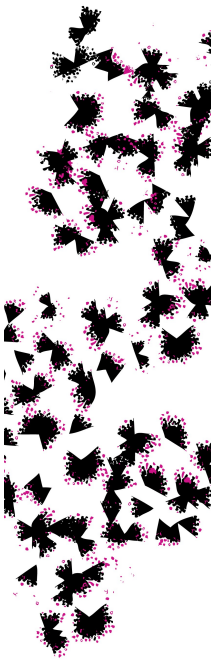

# Development of the Low-level Software Architecture for the Sherpa Robot Arm

W. (Willem) Boterenbrood

BSc Report

**Committee:**
Dr. R. Carloni
Dr.ir. M. Fumagalli

August 2015

024RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE.

MIRA CTIT
BIOMEDICAL TECHNOLOGY AND TECHNICAL MEDICINE

# Development of the low-level software architecture for the Sherpa robot arm

W. Boterenbrood

August 26, 2015

# Summary

The SHERPA project aims to improve rescuing activities in alpine environments by using smart collaboration between humans and robots. Within the SHERPA work, a seven degrees of freedom robotic arm has been developed. The robotic arm is mounted on a ground rover and it needs to dock and undock the drones that need to swap their battery packs.

In this BSc project, a low-level software interface has been developed that is able to drive the arm from ROS. A ROS software node is presented that can drive the motors in the arm over a CAN bus. The ROS software node supports driving motors on multiple CAN buses, can drive a differential pair of motors as two separate joints and is built in way that makes it relatively easy to add support for more types of CAN devices. A Graphical User Interface is provided to test the ROS software node.

# Contents

**1 Sherpa Project Overview** **1**
1.1 Interaction . . . . . 1
1.2 The SHERPA team . . . . . 1

**2 Introduction** **3**
2.1 Existing design choices . . . . . 3
2.2 Task list . . . . . 3
2.3 Task details . . . . . 3

**3 Electronic Design** **5**
3.1 Existing design choices . . . . . 5
3.2 Interfacing all components to the computer platform and providing power . . . . 5
3.3 DC-DC converter selection . . . . . 6
3.4 Elmo Whistle selection . . . . . 7

**4 Software architecture** **8**
4.1 Pre knowledge . . . . . 8
4.2 Design Start . . . . . 8
4.3 Software design description . . . . . 8

**5 ROS-CAN Interface design** **11**
5.1 C++ CAN bus projects research . . . . . 11
5.2 Design decisions . . . . . 11
5.3 Start-up and initialization . . . . . 13

**6 Joint Controller design** **14**
6.1 Inner workings . . . . . 14

**7 Test GUI design** **16**

**8 Calibration of the shoulder joints** **17**

**9 Tests and validation** **18**
9.1 Shoulder rotation, non differential drive . . . . . 18

**10 Conclusion and future work** **25**
10.1 Conclusion . . . . . 25
10.2 Future work . . . . . 25

**A Component list** **27**

**B DC-DC converter list** **28**

**C Elmo motor controller programming** **29**

C.1 Example1 . . . . . 29

C.2 Example2 . . . . . 31

C.3 Example3 . . . . . 31

**D Software User Manual** **33**

D.1 Configuration file . . . . . 33

D.2 Compiling the source files . . . . . 36

D.3 Running the software . . . . . 36

D.4 Rostopics and messages usage . . . . . 37

**E Test GUI User Manual** **39**

E.1 TestGUI controlling the ROS-CAN interface . . . . . 39

E.2 TestGUI2 controlling the Joint Controllers . . . . . 40

**Bibliography** **41**

# 1 Sherpa Project Overview

This bachelor assignment is part of the development of the robot arm for the SHERPA project (Marconi et al., 2012). The SHERPA project is a European project that aims to improve rescuing activities in alpine environments. This is achieved by smart collaboration between humans and ground & aerial robots.

## 1.1 Interaction

The SHERPA project consists of separate robotic components that operate as a team in the environment. All components communicate with each other through a software framework using wired or wireless communication. All components are controlled by humans that coordinate the tasks of all robots.

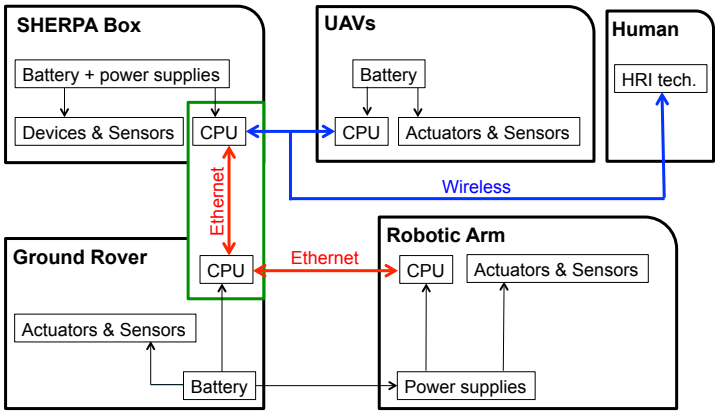


**Figure 1.1:** Communication

## 1.2 The SHERPA team

The trained human rescuers are skilled individuals that do the real rescue work. Because of hostile terrain and required focus they can cover limited ground and have limited time. Unmanned aerial vehicles (UAVs) are used to provide overview and scouting information to the rescuers. Because the rescuers are busy with their main tasks the control of these UAVs must take little effort from them. To achieve this little effort control UAVs that can operate autonomously with predefined tasks are used, software is developed that connects all components together to exchange tasks and data and human gesture control is implemented to give simple commands to the UAVs.

Small flying UAVs can fly at low-altitude and can get close to a target. These UAVs have a high-maneuverability to fly around obstacles and in confined spaces. Due to limited battery capacity these UAVs must operate close to the ground rover that provides a way to supply the UAVs with new batteries.

**Figure 1.2:** Small hovering UAV

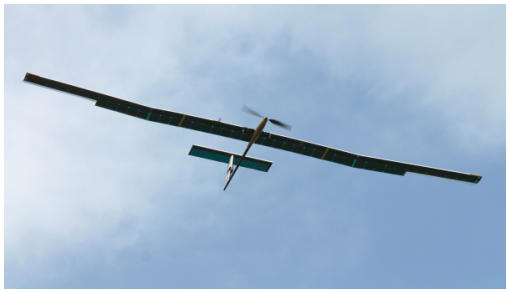

**Figure 1.3:** Large glider UAV

Flying UAVs fly at high-altitude and are able to patrol large areas with a limited amount of energy and fly in critical weather conditions. The information captured by these UAVs help the coordination and optimization of the activities of the rescue team and complement the capabilities of the small UAVs.

A ground rover transports the Sherpa box, the robot arm and the main batteries to power the Sherpa box, the robot arm and itself. The Sherpa box houses the central intelligence and functions as a docking station for the small UAVs. When docked the battery of a UAV can be swapped for a full one and the old one can be recharged.

**Figure 1.4:** Ground rover with robot arm and Sherpa box

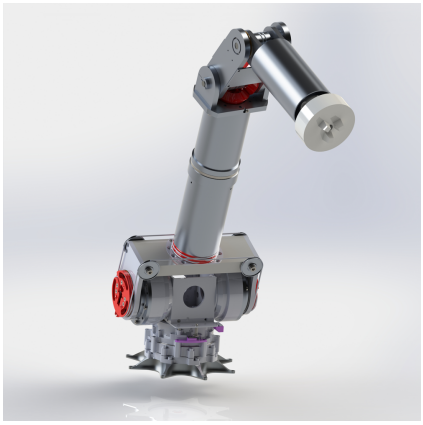

**Figure 1.5:** Robot Arm

The robot arm docks and undocks the UAVs to the Sherpa box and can catch a flying UAV from the air. It contains a 3D camera to avoid obstacles and locate the precise location of a landed UAV. The robot arm contains a programmable stiffness. When making large movements a low stiffness makes it less precise but prevents the rover from tilting when hitting an obstacle, while making small movements a high stiffness makes it more precise so it is able to place the UAV on the Sherpa box. The arm consists of a shoulder with 3 degrees of freedom (DoF), the lower rotation is driven by a single motor, both other rotations of the shoulder are driven by a differential drive setup, where one rotation is done by moving both motors in the same direction and the other rotation is done by driving both motors in opposite directions. Above the shoulder is the elbow with one DoF and the wrist with 3 DoF.

# 2 Introduction

## 2.1 Existing design choices

Before the start of this bachelor assignment previous work has been done on the design of the robot arm. The mechanical design of the robot arm and some of the components to be used have already been selected. This bachelor assignment consist of two parts in the development of the robot arm. A minor part is electronic design and the major part is low-level software design. The following tasks are part of this assignment.

## 2.2 Task list

- Creating an overall electronic schematic which connects all the electronic components to the computer platform and power supply
- Creating a high-level schematic of the software architecture which integrates the low-level software that is developed in this assignment
- Creating a ROS (ROS.org, 2015a) node that provides an interface between the CAN bus (CiA, 2015) and ROS that supports Elmo motor controllers and can be extended easily with support for other sensors on the CAN bus.
- Creating a ROS node that can translate the native commands from and to the motor controllers to standard units and which can drive a differential pair of motors as two separate joints
- Creating a ROS node that provides a GUI with sensor readings as output and target position, speed and torque controls as inputs which can be used to test the other ROS packages and the robot arm itself.
- Extend the GUI ROS node with joystick support to drive the robot arm.

## 2.3 Task details

The overall electronic schematic consists of connecting the position sensors, the force/torque sensor (ATI Industrial Automation, 2015), the VI sensor (Skybotix, 2015), the motors, the network connection to the Intel NUC computer platform (Intel Corporation, 2015) and supplying the required power to all components. If the NUC does not provide sufficient I/O interfaces, the interfaces must be extended or another computer platform must be found that provides enough I/O. Many components have a CAN bus interface. The computer will have access to the CAN bus components via three USB-CAN interfaces. The robot arm is supplied with a 48 Volt DC power supply, all components of the arm that require a different voltage will get their power from separate commercially available DC-DC converters or from the computer platform.

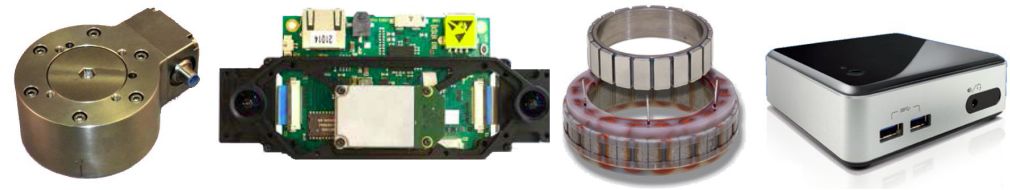

**Figure 2.1:** The F/T Sensor, VI Sensor, Robodrive Motor and NUC Computer

The high-level software design should provide an overview of the communications between the separate ROS packages, and other software and hardware. The low-level software development

consists of creating a number of ROS nodes. The first ROS node will provide an interface/driver that provides a way to control the devices on the CAN bus with standard ROS messages. Another ROS node translates the native commands from and to the motor controllers to standard units and can drive a differential pair of motor controllers as two separate joints, so the actual controller doesn't need to handle native motor controller units and doesn't need to be aware of joints with a differential drive. The last ROS node will provide a GUI that displays the sensor readings and allows a human to enter target positions, velocities torques and start calibration of a joint. This last node will be used to test the other ROS nodes and the robot arm itself and will be extended with a joystick interface to control the arm.

# 3 Electronic Design

## 3.1 Existing design choices

Before this bachelor assignment started many components were already chosen for the robot arm, and the shoulder was already under construction. The list all these components can be found in appendix A. To allow easier servicing of the rover/arm/box platform many components are the same brand and model, as this requires less stock of spare components. The Intel NUC (Intel Corporation, 2015) computer is the same as the one used by the Sherpa box and is software compatible with the x86 computer used in the rover, also all computers run Ubuntu Linux. The usage of identical instruction set processors and operating systems allows for software code sharing and future computer platform sharing. Reducing the amount of computers decreases costs and decreases power consumption. The USB CAN Interfaces used are EMS Dr. Thomas Wünsche CPC-USB/ARM7 (EMS Dr. Thomas Wünsche, 2015) units which are also used in the rover. The Elmo Whistle Elmo motor controllers used in the arm are also used in the rover and are a standard component in the RaM group at the University of Twente, so there is already some knowledge about them and some testing setups are already available. The range of available models, the amount of compatible motors, extensive range of compatible feedback mechanisms and built in PI controller makes the Elmo Whistle a very versatile component that is usable to drive all motors in the robot arm. The robot arm needs absolute encoders to determine the angle of all joints as the variable stiffness decouples the motor position from the the joint angle. The RLS AksIM (RLS, 2015) range of absolute encoders combine high accuracy, small size, and multiple interface options. The motors for the shoulder have been chosen based on required torque, size and weight, to determine the torque requirements an analysis on a model of the arm was done before the start of this assignment. The Kollmorgen RBE 02210-A (Kollmorgen, 2015) and the TQ Goup Robodrive 70x18 (TQ Group, 2015) have been chosen for the shoulder, the Robodrive 50x14 has been chosen for the elbow, while the rest of the motors will need to be determined at a later time.


