NTNU
Kunnskap for en bedre verden

Department of Engineering Cybernetics

TTK4900 - Master project

Retrofitting a legacy robotic arm using open-source solutions

Author:
Kristian Blom

07.06.24

Table of Contents

List of Figures

List of Tables

1 Preface

2 Introduction

2.1 Purpose

This report describes the master thesis project conducted by Kristian Blom during the spring semester of 2024. The project builds directly on the specialisation project concluded the previous semester of autumn 2023. This report contains relevant excerpts from the specialisation project report.

2.2 Background and motivation

From the initial project description: *This project is defined and commissioned by a student in co-operation with Omega Verksted (OV). It builds on the specialisation project named "From hardware to control algorithms: Retrofitting a legacy robot using open source solutions", in which new control hardware was developed for an old robotic arm. The arm is a 6DOF industrial arm originally developed for chemical analysis and consists of a 5DOF arm with a pincer/manipulator installed on a rail. The developed hardware controls 6 brushed DC motors and processes information from 6 incremental encoders, 6 current sensors, 3 accelerometer/gyroscope units, 2 optocouplers and 1 mechanical switch.*

This document is not just a thesis, but also intended for people with less experience, and the language generally tries to reflect that

2.3 Project scope

The primary goal of this project has been to develop a robotic software system compatible with hardware developed during the specialisation project, such that the robotic arm may be operated by a non-expert third party. The primary use case is the mixing of beverages during informal meetings at Omega Verksted. The secondary goal of the project is that the software system should be readily expandable to support more sophisticated use cases and sensors, such as 3D printing and robotic vision, by an expert third party.

Project scope as itemised list:

1. Implement and verify required hardware improvements from the specialisation project.
2. Write hardware drivers for the relevant microcontroller peripherals.
3. Develop and implement a software architecture around the primary and secondary goals of the project.
4. Develop and implement tests to evaluate the system's performance.

3 Abbreviations and glossary

4 System specification

This section provides a high level description of the robotic system(4.1), a description of the hardware developed during the specialisation project(4.2), and a requirement specification for the software developed in this master project(4.3). The latter elaborates on scope items 2 and 3 from section 2.3.

4.1 Overview

The system consists of a robotic arm known from the original manufacturer, Beckman Coulter, as Optimized Robot for Chemical Analysis (ORCA); hardware developed as part of the specialisation project; and a general purpose desktop computer running the Linux/Ubuntu operating system. The arm is actuated by 6 DC motors, each connected to one joint via belts and gears within the arm. The arm is illustrated in figure 1, annotations marking each joint's positive direction. The gripper is mounted on a linear actuator, and is designed to bend around an object placed near the middle. It is capable of lifting and manipulating objects of approximately one kilogram.

The arm has several "hardpoints" on which PCBs or other equipment may be mounted, named after their physical location on the arm. The three control units developed in the specialisation project are mounted on the torso, shoulder and hand hardpoints, respectively. They are each responsible for the control of two joints, named after their function and/or physical location. For instance, the shoulder control unit is mounted on the shoulder hardpoint, and is responsible for controlling the elbow and wrist joints. A high level overview of the robotic system is presented in figure 2.

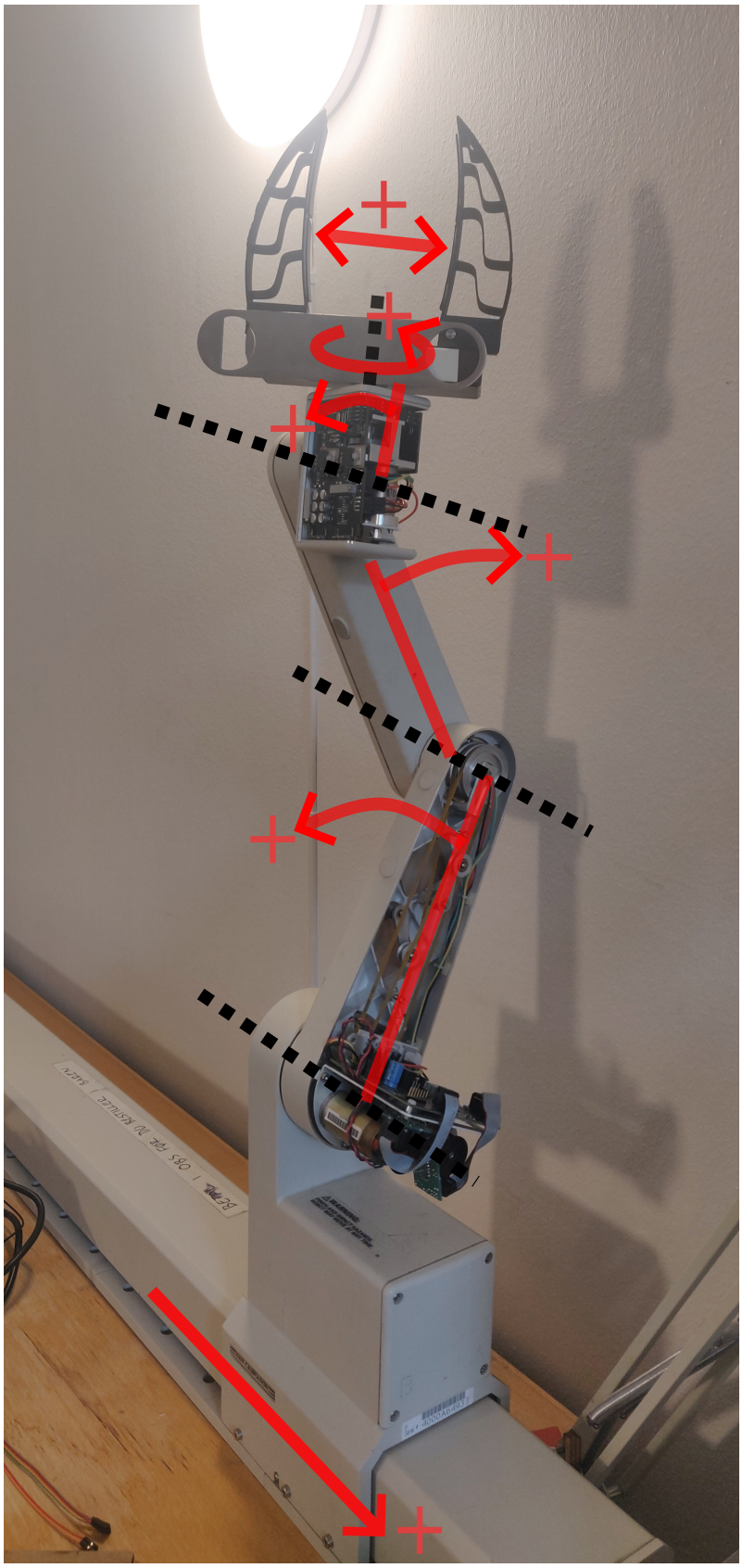


Figure 1: The ORCA arm, illustrated with joints and their directions. From the bottom up, joints are named rail, shuolder, elbow, wrist, twist and pinch.

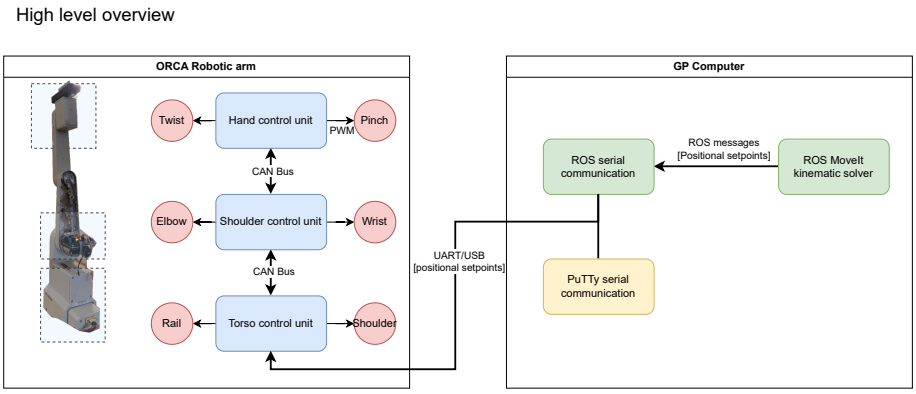


Figure 2: An overview of the robotic system

4.2 Hardware

As mentioned in section 2.2, all mechanical and some electromechanical parts were kept during the specialisation project in order to limit its scope. The parts are presented in table 1, originally presented in the specialisation project report. These parts were all found to be in working order, and replacing them would have required modification of the arm chassis.

For an in-depth discussion and presentation of the hardware, see sections 6 and 7 of the specialisation report. A key point from those sections is that there are space limitations in the arm preventing IMUs from being placed such that they may interface with their respective joint controllers directly. As a consequence, a critical function of the CAN bus is to enable transmission of IMU data between control units.

The following subsections explain each PCB developed in the specialisation project, and are summarised in figure 3. Figure 4 provides a detailed description of the arm, and was originally presented in the specialisation project report[2].

Table 1: Original parts kept in the project

Part name	**Description**	**Part no.**
Rail motor	Pittman 40mm 30.3V BDC motor	9233C59-R1[9]
Shoulder motor	Pittman 54mm 30.3VDC BDC motor	14205C389-R1[10]
Elbow motor	Pittman 40mm 24V BDC motor	9234C59-R1[9]
Wrist motor	Pittman 30mm 30.3V BDC motor	8224C143-R2[8]
Twist motor	Escap 23mm 23LT2R, 12V BDC motor	23LT2R[5]
Pinch motor	Escap 23mm 23LT2R, 12V BDC motor	23LT2R[5]
Wrist rotational sensor	TT Electronics Photologic Slotted Optical Switch	OPB971N51[4]
End switch	TT Electronics Photologic Slotted Optical Switch	OPB971N51
Motor encoder	Optical quadrature encoder 500CPR	HEDS-9100[18]

