Design and control of a service robot

The birth of AMIGO

T.T.G. Clephas

CST 2011.060
Summary

This report describes the creation of the service robot depicted in Figure 1. At the department of Mechanical Engineering of the Eindhoven University of Technology there was a need for such a robot for several projects like the RoboEarth project, the Bobbie project and for the RoboCup @Home competition. The robots design is based on a new design of the Tech United TURTLE. This platform has been equipped with two Philips Experimental Robotic Arms to be able to grasp objects in a household environment. Due to the lifting mechanism, the robot can pick objects from the floor as well as from a table. To drive around, the robot makes use of four omni wheels, making it a fully holonomic platform that is capable of navigating through wheelchair-accessible areas. The main reason of using four wheels is to prevent the robot from tipping over in the case of a sudden stop.

Furthermore the robot is equipped with

- a laser range finder, for a two dimensional view of the environment,
- a Microsoft Kinect, for vision and three dimensional perception,
- two EtherCAT stacks from Beckhoff that provide the communication with the hardware,
- two Dynamixels, for the pan and tilt of the head and
- three AOpen computers.

The robot is going to use the Robotic Operating System. ROS is an open-source meta-operating system with already a lot of basic functionality for mobile manipulators available.

Since the robot makes use of open-source software there was a need for a open-source control architecture. The OROCOS framework is chosen to provide this functionality. Several components have been written to ease future design of controllers. The components provide basic functionality like communicating with the hardware or do operations on signals. These components can be deployed using a single script to determine the layout of the control structure. By using this configuration the low-level control, which consists of separate controllers for each wheel, is achieved.

The robot, named AMIGO, already performed in two successful demonstrators for RoboEarth and achieved 6th and 14th place in the RoboCup @Home competition in respectively the RoboCup German Open 2011 and the RoboCup World Cup 2011.
C  List of all components in the orocos_components package 61
D  Diagonal terms of the identification 63
E  Specification sheet 67
Chapter 1

Introduction

1.1 Background

Robots become more important in the everyday life of people. Think of autonomous vacuum cleaners and lawn mowers already present in households. Service robots are a class of robots that serve a more universal purpose; they help with various tasks in the household, rather than performing one specific task like a robotic vacuum cleaner.

1.2 Motivation

The Eindhoven University of Technology (TU/e) is participating in multiple projects involving service robots. One of them is the Bobbie project, this project will result in new methods to design a robot system, using standardized architectures, which can safely work in a care situation [1]. Another one is the RoboEarth project, which aims to design a World Wide Web for robots, the project will create a rich repository of shared knowledge from the experience of multiple robots [2]. Finally there is the RoboCup @Home competition. Tech United is the RoboCup team of the TU/e which plays in the Middle Size League and the @Home League. In the @Home League a robot has to perform several skills like understanding commands, recognize and follow persons and get arbitrary objects from a room.

No suitable robot to act as a demonstrator or participant in these projects was present at the department of Mechanical Engineering. There is however a lot of knowledge present within the department about mobile robots due to the soccer robots participating in the RoboCup Middle Size League competition.

The robot is going to use the Robotic Operating System [3]. ROS is an open-source meta-operating system with already a lot of basic functionality for mobile manipulators available.

1.3 Problem statement

The main goal of the project is to deliver a fully functional service robot operating on the ROS framework. This objective can be divided into several smaller goals.
1.3.1 Base

A new design of the Tech United soccer robot was available for the robot. It has to be determined if this wheel configuration is the most suitable one for a service robot. If so, the design must be adapted to the specific needs of a service robot, like coping with small obstacles and providing enough stability for a relatively tall robot.

1.3.2 Reach of the arms

The arms for the robot had already been chosen, they were the Philips Experimental Robotic Arms [4]. These have the same reach as a human arm, which means that if they are fixed on the robot, it will never be able to grab something from a cabinet as well as the ground. Therefore, a mechanism has to be designed in the base to increase this reach by being able to move the shoulders.

1.3.3 Sensors and actuators

In order to perceive the environment, the robot has to be equipped with several sensors and actuators. At least a two dimensional view of the environment is required for navigating through rooms. Furthermore a three dimensional view of the environment is required for grasping objects. And furthermore, the robot needs actuators with encoders to be able to move, computers to process all this data and an field bus to communicate with the hardware.

1.3.4 Software

To make the robot act, controllers have to be designed for all the actuators. Here to a real-time environment has to be chosen within the open-source scope of the project. Controllers have to be implemented preferably in a reusable architecture for future use.

1.4 Outline

The first part of the report describes the design of the robot. Since a household environment is typically designed for humans to operate in, the robot has about the size, weight and reach of an average human. Chapter 2 explains the choice for using omni wheels because of their holonomic principle and the choice for using four omni wheels to provide enough stability for the robot. In Chapter 3 the need for a lifting mechanism to extend the reach of the arms is explained. Here it also becomes clear why a spindle mechanism is used for the vertical movement of the upper body.

On the robot a lot of standard hardware components have been used. For example the arms, the vision system and the computers. In Chapter 4 the main components are described. In chapter 5, the communication between all the computers is explained. The three computers within the robot have to be able to communicate fast and reliably without having the robot attached to a cable the whole time.

The second part describes all the software used on and developed for the robot to perform basic operations. As described in Chapter 6, the robot uses the Robotics Operating System (ROS) for all middle and high-level software. The low-level software makes use of the OROCOS framework for real-time performance. Within this OROCOS framework several components have been created to ease future implementation of controllers. These components consist of filters like lead-lag and low-pass and of communication components to communicate with the hardware and the ROS software. The design of these components can be found in Chapter 7. Chapter 8 describes the
low-level control of the base which is created using these components. The final chapter contains the conclusions and recommendations.
Chapter 2

Mobile platform design

The design of a robot is usually based on the environment it has to operate in, or the task it is supposed to perform. This robot is supposed to perform numerous tasks in a household environment. So it serves a more general purpose than industrial robots for example. Since the robot has to perform in an environment that is designed for people, it is most convenient if the robot has the capabilities and reach of a normal person. For this reason, the arms chosen for the robot are the Philips Experimental Robotics Arms. These have the same dimensions and degrees of freedom as an average human arm. The arms are shown in Section 4.6.

2.1 The driving base

The robot should be able to drive around in a normal household environment. This means it should be able to drive over small obstacles like doorsteps and cables. The robot does not have to deal with large obstacles or very uneven surfaces. Therefore, a normal driving platform would suffice, there is no need for track wheels or a walking platform which are more difficult to create. A moving base can be full-holonomic, quasi-holonomic or non-holonomic. Holonomic means that the controllable degrees of freedom are equal to the total degrees of freedom. This means that the robot can move in any direction without needing to turn first. Full-holonomic platforms can immediately move in any direction in contrary to quasi-holonomic platforms that might need some time to position its wheels. Non-holonomic platforms, that steer like a car or a tank, always need some clearance to manoeuvre around, think of parallel parking. This is undesired since the robot will often have limited space available in a room. Furthermore, it is also more convenient to be able to drive sideways if objects on a table are out of reach for example. Because of these benefits, a holonomic platform is chosen.

The most common ways to achieve a holonomic platform are caster wheels, omni wheels or mecanum wheels. Caster wheels are wheels that can also be actively be rotated around a vertical axis. This way all the wheels can be set in the driving direction making it a quasi-holonomic platform. Omni wheels are wheels with little rollers at the sides. This way the wheel is driven in one direction and can roll freely in the perpendicular direction. Mecanum wheels have the same principle as omni wheels, but the rollers are placed under an angle. This means a different setup of the wheels can be realized since the direction of which the wheel can roll freely is different. In Figure 2.1, the different types of wheels considered are shown. The omni and mecanum wheels are only actuated in their rotational direction, only one actuator is needed. On the contrary, a caster wheel requires at least two actuators, one for the vertical rotation and one for the rotation of the wheel itself.

Caster wheels are better in coping with small obstacles since the wheels are always in the direction
of the movement. This makes it easier to drive over these obstacles. An omni or mecanum wheel, however, can encounter an obstacle in a direction perpendicular to the wheel. In this case the smaller roller has to be able to roll over the obstacle which is more difficult due to its smaller radius. The main disadvantages of caster wheels are the space required and the number of actuators. The reason caster wheels need more space is because they need room to rotate, this means that the actual space it occupies is larger than the volume of the components. This volume is estimated to be equal to or bigger than that of the omni or mecanum wheels since all contain a wheel and an actuator of about the same size since the same amount of driving power is needed. Regarding the actuators, at least two actuators are needed per wheel for the rotation and the forward movement. With three or four caster wheels this leads to six or eight actuators making the system highly over actuated. With omni or mecanum wheels the system is only over actuated with one actuator in the case of four wheels.

A positive side of using omni and mecanum wheels is that the robot can be pushed around in an arbitrary direction. Especially in the early software development, were the robot might not always be capable of moving itself, this can be very useful. Another important factor is the that there is quite some experience on the TU/e with omni wheels on the soccer robots from Tech United. A new mechanical design of the next generation Tech United TURTLE [5] has been made. Based on the assessment summarized in Table 2.1 and on the availability of an omni wheel platform, a choice has been made to use omni wheels.

Table 2.1: Assessment of omni and mecanum wheels versus caster wheels.