**Figure 3.1:** AksIM Absolute encoder, EMS USB CAN Interface, Elmo Whistle Motor Controller, Kollmorgen RBE Motor

## 3.2 Interfacing all components to the computer platform and providing power

The robot arm will be connected to the rover platform by a wired LAN connection, the rover has a built-in Ethernet switch that connects the arm, rover and Sherpa box computers together. The VI sensor also needs a wired LAN connection. The NUC platform has only one LAN connection, an external router can be used but takes extra space and power. A mini PCI express LAN card can be connected to the NUC which fits inside the NUC and is powered from the NUC. The Rover supplies the robot arm with 48V DC which can be used to power the motor controllers but none of the other components. The absolute encoders require 5V and the VI Sensor requires 12V, the Force/Torque sensor requires between 12V and 24V and the NUC requires between 12V and 19V. So at least two extra voltages are needed which can be provided by standard DC-DC converters. Three USB CAN bus interfaces have been chosen to make sure enough

bandwidth is available. All devices for one degree of freedom (DoF) are grouped together and will be connected to the same CAN bus. This has the advantage of easier message grouping in the ROS software and it means communication between them is possible without help of the ROS software if necessary. The schematic can be found in Figure 3.2. In the schematic all components are connected to the power sources and to the computer platform. With every connection made there are still some USB ports available on the NUC platform for future use so this setup is flexible enough and the selected NUC is a good fit for the robot arm.

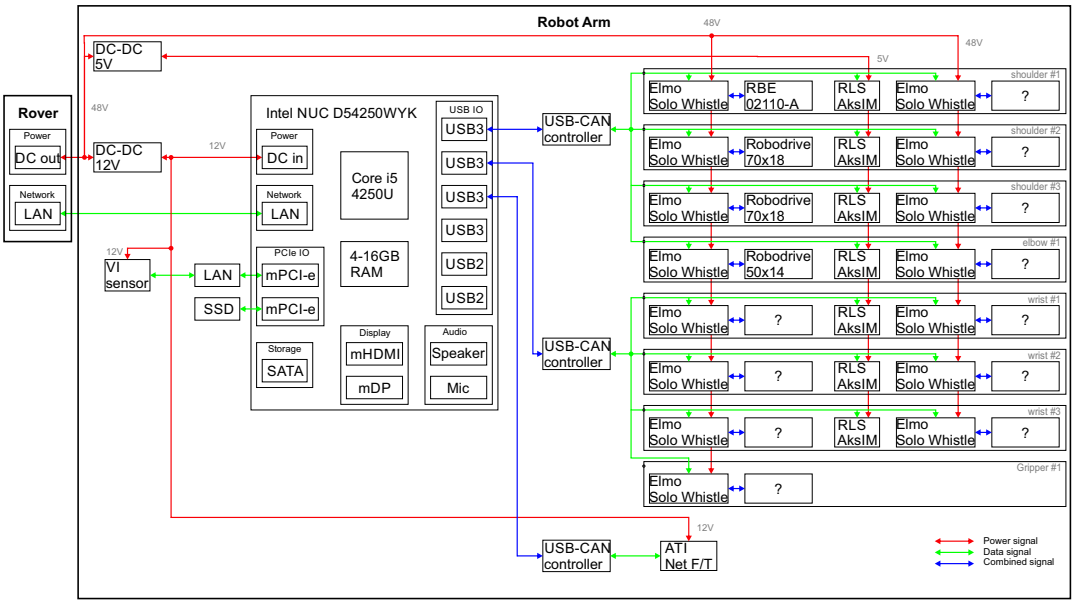


**Figure 3.2:** Electronic Connections of the robot arm

## 3.3 DC-DC converter selection

For the DC-DC converters commercially available models will be used. The 12V line powers the Force/Torque sensor (ATI Industrial Automation, 2015), VI Sensor (Skybotix, 2015) and the computer (Intel Corporation, 2015) and should be able to provide at least 7.1A and the 5V line 1.1A (See power requirements in Appendix A). A list of DC-DC converters that can provide at least 12V 7.5A and 5V 1.5A and which can handle an input voltage of at least 36-60V can be found in Appendix B. An important aspect of the converter is the efficiency as the rover is battery powered, so a higher efficiency will allow for increased operation time. A small size and low price are also important. Only the smaller modules with a price below 50 euro for the 5V converter and only modules below 75 euro for the 12V converter are listed. For the 5V converter the GE KHHD006A0A41Z (Farnell, 2015b) module is chosen as it combines a high efficiency with high power output capability and it has a low ripple and good regulation specifications. For the 12V converter the GE EKV010A0B41Z (Farnell, 2015a) is chosen. Both DC-DC converters will still need in and output capacitors and will need to be mounted on a PCB which has to be designed.

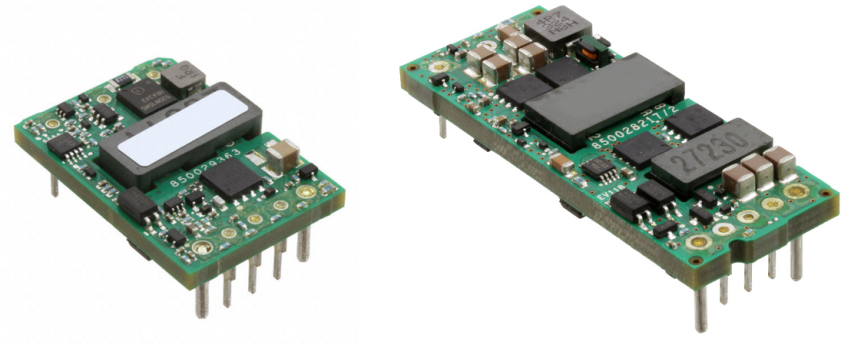

**Figure 3.3:** General Electric KHHD006A0A41Z and EVK011A0B41Z DC-DC Converters

## 3.4 Elmo Whistle selection

The Elmo Whistle motor controllers have been selected to drive the motors in the arm, to determine which version is needed the motor specifications are consulted. The shoulder uses two motor types, the specifications of both motors are listed in Table 3.1 and the specifications of the Elmo Whistle models are listed in Table 3.2. The 60V Elmo line is the perfect match, with a nominal input of 50V which is very close to the 48V DC supply. The Elmo Whistle should be able to provide the full continuous power needed by the motor. The peak current and power of the motors can be extremely high according to the specifications. Choosing an Elmo Whistle model that can handle 75% of the peak motor current will be enough as the motors should never reach this peak current in the arm. Using a 75% peak requirement enables the use of a lower capacity model for all the motors in shoulder and decreases the project cost. The Whistle 10/60 is selected for the Kollmorgen RBE 02110-A motor and the TQ Group Robodrive 50x14 motor. The Whistle 20/60 is chosen for the TQ Group Robodrive 70x18 motor.

| Brand | Model | Voltage | Current | Power | Peak Current | Peak Power |
|---|---|---|---|---|---|---|
| Kollmorgen | RBE 02110-A | 38V | 6.34A | 241W | 25.3A | 960W |
| TQ Group | Robodrive 70x18 | 48V | 7.0A | 370W | 43.75A | 2100W |
| TQ Group | Robodrive 50x14 | 48V | 3.5A | 145W | 25A | 1200W |

**Table 3.1:** Motor power requirements for the shoulder

| Brand | Model | Nominal Voltage | Max Voltage | Current | Peak Current |
|---|---|---|---|---|---|
| Elmo | Whistle 15/48 | 42V | 48V | 15A | 30A |
| Elmo | Whistle 20/48 | 42V | 48V | 20A | 40A |
| Elmo | Whistle 1/60 | 50V | 59V | 1A | 2A |
| Elmo | Whistle 2.5/60 | 50V | 59V | 2.5A | 5A |
| Elmo | Whistle 5/60 | 50V | 59V | 5A | 10A |
| Elmo | Whistle 10/60 | 50V | 59V | 10A | 20A |
| Elmo | Whistle 15/60 | 50V | 59V | 15A | 30A |
| Elmo | Whistle 20/60 | 50V | 59V | 20A | 40A |
| Elmo | Whistle 1/100 | 85V | 95V | 1A | 2A |
| Elmo | Whistle 2.5/100 | 85V | 95V | 2.5A | 5A |
| Elmo | Whistle 5/100 | 85V | 95V | 5A | 10A |
| Elmo | Whistle 10/100 | 85V | 95V | 10A | 20A |
| Elmo | Whistle 15/100 | 85V | 95V | 15A | 30A |
| Elmo | Whistle 20/100 | 85V | 95V | 20A | 40A |

**Table 3.2:** Elmo Whistle Models

# 4 Software architecture

## 4.1 Pre knowledge

Before starting the design of the software architecture knowledge about ROS, C++ and the CAN bus is necessary. To learn ROS the following sources were used:

- ROS concepts (ROS.org, 2015a)
- basic ROS tutorials (ROS.org, 2015b)

To learn C++ the following sources were used:

- C++ course at learncpp.com (Alex, 2015)
- C++11 Rocks book (Korban, 2014)

To learn about the CAN bus the following sources were used:

- What is CANopen presentation (ElmoMC, 2014b)
- CANopen DS 301 Implementation Guide (ElmoMC, 2014a)
- CAN tutorial (Computer Solutions Ltd, 2014)

## 4.2 Design Start

At the start of this assignment many software design decisions were already made, ideas were discussed and a sketch of the high-level software architecture was made. The design is based on a bottom-up approach where the low-level functionality was listed and then divided to groups of tasks that were assigned to ROS nodes. The tasks were grouped based on the arm joints and the communication flows. The high-level overview can be found in Figure 4.1. The design decisions and message structure on which the overview is based is listed in Table 4.1 and Table 4.2.

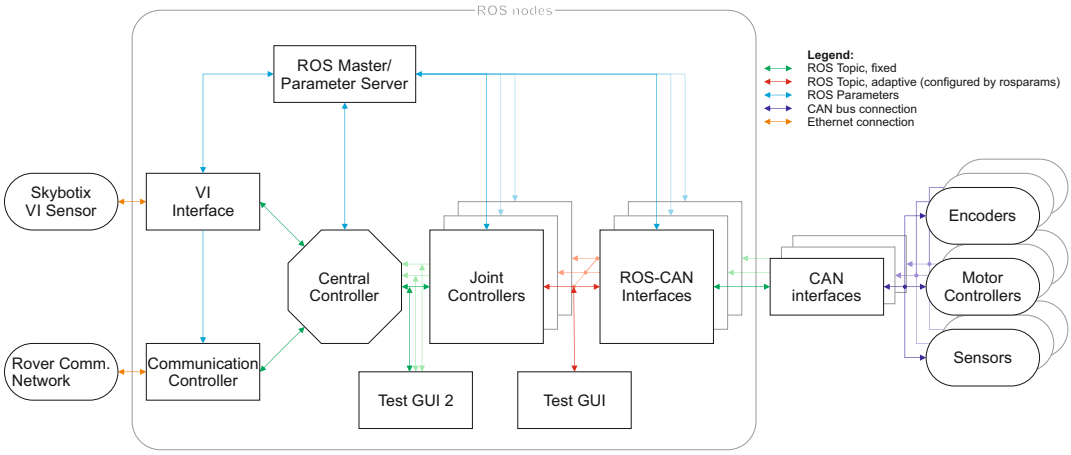


**Figure 4.1:** Software Architecture overview

## 4.3 Software design description

The robot arm communicates with the rest of the Sherpa project components through the ROS-MicroBLX bridge (made by the Sherpa team from KUL, Leuven) with the TST framework and

the world model (made by the Sherpa team from LKU, Linköping). All the communication is handled by the Communication Controller node.

The robot arm software communicates with a VI sensor and a number of motor controllers, encoders and sensors that are all connected through multiple CAN buses. The VI Interface controls the communication between the VI sensor and the Central Controller. The ROS-CAN interface nodes control one CAN bus each and translate the ROS topic messages to and from CAN frames. The number of ROS-CAN Interface nodes equals the number of CAN buses and is configured in the ROS Parameter Server. The Joint Controller nodes translate the native units of the CAN devices to standard units, so the Central Controller does not care what units are used in each CAN device. The Joint Controller nodes also separate each degree of freedom as a separate topic and do the translation of differential driven joints (as used in the shoulder).