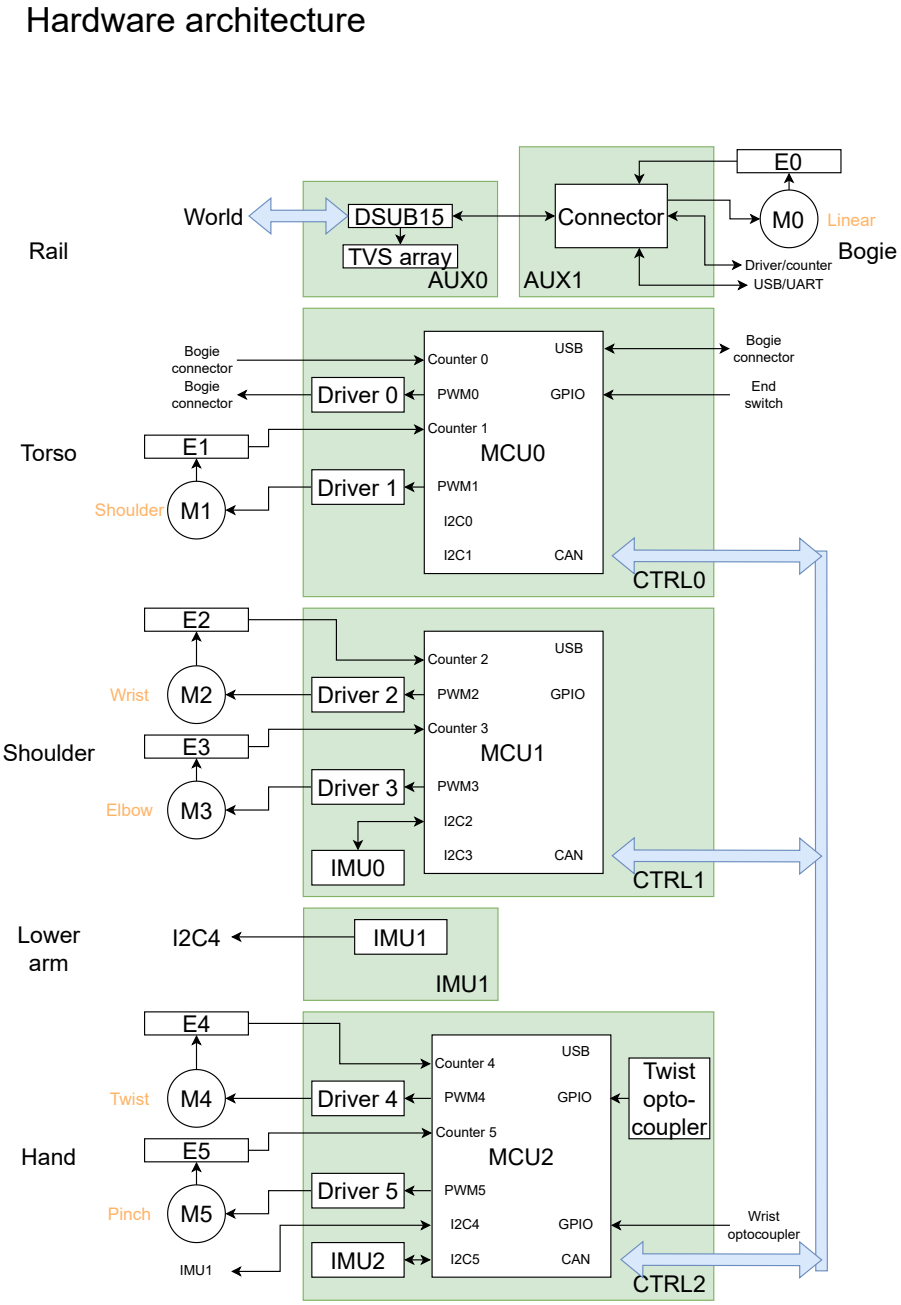
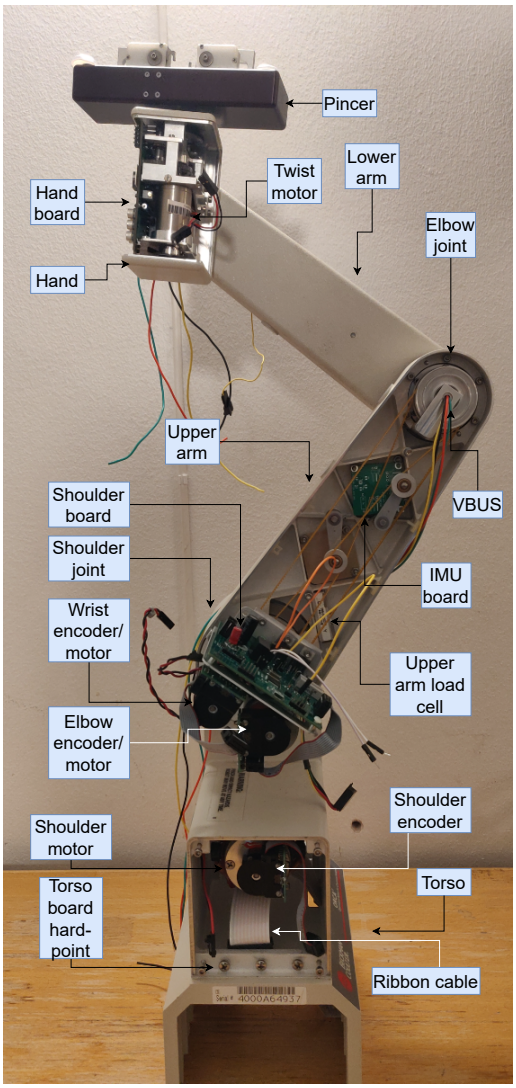
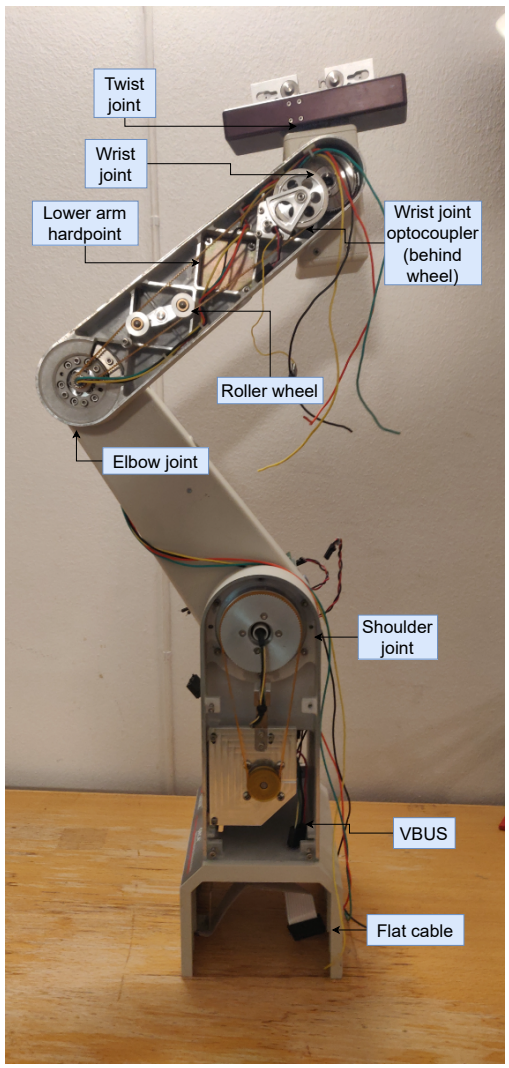


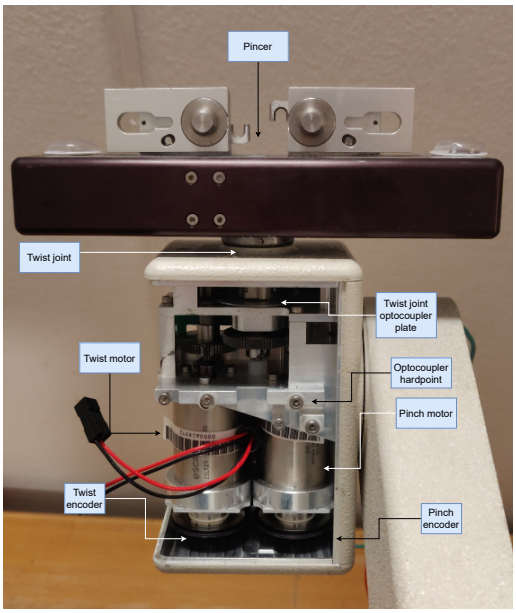
Figure 3: Hardware architecture. Green squares represent a PCB designed in the specialisation project, with key components as white squares. Left hand keywords indicate PCB mount location inside the arm. Originally presented in the specialisation report.



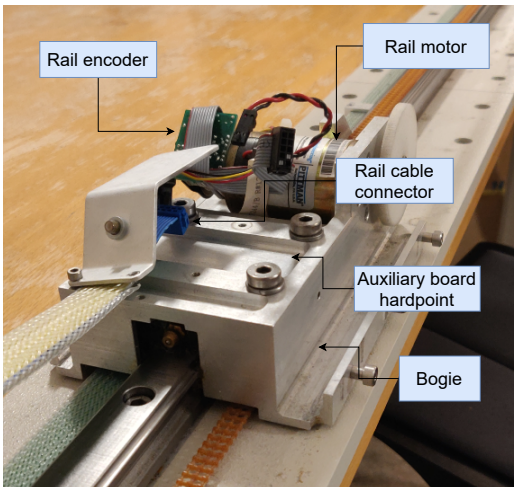
Arm viewed from the front.



Arm from the back.



A more detailed image of the hand and pincer.



The bogie on which the torso is mounted.

Figure 4: The arm with annotations. Orginally presented in the specialisation report

4.2.1 Auxiliary 0: Rail

The rail PCB is responsible for the arm's interface, a DSUB15 connector with signals for USB, UART and power. It is mounted on the rail hardpoint, and provides a socket for the 16 wire ribbon cable through which it interfaces with the bogie PCB(4.2.2). Additionally, it protects the USB and UART data lines from voltage spikes via its TVS array CDSC706[3]. The TVS array is clamped at 5V sourced from the torso control unit(4.2.3).

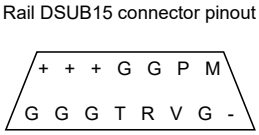


Figure 5: DSUB15 pinout

Table 2: Legend for figure 5, DSUB15 connector

Legend	**Meaning**
+	Input voltage
G	Ground
P	USB_DP
M	USB_DM
T	UART Tx
R	UART Rx
V	USB_VBUS
-	Not connected

4.2.2 Auxiliary 1: Bogie

The bogie PCB provides sockets for the 16 and 20 wire ribbon cables, as well as the rail motor and encoder connectors. Its purpose is to bundle the 16 wire ribbon cable with the rail motor and encoder wires into the 20 wire ribbon cable, through which it interfaces with the torso control unit(4.2.3).

4.2.3 Control unit 0: Torso

The torso control unit is responsible for the control of the rail and shoulder joints via motor 0 and motor 1 respectively, as well as external communication via the USB/UART lines. It interfaces with the upper arm IMU, IMU0 (4.2.4), via the shoulder control unit and the CAN bus. IMU0 is used in conjunction with E1 for the estimation of shoulder joint position. See table 3 for a presentation of the symbols in figure 3.

4.2.4 Control unit 1: Shoulder

The shoulder control unit is responsible for the control of the elbow and wrist joints via motors 2 and 3 respectively. It interfaces with the lower arm and hand IMUs, IMU1 and IMU2, via the hand control unit (4.2.6) and the CAN bus, for control of the elbow and wrist joints in conjunction with encoders 2 and 3, respectively. Additionally, it is responsible for direct communication with IMU0 via I2C. For remaining items, see table 3.

Table 3: Control unit 0

Symbol	Description	Unit name
Encoder 0 (E0)	Registers movement in the rail motor, used in estimation of the rail joint position. Relative, quadrature	HEDS-9100
Encoder 1 (E1)	Shoulder motor, see E0	HEDS-9100
End switch	Optical sensor, detects linear rail end position	See table 1
Driver 0	Motor driver, supplies power to the rail motor and provides current draw output. PWM control, analog current mirror	DRV8251A
Driver 1	Shoulder motor, see driver 0	DRV8251A
IMU0	Upper arm IMU	LSM6DSM
Microcontroller 0 (MCU 0)	Processing unit for control unit 0	STM32F303
Motor 0 (M0)	Linear rail motor, brushed DC	See table 1
Motor 1 (M1)	Shoulder joint motor	See table 1

4.2.5 IMU board 1: Lower arm

The IMU board serves as a mount point for the lower arm IMU, IMU1, which is used in the estimation of elbow joint position, and interfaces with the hand control unit via I2C. The IMU board also interfaces with the wrist optical sensor, relaying 5V from the hand control unit and sensor output to the hand control unit. IMU board 1 is mounted on the lower arm hardpoint.

4.2.6 Control unit 2: Hand

The hand control unit is responsible for the control of the twist and pinch joints via motors 4 and 5 respectively. It interfaces with the twist optical sensor and the wrist optical sensor via GPIO interrupts, and the CAN bus. Additionally, it is responsible for direct communication with IMUs 1 and 2 via I2C.

The twist optical sensor activates when the twist joint is in one specific position. As the twist joint has no hardpoints on which an IMU may be mounted, this sensor is essential to the estimation of twist joint position.

The wrist optical sensor activates for a set of joint positions, and may be used in the estimation of wrist joint position.

For remaining items, see table 3.

4.3 Software

The hardware presented in section 4.2 provides constraints for the software architecture. As mentioned, the software covers scope points 2 and 3 from section 2.3. This section elaborates on the functional requirements of the system, and presents a requirement specification which the finished software should fulfill.

Some keywords must be defined in the context of the requirement specification:

Must, shall: Denote a requirement which the failure to heed would significantly compromise the system's ability to achieve the project's two goals.

Should: Denotes a requirement which the failure to heed would only reduce the system's ability to achieve the project's two goals

4.3.1 Functional analysis

The primary goal for the system is to be able to "mix beverages" in informal settings, and be useable by non-expert personnel. This involves manipulating glasses, bottles and other objects that would typically fit in a human hand[1]. In order to achieve this, the system must implement the following functions:

- F1: Control the position and orientation of the pincer with an accuracy such that it may manipulate objects commonly involved with the mixing of beverages
- F1.1: Control the position of 6 joints concurrently
- F1.2: Take positional setpoints from a source external to the arm

"Informal settings" imply the presence of personnel and equipment which may not be accustomed to or intended for working with a robotic system during operation. Non-expert personnel refers to persons who may be familiar with robotic or autonomous systems in general, but do not have intimate knowledge of this system. In order to ensure safe operations in such a setting, the system should implement the following functions:

- F2: Operate in a predictable manner
- F2.1: Avoid fast or jerking motions
- F2.2: Follow predictable and/or intuitive movement patterns
- F3: Detect and respond when safe operational parameters are exceeded
- F4: Provide a user interface which requires limited preparation and/or education to make use of.

4.3.2 Architectural requirements

The secondary goal for the system is that it should be readily expandable to support use cases requiring a higher degree of movement accuracy, and/or more advanced sensors than currently exist within the hardware, by expert personnel. Expert pesonnel refers to persons who are experienced with robotic systems and embedded programming. This puts constraints on the development of the system's architecture:

- AR1: The system should consist of clearly defined modules
- AR1.1: A module's interface should be clearly defined
- AR1.2: A module should be limited in scope and purpose
- AR1.3: It should not be possible to pass information to a module outside of its interface.
- AR1.4: Naming conventions should apply across modules
- AR2: The system should adhere to common standards and best practices for embedded programming
- AR2.1: SOLID principles should be adhered to, to the extent that they apply
- AR2.2: Modules should aim for low coupling and high cohesion
- AR3: Design patterns should apply across modules

[1] This is a postulate

4.3.3 Documentation requirements

The secondary goal, as well as this being a master thesis project, sets expectations for code documentation:

- DR1: All modules should be documented
- DR1.1: All functions, classes, structs et cetera should have unique documentation
- DR2: Documentation should be readily available
- DR3: Documentation conventions should apply across modules
- DR3.1: Language should be similar across modules

5 Theory

SOLID Coupling and cohesion, other relevant concepts CAN UART TVS arrays, voltage clamping Switching vs linear voltage regs Interrupts

5.1 Inter-integrated circuits (I2C)

5.2 Software development

5.3 Communication protocols

Message round trips vs bitrates

5.4 Circuit design

5.5 ADC voltage conversion

5.6 CAN bus

Also cover CAN filter setup, as promised in peripheral config

5.7 UART

5.8 Timer frequency

interrupt frequency, also counter period vs capture/compare (duty cycle) in pwm

6 Tools and workflow

This section describes the tools used in the project, for the purpose of reproduction and/or future development.

