



Norwegian University of
Science and Technology

Mobile Autonomous Robot: Remote Operation

Tommy Berntzen

Master of Science in Cybernetics and Robotics

Submission date: June 2017

Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Title: Mobile autonomous robot: Remote operation

Student: Tommy Berntzen

Problem description

The goal of this thesis is to have a more complete system for a mobile autonomous robot with a robot manipulator. This work is a continuation of several theses and projects conducted on the same system previously, and a direct continuation of the specialization project completed during the authors penultimate semester. The main focus is the unification of previously found solutions, so that everything operates on the same platform, and also re-implementation of the solution for the robot manipulator, using the methods found during said project. In addition, a user manual was found to be desirable, with the intent of making future work and implementations easier. This should include a condensed guide to hardware and software used. To complete these tasks, the student should:

- Use a suitable method to migrate previous solution to one on-board computer.
- Implement a concept for control of the robot manipulator through Robot Operating System.
- Explore viable concepts for a computer vision system for producing depth maps using the available hardware.
- Expand Operator Control Station concept started during the preceding work.
- Write a user manual for the whole robot system.

Supervisor: Tor Engebret Onshus

Summary and conclusion

Summary

This thesis aims to develop a concept for unifying previous solutions on the topic of robotized maintenance and remote operation, as well as exploring possible solutions for a stereo vision system to facilitate the development of a collision avoidance system for use with the robot manipulator, which is a part of the robot system. There has also been a development of a concept for an Operator Control Station, intended to give the user remote presence when executing various tasks and production of a user manual for the whole system.

The system framework is based on Robot Operating System, serving as a server on the on-board computer. The whole system is divided into a server and a client, where the server is located on the mobile robot and the client is a remote computer acting as an Operator Control Station. The server and client communicate wirelessly through a LAN.

The robot manipulator, a SCORBOT-ER 4u manufactured by Intelitek, makes use of proprietary software and hardware for communication with and control from a computer, making the manipulator non-compatible with operating systems other than Microsoft Windows. Thus there was a need for an interface in order to link the manipulator to the rest of the ROS system. To this end, MATLAB was used to create an interfacing node for handling commands sent to and from the robot manipulator.

A stereo vision system for producing depth maps was explored, using the OpenCV library and two TP-Link IP cameras, which were the camera setup that was available at the time of writing this thesis. The purpose of this was

to find a viable solution for facilitating a collision avoidance system for the robot manipulator.

The Operator Control Station was developed on the Qt framework, and was a continuation of previous attempts for implementing a graphical user interface. It is divided into two modes, the Drive Mode, which is intended to facilitate a previous solution for autonomous mapping and localization using SLAM, and the Manipulator Mode, which shows the control of the robot manipulator with a joystick. Sensory feedback is given in both modes, and consists of direct camera feeds and produced maps.

Lastly a user manual, called MAR User Manual, was written. It is produced as a stand-alone document, and is meant to lighten the burden of research when starting new projects on the system.

Conclusion

The previous solutions were successfully migrated to one on-board computer through the use of a virtual machine, and an interface in MATLAB successfully connects the robot manipulator to the ROS system. A possible solution for a stereo vision system was found, and it was concluded that satisfactory results using the current hardware was not possible. A working concept for an Operator Control Station was implemented for the robot manipulator, with satisfactory performance. The user manual was produced to a satisfactory standard. Thus, the conclusion is that the overall goals of this thesis was met.

Sammendrag og konklusjon

Sammendrag

Denne oppgaven tar sikte på å utvikle et konsept for å forene tidligere løsninger rundt temaet robotisert vedlikehold og fjernoperasjon, samt å utforske løsninger for et stereosynsystem for å legge til rette for utviklingen av et kollisjonsunngåelsessystem for bruk med robotmanipulatoren, som er en del av robotsystemet. Det har også blitt gjennomført utvikling av et konsept for en operatørkontrollstasjon (Operator Control Station), som skal gi brukeren ekstern tilstedeværelse ved utførelse av ulike oppgaver, og produksjon av en brukerhåndbok for hele systemet.

Systemrammen er basert på Robot Operating System, som tjener som server på innebygd datamaskin. Hele systemet er delt inn i en server og en klient, hvor serveren befinner seg på mobilroboten og klienten er en ekstern datamaskin som fungerer som operatørkontrollstasjon. Serveren og klienten kommuniserer trådløst via et LAN.

Robotmanipulatoren, en SCORBOT-ER 4u produsert av Intelitek, benytter seg av proprietær programvare og maskinvare for kommunikasjon med og kontroll fra en datamaskin, noe som gjør at manipulatoren ikke kompatibel med andre operativsystemer enn Microsoft Windows. Dermed var det behov for et grensesnitt for å koble manipulatoren til resten av ROS-systemet. Til dette formål ble MATLAB brukt til å opprette en grensesnittsnode for håndtering av kommandoer sendt til og fra robotmanipulatoren.

Et stereosynsystem for å produsere dybdekart ble utforsket ved hjelp av OpenCV-biblioteket og to TP-Link IP-kameraer, som var det av kamerasystemer som var tilgjengelig under arbeidet med denne oppgaven. Hensikten med dette

var å finne en levedyktig løsning for å legge til rette for et system for kollisjonsunngåelse for robotmanipulatoren.

Operatørkontrollstasjonen ble utviklet ved hjelp av Qt-rammeverket, og var en videreføring av tidligere forsøk på å implementere et grafisk brukergrensesnitt. Den er delt inn i to moduser, Drive Mode, som skal legge til rette for en tidligere løsning for autonom kartlegging og lokalisering ved hjelp av SLAM, og Manipulator Mode, som viser styringen av robotmanipulatoren med en joystick. Sensorisk tilbakemelding er gitt i begge tilfeller, og består av direkte kameraoverføringer og eventuelle produserte kart.

Til slutt ble en brukerhåndbok, kalt MAR User Manual, skrevet. Den er produsert som et frittstående dokument, og er ment å lette mengden av forskning som trengs for å komme i gang når nye prosjekter på systemet eventuelt startes.

Konklusjon

De forrige løsningene ble overført til en innebygd datamaskin ved bruk av en virtuell maskin, og et grensesnitt i MATLAB forbinder robotthåndtereren til ROS-systemet på en tilfredsstillende måte. En mulig løsning for et stereosystem ble funnet, og det ble konkludert med at tilfredsstillende resultater ved bruk av gjeldende maskinvare ikke var mulig. Et fungerende konsept for en operatørkontrollstasjon ble implementert for robotmanipulatoren, med godkjent ytelse. Brukerhåndboken ble produsert til en tilfredsstillende standard. Dermed konkluderes det med at de overordnede målene for denne avhandlingen oppfylt.

Preface

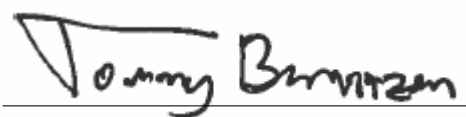
This Master's Thesis was written during the spring of 2017 at the Norwegian University of Science and Technology, Department of Engineering Cybernetics. It concludes the 10th and final semester of the 5 year Master's degree program Cybernetics and Robotics.

This work is a continuation of previous work done on the robot system presented here. The overarching goal is to have a fully autonomous robot system, with remote operation and control capabilities. This thesis contributes to that end by exploring stereo vision systems using OpenCV for using with a robot manipulator, as well as facilitating joystick control of the manipulator through the Robot Operating System (ROS) framework.

One of the main challenges when starting was the vast variety of solutions to build from, and proprietary software that needed to be circumvented in order to use the ROS framework.

I want to thank my supervisor Tor Onshus for guidance and talks. I also want to thank my fellow students, for interesting conversations and discussions throughout the last year. Lastly, I want to thank my girlfriend Madelen for all the support and motivation during this final semester.

It is assumed that the readers of this Thesis have some background in technology, and a basic understanding of programming and electronics.



Tommy Berntzen

Trondheim, June 2017

Contents

| | |
|--|-------------|
| Summary and conclusion | i |
| Sammendrag og konklusjon | iii |
| Preface | v |
| List of Figures | xi |
| List of Acronyms | xiii |
| 1 Introduction | 1 |
| 1.1 Motivation | 1 |
| 1.2 Prerequisites and starting point | 3 |
| 1.2.1 Preceeding work | 3 |
| 1.3 Equipment and software | 5 |
| 1.3.1 Hardware | 5 |
| 1.3.2 Software | 6 |
| 1.4 Thesis structure | 7 |
| 1.4.1 Appendices | 8 |
| 2 System concept | 9 |
| 2.1 Introduction | 9 |
| 2.2 Robot Operating System | 9 |
| 2.3 Implementation outline | 12 |
| 3 Robot manipulator | 13 |
| 3.1 Introduction | 13 |
| 3.2 Background theory | 13 |

| | | |
|----------|--|-----------|
| 3.2.1 | Forward kinematics | 13 |
| 3.2.2 | Inverse kinematics | 15 |
| 3.3 | Using a virtual machine | 15 |
| 3.3.1 | VM networking | 16 |
| 3.4 | Control through ROS | 18 |
| 3.4.1 | Communicating with the USB Controller | 18 |
| 3.4.2 | Sending commands from ROS | 19 |
| 4 | Stereo vision system | 21 |
| 4.1 | Introduction | 21 |
| 4.2 | Exploration of possible solutions | 21 |
| 4.3 | OpenCV | 24 |
| 4.4 | Background theory | 24 |
| 4.4.1 | Epipolar geometry | 25 |
| 4.4.2 | The essential and fundamental matrices | 27 |
| 4.4.3 | Undistortion | 29 |
| 4.4.4 | Calibration | 29 |
| 4.4.5 | Rectification | 30 |
| 4.4.6 | Disparity map and triangulation | 31 |
| 4.5 | Camera setup and calibration | 32 |
| 4.5.1 | Camera setup | 32 |
| 4.5.2 | Calibrating the cameras | 32 |
| 4.6 | Depth map and collision avoidance | 34 |
| 4.6.1 | Producing a depth map | 34 |
| 4.6.2 | The pan-tilt unit | 36 |
| 4.6.3 | Collision avoidance | 36 |
| 5 | Operator Control Station | 39 |
| 5.1 | Introduction | 39 |
| 5.2 | Qt | 39 |

CONTENTS

| | | |
|-----------|--|-----------|
| 5.3 | Main menu | 41 |
| 5.4 | Manipulator mode | 42 |
| 5.4.1 | Joystick | 43 |
| 5.5 | Drive Mode | 43 |
| 6 | MAR User Manual | 45 |
| 7 | Results | 47 |
| 7.1 | MATLAB interface | 47 |
| 7.2 | Stereo vision system | 47 |
| 7.3 | OCS | 49 |
| 7.4 | User manual | 50 |
| 8 | Discussion | 51 |
| 8.1 | General assessment of the system | 51 |
| 8.1.1 | Development tools | 51 |
| 8.2 | USB Controller interface | 52 |
| 8.2.1 | Assessment | 52 |
| 8.2.2 | Weaknesses | 52 |
| 8.3 | Stereo vision system | 53 |
| 8.4 | OCS | 54 |
| 8.4.1 | Joystick | 54 |
| 8.5 | User manual | 55 |
| 9 | Recommendations and further work | 57 |
| 9.1 | Stereo vision system | 57 |
| 9.2 | Moving the OCS to Ubuntu | 58 |
| 9.3 | Joystick | 59 |
| 9.4 | Eliminating the USB Controller | 59 |
| 10 | Bibliography | 61 |

| | | |
|----------|---|-----------|
| A | MAR user manual index | 67 |
| B | DVD Contents | 69 |
| C | Camera matrices | 71 |
| C.1 | Intrinsic matrices | 71 |
| C.2 | Extrinsic matrices | 71 |
| D | Installation and configuration | 73 |
| D.1 | Hardware setup | 73 |
| D.2 | Installation | 73 |
| D.3 | Configuring the project | 74 |
| D.3.1 | ROS workspace | 74 |
| D.3.2 | Virtual machine | 74 |
| D.3.3 | Network configuration | 74 |
| D.4 | System launch | 74 |
| E | Troubleshooting | 77 |
| E.1 | Connecting to the Intelitek USB Controller from virtual machine | 77 |
| E.2 | Pan-tilt unit | 77 |

List of Figures

| | | |
|------|---|----|
| 1.1 | Platform concept | 2 |
| 2.1 | System concept | 9 |
| 2.2 | Example of a ROS setup | 11 |
| 2.3 | Abstracted view of the robot manipulator system | 12 |
| 3.1 | MATLAB interface concept | 18 |
| 3.2 | MATLAB communication concept | 19 |
| 3.3 | MATLAB node sample | 20 |
| 4.1 | stereo_image_proc | 22 |
| 4.2 | MATLAB Stereo Calibrator | 23 |
| 4.3 | Epipolar geometry | 26 |
| 4.4 | Essential matrix | 27 |
| 4.5 | Stereo rectification | 30 |
| 4.6 | Triangulation | 31 |
| 4.7 | Finding corners of chessboard | 33 |
| 4.8 | Rectified images | 34 |
| 4.9 | CvBridge | 34 |
| 4.10 | Camera view illustration | 36 |
| 5.1 | OCS Main menu | 41 |
| 5.2 | OCS Manipulator mode | 42 |
| 5.3 | OCS Drive mode | 44 |
| 7.1 | Stationary depth map | 47 |
| 7.2 | Original image | 47 |
| 7.3 | Real-time stationary object | 48 |
| 7.4 | Real-time original image | 48 |

LIST OF FIGURES

| | | |
|-----|--------------------------------------|----|
| 7.5 | Real-time object in motion | 49 |
| 7.6 | Real-time original image | 49 |
| 7.7 | Manipulator Mode activated | 49 |

List of Acronyms

API application programming interface. 24

DOF degrees of freedom. 14

GUI graphical user interface. 39

HID human interface devices. 43

IDE integrated development environment. 6

MAR mobile autonomous robot. 3

MTIS MATLAB Toolbox for the Intelitek Scorbot. 18

OCS Operator Control Station. 3

POV point of view. 36

PTU pan-tilt unit. 5

ROS Robot Operating System. 4

SLAM simultaneous localization and mapping. 4

SSD solid-state drive. 3

VM virtual machine. 12

Introduction

1.1 Motivation

Global oil and gas demand continue to grow moderately [20], and there is now a need to obtain the resources from more hostile and remote environments [10]. Thus the oil and gas companies are looking to lower production cost and improve efficiency, quality and safety [10]. The potential for extensive use of automation and robot technology is evident [3].

The oil and gas industry often presents harsh and even hazardous environments for their workers, often resulting in injuries or worse [16]. Robotic systems could be designed to withstand most of the environmental hazards, and would also be considered to be disposable, limiting the risk of platform accidents to be purely economical. Designing the robots to work 24/7 could also lower the risk of accidents altogether [3], increasing the accuracy and work load capability when compared to normal human workers.

Development of robotic systems for topside oil & gas facilities as an alternative to subsea development has been extensively researched both academically and industrially. According to studies, the prospect of developing unmanned wellhead platforms seem promising [33][4]. The concept is already in wide use on the Danish and Dutch shelves, but the concept is new for use on the Norwegian shelf [29]. The concept of normally-unmanned automated topside platforms has also been proposed as an alternative for larger maintenance operations[1], as shown in Fig 1.1. This concept corresponds to the Type 0 platform proposed by Ramboll in their report [33], a complex platform that would include a helideck and fire water systems.

