Control of the Philips Experimental Robotic Arms using EtherCAT

M.J.J. Baeten
CST 2013.079
# Contents

1 Introduction .................................................. 1

2 Hardware ....................................................... 3
   2.1 Philips Experimental Robotic Arm ........................................ 3
   2.2 EtherCAT PCB ................................................. 4
   2.3 Hardware layout ............................................. 6

3 Control loop ..................................................... 7
   3.1 Control loop for USB I/O using Orocos .................................... 7
      3.1.1 The Orocos Toolchain .......................................... 7
      3.1.2 USB I/O .................................................. 7
   3.2 Adaptations for EtherCAT I/O ....................................... 10
      3.2.1 The EtherCAT Protocol ........................................ 10
      3.2.2 EtherCAT I/O ............................................ 10

4 Safety .............................................................. 15
   4.1 Safety Measures in Orocos ........................................... 15
   4.2 Safety on FPGA level ........................................... 16

5 Results ............................................................ 19
   5.1 Step response ................................................ 20
   5.2 Trajectory .................................................... 22
   5.3 Feedforward .................................................. 26

6 Conclusions and Recommendations .................................. 29
   6.1 Conclusions .................................................. 29
   6.2 Recommendations ........................................... 29
Chapter 1

Introduction

With the growing population of elderly in the society and increasing costs of care there is a need for robotic aid in health care. AMIGO [1], the care robot of the Mechanical Engineering faculty of the Technical University of Eindhoven, is used in various research projects: Bobbie [2], Robotic Open Platform [3], RoboEarth (R3-Cop) [4]. The Bobbie project aims to stimulate modularity and standardization in robotics research to make exchange of hardware and software easier. AMIGO is also competing with care robots across the world in the RoboCup@Home competition [5]. This competition within the RoboCup was started to stimulate research on care robots.

EtherCAT I/O is currently implemented on AMIGO for the base and spindle together with USB I/O for the robotic arms. Because of standardization and modularity, EtherCAT I/O for all degrees of freedom of AMIGO is desired. Therefore, for AMIGO, AMIGO2 and the Deboflex project, deskinning of pork legs by autonomic robotic arms, EtherCAT I/O boards were developed by the Technical University of Eindhoven and Neways.

The goal of this project is to implement the newly developed printed circuit board (PCB) which make use of the EtherCAT protocol on the Philips Experimental Robotic Arms of AMIGO. Since these EtherCAT boards are also going to be used in AMIGO2 this project is a test case for the use of these boards. The replacement of the USB I/O by EtherCAT I/O also solves the problem that the USB boards have with static electricity during the winter. A subgoal of this project is making the control structure generic by using standard configurable components configured by a configuration file.

The hardware is the subject of Chapter 2, where the layout of the arm, and concepts of the EtherCAT PCB are covered. The control structure is discussed in Chapter 3 containing the use of Orocos, and the differences between USB I/O and EtherCAT I/O. In Chapter 4 the safety measures on controller and FPGA level is the subject and in the last chapter, the results are shown. This chapter contains step responses, trajectories and feedforward compensation.
Chapter 2

Hardware

This chapter contains a description of the hardware of the PERA on EtherCAT. The first section provides a description of the Philips experimental robotic arms. In the second section, the EtherCAT PCB’s developed by Neways are discussed and the last section provides a detailed layout of the connections of all sensors, encoders and motors.

2.1 Philips Experimental Robotic Arm

The anthropomorphic Philips Experimental Robotic Arm has humanlike dimensions and degrees of freedom (DOF). The DOF of the PERA are shown in Figure 2.1. The PERA consists of three differential drives and a rotational drive. Furthermore, the gripper is controlled by a motor in the lower arm which pulls on a cable to close the gripper. Equation 2.1 shows the Degrees of Freedom defined for both arms, where $S$, $E$, $W$ and $G$ represent respectively the shoulder, elbow, wrist and gripper. In Table 2.1 the degrees of freedom controlled by each motor are shown.

$$
[q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8]^T = [R_S \ P_S \ Y_S \ P_E \ Y_E \ P_W \ Y_W \ GR]^T \quad (2.1)
$$

<table>
<thead>
<tr>
<th>Motor</th>
<th>Degree of Freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TS_1 + TS_2$</td>
<td>$q_1 + q_2$</td>
</tr>
<tr>
<td>$TS_3$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$TE_4 + TE_5$</td>
<td>$q_4 + q_5$</td>
</tr>
<tr>
<td>$TW_6 + TW_7$</td>
<td>$q_6 + q_7$</td>
</tr>
<tr>
<td>$TG$</td>
<td>$q_8$</td>
</tr>
</tbody>
</table>