6.1 Software

6.1.1 Tools

KiCad 7, CERN distro[SOURCE] was used in the development of PCBs in the project's initial phase, without non-standard plugins. KiCad is an open source, freely available CAD software for the design of PCBs. When necessary, component footprints were downloaded from the UltraLibrarian[SOURCE] website when available, or otherwise drawn in KiCad based on component datasheets. These footprints may be found in FOLDER.

Git was used during all phases of the project, and the project's repository may be found at Github[SOURCE].

STMCubeMX[SOURCE] was used for the initialisation of microprocessor peripherals, i.e. the generation of peripheral drivers, and the generation of the project's Makefile. STMCubeMX is a configuration tool published and maintained by ST, the manufacturer of the STM microprocessor family, providing a GUI through which the user may create a pinout, enable interrupts et cetera for their microprocessor. It is freely available, but requires the registration of a user account with ST[SOURCE]. The tool generates .h and .c files for the relevant peripherals based on choices made in the GUI, as well as either a Makefile for use with development tools outside the ST software ecosystem, or the equivalent for use with ST's own STMCubeIDE IDE. This project utilised the Makefile option as it seeks to minimise the use of non-open-source solutions.

VSCode[SOURCE] with ST extension[SOURCE] was the primary IDE for this project. The extension *stm32-for-vscode* was used to build and flash binaries for the MCUs.

PuTTy was used for serial communication with and debugging of the MCUs.

ROS2 Iron

MoveIt

6.1.2 Workflow

MCU binaries:

- Set peripheral configuration parameters in STMCubeMX
- Ensuring that the "overwrite user code" option is unchecked, generate code.
- Edit generated files as necessary, ensuring code is added between USER CODE labels to avoid being overwritten the next time STMCubeMX is used to generate/update peripheral settings.
- Add and edit source files in the TTK4900_drivers folder under the root folder created by STMCubeMX.
- Ensure source files are registered in the Makefile generated by STMCubeMX.
- Build binary file using the ST extension in VSCode.
- Flash binary to MCU using the ST extension and ST-LINK unit embedded in the development kit6.2.

ROS nodes

- Do stuff

6.2 Electronic/Hardware

6.3 Mechanical

- KiCad
- Git
- ST CubeMX
- VSCode (extensions: ST, C/C++)
- Development kit
- Angular measurement tool
- Power source
- Oscilloscope
- MoveIt
- Putty
- Categories: Electr(on)ic, Software, Mechanical
- Fusion 360
- Prusa Slicer

7 Hardware design

This section discusses the hardware presented in section 4.2 in further detail. Hardware produced for this project is an iteration of hardware produced for the specialisation project[2], and is dubbed Mk1.1. The material presented overlaps with sections 7 and 8 of the specialisation project report[2], but summarises key improvements over Mk1. Relevant copper layers are presented as a visual reference for pin headers and connectors.

Additionally, this section discusses results from the verification of Mk1.1 hardware. As they lay the foundation for the master project they are not considered "results" as such, and will only be adressed to the extent to which they affected the project's software implementation in section 15.

7.1 Improvement matrix

Table 4 summarises the "Further work" section of the specialisation report and was originally presented there. Each entry in the left column represents a point of improvement, and a cross in a subsequent column indicates that the issue affects the hardware unit in the top row of that column. The matrix is presented here as it was a key tool in organising the design of Mk1.1 and summarises a significant part of the master project, but not all improvement points will be adressed here.

Table 4: A summary of further work

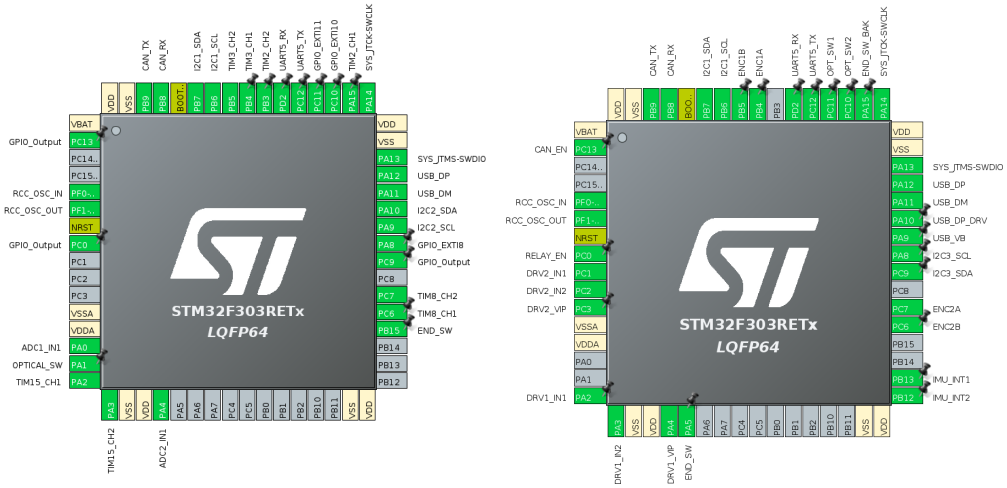
Item/affects unit	**Hand**	**Shoulder**	**Torso**	**Bogie**	**Rail**	**IMU (board)**	**MCU**
USB VBUS voltage divider		X	X				
USB VBUS pin		X	X				X
USB pullup on DM		X	X				
USB pullup transistor		X	X				
IMU 5V and 3.3V lines	X	X				X	
IMU/OPT header design	X	X				X	X
IMU header placement		X					
CAN verification	X	X	X				
CAN/48V connector placement			X				
CAN transciever placement	X						
Nylon bolts	X						
Mount point placement		X	X		X		
Mount point size		X	X				
20 pin connector			X	X			
Motor driver to PWM output	X	X	X				X
Motor driver ADC/PWM							X
Motor driver header silk	X						
Optocoupler 5V	X					X	X
Wrist optocoupler function	X					X	X
Bulk cap/pin header swap	X						
1.27mm jumpers	X						
Switching regulators	X	X	X				
Horisontal voltage regulators			X				
Voltage regulator LED resistor	X	X	X				
Encoder circuits							
Encoder header rotation				X			
TVS array					X		
Motor driver relay control		X	X				
Motor characterisation							
PCB production	X			X	X		

7.2 MCU pinout

The MCU pinout lays the foundation for PCB layout, and was changed first. The pinout is presented in figure 6.

- USB VBUS detection was set to pin PA9 in accordance with AN4879 chapter 2.6[12].
- PA10 was set to output as USB DP driver in accordance with AN4879 figure 5.
- PB12, PB13, PC10 and PC11 were opened as interrupt inputs to accommodate optical switches and IMU programmable interrupts, as well as ensuring optical sensor inputs are placed on 5V tolerant pins according to table 13 ofthe MCU datasheet[16].
- PWM output for motor driver 2 was moved to PC1 and PC2 from PC6 and PC7 in order to simplify PCB layout by gathering all motor driver outputs along one edge of the MCU.
- Current sense ADC input for motor driver 2 was moved from PA0 to PC3 in order to avoid an interference mode between peripherals, see elaboration below.

A key finding from the specialisation project was the discovery of an interference mode between the TIM2 (timer 2) and ADC1 peripherals[2], see chapter 13.3[2]. When TIM2 was configured for PWM generation, and ADC1 was activated on pin PA0, PA0 acted as an output pin with an analog voltage output proportional to the PWM duty cycle on TIM2. The cause of the interference mode could not be established beyond the fact that PA0 may also be configured for output from TIM2. The solution was to move ADC1 to pin PC3, which is not compatible with TIM2. Additionally, TIM2 was not used for PWM generation. For more information about timer configuration, see section 8.8.



MCU pinout configuration prior to hardware improvements

MCU pinout configuration after hardware improvements

Figure 6: Comparison of the old and new MCU pinout configuration

7.3 IMU board

As mentioned in section 4.2, the IMU boards act primarily as mounting points for the IMUs, and were to be mounted on hardpoints in the upper and lower arm sections, see figure 4. Secondarily, the lower arm IMU board would act as an interface with the wrist optical sensor. The LSM6DSM IMU has two configurable interrupt output pins[13] in addition to its I2C interface, and it was

[2]The report mentions pin PA4. This is erroneous; the interference was present on pin PA0

decided that at least one of these should be available for use in the system in the event that they prove relevant to the software implementation. The circuit is presented in figure 21.

The OPB971 optical sensor requires an input voltage of minimum 4.5V[4], while the LSM6DSM IMU has a maximum input voltage of 3.6V[13]. In order to save space in the wrist joint hole, it was decided that only the 5V line should be pulled from the hand control unit rather than both 3.3V and 5V lines. The ZMR330 step-down regulator, which outputs 3.3V for an input voltage above 4.8V[7], was intruduced as a replacement for a simple voltage divider in Mk1.

Figure 7 shows the improved IMU layout. Several headers/jumpers have been introduced to accommodate the various signal output options, and the PCB's interface header `I2C_PWR1` has been reduced from 8 to 6 pins compared to Mk1 (see chapter 9.8 of the report[2]). Jumper `INT_SEL1` lets the user select IMU interrupt 1 or 2 for output to the hand control unit. Jumper `INT_EN1` lets the user select whether or not to send the selected interrupt to output. The two jumpers implement the following boolean function:

$$I_{I2C\_PWR1} = (1_{INT\_SEL1} \oplus 2_{INT\_SEL1}) \cdot (I_{INT\_EN1} \oplus O_{INT\_EN1}) \tag{1}$$

If `INT_EN1` is set to O, the interrupt selected by `INT_SEL1` will be grounded. The other interrupt will be left floating.

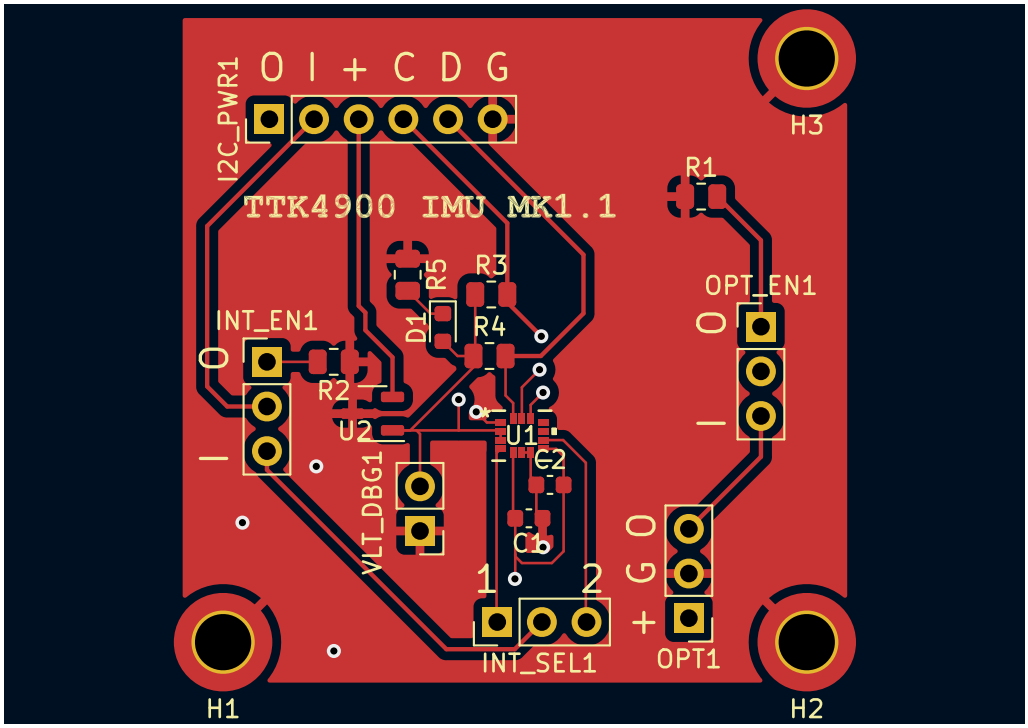


Figure 7: Front copper and silk layers of the improved IMU board

7.4 Rail

Rail board mount points were reevaluated for Mk1.1 with minor adjustments. The specialisation report suggests replacing the CDSC706 TVS array with one capable of protecting all 14 lines entering the arm. This was elected against due to uncerainty regarding correct clamping and the unprotected lines to some extent being protected by diodes on the control boards (see section 8.2.1 of the specialisation report). The circuit is presented in figure 15.

