbf{R}_C(\phi) = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_x(\gamma) = \begin{bmatrix} c_\alpha c_\beta & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma \\ s_\alpha c_\beta & s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma \\ -s_\beta & c_\beta s_\gamma & c_\beta c_\gamma \end{bmatrix} \quad (3.16)$$

The total camera transformation matrix,  $\Delta\mathbf{T}_C$  is found by combining  $\Delta\mathbf{p}_c$  and the rotation matrix  $\Delta\mathbf{R}_C$ .

$$\Delta\mathbf{T}_C = \begin{bmatrix} \Delta\mathbf{R}_C & \Delta\mathbf{p}_c \\ \mathbf{0} & 1 \end{bmatrix} \quad (3.17)$$

It is important to be aware of which base frame you are working in. These coordinates are in the camera frame and have to be converted into the TCP-frame before the commands are sent to the KR C4 controller.



## Chapter 4

# Practical Setup

This chapter gives an overview of the steps needed in order to get a working system. The different software and hardware applied in this project is presented in Appendix A.

### 4.1 Robot Cell

The KUKA Agilus robot cell consists of two identical six axis robots. Both equipped with a KUKA KR C4 controller. This is connected to a Siemens safety PLC over ProfiNET. Safety features like emergency stop and door-switch connects to the safety PLC.

The model in Figure 4.1 shows the layout of the robot cell and is created using VisualComponents.



Figure 4.1: Graphic model of the KUKA Agilus robot cell at Department Of Production and Quality Engineering, NTNU

## 4.2 Information Flow

ROS distributes data in a quite clever way. The system is built up by *nodes* and *topics*. The node is a program written in C++ or Python. This is where the different algorithms and calculations are executed. These nodes can publish and/or subscribe to topics, which essentially is an address carrying the distributed data. Figure 4.2 gives an overview of the information flow in the system. The blue box contains the vision system with nodes in rectangles and topics in ovals. The  $X'$ ,  $Y'$ ,  $Z'$ ,  $A'$ ,  $B'$ ,  $C'$  values passed on to the *kuka\_driver* is the desired translations and rotations of the manipulator. These are small steps of the total transformation  $\Delta T_C$ . All movements are executed in the tool-frame which is calibrated with its origin in front of the camera lens.

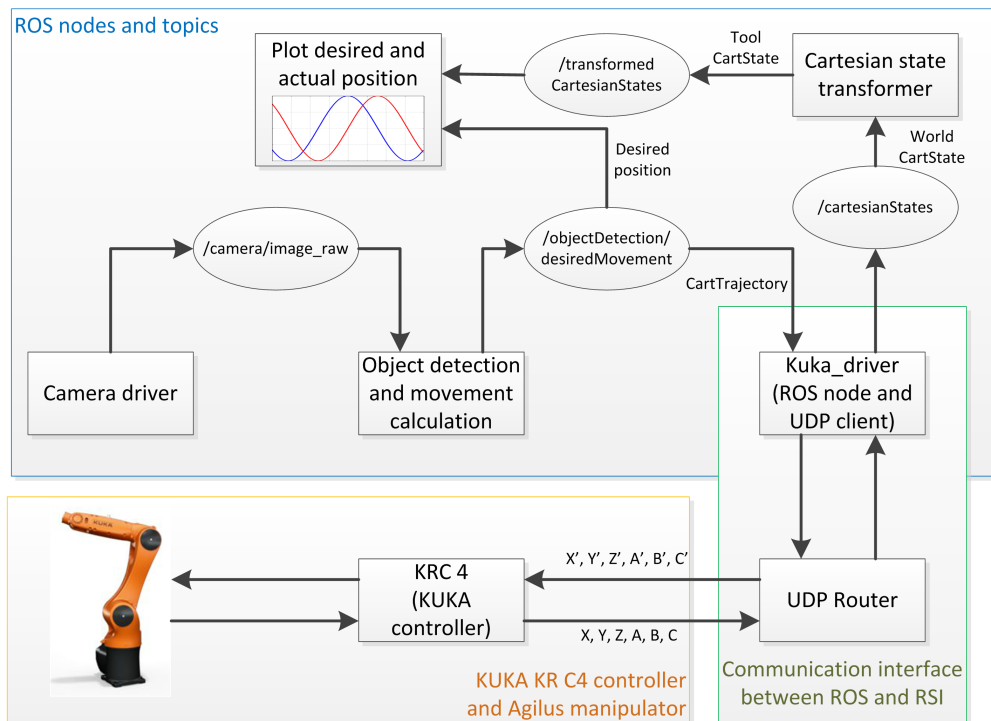


Figure 4.2: Information flow of the visual servoing system

### 4.2.1 Image Capturing

The image is captured using a Prosilica camera (subsection A). The image stream is published on the ROS topic `/camera/image_raw` using the "camera\_aravis" ROS package as driver. This needs to be converted from ROS image type to OpenCV image type with the help of the `cv_bridge` interface before the OpenCV library can be taken advantage of.

### 4.2.2 Object Detection and Calculations

Object detection and the desired movement of the manipulator was programmed as a ROS node with C++. The many features of the OpenCV library has also been utilized in this work.

The algorithm publish the total offset between camera pose and desired camera pose,  $\Delta T_C$ , and the next trajectory point for the manipulator, which is a small

fraction of this transformation. This fraction is calculated with the P-regulator described in section 5.2.1.

### 4.2.3 UDP Connection

PhD candidate Lars Tingelstad has created a Python UDP router for joint angle communication between the Linux computer and KUKA controller. This has been modified for cartesian commands and used in this project. Its task is to set up the RSI connection with the KUKA KR C4 controller and distribute messages between this and the *kuka\_driver* ROS node.

The *kuka\_driver* is a ROS node and UDP client. It subscribes to the cartesian commands calculated by the visual system, and publish these to the UDP router. It also receives the actual cartesian manipulator positions and distribute these on the ROS topic `"/cartesianStates"`.

### 4.2.4 Robot Sensor Interface

The KUKA Robot Sensor Interface (RSI) is a program for passing of information between the KR C4 controller and a sensor-system, such as a computer. This information is passed in an XML file over an Ethernet connection. To control the manipulator, there are two main methods used, joint and cartesian correction.

Joint correction sends the next point for each joint on the manipulator. For the Agilus robot used here, there are six joint values with a small step for each package sent. When using cartesian correction the next position is sent in the form of coordinates. These can be in relation to the world, end-effector or tool base.

The cartesian correction has been used in this project since the coordinates calculated in the object detection algorithm is cartesian as well. This saves the extra coding of inverse kinematics and minimize the error sources. The commands are sent to KR C4 in the tool-frame with the origin in the cameras TCP. Transformation matrix is shown earlier in Figure 3.5.

The setup for RSI on the KR C4 consist of four files describing the send/receive parameters, IP configuration, and different safety settings such as maximum stepwise correction, safety speed limits etc. These can be seen in the "RSI" folder on the digital appendix.



### 4.2.5 World-pose to TCP-pose

The actual cartesian states published by the KR C4 controller are referenced to the world frame, while the calculated trajectory references to the moving TCP frame. The actual cartesian coordinates are therefore sent to the ROS node *actual\_state\_transform*. This transforms the coordinates so they are referenced to the starting position of the TCP frame. This made it easier to compare the desired and actual manipulator positions during the step tests in chapter 5.



## Chapter 5

### Results

There has been performed two different visual servoing experiments. Both use the same camera, object and control algorithms to measure the necessary movements. The first test was performed with a stepper-motor only capable of 1D translation. The second test was done on the KUKA Agilus robots with six axes and can operate in 3D space.

#### 5.1 Stepper-motor

The setup in Figure 5.1 was used for real-life testing of the coordinate outputs from the object detection algorithms. The main objective was to test the coordinate calculator algorithm on a simpler system than the Agilus robot.

A stepper-motor was controlled with a Arduino Uno card, which receives the distance to the tracked object from ROS. This micro-controller then calculates the necessary movement in order to stay in the right position according to the object (in this instance 300 mm).