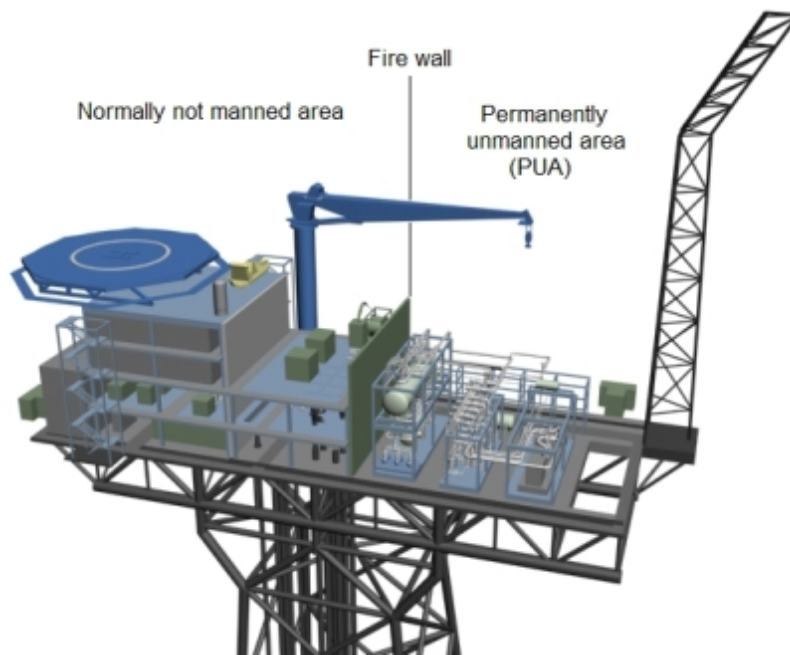


Figure 1.1: Platform concept, image from [1]

The most important scheduled operations on an offshore oil and gas facility include inspection monitoring and maintenance [10]. This means that the robot must be able to navigate autonomously or semiautonomously, be able to manipulate certain objects either autonomously or by teleoperation, be equipped with sensors for position tracking and perception of surroundings and be able to relay information to a control station, preferably through wireless communication [10].

This Master's Thesis aims to contribute towards providing a workable concept with regards to the topics discussed in this section. Focus on environmental hazards and physical robustness of the robot system is a major part of construction of such solutions for practical use [10]. This will however not be addressed here, as the focus is on developing concepts and software solutions in a closed environment.

1.2 Prerequisites and starting point

The project thesis written in the 9th semester[8], forms the basis for this Master's thesis. It was decided that the robotic system, hereby called the mobile autonomous robot (MAR), would benefit from being migrated onto one on-board computer, both for convenience and efficiency. This means upgrading the solid-state drive (SSD) and reimplementing the solution for the robot manipulator based on the solutions discussed in the aforementioned project thesis. The hope is that this, along with the production of a user manual for the whole system, will result in a full and usable system, and also facilitate further development and parallel solutions. The current Operator Control Station (OCS) also needs an overhaul, which will be addressed in this thesis. Thus the main focus of this thesis will be the development of a concept for control of the robot manipulator, exploration of a possible stereo vision system using the current camera system, the development of a concept for a more complete OCS, and the production of a user manual.

1.2.1 Preceding work

There has been a lot of theses written on or about the robot system used in this thesis. The concept was started as early as 2005, then as a standalone robot manipulator, through to the mounting of the robot manipulator on a mobile platform in 2013. This work is a direct continuation of the Project Thesis written in the 9th semester, in the fall of 2016 [8]. As a part of that thesis, there was given an evaluation of some of the previous works that might be used as a basis for further work. A short recap of some of the earlier work that has been done on this system will be given here, for convenience and completeness. For a more complete survey of the previous work, please consult the aforementioned project thesis.

Robot manipulator and telepresence: The system is built around the

SCORBOT-ER 4u. Kristian Bekken implemented a system for telepresence and collision avoidance for the robot manipulator in 2010[6], which was a continuation of work dating back as far as 2005 [21].

Building the mobile platform: The robot manipulator was mounted on a mobile platform in 2013, being the focus of the Master's thesis written by Petter Aspunvik [5]. This thesis has served as a manual for part of the connections and functionality of the robot system.

Simultaneous localization and mapping (SLAM): Parallel to Aspunvik's building of the mobile platform, Mikael Berg developed a system for SLAM written in the Go programming language [7]. The solution runs within Windows 7, and was deployed on the on-board computer.

SLAM and Robot Operating System (ROS): During the spring of 2016, Vegard Lindrup developed a system for SLAM, configured to run on ROS [23]. The solution uses RTAB-Map to survey the surroundings, and a built navigation stack to navigate autonomously against easy targets. He also developed functional concepts for remote control over Bluetooth, and an OCS developed in Qt.

Oculus Rift and Leap Motion: During the spring of 2016, parallell to the work of Vegard Lindrup, Ole Magnus Siqveland developed a concept for using virtual reality with remote operations [47]. The solution uses the Oculus Rift and Leap Motion, a joystick and the Microsoft Kinect to perform remote operations. The solution is not interesting in regards to this thesis, but is included for completeness, and the fact that it might be interesting for future work.

1.3 Equipment and software

A short list of the hardware and software used in this specific thesis will be presented here. As a part of this Master's thesis, a user manual for the robot system in question has been written. For a full overview of the hardware available on the robot system, consult the MAR User Manual, located on the DVD in the folder MAR User Manual. The index of the user manual is found in Appendix A.

1.3.1 Hardware

SCORBOT-ER 4u:

The SCORBOT-ER 4u is a robot manipulator designed for educational purposes, by having an open structure for easy access. It has five axes and a servo gripper, with optical encoders for each axis for feedback. Produced by Intelitek.

USB Controller:

The USB Controller is designed by Intelitek to provide advanced control features to the SCORBOT-ER 4u, connected to a computer through USB. It controls the 24V power supply to the manipulator motors, and reads the encoder and microswitch signals on the robot manipulator. It is connected to a computer via USB, and to the SCORBOT-ER 4u through a proprietary

Pan-tilt unit (PTU):

Produced by Directed Perception (now FLIR), communicates with the on-board computer through a control box.

IP cameras:

Two TP-Link TL-SC3430. They are H.264 Megapixel surveillance cameras, and support video and audio transfer over most transfer protocols.

The viewing angle for each camera is 46° horizontally and 35° vertically.

Routers:

Two TP-Link TL-WR841N, 300Mbps Wireless N Routers.

Joystick:

Logitech Force 3D Pro. Supports movement in x-, y- and z-directions, as well as having twelve configuration buttons, two fire-buttons and a point-of-view hat.

1.3.2 Software

List of used software, frameworks, libraries and integrated development environments (IDEs), with specific packages or usage stated when deemed necessary.

- MATLAB 2015b, toolboxes:
 - Robotics System Toolbox
 - ScorBotToolbox [25]
- ROS Indigo Igloo, additional packages:
 - web_video_server
 - flir_ptu_driver
 - cv_bridge
- Qt 5.7
- Microsoft Visual Studio 2015 (for the C++ compiler, using MSVC 15 with Qt)
- OpenCV 2.4

1.4 Thesis structure

The thesis is divided into eight chapters covering various topics. Chapters 3-6 are structured as modules covering the respective goals presented in the problem description. In addition to this there is an enclosed DVD containing the implementation files and relevant documents.

Chapter 2 - System concept

This chapter will present the overall concept of the complete MAR, before outlining the concept for this specific thesis. It will also give a short introduction to the ROS framework used.

Chapter 3 - Robot manipulator

This chapter presents some background theory for control of a robot manipulator, as well as the implementation and work that was done concerning this.

Chapter 4 - Stereo vision system

This chapter presents the exploration of possible solutions for creating a stereo vision system, using the available hardware, and a brief introduction to the method that was chosen, as well as showing the implementation attempt and procedure.

Chapter 5 - Operator Control Station

This chapter will give a brief introduction to the framework used to create the OCS, and present the OCS concept with the intended functionality.

Chapter 6 - MAR User Manual

This chapter presents the motivation behind creating the user manual, as well as an overview of the content of the finished product.

Chapter 7 - Results

This chapter presents the results of the various implementations.

Chapter 8 - Discussion

In this chapter, the implementations and results will be discussed, arguing for the degree of success of the solutions.

Chapter 9 - Recommendations and further work

This chapter will present possible future work, and recommend changes to the system to further improve the solutions.

1.4.1 Appendices

Appendix A - MAR user manual indexes

The full index of the MAR user manual.

Appendix B - DVD Contents

Description of the contents of the enclosed DVD.

Appendix C - Camera matrices

Camera matrices obtained during calibration.

Appendix D - Installation and configuration

Guide to installing, configuring and running the system

System concept

2.1 Introduction

This chapter will give a brief outline of the modules implemented and the system as a whole. It will start with a short introduction to ROS, which is the base framework for the work and implementations, before presenting the aforementioned modules. The abstracted system concept for the full MAR is shown in Figure 2.1

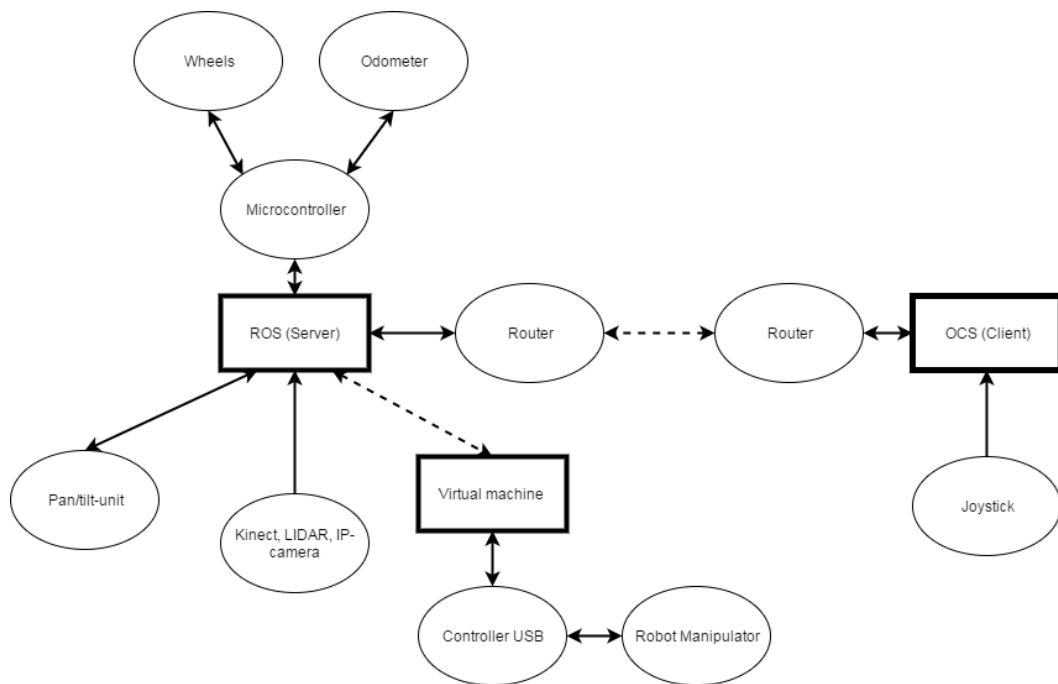


Figure 2.1: System concept

2.2 Robot Operating System

ROS is a flexible framework for writing robot software [35]. The foundation of ROS is a message passing interface providing inter-process communica-

tion, referred to as middleware [39]. The built-in messaging system manages all the details between distributed nodes via an anonymous and asynchronous publish/subscribe mechanism. An example of the way message passing and node registration works is found in Figure 2.2.

ROS acts like a meta-operating system, providing services such as hardware-abstraction, low-level device control, package management and implementation of commonly-used functionality. Language independence, or the ease of implementing the framework in any modern programming language, is an overall goal of the ROS framework, and it is already implemented in C++, Python and Lisp, with experimental libraries in Java and Lua [43]. The main concepts in ROS is divided into three levels, the Filesystem level, the Computation Graph level, and the Community level [38]:

- The **Filesystem level** consists of
 - *Packages*: the main unit for organizing software. May contain nodes, libraries, datasets, and so on.
 - *Metapackages*: Specialized packages which represents a group of related packages.
 - *Package manifests*: Metadata about a package.
 - *Repositories*: Collection of packages which share a common VCS system.
 - *Message (msg) types*: Message description that defines the data structures for messages in ROS.
 - *Service (srv) types*: Service descriptions, defines the request and response data structures for services in ROS.
- The **Computation Graph level** consists of:
 - *Nodes*: Processes that perform computation.

- **Master:** Provides name registration and lookup to the rest of the Computation Graph.
 - *Parameters Server:* Allows data to be stored by key in a central location, part of the Master.
 - *Messages:* Data structures for communication between nodes.
 - *Topics:* A name that is used to identify the content of a message. I.e. a node interested in a specific type of data subscribes to the appropriate topic.
 - *Services:* Defined by a pair of message structures, designed for request/reply interactions, not provided by the many-to-many, one-way transport given by the publish/subscribe model.
 - *Bags:* Format for saving and playing back ROS message data.
- The **Community level** consists of the ROS distributions, repositories with code provided by various institutions, Wiki pages and general community communication (forums, mailing lists, etc.).

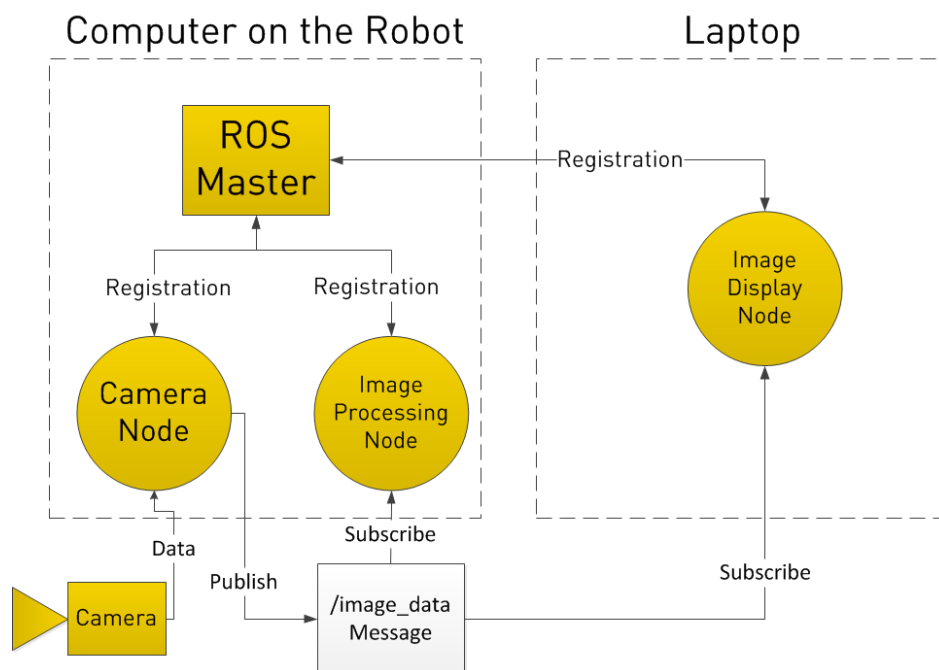


Figure 2.2: Example of a ROS setup, image from [11]

For a more thorough look at ROS and its functionality, the reader should consult Lindrup [23], or the ROS Wiki itself [44].