<table>
<thead>
<tr>
<th></th>
<th>Omni/mecanum wheels</th>
<th>Caster wheels</th>
<th>Omni/mec.</th>
<th>Caster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actuators per wheel</td>
<td>1</td>
<td>≥2</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>Drive passively</td>
<td>yes</td>
<td>Max. one direction</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>Build-in size</td>
<td>Wheel size + actuator</td>
<td>Space for rotation + actuators</td>
<td>+</td>
<td>–</td>
</tr>
<tr>
<td>Small obstacles</td>
<td>Orientation dependent</td>
<td>Well</td>
<td>–</td>
<td>+</td>
</tr>
<tr>
<td>Holonomic</td>
<td>Full</td>
<td>Quasi</td>
<td>++</td>
<td>+</td>
</tr>
</tbody>
</table>

2.2 Design of the omni wheels

The omni wheels are designed to be robust for different obstacles like cables anddoorsteps. The ability to drive over obstacles increases with the radius of the wheels. For this reason the diameter of the omni wheels was chosen on 150 mm. This size should allow the robot to drive over minor obstacles and the wheels would still fit in the design of the robot. Since the robot is full-holonomic, it can hit an obstacle in a perpendicular direction. Therefore the rollers on the circumference have
also been designed with a relatively large radius. In order to increase the grip on the wheels, rubber rings have been put around the rollers. All the rollers are individually mounted with bearings in axial and radial directions see Figure 2.2. In this figure an exploded view of one omni wheel can be seen. The whole wheel is mounted onto the motor axle of a Maxon motor. The bearings in the gearbox of this motor are strong enough to handle the load on the wheels, this way no additional bearings are required.

2.3 Wheel configuration

The new mechanical design of the next generation Tech United TURTLE consists of four omni wheels but this could fairly easily be adapted to a three wheel configuration. The advantage of this is that it is a statically determined configuration. This means all three wheels will always have contact with the floor under normal circumstances. In a four wheel configuration one or two wheels could loose contact with the ground if one wheel is standing on a doorstep for example.

Figure 2.3 shows two layouts for a three an a four wheel configuration. The wheel sizes of the configurations are identical since they experience the same challenges as explained in Section 2.2. These two configurations have been fit on a design with maximum dimensions of 600 × 600 mm. In this Figure also the tilting line of the robot is displayed. If the location of the center of mass of the robot would pass this line, the robot would fall over. This might happen because of a collision or heavy breaking.

To determine the risk of falling over, the location of the center of mass (c.o.m.) is determined. The worst case scenario for falling over is that the robot is lifting a load of 3 kg with fully extended arms and an extended base. Appendix A shows that the center of mass is 60 mm in front of the middle point of the robot and at a height of 600 mm. The center of mass is displayed in Figure 2.4. Since the location of the center of mass and the tilting line is known, the angle $\alpha_{\text{tilt}}$ at which the robot would fall over can be easily determined:
\[ \alpha_{\text{tilt}} = \tan^{-1}\left( \frac{x_A - x_{\text{com}}}{z_{\text{com}}} \right) \]  

(2.1)

\( x_{\text{com}} \) and \( x_A \) are the distances of the center of mass and the point A, relative the the middle point of the robot and \( z_{\text{com}} \) is the height of the center of mass measured from the bottom of the wheels. The values of \( \alpha_{\text{tilt}} \) are 14.6° and 6.7° for four and three wheels respectively. This confirms the fact that the robot is more likely to fall over if equipped with only three omni wheels.

The potential energy \( U \) of the robot at these angles can be determined using the change of height of the center of mass \( \Delta h_{\text{com}} \):

\[ h_{\text{com}\@\alpha_{\text{tilt}}} = \sqrt{(x_A - x_{\text{com}})^2 + z_{\text{com}}^2} \]  

(2.2)

\[ \Delta h_{\text{com}} = h_{\text{com}\@\alpha_{\text{tilt}}} - z_{\text{com}} \]  

(2.3)

\[ U = mg\Delta h_{\text{com}} \]  

(2.4)

The case considered is a blocking of the front wheels with no slip. The blocking of two wheels is unlikely but if something was dropped into the wheel case one wheel could easily block. Furthermore, the wheels have a lot of grip on a compliant floor making this a realistic worst case scenario. If damping is neglected between the wheels and the floor, all the kinetic energy \( K \) the robot has is converted into a rotational motion around point A in Figure 2.4. So if the kinetic energy of the robot is larger than the potential energy at the tipping point, the robot will fall over:

\[ K > U \]  

(2.5)

\[ \frac{1}{2}mv^2 > mg\Delta h_{\text{com}} \]  

(2.6)

\[ \frac{1}{2}mv^2 > mg(h_{\text{com}\@\alpha_{\text{tilt}}} - z_{\text{com}}) \]  

(2.7)

\[ v^2 > 2g\left( (x_A - x_{\text{com}})^2 + z_{\text{com}}^2 - z_{\text{com}} \right) \]  

(2.8)

The value of the maximum velocity for a three wheel configuration is 0.29 m/s (1.0 km/h) and for a four wheel configuration it is 0.62 m/s (2.3 km/h). This means that if a four wheel platform would come to a sudden stop going faster than 0.6 m/s it would fall over. Unfortunately this has also been experimentally established.

Furthermore the current design of the next generation Tech United TURTLE can be scaled up to any value. Increasing the four wheel base to a 700 × 700 mm platform would increase the maximum velocity by 22%, but this would make it harder to navigate through doors and it would
Figure 2.4: The robot will fall over if the center of mass (c.o.m.) will be at the right side of the point A.
limit the range of the arms outside the base. A $600 \times 600$ mm platform was considered big enough to prevent the robot from falling.

Having these results a decision was made to keep the current design of the next generation Tech United TURTLE at four wheels but to expand it to $600 \times 600$ mm. Note that the maximum velocity of the robot can easily be increased by lowering the robot when driving, since it is a function of the height of the robot.
Chapter 3

The lifting mechanism

In order to pick up objects from the floor as well as from a tabletop or kitchen cabinet, the robot needs a large vertical range. Since the arms have only a limited operating range, the upper body should be able to move vertically. Three main options were considered to achieve this. One was a scissor lift as depicted in Figure 3.1. The main advantage is that a large vertical range can be reached with little space required in folded position. However due to the large amount of joints, play will accumulate and the stiffness will decrease significantly. Option two is a robot with two joints as in Figure 3.2. The advantages are a large vertical range and easy wiring due to only rotating components. Also the horizontal range of the arms is extended. This construction is however harder to realize on a short notice since the joints have to be custom designed and most likely some FEM analyses are required to make the construction strong enough. Furthermore, since a lot of torque is required on the joints, it is hard to guarantee some safety, since in case of failures in soft- or hardware the robot might collapse quickly. The third option is a vertical slider like in Figure 3.3. This spindle mechanism uses a lead screw and a ball nut to create the vertical movement. The main advantage of this is that it consists of only standard components and that it is easily controlled. Also the safety aspect is relevant: due to the high gear ratio the robot will in the worst case scenarios only slowly founder.
Figure 3.1: Scissor elevator:
This mechanism requires little space but its stiffness is low when fully extended.

Figure 3.2: Two-joint mechanism:
High reach but harder to implement and to secure
Figure 3.3: Spindle elevator:
Stiff design, reach is limited.
Figure 3.4: Section view of the front half of the robot.

1) is the motor driving the spindle,
2) is a brake acting on the shaft of the motor
3) is the encoder of the motor
4) is a 4.8:1 gearbox
5) a tooth-belt transferring the power of the motor to the spindle
6) is the spindle
7) is the spline-nut lifting the torso when the spindle is rotated
8) are two rails guiding the torso in a vertical direction
9) a tube protecting the spindle and transferring the force from the spline-nut to the torso
10) a slip coupling, preventing damage if the nut 7) reaches end stop
11) the batteries powering the robot
12) A Beckhoff EtherCAT stack, see Section 4.3
3.1 Drive chain

To reduce friction, a spline-nut is chosen to move over a spindle. The chosen spindle is capable of lifting a load of almost 5 kN according to the specifications [6], which is enough to lift the 26.8 kg estimated for the torso, see Appendix A.

In Appendix B the required torque and rotational speed for motor / gearbox combination is calculated. The motor should deliver a torque of 21.5 Nm at a speed of 120 RPM.

A combination of a 90 W Maxon motor and a gearhead with a 4.8:1 reduction can deliver this amount of torque and speed. It is even possible to lift the robot faster than the 5 seconds used in the calculation by overloading the motor. This is possible since the motor only has to deliver this power for a short amount of time.

The torque is transmitted from the gearbox to the spindle using a tooth belt. Using a tool from the manufacturers website [http://smarthost.maedler.de/maedlertools/maedler.html](http://smarthost.maedler.de/maedlertools/maedler.html), a tooth belt is chosen. The values used for the tool are an input torque of 0.3 Nm at 1335 RPM and a safety factor of 10. The tooth belt has a width of 10 mm and the gears have each 20 teeth.

3.2 Construction

The construction of the spindle mechanism is shown in Figure 3.4. In this figure it can be seen how the motor and gearbox are connected to the spindle with the tooth belt. The spindle is supported at the bottom with a bearing. A spline nut is put around the spindle to provide the vertical movement.