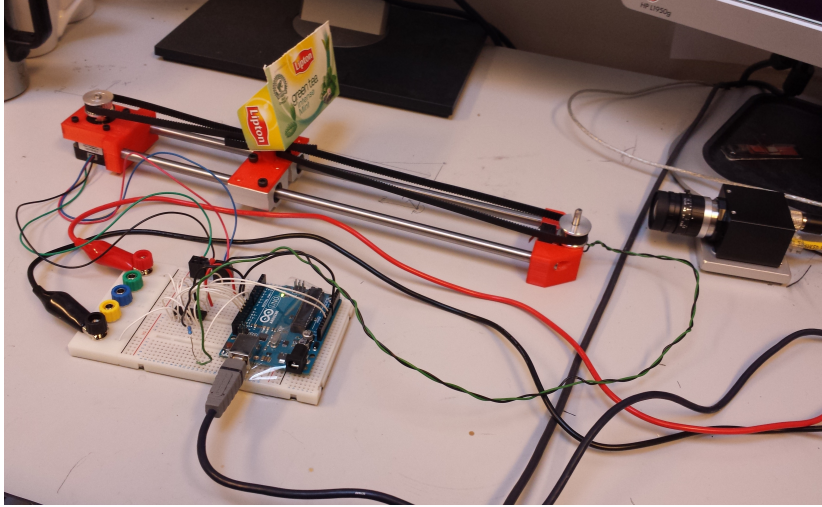


Figure 5.1: Setup for 1D tracking with Arduino Uno and stepper-motor

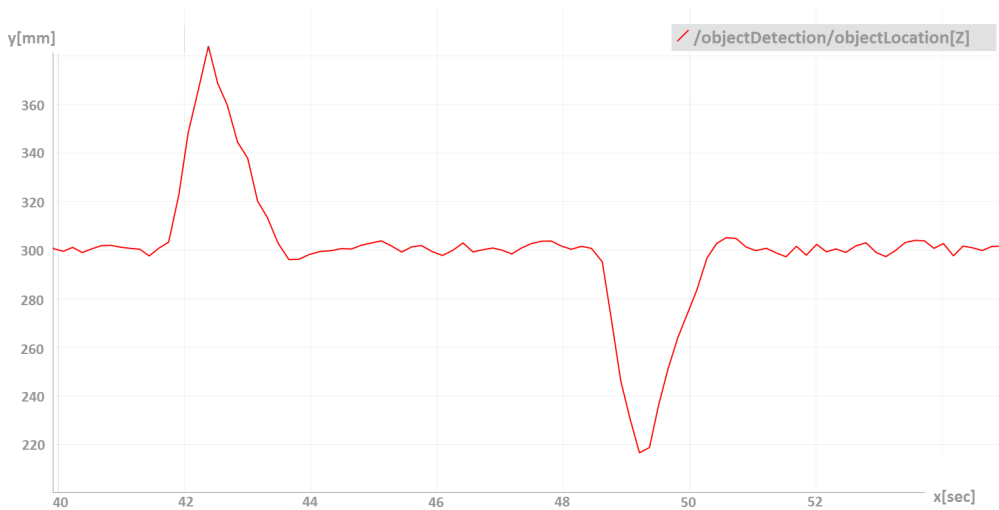


Figure 5.2: Stepper motor responsiveness to distance calculation algorithm

Figure 5.2 shows the responsiveness of the stepper-motor when the distance between camera and object is changed. It is observed that the reference distance of 300 mm is restored after approximately a second when the step is 80 mm.

The Arduino code is found in appendix B and in the digital appendix.

## 5.2 KUKA Agilus

This section evaluates the performance of the implemented visual servoing system. One robot is holding a target, while the other, equipped with a camera, is tracking and following this moving object. A number of graphs is presented, showing the relation between desired pose of the robot based on the calculations of the visual system, and the real position feed back from the KUKA KR C4 controller. All values and plots are referenced to the TCP frame from Figure 3.5.

### 5.2.1 Regulator Adjustment

Six proportional regulators are used in the control loop (PBVS control in Figure 2.2). One for each translational and one for each rotational command. Ziegler-Nichols method for regulator adjustment [12] is applied and shown in Table 5.1.

Control Type	$K_p$	$T_i$	$T_d$
$P$	$0.5 * K_c$		
$PD$	$0.65 * K_c$		$0.12 * T_c$
$PI$	$0.45 * K_c$	$0.85 * T_c$	
$PID$	$0.65 * K_c$	$0.5 * T_c$	$0.12 * T_c$

Table 5.1: Ziegler-Nichols table for adjustment of regulators.

$K_c$  - critical gain

$T_c$  - period time with critical gain

$K_p$  - proportional gain

$T_i$  - integration time

$T_d$  - derivation time

Critical gain was found to be approximately 0.01 for both the translational and rotational response. After running tests with different  $K_p$  values around  $0.5 * K_c$ , the best results was found at  $K_p = 0.005$  for position and  $K_p = 0.004$  for rotation. Response of the manipulators actual position with  $K_p = K_c = 0.01$  is shown in Figure 5.3.

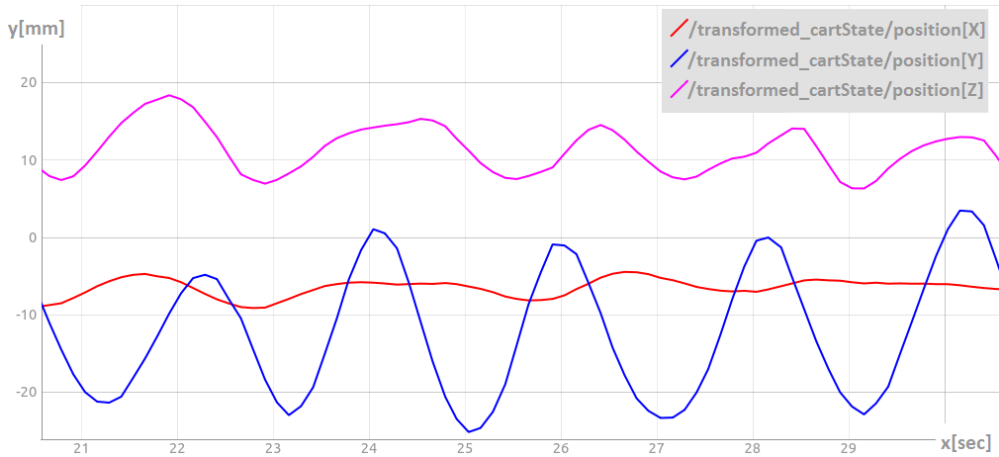


Figure 5.3: Manipulator position feedback with  $K_p = K_c = 0.01$

The KR C4 controller also have internal regulators for the joint servos with current, velocity and position control.

### 5.2.2 System Performance

The manipulators response has been analyzed for a series of step-tests, both translational and rotational. The graphs named `"/desiredMovement/desired_position[X/Y/Z/A/B/C]"` is the offset calculated by the object detection algorithms, and show the real time  $\Delta T_c$ . Graphs named `"/transformed_cartState/position[X/Y/Z/A/B/C]"`, is the manipulators actual position in relation to the starting position of the TCP base. The y-axis of the plots are in mm for translations and degrees for rotations. The x-axis shows the elapsed time in seconds.

Figure 5.4 shows a step-test for a 3D translation. The visual system measures an offset of approximately 140, -70 and 120 mm in  $X$ ,  $Y$  and  $Z$  direction. When the manipulator is activated after 5.5 seconds, the actual positions move towards the reference given by the visual system, and stabilizes after about 2.5 seconds. The position stabilize around the desired offset which is zero, even though only a P-regulator is used as position regulator. There is no need to introduce an integral part in the regulator.

A derivative part would probably speed up the correction process of the robot. The noise on the offset values calculated would however create problems for a PD-regulator. With a properly filtered signal this would be an interesting

system to implement.

When comparing the initial offset given by the robotic vision, and the resulting stable position of the manipulator, there is a small deviation. All three measurements are off by 10-20 mm. There can be a number of reasons for this error, like inaccurate measurement of object test image or calculation error. The most likely is however inaccurate camera calibration. The step-test is taken with the object's starting position in the edge of the cameras field of view. This is where the image is most distorted due to the lens curvature, and in order to correct for it, accurate calibration is important. This process was possibly not given enough attention in the experiment set-up.

The initial position error does not seem to affect the manipulators movement to any extent. When the camera moves closer to the desired position, and the object is centered in the image, the distortion error diminishes [13]. The relative accurate positioning of the system is verified by a manual distance measurement after a step-test has been conducted. This is shown in Figure 5.5.

Figure 5.6 show the manipulators response to a  $15deg$  rotation around the  $X$  axis and  $9deg$  rotation around the  $Y$ -axis. The plotted graphs are named with the KUKA convention of  $A$  being rotation around the  $Z$ -axis,  $B$  around  $Y$ -axis, and  $C$  around the  $X$ -axis. The actual manipulator position stabilize very close to the reference point. This shows the good estimation of the coordinate calculator in relation to rotations. It is important to have in mind that this rotational test was done with the object centered in the image. Image distortion is therefore minimal.