2.3 Implementation outline

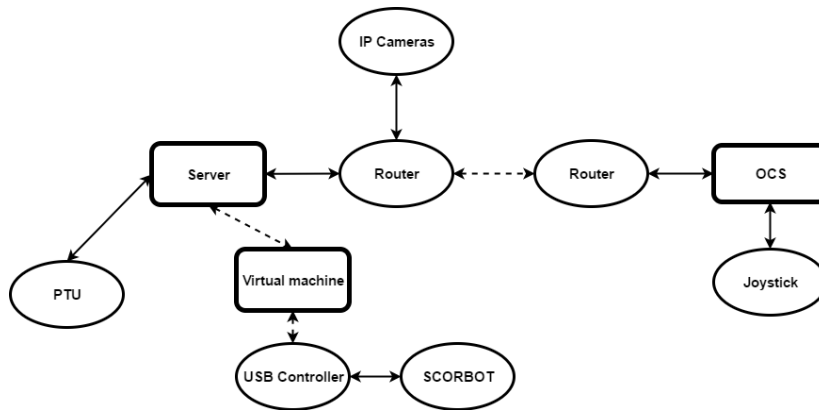


Figure 2.3: Abstracted view of the robot manipulator system

The abstracted outline of the robot manipulator system is shown in Figure 2.3. The main parts are the Server, which is the on-board computer running ROS, the OCS, located on a remote computer, and the virtual machine (VM), facilitating the communication between the ROS system and the SCORBOT-ER 4u.

Robot manipulator

3.1 Introduction

This chapter will present some theory behind the control of a robot manipulator, as well as presenting the proposed solution for controlling the robot manipulator, a SCORBOT-ER 4u, through the ROS framework. First, there is an explanation of what was done to relocate the solution to the on-board computer, which is one of the project goals. Next there is a presentation of the developed concept for control, using MATLAB.

3.2 Background theory

This section will present a short summary of some of the key concepts regarding robot kinematics, included for completeness and convenience. For a more thorough look at the theory behind kinematics, the reader is referred to Bekken [6] and Berntzen [8]. Presented is an introduction to the two main concepts, namely forward and inverse kinematics, which is based on the book *Robot modeling and Control* [48]. Rotation matrices and coordinate system representations will not be presented here, and the reader should consult the aforementioned book for in-depth review of such topics.

3.2.1 Forward kinematics

A robot manipulator can be seen as a set of links connected by joints, which are either revolute (rotary) or prismatic (linear). The concept of forward kinematics is to represent the position and orientation of the end effector, the end

of the robot arm interacting with the environment, in terms of the joint angles/positions. As the SCORBOT-ER 4u is an articulated arm (all revolute joints), the text will assume rotary capability when mentioning joints from this point on. The joint configuration is usually defined relative to a fixed coordinate system, called the base frame, to which all objects are referenced. A simple example of this is given by a two-link planar robot, with two revolute joints, i.e. 2 degrees of freedom (DOF). The (x,y) coordinates of the tool would then be expressed as

$$\begin{aligned}x &= a_1 \cos \theta_1 + a_2 \cos(\theta_1 + \theta_2) \\y &= a_1 \sin \theta_1 + a_2 \sin(\theta_1 + \theta_2)\end{aligned}$$

where a_1 and a_2 are the lengths of the respective links. For robots with more than 2 DOFs, i.e. n-link analysis, the mathematical representation becomes more complex, and the concept called the Denavit-Hartenberg convention is used to represent the configuration kinematics. This gives a systematic approach to obtaining a homogeneous transformation matrix consisting of the product of four basic transformations. An example is

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \quad (3.1)$$

where the matrices *Rot* and *Trans* represents rotation and translation about their respective axes related to link i and joint i, and the parameters a_i , α_i , d_i and θ_i are the link length, link twist, link offset and joint angle, respectively. Three of the four parameters will be constant for any given link, while the fourth is denoted the joint variable (θ_i for revolute joints, and d_i for prismatic joints). The forward kinematic equations can then be represented on a general basis as the transformation matrix

$$H = T_n^0 = A_1 A_2 \cdots A_n$$

3.2.2 Inverse kinematics

The problem of inverse kinematics can be said to be the opposite of forward kinematics, i.e. the goal is to calculate the required joint configuration from a desired position and orientation for the end effector. The general statement of this problem is as follows: Given a 4x4 homogeneous transformation

$$H = \begin{bmatrix} R & o \\ 0 & 1 \end{bmatrix} \in SE(3)$$

find a solution, or solutions, such that

$$T_n^0(q_1, \dots, q_n) = H$$

where

$$T_n^0(q_1, \dots, q_n) = A_1(q_1) \cdots A_n(q_n)$$

as described in Eq. 3.2.1, where q are the joint variables. Here H is the desired position and orientation of the end effector, so the task is to find the values for the joint variables that satisfies Eq. 3.2.2.

3.3 Using a virtual machine

During the work with the preceding project [8], it was concluded that it would be beneficial to gather all solutions on a single computer. The main arguments are physical space utilization on the mobile platform, total weight and software accessibility. Physical space and total weight is particularly important considerations, as the wheel drive system has been proved to be somewhat fragile [23]. The main issue with realizing this is, as mentioned earlier, the USB Controller for the SCORBOT-ER 4u, which requires a Windows platform to communicate with the computer.

One possible solution is to circumvent the USB Controller completely. This would mean to reconfigure the way that commands and readings are sent to

and from the Scrobot, and would probably be a very work-intensive endeavour. As this is out of the scope of this thesis, alternative solutions had to be found. Using the program Wine, which emulates a Windows environment in Ubuntu, was considered as a possible solution. However, as Wine did not already provide support for the Scrobot setup, new implementations would have to be made. Thus this solution was also discarded, as it would probably take away too much time from the actual goals of the thesis.

The solution that was eventually chosen was to use a VM. The hypervisor that was chosen is Virtualbox, primarily because of its freeware status, and the ability to connect to USB devices. While the ability to use USB devices is an extra feature that require the purchase of an extension to the program, it seems to be free as long as it is under a student/academic license. Thus there would be no infringement while using it in this project.

3.3.1 VM networking

Communication to and from the VM is crucial for the viability of this solution. Not only does it have to receive commands over ROS topics, but it should also be able to send confirmation messages back. Below follows a brief list of some of the network types available through the Virtualbox interface, and their primary uses. They are found in the VirtualBox manual, chapter 6 [12].

NAT network: Network Address Translation is the default networking mode in VirtualBox, and also the easiest. It works by connecting the virtual machine to a "router", i.e. the VirtualBox networking engine, which maps traffic from and to the virtual machine transparently. Each "router" is placed between the virtual machine and the host. The downside of this is that just like a private net-

work behind a router, the virtual machine is unreachable from the outside world.

Bridged networking: With bridged networking, a device driver on the host system is used, that filters data from the physical network adapter. The driver is thus called a "net filter" driver. This allows VirtualBox to intercept data from the physical network and inject data into it, ensuring two-way communication. It is effectively creating a new network interface in software, which when used by the VM acts to the host as if the virtual machine is physically connected to the interface using a network cable.

Host-only networking: Host-only networking is a hybrid between bridged networking and internal networking (not featured here). In essence, it allows the VM to talk to the host computer as if it was connected through a physical Ethernet switch, without the need for a physical networking interface. This does however not support communication outside of the host.

There are several different networking modes available in VirtualBox, but these three are the ones that are most likely to be used. The main networking modes used in this project are NAT for access to the "outside" world, and bridged networking for communication with the host computer. By design, the virtual machine does not communicate directly with the OCS, as the information flow should pass through the ROS core. As all communication is designed to go through the ROS core (which is located on the host computer), host-only networking could also have been used.

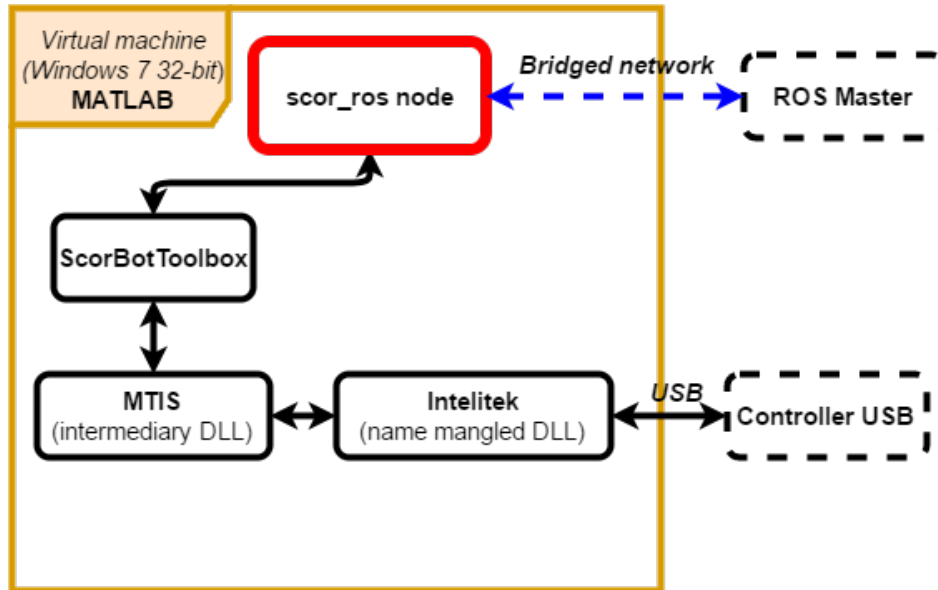


Figure 3.1: MATLAB interface concept

3.4 Control through ROS

3.4.1 Communicating with the USB Controller

As mentioned in the preceding project thesis [8], the control of the Scorbot-er 4u is as of now limited to using a Windows 32-bit platform, which means that the ROS implementation cannot be used directly. Following on the solutions found during the earlier work, an interface has been developed in MATLAB. This is done using the fact that MATLAB supports communication with ROS systems through the Robotics Controller toolbox [24]. The communication with the USB Controller [18] is done using the ScorBot Toolbox written by Kutzer [25], which is a continuation of the MATLAB Toolbox for the Intelitek Scorbot (MTIS) toolbox created by Esposito et al. [13], discussed in the aforementioned project thesis [8]. The interface concept is shown in Figure 3.1, and is inspired by the MoveIt! Scorbot-ER 4u implementation [28], which was found during the work with the project thesis.

The actual link consists of a handful of subscribers and publishers, linking the ROS topics to the appropriate ScorBot functions using callback functions. A

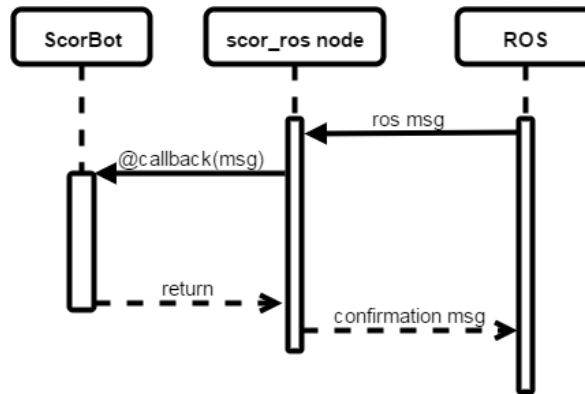


Figure 3.2: MATLAB communication concept

sample of the code is shown in Figure 3.3. The implementation is event-based through the callback functions, which means that action will only be taken as new information is received. With careful publishing on the topics subscribed to by the scor_ros node, this should provide a sufficient interface for communicating with the ROS system seemingly natively.

A focus when designing the interface was that the interface itself shouldn't contain functionality not expected to be provided by a hardware interface. This means that there shouldn't be any complicated computations or decisions being made by the interface, "out of reach" of the user. The only added functionality is a hard-coded joint limit check to ensure the avoidance of self-collision, implemented directly as a check before every movement. This may however be expected to be handled by such a device, so they do not conflict with the design philosophy.

3.4.2 Sending commands from ROS

The commands for the robot manipulator are sent from the OCS, the method for sending these commands will be presented in Chapter 5. On the ROS side they are received by a TCP socket, based on the implementation done by Lindrup [23]. This then translates the information sent from the OCS, and publishes the commands as the appropriate ROS topics.

```

%% SCORBOT init
pause(10)
%Attempt to initialize, prepare to send results
ScorInitRmsg.Data = ScorInit;
pause(1)
%Attempt to home, prepare to send results
ScorIsHomeRmsg.Data = ScorHome;
pause(1)

send(ScorInitR,ScorInitRmsg) % Send "Init is done"
pause(1)
send(ScorIsHomeR,ScorIsHomeRmsg)% Send "Scorbot is homed"
pause(1)
if ScorInitRmsg.Data == 1 && ScorIsHomeRmsg.Data == 1
    ScorBSEPRRmsg.Data = ScorGetBSEPR; %obtain joint configuration
    send(ScorBSEPRR,ScorBSEPRRmsg)
else
    warning('SCORBOT did not initialize and/or home correctly')
end
pause(1)


---


%% ROS subscribers
%Publishers for testing...
ScorGetInitPub = rospublisher('/ScorInitCmd', 'std_msgs/Bool');
InitMsg = rosmesssage(ScorGetInitPub);
%Get init message
ScorGetInit = rossubscriber('/ScorInitCmd',@ScorInitCall);

%Get message of control on/off
ScorSetCtrlPub = rospublisher('/ScorGetCtrl','std_msgs/String');
CtrlMsg = rosmesssage(ScorSetCtrlPub);
ScorGetCtrl = rossubscriber('/ScorGetCtrl',@ScorSetCtrlCall);

```

Figure 3.3: MATLAB node sample

Stereo vision system

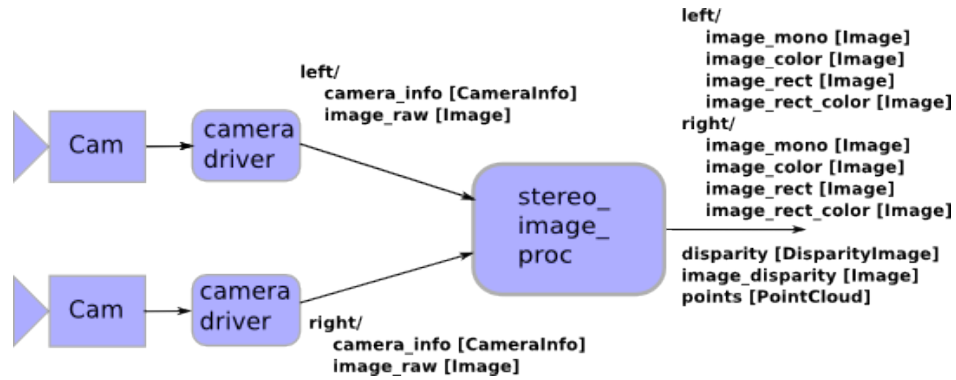
4.1 Introduction

In this chapter, the work on exploring a possible solution for the stereo vision system will be outlined. It will contain a summary of some of the methods considered, a brief description of the method that was chosen, some background theory relating to the chosen method and a presentation of the resulting attempt at an implementation. Lastly, the results of the implementations will be presented.

4.2 Exploration of possible solutions

There are numerous systems for stereo vision processing available. Below is a presentation of some of the found possible solutions for processing stereo images, and a short discussion as to whether or not they would be usable with the current stereo camera setup available in this project.