The Central Controller node makes all control decisions based on sensor data and the requested tasks from the TST framework. It contains the arms TF model and provides the data that is needed to update the robot arm part of the world model to the Communication Controller node.

| **Decision** | **Motivation** |
|---|---|
| A configuration file with a list of all devices on the CAN buses and the corresponding CAN bus, CAN id, gear ratio and other configuration data is imported to the parameter server at start-up. | All ROS nodes can access this configuration data and use this to configure themselves. The ROS nodes are started after the configuration import is done. The configuration data is saved in a file that can be imported in the ROS parameter server. |
| Each CAN bus will communicate with one process only. | This way there is no possibility that multiple processes will try and access the same CAN bus resulting in a conflict and each CAN bus can communicate simultaneously without waiting for each other. For each CAN bus a separate instance of the ROS-CAN interface node is created to achieve this. |
| The rate at which CAN messages are send must be configurable. | To prevent flooding the CAN bus a rate limiter should be used, this limiter can be constructed with queue of messages to be send and a timed loop that pauses for a specified period before sending the next message. |
| The ROS nodes that communicate directly with the CAN buses will map the in- and output communications to topics with names based on the joints they represent. | With this setup the rest of the system doesn't need to know which sensor/actuator is connected to which CAN bus but can just communicate with a specific joint. The ROS parameter server will contain a list of devices on the CAN bus with corresponding bus and topic names that can be used by the ROS-CAN interface node. |

**Table 4.1:** Initial design decisions part 1

| Decision | Motivation |
|---|---|
| The joint controller nodes start the built-in calibration process at start-up and only start communication with the central controller once the calibration is complete. | This prevents movement of a joint before it is calibrated. The joint controller will set a calibration complete flag in the ROS parameter server for each node that has completed calibration, so the central controller (and test GUI) can check when they are able to control the arm and start sending control messages. Calibration is only needed for joints that do not have an absolute encoder as position detector. Sending a calibration request to a node that has an absolute encoder as position detection will set the calibration complete flag immediately. |
| The central controller node (and test GUI) will be able to request another calibration process when the arm is operating without having to restart the software. | When a problem is detected where the arm did not move to the position it is supposed to be a recalibration might fix this. |
| All messaging between ROS nodes will by default be done by standard ROS topics. | Topics are easier to debug and more flexible than services due to their ability to support multiple senders and receivers. |
| All joints will be presented to the Central Controller as independent joints. Communication to and from these joints will be in standard units (position in radians and speed in radians per second). | The central controller doesn't have to deal with differential drives and device/configuration specific units, the translation is done by the joint controllers, which read all required configuration data from the ROS parameter server and choose the correct transformations automatically. |
| All code will be written under Ubuntu Linux in C++. | C++ code is faster than Python which results in less processor usage and therefore uses less energy and allows for more software on the same computer. (ROS fully supports C++ and Python and is made to run on Ubuntu Linux.) |

**Table 4.2:** Initial design decisions part 2

# 5 ROS-CAN Interface design

## 5.1 C++ CAN bus projects research

Before starting to write code a search on the internet for projects that might be usable for this assignment was done. The following C++ CAN Linux projects on the internet were found:

- BlueBotics Librover project (BlueBotics, 2013), the C++ code for their rover vehicle that uses a CAN bus for the motors.
- IPA Canopen project (Fraunhofer, 2015), a C++ project that connects ROS with the CAN bus.
- CAN Festival project (Festival, 2015), an ANSI-C platform independent CANopen stack.
- cob_generic_can ROS package (Connette and Gruhler, 2015), a C++ project that connects ROS with the CAN bus.

For this project the EMS USB to CAN adapter was chosen as it was also used by BlueBotics for the Sherpa Rover platform. When connecting the EMS USB to CAN adapter to a Linux host the SocketCAN driver from the Linux kernel is loaded. Linux SocketCAN is the default way to use CAN under Linux that supports many CAN adapters. The BlueBotics Librover project uses the same EMS USB to CAN adapter, their code uses the EMS CAN development kit software and drivers and not the SocketCAN driver. The cost of the EMS CAN development kit was such that it was decided to not use that but use SocketCAN instead. The IPA CANopen and cob_generic_can are not generic CAN packages but made for one specific robot only, both use the Peak Systems CAN bus adapters which have their own PCAN drivers and commands (Before SocketCAN was introduced in the Linux kernel PCAN was the standard for using CAN with Linux, which is why PCAN is still used in most Linux projects). CAN Festival is made in C not C++ but does support using SocketCAN. Getting CAN Festival to work under Ubuntu 14 with the EMS USB to CAN adapter and the Elmo Whistle motor controller failed. CAN Festival is quite a big project and the code is not C++ but C, most documentation and examples use the PCAN driver and before being able to use it a lot of code needed to be added for supporting the ELMO binary commands or the CANopen CiA 402 generic motor control standard. First an easier way to make the EMS USB to CAN adapter and the Elmo Whistle work under Linux needed to be found.

Ubuntu Linux has a package called can-utils, a SocketCAN userspace utilities and tools package. With this package it was possible to send commands to the Elmo over the CAN bus and receive messages back. The source files can be downloaded and a way was found to send and receive CAN messages to and from the SocketCAN driver. Using this knowledge and writing my own C++ ROS/CAN package seemed like much less work than understanding the inner workings of CAN Festival and adapt that for this project. The BlueBotics Librover project and both PCAN projects could be adapted to SocketCAN but did not provide all required functionality as the parts that could be used were mostly simple functions to translate a basic command. They did not provide a way to deal with multiple CAN buses, multi message SDO communication and using a differential drive setup.

## 5.2 Design decisions

The block level design can be found in Figure 5.1. The ROS-CAN Interface has a part to send and a part to receive CAN frames, the CANSender and CANReceiver blocks, both with their own queue to prevent loosing messages. To support multiple CAN device types a separate control block for each device type can be added, for now only Elmo motor controllers are supported in

the code (Elmo::Node). Because the number of devices/nodes on a CAN bus are not known in advance and each CAN device/node should have its own state variables and configuration parameters, a device specific control block is dynamically created for each CAN device/node that holds all device specific functions, state variables and configuration parameters. This way the amount of devices is limited only by the amount of RAM and different node id's on the CAN bus while only using memory for devices that are connected. The incoming CAN frames are send to the right device specific control block by the NodeContainer. The NodeContainer is responsible for creating the instances of the device specific control block and keeping a list that maps each CAN node id to the corresponding device specific control block. In C++ it is not possible to create a list with different objects, in order to prevent creating a separate list for each device type, each device specific control block is derived from a generic class (Node). The NodeContainer will have a list of pointers to these generic parts instead of the device specific parts so one list can be used. To access functions that are part of the device specific control block from the NodeContainer, virtual functions in the generic class are used. This generic class is also used for device independent variables and functions, which allows for smaller device specific classes and less duplicate code. Each of the device specific control blocks and there base class subscribe and publish directly to ROS. Automatic handling of multi-message SDO upload CAN communication and handling of emergency messages is done by the CANController. The device specific control blocks can start a SDO upload with one command and receive one answer back, because the CANController will handle the multi-message communication and combine the received data in one answer. SDO download is not implemented yet as this was not needed, emergency messages will only display an error on the command line at this moment, so this will need to be extended in the future.

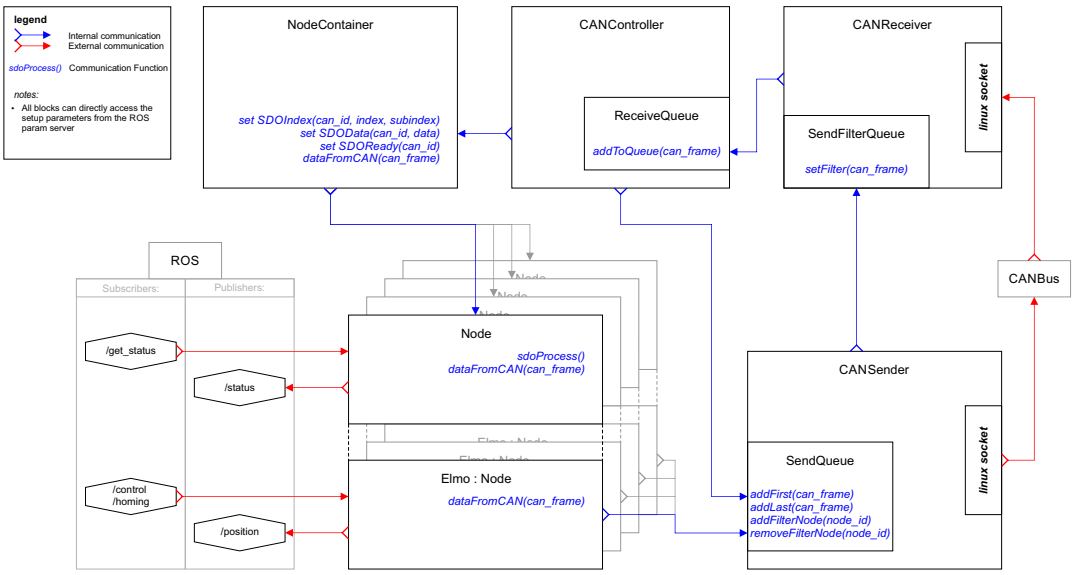


**Figure 5.1:** ROS-CAN interface diagram

The CANSender sends all frames from the SendQueue to the CAN bus, before sending a frame to the CAN bus it will send it to the SendFilterQueue of the CANReceiver. When the CANReceiver receives a frame it will compare this with the SendFilterQueue and if a match is found it will drop the frame and remove it from the SendFilterQueue, if no match is found the frame is added to the ReceiveQueue. The CANController will process al frames from the ReceiveQueue, can frames that are part of an SDO upload are handled automatically and once complete the SDO index and data are sended to the NodeContainer and the SDO ready flag is set, other CAN data will be send directly to the NodeContainer. The NodeContainer forwards the incoming can frames and SDO data to the corresponding Node. The Nodes will handle incoming can

frames and ROS messages and are able to send data to ROS and the CANbus. The CANSender allows mesages to be added to the front or back of the SendQueue and has the option to set a filter to ignore messages from a specific CAN device.

### 5.3 Start-up and initialization

When the ROS-CAN Interface is started it checks the ROS parameter server for the configured CAN busses. For each bus a separate copy of above block structure is created and for each configured CAN device/node a separate copy of the Node and device specific part is created. (For now only the device specific part for Elmo motor controllers is available.)

After all blocks are created the CANSender blocks will each reset their CAN bus and listen for attached devices/nodes. For each found node its name, hardware version and software version is requested. Then an overview of all configured and found devices/nodes is displayed in ROS messages and all devices are given the switch to operational mode command. Now the ROS-CAN Interface is ready for use through the ROS topics

# 6 Joint Controller design

The Joint Controller translates the native Elmo units to and from standard units and separates the differential motor pairs into two independent joints. This is done so the Central Controller doesn't need to deal with differential joints and native units. See Figure 6.1 for the design of the joint controller.

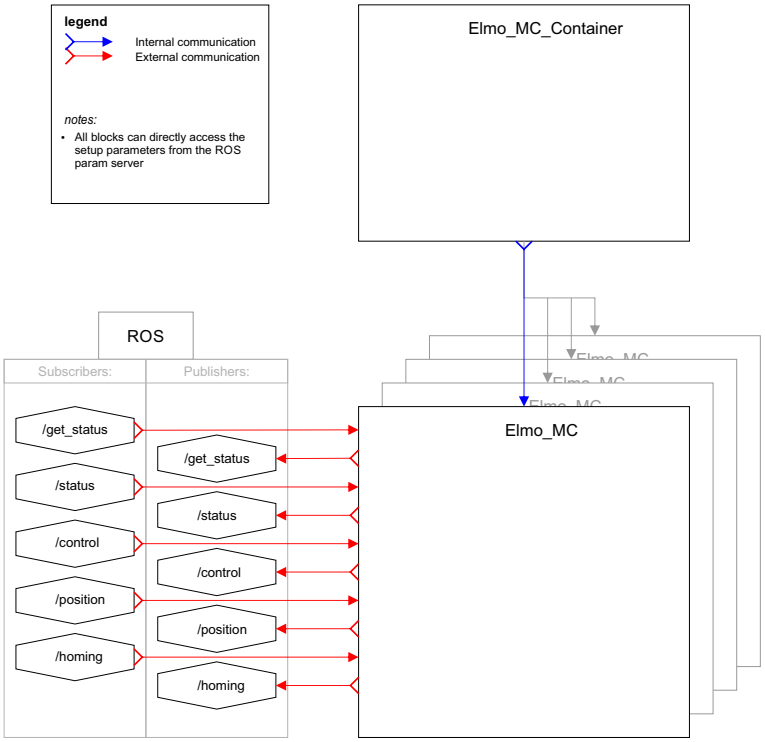


**Figure 6.1:** Joint Controller diagram

The Joint Controller consists of a container (Elmo_MC_Container) that holds all Elmo_MC instances. For each Elmo device on the CAN bus an instance of an Elmo_MC object is created. The Elmo_MC object will subscribe and publish to the ROS topics belonging to that device, the information about the topics is retrieved from the ROS parameter server. When a control or position message arrives the units are converted from standard units to device specific units, the required conversion constants are retrieved from the ROS parameter server.

## 6.1 Inner workings

When an Elmo_MC instance retrieves all required information from the ROS parameter server it checks if it is part of a differential drive pair. When the instance is part of a differential drive pair and its node number in the ROS parameter server is higher than the other drive of the differential pair it deletes itself. If the instance has the lower number of the pair it also subscribes and publishes to the topics of the other drive of the pair. This instance will use the lower node number as the sigma joint that adds the position of both motors, and it will use the higher node number for the delta joint that subtracts the position of the first with that of the second motor. When a calibration request is received for a joint of a differential pair the request will not be

forwarded to the ROS CAN Interface, as done with a non-differential drive, but it will start a special calibration control thread that takes over the calibration of both motors of the differential pair. The Joint Controller is also responsible for joint limit control. If a joint comes outside its minimum and maximum position range the motor is immediately stopped, if there are still motion commands in the queue they are deleted and any motion commands that arrive are discarded. The joint controller will enable the motor in position mode and move the joint to the closest edge of the normal operating range and enable normal operation again when done.