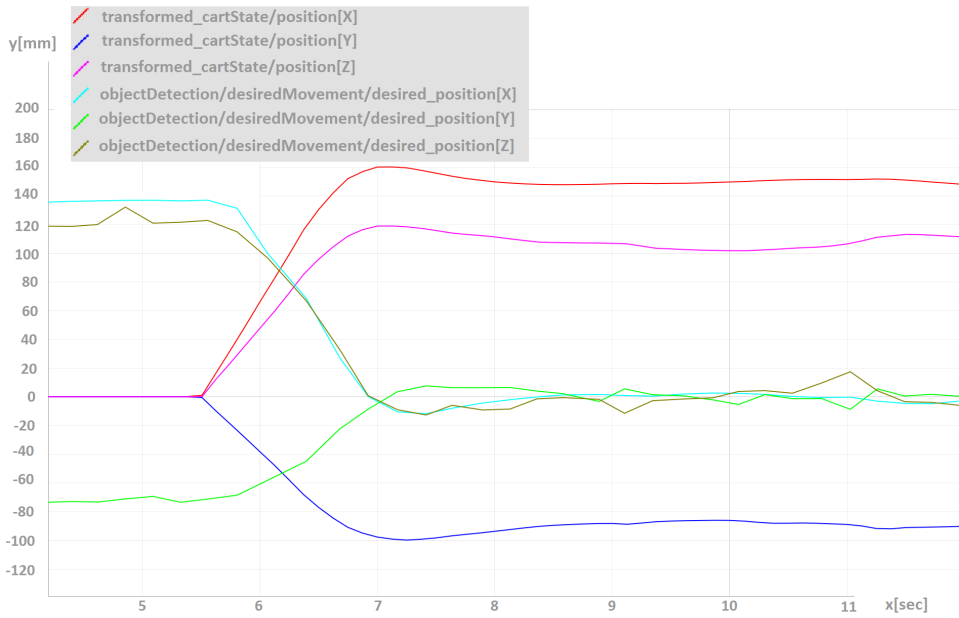


Figure 5.4: Manipulator response on a three dimensional position step

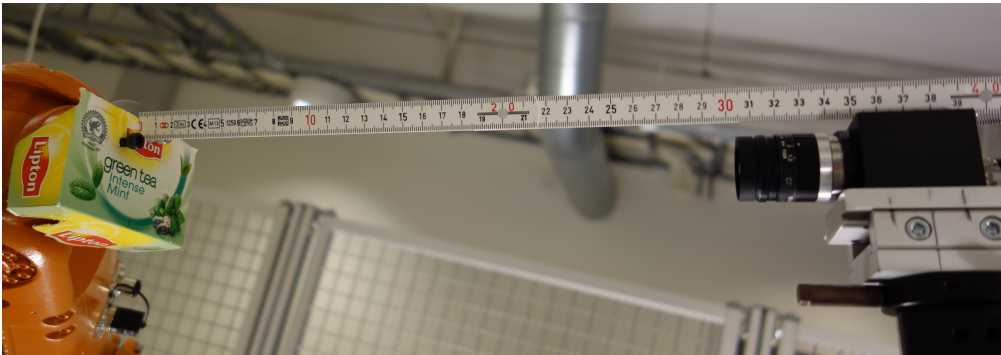


Figure 5.5: Manual measurement of distance to object after visual servoing translation



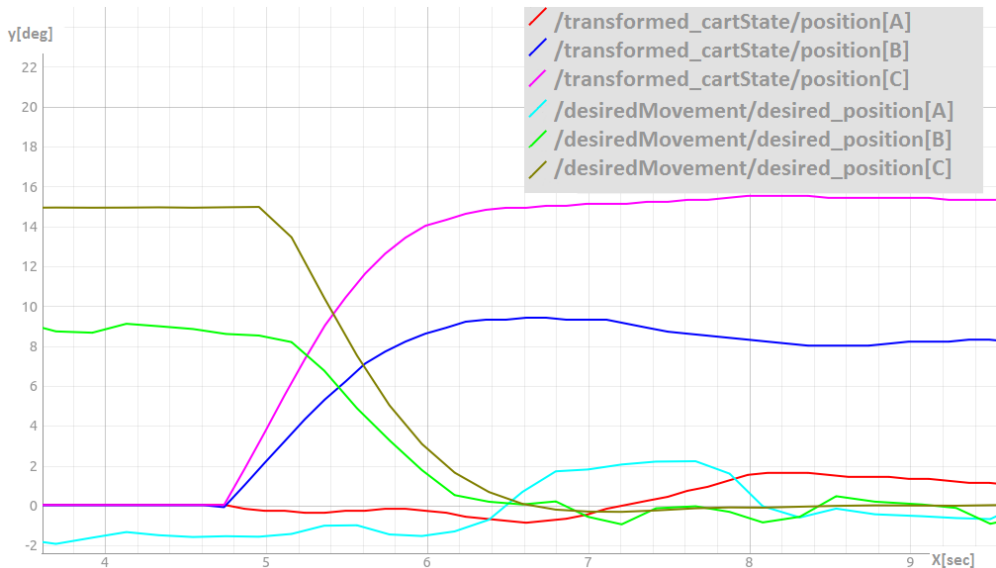


Figure 5.6: Manipulator response to a rotational step around axis X and Y

### 5.2.3 Reference Filter

The offset calculations based on the SIFT and SURF algorithms show some variations on the output used as signal for the manipulator trajectory. In Figure 5.7 the variations on the SIFT calculation has an amplitude of up to 12mm. This is not a problem when tracking the moving object, but when the manipulator approach the reference point it fluctuates around it. The impact on the positioning is one aspect that needs to be considered if a similar system is to be used in real operations. Another consideration is that this unnecessary positioning around the reference wears on the servos and brakes of the manipulator. For both these reasons it is desirable to avoid these fluctuations.

A "moving-average filter" was implemented on the visual servoing system after the model in equation 5.1 [14].  $N^*$  is the number of past values to average over.

$$y_F(k) = \frac{1}{N^*} \sum_{i=k-N^*+1}^k y_i \quad (5.1)$$

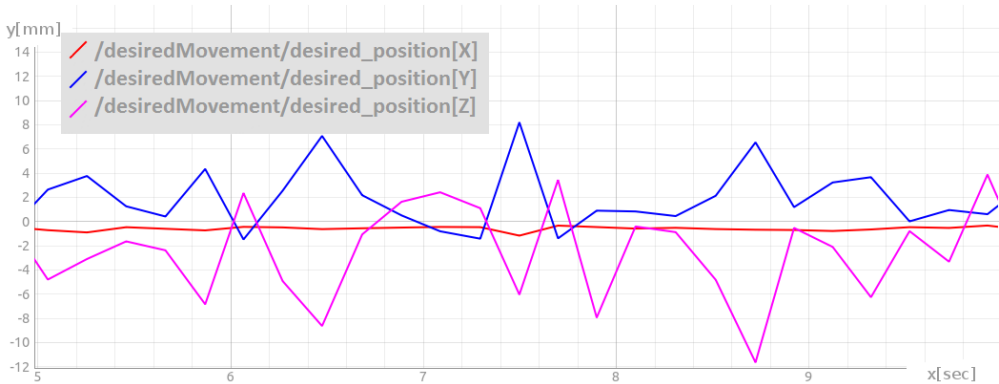


Figure 5.7: Unfiltered XYZ reference points from sensor system

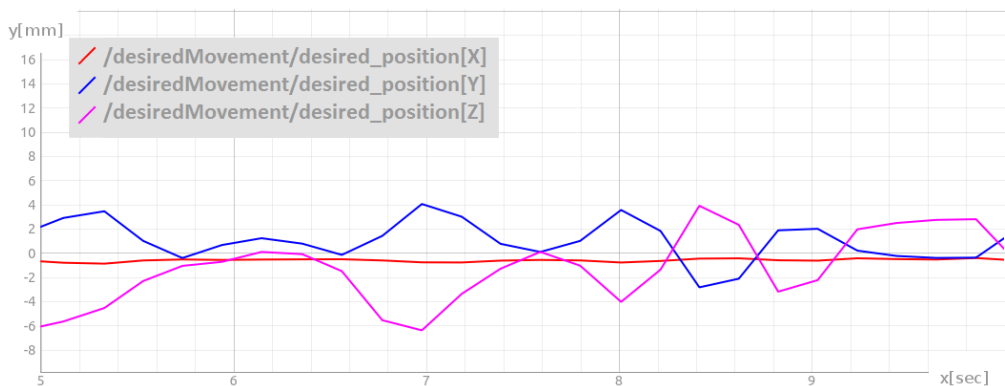


Figure 5.8: Filtered XYZ reference points from sensor system

Figure 5.7 and 5.8 show the position calculated by the coordinate calculator with unfiltered and filtered signal. Average is taken over the last two values. The variations after filtering are as seen considerably smaller than the raw signal. An interesting observation is that the X-value is almost stable while there are large variations in the Y and Z direction. This points to the interpretation that it is the offset due to rotation that cause the largest sensor disturbance. This makes sense, since at a distance of  $300\text{mm}$  a small rotation error will lead to considerably larger offsets in the camera frame (as explained in section 3.3.3). This has not been investigated thoroughly and a definite conclusion can not be drawn.