Through ROS there is for example the *image_pipeline* stack, which includes components for stereo image processing, depth processing and visualization [42]. It is designed to process raw camera images into inputs to vision algorithms in the categories mentioned [42]. However, it does require that the images are gathered through a conforming ROS camera driver node, of which there are individual stacks for a selection of cameras, including the Microsoft Kinect [37]. Specifically, in terms of stereo image processing, *image_pipeline* include the node *stereo_image_proc*, which undistorts and colorizes raw images, as well as performing rectification and computation of disparity maps [45]. The node setup is shown in Figure 4.1.

Figure 4.1: `stereo_image_proc`, image from [45]

It is based on OpenCV vision algorithms, but as mentioned earlier, it requires cameras that are supported by ROS camera drivers. In addition, it depends on calibrated cameras, and for good results the cameras should be synchronized [36]. This requires hardware support, which is not provided by the TP-Link IP cameras used in this project. In order to utilize the supported image processing tools provided by ROS, a set of camera drivers would have to be implemented for the IP cameras, which is deemed out of the scope of this thesis.

Another possibility would be to use RTAB-Map for processing the stereo images [19]. This process would be similar to the solution presented in the work of Lindrup for mapping with respect to the mobile platform [23]. It does include a ROS wrapper, and would therefore fit in well with the rest of the system, as demonstrated by Lindrup.

RTAB-Map uses the *image_pipeline* stack for processing the raw camera images as well, and the issues with the previously discussed solution would thus be inherited if using RTAB-Map.

MATLAB also provides algorithms and functions for processing stereo images, through the Computer Vision System Toolbox [50]. It contains apps for camera calibration, and functions for undistorting and rectifying stereo images. These functions, as it was with the aforementioned ROS stack, ref-

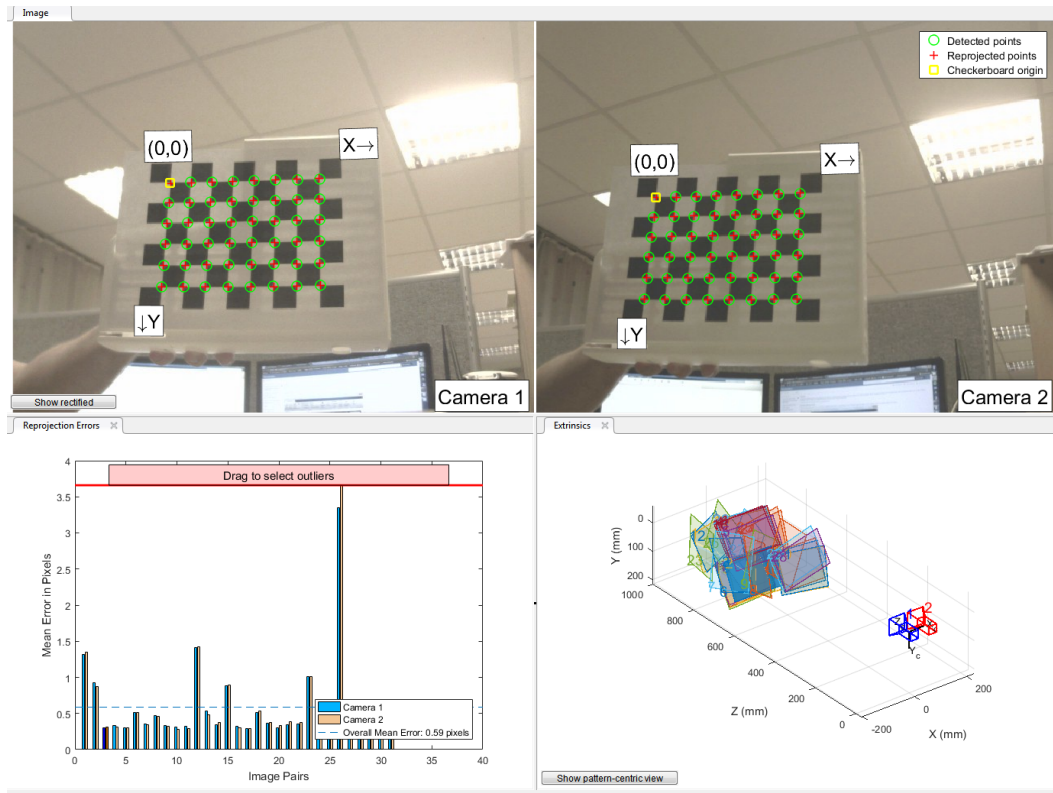


Figure 4.2: MATLAB Stereo Calibrator

erence the OpenCV library.

Stereo camera calibration is done using the Stereo Calibration App, which takes two sets of stereo images and produces rectified images as well as the necessary camera matrices. The process was tested, and one of the results from the calibration app is shown in Figure 4.2. As can be seen, the calibration app produced decent results.

Although MATLAB supports the ROS framework, a fact that is already utilized in the implementation of the robot manipulator control, it is argued that the addition of this method would only serve to increase the complexity of the system, both in terms of navigating the solutions and running the system. This is not desirable, as the system structure is already large.

The decision fell on exploring the possibility of using the OpenCV library directly, trying to use the functionality of the library without going through

third-party solutions, to somehow circumvent the issue with camera synchronization and lack of camera drivers.

4.3 OpenCV

OpenCV is an open source computer vision library, built to provide a common infrastructure for computer vision applications [30]. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS [30]. According to the official website [30], the library contains more than 2500 optimized algorithms, including stereo vision, facial recognition and tracking. It is released under the BSD license, and is thus free for both academic and commercial use. The library has a modular structure that includes several shared and static libraries [31]. Below are some of the modules that are used in this thesis, with descriptions according to the application programming interface (API) reference [31]:

- core:** Basic data structures
- imgproc:** Image processing, including filtering, transformations etc.
- calib3d:** Multiple-view geometry, camera calibration etc.
- highgui:** Interface to video capturing.

4.4 Background theory

In order to utilize collision avoidance, an overview of distances to the objects are needed. As the equipment available consists of two cameras, mounted to emulate the eye-orientation of a human, the concept of stereo imaging will be used. Below follows an introduction to the theory behind stereo imaging, and how to use that for creating depth maps. The following are the necessary

steps when using stereo imaging with two cameras, according to Learning OpenCV [9];

1. Mathematically remove radial and tangential lens distortion.
2. Adjust for angles and distances between cameras, i.e. rectification.
3. Locate the same features in both cameras, and produce a disparity map.
4. Knowing the cameras geometric arrangement, calculate distances using triangulation.

This section will try to explain a few of the basic concepts of stereo vision, for an in-depth look at the theory, the reader is referred to Chapter 9 of the book by Hartley and Zisserman [17], and Chapter 12 of the book by Bradski and Kaehler [9].

4.4.1 Epipolar geometry

The epipolar geometry between two views is the geometry of the intersection of the respective image planes, where the pencil of planes, i.e. the set all of planes through a line, uses the baseline as axis [17]. Figure 4.3 displays some key concepts of epipolar geometry using two pinhole models. The points O_l and O_r correspond to the center of projection for the left and right camera, respectively. The epipoles, e_l and e_r , correspond to the intersection of the baseline in the corresponding image plane, or "the image of the center of projection of the other camera" [9].

The point P is the actual viewed point, and the points p_l and p_r are the projections of that point on the respective image planes. The plane formed by the viewed point P and the epipoles is called the epipolar plane, marked in grey in the figure. The lines in the image planes through the respective epipole and projection point are called epipolar lines. The facts stated here can be summarized as follows (taken in part from [9]):

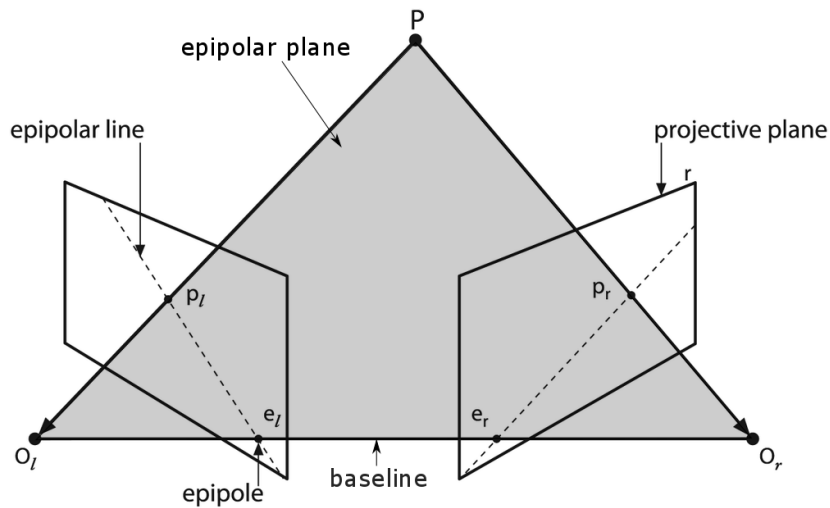


Figure 4.3: Epipolar geometry (Edited, original image from [9, p. 420])

- Every 3D point in view of both cameras is contained within the epipolar plane.
- The epipolar constraint states that given a feature in one image, the corresponding feature in the second image must lie on the corresponding epipolar line.
- The order of features is preserved, i.e. of two points visible in both images, the leftmost point horizontally in one will also be the leftmost point in the other.

If the cameras are row-aligned, i.e. the image planes are coplanar and the image rows are exactly aligned, the epipole lines would go on to infinity [9]. Therefore, instead of searching through the corresponding epipolar line, a search is conducted in the corresponding row. The algebraic representation of epipolar geometry is called the fundamental matrix [17]. The fundamental matrix contains information about the translation and rotation that relate the cameras in physical space, as well as information about the intrinsics, the internal properties like focal lengths and principal points, of both cameras [9].

4.4.2 The essential and fundamental matrices

The fundamental and essential matrices are used to represent the epipolar geometry algebraically [17]. The following mathematical definitions are from [9].

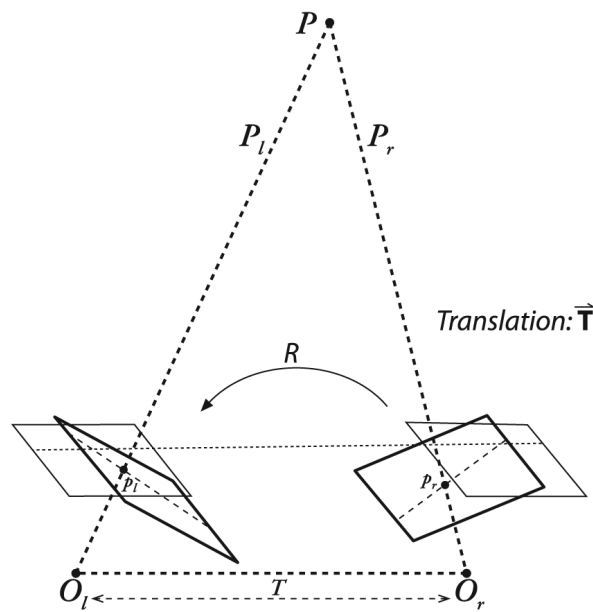


Figure 4.4: Essential matrix¹

Essential matrix

The essential matrix is purely geometrical, and relates the physical location of the point P as seen by the left camera to the location of the same point as seen by the right camera [9]. The goal of this is shown in Figure 4.4. In other words, it relates the observed locations of p_l and p_r , as mentioned earlier. A set of coordinates is defined with respect to either the left or right camera. Centering the coordinate frame at O_l would mean that the location of the point is denoted P_l . T is a vector from O_l to O_r , which means that the location of O_r is T with respect to the coordinate frame centered at O_l . The point P as seen by the right camera is then defined as $P_r = R(P_l - T)$, where R is the rotation matrix. The key is the introduction of the epipolar plane,

¹Image from [9, p. 419]

mentioned earlier, which contains all the points and vectors used. q_l and q_r are the points on the image planes in the image coordinates (pixels). The equation for all points \mathbf{x} on a plane with normal vector \mathbf{n} , passing through a point \mathbf{a} , follows the constraint $(\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} = 0$. Considering that the epipolar plane contains both P_l and T , this can be written as

$$(P_l - T)^T(T \times P_l) = 0$$

Rewriting the relationship between P_l and P_r , gives $(P_l - T) = R^{-1}P_r = R^T P_r$, which in turn gives

$$(R^T P_r)^T(T \times P_l) = 0$$

Rewriting the cross product as a matrix multiplication gives $T \times P_l = SP_l$, where S is a skew-symmetric matrix

$$S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix}$$

This gives $P_r^T R S P_l = 0$, where $RS = E$, the essential matrix.

Fundamental matrix

The pixel coordinate p is related to the point in the image plane q by $q = Mp$ or $p = M^{-1}q$, where M is the camera intrinsics matrix. This means the equation for E can be written as

$$q_r^T (M_r^{-1})^T E M_l^{-1} q_l = 0$$

where the intrinsic matrix is given by

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Here, f_x and f_y are the focal lengths, given as a product of the physical focal length (denoted f earlier) and the size of the individual camera elements, s_x and s_y . I.e., $f_x = f s_x$ and $f_y = f s_y$. c_x and c_y are parameters introduced to model possible displacement of the center of coordinates on the projection screen away from the optical axis. Thus, the fundamental matrix F is defined as $F = (M_r^{-1})^T E M_l^{-1}$. Note that if the images are rectified, and the points are normalized by the focal lengths, the intrinsic matrix M becomes the identity matrix, and $F=E$.

4.4.3 Undistortion

For real, non-pinhole lenses, the image model is usually not linear, and contains a deviation, where the most important one is caused by radial distortion [17]. In general, radial distortion is a type of deviation that depends on the distance between the imaged point and the optical axis (the focal length), and that changes the distance between the image center and the image point [15]. One instance of the effect is shown in Figure 4.5 (a).

4.4.4 Calibration

Camera calibration is the action of defining the intrinsic and extrinsic parameters of the cameras relative to a fixed world coordinate system [15]. In short, stereo calibration means to find the rotation matrix R and the translation vector T between the two cameras, as discussed earlier.

Single camera calibration

Single camera calibration is a procedure to find a model of the camera's geometry, as well as a distortion model of the lens [9]. These two combined define the intrinsic parameters of a camera. In a sense, camera calibration

mathematically corrects the main deviations from the pinhole model that the camera lenses imposes [9], as mentioned earlier.