The stiffness of the construction comes from four rails as depicted in Figure 3.4. These rails fix the translational motion in $x$, $y$ and $\theta$ direction, the $z$ direction is prescribed by the spindle. The tube, nr. 9) in Figure 3.4, fixes the rotation around the $x$ and $y$ axis. This way all the degrees of freedom of the torso are fixed.

To prevent damage to the robot if the spline-nut reaches an endstop, a slip coupling is added that connects the tube to the torso. Now, if the spline-nut reaches an endstop, the whole tube, which is connected to this nut, starts to rotate through the slip coupling. Since the spindle and the tube have the same speed, there is no vertical movement. The endstops are implemented in the rails which have a limited range.

To solve the problem of the wiring, a flexible cable guidance is used. In Figure 3.5 it can clearly be seen in the back of the robot.
1) The cable guide
2) Three computers
3) A Beckhoff EtherCAT stack, see Section 4.3
Chapter 4

Additional hardware components

In the previous chapters the design of the robot has been explained. However, a bare frame is not a robot yet; a robot has to interact with the environment. Therefore, it is equipped with sensors and actuators. To perceive the environment the robot is equipped with two main sensors described in Section 4.1 and 4.2. The first is a laser range finder which is mainly used for navigation. The second is a Microsoft Kinect containing both an RGB camera and a depth camera. Furthermore, for actuation the robot is equipped with motor/encoder combinations on the wheels, spindle and neck. The conversion to and from digital signals for the motor/encoder combinations is done by an EtherCAT stack from Beckhoff, described in Section 4.3. The neck uses a special type of motor/encoder combination named Dynamixels. These type of actuators are discussed in Section 4.4. Another important part of the robot are the arms described in Section 4.6. The control of all these components is done by three AOpen computers, described in Section 4.5. The way the whole robot is powered is discussed in Section 4.7. Finally it is very important to let all this hardware interact with the environment in a safe way. How this is ensured is described in Section 4.8.

4.1 Laser range finder

In order to drive around, the robot needs to be aware of obstacles around it. Furthermore, a topographic map of the environment is required for navigation. In the research community of mobile robots, direct measurement of ranging data using a laser range finder (LRF) or laser imaging detection and ranging (LIDAR) is a common method [7].

A laser range finder is a device which uses a laser beam to determine the distance to an object. The most common form of laser range finder uses the time of flight principle by sending a laser pulse in a narrow beam towards the object and measuring the time taken by the pulse to be reflected off the target and returned to the sender. In laser range finders, the laser beam is aimed at a rotating mirror. This way a two dimensional array of points is created, like in Figure 4.1.

Instead of laser range finders other methods can be used such as stereo vision, a time of flight camera or a Kinect as described in Section 4.2. However, an LRF is typically faster than the other methods, having an update frequency up to 40 Hz and since the data is only two dimensional, it can also be processed faster. Another advantage is the wide angle; an LRF can have scan angles up to 270°. This way the robot can also see on the sides, using only one sensor.

A different common use of laser range finders in industry is safety. Certified laser range finders can be used to disable hardware in certain circumstances. For example, the amplifiers of the actuators can be disabled if it is coming to close to an object or vice versa. The main two reasons not to
implement this on the robot where 1) the lack of data provided by these LRFs. They only give some status signals, not all the laser data which is needed for navigation. And 2), in some cases it is desired for the robot to approach an object really close. In such a case, a relatively simple bumper detection might suffice as safety. This is not (yet) implemented on the robot.

In Table 4.2, a comparison between suitable laser range finders has been made. The Hokuyo URG UTM-30LX is a close second in terms of performance to the Sick LMS 100, but is only 20% of its size. Therefore a choice has been made for the Hokuyo.

Table 4.1: Hokuyo UTM-30LX specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power source</td>
<td>12 V ±10%</td>
</tr>
<tr>
<td>Current consumption</td>
<td>0.7 A (max. 1.0 A)</td>
</tr>
<tr>
<td>Detection range</td>
<td>0.1 to approx. 60 m (&lt;30 m guaranteed)</td>
</tr>
<tr>
<td>Laser wavelength</td>
<td>870 nm, Class 1</td>
</tr>
<tr>
<td>Scan angle</td>
<td>270°</td>
</tr>
<tr>
<td>Scan time</td>
<td>0.025 s/scan (40.0 Hz)</td>
</tr>
<tr>
<td>Angular Resolution</td>
<td>0.25°</td>
</tr>
<tr>
<td>Interface</td>
<td>USB 2.0 with trigger port</td>
</tr>
<tr>
<td>Weight</td>
<td>0.233 kg</td>
</tr>
<tr>
<td>Measurement error</td>
<td>0.1 to 10 m (±30 mm)</td>
</tr>
<tr>
<td></td>
<td>10 to 30 m (±50 mm)</td>
</tr>
</tbody>
</table>
### Table 4.2: Laser range finders comparison

<table>
<thead>
<tr>
<th></th>
<th>Hokuyo UTM-30LX</th>
<th>Sick LMS 100</th>
<th>Hokuyo URG-04LX</th>
<th>Keyence SZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>40 Hz</td>
<td>50 Hz</td>
<td>10 Hz</td>
<td></td>
</tr>
<tr>
<td>Angle</td>
<td>270°</td>
<td>270°</td>
<td>240°</td>
<td>270°</td>
</tr>
<tr>
<td>Resolution</td>
<td>0.25°</td>
<td>0.25°</td>
<td>0.352°</td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>30 m</td>
<td>30 m</td>
<td>4 m</td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td>≤10 m: ±30 mm</td>
<td>30 mm</td>
<td>≤1 m: ±10 mm</td>
<td>≥1 m: 1%</td>
</tr>
<tr>
<td></td>
<td>&gt;10 m: ±50 mm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>60x60x87</td>
<td>106x102x152</td>
<td>50x50x70</td>
<td>100x104x149</td>
</tr>
<tr>
<td>Safety</td>
<td>No</td>
<td>Safety or data output</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 4.2 Microsoft Kinect

In order to recognize and identify persons, objects and obstacles a camera needs to be mounted onto the robot. However, apart from the two-dimensional image, there is also a need for three-dimensional information about the environment. This can be used for the recognition but also for example to detect objects that are out of sight of the LRF, like a tabletop. This three-dimensional information can be obtained for example by correlating features from two images (stereo vision) or using a time of flight camera.

A first choice has been made for the Bumblebee® XB3 CCD Camera. It features three 1.3 megapixel sensors and has two baselines available for stereo processing. The extended baseline and high resolution provide more precision at longer ranges, while the narrow baseline improves close range matching and minimum-range limitations [8].

In November 2010 Microsoft launched the Kinect, shown in Figure 4.3. It is designed for the XBox 360, a video game console. The Kinect is capable of producing depth and RGB streams at a price much lower than traditional range sensors. It projects an infrared grid across the scene to obtain deformation information of the grid to model surface curvature [9]. We use the OpenKinect driver framework for the Kinect that produces 640 × 480 RGB and depth images at 30 fps.

![Figure 4.3: Microsoft Kinect](image)

![Figure 4.4: Bumblebee® XB3 CCD Camera](image)

#### Table 4.3: Microsoft Kinect specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal field of view</td>
<td>58°</td>
</tr>
<tr>
<td>Vertical field of view</td>
<td>45°</td>
</tr>
<tr>
<td>Resolution</td>
<td>VGA (640H × 480V)</td>
</tr>
<tr>
<td>Frame Rate</td>
<td>30 fps</td>
</tr>
<tr>
<td>Depth sensor:</td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td>VGA (640H × 480V)</td>
</tr>
<tr>
<td>Frame Rate</td>
<td>30 fps</td>
</tr>
<tr>
<td>Range</td>
<td>0.8 - 3.5 m</td>
</tr>
<tr>
<td>Spatial x/y resolution</td>
<td>3 mm (@ 2 m distance)</td>
</tr>
<tr>
<td>Depth z resolution</td>
<td>1 cm (@ 2 m distance)</td>
</tr>
</tbody>
</table>

#### Table 4.4: Bumblebee specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal field of view</td>
<td>66°</td>
</tr>
<tr>
<td>Trigger signal</td>
<td>yes</td>
</tr>
<tr>
<td>Image sensor:</td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td>1280H × 960</td>
</tr>
<tr>
<td>Frame Rate</td>
<td>16 fps</td>
</tr>
</tbody>
</table>
4.3 Input / output

The input and output (I/O) of the hardware to the computers is done via several communication protocols. The Microsoft Kinect and the laser range finder are equipped with a USB-interface. The laser range finder also provides a trigger signal so that time-stamping of the signals can be in synchronization with the computer clocks. The Philips arms have on-board dedicated USB amplifiers. These amplifiers include a controller, but it is possible to access the amplifiers and encoders directly as well, so the controllers can be built and implemented on the computers.

For the ease of wiring, reconfigurability and modularity of the remaining communication, a fieldbus system was chosen. The TU/e has already quite some experience with the EtherCAT protocol. EtherCAT, Ethernet for Control Automation Technology, is an open high performance Ethernet-based fieldbus system. The development goal of EtherCAT is to apply Ethernet to automation applications which require short data update times with low communication jitter and low hardware costs [10]. A big advantage of this system is that it can be controlled using a normal Ethernet port of a computer. The implementation of the EtherCAT is done with the Beckhoff EtherCAT terminals. The robot has two Beckhoff stacks with a total of 15 slaves:

**EK1100** (2x) The EK1100 coupler connects the Ethernet cable with the EtherCAT Terminals (ELxxxx)

**EL5105** (6x) The EL5101 EtherCAT Terminal is an interface for the direct connection of incremental encoders with differential inputs. These encoders are used for the four wheels and the spindle motor.