# 7 Test GUI design

To test the ROS CAN Interface a GUI was made that shows the position and status information of the first three CAN devices/nodes and it allows to input control messages, send a calibration or status request. It also provides an interface to use a joystick to control the CAN devices/nodes. It does not support differential drives as the differential drive control is done by the Joint Controller. The joystick interface has very limited differential drive support as the GUI allows a joystick to map the addition or difference of the joystick X and Y axis (XY+, XY-) to a CAN device/node. The units in the GUI are in counts (Position Mode), counts per second (Velocity Mode) and mA (Torque Mode), all units are integers so no decimals are used. The

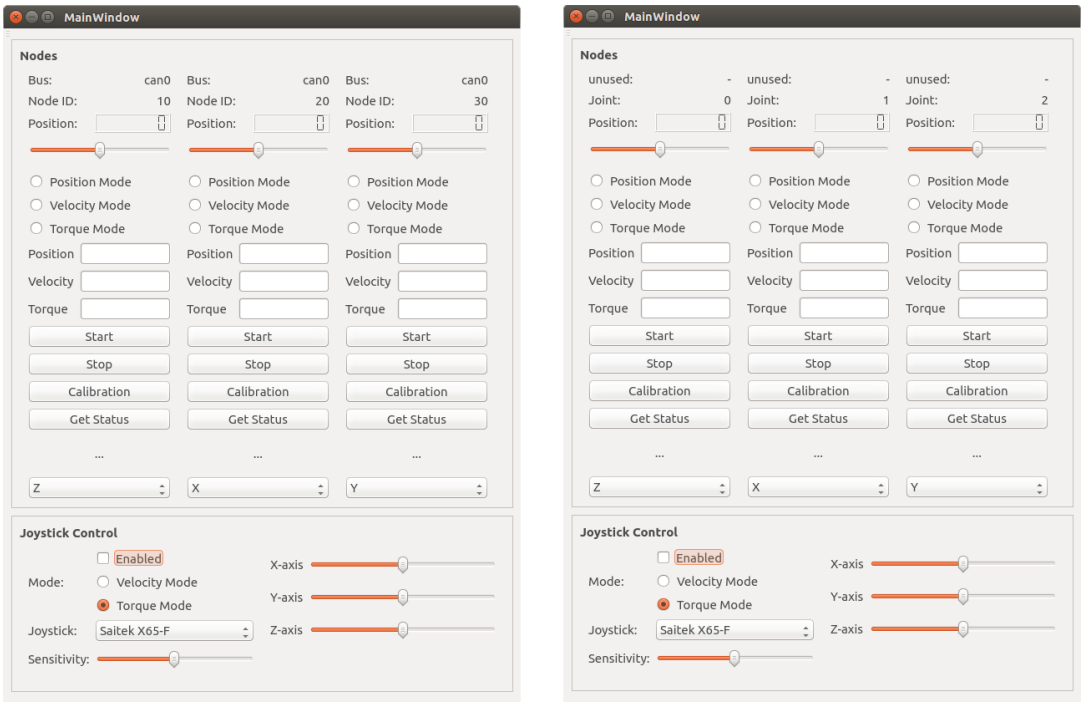


**Figure 7.1:** Test GUI and Test GUI 2

The test interface is created with QT Creator and written in C++ and QT5. To learn QT5 the QT Widget tutorials (The Qt Company, 2015) are used.

To test the Joint Controller the GUI was copied and changed to publish and subscribe to the topics of the Joint Controller instead of the ROS CAN Interface. The units are changed by the Joint Controller and are in radians, radians per second and Amperes, all units are floating point (C++ double) so decimals can be used. Because the Joint Controller takes care of the differential drives the second test GUI does fully support differential drives.

The User Manual for the Test GUI can be found in Appendix E.

# 8 Calibration of the shoulder joints

The shoulder joints in the first version of the robot arm, without the variable stiffness modules, do not have any absolute encoders to determine where they are. The position information provided by the motors is a relative position and can only be used when a known reference is available to calibrate the relative position to an absolute position. The determine the absolute position of the lower rotation requires an external reference. The differential drive has mechanical limits that could be used by detecting current increase or a constant relative position during velocity movement. This approach did not work well as the the motors in the differential drive are so powerful they pulled the cable from the cable clamps, and the differential drive rotation is mechanically limited only after the cables are already moving outside of their routed guides. To solve this all reference positions are created by micro switches that are wired to the Elmo motor controllers.

The lower rotation joint calibration switch is placed in a mount that is secured to the base of the shoulder, a carriage bolt is added to the rotating differential drive and presses the micro switch when it is position directly above the switch. The configuration can be seen on the left picture in Figure 8.1.

For the differential drive switches for both ends of the joint rotations are used and wired to the motor controllers. The motor controllers are programmed so that they will stop motion when one of the end stop switches is pressed to prevent the motors for pulling the steel cables from the end caps. For the up/down rotation both movement extremes are detected with switches with a hinge lever. The configuration with the hinge switches can be seen on the left picture in Figure 8.1. The upper rotation joint also has both movement extremes detected by switches that are mounted directly to the aluminium bar, that connects both sides of the differential drive. These switches are pressed by a lever that is secured to the rotating shaft. This configuration can be seen on the right picture in Figure 8.1.

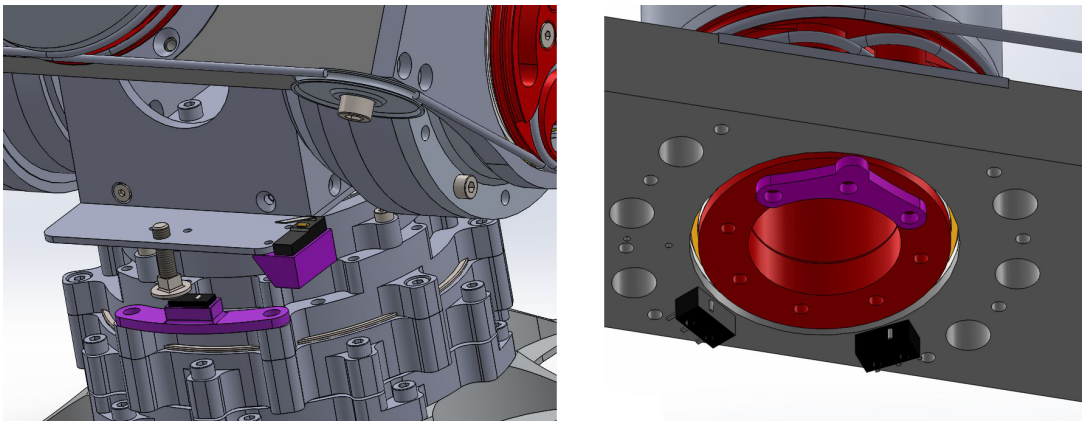

**Figure 8.1:** Position references with micro switches for the shoulder

The mounts for the switches and the lever for the upper rotation joint are 3D printed, the design was created with Solidworks. The Getting started with Solidworks tutorials (Dassault Systemes, 2015) were used to gain the required knowledge to design these.

# 9 Tests and validation

## 9.1 Shoulder rotation, non differential drive

To determine the response of the arm and software ROS Topic data was logged from the Joint Controller. In the following graphs the control command is sent on time t=0. The time between the send control command and the arm movement includes the control messages going from Joint Controller to ROS-CAN interface to the CAN bus and the Elmo. The position data is what is received from the Joint Controller, so its the position data that has been send from the Elmo to the CAN bus to the ROS-CAN interface and the Joint Controller.

In Figure 9.1 at time t=0 a control command was send that requests a position movement to 1.0 rad from the center position. The Elmo is not currently in position mode, so first the motor is turned off, the mode changed, and finally the motor is turned on. The software is programmed to wait 500ms after a motor turn on or turn off command. The Elmo discard any messages send shortly after turning the motor on or off, to prevent losing messages the software will stop sending any messages to an Elmo that has just been send a motor on or off command and queue the control messages for sending after this time. In position mode the Elmo will try to move the joint to the requested angle as fast as possible. In the current test setup a 25V 1A powersupply is used, during the fast movements the voltage drops down as it is unable to provide enough energy. This clearly influences the smoothness of the movement.

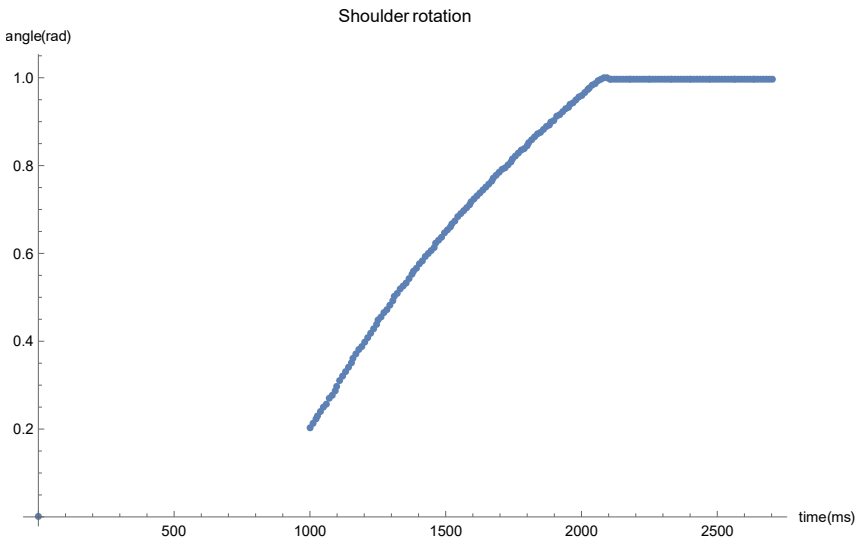


**Figure 9.1:** Position Control

In Figure 9.2 at time t=0 a control command was send that requests a position movement to -1.0 rad from the center position. The Elmo is currently in position mode, so no mode switch is necessary and no delay is introduced by the software.

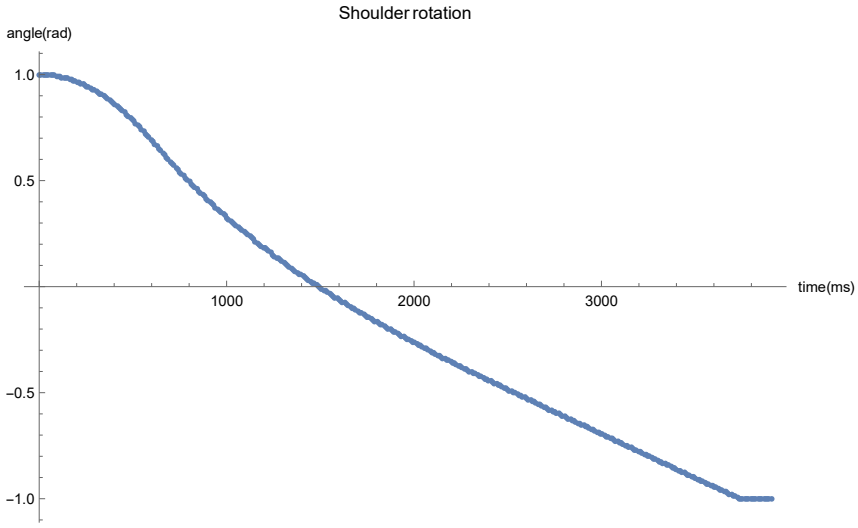


**Figure 9.2:** Position Control

In Figure 9.3 at time t=0 a control command was send that requests a velocity movement with a low speed of 0.1 rad/s. The Elmo is currently in position mode, so a mode switch is necessary and the delay is introduced by the software. Due to the low speed the power supply does deliver enough power which results in a much smoother movement.

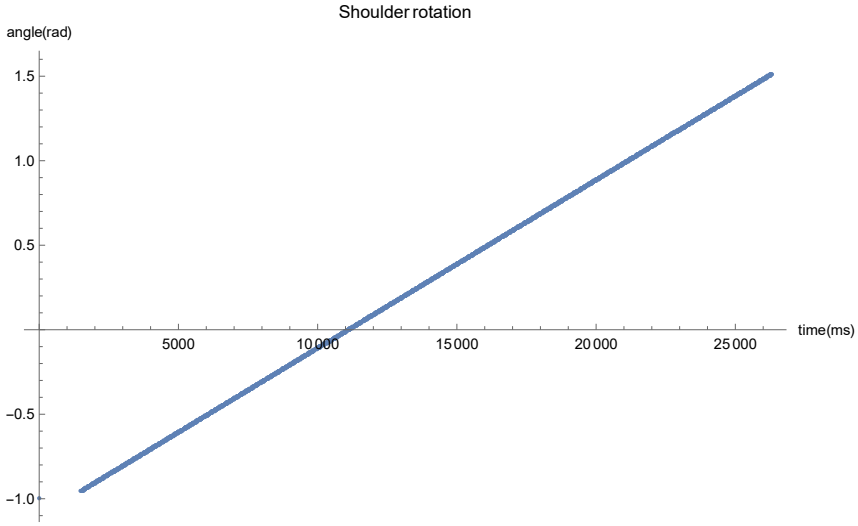


**Figure 9.3:** Velocity Control

The first test is repeated with a powersupply than can deliver over 20A, the measurement data is plotted in Figure 9.4 The 20A powersupply improves the arm rotation as it stays at a higher speed (orange) for a longer time. A linear interpolation function is created with Mathematica (Wolfram, 2015) of both measurements, to match the data points a 9th order function was used. The linear interpolation and its derivative (velocity) are plotted in Figure 9.5. The velocity with the 1A powersupply (yellow) is dropping after around 100ms while the velocity with the 20A powersupply (orange) is reasonably constant until the Elmo decelerates the movement after around 400ms. The blue line is the postion with the 1A powersupply and the green line with the 20A powersupply.