Stereo calibration

By using single camera calibration for the two cameras separately, the point P can be put in the camera coordinates by $P_l = R_l P + T_l$ and $P_r = R_r P + T_r$ for the left and right camera, respectively. Here, P_l and P_r denote the locations of the 3D point P in the respective coordinate systems, while R_l , R_r , T_l and T_r denote the rotation and translation from the camera to the 3D point. The two views of P are then related by $P_l = R^T (P_r - T)$, where R and T are the desired rotation matrix and translation vector between the cameras, here into the left coordinate frame [9]. This gives the simple relations

$$R = R_r R_l^T$$

$$T = T_r - R T_l$$

4.4.5 Rectification

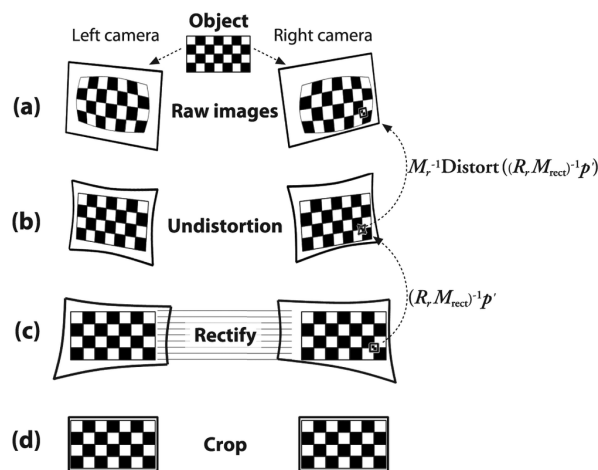


Figure 4.5: Stereo rectification, image from [9, p. 438]

Image rectification is in essence to replace the images with two projectively equivalent images on a common image plane, parallel to the baseline [15]. In these projections the epipolar lines run parallel with the x-axis, and match

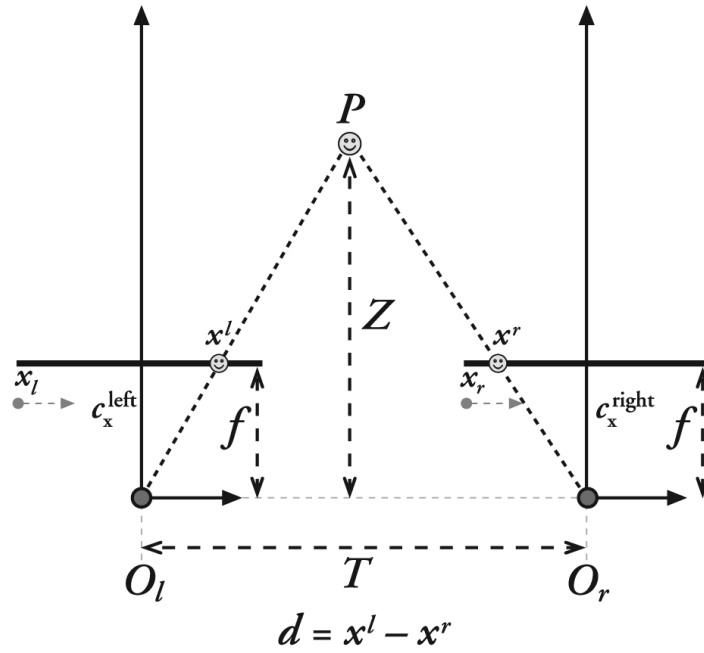


Figure 4.6: Triangulation, image from [9, p. 416]

up in both views [17]. This means that there is no disparity, i.e. difference in point location, in the y-direction, and so the search for corresponding pixels is much easier. The outcome of rectifying the images is shown in Figure 4.5. Having the epipolar lines run parallel with the x-axis is another way of saying that the epipoles should be mapped to an infinite point $(0, 0, 0)^T$ [17]. The result will be four terms for each camera, namely a distortion vector, a rotation matrix T_{rect} , a rectified camera matrix M_{rect} and the unrectified camera matrix M . These terms are used to make a map to interpolate pixels from the original image to the new rectified image [9]. For a more in-depth look at various algorithms and the underlying math, the reader is referred to Chapter 12 of Bradski [9] and Chapter 11 of Hartley and Zimmerman [17].

4.4.6 Disparity map and triangulation

The final step is triangulation, which produces a depth map using the geometric arrangement of the cameras and the produced disparity map [9]. The disparity is the difference in the image location of the same 3D point when

projected under the perspective of the different cameras [46]. For a rectified image, this means the difference in the location on the corresponding scan-line, or pixel row. Figure 4.6 shows this notion for a perfectly undistorted and aligned stereo setup. Here we can then see that the relationship between the depth (or distance) to the object, and the disparity, is given by:

$$\frac{T - (x^l - x^r)}{Z - f} = \frac{T}{Z} \implies Z = \frac{fT}{x^l - x^r} = \frac{fT}{d}$$

In other words, the depth Z is inversely proportional to the disparity d .

4.5 Camera setup and calibration

4.5.1 Camera setup

During initial testing some there was experienced some lag between the cameras, a currently unexplained phenomenon. To compensate for this there were some tests done with different camera parameters and type of video being collected. A compensation between quality and lag was found with streaming MJPEG video at a resolution of 640x480, with *Good* video quality and 15 frames per second.

4.5.2 Calibrating the cameras

Before implementing stereo vision, the cameras had to be calibrated. The calibration is done with a slightly modified version of the function `cv::stereoCalib`, a sample function from the openCV library². The camera intrinsic matrices were found with a resolution of 640x480, and will need to be scaled for different resolutions. The extrinsic matrices should not need to be scaled. These matrices are the key to performing operations on the cameras, and it

²http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

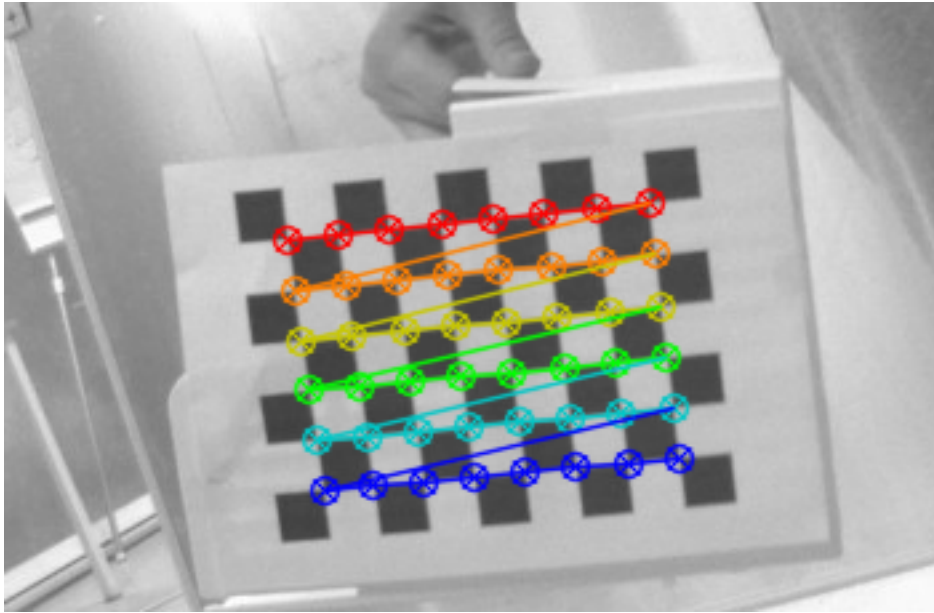


Figure 4.7: Finding corners of chessboard

is therefore imperative that they are as accurate as possible. The found camera matrices, M , are presented below. The full set of intrinsic and extrinsic matrices are found in Appendix C.

$$M_{left} = \begin{bmatrix} 795.8546 & 0 & 319.9971 \\ 0 & 796.8261 & 239.3465 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{right} = \begin{bmatrix} 795.8546 & 0 & 321.2961 \\ 0 & 796.8261 & 241.9785 \\ 0 & 0 & 1 \end{bmatrix}$$

The RMS reprojection value is currently around 6, which means that the reprojection "misses" the original placement by 6 pixels. This is not nearly good enough, as a successful calibration should yield sub-pixel errors. The working theory is that this is caused by the poor image quality and slight lag between the cameras.

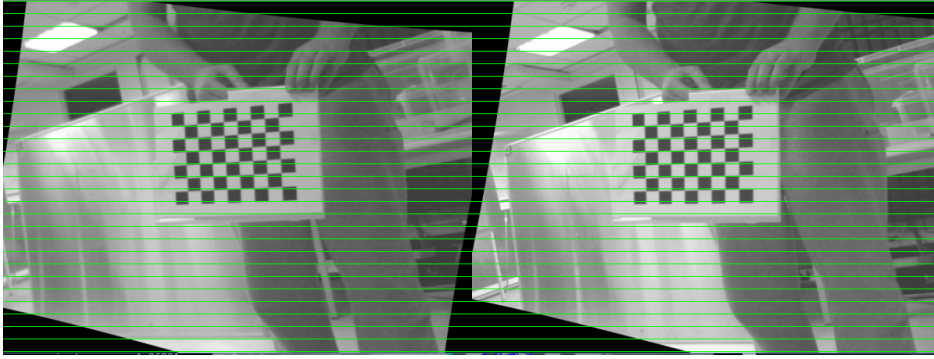


Figure 4.8: Rectified images

4.6 Depth map and collision avoidance

4.6.1 Producing a depth map

Based on the example function *stereo_match* from within the *opencv* samples, a function for producing a visualized depth map is made. The program is made as a ROS node called *mar_stereo_vision*. It takes the images from the IP cameras in real time, process the gathered frames and publish the result using the *image_transport* package. Using *opencv_bridge*, one can easily convert between OpenCV images and ROS Image message, as illustrated in Figure 4.9.

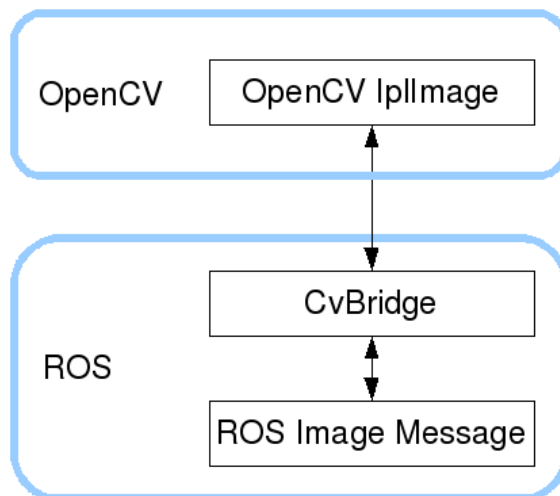


Figure 4.9: CvBridge, image from [40]

The full procedure is as follows:

1. Initialize cameras as VideoCapture object
2. Check that both cameras are connected
3. Stream the camera frames to an image matrix
4. Rectify and undistort the captured frames
5. Produce the disparity/depth map
6. Calculate the depth information from the disparity map
7. Publish the depth map and the depth values

The rectification is done using the openCV function *stereoRectify*. This function takes the intrinsic and extrinsic matrices found during the calibration, and gives the rectification transform and projection matrices for the respective cameras. It also calculates the disparity-to-depth mapping matrix. Undistortion is initialized by using the already available function *initUndistortRectifyMap*. This takes the camera matrices and the distortion coefficients for the respective cameras, and the rectification transformation computed in the previous step, and computes the undistortion and rectification transformation as maps, used to remap the camera images so that they share the same scan lines.

The disparity map is then produced using a block matching algorithm for rectified image pairs. This is already implemented in openCV through the *StereoBM::operator()*. It is important to note that one must use the rectified image pairs for the computation to be successful. This sequence of operation correspond to the sequence declared earlier. The resulting image should then be published as a ROS image, in order to access it through the OCS.

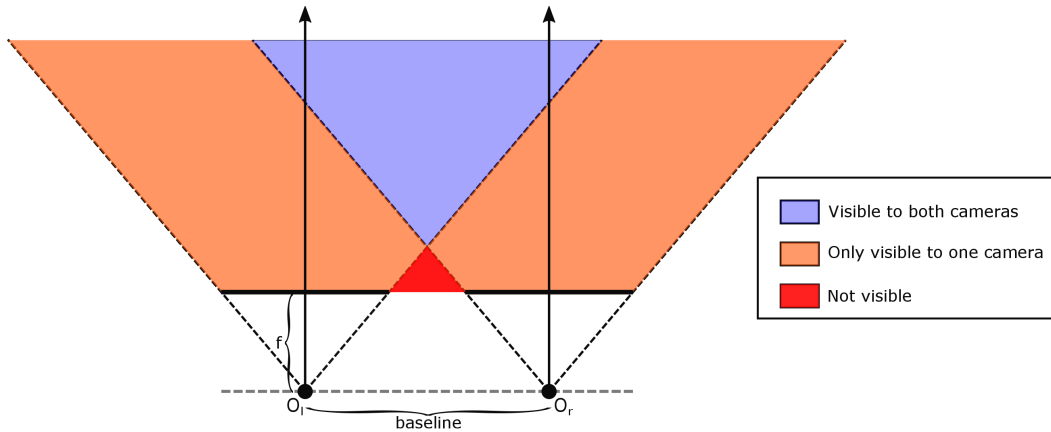


Figure 4.10: Camera view illustration

4.6.2 The pan-tilt unit

The usable view-field of the cameras, as illustrated in figure 4.10, is significantly narrower than the actual view-field. O_r and O_l represent the camera centers, f is the focal length. As the IP Cameras used in this project have a narrow horizontal view, the ability to pan the cameras in the direction of movement is crucial for realizing collision avoidance. An illustration of the stationary horizontal view is shown in Figure 4.10. The pan-tilt unit is controlled through a ROS package called *flir_ptu_driver*, as described on the ROS Wiki [41]. The full command reference can be found there.

. The idea is that the PTU commands ghosts the movement commands issued to the robot manipulator, so the cameras are always "ahead" of the manipulator. The thought behind this is that this will improve the reaction time of the collision avoidance system. In addition to this, it will also be possible to control the PTU directly, by using the point of view (POV) hat on the joystick.

4.6.3 Collision avoidance

With successful results from the depth map production, the physical distance to the objects can be done through the OpenCV function *reprojectTo3D*.

CHAPTER 4. STEREO VISION SYSTEM

This function calculates the x , y and z position for each pixels. These values would then be sent to the command publisher for the robot manipulator, to override potential movement in case of impending collision.

Operator Control Station

5.1 Introduction

The OCS is the key to having a remote controlled system. Designing the graphical user interface (GUI) was driven by usability and functionality. Its purpose is to link the user to the MAR, giving the user control and surveillance options. The idea is to split the OCS into two modes, one for driving and mapping, which connects to the implementations done by Lindrup [23], and one for controlling the robot manipulator. This is done by utilizing the *stackedWindows* type, which gives the ability to switch between windows when triggered. The reasoning behind this is that the layout would be messy if all the information were to be displayed at the same time.

Also, as a safety measure, it would be desirable to fully prevent accidental movement of the robot manipulator while driving. This is because the robot is not aware of the arm orientation while driving, and so would not account for the arm in terms of collision avoidance.

The following sections will display the concept for the different windows, and also describe the basic functionality.

5.2 Qt

Qt is a framework for creating cross-platform user interfaces [32]. Supported platforms include Linux, OS X, Windows, Android, iOS and others [32]. Qt also provides its own IDE, called Qt Creator. The framework is not a standalone programming language, but is written in C++. It uses a preprocessor to extend the C++ language, and parse the source files to generate standard

C++ sources. This means that the framework itself and applications can be compiled by standard C++ compilers, e.g. MinGW and MSVC [32]. Qt contains a number of modules, and below are some of the modules used in this thesis, with descriptions from the Qt documentation [51]:

- Qt Core:** Base library that provides containers, thread and event management, etc.
- Qt GUI:** Base classes for GUI components. Includes OpenGL.
- Qt Widgets:** Extends Qt GUI with C++ widgets.
- Qt Network:** Provides classes to deal with network communication.