**EL1008** (2x) The EL1008 digital input terminal acquires a maximum of eight binary control signals from the robot. These terminals read the status of the amplifiers and the status of the four fuses located at each battery. If an additional EtherCAT slave like the EL1018 would be added to the stack, the trigger signal of the laser range finder can be read back. The current slaves (EL1008) have a 3.0 ms input filter which prevents this.

**EL2008** (2x) The EL2008 digital output terminal sends a maximum of eight binary control signals of 24 V to the robot. The signals from these terminals are used to operate the TU/e-lights, the amplifiers and the spindle break. The signal to enable the amplifiers can be interrupted by the emergency buttons as described in Section 4.8.

**EL3102** (1x) The EL3102 analog input terminal processes two signals in the range between -10 and +10 V. The voltage is digitized to a resolution of 16 bits. One of these inputs is used to measure the voltage of the batteries. In the future this might be extended to measure the current as well.

**EL4038** (1x) The EL4038 analog output terminal generates eight signals in the range between -10 and +10 V. The voltage is supplied to the process level with a resolution of 12 bits. This terminal provides the signal for the amplifiers of the five motors driving the wheels and the spindle.

**EL6022** (1x) The EL6022 serial interface enables the connection of devices with two RS232 or two RS422/RS485 interfaces. These interfaces are used by the Dynamixels described in Section 4.4

It can be seen that the amplifiers and encoders of the Philips arms are not controlled by the EtherCAT stacks. To make the I/O of the robot more uniform it is recommended to provide these arms with an EtherCAT interface instead of USB.

4.4 Dynamixels

The neck of the robot consists of two Dynamixels. A Dynamixel is an actuator with a built-in controller. It consists of a DC motor, a gearbox and a control circuitry. Dynamixels can be mounted on top of each other, making them very modular.
The first design of the robot contained a custom pan-tilt unit, which was damaged during the first tests. It was replaced by Dynamixels because of their ease of use. It led to less cables through to cable belt, since only one data connection using two was needed instead of 12 cables for the encoder and amplifiers of the motors.

The pan-Dynamixel is placed inside the upper body to be able to have the tilting joint as low as possible. In other words, with the same height of the head, the part of the neck above the tilt joint is longer. As a result, if the robot looks down, the view of the Kinect is less affected by the covers from the upper body. The RX-28 Dynamixels used for the neck are displayed in Figure 4.5. These are capable of delivering a torque of 3.6 Nm and have a minimum control angle of 0.29°.

It has not been tested how accurate the Dynamixels operate. The Dynamixels operate via the RS485 Asynchronous Serial Communication Protocol. On the EtherCAT stack, as described in Section 4.3, an EL6002 serial interface enables the connection to the RS485 interface used by the Dynamixels.

### 4.5 Computer hardware

The brain of AMIGO is located in three computers inside the base. For computational power it holds during development that more is better. For comparison, the PR2, a robotics research and development platform from Willow Garage [11], holds two on-board servers containing each two Quad-Core i7 Xeon processors (8 cores) with 24 GB of ram. However due to the slim design of AMIGO, limited space was available. The combination of computers given the most computational power within the space available were three AOpen computers containing a Dual-Core i5 processor and 8 GB of ram. One computer is equipped with an extra Ethernet card for EtherCAT.
Table 4.6: AOpen DE57-HA specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>166 × 48 × 157</td>
</tr>
<tr>
<td>Weight</td>
<td>1.22 kg</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel® Core™ i5</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB DDR3 1066 MHz</td>
</tr>
<tr>
<td>LAN</td>
<td>Intel® Gigabit Ethernet</td>
</tr>
<tr>
<td>USB</td>
<td>USB 2.0 Port × 4</td>
</tr>
<tr>
<td>Storage</td>
<td>S-ATA 500 Gb</td>
</tr>
<tr>
<td>Power</td>
<td>90 W max</td>
</tr>
</tbody>
</table>

4.6 Arms

The arms used for the service robot are the Philips Experimental Robotics Arms (PERA). They are a prototype developed by Philips Research. The arms have roughly the same dimensions and payload, one stretched arm can lift 1.5 kg, of a human arm. The degrees of freedom of the arms can be seen in Table 4.7, the lengths of the joints are depicted in Appendix E. More information about the hardware and control of the arms can be found in the report of B. Willems [4].
Table 4.7: Philips Experimental Robotic Arms specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joints</td>
<td>3x differential drive</td>
</tr>
<tr>
<td>Shoulder yaw</td>
<td>180°</td>
</tr>
<tr>
<td>Shoulder roll</td>
<td>90°</td>
</tr>
<tr>
<td>Shoulder pitch</td>
<td>180°</td>
</tr>
<tr>
<td>Elbow yaw</td>
<td>210°</td>
</tr>
<tr>
<td>Elbow pitch</td>
<td>145°</td>
</tr>
<tr>
<td>Wrist yaw</td>
<td>90°</td>
</tr>
<tr>
<td>Wrist pitch</td>
<td>114°</td>
</tr>
<tr>
<td>Max load</td>
<td>1.5 kg (straight arm)</td>
</tr>
<tr>
<td>Force control</td>
<td>All joints</td>
</tr>
<tr>
<td>Safety</td>
<td>Slip coupling on all joints</td>
</tr>
</tbody>
</table>

4.7 Power

The robot acquires its power from four Makita Ni-MH (BH2443) batteries. They are located two by two at the sides of the base as seen in Figure 1. The batteries can deliver 3.3 Ah at 24 V each. The reason to use these batteries is the long experience of Tech United with the batteries in the middle size league. The batteries are easily exchangeable on the fly. Since the batteries are all connected in parallel, they can be exchanged one by one without the need to power down the whole robot. This way development can continuously take place while the batteries are recharged.

All the power of the electronics in the robot is provided by DC-DC converters to ensure the right voltage. The motors are connected to the raw power of the batteries. Furthermore the voltage of the batteries can be measured. A small board proportionally brings back the voltage to a maximum of 10 V which is then read back at an analog input port of the EtherCAT stack.

4.8 Hardware safety

It is important that the robot operates safely in its environment. However, the software cannot be guaranteed to be safe in the early stage of development and most likely it cannot be guaranteed in later stages as well. Therefore, is chosen to be able to disable all the actuators on the robot and to apply a brake to the lifting mechanism. The effects of this choice will be given for the three most important actuators: the arms, the spindle and the wheels.
If the actuators of the arms are disabled the arms will relatively slowly drop down. In later stages it might be desired to hold the position of the arms instead of disabling the power, this way it will for example not spill the glass of water over itself. The current hardware design of the arms however does not accommodate such a feature. Due to the light weight of the arms, see Section 4.6, the drop will not cause a safety issue.

The spindle mechanism is equipped with a brake on the end of the motor. When the power is disabled, it will clamp the motor axle. This way the robot will stay at the same height, preventing something or someone to be squeezed between the lower and upper body. Furthermore if the brake would not enable for some reason, the spindle would slowly founder due to its high gear ratio. However, a snapping of the tooth belt, described in Section 3.1, will still cause the robot to go down fast since the inertia and friction of just the spindle is much lower than the inertia and friction of the spindle-motor combination.

The actuation of the base will impose the highest risk. The robot is equipped with $4 \times 150$ W motors which have in combination with the amplifiers a peak power of more than 1 kW. For normal speeds, which the robot should not exceed, disabling the power will result in a quick stop. At high speeds there is a risk of tipping over (Section 2.3) if brakes would be applied. So disabling the power was decided to be the safest option.

The power to the amplifiers of the wheels and arms can be disconnected by two emergency buttons. One is located at the back on top of the computers. The second one is a certified wireless emergency button from Tyro products. This emergency button works with a heartbeat to ensure that the button is never out of range.

The safety of the arms only consists of disabling the amplifiers of the arms. But due to the light weight of the arms and the relatively weak actuators, this is sufficient. In future the arms might be designed different in order to keep them at the current location if an emergency button is pressed.
Chapter 5

Communication

5.1 Network setup

The robot has three computers, one router and a Gigabit Ethernet switch. The computers and the router each have a one Gigabit connection to the router. The router, a Cisco Linksys E2000, acts as an access point to connect to the local wireless network. Another router separate from the robot provides this wireless N network and acts as a DHCP server for the operating computers connected to this router. This way all computers can communicate with each other and with the robot. In Figure 5.1 an overview of the network can be seen.

If a wired connection is preferred over a wireless connection, for performance reasons or regulatory reasons during an @Home competition, the router on amigo can be unplugged and the switch on the robot can be connected to the external switch or router. This way the infrastructure stays the same, only the bandwidth is increased due to the wired connection. This is especially useful during development since a lot of data can be visualized over a fast connection.

One main advantage of this infrastructure where one can choose between a wireless or wired connection is that software development can take place on the robot itself. This makes debugging and testing new features very easy and fast. Furthermore the robot is always able to connect to the internet, which is useful for keeping the systems up to date, but this is also necessary to connect to the RoboEarth database.