Table 2.1: Degrees of Freedom controlled by each motor
2.2 EtherCAT PCB

For AMIGO and the Deboflex project a six-layer printed circuit board (PCB) was developed by Neways. The PCB is developed for the EtherCAT protocol which will be discussed in Chapter 3. This EtherCAT PCB is shown in Figure 2.2. This print is designed to control three Degrees of Freedom and therefore consists of three motor outputs, three encoder inputs, six sensor inputs and a spare I/O connector. The spare I/O connector is not used in the arm but could be used to control motors with an external amplifier. Furthermore there are two connectors, an eBus input and output, for the communication with the EPC coupler print and the other slaves. The slaves are connected in series by eBus cables, two twisted pairs, to the EPC coupler print. The EPC coupler print converts the telegrams that passes from Ethernet 100BASE-TX to E-bus signal representation. The motor outputs are Pulse Width Modulation outputs where a fast switching on/off signal is sent to the motors. The average current and voltage sent to the motors therefore can be controlled by the on/off times. The PWM signal has a frequency of $15[kHz]$ and therefore has no effect on the load. The six sensor inputs are analog inputs that measure and digitalize a voltage between 0 and 5V. This layout is further shown in Section 2.3.

The print has two power inputs, the 24V and the 24V Ctrl. The first is connected with the emergency switch. When the emergency button is pressed, the relays disable the power to this input and the arm motors immediately stop. The 24V Ctrl, however, is directly connected to the batteries of AMIGO and can therefore only be disconnected by the On/Off switch of AMIGO. The print can furthermore be adapted so that the Pulse Width Modulation signal (PWM) is send over the encoder cables. This is needed for the differential gear of the wrist since these two motors are of a type that require power via the encoder cables. To protect the encoder cables for high currents, a circuit on the print that limits the power sent to the motors to $3[A]$ can be activated by removing two resistors from the print. The amplifiers of the print are H-bridges which are integrated circuits on the back of the print. These components can heat up easily with a high workload of the arms and therefore there is a possibility to attach a heat sink on the back of the print.

Figure 2.1: Degrees of Freedom of the Philips Experimental Robotic Arm, Willems
2.2. **ETHERCAT PCB**

![Figure 2.2: EtherCAT PCB](image-url)
2.3 Hardware layout

In this section the layout of the new control structure is shown. In Figure 2.3, the complete in- and output on the Ethernet port of AMIGO 2 is visualized. This PC does the complete in- and output of the robot except the visual sensors, i.e., the laser range finders and the Kinect cameras.

The first Beckhoff stack in AMIGO consists of the encoder modules of the wheels, slaves 1002-1005, and the spindle encoder module, slave 1006. Slave 1007 is a spare encoder module. The second Beckhoff stack has two Digital Input modules (EL1008) and two Digital output modules (EL2008). The first digital input module is used for the four fuses and the two emergency buttons and the second input module is used for the spindle endstop. The first digital output module is used to connect the lights of AMIGO. Furthermore, this stack consists of an eight channel analog output module (EL4038) for the wheels and spindle output, an analog input module (EL4038) for the battery value, and a RS485 module (EL6022) for the dynamixels of the head.

To make clear how the sensors, encoders and motors are connected to the PCB’s a distinction has to be made in actuator space and joint space. Inevitably, the motors and encoders are in actuator space. For the force and absolute position sensors it is not that trivial. The force sensors are placed directly on the motors and therefore in actuator space. The absolute position sensors, however, are placed directly at the joints and therefore in joint space. Therefore the force sensors and encoders have to be mapped to joint space to be used in the control loop. The motor outputs have to be mapped from joint space to actuator space. This mapping is shown in Chapter 3.
Chapter 3

Control loop

In this chapter a description is provided of the control loop of the Philips Experimental Robotic Arms on AMIGO. In the first section the control loop will be discussed and in the second section, a deeper analysis of the differences with the old control loop is done.

3.1 Control loop for USB I/O using Orosco

The Robot Operating System (ROS) [7] is used for the higher level software but Orosco is used for the real time control since this is not possible within ROS. To get an understanding of the existing control loop, first the Orosco environment is discussed.

3.1.1 The Orosco Toolchain

Orosco, which stands for Open Robot Control Software [8], is a collection of C++ libraries for robot control developed by the K.U. Leuven. Within Orosco there are three main projects, the Kinematics and Dynamics Library, the Bayesian Filtering Library and the Orosco Toolchain. The Orosco Toolchain, is the project that is used for the control loop. This toolchain is used to create a real-time control loop with modular and run-time configurable software components. Every Orosco application consists of Orosco components that communicate with each other over ports. Components can be written by the user or can be generic components from the Orosco Component library. A deployer is started which runs an Orosco Program Script (.ops) that loads, configures, connects, and starts components. In the configuration step, certain parameters are loaded. For example in the case of a matrix multiplication component, the matrix size and all its values are loaded. The ports connecting the components can either be EventPorts or normal Ports. EventPorts trigger the component, where normal ports do not. Every component should therefore be triggered by one or more EventPorts or run on a fixed frequency. The state machine as shown in Figure 3.1 shows the state in which an Orosco component can be. The Init state is the state in which the component is during the creation of the component. After that it enters the stopped or PreOperational state, where the latter requires a configure() call to go to the stopped state. For each API function, a hook is available in which the user can put code. From the stopped state it can enter the running state via the start() call with the startHook() and similarly go back to the stopped state with the stop() call and stopHook(). The code that is executed every time the component is triggered is stated in the updateHook().

