MASTER THESIS

A Mixed-Reality Platform for Robotics and Intelligent Vehicles

School of Information Science, Computer and Electrical Engineering
Halmstad University - Sweden
in Cooperation with
Information Technology and Systems Management
University of Applied Sciences Salzburg - Austria

Norbert Grünwald, BSc

Supervisors: Roland Philippsen, Ph.D.
FH-Prof. DI Dr. Gerhard Jöchtl

Halmstad, May 2012
A Mixed-Reality Platform for Robotics and Intelligent Vehicles

Master Thesis
Halmstad, May 2012

Author: Norbert Grünwald, BSc
Supervisors: Roland Philippsen, Ph.D.
            FH-Prof. DI Dr. Gerhard Jöchtl
Examiner: Prof. Antanas Verikas, Ph.D.

School of Information Science, Computer and Electrical Engineering
Halmstad University
PO Box 823, SE-301 18 HALMSTAD, Sweden
Author’s Declaration

I, Norbert GRÜNWALD born on 16.12.1979 in Schwarzach, hereby declare that the submitted document is wholly my own work. Any parts of this work, which have been replicated whether directly or indirectly from external sources, have been properly cited and referenced.

Halmstad, May 31, 2012

______________________________
Norbert GRÜNWALD

______________________________
Personal number
I hear, I know. I see, I remember. I do, I understand.

Confucius

I want to thank my supervisors Roland Philippsen, Ph.D. and FH-Prof. DI Dr. Gerhard Jöchtl for their guidance. Their ideas and suggestions were a much appreciated help for the realization of the project and this thesis. I also want to thank Björn Åstrand, Ph.D. and Tommy Salomonsson M.Sc. for providing me with hardware and tools, so that I could build the system.

My deepest gratitude goes out to my family, especially to my parents Johann and Hannelore Grünwald, whose support made it possible for me to pursue my studies.
Details

First Name, Surname: Norbert GRÜNWALD  
University: Halmstad University, Sweden  
Degree Program: Embedded and Intelligent Systems  
            Intelligent Systems Track  
Title of Thesis: A Mixed-Reality Platform for Robotics and Intelligent Vehicles  
Keywords: Mixed Reality, Robotics, Intelligent Vehicles  
Academic Supervisors: Roland Philippsen, Ph.D.  
            FH-Prof. DI Dr. Gerhard Jöchtl

Abstract

Mixed Reality is the combination of the real world with a virtual one. In robotics this opens many opportunities to improve the existing ways of development and testing. The tools that Mixed Reality gives us, can speed up the development process and increase safety during the testing stages. They can make prototyping faster and cheaper, and can boost the development and debugging process thanks to visualization and new opportunities for automated testing.

In this thesis the steps to build a working prototype demonstrator of a Mixed Reality system are covered. From selecting the required components, over integrating them into functional subsystems, to building a fully working demonstration system.

The demonstrator uses optical tracking to gather information about the real world environment. It incorporates this data into a virtual representation of the world. This allows the simulation to let virtual and physical objects interact with each other. The results of the simulation are then visualized back into the real world.

The presented system has been implemented and successfully tested at the Halmstad University.
Contents

1 Introduction
  1.1 Mixed Reality ............................................ 1
  1.2 Benefits of Mixed Reality in Robotics ..................... 2
  1.3 Social Aspects, Sustainability and Ethics .................. 3
  1.4 Problem Formulation and Project Goals .................... 4
  1.5 Summary ................................................. 5

2 Building Blocks of a Mixed Reality System .................. 7
  2.1 Components and Subsystems ................................ 7
  2.2 Middleware ............................................... 8
    2.2.1 ROS .................................................. 9
  2.3 Simulator ................................................. 12
    2.3.1 Webots .............................................. 13
  2.4 Sensors .................................................. 15
    2.4.1 Laser Scanner ......................................... 16
    2.4.2 Camera ............................................... 17
  2.5 Robot .................................................... 18
    2.5.1 Selection Criteria .................................... 18
    2.5.2 Actual Models ........................................ 19
  2.6 Tools ..................................................... 20
    2.6.1 Tracking System ....................................... 20
    2.6.2 Visualization of Virtual Objects ...................... 22
    2.6.3 Coordinate Transformation ............................ 24
  2.7 Summary .................................................. 26

3 Implementation ............................................. 27
  3.1 Used Hardware and Software ................................ 27
  3.2 Implementation and Interaction .............................. 29
    3.2.1 Overview .............................................. 29
    3.2.2 Visualization of Sensor Data .......................... 31
    3.2.3 Mix Real World Camera Data with Simulation ......... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.4</td>
<td>Teleoperation</td>
<td>35</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Tracking of Physical Objects</td>
<td>36</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Visualization of Virtual Objects</td>
<td>38</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Robot Control</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Summary</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Demo System</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview of the Demo System</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Hardware and Software</td>
<td>42</td>
</tr>
<tr>
<td>4.3</td>
<td>Integration</td>
<td>43</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Robot</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Example of Interaction</td>
<td>46</td>
</tr>
<tr>
<td>4.5</td>
<td>Summary</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Results</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Discussion</td>
<td>51</td>
</tr>
<tr>
<td>5.3</td>
<td>Outlook</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>Acronyms</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>Product References</td>
<td>56</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>A.1</td>
<td>Webots Installation Information</td>
<td>59</td>
</tr>
<tr>
<td>A.2</td>
<td>Bounding Box for Laser Scanner</td>
<td>59</td>
</tr>
<tr>
<td>A.3</td>
<td>Video Input Device Driver for IP-Cameras</td>
<td>59</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Overview of some important ROS commands. .................. 11
2.2 Features and specifications of the SICK LMS-200 ................ 16
2.3 Feature comparison of the two used cameras. .................. 18
2.4 Markers for object tracking. ................................. 21

3.1 Specifications of the Linux host .............................. 28
3.2 Specifications of the Windows (Matlab) host .................. 28

4.1 Packet format.................................................. 44
4.2 VU Message used to steer the robot. ......................... 45
4.3 Control characters used in the framing. ....................... 45

5.1 Assessment of the project goals. ............................. 50
5.2 Rating symbols for project assessment. ....................... 50

List of Listings

3.1 LaserScan message ............................................ 33
3.2 Image message ............................................... 35
3.3 Joy message .................................................. 36
3.4 Matlab Position Message ...................................... 37
3.5 Position message ............................................... 37
3.6 Map message .................................................. 39
3.7 Twist message .................................................. 40

4.1 Packet format .................................................. 44
4.2 PIE message for robot steering .............................. 45
<table>
<thead>
<tr>
<th></th>
<th>99-matrix.rules</th>
<th></th>
<th>59</th>
</tr>
</thead>
</table>
## List of Figures

1.1 Interaction of physical and digital objects. ........................................ 2
2.1 Major parts of a Mixed Reality system. ........................................... 8
2.2 Visualization of ROS nodes using rxgraph. ...................................... 9
2.3 User interface of the Webots simulator. ........................................... 13
2.4 SICK LMS-200. ................................................................. 16
2.5 Field of view .............................................................................. 16
2.6 Sony SNC-RZ30 .......................................................................... 17
2.7 Prosilica GC1350C ................................................................. 17
2.8 Considerations for robot evaluation. ............................................... 19
2.9 Alfred ....................................................................................... 20
2.10 PIE. .......................................................... .......................... 20
2.11 Khepera III. ................................................................. 20
2.12 Visual output of the tracking software. ........................................... 21
2.13 Principle of the visualization. ..................................................... 23
2.14 Image of the real projection. ..................................................... 24
2.15 Principles of homography. ......................................................... 25
3.1 Overview of the system’s different modules and components. ............. 29
3.2 Message flow for visualization of real sensor data. .............................. 31
3.3 Message flow for visualization of simulated sensor data. ..................... 32
3.4 Overlay of sensor data onto the virtual scene. ................................... 32
3.5 Real-world image data fed into the controller of a simulated robot. ....... 34
3.6 Integration of real world camera data into the simulation. ................. 34
3.7 Message flow for teleoperation. .................................................. 35
3.8 Message flow of the object tracking. ............................................. 36
3.9 Control flow of the tracking software ............................................. 38
3.10 Message flow of the visualization subsystem .................................... 38
3.11 Message flow of the robot control ............................................... 40
3.12 Message flow of the robot control with additional map information .... 40
4.1 Demo System ............................................................................. 41
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Projector and camera mounted to the ceiling</td>
<td>42</td>
</tr>
<tr>
<td>4.3</td>
<td>Wii Controller</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Message flows</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>Connection between robot and Mixed Reality system</td>
<td>44</td>
</tr>
<tr>
<td>4.6</td>
<td>Robot approaches the ball</td>
<td>46</td>
</tr>
<tr>
<td>4.7</td>
<td>Robot “kicks” the ball</td>
<td>46</td>
</tr>
<tr>
<td>4.8</td>
<td>Ball rolls away</td>
<td>46</td>
</tr>
<tr>
<td>4.9</td>
<td>Physical robot and virtual object</td>
<td>47</td>
</tr>
</tbody>
</table>
1

Introduction

Development and testing of robots can be a costly and sometimes dangerous process. The use of Mixed Reality (MR) technologies can help to reduce or even avert these difficulties [1]. But research is not the only field that can benefit from Mixed Reality. MR can also be a valuable tool in education [2].