Except the wireless and wired option to the external network there is a third option. This third option is to unplug the router on the robot and attach a computer with a fixed IP address to the switch on the robot. In this case the robot can only be controlled by this computer. This can be useful if only a starting command has to be given.
Within the robot

Figure 5.1: Network layout, the switch within the robot can be connected to 1) the access point within the robot for wireless communication, 2) a switch for communication with multiple computers or 3) a single computer without access to the internet.
5.2 Software communication

The communication of all the executables on the computers is managed by the ROS master. The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer [12]. This can happen either locally or over the network.

5.3 Timing

Since all the computers receive sensor data in some sort, it is important that the system clocks in these computers are in sync. For this reason the first computer on the robot acts as a time server. The other computers on the robot synchronize with this computer to keep the times as exact as possible. The estimated time difference is $16 \mu s$ according to the Network Time Protocol (www.ntp.org).
Chapter 6

Software architecture

6.1 Robot Operating System

ROS is an open-source project to build a meta-operating system for mobile manipulation systems. The software which drives the robot can be divided into several subcomponents. These include reading the sensor data, controlling the motors but also higher level nodes like planning, localize objects and state executioners. In order to let these nodes communicate with each other we use the Robot Operating System (ROS).

‘ROS provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more’ [3].

A lot of the hardware and desired capabilities of the robot have been implemented on other robots using ROS as well. This means a lot of basic functionality is already available. ROS also takes care of the communication between nodes, even amongst several machines. In this case the first computer of the robot always services a roscore. All nodes, including nodes on the other machines, subscribe to this core. The role of the roscore is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. In this robot the newest version of ROS called diamondback is used.

6.1.1 Teleoperation

To demonstrate the functionality of the hardware, three teleoperation nodes have been written in ROS. The most extensive one uses a SpaceNavigator from 3Dconnexion. This is a 3D mouse which acts as a 6 DOF joystick to control the robot. With the click of a button, this joystick switches between different elements of the robot to teleoperate. It can switch between the head (pan/tilt), the base (x, y and θ velocities) and the arms (6 DOF of the gripper). This makes it easy to test the hardware of the robot as well as giving interactive demos.

Another teleoperation node uses the numpad from the keyboard to drive the robot around, the main advantage of this node is that it requires no additional hardware. The last one uses the accelerometers of a Nintendo Wii remote to drive around. By tilting the remote forward the robot starts to drive and by tilting it sideways the robot makes a turn. This is especially useful if you want to drive around without taking a whole laptop with you. In the future a PlayStation 3 controller might be added to this list. This controller has more buttons and has two joysticks which give more freedom to control the robot, this way for example the arm joints can be controlled using the controller.
6.2 OROCOS Real-Time Toolkit

Since ROS is not a (semi) real-time environment, other free software control projects like OROCOS [13], A Robot Control "C" Library [14] and The Chimera II Real-Time Operating System [15] were investigated. The OROCOS Real-Time Toolkit [13] was chosen because of the modularity and the support for the EtherCAT modules from Beckhoff used in the service robot. The presence of these EtherCAT drivers means that the communication with the hardware is already present, which is a big advantage for the quick startup of the robot.

OROCOS stands for Open Robot Control Software and is developed by the Katholieke Universiteit Leuven [13]. With this toolkit it is possible create real-time robotics applications using modular, run-time configurable software components. It provides a generic real-time core that can be used to build applications with various architectures [16]. The idea is similar to the ROS layout, it is created as a portable C++ library and a ROS wrapper is available. This means that the real-time environment can listen and publish to ROS topics. For example to receive reference values and to publish sensor data. In Chapter 7 it is explained how the low-level controller structure of the robot looks like using several OROCOS components.

The current implementation is a soft real-time environment. The performance of this soft real-time environment has not extensively been investigated. Better performance might be gained by using a real-time kernel like Xenomai or RTAI. However, the effort of implementing this should be measured against the gain in performance [17].
Chapter 7

OROCOS Components

7.1 Deploying components

In the previous chapter a choice has been made to use the OROCOS framework for all the real-time processes of the robot. In the OROCOS Component Library (OCL) from the K.U. Leuven there are already a number of standard OROCOS Components available. One of these components is the DeploymentComponent available for loading and configuring other components using an OROCOS script [18].

This framework and these components make it easy to write an own library of components that can easily be deployed using the DeploymentComponent. In Figure 7.1 the basic steps of deploying components is shown: an .XML or .ops file contains instructions for the DeploymentComponent where to look for components, which component types to create, which name they must be given and how their internal thread is configured (priorities, periods,...). Furthermore this file describes the network interconnections between all components and how data must be relayed from one component to another. These components are then imported, created, configured and finally started if specified by the .ops file. [18]

Figure 7.1: This figure shows the basic steps of deploying components using the DeploymentComponent.
7.2 Component layout

Each component typically has one or more input ports and an output port. These ports are used to let the components communicate with each other. These ports can also be streamed to ROS topics. The components are then able to publish to or read from a ROS topic.

Every component also consists of several states it can reside in. The states are depicted in Figure 7.2. All self-written components start in the PreOperational state. They are configured by the .ops file. If this returns true it enters the Stopped state upon which the startHook() can be called. The component is then in the Running mode where the updateHook() is called at a certain frequency or until it is triggered. A component can be triggered if it has so called trigger ports. If new data is received on any of these ports, the updateHook() will be run once.

Figure 7.2: An overview of the states a component can have. Source: OROCOS Cheat Sheet [19]. NB. The exceptional states are not used in the base controllers.

7.3 Components

At first one component for the entire low-level control of the base was created. This was one complex component taking care of reading the reference values from ROS, controlling the wheels and including safety features needed for the base. After this, controllers had to be designed for both the arms and the spindle. In the future it might be needed to control even more actuators. In the current setup already a total of nineteen signals have to be controlled.

To ease future implementation of low-level controllers, the control design is divided into separate components, each component doing only one task or operation. This means that for an easy controller you would need five components like depicted in Figure 7.3. One for reading the reference values from ROS, one for receiving and sending signals to the actual hardware and a Gain and lead-lag component acting as the controller. This way, controllers can be built and designed in a modular way, like it is done for example in Matlab Simulink.

All the components use vectors to communicate. This way all the components can be linked to each other and multiple values can be send at once. This prevents the use of a lot of the same components if for example a 7 DOF arm needs to be controlled. Next, a list of all the components
created in the library is given. After every component is stated if it runs at a fixed frequency rate or that it is triggered by an trigger port as described in Section 8.1.

7.3.1 Mathematical operations

- **Gain**
  
  Triggered component, it consists of one input port and one output port.
  
  This component multiplies every incoming vector with a gain. This gain is set by a parameter.

- **Gains**
  
  Triggered component, it consists of one input port and one output port.
  
  This component does the same as the Gain component, but every element in the vector can have a different value for the gain.

- **Addition**
  
  Consists of two input ports and one output port, the first input port is triggered.
  
  The two vectors received will be added element-wise. The component will give an error is the sizes of the vectors don’t match. Note that if both ports would be a trigger port, the component would output a vector at every trigger. This means all operations after this will run at a higher frequency than either of the inputs, which is unwanted in most systems.

- **Subtraction**
  
  Consists of two input ports and one output port, the input port that is subtracted is triggered.
  
  The vector received on the second port is subtracted from the vector received by the first port. The reason the triggering is done on the second port is only because a control-loop as in Figure 7.3 will no be triggered by the reference but will run at the rate of the loop. There is still room for improvement in the Addition and Subtraction components. They could be merged into one universal component that can add or subtract multiple signals by setting a parameter like ‘+-’. The port(s) to be triggered could be indicated by an exclamation mark.

- **MatrixTransform**
  
  Triggered component, it consists of one input port and one output port.
  
  The incoming vector of the component is multiplied with an \( m \times n \) transformation matrix, where \( n \) is the size of the incoming vector and \( m \) the size of the outgoing vector. The parameters of this component are \( m, n \) and the transformation matrix itself.

- **Integrator**
  
  Consists of two input ports and one output port, the first input port is triggered.
  
  This component integrates the elements of the vector, obtained by the first port, simply by multiplying the current input value with the sample-time and adding it to the previous output value. This can be improved, but it is good enough since it is currently only used for converting the reference velocity to position. The second port is used to reset the integrator. If a new value is received at this port, the integrator resets itself to this value and continues operation.
7.3.2 Control components

- **LeadLag**
  
  *Triggered component, it consists of one input port and one output port.*
  
  This Component acts as a lead-lag filter on the elements of the input vector. The zero and pole can be set by parameters for each vector within the element separately. The algorithm is written by B. Mrkajić [20] and uses the transposed-direct-form II (TDF-II) structure for implementation and Tustin method with prewarping for discretization.

- **SecondOrderLowPass**
  
  *Triggered component, it consists of one input port and one output port.*
  
  This component is the same as the LeadLag component, only it acts as a second-order low-pass filter and it has the pole frequency and the pole damping as parameters.

7.3.3 Input/Output

- **SoemMaster** (This component is not part of the library but is included for completeness.)
  
  *Consists of multiple input ports and multiple output ports, one per slave, runs at a fixed freq.*
  
  For communication with the EtherCAT stack there is already a SoemMaster component present in the git repositories of K.U.Leuven. It is a wrapper for the Simple Open EtherCAT Master [21]. When this component is configured it detects all the slaves attached to the computer and creates a module with one port for every slave available. The type of the port depends on the slave.