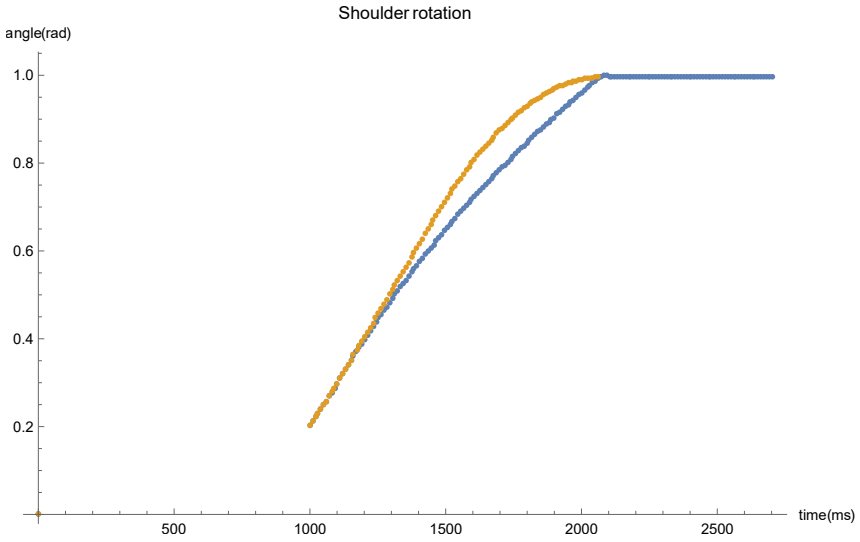


**Figure 9.4:** Position Control, Measurement Data

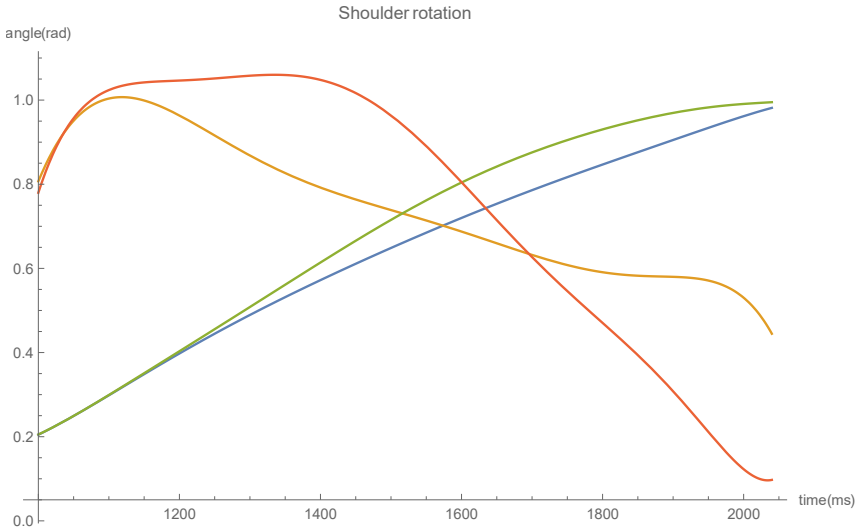


**Figure 9.5:** Position Control, Linear Interpolation

The velocity test is repeated with the 20A powersupply, a speed of -0.5 rad/s, the measurement data is plotted in Figure 9.6. The linear interpolation (blue) and its derivative (velocity, yellow) are plotted in Figure 9.7. The velocity is very close to the set velocity and it is quite constant.

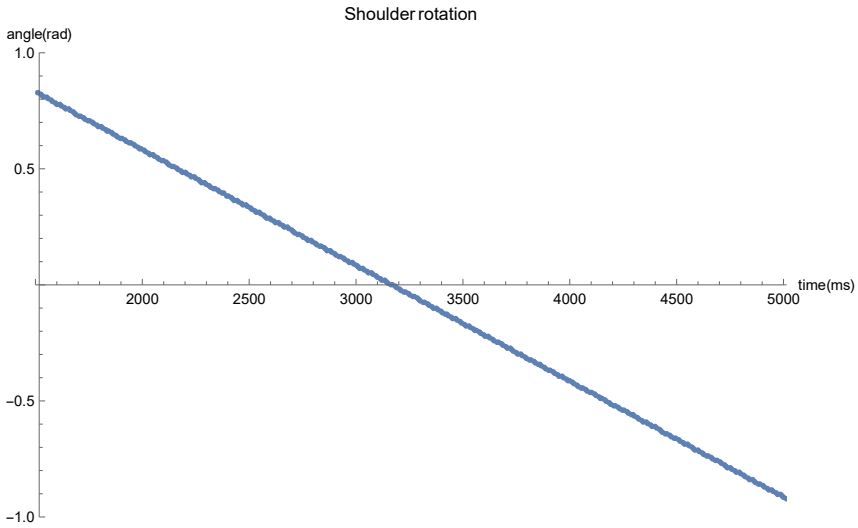


**Figure 9.6:** Velocity Control, Measurement Data

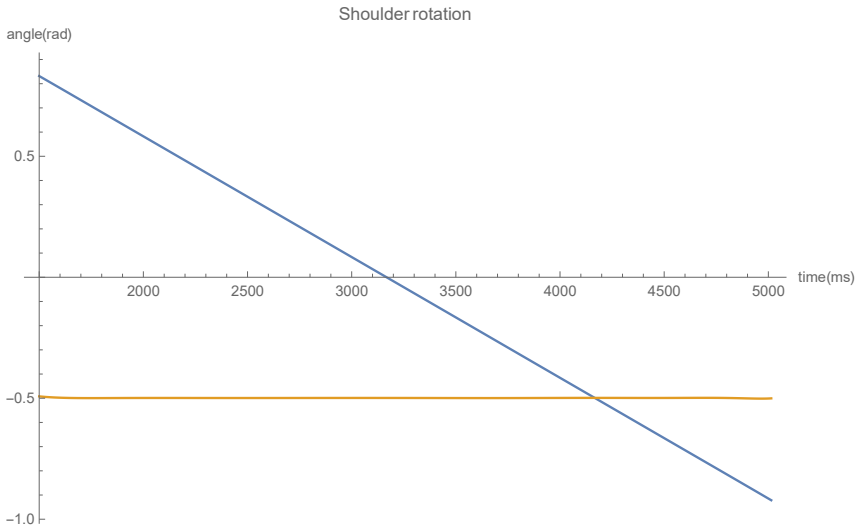


**Figure 9.7:** Velocity Control, Linear Interpolation

Figure 9.8 shows the measurement data of a torque control input of 1A. The corresponding linear interpolation (blue) and its derivative (velocity, yellow) are plotted in Figure 9.9. The velocity at 1A is close to 0.5 rad/s but it is not very constant. A 1A powersupply is not enough to ensure a speed of over 0.5 rad/s for the shoulder rotation, this is also seen in Figure 9.4 where the 1A powersupply drops the rotational speed (yellow) while the same command with the 20A powersupply does not show the same behaviour.

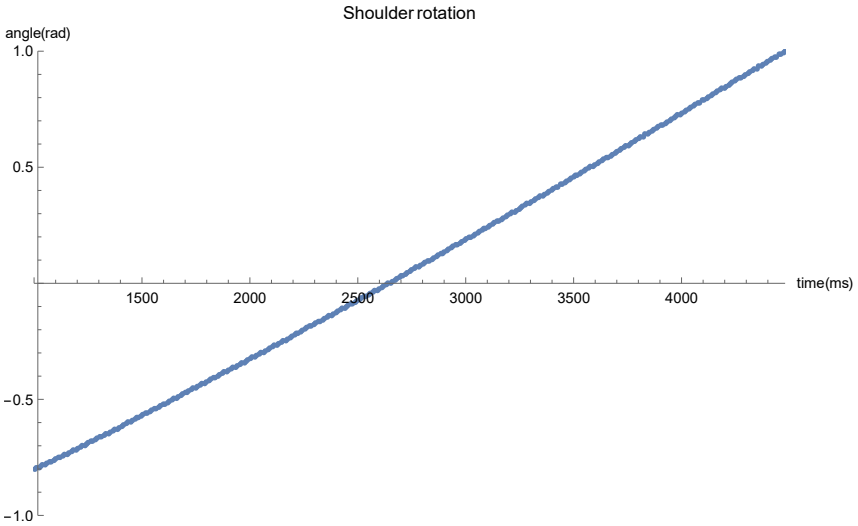


**Figure 9.8:** Torque Control, Measurement Data

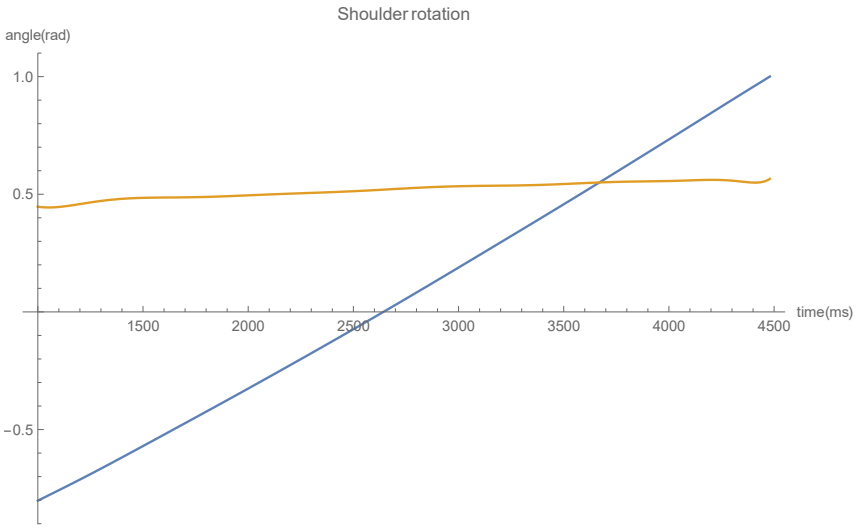


**Figure 9.9:** Torque, Linear Interpolation

In Figure 9.10 at time t=0 a control command was send to joint 1 (the joint that moves when both motors of the differential drive move in the same direction, further called the $\Sigma$-joint) to move to the center position. In the figure both joint positions of the differential drive are displayed.

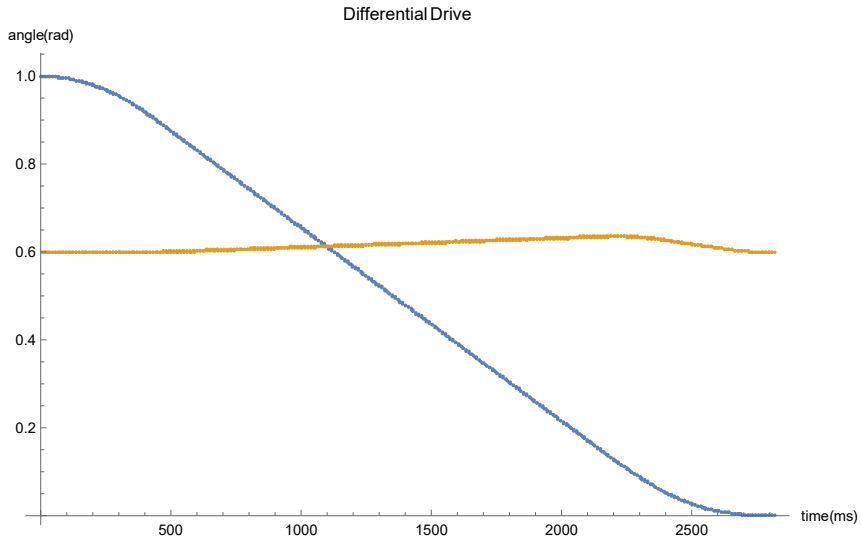


**Figure 9.10:** Differential drive position control

In Figure 9.10 the differential drive should only move the $\Sigma$-joint, but joint 2 ((the joint that moves when both motors of the differential drive move in opposite direction, further called the $\Delta$-joint) is also moving a bit though it ends at the position it started from. The problem is that both parts of the differential drive are not exactly the same and one of the drives is moving a bit faster than the other one, which results in the movement of the $\Delta$-joint.

In Figure 9.11 at time t=o a control command was send to joint 2 (the $\Delta$-joint of the differential drive) to move to position -1 rad. In the figure both joint positions of the differential drive are displayed.

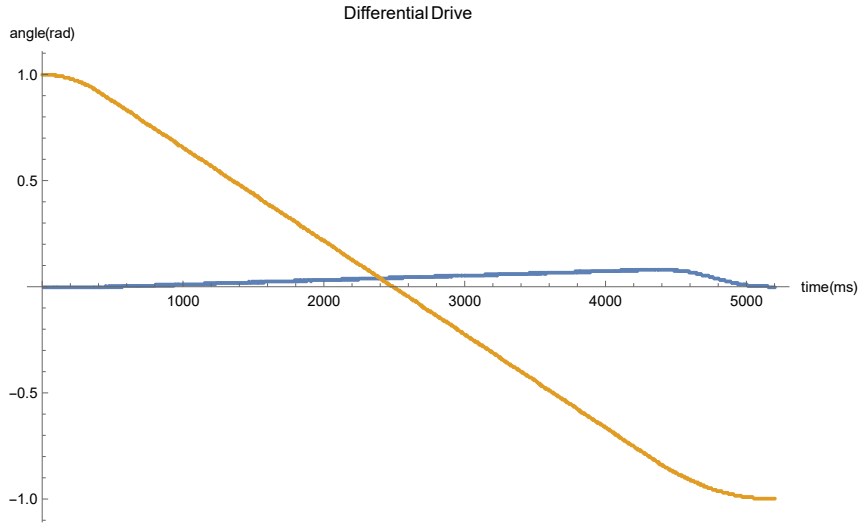


**Figure 9.11:** Differential drive position control

In Figure 9.11 the differential drive should only move the $\Delta$-joint, but the $\Sigma$-joint is also moving a bit though it ends at the position it started from. This is the same problem as seen in Figure 9.10.