3.1.2 USB I/O

A visualization of the existing control loop is shown in Figure 3.2 where each circle represents an Orosco component and each arrow represents a port. The dashed lines represent the connections with ROS topics, called ROS streams. The higher level software such as planning, navigation etc., runs in ROS and communicate with the control loop using these ROS topics starting with a “/” that are shown besides the ROS stream ports. Furthermore, the red arrows indicate the negative
feedback loop, the green arrows the reference set point and the blue arrows the output. The ports in the negative feedback loop, the red arrows, are all EventPorts except the inport of the PERA_IO component which runs on the loop frequency. This component therefore triggers all components subsequently.

Some components will be discussed to explain the concepts in the existing control loop. The first component is the Supervisor which has three functions. First, the Supervisor monitors errors, controller saturation values and the emergency button status. Second, the Supervisor component generates the joint references during the homing procedure. The last function of the Supervisor is sending a status message to the dashboard. The SensorTorques component calculates forces from the measured voltages received from the PERA_IO component. Another part of the control loop is the feedforward to compensate for gravity. Since gravity is a constant and known force, it can be compensated by feedforward. Three components, together forming the gravity compensation feedforward, calculate this feedforward control signal and add it to the negative feedback control signal before it goes into the plant. The real time derivator (RTD) calculates the velocities of the joints which at the moment are not used. The Diagnostics component sends diagnostics messages to ROS such that certain debugging information is available in ROS as well. Lastly the OutputLimiter is a saturation component which makes sure that the torques sent to the amplifiers are not higher than a certain value. This was initially implemented to avoid the controller from sending values outside the 16-bit range, however the bounds were tuned down to limit the maximum output of the motors as a safety precaution.
3.1. CONTROL LOOP FOR USB I/O USING OROCOS

Figure 3.2: Control loop USB
3.2 Adaptations for EtherCAT I/O

3.2.1 The EtherCAT Protocol

The EtherCAT protocol, which stands for Ethernet for Control Automation Processes was released by the German company Beckhoff\cite{9} in 2003. Beckhoff founded EtherCAT Technology Group (ETG) and donated the EtherCAT rights in 2004 to ETG to promote the EtherCAT protocol. The most significant property of EtherCAT is that it uses "Processing on the fly". This means that before the data in a node is processed it is already sent to the next node which creates a very fast network suitable for real time control. Since a high bandwidth utilization is reached while still running on low cost Ethernet components makes it an in-expensive high performance system.

The Simple Open EtherCAT Master library (SOEM) is used as a master device as is already used for the base and spindle. SOEM is developed by Arthur Ketels, and the K.U.Leuven added SOEM to Orocos. The Simple Open EtherCAT Master makes use of a number of states in which the slaves can be. These states are Init, Pre-op, Safe-Op, and Op. Two different kinds of process data can be sent from the master to the slaves, mailbox communication and process data communication. Mailbox communication is used for configuration of process data, configuration of device specific parameters, and diagnosis information of EtherCAT slaves. Process data communication is used for sending the actual process data. In the Initialization state, the first state after booting, no process data or mailbox communication is possible and in this state the master initializes the sync manager channels 0 and 1 for mailbox communication. During the transition from Init to Pre-op, the second state after booting, the master checks if the mailbox is initialized correctly. In the Pre-op state, mailbox communication is possible but no process data communication. In the Pre-op, the sync manager channels are initialized for process data. In the transition from Pre-op to Safe-op this initialization is checked. Now in the Safe-op mailbox communication and process data communication is possible with the only limitation that the slave output is still in safe state. The last step when booting is from the Safe-op to the Op state, where the master checks if at least one valid process data package was sent. In the final Op state, mailbox communication and process data communication is possible. There is also a fifth state, called boot which can be used to update the slave firmware.

3.2.2 EtherCAT I/O

The most significant change in the control loop is the replacing of the PERA_IO component with the SOEM component. The new control loop is shown in Figure\ref{fig:3.4} and a detailed scheme of the changes in the I/O is shown in Figure\ref{fig:3.3}. The PERA_IO component is one component that handles the complete I/O for the four USB prints where for the EtherCAT slaves this is a litte different. SOEM configures each slave with the soem_armEthercat driver, which is written for these particular EtherCAT slaves. Therefore the components outside the grey dashed area in Figure\ref{fig:3.3} are single components while there are three identical SOEM slaves. An important difference with USB controlled arm is that the I/O makes use of SOEM_beckhoff messages. The three new components are introduced to generate these messages, as done in AnalogOutsPera and to read out the messages as is done in AnalogInsPera and ReadEncoders.