1.1 Mixed Reality

Due to vast improvements in processing power and sensor technologies, the fusion of the “Real World” with “Virtual Information” has become more and more powerful and useable. This combination of the physical and the virtual world is called Mixed Reality. MR is the genus for a broad field of applications. It is divided into two major subgroups, Augmented Reality (AR) and Augmented Virtuality (AV) [3].

The more prominent of these subgroups is Augmented Reality. AR applications are used to enrich the physical world with virtual data. It can provide the user with additional information about its surrounding. Augmented Reality has been used for many years, mainly in military applications like heads-up displays for fighter jet pilots and similar devices. But with extensive improvements in consumer technologies, especially with the rise of smart-phones, Augmented Reality has become known and available to the general public. Nowadays it is used in many forms, like in driving assistance systems [33], toys for children [34] or for entertainment purposes [35]. Another nice and promising example of Augmented Reality is Google’s Project Glass [36] which is currently under development.

The second subgroup is Augmented Virtuality. In AV, real world objects are “transferred” into the virtual space where they can interact with simulated entities. An example for AV are virtual conferencing and collaboration systems [4]. The users are placed in virtual conference rooms, where they can interact with each other.
1. Introduction

Fig. 1.1: Interaction of physical and digital objects.

1.2 Benefits of Mixed Reality in Robotics

Mixed Reality can help to speed up the development, especially during the debugging, integration and testing stages [1].

Advantages of Mixed Reality:

- Faster prototyping
- Separated testing
- Repeatability
- Comparability of test results
- Automation
- Visualization
- Safety
- Lower costs

Using MR technologies allows for faster prototyping of new ideas. If certain hardware parts or environmental requirements are not accessible, they can be simulated, while the rest of the system can run on real hardware interacting with physical objects.

The ability to simulate parts of the system, allows for a better separation while testing individual modules. This prevents distractions and interferences due to problems in other parts of the system, like a malfunctioning sensor. Large-scale robotic systems consist of many different modules, based on very different fields of engineering. Often it is very difficult for a single person to cover all aspects which are required for the operation of
the whole system [5]. Being able to separate the modules and leave all “unnecessary” parts to the simulation, where a perfect behavior can be guaranteed every time, reduces dependencies, interferences and side-effects, and lets the developer concentrate on his current task.

With MR testing can be automated and test cases can be repeated. Having the opportunity to repeat tests exactly the same way as before, gives a better comparability of achieved results. The behavior and reactions of a robot to certain input can be better analyzed, debugged and compared.

Mixed Reality also gives the developer new tools to visualize and interpret data. Visualization can give engineers a better understanding of the robot’s view of the world and support the debugging of otherwise hard to find errors [1].

When it comes to testing, MR can help to increase safety while speeding up the testing [6]. Testing certain features, like safety systems that should prevent physical contact between the robot and objects in it’s environment, can be risky and time consuming. Tests have to be carried out very carefully to make sure that everything works like expected and to prevent crashes in case of malfunctions. With Mixed Reality these tests can be speed up. MR can feed the robot with simulated sensor data of his environment. This sensor data can contain virtual obstacles that the robot has to avoid. If the machine does not react as expected, then there is no physical contact and therefore no harm to the equipment or humans.

Each of these single advantages alone, already leads to reduced costs. Combined, the savings can be tremendous.

1.3 Social Aspects, Sustainability and Ethics

A big part of the research at the Intelligent Systems Lab [7] at Halmstad University has to do with developing intelligent vehicle technologies. These vehicles are destined to increase productivity while simultaneously improving safety at the workplace. Another field of research is the development of new intelligent vehicles and safety features for regular cars. Improvements in this area can be directly transferred into new products that help to prevent accidents and safe lives.

As already stated in chapter 1.2, Mixed Reality can help to speed up development and saving costs. Through the faster, safer development and testing and due to the reduction in costs, MR can help to bring advancements in safety systems quicker to the market. All of this while retaining the same quality and reliability, or perhaps even improve it. But MR is not only speeding up the time-to-market, in many cases it actually makes it possible
to put such systems on the market in the first place. Because legislators and insurance companies need to rely on thorough testing before allowing next-generation active safety systems onto public roads.

To make sure that active safety systems actually work and are safe, a new project has been started recently. It is called *Next Generation Test Methods for Active Safety Functions* (NG-TEST) [8] and will focus on developing and establishing a framework for validation and verification of these safety systems. One entire work-package of this project is dedicated to investigating Mixed Reality for automated tests of automotive active safety systems.

Another big advantage of Mixed Reality are it’s capabilities for use in education [9]. Practically orientated courses in robotics require actual hardware for students to work with. But high costs of the components and limited budgets pose a hurdle. Often there is not enough hardware available, so students need to share or rely on simulations only. Sharing is a problem, because it creates artificial delays and breaks that can have a negative influence on motivation and also on the reputation of the course. Simulation on the other hand can often be seen as boring. Having the ability to work with a robot that you can actually touch and interact with can be far more motivating for students than just staring at a computer screen. Mixed Reality can help here too. Students don’t need a whole robot anymore. They can start to implement with the help of the simulation and then - for example - switch from virtual to real sensors. Through coordination, delays can be reduced or even eliminated. Small tests that otherwise would block a whole robot, can now be split and the available hardware can be shared more efficiently. Other good examples that show how Mixed Reality can be used in education, can be found in the papers by Gerndt and Lüssem[10] and by Anderson and Baltes [2].

In summary the advantages that Mixed Reality offers, make research and education more cost effective, secure and efficient.

### 1.4 Problem Formulation and Project Goals

As we have learned from the previous chapters, research and development of robots is a tedious and costly process. Especially during the debugging and testing phase, a lot of time and money is spent to ensure a safe and correct behavior of the machine. With the help of MR these problems can be diminished.

The reason for this Master Thesis is to create a system that can serve as a basic foundation for the research of Mixed Reality at the Halmstad University Intelligent Systems Laboratory [7]. The outcome of the project should cover the basic needs and requirements for a MR system and fulfill the following aspects:
• A mobile robot simulator
• A physical mobile robot coupled to the simulator
• A simple yet effective teleoperation station
• Extensive documentation to build on these foundations
• Visualization of real sensor data
• Injection of simulated sensor data into physical robot

Chapter 5.1 will come back to this definitions and compare the expected goals with the actual outcome of the project.

1.5 Summary

This chapter has shown that Mixed Reality is a promising new tool for the development of robots and intelligent vehicles. It offers many advantages for education and research. The next chapters will present, how the goals that have been defined here can be realized to create a basic MR framework.
Building Blocks of a Mixed Reality System

This chapter deals with the many different technologies that are required to build a Mixed Reality system. It will give an overview about the different subsystems and modules that are needed and it will explain the reasons behind the selection of certain tools and technologies.

2.1 Components and Subsystems

As we have learned in chapter 1.1, Mixed Reality incorporates many different areas of robotics and software engineering.

Some of the parts that are required for a MR system include:

- Sensors
- Computer vision
- Simulation
- Distributed systems architecture
- Computer graphics

A Mixed Reality system can utilize various different sensors to gather knowledge about the physical world. This includes the environment but also the objects in it. Sensors can be used to retrieve information like the location and size of a robot or obstacle. Often cameras are used to acquire this kind of information. In this case Computer Vision methods and algorithms are required to analyze the camera images and to extract the required data. On the virtual side, a simulator is used to (re-) create the physical environment and to fill it with digital objects, like robots or obstacles. To connect all the different parts, a distributed systems architecture is advisable. It allows for a flexible and modular design.
and supports the creation of re-usable modules. Once all the different kinds of data have been collected, mixed and merged, it is time to present the result to the user. This can be realized in many different ways. From basic 2D representations to expensive 3D visualization, the possibilities are numerous.

In addition to these "core" components, usually some kind of physical robot is needed too. When the Mixed Reality system is connected to the robot, it can be used for tele-operation. If the robot has an on-board camera, the images recorded by the robot can be streamed back to the tele-operation station. The received frames can then be augmented with additional information and presented to the user. In addition to that, Mixed Reality can inject “fake” sensor data into the control logic of the robot and let it react to virtual objects.

Fig. 2.1 shows a rough overview of the different types of components or sub-systems. As can be seen, the central part of the system is the middleware. The middleware acts as the backbone of the MR system and connects all the other components with each other.

![Diagram of Mixed Reality system components](image)

Fig. 2.1: Major parts of a Mixed Reality system.

### 2.2 Middleware

The middleware is (one of) the most important parts in the MR system. As requirements on the system are changing from project to project, the ability to quickly adopt, replace and change components is of utmost importance. Using a flexible system, enables us to easily add and remove sensors and other hardware but also allows to replace core components like the simulator with tools that might be better suited for the new task.