Table 6 explains the silk symbols on the 16 pin connector socket of the rail board as seen in figure 8, which shows the back copper and silk of the rail board. The silk should be interpreted as an

Table 5: IMU board header pin legend

I2C_PWR1	**Meaning**	**OPT_EN1**	**Meaning**
O	Optical output	I	Optical sensor output enable
I	IMU interrupt output	O	Optical sensor output disable
+	5V in	**OPT1**	**Meaning**
C	I2C Clock	+	Optical sensor 5V input
D	I2C Data	G	Optical sensor ground
G	Ground	O	Optical sensor output
INT_EN1	**Meaning**	**INT_SEL1**	**Meaning**
O	Disable IMU interrupt output	1	IMU interrupt 1 select
I	Enable IMU interrupt output	2	IMU interrupt 2 select
VLT_DBG1	**Meaning**	-	-
3.3V	output debug	-	-

overlay of the pins, such that the label "5" corresponds with connector pin 8. The 14 pin header on the upper right of figure 8 is connected to the DSUB15 connector presented in section 4.2, silk symbols explained in table 2. The circuit is presented in figure

Note: The input voltage pins were initially left partially unconnected due to a design error. This was corrected, and the rail version number was elevated to 1.2.

Table 6: Rail board 16 pin connector

16 pin connector	**Meaning**
G	Ground
P	USB_DP
M	USB_DM
V	USB_VBUS
R	UART Rx
T	UART Tx
5	TVS 5V clamping
+	Input voltage

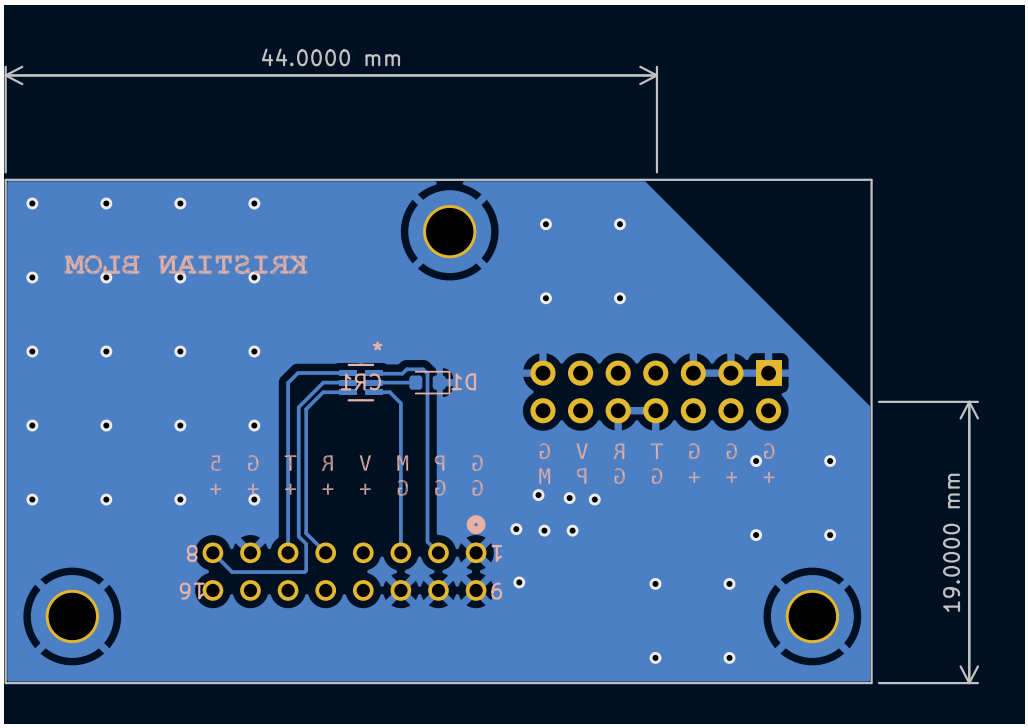


Figure 8: Back copper and silk layers of the improved rail board

7.5 Bogie

A key error in the design of the bogie board was the pin assignment of the 20 pin connector socket through which it interfaces with the torso control unit. It was discovered that one row of pins on the connector had been mirrored compared to the pin assigment on the torso board due to differing pin numbering schema in the circuit schematic.

The improved bogie board is presented in figure 9, and the silk symbols for the 20 pin connector socket is presented in table 7. The silk symbols for the 16 pin connector socket is presented in table 6. The circuit is presented in figure 16.

Note: Finding the cause of the connector issue took two attempts, and the bogie version number is therefore 1.2.

Table 7: Bogie board 20 pin connector socket, end switch connector header

MAIN1	**Meaning**	**END_SWITCH1**	**Meaning**
ES	End switch output	G	Ground
VB	USB_VBUS	E	End switch output
DM	USB_DM	+	5V input
DP	USB_DP		
5	5V output		
B	Encoder ch B		
A	Encoder ch A		
T	UART Tx		
R	UART Rx		
G	Ground		
+	Input voltage		
M1	Motor input 1		
M2	Motor input 2		

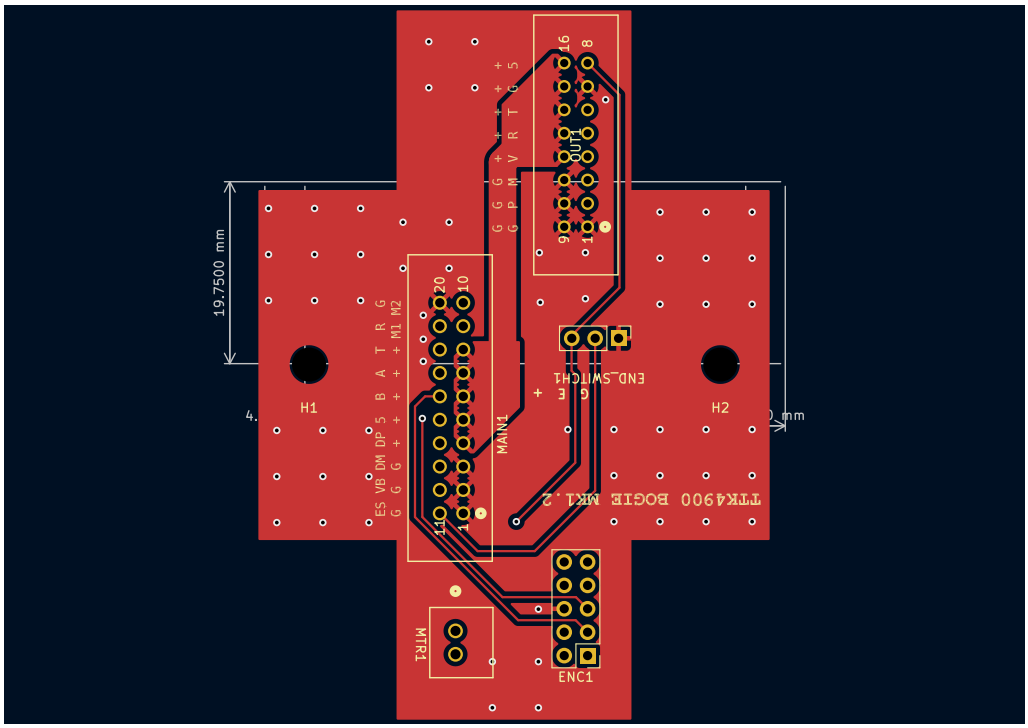


Figure 9: Front copper and silk layers of the improved bogie board

7.6 Torso

In addition to changes made uniquely to the torso, this section summarises improvements made to the various assemblies which make up the control units. Assemblies were originally introduced in section 8.2 of the specialisation report. Figure 11 presents the improved torso control unit. Legends for various pin headers are presented in 8. The torso circuit is presented in figure 17.

7.6.1 USB assembly

The USB assembly is present in the torso and shoulder control units. In the torso it is used for communication with the external control computer, as suggested in figure 3, and the data lines are pulled to a micro USB port on the board as well as the torso unit's 20 pin connector. In the shoulder it acts as a test/debug unit, and the data lines are pulled to a micro USB port which is covered when the arm is in normal operation. The USB assembly circuit is presented in figure 24.

The circuit was redesigned in accordance with chapter 2.6 and figure 5 of AN4879[12], see figure 10. R1 was set to $33k\Omega$, R2 to $82k\Omega$. The specialisation report suggests to add a transistor between PA10 (USB DP driver) and the DP line such that the MCU would not drive the line directly. However, this was deemed unnecessary on the basis that figure 5 and the following quote imply that the DP line may be driven directly via a resistor: "A DP pull-up must be connected only when VBUS is plugged. A GPIO from the MCU is used to drive it after the VBUS detection.[...]" - Chapter 3.1.1 of AN4879.

Figure 5. **USB FS upstream port without embedded pull-up resistor in self-powered applications**

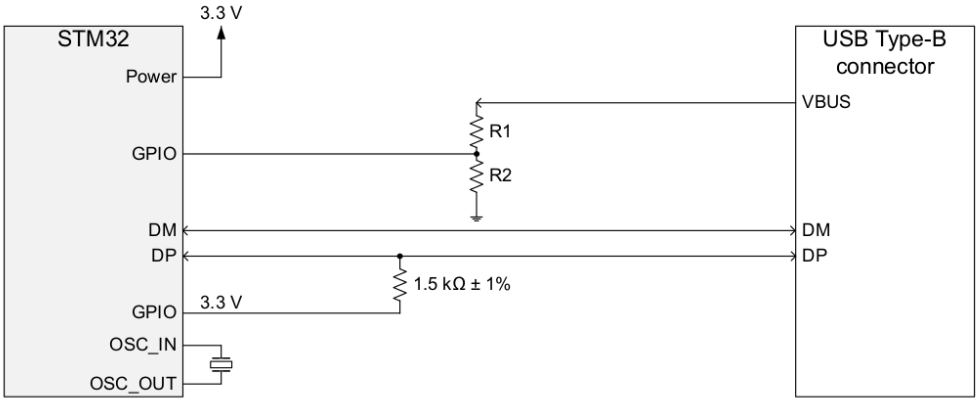


Figure 10: Figure 5 from ST AN4879, illustrating a valid USB circuit

Based on this, a $1.5k\Omega$ resistor was placed between PA10 and PA12. In order to fulfill the requirement of only connecting the DP pullup when VBUS is plugged, it was decided that an interrupt would be used to activate PA10 in software upon VBUS-detection, and that the pin would otherwise be held in its reset state.

Additionally, AN4879 chapter 2.3 recommends protecting data lines with a USBLC6 IC, which was also implemented.

7.6.2 CAN bus assembly

The CAN bus assembly is present in all three control units, and consists of the TJA1057BT[11] CAN transciever and its power supply components. The CAN_L/H lines are connected to the other CAN transcievers, while the CAN_Rx/Tx lines are connnected to the transciever's MCU. The circuit is presented in figure 23.

The CAN bus assembly was not satisfactorily verified during the specialisation project: *The CAN circuit was tested between the development kit and the MCU on the breadboard. While a correctly configured CAN message could be observed by oscilloscope on any point of the data lines, no indication was observed that the recipient MCU (breadboard) had registered the message.*(section 13.1.1)

The cause was likely that the wrong CAN transciever had been used in one of the test units as well as ordered for Mk1.1: TJA1057**G**T instead of TJA1057**B**T. The BT has a pin dedicated to measuring logic voltage levels other than the CAN standard of 5V, while the GT does not (TJA1057[11] chapter 6, pin 5).

7.6.3 Motor driver assembly

The motor driver assembly is present in all three control units, and consists of two DRV8251A motor drivers[19], motor bulk capacitors and, in the case of the torso and shoulder control units, a JV-3S-KT relay operated via a 2N551 NPN BJT transistor from the MCU's PC0 pin. The transistor and relay act as a dead man's switch for the motor drivers, ensuring that motors will lose power should the MCU go offline. The motor drivers are controlled from the MCU by PWM signals. A key function of the motor drivers is their proportional current output: pin 1 will output a current proportional to the current drawn by the motors (I_{PROPI}), which is measured as a voltage (V_{IPROPI}) across the current resistor (R_{IPROPI}). The assembly is presented in figure 25.

In addition to changing the choice of transistor, motor driver PWM signal lines were changed in

accordance with the update to MCU pinout described in section 7.2 for Mk1.1.

7.6.4 Voltage regulator assembly

The voltage regulator assembly is present in all three control units, and consists of two adjustable LM317HV linear voltage regulators[20] and their adjustment circuits. The voltage adjustment potmeters RVn are set such that the regulators step the system input voltage down to 3.3V and 5V, respectively, in order to supply relevant ICs. The LM317HV were found to rise to a substantial temperature at an input voltage of 20V during the specialisation project, and the target input voltage for the system is 48V[3]. The report therefore suggests replacing the LM317HV with an equivalent switching regulator, as switching regulators tend to develop less heat than linear regulators. However, due to the limited time available for implementation of Mk1.1, this was elected against. The assembly circuit is presented in figure 26.

PCB footprints of the regulators were changed to horisontal from vertical, in part to improve heat dissipation by using the PCB itself as a heat sink, and in part to make placement of the regulators independent of the mount points available on the external heat sink which the regulators were attached to in Mk1.

Additionally, the output voltage indicator LED resistor was increased from 200Ω to 1.5$k\Omega$ in order to dim the LEDs and reduce eye strain when working with the control units.