The AnalogInsPera receives six messages, three with voltages (force measurement) and three with positions and sends two vectors, voltage and position. The AnalogOutsPera component splits the PWM vector into three messages to be sent to each slave. The ReadEncoders component takes nine encoder messages and puts them into a single vector. Also this component takes over the calculation from encoder counts to SI units, which was done in the matrix multiplication of the MotorToJoinAngles component. This component therefore now has a more transparent function. The matrix multiplication is shown in Equation\ref{eq:3.1} Note that since an extra DOF can be controlled by the PCB’s the mapping is not square anymore. The last two ports are boolean Ports, the enablePort and the reNull port. The Renull function is transferred from the PERA_IO component to the ReadEncoders component and therefore can be a single port. While the enablePort should be connected to each slave.

The controllers that were tuned for the USB controlled arms are tuned in the time domain and integrating action was added to improve the performance of the controllers. The integrating action
was already implemented in the control loop since it was used for wrist joints only the rest of the integrating action was set to zero. The reason why the integrating action was not added at the time was a safety reason but now it is believed that with a proven and solid supervisor this will not hold anymore and the improvement of the performance is desired. At last, the output limiter is removed from the control loop since the safety precaution it was used for is unnecessary and it can introduce an unwanted non-linear effect into the system.

\[
\begin{bmatrix}
0.5 & -0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.5 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -0.5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 & 1 \\
\end{bmatrix}
\begin{bmatrix}
TS_1 \\
TS_2 \\
obs \\
TS_3 \\
TE_1 \\
TE_2 \\
TW_1 \\
TW_2 \\
TG \\
\end{bmatrix}
= 
\begin{bmatrix}
J_1 \\
J_2 \\
J_3 \\
J_4 \\
J_5 \\
J_6 \\
J_7 \\
J_8 \\
\end{bmatrix}
\] (3.1)

![Diagram](image)

**Figure 3.3:** Slave in- and output for USB and EtherCAT
SOEM Beckhoff driver for the EtherCAT slaves

The SOEM component in the control loop is a generic master component which controls a number of slaves. Each PCB in the arm is a separate slave in the same way as each wheel of the base has its own slave only for the arm every three DOF have one print. For every type of slave that is used a driver is needed that communicates with the slave. For all standard Beckhoff modules drivers are provided by Orocos, part of the soem_beckhoff_drivers package. Since the custom build EtherCAT PCB’s are not standard Beckhoff modules, a new driver was written by Sava Marinkov and Ruud van den Bogaert. The main function of this driver is to extract a data struct from the slave, and to write a data struct to the slave. The struct extracted from the slave is shown in Table 3.1 and the struct written to the slave is shown in Table 3.2. The driver component also puts the extracted data into messages to be sent to the right components.

The smallest suitable data types are used in the struct to minimize data flow. For example, four bytes of data is needed for a float, an unsigned 32-bit integer uses four bytes, an unsigned 16-bit integer two bytes and a unsigned 8-bit integer one byte. The status_register, message_index and all three encoder_angles are unsigned integers so an unsigned 16-bit integer is used for these members. The force and absolute position sensors are digitalized on the print and converted to an unsigned integers since only positive voltages are measured. The data that is sent from the driver to the slaves is a heartbeat, and three PWM values. Since the PWM values lie within the range of -100 and 100, an 8-bit integer is used. Since a small positive integer is enough for the heartbeat, a unsigned 8-bit integer is used.

Once the data is received from the slave, the driver extracts the data and puts it into messages to be sent to the right components. The status_register and the message_index are not sent to an Orocos component. Although both are useful for debugging to analyze data flow, this data is accessible directly from Orocos. The SensorTorques component requires voltages to calculate the torques from the measured voltages of the sensors. Therefore a conversion is done and an AnalogMsg with three voltages is sent to the SensorTorques component. The position measurement is sent without a conversion back to a position in radians since homing of the arm is done using the integer values directly. Therefore a DigitalMsg is sent to the Supervisor with three position measurements. The encoder counts that are calculated by the print from the quadrature encoder signal are put into EncoderMsgs and since these messages can typically only store one encoder value, there are three messages sent over three different ports to the ReadEncoders component. The last function of the driver is writing the PWM values. The value received from the controller is saturated to -100 and 100 and sent to the slave. However before the driver sends a PWM value it checks a boolean enable value received from the supervisor. When this value is not received or false, the driver will send a PWM value of zero to stop the arm.