There is a number of suitable systems which can be considered for this job. Systems like Player [11], Orca [12], YARP [13] or ROS [5] all have comparable feature sets and
fulfill the requirements that are needed. Some related MR projects have even decided to implement their own middleware to solve specific needs [14]. There are a number of papers who deal with the different systems, give an overview about their advantages and disadvantages and compare the features [15, 16]. Influenced by the findings of these papers the decision was made to use the so called Robot Operating System (ROS) as the middleware.

One of the most important factors for this decision is the flexible and modular design of ROS. But also the easy integration into existing software packets is very important. Because of its lightweight design it allows for a quick and trouble free integration of the robot simulator and other important parts.

2.2.1 ROS

Despite the name, ROS is not a typical operating system. It is a set of tools and programs that can be used to implement robots. The goal is to provide an environment that eases the reuse of existing code and to create a set of standard functions and algorithms that can be used for different applications. ROS is a framework that supports rapid prototyping and is designed to be used for large-scale integrative robotics research [5].

ROS is open source. The main development and hosting is carried out by Willow Garage [17]. The system is used in numerous projects around the world and therefore it is well tested. It comes with many existing interfaces and drivers for common sensors and other hardware. It also features test and trace tools for debugging, as well as tools for visualization of various data streams. Recording and playback of dispatched messages allows for easy reproducibility of system activity.

Fig. 2.2: Visualization of ROS nodes using rxgraph.
ROS design criteria [5]:

- Peer-to-peer
- Tools-based
- Multi-lingual
- Thin
- Free and Open-Source

ROS splits its different modules in so called packages. These packages contain nodes, services and messages. These nodes and services are the building blocks of the system. They are used to control hardware components, read sensors but also to offer algorithms and functions to other modules. Every node is a process running on a host. They can all run on the same host, but it is also possible to distribute the system over multiple hosts. This way the processing load can be spread and computation intensive task can be outsourced to dedicated machines. The communication between these nodes is done via messages. Therefore a node first has to register at a central core, called the master, and name the kind of messages it wants to receive and publish. The master is used to let the individual nodes find each other. The communication between individual nodes is based on decentralized peer-to-peer methods. [5][37]

To group and organize related packages, ROS uses so called stacks. Stacks have the ability to instantiate multiple nodes at the same time, using a single command. This is an important feature, especially in large scale projects where numerous modules work together. The demo system, which is described in chapter 4, uses a stack to launch the single nodes that are required for operation. [5][37]

Using ROS as middleware opens up a repository of existing hardware drivers, algorithms and functional modules. Connecting a ROS based robot to the system becomes a lot easier, as the control modules just have to connect to the existing master node and can start to interact with the rest of the system. If a stronger separation is required, ROS offers namespaces to separate different groups of nodes. So in case, one would like to have a stronger separation between the MR system’s modules and the robot’s modules, namespaces can be used to achieve that. Alternatively a second instance of roscore can be launched to completely separate the different systems. [5][37]
Tbl. 2.1: Overview of some important ROS commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roscore</td>
<td>Starts the ROS master</td>
</tr>
<tr>
<td>roslaunch</td>
<td>Start packages and stacks</td>
</tr>
<tr>
<td>rosrnn</td>
<td>Start single nodes</td>
</tr>
<tr>
<td>rostopic</td>
<td>List topics, trace messages</td>
</tr>
<tr>
<td>rosmesg</td>
<td>Inspect messages</td>
</tr>
<tr>
<td>rosparam</td>
<td>Interface for the parameter server</td>
</tr>
<tr>
<td>robar</td>
<td>Record and playback messages</td>
</tr>
<tr>
<td>rxgraph</td>
<td>Graphical representation of running nodes and topics</td>
</tr>
</tbody>
</table>

Tbl. 2.1 shows only a fraction of the available ROS commands. Full documentation of all commands can be found on the ROS website [37].

**Advantages of ROS:**
- Source publicly available
- Fast growing user base
- Drivers and Tools available
- Lightweight

**Disadvantages of ROS:**
- Linux only
2.3 Simulator

Several different robot simulators have been considered for the use in this MR system. Some of them are listed below:

- Easy-Rob
- Gazebo
- Microsoft Robotics Studio
- Simbad
- Stage
- Webots

Based on previous surveys and comparisons [15][16][18] the choice could be narrowed down. The final decision was based on these points:

- Easy integration with ROS
- Runs on Linux
- Accessible user interface
- Completeness in terms of features (physics, visualization, “hardware”)
- Professional support

Microsoft Robotics Studio and Easy-Rob were quickly dismissed, as both run on Windows only, which does not work well with the Linux based ROS. Stage is a 2D only simulator, which would be fine for the first applications but would require a change later on.

The final choice was made between Gazebo [38] and Webots [39]. Gazebo is an open source simulator which is tightly connected to Player. But it is also fully integrated into ROS too [19]. Webots is a commercial solution. While this has some disadvantages like being closed source, it also has one big advantage over the other contender. Cyberbotics the company behind Webots offers professional support with fast response to support requests. The integration of Webots with ROS is also very easy to achieve. ROS nodes can be integrated into Webots controllers either via C++ or Python.

For this project Webots seems to offer a more complete solution. And it comes with an Integrated Development Environment (IDE) for modelling, programming and simulation, which makes it more accessible than the competing software.
2.3.1 Webots

Webots is a commercial solution for the simulation of mobile robots. Webots is developed and sold by Cyberbotics Ltd. and was co-developed by the Swiss Federal Institute of Technology in Lausanne.

Figure 2.3 shows the main interface of Webots. There are three main functionalities directly available through the GUI.

On the left side is the scene tree, that holds the structure and information about the environment and all the objects used in the system. Together with the 3D representation this can be used to edit the world directly. Here you can add objects to the scene, remove them or modify their properties. Webots comes with a library of sensors and actuators, that can be attached to the robots. The sensor library includes most of the common sensors that are used for robots, like [20]:

- Distance sensors & range finders
- Light sensors & touch sensors
- Global positioning sensor (GPS)
- Compass & inclinometers
- Cameras
- Radio and infra-red receivers
- Position sensors for servos & incremental wheel encoders
It also comes with a set of actuators, like:

- Differential and independent wheel motors
- Servos
- LEDs
- Radio and infra-red emitters
- Grippers

While most of the modeling can be done using the built-in editor, it is also possible to import models from an external modeling tool. Webots uses the VRML97 standard \[21\] for representing the scene. Using this standard it is possible to interchange 3D models with other software.

On the right side of the user interface is the code editor, which can be used to develop and program the robot’s behavior. Webots supports multiple programming languages like C/C++, Java, Python or Matlab \[40\] and can interface with third party software through TCP/IP.

One important thing to note is that Webots has very strict Application Programming Interfaces (APIs).

A regular robot controller has only access to the same kind of information, that a physical robot would have in the real world. It can only interact with it’s sensors and actuators. There exists no way to access other parts of the simulation. Also there is no access to the graphics stack. The supervisor is an extended robot controller. The supervisor has access to the world, can make changes like moving objects around and it can control the state of the simulation. But the supervisor has also only a limited set of APIs that it can use.

The only way to integrate visualization directly into the virtual world is to “abuse” the physics plugin API. This API is originally meant to extend the built-in physics with custom code. The physics plugin is the only component, that has access to the graphics engine of the simulator. At the end of each simulation step, the plugin is called. It then can draw directly into the scene using OpenGL commands. Chapter 3.2.6 shows how the information acquired from a laser scanner can be visualized in the simulator by using the physics plugin.

Proper usage of Webots on Ubuntu Linux requires some additional post-setup modifications. Details can be found in the appendix A.1.
Advantages of Webots:

- Complete development environment
- Includes library of sensors and actuators
- Supports multiple Platforms
- Easy integration with ROS
- Well known, proven product
- Professional support

Disadvantages of Webots:

- Licensing costs
- No source code available
- API restrictions

2.4 Sensors

In Mixed Reality sensors can belong to two very different groups. The first group is an integral part of the MR system itself. They act as the eyes and ears of the system and are used to gather information about the real world. The information of these sensors is used to create a link between the physical world and the simulated world. For example, based on this sensor data the position of a physical robot can be kept in sync with it’s virtual representation.

The second group of sensors is part of the robot. The information generated by these sensors normally goes directly to the control logic of the robot. With MR we can tap in and mirror or even redirect this data to the Mixed Reality system. The MR system can then feed this information into the simulation or use it for visualization. For example the range information of a distance scanner, can be overlayed onto a video feed from the environment. Or even directly onto the environment itself, using a projector.

Two types of sensors have been tested with the system so far. The first sensor which was incorporated into the system is a laser-scanner the second type are cameras that operate in the visible spectrum.
2.4.1 Laser Scanner

For this project the SICK LMS-200 laser scanner was used. The LMS-200 uses a RS-422 high speed serial connection to communicate with the host. Figure 2.4 shows an image of the LMS-200.