# 10 Conclusion and future work

## 10.1 Conclusion

In this assignment all the tasks from the initial task list found in Chapter 2 have been completed with two exceptions. The problem with the differential drive movements where the movement of the $\Sigma$-joint temporarily influences the position of the $\Delta$-joint and the other way around (See Figure 9.10 and Figure 9.11 is something that needs additional work. The other problem is that the current software only supports Elmo motor controllers on the CAN bus and support for the force/torque sensor and absolute encoders is missing. Work has been done to make adding these devices relatively easy, the existing code will only require minor changes and almost all device specific code can be added in a separate device file. The current Elmo implementation also resides in a separate file that can be used as a template to add new device support. The Joint Controller node is only necessary for the Elmo motor controllers and does not need to be changed.

## 10.2 Future work

The electronics decisions made in this assignment will need to be rechecked. Changes in the existing components and addition of components can influence the choices made in this assignment. The software made in this assignment will need to be extended and improved on before it can be used on the Sherpa arm. Some software components that are required for the arm to work are not yet created. To continue work on this project the following task lists should be considered for new assignments.

| Task | Description and motivation |
|---|---|
| DC-DC Converter evaluation | The DC-DC converters selection should be evaluated when all components have been chosen, if more components are introduced that require either 5V or 12V a more powerful converter might be required. |
| DC-DC Converter external capacitors | The DC-DC converters require external capacitors for stable operation, this can be done by designing a PSB that contains both converters and the capacitors. |
| Elmo PCB design | The Elmo Whistles are connected to an existing prototype PCB, this PCB is very large and hinders integration of the Elmo's in the arm. To reduce spikes on the voltage bus a few small but high-capacity capacitors should be integrated onto the PCB. |

**Table 10.1:** Future work task list, electronics

| Task | Description and motivation |
|---|---|
| Improving differential drive performance | The behavior of the differential drives is not very good due to the differences in both joints. There are two ways to improve this. First better Elmo PI Controller settings maybe found that decrease the difference in speed between both joints and second the joint controller differential drive handling can be improved to add feedback from the position output into the control loop. |
| Extending device support | Currently the software only fully supports the Elmo motor controller, Force/Toque sensor support and absolute encoder support can be added if required. The current development version of the shoulder doesn't have these. Preliminary work has been done to support these so the existing code needs to be changed only on a few places, most new code can be done in a separate device file. |

**Table 10.2:** Future work task list, existing software

| Task | Description and motivation |
|---|---|
| Central Controller development | The Central Controller is the most important missing piece of software as it connects all in- and outputs together and makes the decisions on how the arm will move. |
| VI Interface | The VI interface software node needs to give useful input about the environment to the Central Controller, to let the arm move around obstacles and find the exact location of the UAVs before gripping and docking them. |
| Communication Controller | The Communication Controller will need to translate the incoming data from the rest of the UAVs, rover and Sherpa box (task requests and world model data) to a format that can be handled by the central controller. It also needs to send data back (task request, task acknowledgments and arm position data). |

**Table 10.3:** Future work task list, new software

# A Component list

| Component | Brand | Model | Voltage | Current | Power |
|---|---|---|---|---|---|
| Computer | Intel | NUC D54250WYK | 12 | 5.5 | 65 |
| Memory | Crucial | CT51264BF160BJ | - | - | - |
| Storage | Kingston | SSD 120GB mSATA | - | - | - |
| LAN mPCI-e | Jetway | ADMPEIDLA i350 | - | - | - |
| VI Sensor | Skybotix | VI Sensor | 12 | 0.83 | 10 |
| USB-CAN 1 | EMS | CPC-USB/ARM7 | - | - | - |
| USB-CAN 2 | EMS | CPC-USB/ARM7 | - | - | - |
| USB-CAN 3 | EMS | CPC-USB/ARM7 | - | - | - |
| F/T Sens 6 DoF | ATI | Net F/T | 12 | 0.83 | 10 |
| MC shoulder 1a | Elmo | Whistle | 48 | ? | ? |
| MC shoulder 1b | Elmo | Whistle | 48 | ? | ? |
| MC shoulder 2a | Elmo | Whistle | 48 | ? | ? |
| MC shoulder 2b | Elmo | Whistle | 48 | ? | ? |
| MC shoulder 3a | Elmo | Whistle | 48 | ? | ? |
| MC shoulder 3b | Elmo | Whistle | 48 | ? | ? |
| MC elbow 1a | Elmo | Whistle | 48 | ? | ? |
| MC elbow 1b | Elmo | Whistle | 48 | ? | ? |
| MC wrist 1a | Elmo | Whistle | 48 | ? | ? |
| MC wrist 1b | Elmo | Whistle | 48 | ? | ? |
| MC wrist 2a | Elmo | Whistle | 48 | ? | ? |
| MC wrist 2b | Elmo | Whistle | 48 | ? | ? |
| MC wrist 3a | Elmo | Whistle | 48 | ? | ? |
| MC wrist 3b | Elmo | Whistle | 48 | ? | ? |
| Motor shoulder 1a | Kollmorgen | RBE 02110-A | 48 | 5 | 241 |
| Motor shoulder 1b | ? | ? | 48 | ? | ? |
| Motor shoulder 2a | TQ Group | Robodrive 70x18 | 48 | 7 | 370 |
| Motor shoulder 2b | ? | ? | 48 | ? | ? |
| Motor shoulder 3a | TQ Group | Robodrive 70x18 | 48 | 7 | 370 |
| Motor shoulder 3b | ? | ? | 48 | ? | ? |
| Motor elbow 1 | TQ Group | Robodrive 50x14 | 48 | 3.5 | 145 |
| Motor elbow 1b | ? | ? | 48 | ? | ? |
| Motor wrist 1a-3b | ? | ? | ? | ? | ? |
| Abs Enc. shoulder 1 | RLS | AksIM | 5 | 0.15 | 0.75 |
| Abs Enc. shoulder 2 | RLS | AksIM | 5 | 0.15 | 0.75 |
| Abs Enc. shoulder 3 | RLS | AksIM | 5 | 0.15 | 0.75 |
| Abs Enc. elbow 1 | RLS | AksIM | 5 | 0.15 | 0.75 |
| Abs Enc. wrist 1 | RLS | AksIM | 5 | 0.15 | 0.75 |
| Abs Enc. wrist 2 | RLS | AksIM | 5 | 0.15 | 0.75 |
| Abs Enc. wrist 3 | RLS | AksIM | 5 | 0.15 | 0.75 |
| Gripper actuator | ? | ? | ? | ? | ? |

**Table A.1:** Component list

# B DC-DC converter list

| Brand | Model | Efficiency @ 48V | | | Load | | Ripple | Size |
|---|---|---|---|---|---|---|---|---|
| | | 2.5A | 5A | 10A | Min | Max | Typ. pp | mm |
| GE | EHHD010A0B41Z | 87% | 92% | 92% | 0A | 10A | 200mV | 58x23x9 |
| GE | EHHD010A0B41HZ | 87% | 92% | 92% | 0A | 10A | 200mV | 58x23x13 |
| GE | EVK011A0B41Z | 90% | 93% | 95% | 0A | 11A | 100mV | 58x23x9 |
| GE | EHHD020A0B41Z | 85% | 93% | 95% | 0A | 20A | 200mV | 58x23x9 |
| GE | EHHD020A0B41HZ | 85% | 93% | 95% | 0A | 20A | 200mV | 58x23x13 |

**Table B.1:** 12V DC-DC Converters

| Brand | Model | Efficiency @ 48V | | | Load | | Ripple | Size |
|---|---|---|---|---|---|---|---|---|
| | | 2.5A | 5A | 10A | Min | Max | Typ. pp | mm |
| Murata | NCS12S4805C | 65% | 78% | 86% | 0.24A | 2.4A | 70mV | 32x20x10 |
| GE | KHHD006A0A41Z | <70% | 77% | 86% | 0A | 6A | 60mV | 33x23x9 |
| GE | KSTW006A0A41Z | <70% | 77% | 86% | 0A | 6A | 60mV | 33x23x9 |
| TracoPower | TEN 8-4811 | ? | ? | N/A | ? | 1.5A | 50mV | 32x20x10 |
| TracoPower | TEN 8-4811WI | ? | ? | N/A | ? | 1.5A | 50mV | 32x20x10 |
| GE | KSTW010A0A41Z | <70% | 72% | 84% | 0A | 10A | 60mV | 33x23x9 |
| Murata | UEI15-050-Q48P-C | 65% | 76% | 84% | 0A | 3A | 60mV | 28x25x8 |
| XP Power | JCJ0848S05 | ? | ? | N/A | 0A | 1.5A | 75mV | 32x20x10 |
| XP Power | JCA0848S05 | ? | ? | N/A | 0A | 1.6A | 50mV | 26x20x10 |
| TracoPower | THN 15-4811 | 60% | 74% | 85% | 0A | 3A | 100mV | 25x25x10 |
| XP Power | JCJ1048S05 | ? | ? | N/A | 0A | 2A | 75mV | 32x20x10 |
| XP Power | JTF1048S05 | ? | ? | N/A | 0A | 2A | 85mV | 32x20x10 |
| TracoPower | THN 15-4811WI | 60% | 74% | 85% | 0A | 3A | 100mV | 25x25x10 |

**Table B.2:** 5V DC-DC Converters

# C Elmo motor controller programming

Before the software can drive a motor the Elmo motor controller should be programmed. The basic programming to setup the Elmo with the motor should be done with the Elmo Composer (Elmo Motion Control Ltd., 2013) software that is provided by ElmoMC for its motor controllers. The wizard should be run to determine basic settings and the direction of movement. The velocity and position PI controllers should be tuned with Elmo Composer so that it operates well and will allow a speed command to operate the motor close to the set speed and allow the position command to work correctly with both large and very small movement. (A position movement that sets the new position to 1 count more or less should work correctly and so should a position movement that moves the motor from one extreme to the other.

## C.1 Example1

The limits should be programmed in both the Elmo and in the software parameter list. To do this the following example is used: A joint with an end detection switch on each end, a full rotation of 600 steps or counts, minimum position of -220 counts, maximum position of 220 counts, low end switch connected to input 3 and high end switch connected to input 4. See Figure C.1. When the joint hits an end detection switch the software will stop the motion and set the joint within the normal operating range. When the joint can physically overshoot the position of the end detection switch, the Elmo absolute minimum and maximum position should be set to this range instead of the position end detection switch range.

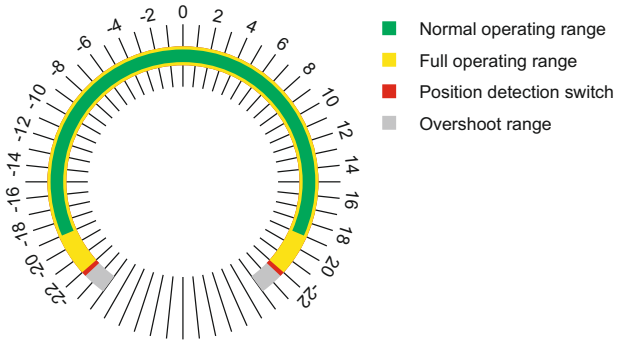


**Figure C.1:** Position limitations of a joint with end detection switches on each end of the operating range. (Scale 1:10)

### C.1.1 Position limits

The Elmo has a low and high position limit (LL[3] and HL[3]) these should be set to the extremes of the overshoot range, because the Elmo will not accept any movement commands when beyond these positions. In the example from figure C1 LL[3]= -240 and HL[3]= 240.

Next to the absolute minimum and maximum position the are also minimum and maximum command input positions (VL[3] and HL[3]). These should be set to the extremes of the full operating range. Any input command with a position outside of these limits will are truncated to these limits. In the example from figure C1 VL[3]= -220 and VH[3]= 220.

The position limits do not work in toque mode! The end detection switches programmed as stop switches do limit the movement as does the software.

### C.1.2 Velocity limits

The velocity minimum and maximum command inputs (VL[2] and VH[2]) truncate any input command with a velocity outside of these limits. These should be set at the maximum speed allowed for all mechanical components (take into account any transmission and conversion of RPM to counts per second) or the maximum wanted application speed, whichever is lower.

The Elmo has a low and high velocity limit (LL[2] and HL[2]) If the Elmo detects a velocity outside of these limits the motor is stopped and a software reset is necessary to continue operation. So these limits should be used as a safeguard only and set higher than the velocity minimum and maximum command input limits.

The velocity limits do not work in toque mode! In toque mode there is no velocity limit at all.

### C.1.3 Acceleration/Deceleration limits

The maximum acceleration and deceleration can be set in counts per squared second with the AC and DC commands.

Acceleration/Deceleration command only work in position and velocity modes.

### C.1.4 Current limits

Current limits should be set according to motor, Elmo motor controller or power supply current limits, whichever is lower. The continuous current can be set in A with CL[1], the peak current with PL[1] and the peak duration in seconds in PL[2]

Current limits work in all operation modes.

### C.1.5 Inputs

To set the end detection switches as stop motion switches on the Elmo the corresponding inputs should be set. This is done with command IL[n]= 21 where n is the input number. The input should be low when the switch is not pressed. In the example from figure C1 IL[3]= 21 and IL[4]= 21.

### C.1.6 Saving the parameters

When all limits are set and send to the Elmo they stay only until the next reset or power cycle of the Elmo. To permanently save this to the flash memory use the SV command to save the memory to flash. Do not do this when trying out different values as saving to many times will destroy the flash and make the Elmo useless.