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned 16-bit integer</td>
<td>status_register</td>
<td>General system status register</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>encoder_angle_1</td>
<td>Encoder Counts of encoder motor 1</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>encoder_angle_2</td>
<td>Encoder Counts of encoder motor 2</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>encoder_angle_3</td>
<td>Encoder Counts of encoder motor 3</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>force_1</td>
<td>ADC value of force sensor input 1</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>force_2</td>
<td>ADC value of position sensor input 1</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>force_3</td>
<td>ADC value of force sensor input 2</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>position_1</td>
<td>ADC value of position sensor input 2</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>position_2</td>
<td>ADC value of force sensor input 3</td>
</tr>
<tr>
<td>unsigned 16-bit integer</td>
<td>position_3</td>
<td>ADC value of position sensor input 3</td>
</tr>
<tr>
<td>unsigned 8-bit integer</td>
<td>message_index</td>
<td>Message index counter</td>
</tr>
</tbody>
</table>
3.2. ADAPTATIONS FOR ETHERCAT I/O

Table 3.2: Struct out_armEthercat

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Member Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit integer</td>
<td>pwm_duty_motor_1</td>
<td>PWM duty cycle for motor 1</td>
</tr>
<tr>
<td>8-bit integer</td>
<td>pwm_duty_motor_2</td>
<td>PWM duty cycle for motor 2</td>
</tr>
<tr>
<td>8-bit integer</td>
<td>pwm_duty_motor_3</td>
<td>PWM duty cycle for motor 3</td>
</tr>
<tr>
<td>unsigned 8-bit integer</td>
<td>heart_beat</td>
<td>Heart beat</td>
</tr>
</tbody>
</table>

PC to Slave communication

A problem encountered during testing is that the slaves are sensitive to losing data packages. For every missed data package a warning is printed to the screen. This problem increases for smaller distances between the power cables from the motors and the eBus cables and for higher loads. The fast switching PWM signals clearly interfere with the data flow on the eBus cables. At this point, the control loop is robust enough to cope with this problem. However, this is a problem to keep an eye on.

A different communication problem occurs during startup. The SOEM master component sends a command to all slaves to go to the next state as discussed in Subsection 3.2.1. After the master sends the command to go to the next state it will wait for 2\[s\] while it continuously checks if the slaves reached the next state. However, if the command send from the master is not received by the slaves, the master will not try again. Therefore it could happen that the slaves did not receive the command sent from the master, while the master is waiting. This is a known issue for SOEM but in some way our slaves are very sensitive for this problem. Therefore, in the SOEM master component a loop is introduced which simply tries again if not all slaves went to the next state.
Figure 3.4: Control loop EtherCAT
Chapter 4

Safety

Safety in robotics is one of the most important aspects since a robot that is not completely safe will never be accepted by people. Therefore, a number of safety measures are taken that switch off AMIGO when certain conditions are met. Most of these safety mechanisms are unchanged but there are also some new safety measures for the arms on EtherCAT. Therefore a complete description of the safety measures for the PERA control using EtherCAT is given in this chapter. In the first section, an overview of the safety measures in the control loop in Orocos is provided and in the second section, the safety measures in the FPGA software are discussed.

4.1 Safety Measures in Orocos

Error threshold check

When one of the servo errors exceeds a certain threshold, the motors of the arm are disabled. For example when a goal is given to the arm where the arm cannot physically go. The manner in which the output to the motors is stopped is slightly different for USB and EtherCAT. The supervisor which monitors the servo errors, sets the boolean parameter enable to false when one of the servo errors exceeded its threshold. The PERA_IO component sends a message to the USB boards to disable the amplifiers while the driver of the EtherCAT boards sends a PWM value of zero to the slaves. Parameters for this safety measure are the error threshold values of each joint.

Maximum saturation time check

The error threshold check is not sufficient when a goal is sent to the arm where the arm cannot physically go but where the arm can reach the desired goal to within the maximum error allowed by the error threshold check. In this case, especially due to the integrating action, the control signal will increase since the arm cannot reach its goal. The maximum saturation time measure checks if the control value exceeds a certain threshold for a certain period. When one of the control values exceeds its threshold longer than the given period the arm will be stopped in the same way as the error threshold check does, by disabling the amplifiers (for USB) or sending an PWM value of zero (for EtherCAT). Parameters for this safety measure are the control signal threshold values of each joint and the maximum period the control value may exceed this threshold.

Infeasible Joint Reference

This safety is implemented newly in the controller since the old check in the supervisor was not working. Therefore a new saturation component is added to the control loop that saturates the reference output values with the minimum and maximum allowed joint limits. This makes sure that once a joint reference that is out of bounds is sent to the ComputeJointErrors component is
clipped to the bounds. This is a different implementation than the, not working, old check that discarded the infeasible joint references and used the last sent values that were not out of bounds.

**Emergency Button**

The emergency button is a safety measure that does not monitor anything itself but when the operator thinks that AMIGO is going to do something undesired the operator can press the emergency button. When this button is pressed, a relays will shut down the power to the amplifiers of the wheels, spindle and both arms. In case of the spindle the brake is switched on to prevent the spindle from dropping due to gravity. The arms do not have such brakes and will therefore drop. The difference for EtherCAT is that the arm will not drop immediately down but will slowly go down. The difference is that the USB prints disconnect the motors when the emergency button is pressed and the EtherCAT boards short circuit the motors. While the emergency button is pressed the measured joint positions are sent to the interpolator. This ensures two things, first once the emergency button is released again, the arm will remain at its position and not go back to the previous reference where the button was pressed. Secondly if this was not done, the servo error check would disable the arm when the arm falls during the time when the emergency button is pressed.