![Fig. 2.4: SICK LMS-200.](image)

Fig. 2.4: SICK LMS-200.

![Fig. 2.5: Field of view](image)

Fig. 2.5: Field of view [22].

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning angle</td>
<td>180°</td>
</tr>
<tr>
<td>Angular resolution</td>
<td>0.25°; 0.5°; 1°</td>
</tr>
<tr>
<td>Resolution / typical Measurement Accuracy</td>
<td>10mm±15mm</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>0 to +50 °C</td>
</tr>
<tr>
<td>Laser diode</td>
<td>Class 1, Infra-red (λ = 905nm)</td>
</tr>
<tr>
<td>Data transfer rate</td>
<td>RS-232: 9.6 / 19.2 kbd</td>
</tr>
<tr>
<td></td>
<td>RS-422: 9.6 / 19.2 / 38.4 / 500 kbd</td>
</tr>
<tr>
<td>Data format</td>
<td>1 start bit, 8 data bits, 1 stop bit, no parity (fixed)</td>
</tr>
<tr>
<td>Power consumption</td>
<td>Approx. 20 W (without load)</td>
</tr>
<tr>
<td>Weight</td>
<td>approx. 4.5 kg</td>
</tr>
<tr>
<td>Dimensions</td>
<td>155mm (wide) x 156mm (deep) x 210mm (high)</td>
</tr>
</tbody>
</table>

Tbl. 2.2: Features and specifications of the SICK LMS-200 [22]

The LMS-200 has a typical range of about 10 meters and a maximum field of view of 180 degrees (see Fig. 2.5). Tbl. 2.2 shows the technical details of the device. Because of the scanner’s size, weight and power requirements, the usage is limited to bigger robots or stationary operation. Therefore it was not possible to use this scanner with our smaller robots in this project. Nevertheless the scanner was integrated, tested and it’s range measurements could be visualized in the robot simulator (see chapter 3.2.2). Even though the scanner is not suitable for small robots, it still can serve as an external localization device.
There also exists a number of small and light-weight scanners like the ones produced by Hokuyo [41] which are very popular on smaller robots. The implemented procedures to gather and visualize distance information in the Mixed Reality system are independent from the hardware used. Every device that supports the LaserScan message (Listing 3.1) can be used as a drop-in replacement. ROS also comes with support for Hokuyo devices [23].

2.4.2 Camera

Cheap optical sensors in combination with increased processing power and advances in computer vision algorithms have made cameras a more versatile and cost effective alternative to specialized sensors. Many applications do not require high quality optics and can rely on simple and cheap digital image sensors. Therefore many robot designs incorporate cameras as means of information gathering.

Computer vision based on digital image sensors can be a versatile and cost effective way for object detection, tracking, measuring distances or navigation. For many robotic applications the information gathered from these optical sensors is the main input for decision making. A simple example of how an intelligent vehicle can make use of a camera, and the opportunities that MR gives here, can be seen in chapter 3.2.3.

For this project two different types of cameras have been tested. For usage in the tracking system, good quality images at a high frame rate are required. Hence two more advanced cameras have been considered:

![Sony SNC-RZ30](image1)

![Prosilica GC1350C](image2)

Fig. 2.6: Sony SNC-RZ30

Fig. 2.7: Prosilica GC1350C

The first one is a Sony SNC-RZ30 IP surveillance camera. It is a Pan-Tilt-Zoom camera with a resolution of 640x480 pixels at 30 frames per second. It also features up to 25x optical zoom. The camera delivered very good images, even in low light condition.
2. Building Blocks of a Mixed Reality System

<table>
<thead>
<tr>
<th></th>
<th>Sony SNC-RZ30</th>
<th>Prosilica GC1350C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor Type</td>
<td>Sony Super HAD CCD Type 1/6”</td>
<td>Sony ICX205 CCD Type 1/2”</td>
</tr>
<tr>
<td>Resolution</td>
<td>640 x 480</td>
<td>1360 x 1024</td>
</tr>
<tr>
<td>Frames per second</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>Interface</td>
<td>IEEE 802.3 100 Base-TX</td>
<td>IEEE 802.3 1000 Base-T</td>
</tr>
<tr>
<td>Protocol</td>
<td>HTTP/MJPEG</td>
<td>GigE Vision Standard 1.0</td>
</tr>
</tbody>
</table>

The second camera that was tested is the Prosilica GC1350C. The resolution of this camera is 1360x1024 pixels at 20 frames per second (fps). In comparison to the SNC-RZ30 this means that per image 4.5 times more pixels have to be transferred. Therefore it requires a gigabit ethernet connection to transfer high resolution color images at maximum frame rate. Another difference to the Sony camera is, that this camera does not have a fixed lens, but lets the user select a lens that is adequate for the intended purpose.

2.5 Robot

The complete Mixed Reality system should also incorporate a mobile robot, that can be used for experimentation and demonstration. Chapter 1.4 states the requirements for a mobile robot. The next two parts will first define the selection criteria for the optimal robot and then will take a look at the models which were actually considered for use in the demonstration system.

2.5.1 Selection Criteria

The initial specification of the project, also includes a car-like robot. The main requirement for this robot is that it should resemble a car as good as possible. One desired feature is that it uses a car-like propulsion and steering. Hence there are two ways to achieve this. Either find an existing robot kit that already resembles a car, or use a remote controlled car as body and outfit it with a computer and other hardware, like sensors, to transform it into an intelligent vehicle. Figure 2.8 shows an objective tree that incorporates the different requirements and attributes that should be considered in the decision process.

Based on these attributes, several robots, robot kits and remote controlled car models have been surveyed regarding their qualification for this project. But in the end the decision was made to use existing hardware for the demonstration system.
2.5.2 Actual Models

The previous section has shown the attributes, based on which an optimal robot should be chosen. But for the demonstration system it was decided to use existing hardware, instead of buying another robot. This decision has no big influence on the MR system itself, nevertheless it is important to note.

Three different robots from the university’s stock have been considered for use.

The first consideration was to use the *Alfred* robot. But because of its size and weight, the spatial requirements would have been too high. Alfred still got some usage as host for the laser-scanner, during the initial tests of that device.

The *Khepera III* robots were considered as well, primarily because of their small size and the already available virtual representation in the simulation software. For the final implementation of the demonstration system, they were outranked by the third available robot type.

The *PIE* robot is a custom robotics kit, that is used by students in the universities *Design of Embedded and Intelligent Systems* course. This robot features a small ARM
based controller board and can communicate with a base station via a 2.4 GHz RF link. Using some custom written software this robot can be remote controlled via ROS. The implementation of the robots logic can therefore be done on a regular Personal Computer (PC) and the final control commands are then transmitted to the robot via the teleop ROS node. More details can be found in chapter 3.2.4 and in chapter 4.3.1.

2.6 Tools

This chapter describes additional software modules that perform special tasks and are required to complete the Mixed Reality system.

2.6.1 Tracking System

The tracking system used in this work, is based on visual detection of special markers. The spiral detection was developed at the Intelligent Systems Laboratory [24] at Halmstad University. It uses captured images from a camera which is mounted above the area and gives a top-down view of the environment. The result is a 2D view of the environment, which is perfectly sufficient for the given task of tracking mobile robots and stationary obstacles.

The main reason to use a marker based tracking system is, that it allows for an easier setup of the system. There is almost no initial configuration and calibration required. The selected method has the further advantage, that it is very robust and insensitive against changes in brightness, contrast or color.

The markers and algorithms used in this system are based on spiral patterns [25][26].
For simplicity only one spiral marker is used per object, as the direction of the objects is currently not of importance. In case the heading of objects becomes important, a similar approach as described by Karlson and Bigun [24] where multiple markers are used per object, can be added to the system without much change. Table 2.4 shows the eight different spiral markers that can be used to locate and identify objects.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Fig. 2.12 shows the output of the tracking system. This image was taken during the testing of the demonstration system which is described in chapter 4. It shows the detected markers, the regions of interest around each marker and the detected type of spiral. The outer four markers, labeled as “4” are used as boundary markers. When enough boundary markers have been detected, the bound area is visualized by blue lines. The label in the middle of the field (“5”) marks the location of the robot.

![Fig. 2.12: Visual output of the tracking software.](image-url)
To transmit the data from the Matlab host to the MR system, a custom User Datagram Protocol (UDP) communication interface has been implemented. Listing 3.4 explains the text-based message format.

### 2.6.2 Visualization of Virtual Objects

Visualization in a Mixed Reality system can be done in several ways. Collet [27, p. 26] describes two distinct categories of AR visualization: “Immersive AR” and “Desktop AR”. The category of Immersive AR consists of

- Video See-Through
- Optical See-Through
- and Projected Systems.

Video See-Through and Optical See-Through systems require the user to wear special equipment. Normally a head-mounted display, that allows to infuse the virtual data into the real world, is used for this task. Projected Systems however display the virtual information directly in the environment. Desktop AR uses some “external” view on the scene. Normally the AR visualization is happening on a separate PC.