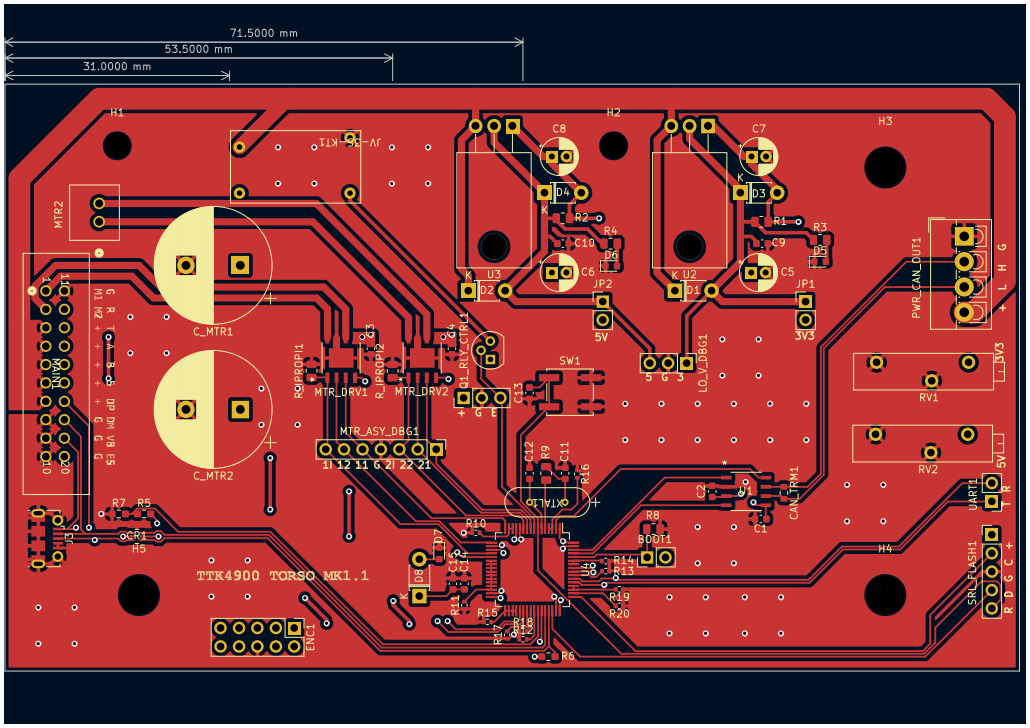


Figure 11: Front copper and silk layers of the improved torso board

7.7 Shoulder

As shown in table 4, changes made in the torso control unit largely apply to the shoulder control unit as well. The two units are almost identical with respects to which assemblies are present, with the IMU assembly being the only difference. The shoulder front copper layer is presented in figre 12, and the shoulder circuit is presented in figure 18.

[3]Investigation during the specialisation project suggested this was the system's original operating voltage.

MTR_ASY_DBG1	**Meaning**	**Q1_RLY_CTRL1**	**Meaning**
1I	DRV1 VIPROPI	+	3.3V
12	DRV1 PWM2	G	Ground
11	DRV1 PWM1	E	Relay enable
G	Ground	**LO_V_DBG1**	**Meaning**
2I	DRV2 VIPROPI	5	5V output
22	DRV2 PWM2	G	Ground
21	DRV2 PWM1	3	3.3V output
PWR_CAN_OUT1	**Meaning**	**SRL_FLASH1**[a]	**Meaning**
+	Sys voltage in	+	VDD Target
L	CANL line	C	Programmer clock
H	CANH line	G	Ground
G	Sys ground out	D	Programmer data
		R	Reset target

Table 8: Pin header legends for the torso control unit. [a]The flash header was designed for use with an ST-LINK programmer, and is described in chapter 6.2.4 of [17].

7.7.1 Mount points

Mount points were reevaluated, and fastening bolts were changed from metal to nylon to prevent damage to the board.

7.7.2 IMU assembly

The IMU assembly is present in the shoulder and hand control units, and consists of the LSM6DSM IMU[13] and its power supply circuit. A modified version is also present in the IMU board, and the circuit is presented in figure 22. The SDA and SCL lines are pulled to I2C port 3 of the hand and shoulder MCUs.

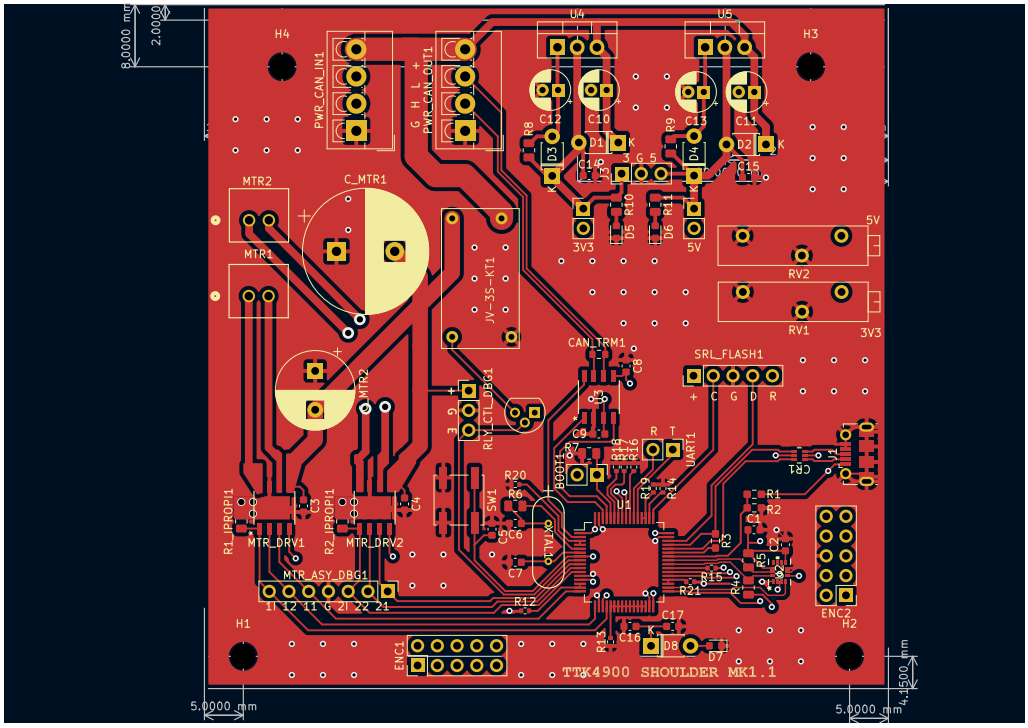


Figure 12: Front copper and silk layers of the improved shoulder board

7.8 Hand

The hand control unit could not be verified during the specialisation project due to a short circuit between the power input line and ground layer. The cause was found to be a misplaced stitching via, and care was taken to avoid this error in this and every other board produced afterwards. As the various subassemblies had largely been verified in the other control units, it was assumed that this was the only error preventing the hand unit from functioning correctly. Pin header legend for the hand control unit is presented in table 9, and boards are presented in figures 13 and 14.

The hand circuit is almsot identical to the shoulder with regards to present assemblies, except that it does not have a USB assembly. The circuit is presented in figure 19.

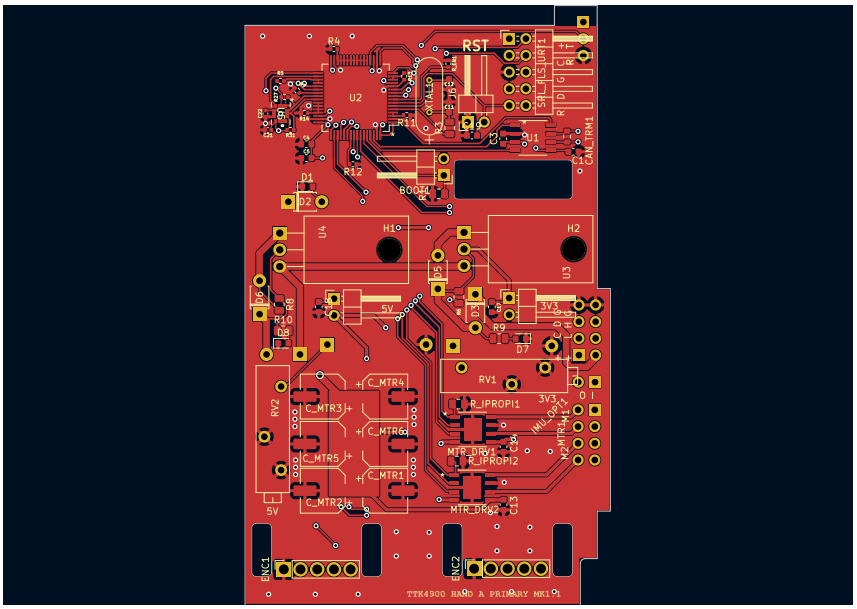
7.8.1 Hand B: Optical sensor

The twist optical sensor is mounted on a separate PCB due to placement constraints in the hand chassis, and connected to the hand control unit via a three pin header. The PCB is referred to as hand B, and the circuit is presented in figure 20.

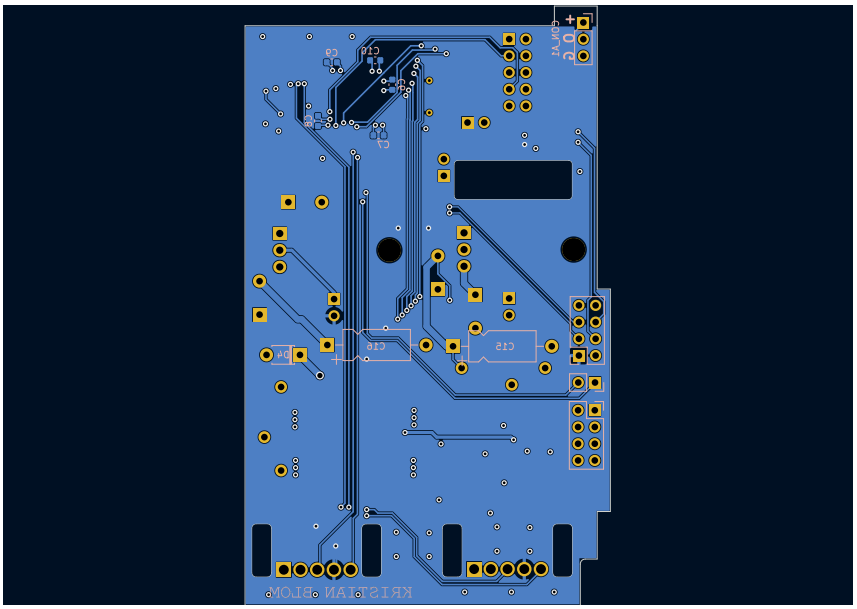
The OPB971 optical sensor activates when its aperture is obstructed, such that a short circuit occurs between the output and ground pins. A voltage sensor placed between the output and voltage input pins will then sense the differential. A breadboard test was conducted around this concept with an LED placed between the input and output pins: when the aperture was obstructed, the LED lit. The optical sensor output pin was routed to a GPIO pin on the hand MCU on the assumption that it would act as a substitute for the LED, sensing the input/output differential upon activation of the optical sensor.

SRL_FLS_URT1	**Meaning**	**PWR_CAN_IMU1**	**Meaning**
+	VDD Target	+	5V output
C	Programmer clock	C	I2C clock
G	Ground	D	I2C data
D	Programmer data	G	Ground
R	Reset target	+	Input voltage
T	UART Tx	L	CANL
R	UART Rx	H	CANH
CON_A1	**Meaning**	G	Ground
+	5V output	**IMU_OPT1**	**Meaning**
O	Twist optical sensor	O	Wrist optical sensor
G	Ground	I	IMU interrupt

Table 9: Pin header legends for the hand control unit. **CAUTION:** The 5V output on the I2C header, adjacent to power input on the CAN/power header, MUST NOT have a voltage applied to it. As before, the symbols are to be interpreted as overlays of the pins.



Front copper of the improved hand control unit



Back copper of the improved hand control unit

Figure 13: Front and back copper layers of the improved hand unit

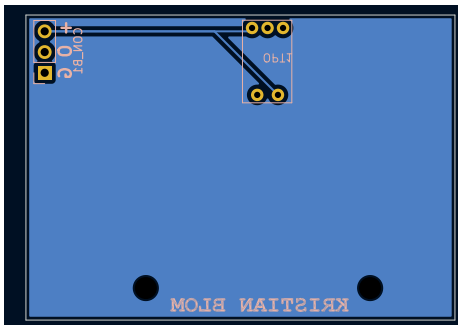


Figure 14: Back copper and silk layers of the hand B board

7.9 Verification

All PCBs were tested before installation into the arm before installation, similar to the specialisation project. They are presented in order of dependency on previously tested systems, sorted by assembly.

7.9.1 Voltage regulators

For each of the control units, voltage adjustment potentiometers RV1 and RV2 were turned until 3.3V and 5V were measured between ground and the relevant pin of `LO_VLT_DBG1`, see fig 26. Jumpers JP1 and JP2 were then connected with jumpers, and voltages were measured again to ensure no voltage drops, which would be indicative of a short circuit, were present. No voltage drops were measured, and the voltage regulators were judged to be operational.

7.9.2 MCU programming

The previously generated MCU pinout was compiled using VSCode and the ST extension (see section 6.1), and attempts were made to flash each of the control units also using the ST extension with the resultant binary file. This yielded no errors, effectively verifying the MCU voltage supply, serial/flash header designs, and the connection between the external computer and ST-LINK programming unit embedded in the Nucleo development kit. The MCUs were deemed operational.