**4.2 Safety on FPGA level**

Some safety features should be implemented on the lowest level, the FPGA. When for example the communication with the boards is lost, safety measures on a higher level can not reach the slaves anymore. Three implemented safety measures are a Emergency button detection, heartbeat signal and a Slave state check.

**Emergency button detection**

When the emergency button is pressed, the voltage on the 24V input drops to zero to stop the output to the motors. When the button is released again it is undesired that the motors immediately continue with an old PWM value. Therefore when the emergency button is pressed, the FPGA will write a zero PWM signal to the motors. This is done as soon as the voltage of the 24V drops below a certain threshold.

**Heartbeat**

When the communication with SOEM is lost, the arm should stop immediately. This is implemented by sending a heartbeat, an 8-bit integer, from Orocos to the slaves. The 8-bit integer changes every time a new value is sent to the slaves. The slave checks if the value received from Orocos is different from the last received heartbeat. If this heartbeat does not change for more than 5[msec], the slave will sent a PWM value of zero to stop the arm. This is not a fatal error since when the communication is restored, the slave will sent the new received PWM values.

For testing purposes the heartbeat check and the emergency button detection can be switched on and off by the heartbeat. The most significant bit of the heartbeat is used to switch off the heartbeat check in the FPGA. The second most significant bit is used to switch off the Emergency button detection measurement. Therefore in the soem armEthercat driver a heartbeat is generated that runs from 0 to 63. In Table 4.1 for all values of the heartbeat the resulted behavior of the FPGA software is shown.

**Slave state check**

Before a PWM signal is sent to the H-bridges on the print a check is performed on the FPGA to make sure that the slave is in the Operational state. This check is an extra safety measure to prevent the arm from uncontrolled motion since only in the Operational state the controller should be able to sent power to the motors.
### Table 4.1: Enabling of Heartbeat and Emergency button detection

<table>
<thead>
<tr>
<th>Bits</th>
<th>Heartbeat check</th>
<th>Emergency button detection check</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-63</td>
<td>00XXXXXX</td>
<td>Enabled</td>
</tr>
<tr>
<td>64-127</td>
<td>01XXXXXX</td>
<td>Enabled</td>
</tr>
<tr>
<td>128-191</td>
<td>10XXXXXX</td>
<td>Disabled</td>
</tr>
<tr>
<td>192-255</td>
<td>11XXXXXX</td>
<td>Disabled</td>
</tr>
</tbody>
</table>
Chapter 5

Results

In this chapter the controllers are evaluated to indicate the differences in the control on EtherCAT and USB. While there are no fundamental differences between control using either the EtherCAT protocol or the USB protocol there are some practical differences. For example during the transformation from USB to EtherCAT, integrators were added to the control loop which were not present for the control on USB. Also the gravity compensation was not tuned for the right arm at the start of the project. At last the sampling frequency was increased from $250 \text{[Hz]}$ to $1 \text{[kHz]}$, since the other EtherCAT components also run on $1 \text{[kHz]}$. In the first section the controllers are compared with a step response and in Section 5.2 a trajectory is fed to all joints to see the performance of the arm in normal operation. In Section 5.3 feedforward is added to the arm to improve the performance of the arm. In Table 5.1 the parameters for both USB control and EtherCAT are shown. The biggest change can be seen in the gain since for EtherCAT control, a PWM value with a maximum value of 100 is sent into the plant. For USB control however, The maximum values to be sent to the slaves were of much larger magnitude. The second change is the addition of the integrator for all joints. Where for the USB arm, contain integrating action. Note that in our software very small values were used since an integrator with a zero at $0 \text{Hz}$ is in the current Orocos integrator component not possible.

<table>
<thead>
<tr>
<th>$Q$</th>
<th>Gain $\times 10^3$</th>
<th>USB</th>
<th>EtherCAT</th>
<th>Lead filter $\text{[Hz]}$</th>
<th>Lowpass $\text{[Hz]}$</th>
<th>Integrator $\text{[Hz]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1$</td>
<td>2500</td>
<td>2000</td>
<td>3/80</td>
<td>25/80</td>
<td>125/0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>$Q_2$</td>
<td>2900</td>
<td>2000</td>
<td>3/80</td>
<td>25/80</td>
<td>125/0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>$Q_3$</td>
<td>1200</td>
<td>1000</td>
<td>3/50</td>
<td>15/40</td>
<td>125/0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>$Q_4$</td>
<td>1600</td>
<td>1750</td>
<td>3/100</td>
<td>100/200</td>
<td>125/0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>$Q_5$</td>
<td>900</td>
<td>1750</td>
<td>4.5/100</td>
<td>100/200</td>
<td>62.5/0.7</td>
<td>0.0</td>
</tr>
<tr>
<td>$Q_6$</td>
<td>1100</td>
<td>1750</td>
<td>4.5/45</td>
<td>5/40</td>
<td>125/0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>$Q_7$</td>
<td>900</td>
<td>2150</td>
<td>4.0/45</td>
<td>5/40</td>
<td>125/0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>$Q_8$</td>
<td>27</td>
<td>20</td>
<td>4.5/45</td>
<td>4.5/45</td>
<td>125/0.7</td>
<td>0.0</td>
</tr>
</tbody>
</table>
5.1 Step response