- **AnalogOut**
  
  *Consists of multiple input ports and one output port, all the input ports are triggered.*
  
  The SoemMaster component accepts a vector of values to set the value for the analog outputs of a single Beckhoff slave. If, for example, the controller for the spindle sets one value, the rest of the values will be set to zero. This is conflicting with the controllers of the wheels, which also send values to the SoemMaster. To circumvent this behaviour the AnalogOut component is written. It accommodates one port for every function and aggregates these values to a single vector which is send to the output port.

- **DigitalOut**
  
  *Consists of multiple input ports and one output port, all the input ports are triggered.*
  
  This component serves the same purpose as the previous one. However it also accommodates some safety. The brake of the spindle and the enable signal of the amplifiers are connected to the same Beckhoff slave. This component makes sure that the amplifiers are never enabled if the brake is active.

- **ReadEncoders**
  
  *Triggered component, it consists of one input port and one output port.*
  
  Since every encoder is attached to a single slave, there is also one message per encoder coming from the SoemMaster. This component takes multiple encoders signals and puts them as separate elements in a vector. This way the rest of the components can accept them.

  The first parameter is a vector of values with the conversion between encoder steps and radians. This way the values of the output have more meaning. From the size of this vector the component determines the number of encoders to be put in one output vector. For the wheels this is typically four and for the spindle just one. Another parameter sets the saturation value of the encoder. The component detects this saturation and compensates for this.
7.3.4 ROS components

- ReadTwistMsg
  Triggered component, it consists of one input port and one output port.
  A component that reads values from a ROS topic and merges them in a vector. This component can only cope with ROS topics that contain messages of the type Twist. This message contains the six velocities corresponding to the six degrees of freedom of an object. Since the base can only move in the \( x \), \( y \), and \( \theta \) direction, only these values are put in the vector. In the future this can be extended to all six degrees of freedom to make the component suitable for a more general purpose.

7.3.5 Custom components used for the control of the base

- PublishOdometry
  Consists of one input port and one output port, runs at a fixed frequency
  To determine the velocity of the robot, the navigation stack in ROS requires that any odometry source publish both a transform and an odometry message over ROS that contains velocity information. That is what this component is for. It derives the position and velocity of the robot based on the encoder data and publishes this to ROS. The reason for the fixed frequency is that there is no need to publish odometry information at 1 kHz over a ROS topic. This value is currently set at 50 Hz.

- BaseSafety
  Consists of multiple input ports and one output port, all the input ports are triggered.
  This component can monitor the values of any vector. In the current setup this component monitors the reference values, the errors and the output voltages of the system. If any of these exceed a value given in the parameters, the component will disable the amplifiers. It actively monitors if it receives new values and if the ports are still connected. Otherwise it will disable the amplifiers as well. One input port is reserved for a reset signal. If a true boolean value is received over this port, the component will enable the amplifiers again.

- BaseReset
  Triggered component, it consists of one input port and one output port.
  This component is designed to be able to reset the controllers in case the BaseSafety component switched off the amplifiers. It receives a reset value from a ROS topic, this means the reset can be done by hand or by a higher level executioner. If a reset value is received it will send a reset signal to the BaseSafety to enable the amplifiers again and it will send the current position to the integrator that integrates the velocity to a position. This will automatically put the errors to zero, since the reference is equal to the position, to prevent the BaseSafety to disable the amplifiers right away.
  Another feature of this component is that it also sends the current position to the integrator if the emergency button is pressed. By doing this, the robot can be moved by hand without the BaseSafety acting on a too large error. This way the robot can be relocated and resume driving if the emergency button is released again.

7.4 Implementation

All the standard components have been put in one ROS package in the tue-ros-pkg repository called orocos_components. The custom components like the BaseReset can be put in separate packages. In this case the amigo_base_controller package. By letting this package depend on the orocos_components package, all the components can be loaded by simply including the amigo_base_controller in the deployment file.
This deployment file, which specifies the design of your controller, can be loaded into the deployer component. The correct components will be loaded and connected to each other. This means that a controller can be designed and implemented without knowledge of C++. However, since the library is still not mature, several components still have to be custom build. Currently the base uses three custom components: one which publishes the odometry of the base, one that monitors the control signals for safety reasons and one component that enables a reset of the safety component.

These components can be improved by sending diagnostic information to ROS. If the safety of the controllers becomes active for whatever reason described in Section 7.3.5, there is no easy way of knowing what went wrong. Easy access to this information will ease troubleshooting on the robot if it halts and the cause of these errors can be resolved fast.

Typically the first component in a control loop, i.e. the one that reads the encoder data runs at a fixed frequency. This determines the update frequency of the whole update hook since the subsequent components have trigger ports and thus will run at the same frequency.

A list of all the components in the orocos_components package is available in Appendix C with no further information given since these components are not used for the control of the base. The names should be self explanatory.

### 7.5 Visualization

The components are loaded and connected in a deployer using an OROCOS Program Script (.ops) file. To check all the connections between the components a Python script has been written that creates a graphical interpretation of the connections between the components with one simple command. Figure 7.3 and 8.8 are generated from such .ops files.
Chapter 8

Low-level control of the base

In Chapter 2 the design of the base is discussed. The robot has been equipped with four omni wheels to drive around. In order to create the controllers for these wheels, the components described in Chapter 7 are used.

8.1 General layout

The robot has four omni wheels giving it three degrees of freedom. The inverse kinematics are used to analyse the kinematic behaviour of the platform. It relates the angular velocity of the wheels to the velocity of the platform in $x$, $y$, and $\theta$ direction.

$$
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix} = \frac{1}{4 r_w} \begin{bmatrix}
\sqrt{2} & -\sqrt{2} & -\sqrt{2} & \sqrt{2} \\
\frac{1}{r_c} & \frac{1}{r_c} & -\frac{1}{r_c} & -\frac{1}{r_c}
\end{bmatrix}
\begin{bmatrix}
\dot{\omega}_1 \\
\dot{\omega}_2 \\
\dot{\omega}_3 \\
\dot{\omega}_4
\end{bmatrix}
$$

(8.1)

$$
\begin{bmatrix}
\dot{\omega}_1 \\
\dot{\omega}_2 \\
\dot{\omega}_3 \\
\dot{\omega}_4
\end{bmatrix} = \frac{1}{r_w} \begin{bmatrix}
\frac{1}{2} \sqrt{2} & \frac{1}{2} \sqrt{2} & r_c \\
-\frac{1}{2} \sqrt{2} & \frac{1}{2} \sqrt{2} & r_c \\
-\frac{1}{2} \sqrt{2} & -\frac{1}{2} \sqrt{2} & r_c \\
\frac{1}{2} \sqrt{2} & -\frac{1}{2} \sqrt{2} & r_c
\end{bmatrix}
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix}
$$

(8.2)

The decoupling matrix $T_v$ is given in Equation 8.1, with the definitions given in Figure 8.1. The same can be done to convert the torques of the wheels to the total force of the robot:
Figure 8.1: Layout of the motion platform indicating the frames

\[
\begin{bmatrix}
F_x \\
F_y \\
M_\theta
\end{bmatrix} = \frac{1}{r_w} \begin{bmatrix}
\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\
r_c & r_c & r_c & r_c
\end{bmatrix} \begin{bmatrix}
\tau_1 \\
\tau_2 \\
\tau_3 \\
\tau_4
\end{bmatrix}
\]

\[
\begin{bmatrix}
\tau_1 \\
\tau_2 \\
\tau_3 \\
\tau_4
\end{bmatrix} = \frac{1}{4r_w} \begin{bmatrix}
\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{1}{r_c} \\
-\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{1}{r_c} \\
-\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & \frac{1}{r_c}
\end{bmatrix} \begin{bmatrix}
T_x \\
F_y \\
M_\theta
\end{bmatrix}
\]

The desired $\dot{x}$, $\dot{y}$, and $\dot{\theta}$ are retrieved from a ROS topic by the ReadTwistMsg component as described in Section 7.3.4. The velocities received over this topic are usually generated by either a teleoperation component as described in Section 6.1.1 or through the navigation stack available in ROS. This stack accepts information from odometry and sensor streams and outputs velocity commands to send to a mobile base. It continuously tracks the velocity and position of the robot with the desired ones and adapts the reference values for the base accordingly. In this way it acts as a cascaded controller over the low-level controllers of the wheels [22].

The task remaining is to create controllers to let the robot follow these reference values. For this, first an identification experiment has been done and next controllers have been designed for the wheels.

### 8.2 Identification

The identification of the system is done by injecting noise after the controller output. This noise consists of a random number with a maximum magnitude of 0.1 Nm. All four wheels have a weak controller with a gain of 0.05 and a lead-lag filter with the zero at 5 Hz and the pole at 20 Hz. The noise is injected onto one wheel at a time. This experiment is repeated four times to cover all the wheels. These measurements have been done while the robot was standing still. One series of experiments has been done while the robot was put on blocks, so the wheels where free in the
air. At the second series of experiments, the robot was standing with all wheels on the floor. The floor was a very compliant floor, so no slip occurred during the experiments.

The coherence of the measurements is given in Figure 8.2(a) and 8.2(b). It can be seen that the coherence is nearly 1 for all wheels in the case of the robot being on the floor. Since the wheels hardly move and there is a weak controller applied, the control output is negligible. So the input to the plant equals the noise, hence the coherence is 1. If the wheels are from the floor, they move more and thus is the controller output more significant and thus the coherence lower. The experiments might be improved by adding a reference signal to the controllers. This has not been done in this series of experiments because the controllers were to weak to follow a reference if the robot was on the floor. Unfortunately there was no opportunity to follow up on these experiments.