For this system, the decision was made to use a projected visualization. This has some advantages, but also some disadvantages. Advantages:

- Multiple spectators can view the scene at once
- No need to wear special equipment
- Direct integration into the real world

The biggest advantage of visualizing data this way, is that the virtual information is directly integrated into the real environment. There is no need for users or spectators to wear special equipment and the mix of reality and virtuality happens exactly at the point of interest. There is no need to view the scene on an external device.

Disadvantages:

- Projector needed
- User interaction can interfere with the projection
- Limited space for projection
- *Flat* visualization

Projector based visualization also has some drawbacks. First of all, you need a projector mounted in a suitable location. Users that are interacting with the system might interfere with the projection. Also the size of the environment is limited due to the range of the projector. In See-Through systems or in a Desktop based solution, the addition of
three dimensional objects is much more sophisticated, as the users point of view can be incorporated into the visualization of the objects. In a projector based system this information is (normally) not available and therefore it is not possible to create a three dimensional representation of objects.

The visualization module uses a simple Qt [42] based application to render the positions of the virtual object. It uses four boundary markers to specify the edges of the area. These boundary markers can be moved around so that they can match up with the real markers in the environment. Once this is done, the simulated coordinates can be transformed and the objects can be visualized at the correct position. Details about the coordinate transformation can be found in chapter 2.6.3.

The visualization system uses ROS to retrieve the required data that it needs for the graphical representation. Therefore it subscribes to the Map topic. When the map data (Listing 3.6) comes in, it extracts the worlds bounding information to update the transformation matrix and then uses the remaining position information for the presentation of the objects.

![Fig. 2.13: Principle of the visualization.](image)

Fig. 2.13 shows an example, how the visualization can look like. In the corners you can see the boundary markers that are used for the coordinate tranformation. The two circles inside stand for the virtual objects. Currently the visualization only supports simple shapes that represent the position of the simulated objects. But for future applications it could get extended, so that different and more complex types of data can be presented. For example it could be used to visualize the sensor “view” of the robot directly onto the environment.

Fig. 2.14 shows a picture taken of the implemented projection. The four spirals on the edges are used for the alignment and perspective correction. The spiral in the center
represents a physical object, which is tracked by the system. The two red dots are the visualization of two virtual robots, which are moving around in this area.

![Image of the real projection.](image)

**2.6.3 Coordinate Transformation**

Coordinate transformation is required because of two reasons.

- Perfect alignment of the camera (and projector) is very hard to achieve. There exists always some translation and rotation that creates a perspective error.
- The optical tracking system and the visualization module use pixels as units of measurement. The MR system internally uses meters to describe distances.

Fig. 2.14 shows an example, where camera and projector are not perfectly positioned. As a result the area has an perspective distortion and does not resemble a perfect rectangle anymore. The coordinates gained from the tracking system and the coordinates used by the visualization, therefore have to be corrected to compensate the error. Another example can be seen in Fig. 2.12.

For this two independent transformations are required. One from the image plane of the camera to the world plane of the simulation. And another one from the world plane to the projector’s image plane. Theses transformations are done by utilizing 2D homography.

Homography is a transformation of coordinates between two coordinate systems. In case that both coordinate systems have only two dimensions it is called 2D homography. Fig. 2.15 shows an example.
The figure shows two planes. One represents the camera’s image plane and the other is the world plane. The goal of the homography transformation is to eliminate perspective errors. In a homography transformation, a point in one plane corresponds to only one point in the other plane. The operation is invertible. To calculate the projective transformation a so called Homography Matrix \((H)\) is required.

\[
\begin{bmatrix}
    x'_i \\
    y'_i \\
    w'_i
\end{bmatrix} = \begin{bmatrix}
    h_{11} & h_{12} & h_{13} \\
    h_{21} & h_{22} & h_{23} \\
    h_{31} & h_{32} & h_{33}
\end{bmatrix} \begin{bmatrix}
    x_i \\
    y_i \\
    w_i
\end{bmatrix}
\]  

(2.1)

The homography matrix can be obtained by using the Direct Linear Transformation (DLT) algorithm. A detailed description of the DLT can be found in Dal Pont et al. [28]. Once the matrix has been found, the coordinates can be transformed with a simple matrix multiplication.

In case of the positioning system, four special markers are used to retrieve the location of the area’s edges. Together with the known locations of the simulation’s edges, the homography matrix can be constructed. The resulting matrix can then be used to transform the coordinates of the detected objects into the meter based coordinate system.

Likewise the same procedure is used in the visualization module. After start-up the operator can adjust projected markers so that they match the real ones in the environment. Together with the known positions of these points in the simulation, a matrix can be constructed and used to transform coordinates from meters to pixels.

Using this technique, the setup and configuration of the system can take place much more quickly and precise. The camera and projector don’t need to be aligned perfectly and still the resulting measurements and projections are precise enough for the use in the Mixed Reality system.
The implementation of the coordinate transformation uses code from Dal Pont et al. [28], which in turn uses functions from the \textit{ALGLIB} [43] (\textit{Open Source Edition}). For this project the code has been adapted to work with version 3.5.0 of the \textit{ALGLIB}.

\section*{2.7 Summary}

A Mixed Reality system can contain numerous different components. In general we can divide the components into three groups. The first group is handling the real world – detecting, recognizing and interacting with physical objects. The second group is covering the virtual side – simulating the digital world and it’s “citizens”. The third group of components brings reality and virtuality together. The components shown in this chapter are only a small selection. Depending on the application of Mixed Reality, other components might be required too.
3

Implementation

Chapter 2 described many different components. Not all of these components were already available - some of them had to be implemented. This chapter will explain the implementation of custom software modules and how different parts of the system communicate with each other and exchange data.

3.1 Used Hardware and Software

Tables 3.1 and 3.2 show the most important information about the hardware and software, that was used to create and test the components.

The host machine, that was used for developing but also for running the demo system (see chapter 4), is a Dell T3500 model. It is equipped with a dual core Intel Xeon processor, 6 GB RAM and a Nvidia graphics card. This system, especially the graphics card, was chosen because of it’s compatibility with Linux. The machine runs the 32-bit version of Ubuntu 12.04 LTS, a Linux based operating system. The LTS (Long Term Support) edition was chosen because of the longer support and the stricter update policy which should prevent possible errors due to package updates [45]. Ubuntu was selected as operating system, because the middleware of choice (see chapter 2.2) is developed and tested for this Linux distribution.

The Matlab [40] based tracking system, runs on a separate machine. The decision to use Windows was made because of licensing constraints. Also because the image analysis is a very computational intensive task, it makes sense to outsource it to a dedicated machine. This prevents possible errors due to delays or other interferences caused by too high CPU load.
### Tbl. 3.1: Specifications of the Linux host

<table>
<thead>
<tr>
<th>Hardware</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Dell Precision T3500</td>
</tr>
<tr>
<td>Processor</td>
<td>Dual Core Intel Xeon W3503 2.40 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>6 GB DDR3 SDRAM</td>
</tr>
<tr>
<td>Graphics</td>
<td>Nvidia Quadro 600</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating System</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Ubuntu 12.04 LTS (x86)</td>
</tr>
<tr>
<td>Kernel</td>
<td>Linux 3.2.0-23.36 (Ubuntu) based on the 3.2.14 upstream Kernel</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ROS</td>
<td>ROS Electric Emys released August 30, 2011 [37]</td>
</tr>
<tr>
<td>Webots</td>
<td>Webots Pro 6.4.4 [39]</td>
</tr>
<tr>
<td>IDE</td>
<td>KDevelop 4.3.1 [44]</td>
</tr>
<tr>
<td>Compiler</td>
<td>GCC 4.6.3</td>
</tr>
</tbody>
</table>

### Tbl. 3.2: Specifications of the Windows (Matlab) host

<table>
<thead>
<tr>
<th>Hardware</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>HP EliteBook 8460p</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Core i5-2520 @ 2.5 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>4 GB DDR3 SDRAM</td>
</tr>
<tr>
<td>Graphics</td>
<td>AMD Radeon HD 6470M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating System</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Windows 7 Professional, SP1, 64-bit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>Matlab 7.11.0 (R2010b) [40]</td>
</tr>
</tbody>
</table>
3.2 Implementation and Interaction

This part deals with the interaction and communication between different modules and subsystems. It explains the kind of connections, protocols and messages that are used. Furthermore it explains certain key details of the implementation of some of the custom components. First we take a look at the different types of modules in the system and then the message flows of selected “tasks” are inspected.

3.2.1 Overview

Integration of the components is done using ROS. Using the ROS infrastructure, each component can be encapsulated in its own module and then communicate with the other modules using messages [29]. This allows for a loose coupling of modules, where one module does not need to know the others. The only thing that must be known is the format of the exchanged messages.

![Diagram of system's different modules and components.](image)

Fig. 3.1: Overview of the system’s different modules and components.