The controllers are tuned in the time domain with step responses since it appeared difficult to obtain frequency response measurements of the arm. This is due to difficulties in data logging and the high amount of play in the arm joints. The step responses with the new tuned controllers are shown in Figures 5.1 and 5.2. The feedforward to compensate for gravity which was only tuned for the left arm on USB was switched off to make a clean comparison of the controllers. It can be seen that the controllers tuned for EtherCAT perform about as good as the USB controllers. The settling times and remaining steady state errors are shown in Table 5.2. For the Settling Time a value of two percent of the final value is used as error bound and for the steady state error the error is evaluated at half a second after the step. Again in this table it can be seen that the EtherCAT controllers perform similar to the USB controllers. Note that there is an exception and that is the significant error seen in shoulder joint $Q_1$. It is believed that this is not due to a bad design of the controller since the controller output saturates. This can be seen in Figure 5.1 where for $Q_1$ the scaled control signal is shown. Also it is only seen in the step responses and not during normal operation. Thirdly in the beginning when the controllers were tuned this problem did not occur. In the next section a trajectory is reviewed since the reference interpolator makes sure that under normal operation these sudden motions are smoothed out to a trajectory with acceptable maximum velocity and acceleration.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Settling Time</th>
<th>Settling Time</th>
<th>Steady State Error</th>
<th>Steady State Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>USB [s]</td>
<td>EtherCAT [s]</td>
<td>USB [mrad]</td>
<td>EtherCAT [mrad]</td>
</tr>
<tr>
<td>$Q_1$</td>
<td>0.32</td>
<td>-</td>
<td>0.9</td>
<td>23</td>
</tr>
<tr>
<td>$Q_2$</td>
<td>0.22</td>
<td>0.23</td>
<td>1.1</td>
<td>0.5</td>
</tr>
<tr>
<td>$Q_3$</td>
<td>-</td>
<td>0.19</td>
<td>4.0</td>
<td>0.5</td>
</tr>
<tr>
<td>$Q_4$</td>
<td>0.25</td>
<td>0.18</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>$Q_5$</td>
<td>0.26</td>
<td>0.24</td>
<td>1.6</td>
<td>1.3</td>
</tr>
<tr>
<td>$Q_6$</td>
<td>0.54</td>
<td>0.14</td>
<td>2.4</td>
<td>0.2</td>
</tr>
<tr>
<td>$Q_7$</td>
<td>0.53</td>
<td>0.12</td>
<td>2.5</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Figure 5.1: Stepresponse for shoulder joints $Q_1$, $Q_2$ and $Q_3$ for USB and EtherCAT
5.1. **STEP RESPONSE**

![Graphs showing step response for different joints](image)

Figure 5.2: Step response for elbow joints $Q_4$ and $Q_5$ and wrist joints $Q_6$ and $Q_7$ for USB and EtherCAT
5.2 Trajectory

As shown in the previous section the reference interpolator will make sure that once a new desired position is fed to the control structure the arm will not perform this step function. The reference interpolator will send the arm as quickly as possible without exceeding the maximum accelerations and the angular velocity. It is therefore interesting to compare both arms to see the difference in performance for the EtherCAT controllers with respect to the USB controllers. Four reference points were fed into the control structure, where the reference interpolator made a trajectory to these points. In Figures 5.3 to 5.9 the results of these experiments are shown. The black line and black dashed line indicate respectively the scaled position and scaled angular velocity reference of the trajectory. The red lines are the servo errors and in blue the controller output is shown. The first conclusion that can be drawn from these experiments is that although the left arm has gravity compensation and the right arm does not, the steady state errors of the EtherCAT controlled arm are smaller. The integrating action of the EtherCAT controllers is responsible for this increase in accuracy. It can be immediately seen that for all joints triangular and or trapezoidal shaped error profiles are seen that show a strong relation with the velocity profile. The physical phenomenon that induces this error is friction since friction depends on the angular velocity. Derivative action should take care of this peak however the controllers were tuned with step responses which resulted in less derivative action. There are two possible options to remove this peak from the error profile. The first is a velocity feedforward and the other is to tune the controllers with more derivative action. In the next section a feedforward is added to decrease the error proportional to the angular velocity to investigate this option.

![Graph showing reference and error for shoulder joint Q1 for USB and EtherCAT](image)
Figure 5.4: Reference and error for shoulder joint $Q_2$ for USB and EtherCAT

Figure 5.5: Reference and error for shoulder joint $Q_3$ for USB and EtherCAT
Figure 5.6: Reference and error for elbow joint $Q_4$ for USB and EtherCAT

Figure 5.7: Reference and error for elbow joint $Q_5$ for USB and EtherCAT
5.2 TRAJECTORY

Figure 5.8: Reference and error for wrist joint $Q_6$ for USB and EtherCAT.