7.9.3 Motor control relay, GPIO

The motor control relay makes an audible click when activated. The program mentioned in the previous test was expanded to include a simple loop activating and deactivating the relay every second by writing to the `RELAY_EN_PIN` on PC0. This tests the aliasing of GPIO pins to human readable names (see section 8.5), the transistor circuit, and the relay. Audible clicks were heard, and relay control was deemed operational. This test applies to the torso and shoulder control units, as the hand does not use a relay.

7.9.4 Motor drivers, PWM generation

PWM was tested using a test program from the specialisation project, updated to the Mk1.1 MCU pinout. The program generates a triangle wave at approximately 0.5Hz, which is used to scale the PWM duty cycle output from pins PA[2,3] and PC[1,2] (see section 8.8) for motor driver 1 and 2, respectively. The duty cycles must be inverse in order for the motor drivers to activate correctly, i.e. if PA2 has 40%DT, PA3 must have 60%DT, see datasheet[19] chapter 8.4.1. This test applies to all control units.

PWM output was verified by probing the `MTR_ASY_DBG1` header, see fig 25, with an oscilloscope. PWM signals with waxing/waning duty cycles according to the generated triangle wave were observed for both motor drivers on all control units. The test was repeated for the motor driver output pins, and the PWM signals were observed here, too, at a voltage identical to the system input voltage of 20V[4].

Addressing the key improvement point from the specialisation project, no voltage was measured on the `DRVn_IPROPI` lines during this test. This implies that the discussed interference mode is not present in the current peripheral configuration, and the motor driver assemblies were deemed operational.

[4]The highest setting of the avaiable power supply

7.9.5 UART data transmission

UART transmission was tested by sending the character string `HELLO, WORLD!` using configuration parameters discussed in section 8.9 to the external computer via the Nucleo development kit's UART-to-USB adapter (see chapter 6.8 of UM1724[17]). The signal was listened for using PuTTy with configuration settings mirroring those of the MCU, and was received successfully from all control units.

UART reception was tested by activating the motor control relay upon reception of the character `R`, detected via the UART reception interrupt handler. This was successful for the torso and shoulder control units. The hand control unit had no simple way to test reception, and this was assumed to be functional based on the results from the torso and shoulder units. UART data transmission was deemed operational.

7.9.6 Twist optical sensor

The twist optical sensor was tested by enabling an interrupt on PC11 upon a rising edge, and using sending a message over UART in the interrupt handler. The sensor was obstructed using a paper sheet, but no interrupt was triggered.

The error was found to be in the design of the sensor circuit. As mentioned in section 7.8, the optical sensor shorts its output pin to ground. Thus, the MCU will always measure 0V between output and ground. To amend this, two resistors of $10k\Omega$ were added between V_{cc} and OUT, OUT and Ground, respectively, of the Hand B connector pin header[5]. Before obstruction, the MCU will measure 2.5V, or half of the optical sensor V_{cc}, which is above the five volt tolerant pin activation threshold of 1.85V for a MCU supply voltage of 3.3V (MCU datasheet, table 66[16]). On obstruction of the sensor, the measured voltage will be 0V.

The interrupt was reconfigured to activate on a falling edge, and the test was repeated. The interrupt was triggered, and the optical sensor was deemed operational. This test applies to the hand control unit.

7.9.7 End switch and wrist optical sensors

The end switch and wrist sensors were tested by applying a 5V input voltage, and measuring the voltage between the OUT and ground pins when the switch was pressed/wrist joint manipulated to obstruct the sensor. Output voltage was found to be 2.7V. As discussed in section 7.9.6, this is above the five volt pin activation threshold, and the sensors were deemed to be operational.

7.9.8 I2C data transfer

I2C was tested by attempting to read the `WHO_AM_I` register of the three LSM6DSM IMU units[13] using a program developed for the specialisation project, and displaying the value via UART. The program utilises the STM32 HAL library discussed in section 8.1, which returns a status message of `HAL_ERROR` if a function fails in hardware. If the test is succesful, IMU assemblies, IMU boards and I2C peripheral configuration have been designed/assembled correctly.

The tests were generally not successful, and further testing was necessary to isolate the error(s). Several units were used in the process:

- Hand control unit: has one onboard IMU (IMU2) on I2C port 3, and one external IMU (IMU1) on I2C port 1.
- Shoulder control unit: has one onboard IMU (IMU0) on I2C port 3.

[5] Changing the PCB design itself was not deemed worth the time and money

- Breadboard IMU: an IMU on a breakout board, previously used for circuit design.
- IMU PCB 1: One copy of the IMU board design seen in figure 7, intended to function as IMU1.
- IMU PCB 2: Similar to IMU PCB 1, made for these tests.
- Adafruit unit: Adafruit MMA8451 accelerometer breakout[1], a COTS IMU bought for these tests, and to act as IMU board replacements should a solution not be found.
- Arduino UNO: Hobby/development kit for embedded programming, explicitly compatible with the Adafruit unit.

Attempts were made at reading the `WHO_AM_I` registers of all availabe IMUs, results presented in table 10.

IMU\MCU	Hand I2C1	Hand I2C3	Shoulder I2C3	Arduino UNO
Hand onboard	-	HAL_ERROR	-	-
Breadboard	WHO_AM_I	-	-	-
IMU PCB 1	HAL_ERROR	-	-	-
IMU PCB 2	HAL_ERROR	-	-	-
Shoulder onboard	-	-	WHO_AM_I	-
Adafruit	HAL_ERROR	-	-	WHO_AM_I

Table 10: Summary of attempts to read IMU registers

No debug headers were included in the I2C/IMU assemblies. Oscilloscope readings were therefore limited to Hand I2C1, where a breadboard was used to insert probes along the data and clock lines. Common for negative (`HAL_ERROR`) results was that the waveform looked correct up until the point where a slave ACK (see 5.1) should have appeared. This indicates that the circuit is correctly designed, and that I2C1 is correctly configured (see 8.6).

Furthermore, the breadboard IMU worked with hand I2C1, and the shoulder onboard IMU worked with I2C3. This proves definitively that I2C peripherals are viably configured, and it strengthens the indication that circuit design is viable: Shoulder and hand onboard IMUs both use the IMU assembly schematic (fig 22), so any difference between the two would be limited to circuit layout. No noteworthy differences were found in the layout of the hand and shoulder onboard IMU circuits. The IMU assembly itself is based on the breadboarded design, which did also work.

One difference between the IMU PCB design (fig 21) and the IMU assembly is that both interrupt pins are never grounded at the same time, see equation 1. No indication was found that this would lead to undefined behaviour beyond a note that the output is "forced to ground" by default. See table 19 of the datasheet[13]. However, when the breadboard IMU circuit was modified to let one or both interrupt(s) float, communication with this unit failed, too. This indicates that the IMU board circuit design may need to be revisited.

All circuits except the Adafruit unit were hand soldered, and some difference in quality may be expected. Soldering was done by reflow oven and solder paste, and had generally been successful thus far in the project. However, due to its 14 pads at a pitch of 0.5mm in an LGA package, the LSM6DSM was significantly more difficult to solder than other ICs. Relevant solder points of MCUs and IMUs were inspected using a stereoscope, but no difference between functional and non-functional units could be established[6]. As a test for visual inspection, IMU board was resoldered with a deliberately uneven amounts of solder paste on the IMU pads. The IMU was clearly tilted after soldering, but there was no indication of short circuiting between pads with too much paste. However, pins with very little paste appeared to have been disconnected as the IMU tilted. None of these clues to improper connections could be seen in the other units[7]. The MCUs were also inspected, and I2C pins appeared to be in order. At this point, six IMUs had been soldered with

[6]No figures could be acquired from the stereoscope

[7]Getting a clear view of the connection was difficult in all cases, which in itself may be a source of error

only two positive results. Considering that all were soldered with little variation in process, it seems improbable that the soldering process should be the only cause of the negative results.

An attempt was made at establishing whether the IMUs themselves may be defunct. This was initially assigned a very low probability considering the effect it would have on ST's business model should it routinely ship defunct batches of ICs, but one result lends the hypothesis credibility: An attempt at reading the shoulder onboard IMU's Z axis acceleration, it failed. Reading other axes, both acceleration and rotation rate, was successful. No further tests could be made to establish whether the remainder of the batch was defunct.

The Adafruit units which were supposed to act as a replacement for the IMU boards also failed when paired with hand I2C1. It is assumed to be because the MMA8451 requires a repeated start condition when initialising communication with the master (Overview chapter[1]). While the STM32F303 is configurable for repeated start condition (chapter 25.2.1 of UM1786[15], this was not implemented due to time constraints.

Summarised:

- The I2C peripheral configuration is viable.
- The IMU assembly circuit is not incorrect.
- The IMU board design may need revisiting – grounding both interrupt pins.
- The soldering process may need to be revisited.
- Two axes are available on the shoulder joint: X and Y.

I2C data transfer was deemed minimally operational.

7.9.9 USB data transfer

Didn't work, lost motivation due to time and UART working.

7.9.10 CAN bus data transfer

In order to verify CAN bus, the hand control unit was programmed to send a CAN message to the torso control unit upon activation of the twist optical sensor via the interrupt handler. The torso control unit was programmed to send a message over UART upon reception of a CAN message from the hand unit. The twist joint was then moved to the position where the sensor should activate. A message was read in PuTTy, and CAN bus was deemed operational.

7.9.11 verification summary

- Voltage regulators correctly output 3.3V and 5V.
- MCUs may be programmed via the Nucleo devkit ST-LINK programmer.
- Motor control relays correctly function as a dead man's switch.
- Motor drivers are controllable via PWM.
- Information may be exchanged with the arm via UART.
- Twist and end stop sensors are operational and in use.
- Wrist sensor is operational, but not in use.
- One of three IMUs are partially operational.

- USB was dropped due to complexity and time, lack of need.
- Information may be exchanged between control units via CAN bus.

7.10 Installation

Following verification, all boards were installed into the arm except the IMU board: No plan existed to make use of the wrist optical sensor, and the IMU, as discussed, was not operational.

The verification process was repeated after installation, and revealed no major design flaws. Certain THT components in the hand control unit touched the hand chassis, and had to be covered in electrical tape, however.

The installation process is described in detail in section 18.1.