5.3 Main menu

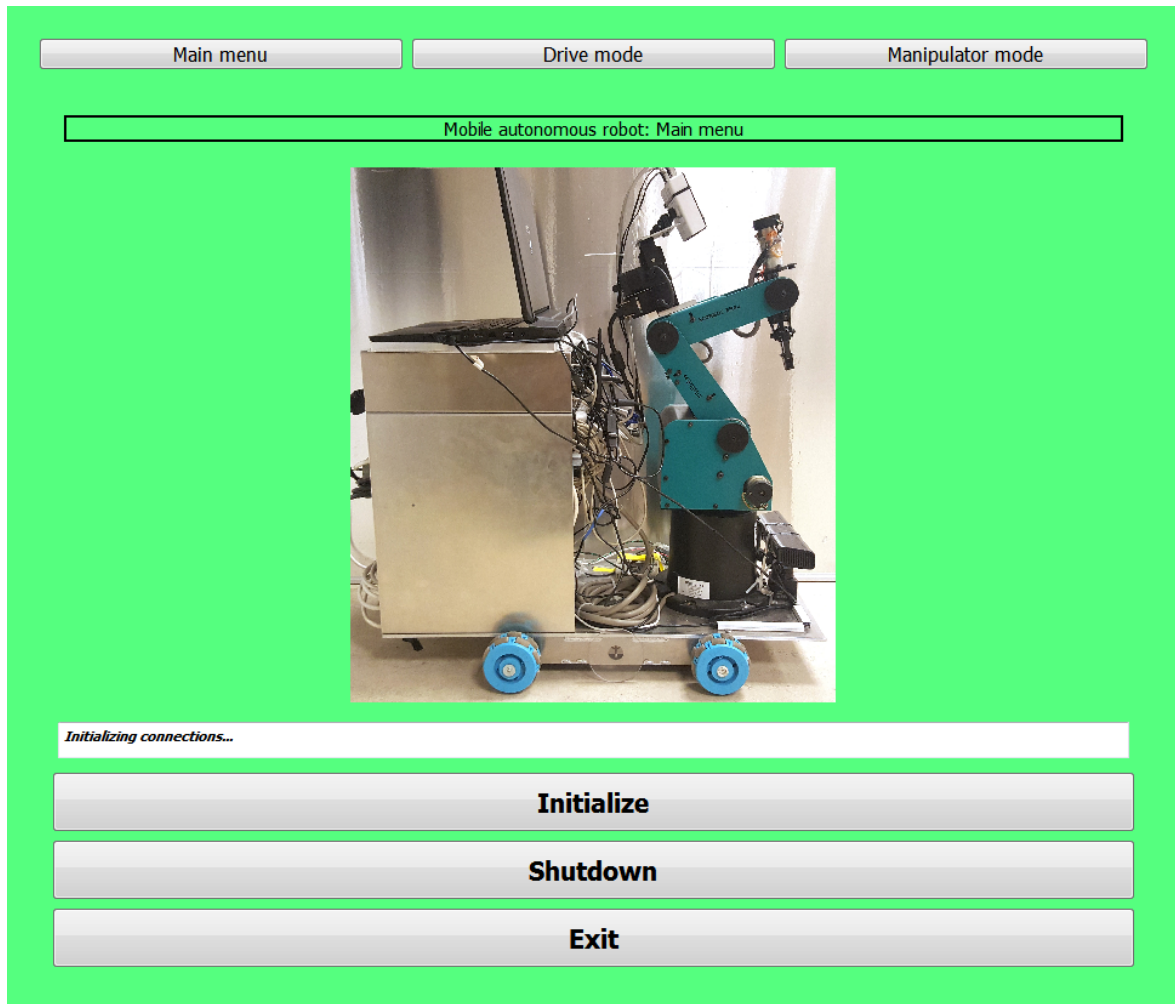


Figure 5.1: OCS Main menu

The idea is to have some sort of main menu, with initialization of TCP/UDP connections, video streams etc., and also an added possibility to shut down connections without exiting the program. Upon exit, all modules should be shut down automatically. There is also a status bar, which displays feedback text, i.e. operator status, errors, etc. This is shown in Figure 5.1

5.4 Manipulator mode

The joystick readings will be sent through the OCS, as a means of making sure no "accidental movement" of the robot manipulator will occur. The thought is that movement of the robot manipulator during the mobile phase is unwanted and poses a security risk (the possibility of crashing the manipulator will be higher when the robot is in movement). Therefore the joystick activation should only be possible when the robot is stationary, or at low speeds. The readings are gathered through a program based on the Raw Input API, and sent over a TCP/IP server set up to communicate with the ROS system. On the right hand side the unprocessed camera feed is placed, and the produced depth map would have been placed on the left hand side. This was omitted due to the poor results of the stereo vision system.

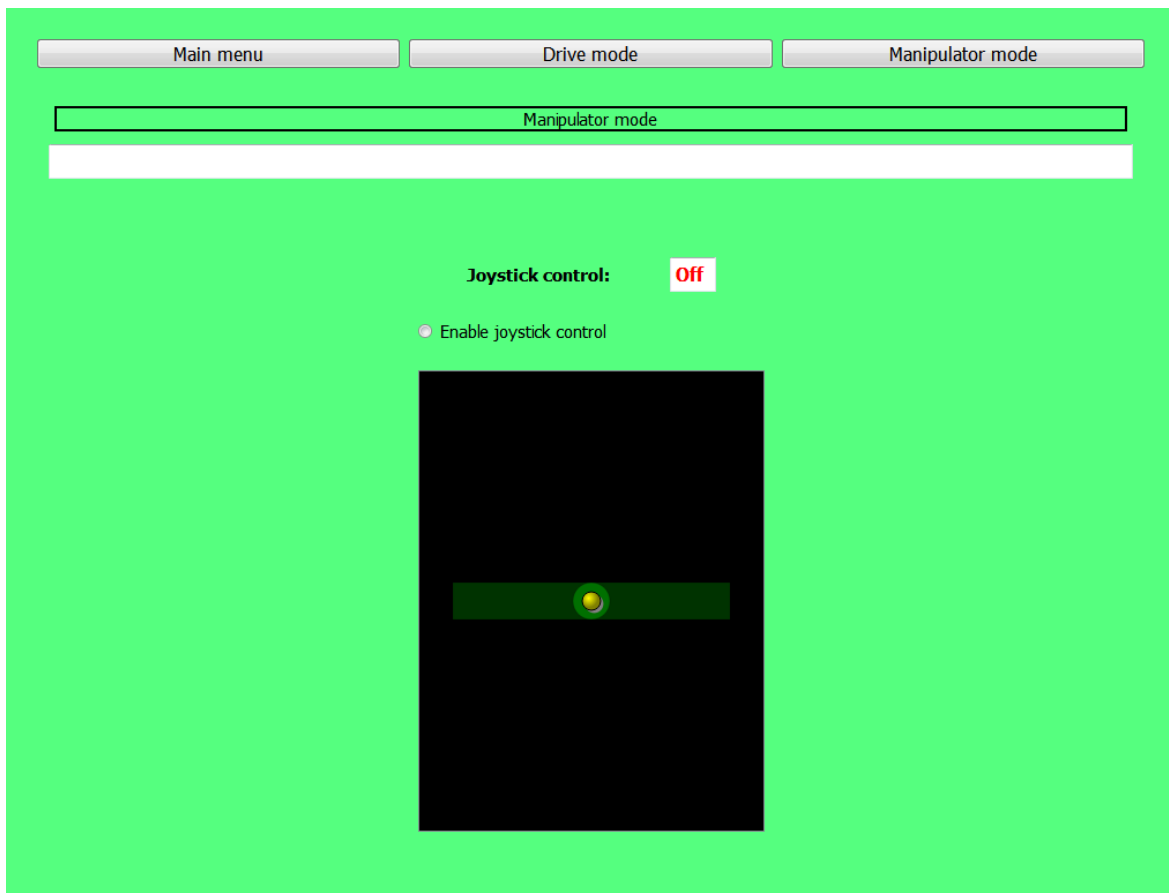


Figure 5.2: OCS Manipulator mode

5.4.1 Joystick

Having the ability to use a joystick to control the robot manipulator is the key to the concept of remote control. It is therefore crucial for the OCS to both read and transmit joystick data. The joystick state has to be read in a suitable format for transmitting to the ROS system. There were several possible solutions explored, among which was the Direct Input API from Microsoft, and *libusb*, a C library that provides generic access to USB devices. As the OCS solution is currently made for Windows, the decision fell on the raw input API from Microsoft which accepts raw input from any human interface devices (HID) [27].

Raw input is centered around the *WM_INPUT* message, which is sent to the window that is getting the raw input, and contains the input code and a handle to the input structure that contains the raw input from the device. To use these messages in Qt, a class based on the Qt class *QAbstractNativeEventFilter*, which provides an interface for receiving native events, was created. The event filter is then initialized for the OCS window, and sends data only when that window is active.

5.5 Drive Mode

The Drive Mode window is intended to include the implementation by Lindrup [23], and the functionality created during his project. As this is a conceptual implementation, full functionality and inclusion of his methods are not present, as this was deemed too labor intensive and out of the scope of this thesis. The unprocessed camera feed is shown on the right hand side, and the produced map would be placed on the left hand side.

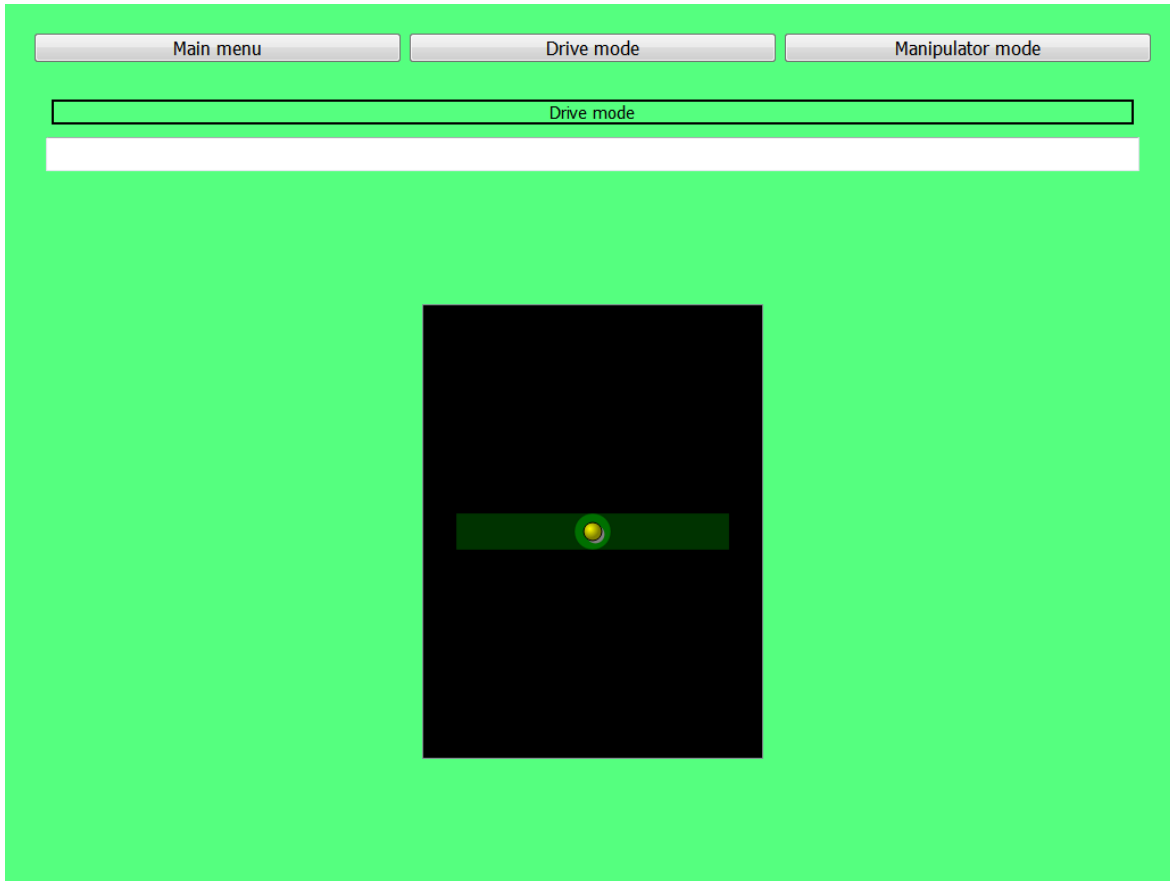


Figure 5.3: OCS Drive mode

MAR User Manual

Producing a comprehensive user manual for the total robotic system called the Mobile autonomous robot was considered to be a good contribution to the future development and operation of the student built robot. The index of the manual is found in Appendix A, and is meant to act as a standalone document. The level of detail within the manual was chosen in a way that the document itself would hopefully be enough to understand and use the system, and the readers should also be able to do a full re-installation of the various parts if necessary. The work on preparing the manual was influenced by an article on user manuals by Dr. Philip Hodgson [34], which presents key tips for designing a user manual. Some of the tips that have been focused on include [34]:

- Make sure the instructions actually map onto the system.
- Present instructions as step-by-step procedures.
- Tell the reader what the functions are, and what they're for (used here for the hardware).
- Avoid text-book look.
- Have hierarchically structured information.
- Include troubleshooting sections.

All the insights and information presented in the user manual is from the authors own work with the system, and also gathered from earlier work concerning specific modules. The thesis of Aspunvik [5] has been consulted for some of the hardware setups and router configuration, and the thesis of

Lindrup [23] has been consulted for the procedures concerning his implementation. The user manual is structured as follows:

- **Introduction:** Short declaration of the purpose of the manual, and an abstract view of the system
- **Hardware:** A comprehensive list of the hardware used on the mobile robot, as well as short descriptions of each component.
- **Cables and connections:** A list of the necessary cables and adapters, as well as connection schematics for power, usb, etc.
- **Software:** Lists of necessary software, libraries and so on needed to run the system, both for the on-board computer, and the OCS.
- **Installation and configuration:** Instructions for installing and configuring the various parts of the system.
- **Running the system:** A guide to running the system, as it is at the time of writing.
- **Troubleshooting:** Declaration of known issues, and their potential solutions.

Results

7.1 MATLAB interface

The MATLAB interface performed as expected, successfully completing the commands sent on the ROS topics. The implemented self collision check also did what it was supposed to do, preventing collision with the rear housing by a safe amount.

7.2 Stereo vision system

The stereo vision system was tested in different modes, with still images taken with the cameras, with live stream of stationary objects, and with objects in motion. The result from the still images is shown in Figures 7.1 and 7.2, and shows promising results. The output is a colorized disparity map, where red indicates close objects and blue indicates objects sufficiently far away. The program is able to pick up most of the objects features, and correctly displays the relative distances.