Figure 5.9: Reference and error for wrist joint $Q_7$ for USB and EtherCAT.
5.3 Feedforward

The feedforward is added in this section to remove this error due to friction. The feedforward is tuned for each joint and the resulting error profiles are shown in Figures 5.10 to 5.13. The USB controlled arm, EtherCAT controlled arm without feedforward and the EtherCAT controlled arm with feedforward are compared with the same trajectory used in the previous section. It can be seen that the EtherCAT controlled arm has a smaller error compared to the USB controlled arm. The RMS values of the errors of the trajectories are shown in Table 5.3. It can be concluded that the EtherCAT controller with the velocity feedforward and the integrators performs better than the USB controllers without integrators and with the gravity compensation feedforward.

Table 5.3: Average Errors (RMS) in all joints [mRad]

<table>
<thead>
<tr>
<th>Joint</th>
<th>Error USB (RMS)</th>
<th>Error EtherCAT (RMS)</th>
<th>Error EtherCAT with FFW (RMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_1</td>
<td>4.0</td>
<td>5.3</td>
<td>1.4</td>
</tr>
<tr>
<td>Q_2</td>
<td>7.4</td>
<td>8.8</td>
<td>1.6</td>
</tr>
<tr>
<td>Q_3</td>
<td>3.5</td>
<td>6.5</td>
<td>0.8</td>
</tr>
<tr>
<td>Q_4</td>
<td>3.1</td>
<td>5.2</td>
<td>1.4</td>
</tr>
<tr>
<td>Q_5</td>
<td>2.9</td>
<td>8.6</td>
<td>0.5</td>
</tr>
<tr>
<td>Q_6</td>
<td>0.5</td>
<td>2.5</td>
<td>0.6</td>
</tr>
<tr>
<td>Q_7</td>
<td>0.6</td>
<td>2.1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Figure 5.10: Reference and errors for shoulder joints Q_1 and Q_2 for USB, EtherCAT, and EtherCAT with feedforward
5.3. FEEDFORWARD

Figure 5.11: Reference and errors for shoulder joint $Q_3$ for USB, EtherCAT, and EtherCAT with feedforward

Figure 5.12: Reference and errors for elbow joints $Q_4$ and $Q_5$ for USB, EtherCAT, and EtherCAT with feedforward
Figure 5.13: Reference and errors for wrist joints $Q_6$ and $Q_7$ for USB, EtherCAT, and EtherCAT with feedforward
Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The first conclusion is that a control structure is presented that makes use of the EtherCAT protocol. This control structure is an adaptation from the existing control structure which was set up in a generic manner. Therefore this project not only resulted in the functioning of the right PERA of AMIGO but also in a generic control structure that can be used in other projects for example AMIGO2 which make use of the EtherCAT PCB’s.

The second conclusion is that this control structure is functioning stable and with safety measures to ensure the safety of both the environment and the hardware of AMIGO itself. The EtherCAT control structure has similar safety measures than the USB control structure but a key difference is the addition of the heartbeat. For the USB arm it was possible that once the amplifiers were enabled and the motors were actuated and the communication of the USB boards failed, that the arm kept moving. For the EtherCAT control structure, this is not possible since a safety measure running on the lowest level listens to a heartbeat from Orocos.

The last conclusion is about the performance of the EtherCAT controllers. With the velocity feedforward and integrating action, the EtherCAT control performance is better than the USB control with gravity compensation and no integrators.

6.2 Recommendations

The first recommendation is to convert the left PERA to EtherCAT control and to use this conversion to investigate if tuning the controllers with more derivative action can make the velocity feedforward obsolete. Also it is an opportunity to investigate the benefits of the feedforward gravity compensation. Subsequently it is interesting to see if the maximum velocities and accelerations set in the reference interpolator can be increased to increase the speed at which AMIGO manipulates objects while still being able to guarantee safe operation of the arm.

A third recommendation is that the control structures of the the base, spindle and PERA should be made more generic. For example the homing procedure is implemented in different manners for these control structures. The goal is a generic control structure for EtherCAT control either for the PCB’s used in the arms or via the Beckhoff stacks for base and spindle. Also the new Supervisor component implemented at the moment for the base and spindle should be added to the arm control structure to be able to start and restart the arms more easily.
At last, for numerous tasks it is desired to control the arm not only in position but also have force control. For example if a person touches the arm it should be able to sense this applied force and move away compliantly. In this case for example a high stiffness in the z-direction could be chosen and a low stiffness in x- and y-direction. Another task that would benefit from this is opening a cabinet where a trajectory should be followed to open a cabinet door. Smart force constraints make following this trajectory more easy since it is not simple to calculate the exact trajectory that the handle of the cabinet should make.
Bibliography