Fig. 3.1 shows an overview of all available components and subsystems. The components have been grouped together, according to their “role” in the system. On the right hand side, we have the parts that belong to the physical world. This includes sensors, robots and other hardware that interacts with the real environment.
3. Implementation

Real world components:

- IP Camera
- Projector
- Laser Scanner
- Webcam
- Input Devices
- Robots

Next are “external” tools, that are not directly integrated into the ROS system, but offer services that the MR system can use. Currently there is only one such tool present:

- Positioning System

In the central part of Fig. 3.1 are the different ROS nodes located. Most of these nodes are responsible for the integration of external hard- and software components. But there are also nodes, that contain software modules for controlling the robots.

- ROS Core
- *position* node
- *visualize* node
- *sick* node
- *camera* node
- *input* node
- *teleop* node
- *rosbot_ctrl* node

Finally on the top of Fig. 3.1 we have the simulator. The simulator can be subdivided into three distinct modules.

- Physics Plugin
- Robot Controller
- Supervisor

Details on the different modules can be found in chapter 2.3.
3.2.2 Visualization of Sensor Data

One of the benefits of using MR technologies is the ability to visualize a robot’s sensor data.

The current version of the Webots simulator has only limited options to visualize sensor data. There is currently no way to display external sensor data directly. The only way to achieve this, is to implement this functionality on your own. Due to restrictions in the APIs that the simulator offers, the only way to access the graphics stack is by creating a physics plugin [30].

The physics plugin is loaded automatically when the simulation starts. As there is no possibility to pass parameters to the plugin, it has to be either adapted specifically to the simulation or it has to get it’s configuration from an external source, like a configuration file on the hard disk. In our case the plugin has been tailored to the simulation it is used with.

When the plugin is initialized, it retrieves a handle of the virtual laser scanner object, used in the simulation (see appendix A.2). Every time the plugin’s main callback function is invoked, it will use this handle to retrieve the coordinates of the virtual laser scanner. Once it has retrieved the coordinates it will draw the area covered by the laser scanner.

The plugin can either paint only the outline of the area or it can also draw the individual laser rays. It is important to know, that the resulting scene is also fed to the virtual cameras. Therefore care must be taken, that the added visualization does not interfere with image analysis algorithms used in other parts of the simulation.

Fig. 3.2 shows the message flow for the visualization of real world data, in this case coming from a laser range scanner. Fig. 3.3 shows how simulated sensor data can be visualized.

![Fig. 3.2: Message flow for visualization of real sensor data.](image)

The `sick` node [31] is responsible for the communication with the laser scanner. In the case of the SICK LMS-200, communication takes place via a RS-422 serial interface. The details of the used protocol are described in the LMS-200 manual [32]. The node receives the laser’s measurements and translates them into a ROS compatible format. The `sick` node then publishes the data using a `LaserScan` message (Listing 3.1). This message
is received by the ROS node in the *Physics Plugin* and the contained data is used for visualization.

![Diagram](image)

Fig. 3.3: Message flow for visualization of simulated sensor data.

When visualizing the information from the virtual laser scanner, the *Robot Controller* “exports” its measurements to ROS using the *LaserScan* message. As with the real sensor information, the *Physics Plugin* will receive and process this data.

![Image](image)

Fig. 3.4: Overlay of sensor data onto the virtual scene.

Fig. 3.4 shows the resulting overlay of sensor data onto the simulation. It shows how the blue laser rays, which are drawn based on the received range data from the sensor, follow the contours of the environment.
Listing 3.1: LaserScan message

```c
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior
# (e.g. a sonar array), please find or create a different message,
# since applications will make fairly laser-specific assumptions
# about this data

Header header # timestamp in the header is the
# acquisition time of the first ray
# in the scan.
#
# in frame frame_id, angles are measured
# around the positive Z axis
# (counterclockwise, if Z is up)
# with zero angle being forward along the
# x axis

float32 angle_min # start angle of the scan [rad]
float32 angle_max # end angle of the scan [rad]
float32 angle_increment # angular dist. btw. measurements [rad]

float32 time_increment # time between measurements [seconds]
float32 scan_time # time between scans [seconds]

float32 range_min # minimum range value [m]
float32 range_max # maximum range value [m]

float32[] ranges # range data [m]
float32[] intensities # intensity data [device-specific units].
```

Listing 3.1 shows the format of the standard ROS LaserScan message.
(Some comments have been stripped to fit on this page.)
### 3.2.3 Mix Real World Camera Data with Simulation

With Mixed Reality, real sensors can be integrated and used in the simulation. In this example the live stream from a real camera is integrated into the control logic of a virtual self-driving vehicle. The simulated vehicle has the ability to automatically adjust its steering to follow the road markings. It can now either use the virtual camera’s images, or it can use the images from the real camera. When the physical camera is used, the car can be steered by using a simple piece of paper with “road markings” on it (see Fig. 3.5).

![Real-world image data is being fed into the lane-keeping controller of the simulated robot.](image)

**Fig. 3.5:** Real-world image data is being fed into the lane-keeping controller of the simulated robot.

The `ipcam` node receives the JPEG compressed image frames from the camera, decodes¹ them into raw images and then publishes the images using the `Image` message (Listing 3.2). The `Robot Controller` receives the `Image` messages and processes the contained image data.

---

¹ The implementation uses the freely available *Mini Jpeg Decoder* written by Scott Graham [46].
3.2.4 Teleoperation

Teleoperation allows the user to remotely operate the robot. This can be used to control the physical robot as well as the virtual one. Fig. 3.7 shows the message flow for the control of a physical robot.

Using ROS, teleoperation can be implemented using a simple setup of two nodes. The input node is responsible for acquiring and interpreting the control information, like commands from the keyboard or a joystick. This data is then transferred to the teleop node which is responsible for steering the (physical) robot. For the data transfer the Joy message (Listing 3.3) is used.

---

**Listing 3.2: Image message**

```plaintext
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image
#
Header header
  # Header timestamp should be acquisition time of img
  # Header frame_id should be optical frame of camera
  # origin of frame should be optical center of camera
  # +x should point to the right in the image
  # +y should point down in the image
  # +z should point into to plane of the image
  uint32 height  # image height, that is, number of rows
  uint32 width   # image width, that is, number of columns
  string encoding  # Encoding of pixels
  uint8 is_bigendian  # is this data bigendian?
  uint32 step   # Full row length in bytes
  uint8[] data  # actual matrix data, size is (step * rows)
```

Listing 3.2 shows the format of the standard ROS Image message. (Some comments have been stripped to fit on this page.)
3. Implementation

Listing 3.3: Joy message

<table>
<thead>
<tr>
<th>Header</th>
<th># Reports the state of a joystick's axes and buttons.</th>
</tr>
</thead>
<tbody>
<tr>
<td>float32[] axes</td>
<td># the axes measurements from a joystick</td>
</tr>
<tr>
<td>int32[] buttons</td>
<td># the buttons measurements from a joystick</td>
</tr>
</tbody>
</table>

Listing 3.3 shows the format of the standard ROS Joy message.

An example on how teleoperation can be used, is shown in chapter 4 where a physical robot is controlled using a Wii Remote [47].

### 3.2.5 Tracking of Physical Objects

To keep the simulation in sync with the real world, the physical objects need to be localized and their positions in the simulation have to be updated accordingly.

![Message flow of the object tracking.](image)

The real world environment is observed using a camera. The captured images are then transferred to the tracking system via IP. Depending on the type and model of the camera, different encodings and transport methods are used. The tested camera SNC-RZ30 for example uses Hypertext Transfer Protocol (HTTP) with Motion JPEG (MJPEG) transmissions.

The tracking system then scans the received images for markers (see chapter 2.6.1). Once a marker is detected it gets classified and stored. When the image processing has finished the stored information is sent to the ROS node using a custom UDP based protocol. Listing 3.4 shows the simple text based message format.
The ROS node then extracts the boundary markers from the message and uses them to update the mapping of pixel coordinates to simulation coordinates. For this purpose a homography matrix is created, using the detected boundary markers and the boundary points of the simulated environment (see chapter 2.6.3). Using this matrix, the remaining coordinates are transformed and packed into a ROS Position message. This message then gets published to the other ROS nodes.

Listing 3.5: Position message

```plaintext
# Reports the id and position of detected objects.
uint16 numPts  # Number of entries in this message
int8[] id      # ID of each entry
float32[] x    # X coordinate of each entry
float32[] y    # Y coordinate of each entry
```

Listing 3.5 shows the format of the custom ROS Position message.

The Supervisor receives the Position message and uses the obtained coordinates to update the positions of the tracked objects in the simulation.

The tracking system is implemented in Matlab. The implementation is based on code provided by Josef Bigun [26]. It uses the Matlab Image Acquisition Toolbox to capture an image stream from the IP camera. Usually a special driver has to be installed. Appendix A.3 lists the required drivers and installation steps for the use with the Matlab Image Acquisition Toolbox.

The number of objects that can be tracked is theoretically unlimited, but has a high impact on the achieved frame rate. Therefore the number of objects should be limited. On the Matlab system (specifications can be found in chapter 3.1) up to six markers could be tracked at 15 frames per second. This number is sufficient for the tracking of the mobile robot used in the demo system.