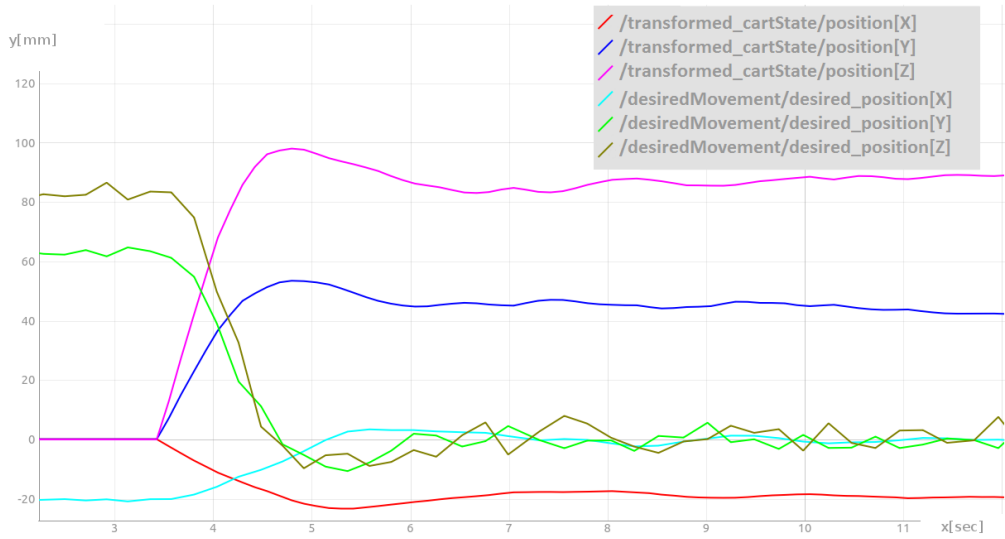


Figure 5.9: XYZ step-test on unfiltered system

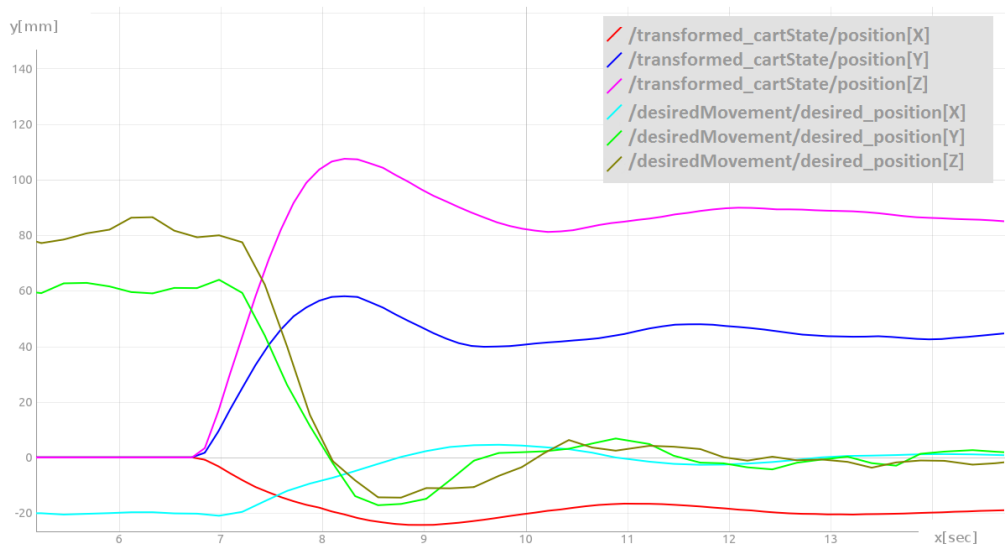


Figure 5.10: XYZ step-test on filtered system

In Figure 5.9 and 5.10 a position step response is plotted with unfiltered and filtered reference signals. It is clear that the filter slows down the process considerably even when averaging over only the two last values. The fluctuations

around setpoint is smaller, but the process use twice the time to stabilize. This is due to the low frequency of the tracking algorithms. The extra time delay introduced by the filter has a large effect. The overshoot is also larger due to the added time delay, and the  $Kp$  value of the filtered process needs adjustment.  $Kp = 0.004$  for the position regulators was found to give good results. Since a larger amount of averaging values would improve the filter, it is desirable to speed up the tracking process. Some possible solutions are listed below.

- A powerful computer would speed up the the tracking algorithm and improve the results without any further adjustments.
- One could also downsize the image resolution, but this could in turn effect the reliability and precision of the tracker.
- The most elegant solution is to use a state predictor for the movement of the object in order to shrink the image area being matched with SIFT/SURF. Only searching for keypoints in the predicted location of the object would increase the speed of the program.

#### 5.2.4 Position Lock

To prevent the unnecessary movement around setpoint, a position lock mechanism was introduced. It monitors the transformation offset and triggers the "position\_lock" flag when the offset is lower than  $5mm$  for position AND  $2deg$  for rotation. This locks the manipulator by sending out  $\Delta T_C = \mathbf{0}$  until the position offset exceeds  $10mm$  OR the rotation offset is larger than  $4deg$ .

Figure 5.11 show the step response with filtering and position lock enabled. It is clear that the fluctuations around the setpoint are gone. This is a good method for positioning of the manipulator if small deviations do not cause problems for the specific task at hand. It is not recommended when tracking a moving object due to the extra delay it introduces. Every time the position change, the tracked object will have a  $10mm$  lead. With the already substantial time delays in the system, it is not optimal to add another.

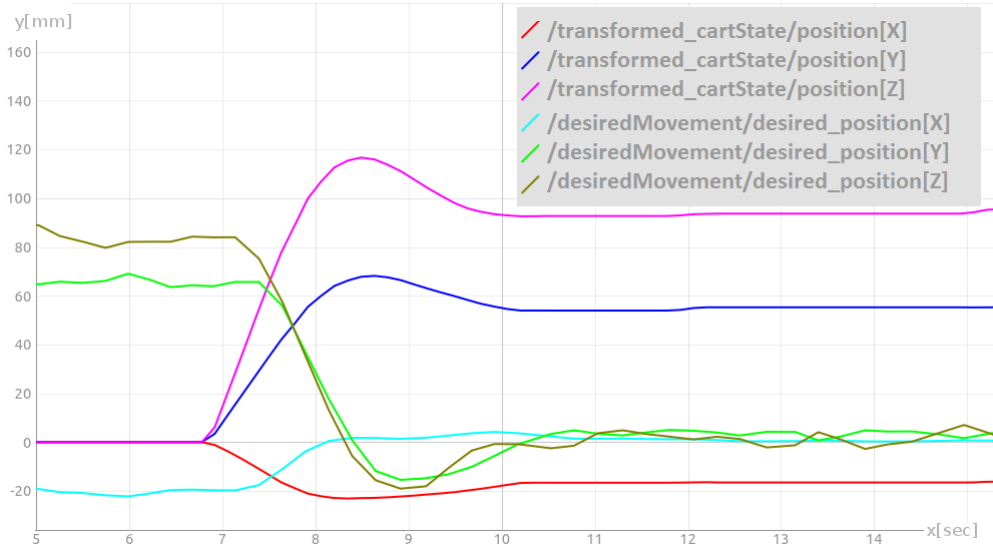


Figure 5.11: XYZ step-test on filtered system with position lock.  $K_p = 0.005$

### 5.2.5 SURF Test

A final test was performed with the same setup as in figure 5.11, only now SURF was used for object detection. The starting position actually had to be modified by a few centimeters because the SURF algorithm was unable to find the object in the same location as used earlier with SIFT. This observation strengthens the results from section 3.2.3. It is also clear that the larger fluctuations in the signal input results in manipulator movement in what should have been the stable area, even with the "position lock" function enabled.

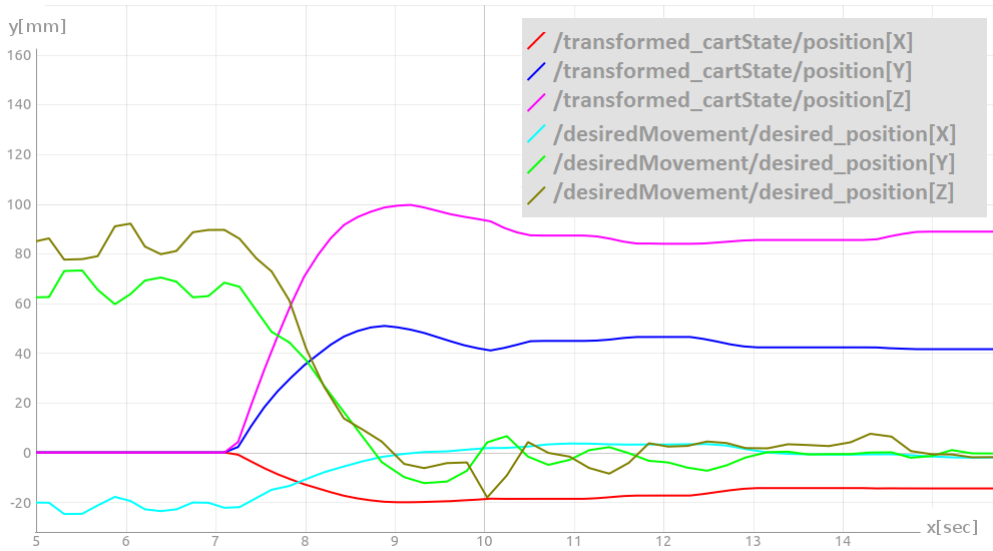


Figure 5.12: XYZ step-test on filtered system with position lock.  $K_p = 0.005$ . SURF used for object detection

### 5.2.6 Maximum Correction

As a safety precaution the RSI interface was programmed with a maximum total and stepwise correction for the motion commands received from the sensor system. As an extra safety barrier there was also put a maximum correction limit to the commands sent from the coordinate calculator programmed in ROS. This was done in order to have a safe working environment and avoid any damage to personnel or equipment. This limited the maximum speed of the tracking system, and faster transformation correction could have been possible without it.