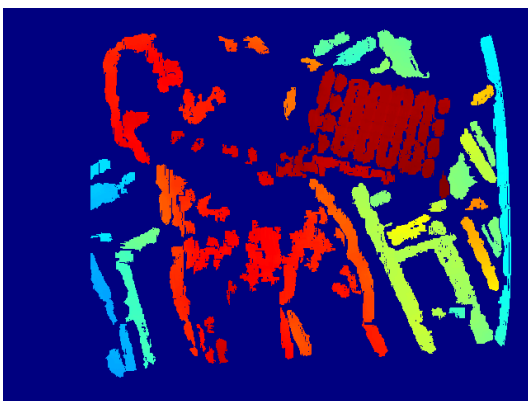


Figure 7.1: Stationary depth map



Figure 7.2: Original image

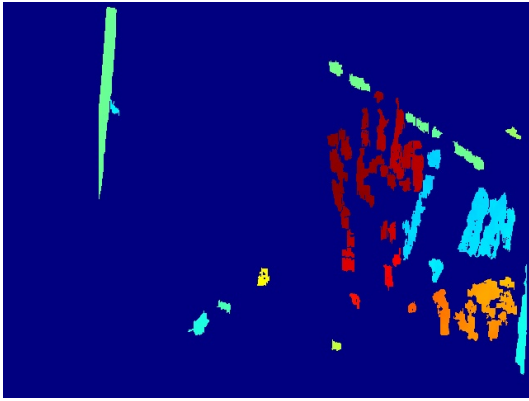


Figure 7.3: Real-time stationary object



Figure 7.4: Real-time original image

Figures 7.3 and 7.4 shows the same program on a real-time video feed with a stationary object. As can be seen the result is not as robust as with the still images, but it still manages to display most of the features of the nearest object, with a lot of unwanted noise and missing features of the surroundings, that were mostly properly displayed in the still-image test. The real-time stream introduces uncertainty in the image processing as the images from the left and right cameras do not always correspond to each other, due to a lack of proper synchronization.

The final and most telling test is shown in Figures 7.5 and 7.6. These images show the result of trying to capture an object in motion. It is evident that the program is not able to keep up with changing positions, and is only able to correctly place fragments of the object. In addition to this, some false positives are seen in the lower left corner of Figure 7.5. In a collision avoidance implementation, false positives like these could be extremely destructive to the overall functionality of the system.

All of the tests shown here are run with pretty harsh filtering of noise on the produced images, which might explain why there are a lack of features in the last two examples. This was however deemed necessary, as the produced depth maps with looser filtering was almost completely noise. There was also experienced some currently inexplicable lag between the cameras, which

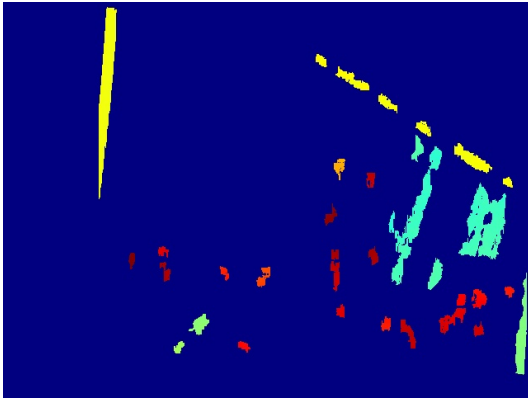


Figure 7.5: Real-time object in motion



Figure 7.6: Real-time original image

probably contributed to the poorer real-time results. This will be discussed further in Chapter 8.

7.3 OCS

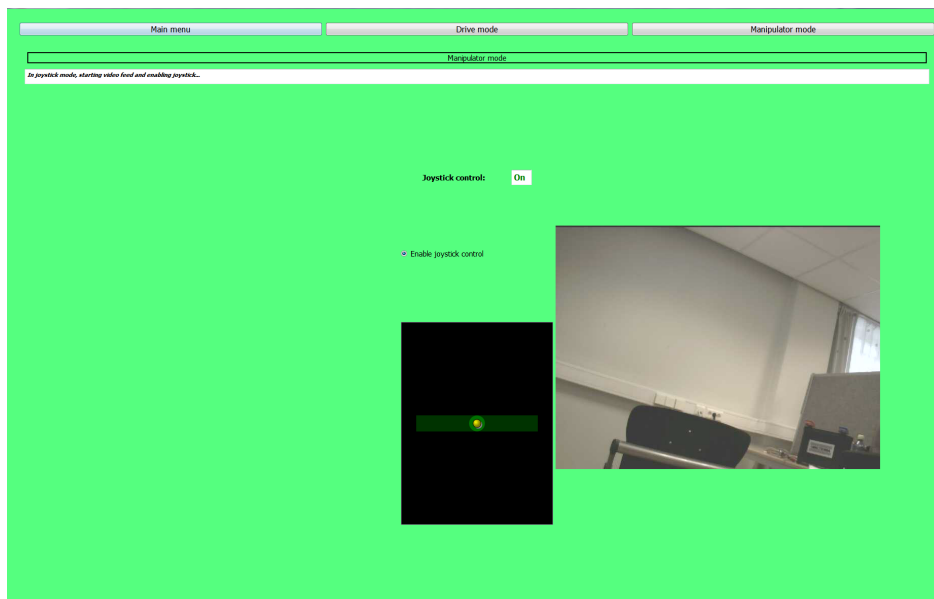


Figure 7.7: Manipulator Mode activated

The OCS performed the functions outlined earlier. Because of the use of the Raw Input API, there was not found a solution for locking the joystick input to the Manipulator Mode window, as the system messages are triggered on a window ID. For now it seems like assigning a valid ID to a sub-window is

not possible. Instead, this was achieved by passing a Boolean triggered by the checked status of the Joystick Control button.

7.4 User manual

There was some internal debate on how to test the success of the user manual. Ideally, there would be performed a full system reset, with a test person spending some time getting the system up and running again, armed with only the implementation files and the user manual. This, however, was deemed to be too time-consuming, considering the risk that the system might not regain the full functionality immediately, something which would be unacceptable as there were other tests of the system carried out at the same time. The evaluation of a user manual should include a user-test, as also stated in the article by Dr. Hodgson [34]. In order to get an objective review of the usability, the manual was distributed to a select group of people, which were asked to review the layout and readability, depth of information, and general functionality. The general feedback was positive, with some minor changes proposed to the structuring and chapter ordering. However, as stated earlier, the absolute usability is impossible to test without a full test with the actual system. Under the circumstances, the method employed for testing was deemed sufficient.

Discussion

This chapter will discuss some of the work and implementations presented in Chapters 3-6, based on some of the results presented in Chapter 7.

8.1 General assessment of the system

The system as it was developed, provides the basic functionality outlined in the problem statement. The planned modules were implemented, though not all modules are as robust as would be desirable. The most important focus of this work, the control of the robot manipulator, and the implementation of stereo vision, was successful. The implementations do however leave room for further improvement and implementations. There were some elements that did not perform satisfactory, specifically the stereo vision system, which will be discussed further.

8.1.1 Development tools

ROS has proved to be a flexible and robust framework for development of robotic solutions, and the performance has been as satisfactory as advertised. The messaging system facilitates large system development, and the vast amount of packages and drivers satisfies most needs when developing a robot system. The basic functionality is also relatively easy to use, which removes the need for extensive research before use.

OpenCV includes a vast amount of algorithms and functions for developing computer vision applications, and has proven to be relatively robust and easy to use. Some basic understanding of computer vision is needed, but not to

the extent of limiting usability.

Qt is an easy to use framework, and users with a knowledge of C++ should not have too much difficulty grasping the Qt concepts. It also supports many different compilers, which is a plus. The need for additional compilers could however be seen as a minor drawback, as it often leads to having to install additional software. However, the only supported joystick-type input device is a gamepad, which is surely the most popular device in use today, but it seems a bit odd that regular joysticks are not supported in a GUI framework.

8.2 USB Controller interface

8.2.1 Assessment

Although the method employed in the current solution involves a lot of unnecessary steps, it does the job as it is intended. MATLAB provides easy message passing with the ROS system through the Robotics Control Toolbox, and the ScorBotToolbox works well. The *scor_ros_node* is fully capable of receiving the commands needed to control the robot manipulator, and also sends various confirmations on the triggered tasks. It would be easily expanded to include the full array of functions available through the ScorBotToolbox, if that would be required.

8.2.2 Weaknesses

As discussed in the preceding thesis[8], having to interface the USB Controller through a Windows platform is highly inconvenient with respects to the current build centered around ROS. Not only does it create unnecessary levels of implementation and communication configurations, it also weakens the system's integrity by adding another possibility for failure or communication drop. Being directly connected to the ROS Master would prevent such

issues, and should be a consideration for further development.

8.3 Stereo vision system

Producing a real-time depth map and having a reliable source of distances to surrounding objects is a key part to implementing collision avoidance. That does however depend on a reliable source of stereo images to process. The current system, as it is, but does not provide satisfactory results, as shown in the previous chapter. There are several reasons why the results

Firstly, the cameras are not synchronized, which produces unwanted disturbances in the images which could be interpreted as false movement when constructing the depth map. If one camera took an image a bit later than the second camera, moving the recognizable object in either direction, it would produce a depth map displaying values that does not correspond to the actual location of the object. There were several attempts to synchronize the cameras, e.g. to the computer clock, but this proved to be unsuccessful. The effect of this was shown in Chapter 7, where the object was barely visible at all, with a lot of noise and false positives.

Secondly, in order to get a continuous stream of images from the cameras, the resolutions had to be dropped quite a bit. The highest resolution that gave semi-stable image flow was 640x480. This again lowers the accuracy of the stereo vision implementation in all steps, as it produces higher inaccuracies and more disturbance in the processed images. The current implementation is not sophisticated enough to take into account such factors.

As mentioned in the exploration of a suitable stereo vision system, there is a lot of hardware that would be better suited for this task, and which is also supported by ROS. For instance, Microsoft Kinect is affordable and highly versatile, as proven in previous theses. Specific recommendations will be given in the next chapter.

8.4 OCS

It should also be noted that the Drive Mode implementation is unfinished, as this was deemed to be out of the scope of this project. The focus has thus been on the Manipulator Mode. As a concept it shows some aspects of what can be done with regards to remote presence. All intended functionality of the Manipulator Mode was implemented successfully, apart from the absence of the produced depth maps, which is due to the problems with the stereo vision system discussed earlier. In retrospect, implementing the OCS on a Windows machine was probably a mistake, as the robustness and flow would have benefited greatly from being able to communicate directly with the ROS Master. Moving the OCS to a computer running as a ROS node should be a priority for future work, and will be recommended in the upcoming chapter.

8.4.1 Joystick

The Raw Input API used to collect the joystick data proved difficult to work with, and produced some odd readings. It was also difficult to find good documentation on how to properly use the functions, so the implementation became more or less trial and error, which was a lot more time-consuming than necessary. Having an easily accessible way to collect joystick input, and transmitting them in a suitable format, would be extremely beneficial. The fact that the Raw Input API locks the OCS implementation to a Windows platform is also a weakness, and in the event the OCS is moved to another operating system, the joystick data collector would have to be re-implemented.

Another issue with the joystick is that it seems to be extremely sensitive to movement, as it fluctuates in read values just from bumping into the table it is standing on. It also does not center itself, which means that if an increased dead zone is not implemented in the software, it will register movement even if left alone.

8.5 User manual

The user manual produced should work as intended, a standalone document providing an introduction to the system and overview of the system functionality. The overall usefulness of the user manual depends on updating the manual with new implementations, when and if they should occur. Should the user manual become outdated, it would become obsolete.

Recommendations and further work

9.1 Stereo vision system

As discussed earlier, the stereo camera rig on the robot manipulator should be replaced with a system that is more suitable. It could also be beneficial to reconsider the placement of the cameras, as the current placement means that the view will be obstructed for certain configurations of the arm.

To re-iterate the main points from the discussion, there needs to be found a more suitable stereo rig setup for computing real-time depth maps. The current market is full of good solutions, and a short list of possible setups will be presented here.

Time-of-flight cameras

Time-of-flight cameras work by illuminating the scene with a modulated light source, and observing the reflected light. The measured phase shift between the illumination and the reflection is then translated to distance. They also offer a low software complexity and fast response time compared to more traditional methods [22]. An example of such a camera is the Kinect for Xbox One [26].

Structured light scanners

Structured light scanners are based on detection the deformation of a pattern of light projected onto the surface of an object. A camera placed at a slight offset from the scanner records the shape of the line at an angle, and the distance to every point is calculated through a process similar to triangulation [2]. An example of cameras which utilize this is the Kinect for Xbox 360, which is currently in use on the mobile

platform.

Traditional stereo cameras

These are cameras with a minimum of two lenses, which are synchronized at hardware level. They use passive 3D sensing, which means that there are no lasers or projectors required. The process used is very similar to the implementation shown in Chapter 4. They can however be quite expensive. Examples of such camera rigs are the Zed 2k Stereo Camera produced by Stereo Labs [49] and the Bumblebee produced by FLIR [14].

If these changes are made, the stereo vision system should be re-implemented to give more successful results. ROS already include stacks and nodes for using this type of hardware to produce point clouds and depth maps, so an implementation shouldn't be too comprehensive. There is also a possibility to revert back to the system based on stereo infrared distance sensors, as used by Bekken [6]. These are currently taken off the robot and missing, the reasoning behind this is unknown.

9.2 Moving the OCS to Ubuntu

As discussed earlier, moving the OCS to Ubuntu would have several benefits. For one, it would provide access to the ROS message system through the ROS Master, which could eliminate the need for extra tcp servers. Having direct access to the published topics could also greatly increase the potential for feedback to the operator, as most states and messages would be easily accessible.

9.3 Joystick

If the recommendation to move the OCS to the Ubuntu operating system is followed, a new method for accessing the joystick would need to be implemented. ROS provides joystick drivers and a *joy_node* that interfaces a generic joystick to ROS and publishes the joystick states¹. This would also give a better and more stable method of collecting joystick input. Continuing the point from the previous section about joystick sensitivity, it might be advantageous to try to fix the looseness of the joystick somehow, or look for a replacement.

9.4 Eliminating the USB Controller

As discussed in the previous chapter, the situation with controlling the robot through the USB Controller does not suit the overall design of the system in ROS. Although the MATLAB interface works sufficiently well, having to run a virtual machine adds unnecessary complexity to the system as a whole. Thus, finding a solution for accessing the SCORBOT-ER 4u directly should be a priority, or at least exploring solutions for making the USB Controller compatible with Ubuntu.

¹<http://wiki.ros.org/joy>

Bibliography

- [1] Aksel A. Transeth, Øystein Skotheim, Henrik Schumann-Olsen, Gorm Johansen, Jens Thielemann, Erik Kyrkjebø. A Robotic Concept for Remote Maintenance Operations: A Robust 3D Object Detection and Pose Estimation Method and a Novel Robot Tool. In *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2010.
- [2] Andreas Georgopoulos, Charalambos Ioannidis and Artemis Valanis. Structured Light Scanners. <https://www.gim-international.com/content/article/structured-light-scanners>. Accessed: 10.06.2017.
- [3] David A. Anisi and Charlotte Skourup. A step-wise approach to oil and gas robotics. In *2012 IFAC Workshop on Automatic Control in Offshore Oil and Gas Production*, Trondheim, Norway, 2012.
- [4] Nnamdi Anyadike. Could unmanned platforms provide the boost the north sea needs? <http://www.offshore-technology.com/features/featurecould-unmanned-platforms-provide-the-boost-the-n> November 2016. Accessed: 06.02.2017.
- [5] Petter Aspunvik. Robotisert vedlikehold. Master's thesis, Norwegian University of Science and Technology, 2013.
- [6] Kristian Saxrud Bekken. Bevegelsesstyring av robotarm og kamera med kollisjonsunngåelse. Master's thesis, Norwegian University of Science and Technology, 2010.