Because the analysis of a full image would take too much time, and as a result the frame rate would drop below a useable value, only certain portions of the input image are processed. To
reduce the amount of data to scan, the code creates Regions of Interest (ROIs) around the interesting parts. On the first run, the whole image is scanned. Every found spiral marker, is then surrounded by a ROI. In consecutive scans only these ROIs are scanned for markers. When the location of a marker changes, the ROI will be updated too. Full rescans of the whole image occur cyclically after a certain amount of processed frames or when the number of detected markers in the current frame is lower than it was in the frame before. A full scan is also triggered when the number of detected spirals exceeds a certain limit. In some cases, interference can lead to false positives, which in turn would increase the processing time per frame. To prevent this the number of detected markers is checked and if it exceeds a predefined maximum, the current ROIs are discarded and a scan of the whole image is induced. Fig. 3.9 shows the main flow of the tracking module.

![Flowchart](image)

**Fig. 3.9: Control flow of the tracking software**

### 3.2.6 Visualization of Virtual Objects

The *Supervisor* keeps track of all objects. It periodically sends out a message (Listing 3.6) that among other things also contains the locations of these objects. The *visualization* node receives these messages and updates it’s graphical representation. Fig. 3.10 shows the message flow.

![Message flow](image)

1. ROS message: Map
2. Device dependent connection (VGA, HDMI, ...)

**Fig. 3.10: Message flow of the visualization subsystem**
3. Implementation

### Listing 3.6: Map message

```
# The map message is actually a std_msgs/String message
# The data string is formatted as follows:
# bounds <potential> <X>/<Y>/0
# object <potential> <X>/<Y>/<A> <TYPE> <ID>
#
# Where:
# <potential> represents the "potential force" of the object
# <X> and <Y> represent the x and y coordinates of the object
# <A> represents the angle (heading) of the object on a 2D plane
# <TYPE> represents the type of the object (0..Robot,1..Obstacle)
# <ID> represents the identification number of the object
#
# Fields are separated by tab stops, entries by line breaks
# There is only one bounds entry, but there can be multiple
# object entries
string data
```

When the visualization node receives the *Map* message, it first extracts the boundary information. Based on this boundary information the internal transformation matrix (see chapter 2.6.3) is updated. After that the information about the objects is processed. For each object the coordinates are extracted and transformed. Then the objects are visualized. Based on the type and id of the object, different colors and shapes can be used.

### 3.2.7 Robot Control

The use of the MR system allows us to move the robot logic freely between a simulated robot and a real robot. To achieve this, a simple Webots *Robot Controller* has been coupled with a ROS node that contains the robot’s logic. The *Robot Controller* in the simulation collects the virtual sensor data and publishes it to the ROS “cloud”. Also it subscribes to *Twist* messages (Listing 3.7), which contain information to drive the motors. The ROS node containing the robot’s logic, receives the sensor information and uses it to make decisions about it’s next action. It then sends the control information back to the *Robot Controller* who will use it to control it’s actuators. Fig. 3.11 shows the message flows.

In it’s simplest form the ROS robot controller just reacts on the given sensor input, similar to a Braitenberg vehicle. In this case every sensor is given a different weight that determines, how much it influences the speed of the wheels. After incorporating all sensors the speed of each wheel has been determined and is handed over to the motor control module. In case of a simulated robot, the control node will sent the speed information back to the simulator using the ROS *Twist* message. This message contains the forward speed and the angular velocity.
3. Implementation

Fig. 3.11: Message flow of the robot control

Listing 3.7: Twist message

```python
# This expresses velocity in free space broken into it’s
# linear and angular parts.
Vector3 linear
Vector3 angular
```

For more advanced control mechanisms the controller can also subscribe to the *Map* messages. They contain information about the environment, like the boundaries of the world and positions of objects. Based on this information, path planning algorithms can be implemented. For the use in the demo system (see chapter 4) the control logic uses the map information to track the physical robot. Fig. 3.12 shows the message flow between the simulation’s *Supervisor* and the control node.

Fig. 3.12: Message flow of the robot control with additional map information

### 3.3 Summary

This chapter has shown how the various different components can interact with each other. Using a ROS based middleware, the communication between modules can be realized using the publish-subscribe pattern. This allows for a clear separation between different modules and therefore provides a basis for modularity and changeability.
This chapter shows the implementation of an actual, working Mixed Reality system. It combines the different subsystems and components that have been described in chapters 2 and 3. The goal is to use these components to build a system that can serve a specific task.

Two interactive scenarios have been realized. In both a player can take control over the physical robot. Using a Wii Controller [34] (Fig. 4.3) the user can teleoperate the real robot and drive around. In addition to the robot, there are also some physical obstacles that can be pushed around. In the first scenario, there are virtual robots that will try to “catch” the real robot while avoiding the real obstacles. In the second scenario, there is a virtual ball, that can be “kicked” around using the real robot (Fig. 4.9).
4. Demo System

4.1 Overview of the Demo System

Figure 4.1 shows a draft of the system. Fig. 4.4 will show a more detailed view of the components and interactions in the Mixed Reality part of the demonstration system.

This system will utilize the following components:

- Tracking and representation of physical objects in the simulation (chapters 2.6.1 & 3.2.5)
- Interaction between virtual and physical objects (chapter 3.2.7)
- Teleoperation of a real robot (chapter 3.2.4)
- A simple autonomous virtual robot (chapter 3.2.7)
- Visualization of virtual data onto physical environment (chapters 2.6.2 & 3.2.6)

4.2 Hardware and Software

- The computers running the Mixed Reality system and the Tracking System are identical to the ones used for development. See chapter 3.1 for details.
- To capture images from the environment, the Prosilica GC1350C is used (Fig. 4.2). See chapter 2.4.2 for details.
- Visualization is realized via an Acer H5360-BD projector (Fig. 4.2).
- To control the robot via teleoperation, a Nintendo Wiimote (Fig. 4.3) is used as input device.

Fig. 4.2: Projector and camera mounted to the ceiling.  
Fig. 4.3: Wii Controller.
4.3 Integration

Fig. 4.4 shows all the different components that are required and the message flows between them. The single flows have already been discussed in chapter 3.

The Mixed Reality system is split into two separate subsystems. One is responsible for controlling the physical robot. The other handles the tracking of physical objects, the interaction of the virtual robot with the physical objects and the visualization of the virtual entities.

As can be seen in Fig. 4.4, the supervisor is the central part of this MR setup. It receives the locations of the physical objects from the tracking system. Then it applies the new information to the simulation. After that it sends out the current state of the virtual world. The virtual robot controller receives this information and uses it to adapt its motion planning to the new situation. It will then command the Robot Controller to move according to the new plan.

The visualization module also receives the new state and uses the information to update its output. The new situation will then be visualized using the projector.
4. Demo System

4.3.1 Robot

In chapter 3.2.4 the generic flow of messages for teleoperation is outlined. For the demonstrator the PIE robot is used. Fig. 4.5 shows the connection between the robot and the Mixed Reality system. The robot has a wireless connection to a base station, which is connected to the MR host using a serial RS-232 connection.

The base station and the robot itself are based on an Olimex SAM7-P256 [48] board. The wireless communication is realized, using a Nordic MOD-NRF24Lx [49] transceiver module.

To transfer data between the host and the robot, a special communications protocol has been implemented. The protocol used here is a simplified version of a communications protocol, that I had developed for another project. Therefore only the relevant parts are discussed here.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIR</td>
<td>Direction of the message</td>
<td>0 ... Base station to PIE 1 ... PIE to Base station</td>
</tr>
<tr>
<td>CID</td>
<td>Connection ID</td>
<td>0x00 - 0x7F</td>
</tr>
<tr>
<td>TYPE</td>
<td>Type of the message</td>
<td>00 ... Control 01 ... Data 10 ... Debug</td>
</tr>
<tr>
<td>SEQ</td>
<td>Sequence number</td>
<td>0 - 64</td>
</tr>
<tr>
<td>CRC</td>
<td>16 bit CRC (Polynomial: 0xC86C)</td>
<td></td>
</tr>
</tbody>
</table>

Listing 4.1 and Tbl. 4.1 show the general message format.
Listing 4.2: PIE message for robot steering

0 8 24 40
-------------------------------
| 82 | v | u |
-------------------------------

Tbl. 4.2: VU Message used to steer the robot.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Speed</td>
<td>Speed encoded as Q3.8</td>
</tr>
<tr>
<td>u</td>
<td>Angular velocity</td>
<td>Angle (0-360°) encoded as short integer</td>
</tr>
</tbody>
</table>

Listing 4.2 shows the message used for driving the robot.

To transfer the messages over the serial link, a framing mechanism is used to separated individual messages in the data stream.

Tbl. 4.3: Control characters used in the framing.