### 5.2.7 Video

A video of the object tracking has been produced and placed in the digital appendix. The manipulator holding the object moves in a predefined path, while the one with camera tracks the object and relocates in order to stay  $300\text{mm}$  perpendicular to it. The computer screens show the visual tracking of the object with a green rectangle around it, and graphs of actual and desired position.

## Chapter 6

# Concluding Remarks

### 6.1 Discussion

The road towards this main objective has been all other than paved, and what were initially thought to be small obstacles has often turned out to need more attention and effort than planned.

Starting out with two unused robot manipulators in an empty robot cell and end up with a working visual servoing system, has been an interesting journey. The objective of creating a working system was always the main focus, and has taken up most of the time. The lack of background knowledge in necessary software like Linux, ROS, OpenCV, Python and VisualComponents has been a challenge from the start-up. The first part of the project was almost exclusively spent getting to know the software and programming interfaces.

The comprehensive practical part has seized most of the available time. The theory section has consequently been limited to explaining the work performed and some background information on key algorithms applied.

The object tracking show some time delay that affects the system response. Solutions have been proposed, and with the adequate time it would be very interesting to implement these for higher tracking speed. With a faster system it is possible to improve the reference filter and test if a PD-regulator would speed up the process.

As an addition to the objectives, a comparison between the SIFT and SURF tracking algorithm was performed. With all the initial problem formulations fulfilled, the undersigned feel that the project is successfully completed.

## 6.2 Conclusion

The focus of this project has always been to create a working visual servoing system for the new Agilus robot cell. This has been achieved with one manipulator holding a moving target, and the other using computer vision to track this target and follow its motions. The system can follow both translational and rotational movements.

The results show small variations in the reference signal created by the object detection and coordinate calculation algorithm. This results in fluctuations around setpoint when the tracked object is not moving. A solution introducing a "position lock" mechanism that triggers when the robot is positioned inside a threshold, was proved relative efficient. If this is a good solution will differ from operation to operation, and depend on the precision needed for the specific task.

The speed of the tracking is not very high, but it is believed to be more than adequate for use in a valve operation with an ROV. It is possible to improve the speed, and potential solutions have been proposed.



### 6.3 Recommendations for Further Work

The visual servoing system for the KUKA Agilus robot cell is now working. As seen in chapter 5, there are several improvements that can be implemented in the future.

In order to improve the system frequency without updating the hardware, it is necessary to minimize the object detection area. An estimator, such as a *Kalman filter*, can be used to scale down this area. If only the area believed to contain the object is searched, it will increase the speed. With SIFT and SURF it is not even necessary that the whole object is in this area as long as enough matches can be made from the visible part.

The object detection used in this project can only detect planar objects. If visual servoing is to be used for localization and approach of 3D objects, a different detection method is needed. The article *From Contours to 3D object Detection and Pose Estimation* [15] can be useful in this work.

A very interesting thought in a longer perspective is to use completely autonomous ROVs to perform valve operations on subsea X-mas trees. If a viable 3D detection is implemented, a library of different valves and handles can be created. An ROV can then approach the X-mas tree, and search for the specific pattern of valves, handles and other recognizable features used on this construction. When a match is made, it can perform a preprogrammed task, such as a valve operation.



## References

- [1] International Organization of Standardization. Petroleum and natural gas industries – drilling and production equipment – wellhead and christmas tree equipment. Technical Report 10423:2009, ISO, 2009.
- [2] B Siciliano, L Sciavicco, L Villani, and G Oriolo. *Robotics: Modelling, Planning and Control*. 2009.
- [3] Kuka kr agilus sixx documentation. [http://www.kuka-robotics.com/res/sps/e6c77545-9030-49b1-93f5-4d17c92173aa\\_Spez\\_KR\\_AGILUS\\_sixx\\_en.pdf](http://www.kuka-robotics.com/res/sps/e6c77545-9030-49b1-93f5-4d17c92173aa_Spez_KR_AGILUS_sixx_en.pdf). Accessed: 07-10-2014.
- [4] Peter Corke. *Robotics, Vision and Control*. 2011.
- [5] Samarth Brahmhatt. *Practical OpenCV*. 2013.
- [6] Opencv home page description. <http://opencv.org/about.html>. Accessed: 25-04-2014.
- [7] Ros package "image\_pipeline". [https://github.com/ros-perception/image\\_pipeline](https://github.com/ros-perception/image_pipeline). Accessed: 06-06-2014.
- [8] Frédéric Devernay and Olivier Faugeras. Straight lines have to be straight. *Machine Vision and Applications 13, 1: 14-24*, 2001.
- [9] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.
- [10] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. 2006.
- [11] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics 2: 164-168*, 1944.
- [12] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. 1942.

## REFERENCES

---

- [13] Duane C. Brown. Decentering distortion of lenses. *Photogrammetric Engineering* 32: 444–462, 1966.
- [14] Dale E. Seborg, Thomas F. Edgar, and Duncan A. Mellichamp. *Process Dynamics and Control, second edition*. 2004.
- [15] Nadia Payet and Sinisa Todorovic. From contours to 3d object detection and pose estimation. *IEEE International Conference on Computer Vision*, 2011.
- [16] International Electrotechnical Commission. Functional safety. Technical Report 61508, IEC, 2010.
- [17] Arduino home page description. <http://arduino.cc/en/Main/ArduinoBoardUno>. Accessed: 24-04-2014.
- [18] Ros home page description. <http://www.ros.org/about-ros/>. Accessed: 24-04-2014.

## Appendix A

# Equipment and Software

### Hardware

Figure A.1 shows the control cabinet for the two KUKA Agilus robots. From top to bottom is the Siemens safety PLC, a network switch, two KR C4 KUKA controllers and computers used for object detection and coordinate calculation.



Figure A.1: The installed control cabinet for the Agilus robot cell

### KUKA Agilus Robot

There is used two identical "KUKA Agilus: KR 6 R900 sixx" manipulators. These have six rotating axes. Denavit-Hartenberg convention is used and showed in Table 2.1.

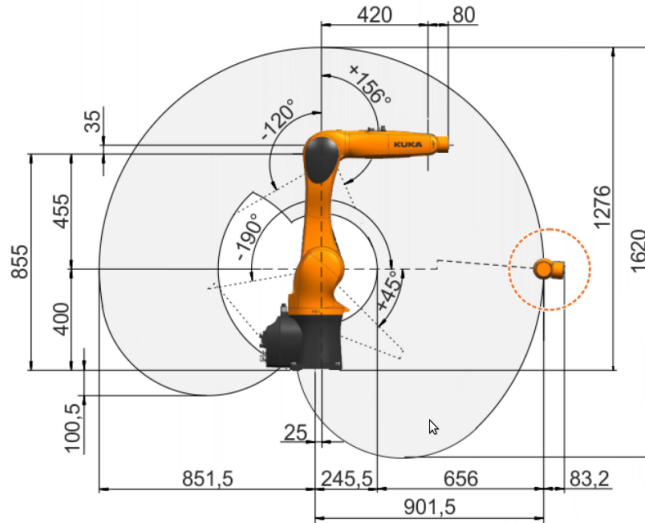


Figure A.2: KUKA Agilus KR 6 R900 sixx dimensions [3]

### Siemens PLC

A Siemens safety PLC controls the hard-wired safety system, such as emergency stop. According to the IEC standard of Functional Safety [16] this PLC meets all the requirements for use in applications up to SIL3.

### Prosilica Camera

The camera used for image input is Prosilica GC650. This is a small grayscale camera with Gigabit Ethernet interface. The resolution is 659 x 493 pixels, and it can capture up to 90 fps.

## GC650/650C



Figure A.3: Prosilica GC650 used for sensor input

### Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button [17].

The amount of inputs/outputs makes it very versatile and suited for use with smaller components. It is easily programmed with the Arduino software.

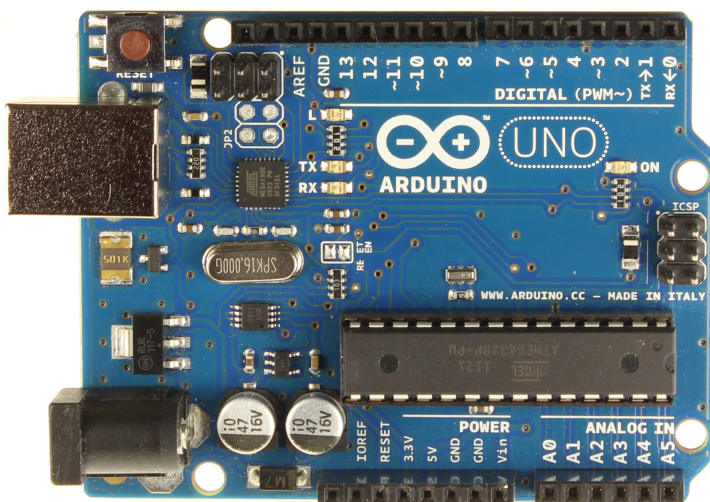


Figure A.4: The Arduino Uno board

### Software

#### ROS - The Robot Operating System

ROS was used for programming and sending of information between camera, computer and robots. It is available as experimental versions in multiple operating systems, but the recommended use is with Ubuntu Linux. The programming itself can be done with Python or C++.