7.11 Circuit diagrams

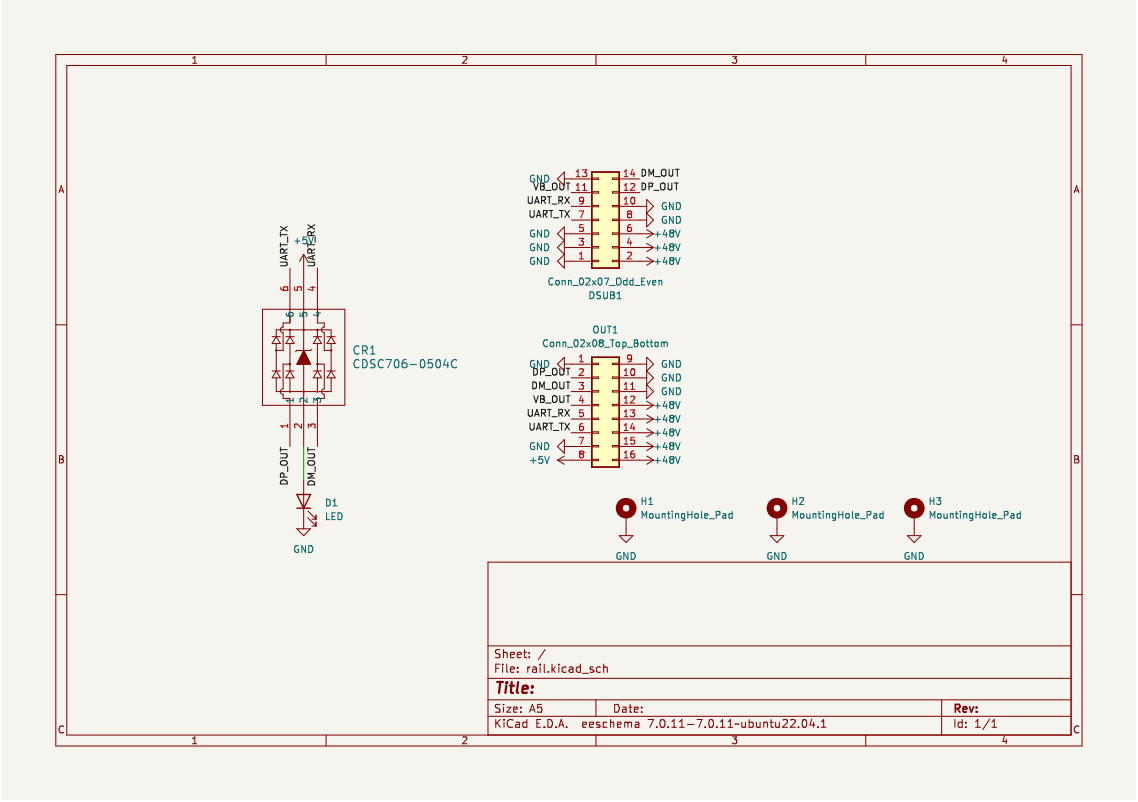


Figure 15: Circuit diagram of the Mk1.1 rail board

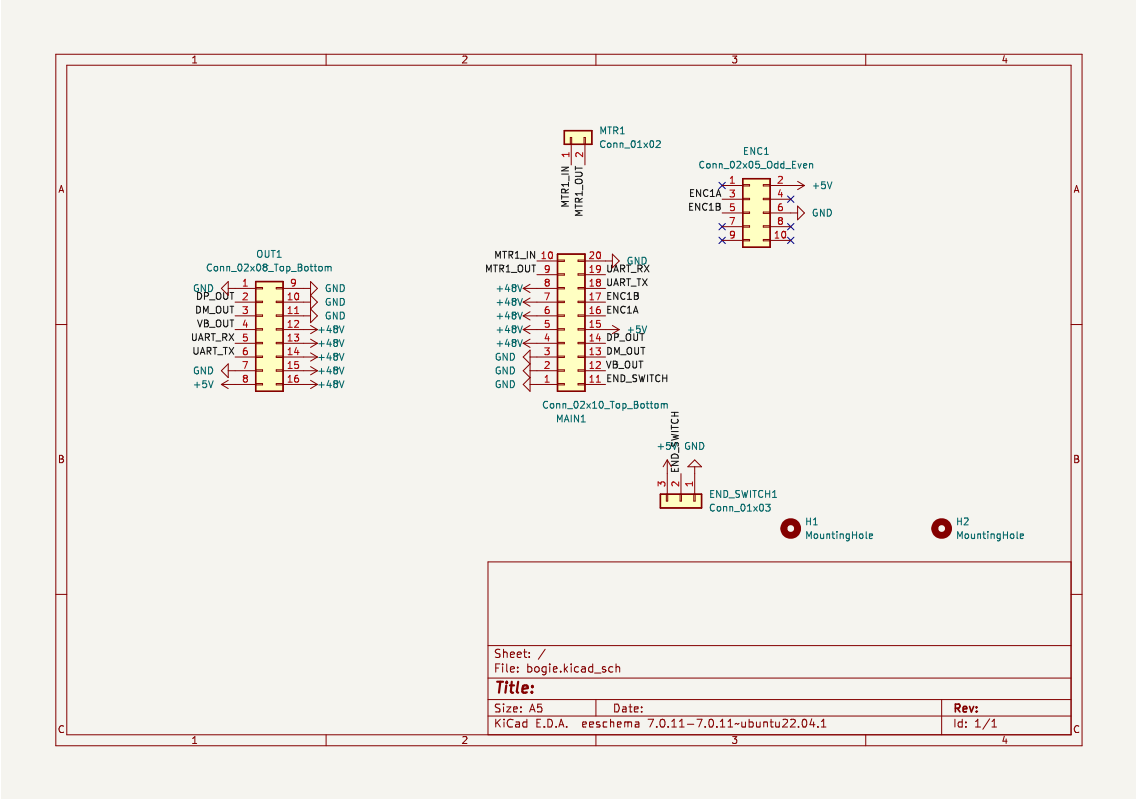


Figure 16: Circuit diagram of the Mk1.1 bogie board

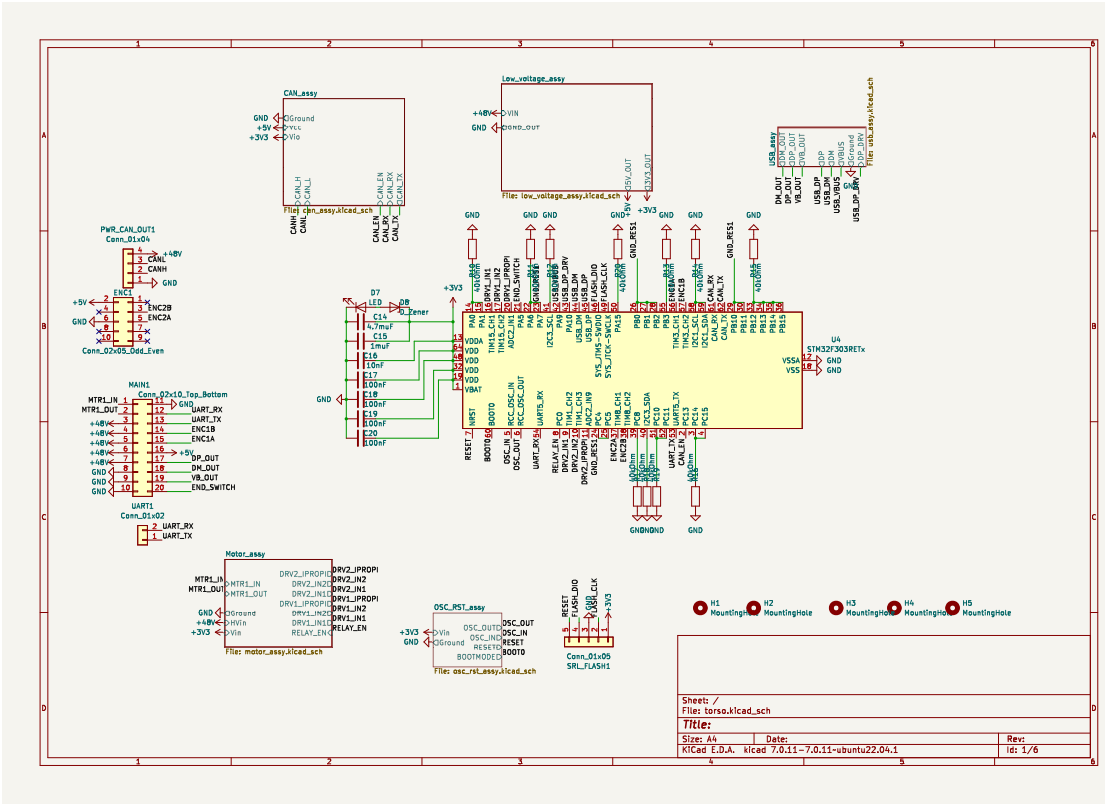


Figure 17: Circuit diagram of the Mk1.1 torso control unit

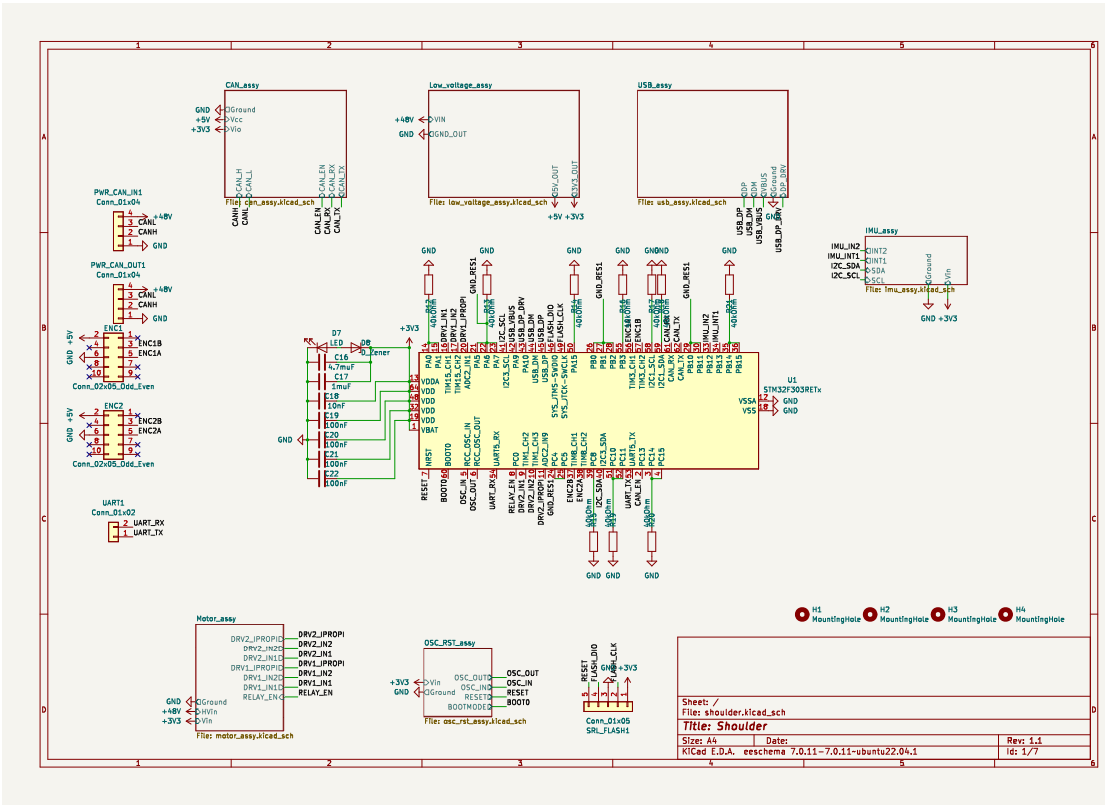


Figure 18: Circuit diagram of the Mk1.1 shoulder control unit

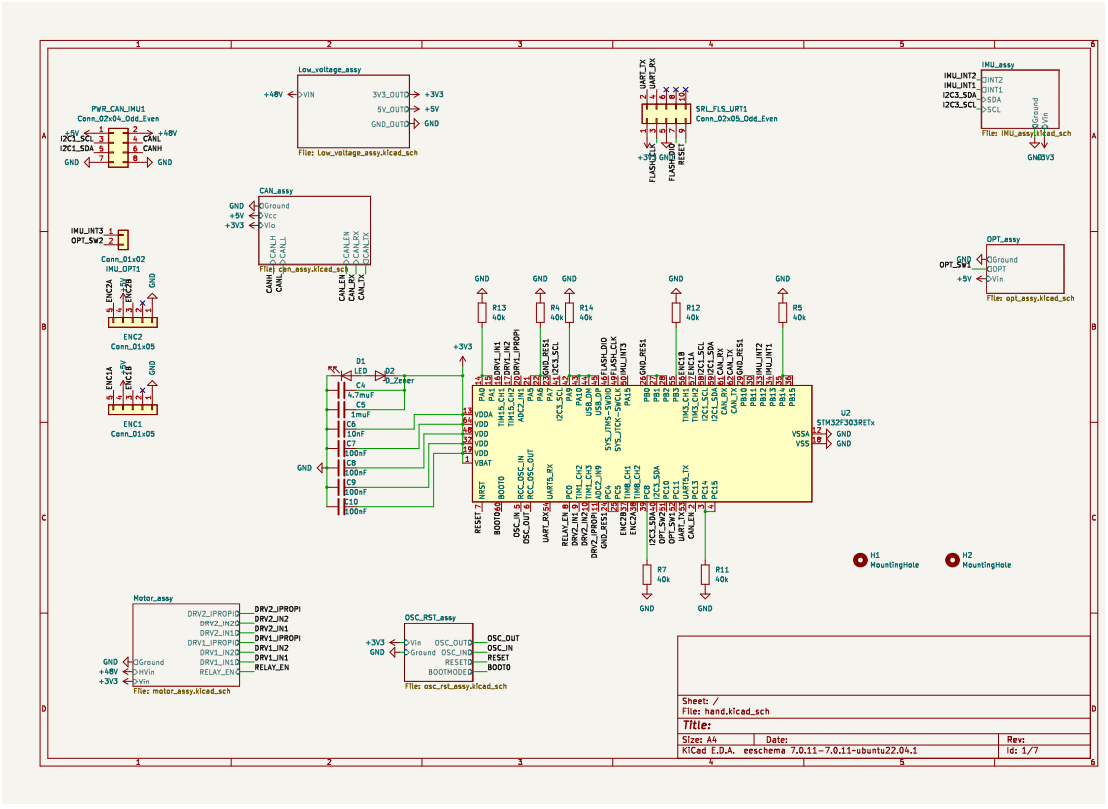


Figure 19: Circuit diagram of the Mk1.1 hand control unit

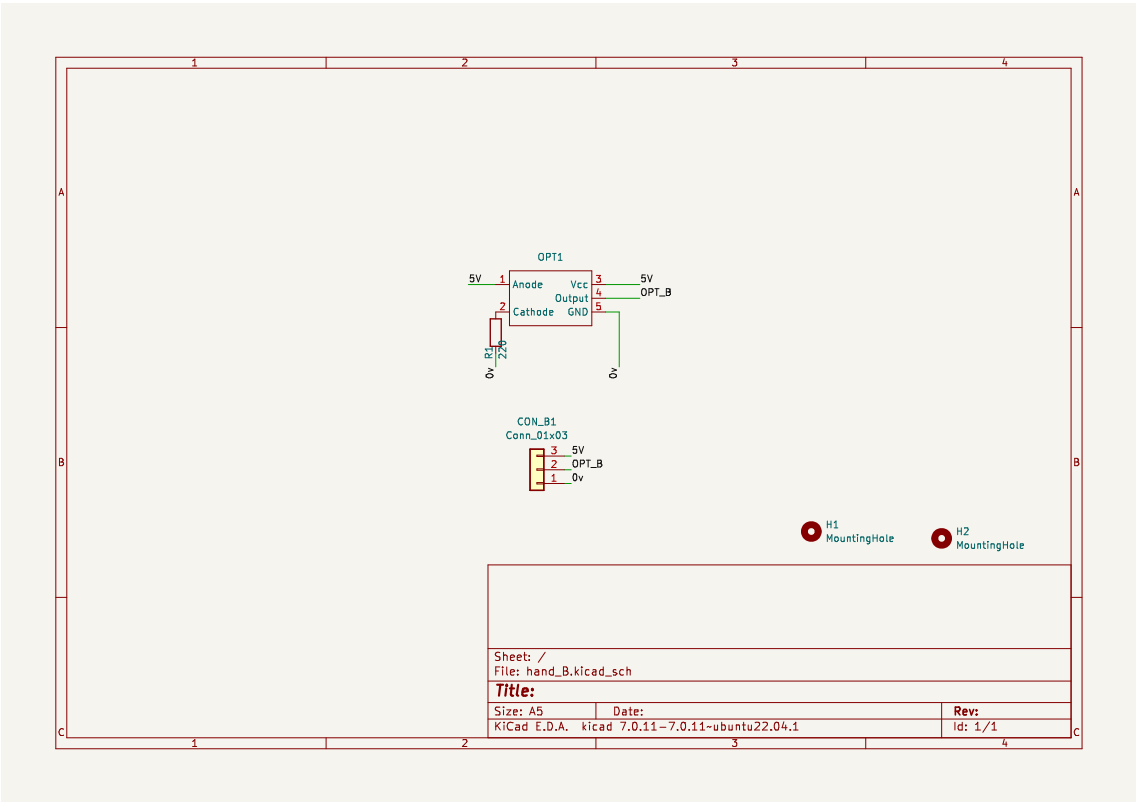


Figure 20: Circuit diagram of the twist optical sensor