- [7] Mikael Berg. Navigation with Simultaneous Localization and Mapping For Indoor Mobile Robot. Master's thesis, Norwegian University of Science and Technology, 2013.
- [8] Tommy Berntzen. Mobile autonomous robot: Remote operation. Project thesis, Norwegian University of Science and Technology, 2016. 9th semester specialization project report.
- [9] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly Media, 2008.
- [10] Heping Chen, Samuel Stavinoha, Michael Walker, Biao Zhang, and Thomas Fuhlbrigge. Opportunities and challenges of robotics and automation in offshore oil & gas industry. *Intelligent Control and Automation*, 5(3):136 – 145, 08 2014.
- [11] Clearpath Robotics. ROS 101: Intro to the Robot Operating System. <http://www.ros.org/core-components/>. Accessed: 17.02.2017.
- [12] Oracle Corporation. Oracle VM VirtualBox® User Manual. <https://www.virtualbox.org/manual/>. Accessed: 18.02.2017.
- [13] Joel M. Esposito, Carl Wick, and Ken Knowles. Matlab toolbox for the intelitek scorbot: An open source robotics education library. In *American Society of Engineering Education Annual Conference*, Vancouver, BC, 2011.
- [14] FLIR. Stereo Vision. <https://www.ptgrey.com/stereo-vision-cameras-systems>. Accessed: 10.06.2017.
- [15] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

- [16] M.A. Gunter, Matthew M., M.P.H. Hill, Ryan, MS O'Connor, Mary B., M.P.H. Retzer, Kyla D., and PhD. Lincoln, Jennifer M. Fatal Injuries in Offshore Oil and Gas Operations - United States, 2003-2010. Technical report, U.S. Center for Disease Control, Apr 26 2013.
- [17] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2004.
- [18] Intelitek Inc. *Controller-USB User Manual*, 2002. Found on <http://www.intelitekdownloads.com/Manuals/Robots/ER-4u/>, Accessed: 12.01.2017.
- [19] IntRoLab. Stereo mapping. <https://github.com/introlab/rtabmap/wiki/Stereo-mapping>. Accessed: 02.04.2017.
- [20] John England. Oil and Gas Industry Outlook 2017. <https://www2.deloitte.com/us/en/pages/energy-and-resources/articles/oil-and-gas-industry-outlook.html>. Accessed: 04.04.2017.
- [21] Kristian Eckhoff. Hmd-styrt robot. Master's thesis, Norwegian University of Science and Technology, 2005.
- [22] Larry Li. Time-of-Flight Camera – An Introduction. <http://eu.mouser.com/applications/time-of-flight-robotics/>. Accessed: 10.06.2017.
- [23] Vegard Stjerna Lindrup. Robotic Maintenance and ROS. Master's thesis, Norwegian University of Science and Technology, 2016.
- [24] MathWorks. Robotics System Toolbox. <https://se.mathworks.com/products/robotics.htmlx>. Accessed: TBA.
- [25] Michael D. M. Kutzer. MATLAB Toolbox for Intelitek Scorbot-ER 4U. <https://www.usna.edu/Users/weapsys/kutzer/>

- `_Code-Development/ScorBot_Toolbox.php`. Accessed: 12.01.2017.
- [26] Microsoft. Kinect hardware. <https://developer.microsoft.com/en-us/windows/kinect/hardware>. Accessed: 10.06.2017.
- [27] Microsoft. Raw Input Reference. [https://msdn.microsoft.com/en-us/library/windows/desktop/ff468895\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ff468895(v=vs.85).aspx). Accessed 24.03.2017.
- [28] MoveIt! SCORBOT-ER 4u. <http://moveit.ros.org/robots/scorbot-er4u/>. Accessed: 12.01.2017.
- [29] Oljedirektoratet. Nytt utbyggingskonsept pa Oseberg. <http://www.npd.no/no/Nyheter/Nyheter/2016/Nytt-utbyggingskonsept-pa-Oseberg/>. Accessed: 06.02.2017.
- [30] OpenCV. About opencv. <http://opencv.org/about.html>. Accessed: 13.04.2017.
- [31] opencv dev team. Opencv 2.4.13 documentation. <http://docs.opencv.org/2.4.13/modules/core/doc/intro.html>. Accessed: 13.04.2017.
- [32] Qt Wiki. About Qt. http://wiki.qt.io/About_Qt. Accessed: 22.11.2016.
- [33] Ramboll Oil & Gas. Unmanned wellhead platforms – UWHP summary report. <http://www.npd.no/no/Publikasjoner/Rapporter/UNMANNED-WELLHEAD-PLATFORMS-/>, March 2016. Accessed: 24.11.2016.

- [34] Dr. Philip Hodgson. Tips for writing user manuals. <http://www.userfocus.co.uk/articles/usermanuals.html>, June 2007. Accessed 30.01.2017.
- [35] ROS. About ROS. <http://www.ros.org/about-ros/>. Accessed: 02.02.2017.
- [36] ROS. camera_calibration. http://wiki.ros.org/camera_calibration. Accessed: 23.03.2017.
- [37] ROS. camera_driver. http://wiki.ros.org/camera_drivers. Accessed: 23.03.2017.
- [38] ROS. Concepts. <http://wiki.ros.org/ROS/Concepts>. Accessed 25.02.2017.
- [39] ROS. Core Components. <http://www.ros.org/core-components/>. Accessed: 02.02.2017.
- [40] ROS. cv_bridge. http://wiki.ros.org/cv_bridge. Accessed: 23.02.2017.
- [41] ROS. flir_ptu_driver. http://wiki.ros.org/flir_ptu_driver. Accessed 10.02.2017.
- [42] ROS. image_pipeline. http://wiki.ros.org/image_pipeline?distro=indigo. Accessed: 23.03.2017.
- [43] ROS. Introduction. <http://wiki.ros.org/ROS/Introduction>. Accessed: 17.02.2017.
- [44] ROS. ROS Wiki. <http://wiki.ros.org/ROS/>. Accessed 25.02.2017.
- [45] ROS. stereo_image_proc. http://wiki.ros.org/stereo_image_proc. Accessed: 23.03.2017.

- [46] Linda Shapiro and George Stockman. *Computer Vision*. Prentice Hall, 2001.
- [47] Ole Magnus Siqveland. Remote Operations using Oculus Rift, Leap Motion and Microsoft Kinect. Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2016.
- [48] Mark W. Spong, Seth Hutschinson, and M. Vidyasagar. *Robot modeling and control*. Wiley, Hoboken, N.J, 2006.
- [49] Stereo Labs. Meet ZED. <https://www.stereolabs.com/zed/specs/>. Accessed: 10.06.2017.
- [50] The MathWorks, Inc. Computer Vision System Toolbox. <https://se.mathworks.com/help/vision/index.html>. Accessed: 23.03.2017.
- [51] The Qt Company. Qt Documentation All Modules. <https://doc.qt.io/qt-5/qtmodules.html>. Accessed 30.03.2017.

MAR user manual index

Contents

| Mobile autonomous robot | User manual |
|---|-------------|
| List of Figures | 3 |
| 1 Introduction | 4 |
| 1.1 Full overview | 4 |
| 2 Hardware | 5 |
| 2.1 Server (on-board computer) | 5 |
| 2.2 Robot manipulator | 6 |
| 2.3 Omni wheels | 6 |
| 2.4 Joystick | 7 |
| 2.5 Routers | 7 |
| 2.6 IP-Cameras | 7 |
| 2.7 Kinect sensor | 8 |
| 2.8 LIDAR | 8 |
| 2.9 Motorcontroller and encoder | 8 |
| 2.10 Batteries | 9 |
| 2.11 Pan-tilt unit | 9 |
| 2.12 Power inverters | 10 |
| 2.13 Operator control station | 10 |
| 3 Cables and connections | 11 |
| 3.1 Schematics | 11 |
| 4 Software | 15 |
| 4.1 Server (on-board computer) | 15 |
| 4.2 Client (OCS) | 15 |
| 5 Configuration and installation | 16 |
| 5.1 Installing ROS and setting up workspace | 16 |
| 5.2 Installing the virtual machine | 16 |
| 5.3 Setting up the routers and IP cameras | 16 |
| 5.4 ROS packages | 19 |
| 5.5 Mobile platform | 19 |
| 5.6 Robot manipulator | 19 |
| 5.7 Operating Control Station | 19 |
| 6 Running the system | 20 |
| 7 Troubleshooting | 21 |
| 7.1 Connecting to the Intelitek USB Controller from virtual machine | 21 |
| 7.2 Pan-tilt unit | 21 |
| 7.3 Joystick | 21 |
| 8 References | 22 |

DVD Contents

This is an outline of the folder structure on the enclosed DVD.

- 3rd-party tools
- Implementation files
 - Scor ROS Node (MATLAB)
 - MAR (ROS)
 - OCS (Qt)
- MAR User Manual
 - External user manuals
- Previous theses
- Thesis

Camera matrices

C.1 Intrinsic matrices

M is the camera matrix and D is the distortion coefficients for the respective cameras. Values are rounded for convenience.

$$M_{left} = \begin{bmatrix} 795.8546 & 0 & 319.9971 \\ 0 & 796.8261 & 239.3465 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D_{left} = \begin{bmatrix} -0.0414 & -0.6801 & 0 & 0 & 0 & 0 & 0 & -1.0385 \end{bmatrix}$$

$$M_{right} = \begin{bmatrix} 795.8546 & 0 & 321.2961 \\ 0 & 796.8261 & 241.9785 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D_{right} = \begin{bmatrix} 379.2752 & 0 & 194.3849 & 0 & 373.1992 & 81.0741 & 0 & 0 \end{bmatrix}$$

C.2 Extrinsic matrices

R is the rotation matrix between the first and second camera coordinate systems and T is the translation matrix between the coordinate systems. R_1 and R_2 are the rectification transforms for the respective cameras, P_1 and P_2 are the projection matrices in the respective rectified coordinate systems and Q is the reprojection matrix.

$$R = \begin{bmatrix} 0.9986 & -0.0048 & -0.0160 \\ 0.0049 & 0.999 & 0.0074 \\ 0.0160 & -0.0075 & 0.999 \end{bmatrix}$$

$$T = \begin{bmatrix} -63.7659 & -2.6956 & 2.9457 \end{bmatrix}$$
$$Q = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & -336.3675 \\ 0.0000 & 1.0000 & 0.0000 & -230.2329 \\ 0.0000 & 0.0000 & 0.0000 & 322.4315 \\ 0.0000 & 0.0000 & 0.01565 & 0.0000 \end{bmatrix}$$

Installation and configuration

D.1 Hardware setup

The hardware setup is extensively detailed in the MAR User Manual, and will thus not be presented here. The password for both the on-board computer and the virtual machine is *1q2w3e*.

D.2 Installation

For the server side, it is assumed that ROS Indigo is installed on a suitable system, like Ubuntu 14.04 used. Guides for installing ROS or Ubuntu can be found on the respective official sites. In addition to a full install of ROS Indigo, the following packages must be installed:

web_video_server

```
sudo apt-get install ros-indigo-web-video-server
```

flir_ptu_driver Installed from source:

```
https://github.com/ros-drivers/flir\_ptu
```

cv_bridge

```
sudo apt-get install ros-indigo-cv-bridge
```

Next a VM running Windows 7 32-bit needs to be installed and set up on the on-board computer running Ubuntu. If using Virtualbox, a guide for this can be found at <https://www.virtualbox.org/manual/ch02.html>. On the VM, install a 32-bit version of MATLAB with Robotics System Toolbox, for example R2015b. Download and install the ScorBotToolbox, a guide

is found on the Mathworks fileexchange <https://se.mathworks.com/matlabcentral/fileexchange/52830-kutzer-scorbottoolbox>.

On the OCS side, ensure that Qt 5.5 or later is installed, downloads and guides can be found at <https://www.qt.io/>, as well as a suitable C++ compiler providing MSVC2015, for example Microsoft Visual Studio 2015.

D.3 Configuring the project

D.3.1 ROS workspace

The first step is to create a catkin workspace, a guide for doing this is found on the ROS website. Move the packages located in the DVD folder *Implementation files/MAR* into the *catkin_ws/src* folder.

D.3.2 Virtual machine

Configure the VM and network settings according to the aforementioned guide. Make sure that the network connection is set up such that communication is possible both to and from the VM. Also make sure that the ERUSB.sys, the driver for the SCORBOT-ER 4u, is located in the appropriate folder. Move the MATLAB node located in the DVD folder *Implementation files/ScorROSNode* to the active folder in MATLAB.

D.3.3 Network configuration

The MAR network should be configured, if it is not a comprehensive guide is found in the MAR User Manual.

D.4 System launch

This assumes that the ROS source code is located in the src folder of the catkin workspace, and that the virtual machine and ScorBotToolbox are suc-

cessfully installed and configured. Firstly, check the following hardware setup on the on-board computer:

1. Verify that the USB Controller is connected to a USB port the on-board computer, and accessible from the VM.
2. Check that the PTU is connected through USB, and that the control box is powered on.
3. Ensure that the router is connected to the computer, and that the IP Cameras are connected to the router.
4. Verify that all hardware is connected to power and operational

On the client computer:

1. Ensure that the joystick is powered and connected to a USB port.
2. Ensure that the router is powered and connected to the computer.

Then on the on-board computer:

1. Open a terminal and cd to the created catkin workspace.
2. Launch roscore.
3. Launch the VM, start MATLAB and run the ScorROSInit file. Verify that the SCORBOT-ER 4u is initializing and homing.
4. In a second window, cd to the catkin workspace, and run `$ source ./devel/setup.bash`.
5. Bring up the necessary nodes with `$ roslaunch mar_scor mar_scor.launch`.

On the OCS computer, copy the OCS project located in the DVD folder *Implementation files/OCS(Qt)* to a suitable location, and run the project through the Qt IDE.

Special note: In order for the MATLAB node to be able to read the messages

sent on the ROS topics, the `ROS_MASTER_URI` and `ROS_IP` needs to be exported. This can be done by opening up the bash file located at `~/.bashrc` and adding these two lines at the end:

1. `export ROS_MASTER_URI=http://ip_addr:11311`
2. `export ROS_IP=ip_addr`

where `ip_addr` is the IP address of the computer running *roscore*. Remember to *source* `/.bashrc` after the file has been edited.

Troubleshooting

E.1 Connecting to the Intelitek USB Controller from virtual machine

The virtual machine seems to have problems locating the driver for the USB Controller, even after the driver is installed. The following checklist should solve the problem

1. Verify that the file *ERUSB.sys* is located at
C:\Windows\system32\drivers
 - If the file is not found, copy from DRIVERS\Scorbot on the CD
2. Locate the device in *Device Manager*
3. Right-click on the device and select *Check for hardware updates*
4. The device should now be recognized as *ERUSBCLASS\ER USB robot controller*, and the problem should be solved

After installing the Scorbot Toolbox for MATLAB, it is important to run ScorUpdate, or else the Toolbox will not be able to gain access to the Scorbot!

E.2 Pan-tilt unit

Currently the pan-tilt unit is controlled by a ROS package ¹. There has been some problems with this setup, mainly that it cannot connect to the PTU or that it cannot send/receive to/from the PTU.

- Check that the user has read/write permissions for the device

¹http://wiki.ros.org/flir_ptu_driver

- E.g. run the command `ls -l /dev/tty*`, where `tty*` is the device location
- If read/write permissions are off, set them using the command `chmod 666 /dev/tty` and/or add the user to the `dialout` group

As stated, there has been experienced some trouble with getting the drivers to communicate with the PTU. Through some testing, it was found that the standard baud rate of 9600 would cause the symbols sent to be scrambled, and so the first thing to do should be to set the baud rate to 38400. If this still doesn't work, downloading the package from source and building it like a ROS package in your workspace seems to do the trick.