In Figures 8.3(a) and 8.3(b) the process sensitivity can be seen. Since the location of the wheels is irrelevant in these experiments the wheels are referred to by the colour in the figures. It can be seen that the cyan wheel has in both experiments the lowest coherence. It has not been investigated why this is the case.

From the phase it can be concluded that there is a 5 ms delay in the system. The cause for this has not been thoroughly investigated. However a 1 ms delay can already be explained by the setup of the Soen component. This component does one read/write cycle every millisecond since it runs at a fixed frequency. This behaviour can be improved by providing the component with trigger ports for the outgoing signals. This way the control action can take place on the moment it is calculated, decreasing delays as a result.

The 0-slope until 40 Hz in Figure 8.3(b) can be explained by the floor acting as an extra stiffness to the system, in this case the wheels.

By dividing the process sensitivity by the sensitivity the open loop behaviour can be found. However, the coherence is very low in the off diagonal terms, see Appendix D. This indicates that there is little interaction between the wheels. The advantage of this is that a diagonal controller can be used. The open loop Figures are displayed in 8.4(a) and 8.4(b).
Figure 8.3: Process sensitivity and coherence of the four wheels, the off diagonal terms of the wheels on the floor can be found in Appendix D

Figure 8.4: The open loop of the four wheels
8.3 Controller methods

8.3.1 Velocity control versus position control

In the Section 8.1 it is stated that the reference signals for the controllers are given in velocity. The control of the base can either be done with velocity control or with position control. Since the reference is given in velocity, it makes sense to use velocity control. Regarding the safety aspect it has the advantage that the error of the control loop is bounded by the amplitude of the reference under normal circumstances. For example if the robot collides, the velocity will be zero, so the error is equal to the reference. With position control, the reference would continue to increase and so would the error. Of course this is assuming there are no other safety features implemented in the robot.

However, this natural bounding of the error means that the control gain has to be sufficient to get the robot moving within this error range. This control approach has been implemented. On a compliant floor where the wheels don not have any slip it worked, however the performance has not been determined. The problem however was on a non compliant floor. If one wheel lost contact with the floor, the system became unstable due to the high gain. Furthermore on very low speeds, the differentiating action on the encoder signal created a lot of noise on the feedback signal, but this can be resolved by using suitable filters. Another disadvantage is that there will always be an error, so the exact velocity is never reached. To prevent this an integrating action could be added, but this would lead effectively to position control.

To summarize, there are three reasons not to use velocity control. These are instability of the robot in case of a wheel losing contact with the floor, a lot of noise on the feedback signal due to the differentiating action on the encoder signals and the fact that there will always be an error in the velocity. For these reasons a choice has been made to integrate the reference velocity to a reference position and to apply position control on this signal.

8.3.2 Coupled versus decoupled control

As mentioned before, the robot has four omni wheels and three degrees of freedom: \( x \), \( y \), and \( \theta \). To control the robot in these directions the control can be decoupled like in Figure 8.5(a).

First control on a decoupled systems has been attempted. On the compliant floor where the identification was done, the decoupled control performed well. However, if one wheel lost grip, it starts spinning fast because it got the same torque applied as the rest of the wheels if moving in one direction. One spinning wheel leads to a rapid speed change in \( x \), \( y \) and \( \theta \) direction if calculated according to Equation 8.1. This could lead to instability of the whole system.

This behaviour can be prevented by detecting the slipping wheel(s) and exclude it temporarily.
from the equation. However, no suitable method was found to detect a slipping wheel or, just as important, detect when a wheel stops slipping.

To illustrate the problem of a decoupled system with slipping wheels take the following example: If a reference velocity in θ direction is given, and two opposite wheels start slipping, the resulting ˙θ is identical to the reference value due to the controllers and Equation 8.1. So despite the controllers work well on the decoupled system, the robot does not follow the reference. For this reason the controllers are applied per wheel. This does not guarantee the following of the reference in the case of two slipping wheels, but it does with one slipping wheel. Furthermore, it will always set the robot in motion, making it unlikely for two wheels to continuously slip.

Since, as mentioned before, no suitable method was found to detect slip, the controllers have to be stable for every situation, with or without a slipping wheel. Therefore the controllers are designed on both situations.

8.4 Controller implementation

In Figure 8.6(a) the measured plant of each of the four wheels is displayed. The -2 slopes are quite similar. Only the -1 slope varies in magnitude.

In order to get enough phase margin a lead-filter with the zero at 5 Hz and the pole at 20 Hz has been added to get a phase-margin of more than 30° in the region up until 25 Hz. A low-pass filter with the pole at 250 Hz and a damping of 0.7 has been added to reduce the noise at higher frequencies. With a gain of 0.5 the system reaches a bandwidth of about 5 Hz if the wheels are from the floor.

The stability of the systems is shown in the Nyquist diagrams in Figure 8.7.

The layout of the complete control structure is shown in Figure 8.8. Since the base is controlled in position, the signal is first integrated to achieve a global position in x, y, and θ direction. Next, these positions are decoupled to get the wheel positions in radians. From this signal the error is calculated and a standard control-loop remains with a gain, lead-lag and a second order low-pass. In the end the signal is outputted to the AnalogOuts component. This component outputs the values of all the controllers working on EtherCAT to the Soem component.

8.4.1 Feed-forward

The feed-forward is done in the coupled form. This means the feed-forward acts on the three input velocities ˙x, ˙y and ˙θ. After multiplying the reference with a feed-forward gain, the values are decoupled and added to the control signal. The reason to do the feed-forward in the decoupled form is that the feed-forward is different for the translation and rotation of the robot. There is currently only feed-forward on the velocity signal present. So no friction or acceleration feed-forward. The gain of the feed-forward on the velocity has been determined by letting the robot drive at two constant velocities while logging the controller output. From the mean of this output during the constant velocity the feed-forward gains have been determined.
Figure 8.6: Plant of the four wheels

(a) Free turning
(b) On the floor (diagonal terms)

Figure 8.7: Nyquist diagrams of the closed loop wheels

(a) Free turning
(b) On the floor (diagonal terms)
Figure 8.8: Diagram of base_controller.ops
This diagram is created with the visualization tool described in Section 7.5
A detailed description of all the components used here can be found in Section 7.3.
Chapter 9

Conclusions & Recommendations

A mechanical design of the service robot is presented. It drives around using omni wheels as a full-holonomic platform, capable of driving over small obstacles. The design of the base is based on the new design of the Tech United TURTLE. This platform has been equipped with two Philips Experimental Robotic Arms. To provide the arms with enough reach a spindle mechanism has been added to give the torso the ability to move vertically. Now the robot is able to pick objects from the floor in the lowest position although only within very close range. In the highest position it can pick up objects from tables.

The robot uses the Robotic Operating System as middle-ware for the processing of sensor data and decision making. ROS has already a lot of basic functionality for mobile manipulators available.

On the low-level software, the real-time OROCOS architecture, has been chosen and used for the controllers in the robot. A small library of several components has been build to design controllers in an easy manner. It consists of basic mathematical operations, controller elements such as lead-lag and low-pass filters and elements to communicate with the actual hardware and higher level software.

Controllers have been build for the base to control the wheels. The wheels are each individually controlled on their position. The wheels are stable even if they loose contact with the floor, when driving over an uneven surface.

The name the robot is given during the project is AMIGO which stands for Autonomous Mate for IntelliGent Operations.

9.1 Achievements

Two successful demonstrators for RoboEarth have been performed using AMIGO as a demonstrator [2]. In the first demonstrator, AMIGO was asked to serve a drink to a patient in a mock-up hospital room using the knowledge within the RoboEarth database. In a second demonstrator, AMIGO downloaded the stored articulation model and its parameters of a cupboard from the RoboEarth database. Using this knowledge, it was able to generate an open-loop trajectory and successfully open the cupboards door. Furthermore is participated twice in a RoboCup @Home competition. In March 2011 AMIGO achieved a 6th place at the RoboCup German Open 2011 and in July 2011 AMIGO became 14th at the RoboCup World Cup 2011.

Based on these achievements the main goal of the project is achieved, a fully functioning service robot has been created that operates on the ROS framework.
9.2 Recommendations

9.3 Improving the grip of the omni wheels

The robot drives much smoother on a compliant surface than on a hard uneven underground. Furthermore on a compliant surface no slip occurs if there are no obstacles present. To get this behaviour also on hard surfaces, the compliance can be built into the omni wheels. This can be done by replacing the hard rubber rings by a soft rubber around the whole circumference like in Figure 9.1.

![Figure 9.1: New roller proposal for the omni wheels](image)

Improve the OROCOS Soem wrapper

The current implementation of the communication of the hardware is with the SoemMaster component developed by K.U. Leuven. This component runs at a fixed frequency. With a sample rate of 1 kHz this means there is always a one millisecond delay since the reading and writing takes place at the same time. Some effort might be put on improving this by providing the component with trigger ports for the outgoing signals. This way the control action can take place on the moment it is calculated, decreasing delays as a result. This will definitely improve controllers running at a lower frequency rate.