### C.1.7 Software limits

For the software configuration the limits should be set in the ROS parameter server. See the Software Usage Manual for instructions how to do this. The limits that are needed are for the software in the above example are listed in Table C.1.

| Parameter Name | Description | Value |
|---|---|---|
| cpr: | Counts per round | 600 |
| pos_min: | Minimum position | -220 |
| pos_max: | Maximum position | 220 |
| pos_margin: | Difference between normal and full operating range | 30 (=220-190) |
| cal_sw_L: | Input number of the low position end switch | 3 |
| cal_sw_H: | Input number of the high position end switch | 4 |

**Table C.1:** ROS Params for example 1

## C.2 Example2

In this example the same setup is used as in example 1 with one change, the end detection switches are removed and one position detection switch is used on position 20 and connected to input 1, see Figure C.2.

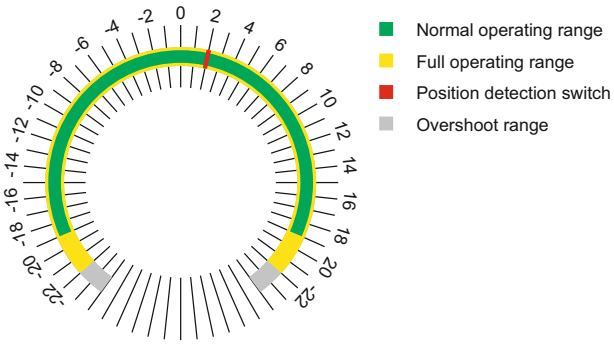


**Figure C.2:** Position limitations of a joint with a position detection switch within the operating range. (Scale 1:10)

The configuration of the limits of the Elmo do not change, the only change is the input configuration, IL[1]= 5. For the software cal_sw_L should be set to 1, cal_sw_H should not be set and cal_pos should be set to 20

## C.3 Example3

In this example a differential drive pair is used. Two motors drive two joints together, the first joint reacts to the sum of both motor positions so it can be called the sigma joint or $\Sigma$-joint, the second joint reacts to the difference of both motor positions so it can be called the delta joint or $\Delta$-joint. When using a differential drive end detection switches must be used on the $\Sigma$-joint or $\Delta$-joint because the first motor will have a minimum and maximum that depends on the other motor and the other way around. In this example the $\Sigma$-joint has a full rotation of 600 steps or counts, minimum position of -220 counts, maximum position of 220 counts, low end switch connected to input 1 and high end switch connected to input 2. The $\Delta$-joint has a full rotation of 600 steps or counts, minimum position of -160 counts, maximum position of 160 counts, low end switch connected to input 3 and high end switch connected to input 4. See Figure C.3.

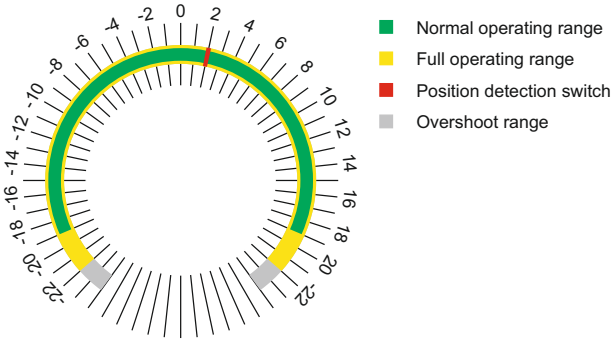


**Figure C.3:** Position limitations of the $\Sigma$-joint (left) and the the $\Delta$-joint (right) of example 3

To determine the motor command limits a graph is made with the first motor position on the horizontal axis and the second motor position on the vertical axes together with the four joint

limits. See figure C5. When the limits of both joints are equal the motor position limits are equal to the joint limits, in this example the limits of both joints are different. To determine the motor command limits take the difference of both maximums and divide it by two (30 = (220-160)/2). Subtract this number from the higher maximum to get the maximum position for both motors and invert the maximum to get the minimum. This only works when both joints have symmetrical limits. The same approach can be used to determine the absolute motor limits from the overshoot range.

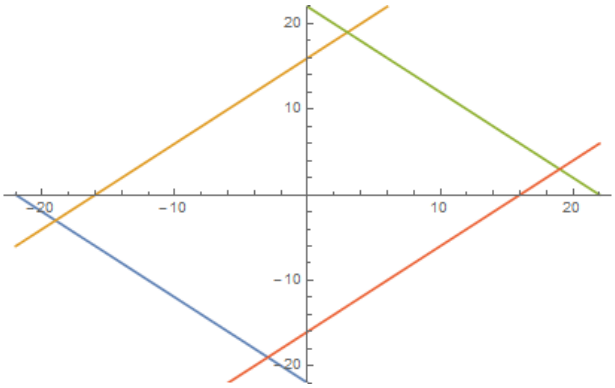


**Figure C.4:** Position limitations of the motors of example 3 (Scale 1:10)

In the example from figure C3 and C4 LL[3]= -205 and HL[3]= 205 for both motors (205=240-(240-170)/2), VL[3]= -190 and VH[3]= 190. The inputs should be set to stop motion, so IL[1]= 21, IL[2]= 21, IL[3]= 21, and IL[4]= 21. The other limits should be set with the information given in example 1. The software configuration for both motors of example 3 are listed in Table C.2

| **Parameter Name** | **Description** | **Value** |
|---|---|---|
| cpr: | Counts per round | 600 |
| pos_min: | Minimum position (of full operating range) of the $\Sigma$-joint | -220 |
| pos_max: | Maximum position (of full operating range) of the $\Sigma$-joint | 220 |
| pos_margin: | Difference between normal and full operating range of the $\Sigma$-joint | 30 (=220-190) |
| pos_min_dd: | Minimum position (of full operating range) of the $\Delta$-joint | -160 |
| pos_max_dd: | Maximum position (of full operating range) of the $\Delta$-joint | 160 |
| pos_margin_dd: | Difference between normal and full operating range of the $\Delta$-joint | 10 (=170-160) |
| cal_pos: | Position to be set after reaching calibration switch | -220 |
| cal_sw_L: | Input number of the low position end switch of the $\Sigma$-joint | 1 |
| cal_sw_H: | Input number of the high position end switch of the $\Sigma$-joint | 2 |
| cal_sw_dd_L: | Input number of the low position end switch of the $\Delta$-joint | 3 |
| cal_sw_dd_H: | Input number of the high position end switch of the $\Delta$-joint | 4 |

**Table C.2:** ROS Params for example 3

# D Software User Manual

## D.1 Configuration file

Before being able to start the software the configuration parameters must be loaded into the ROS parameters server. To do this a configuration file is made that can be loaded by the ROS parameter server.

Here is an example of a ROS parameter server file to configure the software:

```
ros_can_interface:
  topics: [/joint]
  interfaces: [can0, can1]
  queue_size: [100, 100]
  send_interval: [1000, 1000]
  scheduler_interval: [10, 100]
  nodes:
    '0': {topics: joint/shoulder/joint_mc/0, dev_bus: can0, dev_id: 10,
dev_type: 1, cpr: 3600,    pos_min: -1750, pos_max: 1750,
pos_margin: 36, cal_speed: 300, cal_pos: 0}
    '1': {topics: joint/shoulder/joint_mc/1, dev_bus: can0, dev_id: 20,
dev_type: 1, cpr: 6000, pos_min: -1500, pos_max: 1500, pos_margin: 60,
diffdrive: 2, pos_min_dd: -1500, pos_max_dd: 1500, pos_margin_dd: 60,
cal_pos: -1450, cal_sw_L: 1, cal_sw_H: 2, cal_sw_dd_L: 3, cal_sw_dd_H: 4}
    '2': {topics: joint/shoulder/joint_mc/2, dev_bus: can0, dev_id: 30,
dev_type: 1, cpr: 6000, pos_min: -1500, pos_max: 1500, pos_margin: 60,
diffdrive: 1, pos_min_dd: -1500, pos_max_dd: 1500, pos_margin_dd: 60,
cal_pos: -1450, cal_sw_L: 1, cal_sw_H: 2, cal_sw_dd_L: 3, cal_sw_dd_H: 4}
    '3': {topics: joint/shoulder/joint_mc/0, dev_bus: can1, dev_id: 40,
dev_type: 1, cpr: 3600, pos_min: -1750, pos_max: 1750, pos_margin: 36,
ext_pos_input: joint/shoulder/joint4/position}
```

The first line contains the param path, do not change this from the example, all other lines are indented two spaces to define that those are inside the param path.

| Parameter Name | Description |
|---|---|
| topics: | base topic path |
| interfaces: | list of can interfaces |
| queue_size: | the message size for the send and receive queue per can bus |
| send_interval: | interval between each send message for each CAN bus in microseconds (us) |
| scheduler_interval: | interval between all polled data for each device per CAN bus in milliseconds (ms) (For Elmo motor controllers this is the interval between the current position polling |
| nodes: | list of all devices/nodes, all nodes are numbered from 0 to 127 and should be provided in ascending order. To define that the nodes are inside the nodes: list the nodes are indented another two spaces, so four in total. |

**Table D.1:** Software configuration parameters, generic

The nodes itself have a few parameters, some of these parameters are required, other are optional.

### D.1.1 All devices

| Parameter Name | Required? | Description |
|---|---|---|
| topics: | Y | base path of in- and output messages for this node |
| dev_bus: | Y | the CAN bus to which this device/node is connected |
| dev_id: | Y | CAN ID of the device/node |
| dev_type: | Y | the device/node type and name of its in- and output messages, library file and other device specific functionality. |

**Table D.2:** Software configuration parameters, nodes

Currently only dev_type 1 which is ElmoMC SimplIQ Servo Drives are supported.

### D.1.2 Non-differential drives

| Parameter Name | Required? | Description |
|---|---|---|
| cpr: | Y | Counts per round, the amount of counts/steps for a whole round |
| pos_min: | Y | Counts, Minimum position (of full operating range) |
| pos_max: | Y | Counts, Maximum position (of full operating range) |
| pos_margin: | Y | Counts, Difference between normal and full operating range |
| cal_speed: | N | Counts per second, velocity with which the joint is moved towards the switch during calibration, if not specified ext_pos_input should be set |
| cal_pos: | N | Counts, Position to be set after reaching calibration switch, if not specified pos_min will be used or ext_pos_input should be set |
| ext_pos_input: | N | Rostopic location, if an absolute encoder is used this settings specifies the ros topic location to which the current position is advertised, it should be done as a standard ROS message type of type int. If not specified both cal_pos and cal_speed should be set |
| cal_sw_L: | N | Input number of the low position end switch, or single position detection switch |
| cal_sw_H: | N | Input number of the high position end switch |

**Table D.3:** Software configuration parameters, non-differential nodes

### D.1.3 Differential drives

| Parameter Name | Required? | Description |
|---|---|---|
| cpr: | Y | Counts per round, the amount of counts/steps for a whole round |
| pos_min: | Y | Counts, Minimum position (of full operating range) of the $\Sigma$-joint |
| pos_max: | Y | Counts, Maximum position (of full operating range) of the $\Sigma$-joint |
| pos_margin: | Y | Counts, Difference between normal and full operating range of the $\Sigma$-joint |
| diffdrive: | Y | Device/node number of the corresponding drive of this differential pair |
| pos_min_dd: | N | Same as pos_min though for the $\Delta$-joint, if not specified the pos_min value is used |
| pos_max_dd: | N | Same as pos_max though for the $\Delta$-joint, if not specified the pos_max value is used |
| pos_margin_dd: | N | Same as pos_margin though for the $\Delta$-joint, if not specified the pos_margin value is used |
| cal_pos: | N | Counts, Position to be set after reaching calibration switch, if not specified pos_min will be used |
| cal_sw_L: | N | Input number of the low position end switch of the $\Sigma$-joint |
| cal_sw_H: | N | Input number of the high position end switch of the $\Sigma$-joint |
| cal_sw_dd_L: | N | Input number of the low position end switch of the $\Delta$-joint |
| cal_sw_dd_H: | N | Input number of the high position end switch of the $\Delta$-joint |

**Table D.4:** Software configuration parameters, differential nodes

For each differential drive pair a set of independent joints is created, the first joint will have the number equal to the lower node number and is called the $\Sigma$-joint because it responds to the sum of both inputs. The second joint will have the number equal to the higher node number and is called the $\Delta$-joint because it responds to the difference of both inputs.

## D.2 Compiling the source files

1. Copy the source files to the src directory of your catkin workspace

```
cp -r /path-to-sourcefiles/ /path-to-catkin_workspace/src
```

2. Build the ROS nodes

```
cd /path/catkin_workspace
catkin_make
```

## D.3 Running the software

1. Start the canbus (replace the bitrate number with the speed the canbus should run at)

```
ip link set can0 type can bitrate 1000000 listen-only off
ifconfig can0 up
```

2. Start ros

```
roscore
```

3. Load ros parameters in a new terminal (substitute the filename for the one that you created before)

```
rosparam load /path/rosparam-file
```

4. Start the nodes

```
cd /path/catkin_workspace
source devel/setup.bash
rosnode ros_can_interface ros_can_interface_node
rosnode joint_controller joint_controller_node
```

5. Wait for the node to startup, the node is ready after a summary of all nodes is given and the schedulers are started

6. Use rosparam list so see all available topics registered by the node

```
rosparam list
```

7. Start the test gui

```
cd /path/catkin_workspace
rosnode test_gui2 test_gui2_node
```

8. Using the joystick as input

```
cd /path/catkin_workspace
rosparam set joy_node/dev "/dev/input/js0"
rosnode joy joy_node
```

## D.4 Rostopics and messages usage

### D.4.1 Messages between the Joint Controller and the Central Controller Nodes

| Topic Name | Message Type | Direction | Message contents |
|---|---|---|---|
| get_status | std_msgs/bool | Central Controller -> Joint Controller | bool data |
| status | ros_can_interface/status | Joint Controller -> Central Controller | bool ok, string status_description |
| control | joint_controller/mc_control | Central Controller -> Joint Controller | bool stop, uint8 mode, float64 data |
| calibration | std_msgs/bool | Central Controller -> Joint Controller | bool data |
| position | std_msgs/float64 | Joint Controller -> Central Controller | float64 data |

**Table D.5:** Messages between the Joint Controller and the Central Controller Nodes

Control is used to send a control command to the Elmo. If bool stop is true the motor will be stopped and the other data in the message will be ignored. The mode sets the Elmo to one of the following modes with the data meaning:

| Mode | Mode description | Contents description |
|---|---|---|
| 1 | Single feedback position mode | destination position in radians |
| 2 | Velocity mode | destination speed in radians per second |
| 3 | Torque/Current mode | current in A |

**Table D.6:** Control modes

Calibration is used to request the joint to be calibrated, the bool data must be set to true.