<table>
<thead>
<tr>
<th>CHR</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STX</td>
<td>0x55</td>
<td>Marks the start of a new frame</td>
</tr>
<tr>
<td>ETX</td>
<td>0xAA</td>
<td>Marks the end of a frame</td>
</tr>
<tr>
<td>DLE</td>
<td>0x66</td>
<td>Marks the occurrence of a control character in the data stream</td>
</tr>
<tr>
<td>ESC</td>
<td>0x33</td>
<td>Is used to transform a control character to a non-control character</td>
</tr>
</tbody>
</table>

Example:

Data to transmit.     Resulting frame.
\[ S \ T \ U \ V \]     \[ STX \ S \ T \ DLE \ U + ESC \ V \ ETX \]
\[ 53 \ 54 \ 55 \ 56 \]     \[ 55 \ 53 \ 54 \ 66 \ 88 \ 56 \ AA \]
4.4 Example of Interaction

Figures 4.6 to 4.8 show the interaction between the physical robot and a virtual ball.

![Fig. 4.6: Robot approaches the ball.](image)

In Fig. 4.6 the robot approaches the ball. It’s position is tracked and updated in the simulated environment.

![Fig. 4.7: Robot “kicks” the ball.](image)

Fig. 4.7 shows how the robot “kicks” the ball, as the virtual objects collide in the simulation.

![Fig. 4.8: Ball rolls away.](image)

In Fig. 4.8 the virtual ball is rolling away and it’s movement is projected onto the real environment.
4.5 Summary

The demonstration system has been implemented successfully.

Fig. 4.9 shows an image of the demonstration system. In the foreground you can see the robot. Left of it is the projection of a virtual object. The screen in the background shows the same scene visualized by the simulation software.

Fig. 4.9: Physical robot and virtual object.
5

Conclusion

5.1 Results

Here we discuss the outcome of the project. We compare the initial goals against the results of the implementation and the usability in and of the demonstration system.

Six goals were defined in chapter 1.4. Tbl. 5.1 shows the outcome of these goals. Most of the goals were completely fulfilled. Goal number two was only partially completed, because the robot in the demo system is not controlled by the simulator but by a human using the teleoperation equipment. This also leads to a failure of goal number six, as there is no injection of sensor data into the control logic of the robot. Most of the required sub-tasks, like tracking and integration of the physical robot into the simulation or exporting sensor data from the simulator is done. But there has not been a complete integration of the robot itself.
### Tbl. 5.1: Assessment of the project goals.

<table>
<thead>
<tr>
<th>#</th>
<th>Goal</th>
<th>Result</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A mobile robot simulator</td>
<td>The core components including a robot simulator have been successfully selected and integrated into a working Mixed Reality system.</td>
<td>⊕</td>
</tr>
<tr>
<td>2</td>
<td>A physical mobile robot coupled to the simulator</td>
<td>The original plan was to get a car like robot. This task was postponed. Instead one of the available robots has been used for the demonstration system. In the current setup this robot is not directly coupled to the simulator. Instead it is controlled via teleoperation which is separated from the simulation. The robot could be integrated more tightly, for example the readings from it’s distance sensors could be visualized, or it could be fed with virtual sensor data.</td>
<td>⊙</td>
</tr>
<tr>
<td>3</td>
<td>A simple yet effective teleoperation station</td>
<td>A simple way to remote control the robot has been implemented.</td>
<td>⊕</td>
</tr>
<tr>
<td>4</td>
<td>Extensive documentation to build on these foundations</td>
<td>This document together with the source documentation.</td>
<td>⊕</td>
</tr>
<tr>
<td>5</td>
<td>Visualization of real sensor data</td>
<td>Visualization of real sensor data is possible.</td>
<td>⊕</td>
</tr>
<tr>
<td>6</td>
<td>Injection of simulated sensor data into physical robot</td>
<td>In this configuration, simulated sensor data cannot be injected into the robot’s control logic. Most of the required parts have been implemented but a complete setup has not been created.</td>
<td>⊖</td>
</tr>
</tbody>
</table>

### Tbl. 5.2: Rating symbols for project assessment.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊕</td>
<td>⊙</td>
<td>⊖</td>
</tr>
<tr>
<td>goal fulfilled</td>
<td>goal partially fulfilled</td>
<td>goal not fulfilled</td>
</tr>
</tbody>
</table>
5.2 Discussion

As can be seen from the results (chapter 5.1) most of the initial goals have been fully achieved. But some elements are missing. Right now the demonstration system does not make use of the robot’s on-board sensors. Also there are no virtual sensors, which could influence the real robot. So the very first next steps should include to add these missing elements.

The demonstration system proved some of the difficulties with projection based AR visualization, that where described in chapter 2.6.2. User interaction can interfere with the representation of the objects. In combination with camera based tracking this becomes even worse, because the user might also be blocking the camera’s view of the scene. In this case the system cannot track the objects anymore and loses the synchronization between the real world and the simulated one. Depending on the actual usage scenario, the visual problems can be solved by either adjusting the projectors location and direction to avoid the interference or by using back-projection if possible. The same options might apply to the camera based tracking but a better approach would be to add other types of position detection to the system and then fuse the different measurements to get a better result.

Incorporating the different build systems, dependencies and environment variables of ROS, Webots and other components can be tricky at times. From my personal experience, it is the easiest way to adopt the CMake [50] based build system of ROS and incorporate the dependencies of other software packages in here. Webots comes with an IDE for developing code, but from my point of view for more complex projects, it is better to switch to an external software development environment. During this project I tried two different IDEs, KDevelop [44] and QtCreator [51]. Both environments can handle the ROS CMake projects directly.

5.3 Outlook

We now have a functioning Mixed Reality prototype, which can be used for demonstrating the power of this technique and to build on it for new projects and better education. But there is always room for improvement:

The original plan to have a full loop in the Mixed Reality system - meaning that we track the robot and control the robot - was put back in favor of a more interactive demonstrator. Therefore the next step should be to modify the current demonstration system to implement this full simulation controlled behavior.

The approach for visualization that was used in this project is very basic, compared to the advanced rendering options of (for example) the simulator. For future projects it would be nice to either integrate more real world visuals, like a camera stream directly into the simulator’s graphical representation or to incorporate other means of presentation. There already exist some suitable frameworks, like ARDev [27], that could be integrated into the MR system.
Apart from the more ambitious goals of research and education, this could also be used as a nice demonstration for exhibitions. Based on the ball scenario, an air-hockey like “game” could be implemented, where a physical robot and a virtual robot play against each other. The two robots could be controlled either by an artificial intelligence or by a human player.
Acronyms

API . . . . . . . . Application Programming Interface
AR . . . . . . . . Augmented Reality
AV . . . . . . . . Augmented Virtuality
DLT . . . . . . . . Direct Linear Transformation
fps . . . . . . . . frames per second
HTTP . . . . . . . Hypertext Transfer Protocol
IDE . . . . . . . . Integrated Development Environment
IP . . . . . . . . Internet Protocol
MJPEG . . . . . . Motion JPEG
MR . . . . . . . . Mixed Reality
PC . . . . . . . . Personal Computer
PTZ . . . . . . . . Pan-Tilt-Zoom
ROI . . . . . . . . Region of Interest
ROS . . . . . . . . Robot Operating System
UDP . . . . . . . . User Datagram Protocol
Bibliography


Product References


[37] ROS: *ROS*. http://www.ros.org. Online; Date 2012/05/12.

[38] Gazebo: *Gazebo*. http://gazebosim.org/wiki/Gazebo. Online; Date 2012/05/12.


A.1 Webots Installation Information

The original rules file, available through the Cyberbotics website is not working on Ubuntu 12.04 anymore.

```
Listing 1: 99-matrix.rules

# rules to set the protection for matrix dongles
SUBSYSTEM=="usb", ATTRS{idVendor}=="0e50", ATTRS{idProduct}=="000[1-9]", PROGRAM="/etc/udev/scripts/matrix-rule-handler"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0e50", ATTRS{idProduct}=="000[1-9]", MODE="0666"
```

Listing 1 shows the new rules file. This should give every user account access to the dongle and allow for execution of the Webots software without root privileges.

Additionally the following lines should be added to `/etc/profile`

```
export USB_DEVFS_PATH=/proc/bus/usb
export WEBOTS_HOME=/usr/local/webots
```

A.2 Bounding Box for Laser Scanner

For the visualization of the sensor data, the location of the laser scanner in the virtual environment must be known. To retrieve this information, the prototype model must be modified and a bounding box must be added.

A.3 Video Input Device Driver for IP-Cameras

The Image Acquisition Toolbox (IAT) allows Matlab to access DirectShow Video Input devices. This enables Matlab to capture video frames at a much higher frame rate, which is important when tracking moving objects.

Some IP cameras like the Sony SNC-RZ30 use MJPEG streams to deliver the video feed. Matlab does not offer a direct way to support this kind streams. Therefore a special driver is required to capture the MJPEG stream and to represent it in a Matlab conform way. There seems no official Sony driver available, therefore a freeware application/driver was used. The software is called Alax.Info JPEG Video Input Devices and is available through the following link: [http://alax.info/blog/1216](http://alax.info/blog/1216)