Improvement on the OROCOS components

In Chapter 7 several components have been designed to facilitate basic mathematical operations. There is still room for improvement in these components. For example performance of the components and the system as a whole has not been investigated yet. Also the components can be made more universal. For example there is an addition and a subtraction component with only a sign as difference in the code. Furthermore they can currently only cope with two signals since more has not been necessary yet. A possible configurable parameter in such a component could be a list of signs for every required port. So a parameter like ‘++-’ would create a component with three ports. It would add the two vectors on the first two ports and subtract the vector of the last port.

Furthermore the control components could be redesigned to act as a wrapper for the control library designed by B. Mrkajić [20].

Replace USB boards of the arms by EtherCAT

The arms of the robot turned out to be quite unreliable. The USB communication is sensitive to electromagnetic fields inside the robot. For example the clamping of the spindle break results in USB failures of the arm. Furthermore drivers sometimes randomly need to be reloaded for the
arm to operate again. These problems do not occur with the hardware driven by the EtherCAT stack. Furthermore the USB standard is not designed for real-time performance, so on a strained computer the USB bus might not get enough priority. This might result in unstable controllers. To achieve some conformity and modularity in the robot, it would be an improvement to design custom EtherCAT boards for the arms, taking over the functionality of the USB boards. This would also improve the uniformity of the communication with the hardware, since all actuators are then controlled by EtherCAT.

**Investigate the 5 ms time delay in the control loop**

In Section 8.2 it is shown that there is a 5 ms delay in the system. To improve performance it is usefull to investigate the source of this delay.

**Adding diagnostics of the controllers**

Already a beginning has been made in making diagnostic information about the hardware available within the ROS environment. For example the battery voltages can be read back and warnings are given if it is to low. This could be extended by including the status of the controllers. If the safety of the controllers becomes active for whatever reason described in Section 7.3.5, there is no easy way of knowing what went wrong. Easy access to this information will ease troubleshooting on the robot if it halts and the cause of these errors can be resolved fast.
Appendix A

Location of the center of mass

The location of the center of mass $R$ is calculated using the location $r_i$ and the size $m_i$ of the masses:

$$
R = \frac{\sum m_i r_i^2}{\sum m_i} \quad (A.1)
$$

The location of the masses of the arms are taken from the documentation of the arms, the rest of the values are estimated. The values used in Table A.1 are based on the robot being in the highest position with the arms stretched forward holding each a mass of 1.5 kg.

Table A.1: Location of the center of mass with respect to the center of robot in $x$ and $y$ direction and the bottom of the wheels in $z$ direction.

<table>
<thead>
<tr>
<th>Element</th>
<th>$x$ [m]</th>
<th>$y$ [m]</th>
<th>$z$ [m]</th>
<th>Mass [kg]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoulder left</td>
<td>0.0</td>
<td>0.15</td>
<td>1.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Upper arm left</td>
<td>0.14</td>
<td>0.17</td>
<td>1.2</td>
<td>2.9</td>
</tr>
<tr>
<td>Lower arm left</td>
<td>0.41</td>
<td>0.17</td>
<td>1.2</td>
<td>0.8</td>
</tr>
<tr>
<td>Gripper left</td>
<td>0.7</td>
<td>0.17</td>
<td>1.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Shoulder right</td>
<td>0.0</td>
<td>-0.15</td>
<td>1.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Upper arm right</td>
<td>0.14</td>
<td>-0.17</td>
<td>1.2</td>
<td>2.9</td>
</tr>
<tr>
<td>Lower arm right</td>
<td>0.41</td>
<td>-0.17</td>
<td>1.2</td>
<td>0.8</td>
</tr>
<tr>
<td>Gripper right</td>
<td>0.7</td>
<td>-0.17</td>
<td>1.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Lifted load</td>
<td>730</td>
<td>0</td>
<td>1.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Upper body</td>
<td>0.0</td>
<td>0.0</td>
<td>1.2</td>
<td>10</td>
</tr>
<tr>
<td>Base Frame</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>40</td>
</tr>
<tr>
<td>Center of mass</td>
<td>0.0</td>
<td>0.06</td>
<td>0.60</td>
<td>66.8</td>
</tr>
</tbody>
</table>
Appendix B

Required power for spindle motor

The formulas are acquired from NSK catalogue E3161b page B523.

The parameters used are:

- $F_{a_0} = 0$ preload of spindle nut [N]
- $l = 0.4$ lead (pitch) [cm]
- $dr = 0.82$ Root diameter of the spindle [cm]
- $\beta = \tan(l/(0.25 \times \pi \times dr^2))$ lead angle [deg]
- $F_a = 26.8 \times 9.81$ Axial load (force needed to lift the robot [N]
- $\nu_1 = 0.9$ Normal efficiency (given in catalogue)
- $T_u = 1.9$ Friction torque of the support bearings [N cm] (given in catalogue)

Calculating the drive torque $T_m$:

- $K = 0.05/\sqrt{\tan(\beta)}$ Torque coefficient of ball screw
- $T_{p_0} = K \times F_{a_0} \times l/(2 \times \pi)$ Dynamic preloaded torque [N cm]
- $T_{p_{\text{max}}} = 2.5 \times T_{p_0}$ Starting friction torque [N cm]
- $T_{p_{\text{cat}}} = 1$ given Dynamic friction torque from catalogue [N cm]
- $T_a = F_a \times l/(2 \times \pi \times \nu_1)$ Torque it takes to translate the nut over the spindle [N cm]
- $T_m = (T_a + T_{p_{\text{cat}}} + T_u)$ Required motor torque [N cm]

$$T_m = 21.5$$

Motor requirements:

- $t_l = 5$ time [s] required to lift the robot
- $\text{stroke} = 40$ Distance nut has to travel over the spindle [cm]
- $\text{rpm}_{\text{spindle}} = (\text{stroke}/(l \times 10))/tl \times 60$ rotation speed of the spindle to raise robot in $t_l$ [RPM]

$$\text{rpm}_{\text{spindle}} = 120$$
Appendix C

List of all components in the orocos_components package

Abs
AddConstant
Addition
AddNoise
AnalogIns
AnalogOuts
ConstantSignal
Demux
Derivative
Differentiate
DigitalIns
DigitalOuts
DotDivide
DotProduct
DT
EnableDigital
Equals
FilteredData
FirstOrderLowPass
FirstOrderLowPasses
Gain
Gains
GaussianRandomNumber
Integrator
LeadLag
LeadLags
MatrixTransform
Mux
Negate
PD
PID
PIPs
Product
PulseSignal
RampSignal
ReadArmJointsMsgFixed
ReadBattery
ReadEncoder
ReadEncoders
ReadTwistMsg
RealTimeDerivator
ReferenceGenerator
Saturation
SawtoothSignal
SecondOrderLowPass
SecondOrderLowPasses
SensorTorques
Sign
SineWave
SkewedNotch
SkewedNotches
Sqrt
StateSpace
StepSignal
Subtraction
Sum
SumVectors
TrajectoryGenerator
TransferFunction
UniformRandomNumber
VectorConcatenate
WeakIntegrator
WeakIntegrators
WriteArmDataMsg
WriteArmTorquesMsg
ZeroPole
Appendix D

Diagonal terms of the identification

On the next two pages the diagonal terms of the sensitivity and the process sensitivity from the identification described in Section 8.2 are displayed. Since the coherence of the off-diagonal terms is too low, the open loop and plant can not be properly determined and are therefore not included here.
Figure D.1: Sensitivity of the four wheels on the floor, the noise is added to the diagonal terms.

Figure D.2: Coherence of the sensitivity for the four wheels on the floor, the noise is added to the diagonal terms.
Figure D.3: Process sensitivity of the four wheels on the floor, the noise is added to the diagonal terms.

Figure D.4: Coherence of the process sensitivity for the four wheels on the floor, the noise is added to the diagonal terms.
Appendix E: Specification sheet

Computing

Three on-board PCs
Per PC:
- Processor: Dual-Core i5
- 8 GB ram
- 500 GB Hard Drive

Sensors

Base
- Hokuyo UTM-30LX Laser Scanner
Head
- Microsoft Kinect
Arms
- Force sensors per joint
- Force sensor in gripper actuator

Arms

Arm DOFs
- Arm: 4
- Wrist: 3
- Gripper: 1
- Liftable force: 1.5 kg
Arm Link Lengths
- Upper Arm: 320 mm
- Forearm: 280 mm
- Wrist to Gripper Surface: 90 to 180 mm
- Distance between shoulder joints: 219 mm
Arm Range of Motion
- Shoulder yaw: 180°
- Shoulder roll: 90°
- Shoulder pitch: 180°
- Elbow yaw: 210°
- Elbow pitch: 145°
- Wrist Yaw: 90°
- Wrist Pitch: 114°
- Gripper: 90 mm max

Head, Spine & Base

Pan Tilt Head
- Pan: 300°
- Tilt: 120°
Spine
Height Range from Floor to shoulder joints:
- 798 – 1211 mm
Omni Directional Base
- Omniwheels: 4 driven
- Base width & depth: 600 mm
- Speed: 1 m/s

Networks

Gigabit Ethernet
- Cisco Linksys E2000 router
- Gigabit Ethernet switch
EtherCAT Network
- 1 kHz Control to Base Motors
- Trigger signal from Laser Scanner
- Diagnostic signals from robot vitals
- Expandable with further Beckhoff modules
Bibliography