It offers solutions for integration of 3rd party software and libraries like Arduino and OpenCV, which has been taken advantage of in this project.

*The Robot Operating System is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [18].*

#### OpenCV

*OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library [6].* The library has over 2500 algorithms, that are free to use. It is also nicely integrated into ROS, and these tools together makes a good platform for the object detection algorithms.

#### VisualComponents

VisualComponents is a 3D simulation tool for work cells and robots. This is a easy-to-use software with an extensive robot manipulator library, included the KUKA Agilus used in this project. It is here used for creating a model of the Agilus robot cell, and testing paths and movements to be programmed in the demo for Objective 4.

#### KUKA WorkVisual

WorkVisual is a program used for online configuration or monitoring of the KUKA controllers and robots. It was used to set-up names and IP-configuration for the two Agilus robots.



## Appendix B

### Source Code

```

1  /**
2  Object detection and coordinate calculation code. SIFT applied
3  Author: Eirik Anfinsen Solberg.
4  NTNU 2014
5  */
6
7  #include <ros/ros.h>
8  #include <image_transport/image_transport.h>
9  #include <cv_bridge/cv_bridge.h>
10 #include <sensor_msgs/image_encodings.h>
11 #include <opencv2/imgproc/imgproc.hpp>
12 #include <opencv2/highgui/highgui.hpp>
13 #include <opencv2/opencv.hpp>
14 #include <opencv2/nonfree/features2d.hpp>
15 #include <opencv2/features2d/features2d.hpp>
16
17 #include "opencv2/calib3d/calib3d.hpp"
18 #include "opencv2/core/core.hpp"
19 #include <stdio.h>
20 #include <iostream>
21
22 #include <stdlib.h>
23 #include "std_msgs/Float32MultiArray.h"
24
25 #include "eirik_vision/CartTrajectory.h"
26
27
28 using namespace cv;
29 using namespace std;
30
31
32 // Global values
33 float Kp_pos = 0.004, Kp_rot = 0.004;
34 float k_m1[6], k[6];
35 double pi = 3.14159;
36 vector< Point2f > scene_corners(4);
37 //trajectory_msgs::JointTrajectoryPoint desired_movement;
38 eirik_vision::CartTrajectory desired_movement;
39
40
41 class ImageConverter
42 {
43     ros::NodeHandle nh_;
44     image_transport::ImageTransport it_;
45     image_transport::Subscriber image_sub_;
46
47
48     public:
49     ImageConverter()
50         : it_(nh_)
51     {
52         // Subscribe to input video feed
53         image_sub_ = it_.subscribe("/camera/image_raw", 1,
54             &ImageConverter::imageCb, this);
55     }
56
57
58     Mat calcMovement(Mat R, Mat tvec, Mat rvec)
59     {
60         //-- calculates translation vector in mm. Currently in number of pixels --
61         Mat p_co(3, 1, CV_64FC1); // CV_8UC3 // tvec in millimeters. vector from camera frame to
object center
62
63         // z-direction of vector
64         // if z_relation == 1 then dist = 300mm. 820 is focal length. Test object at 300
mm distance
65         double z_relation = tvec.at<double>(0, 2)/820;
66         p_co.at<double>(2, 0) = 300 * z_relation;
67
68         // Movement in x-direction with regards to offset between object and camera
69         double phi = atan(tvec.at<double>(0,0) / (tvec.at<double>(0,2))); // angle of
offset in x-direction
70         p_co.at<double>(0, 0) = p_co.at<double>(2, 0) * tan(phi);
71

```

```

72         // Movement in y-direction with regards to offset between object and camera
73         double tetha = atan((tvec.at<double>(0,1)) / (tvec.at<double>(0, 2)));
74         p_co.at<double>(1, 0) = p_co.at<double>(2, 0) * tan(tetha);
75         // -----
76
77
78         // Desired position of camera in relation to object
79         Mat p_cxo(3, 1, CV_64FC1);
80         p_cxo.at<double>(0,0) = 0;
81         p_cxo.at<double>(1,0) = 0;
82         p_cxo.at<double>(2,0) = 300;
83
84         // Subtract desired position to get reference point at zero
85         Mat delta_p_c(3, 1, CV_64FC1);
86         delta_p_c.at<double>(0,0) = p_co.at<double>(0,0) - p_cxo.at<double>(0,0);
87         delta_p_c.at<double>(1,0) = p_co.at<double>(1,0) - p_cxo.at<double>(1,0);
88         delta_p_c.at<double>(2,0) = p_co.at<double>(2,0) - p_cxo.at<double>(2,0);
89
90         // Calculate rotation contribution to the translation and find total offset
91         double beta = rvec.at<double>(1,0);
92         double psi;
93
94         // offset x-direction
95         if(rvec.at<double>(1,0) < 0) // If negative beta
96             psi = (pi+beta)/2;
97
98         if(rvec.at<double>(1,0) >= 0) // If positive beta
99             psi = (pi-beta)/2;
100
101         if(rvec.at<double>(1,0) >= 0)
102             delta_p_c.at<double>(0,0) = -sin(psi)*sqrt((2*p_cxo.at<double>(2,0)*p_cxo.at<double>
(2,0))*(1-cos(beta)))+delta_p_c.at<double>(0,0);
103
104         if(rvec.at<double>(1,0) <= 0)
105             delta_p_c.at<double>(0,0) = sin(psi)*sqrt((2*p_cxo.at<double>(2,0)*p_cxo.at<double>
(2,0))*(1-cos(beta)))+delta_p_c.at<double>(0,0);
106
107         // offset y-direction
108         double gamma, fi;
109         gamma = rvec.at<double>(0,0);
110         if(gamma < 0) // If negative beta
111             fi = (pi+gamma)/2;
112
113         if(gamma >= 0) // If positive beta
114             fi = (pi-gamma)/2;
115
116         if(gamma >= 0)
117             delta_p_c.at<double>(1,0) = sin(fi)*sqrt((2*p_cxo.at<double>(2,0)*p_cxo.at<double>(2,0))*
(1-cos(gamma)))+delta_p_c.at<double>(1,0);
118
119         if(gamma <= 0)
120             delta_p_c.at<double>(1,0) = -sin(fi)*sqrt((2*p_cxo.at<double>(2,0)*p_cxo.at<double>
(2,0))*(1-cos(gamma)))+delta_p_c.at<double>(1,0);
121
122         // offset z-direction
123         delta_p_c.at<double>(2,0) = p_cxo.at<double>(2, 0) - p_cxo.at<double>(2, 0)*cos(beta) +
p_cxo.at<double>(2, 0) - p_cxo.at<double>(2, 0)*cos(gamma) + delta_p_c.at<double>(2,0);
124
125
126
127         // Moves rotation and position matrix into one common matrix with two columns
128         Mat transformation(3, 2, CV_64FC1);
129         transformation.at<double>(0,0) = rvec.at<double>(0,0);
130         transformation.at<double>(1,0) = rvec.at<double>(1,0);
131         transformation.at<double>(2,0) = rvec.at<double>(2,0);
132         transformation.at<double>(0,1) = delta_p_c.at<double>(0,0);
133         transformation.at<double>(1,1) = delta_p_c.at<double>(1,0);
134         transformation.at<double>(2,1) = delta_p_c.at<double>(2,0);
135
136         cout << "camera frame = " << transformation << endl << endl << "psi" << psi << "beta" <<
beta << "pi" << pi << endl;
137
138         return transformation;
139     }