Position will post a message with the device position in radians with an interval that was set in the rosparams with the reporting_interval variable. Sending of the position messages is initiated after the node gets its first control message, and stopped when the ROS CAN Interface or Joint Controller node is stopped

To check for the status of a node send a get_status message with bool data=true to the get_status topic of the node. The node will then respond by posting a message on the status topic that contains a bool ok, which is true if the device is ok or false otherwise, the string will contain a list of device specific information about the status

Example to use the status messages (use 2 separate terminal windows):

```
cd /path/catkin_workspace
source devel/setup.bash
rostopic echo joint/0/status

cd /path/catkin_workspace
source devel/setup.bash
rostopic pub -1 joint/0/get_status std_msgs/bool true
```

Example to use the control and position messages (use 2 separate terminal windows):

```
cd /path/catkin_workspace
source devel/setup.bash
rostopic echo joint/0/position
```

```
cd /path/catkin_workspace
source devel/setup.bash
rostopic pub -1 joint/0/control joint_controller/mc_control \
-- false 1 0.5
rostopic pub -1 joint/0/control joint_controller/mc_control \
-- true 0 0
```

### D.4.2 Messages between the ROS CAN Interface and the Joint Controller Nodes

| Topic Name | Message Type | Direction | Message contents |
|---|---|---|---|
| get_status | std_msgs/bool | Joint Controller -> ROS CAN Interface | bool data |
| status | ros_can_interface/status | ROS CAN Interface -> Joint Controller | bool ok, string status_description |
| control | joint_controller/mc_control | Joint Controller -> ROS CAN Interface | bool stop, uint8 mode, int32 data |
| calibration | std_msgs/bool | Joint Controller -> ROS CAN Interface | bool data |
| position | std_msgs/float64 | ROS CAN Interface -> Joint Controller | int32 data |

**Table D.7:** Messages between the ROS CAN Interface and the Joint Controller Nodess

Calibration is used to request the motor to be calibrated, the bool data must be set to true. This works only for motors not part of a differential drive pair. The calibration of motors that are part of differential pair is handeld by the Joint Controller, to calibrate the motors of a differential drive pair the calibration request should be send to the joint calibration topic handled by the Joint Controller.

Position will post a message with the device position in counts with an interval that was set in the rosparams with the reporting_interval variable. Sending of the position messages is initiated after the node gets its first control message, and stopped when the ROS CAN Interface node is stopped

To check for the status of a node send a get_status message with bool data=true to the get_status subtopic of the node. The node will then respond by posting a message on the status subtopic that contains a bool ok, which is true if the device is ok or false otherwise, the string will contain a list of device specific information about the status

Example to use the control and position messages (use 2 separate terminal windows):

```
cd /path/catkin_workspace
source devel/setup.bash
rostopic echo joint/shoulder/joint_mc/0/position

cd /path/catkin_workspace
source devel/setup.bash
rostopic pub -1 joint/shoulder/joint_mc/0/control \
ros_can_interface/mc_control -- false 3 200
rostopic pub -1 joint/shoulder/joint_mc/0/control \
ros_can_interface/mc_control -- true 3 200
```

# E Test GUI User Manual

## E.1 TestGUI controlling the ROS-CAN interface

The Test GUI is split in two parts. The upper part shows the information and commands for the individual CAN devices/nodes. The lower part that shows the Joystick control settings. See Figure E.1.

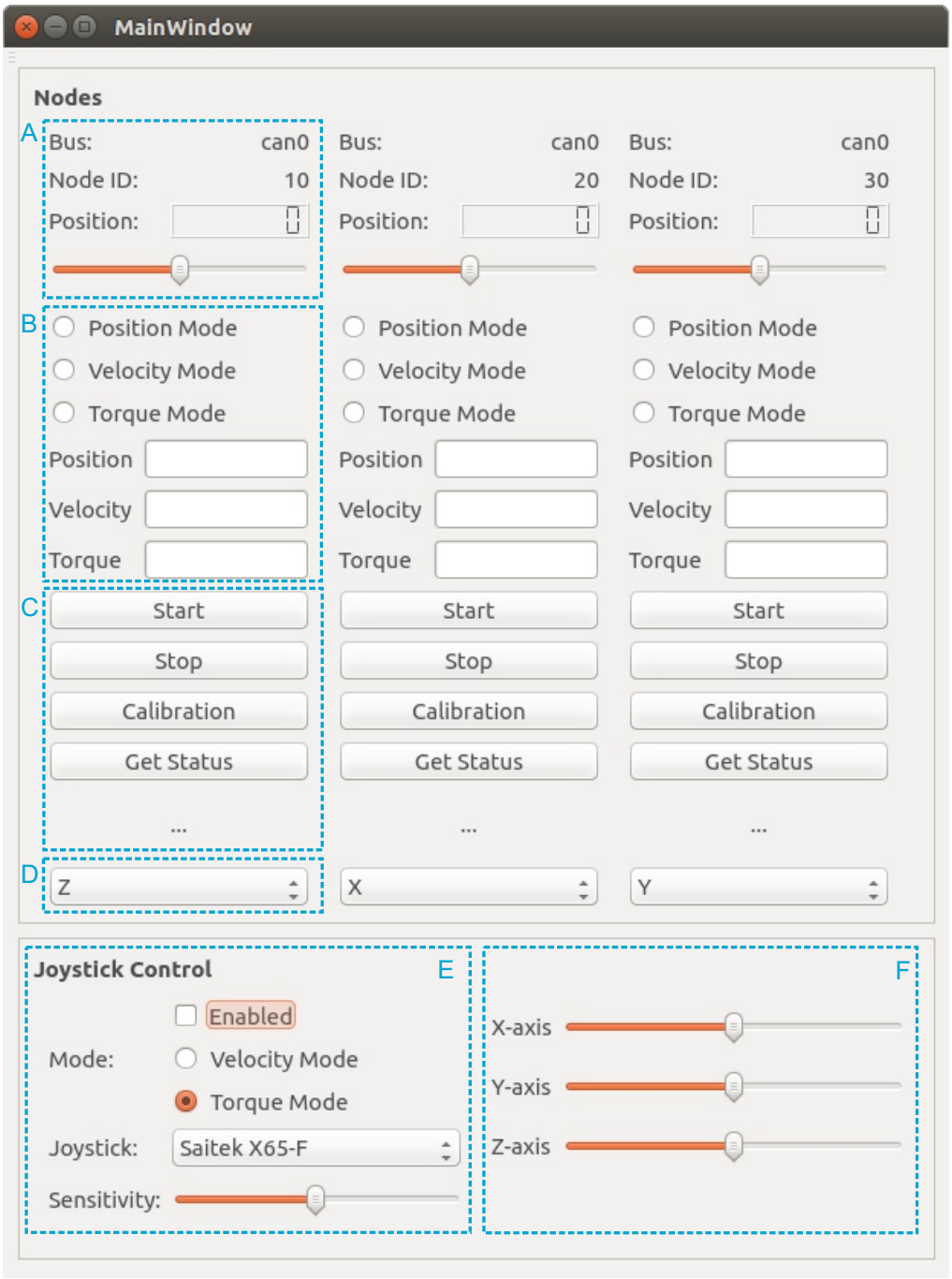


**Figure E.1:** Test GUI

### E.1.1 Nodes

The area marked with A shows the can bus, node ID of the device connected. The Position is given as both a number and a relative position on a slider, the slider limits are taken from the ROS param server (pos_min and pos_max).

The area marked with B has a few settings fields, a radio button that determines the unit mode and for each unit mode an input field. The Test GUI sends and receives messages from the ROS-CAN interface, so units are in counts, counts per second and mA and should be entered as whole numbers

The area marked with C has four control buttons and a status information area. If Start is pressed the GUI wil send a control message to the node with the mode selected with the radio buttons from area B and as data the data from the input field corresponding to the selected mode. The Stop send a control message with a stop command in it. The calibration buttons starts the calibration function and the status button requests the status, only an OK or Fail message is displayed, the detailed status text string is not displayed.

The area marked with D contains the joystick axis that is used to control this node, if joystick control is enabled.

### E.1.2 Joystick Control

The area marked with E shows the joystick control settings. The joystick model can be selected from the drop down menu (the Z axis is sometimes bound to a different axis for different joystick the X and Y are always the same) The sensitivity set with the slider and the control mode can be selected with the radio buttons. The tickbox on top enables or disables joystick control mode.

The area marked with F shows the current joystick position as seen by the software.

## E.2 TestGUI2 controlling the Joint Controllers

The TestGUI2 works and looks almost the same as the TestGUI. Instead of a canbus and node ID it shows the joint number and the units that can be filled in by Position, Velocity and Torque are now in radians, radians per second and A. with the TestGUi2 decimals can be used.

# Bibliography

Alex (2015), LearnCpp.Com.
http://www.learncpp.com/

ATI Industrial Automation (2015), Net F/T.
http://www.ati-ia.com/products/ft/ft_NetFT.aspx

BlueBotics (2013), NIFTi-BlueBotics Librover.
https://github.com/NIFTi-BlueBotics/Librover

CiA (2015), CAN knowledge.
http://www.can-cia.de/can-knowledge/

Computer Solutions Ltd (2014), CAN and CAN-FD a brief tutorial.
http://www.computer-solutions.co.uk/info/Embedded_tutorials/can_tutorial.htm

Connette, C. and M. Gruhler (2015), cob_generic_can.
http://wiki.ros.org/cob_generic_can

Dassault Systemes (2015), Solidworks Tutorials.
http://www.solidworks.com/sw/resources/solidworks-tutorials.htm

Elmo Motion Control Ltd. (2013), Software - Composer.
http://www.elmomc.com/support/downloads-software-tools-main.htm

ElmoMC (2014a), CANopen DS 301 Implementation Guide.
http://www.elmomc.com/support/manuals/MAN-CAN301IG.pdf

ElmoMC (2014b), What is CANOpen?
http://www.elmomc.com/capabilities/4%20.GMAS%20CANOpen%20Field%20Bus%20Communication/0.Getting%20Started%20with%20CANOpen%20Communication/Description/index.html

EMS Dr. Thomas Wünsche (2015), USB/CAN Interface CPC-USB/ARM7.
http://www.ems-wuensche.de/product/datasheet/html/can-usb-adapter-converter-interface-cpcusb.html

Farnell (2015a), GE CRITICAL POWER EVK011A0B41Z Isolated Board Mount DC/DC Converter.
http://nl.farnell.com/ge-critical-power/evk011a0b41z/dc-dc-converter-12v-2-5a/dp/2450698

Farnell (2015b), GE CRITICAL POWER KHHD006A0A41Z Isolated Board Mount DC/DC Converter.
http://nl.farnell.com/ge-critical-power/khhd006a0a41z/dc-dc-converter-5v-6a/dp/2450700

Festival, C. (2015), CAN Festival.
http://www.canfestival.org/

Fraunhofer (2015), ipa320 ipa_canopen.
https://github.com/ipa320/ipa_canopen

Intel Corporation (2015), Mini PC - Intel NUC Board D54250WYB.
http://www.intel.com/content/www/us/en/nuc/nuc-board-d54250wyb.html

Kollmorgen (2015), RBE.
http://www.kollmorgen.com/en-us/products/motors/direct-drive/rbe/

Korban, A. (2014), *C++11 Rocks*.
http://cpprocks.com

Marconi, L., C.Melchiorri, M. Beetz, D. Pangercic, R. Siegwart, S. Leutenegger, R. Carloni, S. Stramigioli, H. Bruyninckx, P. Doherty, A. Kleiner, V. Lippiello, A. Finzi, B. Siciliano, A. Sala and N. Tomatis (2012), The SHERPA project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments, in *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics*.

RLS (2015), AKSIM Off-Axis Rotary Absolute Encoder.
http://www.rls.si/products/aksim-off-axis-rotary-absolute-encoder

ROS.org (2015a), ROS Introduction.
http://wiki.ros.org/ROS/Introduction

ROS.org (2015b), ROS Tutorials.
http://wiki.ros.org/ROS/Tutorials

Skybotix (2015), VI-Sensor.
http://www.skybotix.com

The Qt Company (2015), Qt Documentation - Qt Creator Manual 3.4.0.
http://doc.qt.io/qtcreator/creator-writing-program.html

TQ Group (2015), Robodrive.
http://www.elmomc.com/products/whistle-digital-servo-drive-main.htm

Wolfram (2015), Wolfram Mathematica.
http://www.wolfram.com/mathematica/