```

```

140
141
142 void calcStepwiseMovement(Mat transformation)
143 {
144     std_msgs::Float32MultiArray delta_Tc, test_location;
145     delta_Tc.data.clear();
146
147     // Move coordinates into data type for sending over ROS
148     // The axis are shifted according to the transformation matrix from
149     // camera to TCP. More info under "extrinsic parameters" in the report.
150     // The variables are also grouped according to the KUKA convention which
151     // is (X, Y, Z, A, B, C) where A, B and C is rotation around axis Z, Y and X.
152     delta_Tc.data.push_back(transformation.at<double>(2,1)); // X = Z
153     delta_Tc.data.push_back(-transformation.at<double>(0,1)); // Y = -X
154     delta_Tc.data.push_back(-transformation.at<double>(1,1)); // Z = -Y
155     delta_Tc.data.push_back(-transformation.at<double>(1,0)); // A (rotz) = -roty_cam
156     delta_Tc.data.push_back(-transformation.at<double>(0,0)); // B (roty) = -rotx_cam
157     delta_Tc.data.push_back(transformation.at<double>(2,0)); // C (rotx) = rotz_cam
158
159     //cout << endl << delta_Tc << endl;    // Print unfiltered values
160
161     //----- Moving average filter -----
162     float filtered[6];
163     for(int i=0; i<=5; i++)
164     {
165         k[i] = delta_Tc.data[i];
166         filtered[i] = (k[i]+k_ml[i])/2;
167         delta_Tc.data[i] = filtered[i];
168         k_ml[i]=k[i];
169     }
170
171     // -----
172
173
174
175     //cout << endl << delta_Tc << endl;    // Print filtered values
176
177     // Change rotations form radians to degrees
178     for (int i = 3; i <= 5; i++)
179     delta_Tc.data[i] = (delta_Tc.data[i]/(3.14159))*180;
180
181
182     // Find direction and rotation with highest offset
183     desired_movement.max_offset_pos.clear();
184     desired_movement.max_offset_pos.push_back(0);
185     desired_movement.max_offset_rot.clear();
186     desired_movement.max_offset_rot.push_back(0);
187     for (int i = 0; i <= 5; i++)
188     {
189         if (i < 3)
190         {
191             if (abs(delta_Tc.data[i]) > desired_movement.max_offset_pos[0])
192             {
193                 desired_movement.max_offset_pos.clear();
194                 desired_movement.max_offset_pos.push_back(abs(delta_Tc.data[i]));
195             }
196         }
197         if (i >= 3)
198         {
199             if (abs(delta_Tc.data[i]) > desired_movement.max_offset_rot[0])
200             {
201                 desired_movement.max_offset_rot.clear();
202                 desired_movement.max_offset_rot.push_back(abs(delta_Tc.data[i]));
203             }
204         }
205     }
206
207
208
209     // P-regulator. Kp values are set on top of script
210     desired_movement.next_point.clear();
211     desired_movement.desired_position.clear();
212     for (int i = 0; i <= 5; i++)
213     {

```

```

214         if (i <= 2) // Translation regulator
215             desired_movement.next_point.push_back(delta_Tc.data[i]*Kp_pos);
216
217         if (i > 2) // Rotation regulator
218             desired_movement.next_point.push_back(delta_Tc.data[i]*Kp_rot);
219
220         desired_movement.desired_position.push_back(delta_Tc.data[i]);
221     }
222
223 }
224
225 void Publish()
226 {
227     ros::NodeHandle n;
228     ros::Publisher pub = n.advertise<eirik_vision::CartTrajectory>("objectDetection/
desiredMovement", 1);
229     pub.publish(desired_movement);
230     ROS_INFO("PUBLISHING CAMERA MOVEMENT ON TOPIC 'objectDetection/objectLocation' ");
231     ros::spinOnce();
232 }
233
234
235 Mat calcTransformMatrix()
236 {
237     Mat rvec, tvec, R;
238
239     bool useExtrinsicGuess = false;
240     int flags=P3P;
241
242     vector<Point3d> objectPoints;
243     vector<Point2d> imagePoints;
244
245     Mat cameraMatrix =(Mat_<float>(3, 3) << 820, 0, 361, 0, 823, 225, 0, 0, 1); //From
camera calibration. Values in pixels
246
247     // Set tracking point to object center
248     objectPoints.push_back (Point3d (-120, -70, 0));
249     objectPoints.push_back (Point3d (120, -70, 0));
250     objectPoints.push_back (Point3d (120, 70, 0)); // corner points on
object. x, y, z on original picture
251     objectPoints.push_back (Point3d (-120, 70, 0));
252
253     imagePoints.push_back (Point2d (scene_corners[0].x, scene_corners[0].y)); //
corners from captured image. x, y
254     imagePoints.push_back (Point2d (scene_corners[1].x, scene_corners[1].y));
255     imagePoints.push_back (Point2d (scene_corners[2].x, scene_corners[2].y));
256     imagePoints.push_back (Point2d (scene_corners[3].x, scene_corners[3].y));
257
258
259     Mat distCoeffs(4,1,cv::DataType<double>::type);
260     distCoeffs.at<double>(0) = -0.187112;
261     distCoeffs.at<double>(1) = 0.046056;
262     distCoeffs.at<double>(2) = -0.002983;
263     distCoeffs.at<double>(3) = 0.001150;
264     //distCoeffs.at<double>(4) = 0;
265
266     solvePnP(objectPoints, imagePoints, cameraMatrix, distCoeffs, rvec, tvec, false,
CV_ITERATIVE);//, false, CV_P3P);
267
268
269
270     cout << "RVEC: " << rvec << endl << "TVEC: " << tvec << endl;
271
272     // Convert from rotation vector to rotation matrix with "Rodrigues"
273     // "R" is the rotation matrix of the object in relation to camera
274     Rodrigues(rvec, R);
275
276     Mat transformation = calcMovement(R, tvec, rvec);
277     return transformation;
278 }
279
280
281 // imageCb is called every time a new message is received over ROS
282 void imageCb(const sensor_msgs::ImageConstPtr& msg)

```

```

283 {
284   cv_bridge::CvImagePtr cv_ptr;
285   try
286   { // Convert image from ROS format into OpenCV format
287     cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
288   }
289   catch (cv_bridge::Exception& e)
290   {
291     ROS_ERROR("cv_bridge exception: %s", e.what());
292     return;
293   }
294
295   // ----- SIFT -----
296   // Example from http://docs.opencv.org/doc/tutorials/features2d/feature\_homography/feature\_homography.html#feature-homography used
297   Mat img_show;
298   float nndrRatio = 0.7;
299   double t0 = getTickCount();
300
301
302   Mat objectMat = imread("/home/eirikaso/catkin_ws/src/eirik_vision/te.jpg"); // Grayscale.
303   If color -> convert to grayscale
304   Mat sceneMat = cv_ptr->image; //imread("te.jpg");
305
306   vector< KeyPoint > keypoints0;
307   vector< KeyPoint > keypointsS;
308
309   // Step 1: Extract keypoints from object and scene
310   SiftFeatureDetector detector(2000);
311   detector.detect(objectMat, keypoints0);
312   detector.detect(sceneMat, keypointsS);
313
314   Mat descriptors_object, descriptors_scene;
315   SiftDescriptorExtractor extractor;
316   extractor.compute( sceneMat, keypointsS, descriptors_scene);
317   extractor.compute( objectMat, keypoints0, descriptors_object);
318
319   // Step 3: Match descriptors
320   // Declaring matcher. Choose between FlannBased or BruteForce
321   FlannBasedMatcher matcher; // Flann based
322   //BFMatcher matcher(NORM_L2); // Brute force. Option NORM_L1 or NORM_L2
323   descriptors_scene.size(), keypoints0.size(), keypointsS.size());
324   // Match pictures
325   vector< vector< DMatch > > matches;
326   matcher.knnMatch(descriptors_object, descriptors_scene, matches, 2); // Matching two nearest
327   neighbors
328
329   // Filter only good matches
330   vector< DMatch > good_matches;
331   good_matches.reserve(matches.size());
332
333
334   for (size_t i = 0; i < matches.size(); ++i)
335   {
336     if (matches[i].size() < 2)
337       continue;
338
339     const DMatch &m1 = matches[i][0];
340     const DMatch &m2 = matches[i][1];
341
342     if (m1.distance <= nndrRatio * m2.distance)
343       {good_matches.push_back(m1);}
344   }
345
346   cout << endl << endl << "MATCHES: " << matches.size();
347   cout << endl << "GOOD MATCHES: " << good_matches.size() << endl;
348
349   //----- SIFT END -----
350
351   if ((good_matches.size() >= 7)) // Object found if more than seven good matches
352   {
353     cout << "OBJECT FOUND!" << endl;

```

```

354     vector< Point2f > obj;
355     vector< Point2f > scene;
356
357     for (int i = 0; i < good_matches.size(); i++)
358     {
359         obj.push_back( keypoints0[ good_matches [i].queryIdx ].pt);
360         scene.push_back( keypointsS[ good_matches [i].trainIdx ].pt);
361     }
362
363     // Find homography between matched keypoints
364     Mat H = findHomography( obj, scene, CV_LMEDS); //CV_RANSAC also an option
365
366     // Get corners from object image
367     vector< Point2f > obj_corners(4);
368     obj_corners[0] = cvPoint(0, 0);
369     obj_corners[1] = cvPoint(objectMat.cols, 0);
370     obj_corners[2] = cvPoint(objectMat.cols, objectMat.rows);
371     obj_corners[3] = cvPoint(0, objectMat.rows);
372
373     perspectiveTransform( obj_corners, scene_corners, H );
374
375     // Display image and draw lines around detected object
376     img_show = sceneMat;
377     line( img_show, scene_corners[0] , scene_corners[1], Scalar(0, 255, 0), 2 );
378     line( img_show, scene_corners[1] , scene_corners[2], Scalar(0, 255, 0), 2 );
379     line( img_show, scene_corners[2] , scene_corners[3], Scalar(0, 255, 0), 2 );
380     line( img_show, scene_corners[3] , scene_corners[0], Scalar(0, 255, 0), 2 );
381
382     imshow("Matches", img_show);//img_show
383     cv::waitKey(3);
384
385     cout << "Frame rate = " << getTickFrequency() / (getTickCount() - t0) <<
386     endl;
387
388
389     Mat transformation = calcTransformMatrix();
390     //cout << endl << "delta_Tc " << transformation << endl;
391     calcStepwiseMovement(transformation);
392
393     // Set max speed
394     float max_pos_correction = 0.7, max_rot_correction = 0.5;
395     float steps[6] = {0,0,0,0,0,0};
396     for (int i = 0; i <= 5; i++)
397     {
398         if (i < 3)
399         {
400             if (desired_movement.next_point[i] > max_pos_correction)
401             {
402                 steps[i] = abs(desired_movement.next_point[i]/
403                 max_pos_correction);
404             }
405         }
406         if (i>=3)
407         {
408             if (desired_movement.next_point[i] > max_rot_correction)
409             {
410                 steps[i] = abs(desired_movement.next_point[i]/
411                 max_rot_correction);
412             }
413         }
414     }
415
416     // Find largest step
417     float temp = 0;
418     for (int i = 0; i <= 5; i++)
419     {
420         if (steps[i] > temp)
421             temp = steps[i];
422     }
423
424     // Divide all motions by largest step

```

```

425         if(temp > 0)
426         {
427             for (int i = 0; i <= 5; i++)
428                 desired_movement.next_point[i] = desired_movement.next_point[i]/
temp;
429         }
430
431         //----- Position lock command -----
432
433         /**
434         bool position_lock = false;
435         if(desired_movement.max_offset_pos[0] < 5)
436         {
437             if(desired_movement.max_offset_rot[0] < 2)
438                 position_lock = true;
439         }
440
441         if((desired_movement.max_offset_pos[0] > 10)
442             || (desired_movement.max_offset_rot[0] > 4))
443             position_lock = false;
444
445         if(position_lock == true)
446         {
447             desired_movement.next_point.clear();
448             for(int i = 0; i <= 5; i++)
449                 desired_movement.next_point.push_back(0);
450         }*/
451
452         // -----
453
454         Publish();
455     }
456     else
457     {
458         imshow("Matches", sceneMat);
459         cv::waitKey(3);
460         // Publish empty desired position if no object is found
461
462         desired_movement.next_point.clear();
463         for (int i = 0; i <= 5; i++)
464             {desired_movement.next_point.push_back(0);}
465         cout << "OBJECT NOT FOUND!" << endl;
466         Publish();
467     }
468 }
469 };
470
471
472
473
474 int main(int argc, char** argv)
475 {
476     ros::init(argc, argv, "objectDetection");
477     ImageConverter ic;
478     ros::spin();
479     return 0;
480 }

```



```

1  /**
2  Arduino source code
3  Receives distance to object over ROS and moves stepper motor into position*/
4
5  #include <ros.h>
6  #include "std_msgs/Float32MultiArray.h"
7  #include <Stepper.h>
8
9
10
11 // create a variable stepper
12 Stepper stepper(200 ,10 ,11 ,12 ,13);
13
14 // create ROS node for Arduino
15 ros::NodeHandle nh_ar;
16
17
18 std_msgs::Float32MultiArray test;
19
20 // Receives distance from ROS
21 void messageCb( const std_msgs::Float32MultiArray& msg){
22     test = msg;
23
24     digitalWrite(13, HIGH-digitalRead(13)); // blink the led
25
26     int distance = (int)test.data[5];
27
28     int steps = (distance-300)*1.25; // Set the referance distance to 300 mm and
29                                     // Kp = 1.25 of P-regulator
30
31     moveMotor(steps);
32 }
33
34 // Create subscriber to ROS topic "objectDetection/objectLocation"
35 ros::Subscriber<std_msgs::Float32MultiArray> s("objectDetection/objectLocation", &messageCb);
36
37
38 void setup()
39 {
40     // Code : set up communication with the PC
41     Serial.begin(9600);
42     // Code : set the RPM of the stepper to 60
43     stepper.setSpeed(60);
44
45     pinMode(13, OUTPUT);
46     nh_ar.initNode();
47     nh_ar.subscribe(s);
48 }
49
50
51 void moveMotor(int steps)
52 {
53     int direction = steps/abs(steps);
54     for (int i = 0; i <= abs(steps); i++)
55     {
56         stepper.step(direction);
57     }
58 }
59
60
61 void loop()
62 {
63     nh_ar.spinOnce();
64     delay(10);
65 }

```



## Appendix C

# Digital Appendix

A .zip file is included as digital appendix. This contains:

- The video *agilus\_visual\_servoing.avi* of the object tracking in the KUKA Agilus robot cell at *Department Of Production and Quality Engineering*.
- The two ROS packages *eirik\_vision* and *kuka\_driver* used for object detection, calculation and communication with the KR C4 KUKA controller. A README.txt file with instructions is found in the *eirik\_vision* package.
- Source code for the Arduino stepper motor.
- RSI files configured for cartesian commands. These have to be located in the folder "C:\KRC\ROBOTER\Config\User\Common \SensorInterface" on the the KUKA KR C4 controller for the RSI interface to work.
- The KUKA controller program *RSI\_Ethernet.src*, that is run on the Agilus during tracking.
- Siemens PLC configuration for the ProfiSAFE system.
- Files for the VisualComponents robot cell layout.



Appendix D

Pre-Study Report





**NTNU – Trondheim**  
Norwegian University of  
Science and Technology

# Vision Based Robotic Control

Eirik Anfindsen Solberg

February 2014

PRE-STUDY REPORT

Subsea Technology

Department of Production and Quality Engineering  
Norwegian University of Science and Technology

Supervisor: Professor Olav Egeland

# Contents

- 1 Introduction** **2**
- 1.1 Background . . . . . 2
- 1.2 Objectives . . . . . 2
- 1.3 Limitations . . . . . 2
  
- 2 Method** **4**
- 2.1 Literature . . . . . 4
- 2.2 Project management . . . . . 4
- 2.3 Milestones . . . . . 5
  
- A Gantt diagram** **6**
  
- B Assignment text** **8**



# Chapter 1

## Introduction

This is a pre-study report used to define the work ahead, and to set the objectives for my master thesis in *Subsea Technology* at *NTNU*. The project will be concluded in June 2014 with a finishing thesis.

### 1.1 Background

Remote operations is an important part of most subsea operations. The fact that the industry has grown more concerned with decompression sickness and other hazardous events a diver might be subjected to, has led to a decrease of acceptable diving depth. At the same time, oil companies are operating at deeper water than ever before. This leads to a higher demand for remotely operated equipment, and one type is the ROV (Remotely Operated Vehicle).

The main tool of an ROV is the robotic manipulator(s). These are manually controlled by an operator from the surface. This setup can be challenging with regards to the depth vision and as a result often two ROVs is needed for certain operations. A system that can recognize certain objects and automatically move into position to perform an operation would make this work easier for the operator.

### 1.2 Objectives

The main objectives of this Master's project are

1. Set up the Agilus robots and test simple control features.
2. Create a model of the robotic cell in a graphic simulation system, and test simple movements in this.
3. Set up a robotic vision system for one of the robots.
4. Set up a demonstration demo were one robot holds a moving target and the other one follows the movements.

### 1.3 Limitations

The work will be performed in the lab at the *Department of Production and Quality at Valgrinda*. This is a dry facility and all the experiments will be conducted with no water present.

The result will still be relevant for subsea operation, as marinisation of the equipment is seen as a separate development step.

## Chapter 2

### Method

The work in this project will be a combination of theory, programming and experimental work. The different phases are; selection of vision techniques and algorithms, assembly and setup of robots, testing, computer simulation and programming. In the end there will be a test setup of the different functions and a project report written to document the work.

PhD candidate Lars Tingelstad is working on a project in a larger robotic cell next to the one I am using. We will cooperate on some of the matters, especially in the setup phase. There are also other people working with robotic vision in close proximity to my work station. It is believed that they can answer some of the questions I will have when working with the programming.

ROS (The Robot Operating System) will be used for sensor input such as camera. The programming will also be done with this system. I will need to spend time doing tutorials and getting familiar with the program environment.

#### 2.1 Literature

My project report from TPK-4510 *Robotisert sveising for undervannssystemer* written in the fall 2013 contains work on robotic vision. In addition to the experience I have from this work I will need other sources to gather information. This includes:

- *Robotics - Modelling, Planning and Control* used in the course TPK-4170
- *Robotics, Vision and Control* by Peter Corke
- *Absolute C++* by Walter Savitch
- Relevant scientific articles

#### 2.2 Project management

A Gantt diagram has been created, showing the planned progress of the project. The main purpose of this is to investigate if there will be sufficient time to finish each part. Some phases of the work can run simultaneously while others are dependent on the completion of an earlier phase. The programming of the robot will start when the robot assembly and testing is finished.

I am participating in a course at Svalbard the week before Easter. There will therefore not be done any work in this period, as seen in the Gantt diagram. This will be compensated by increased activity in the rest of the period.

### **2.3 Milestones**

The following milestones will have great significance for the outcome of the project.

- Hand in pre-study report (04. February)
- Finish setup and testing of robots (14. February)
- Implement communication between camera, ROS (Robot Operating System) and KUKA Agilus robots (28. February)
- Create a demo for vision based robotic control (20. May)
- Finish and hand in master thesis (06. June)

Appendix A

**Gantt diagram**