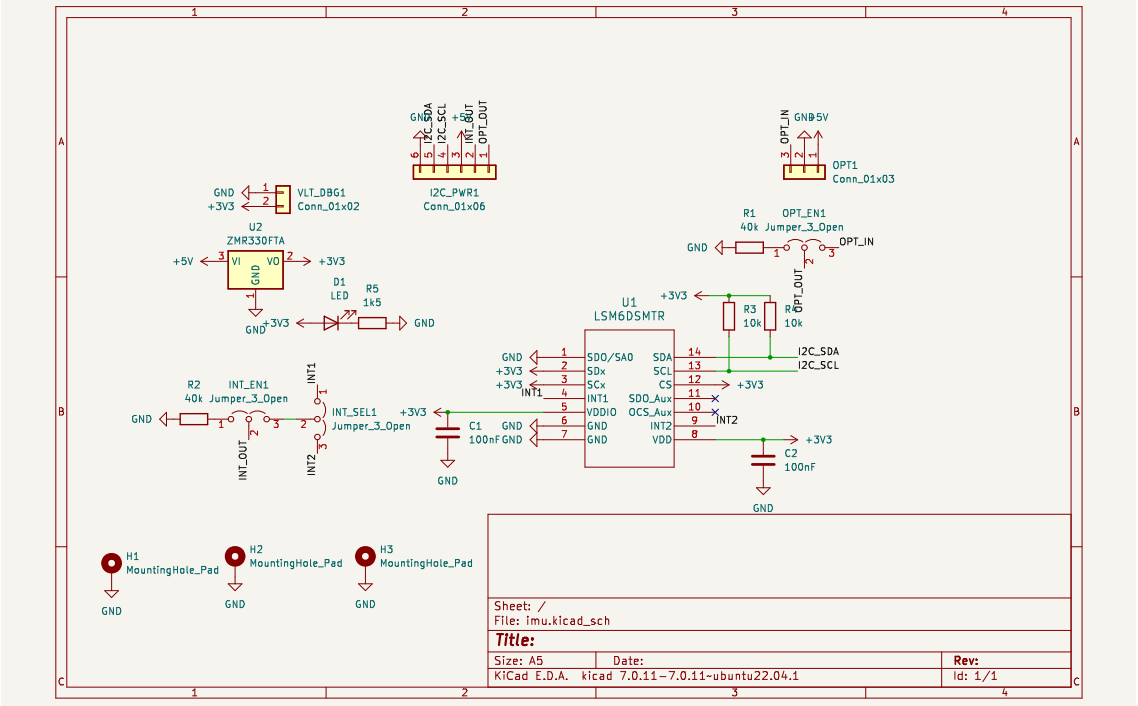


Figure 21: Circuit diagram of the Mk1.1 IMU board

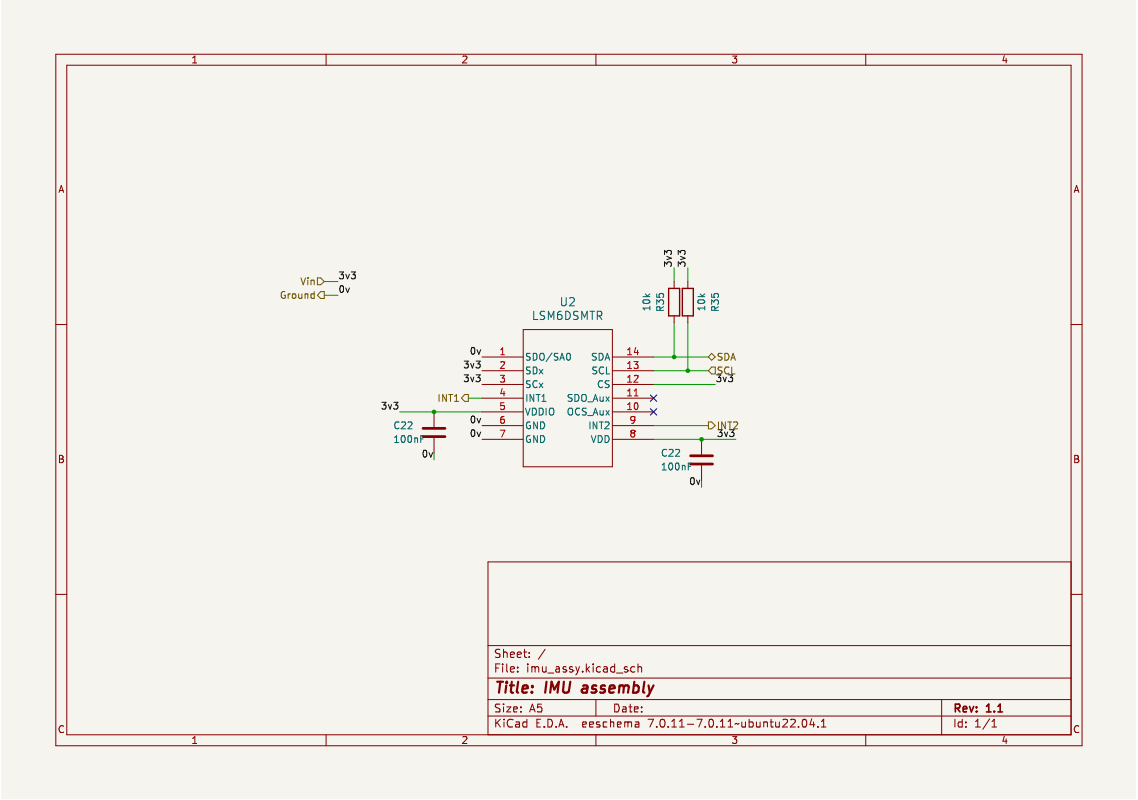


Figure 22: Circuit diagram of the Mk1.1 IMU assembly, present in the IMU board, shoulder and hand control units

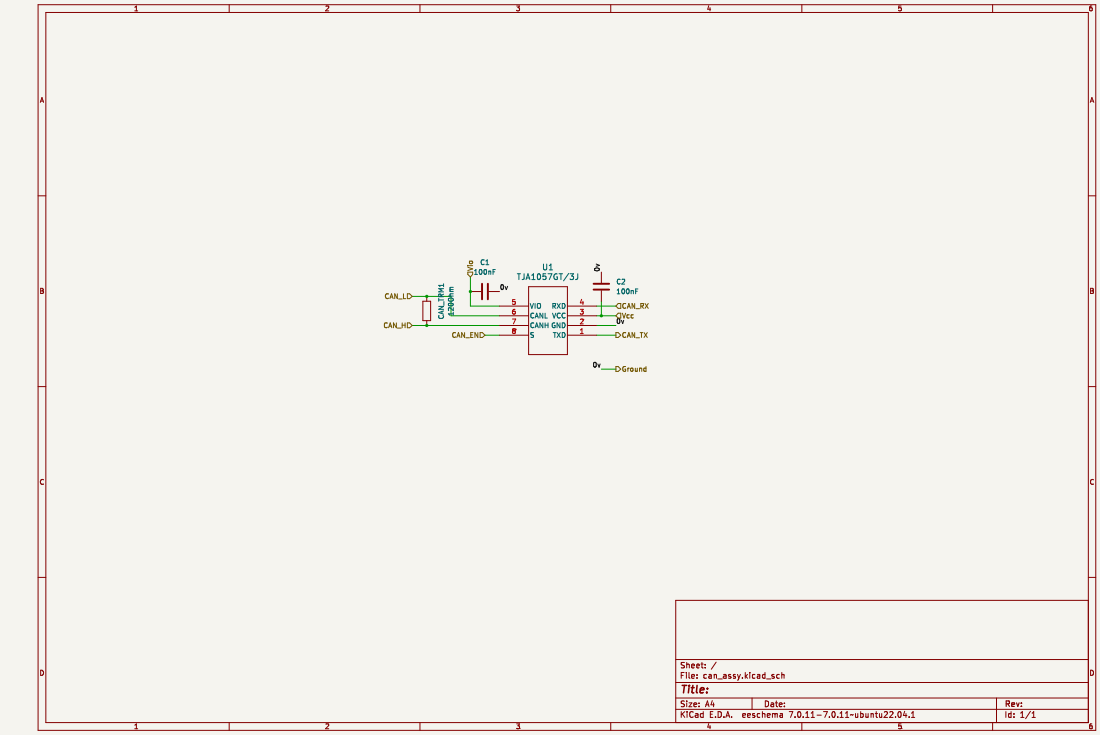


Figure 23: Circuit diagram of the Mk1.1 CAN assembly, present in all control units. Note that the schematic shows the erroneous GT transceiver instead of the correct BT transceiver

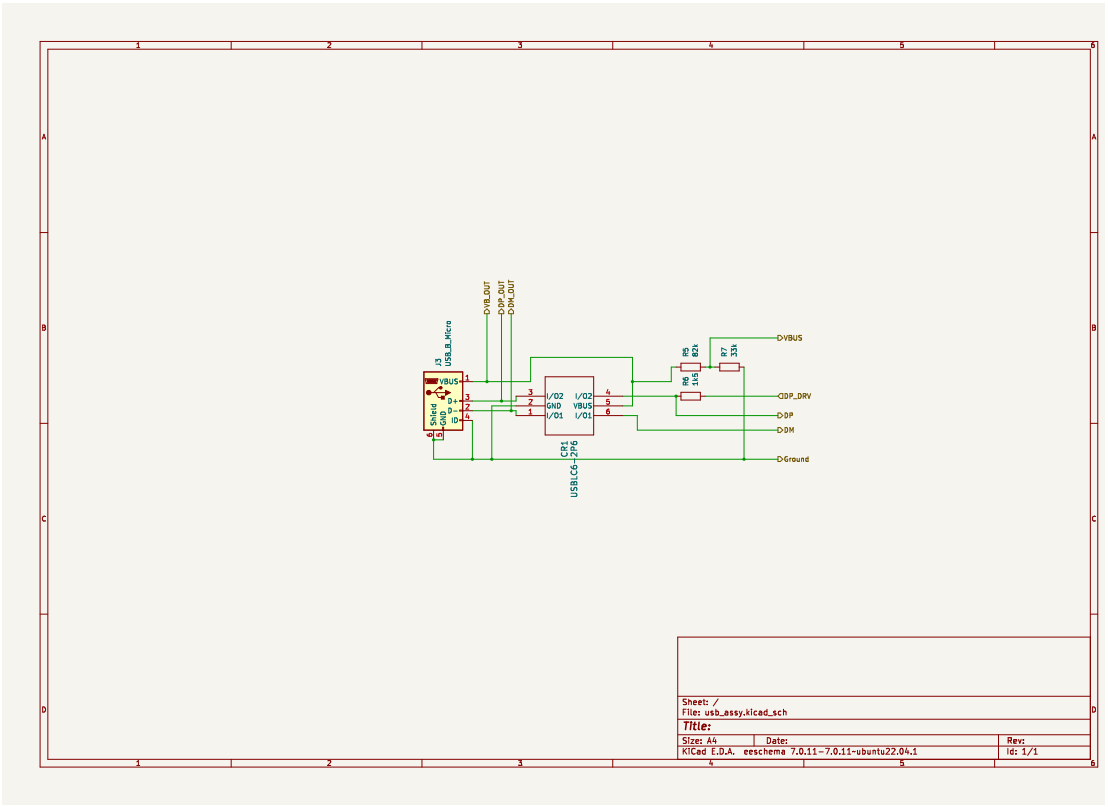


Figure 24: Circuit diagram of the Mk1.1 USB assembly, present in the torso and shoulder control units

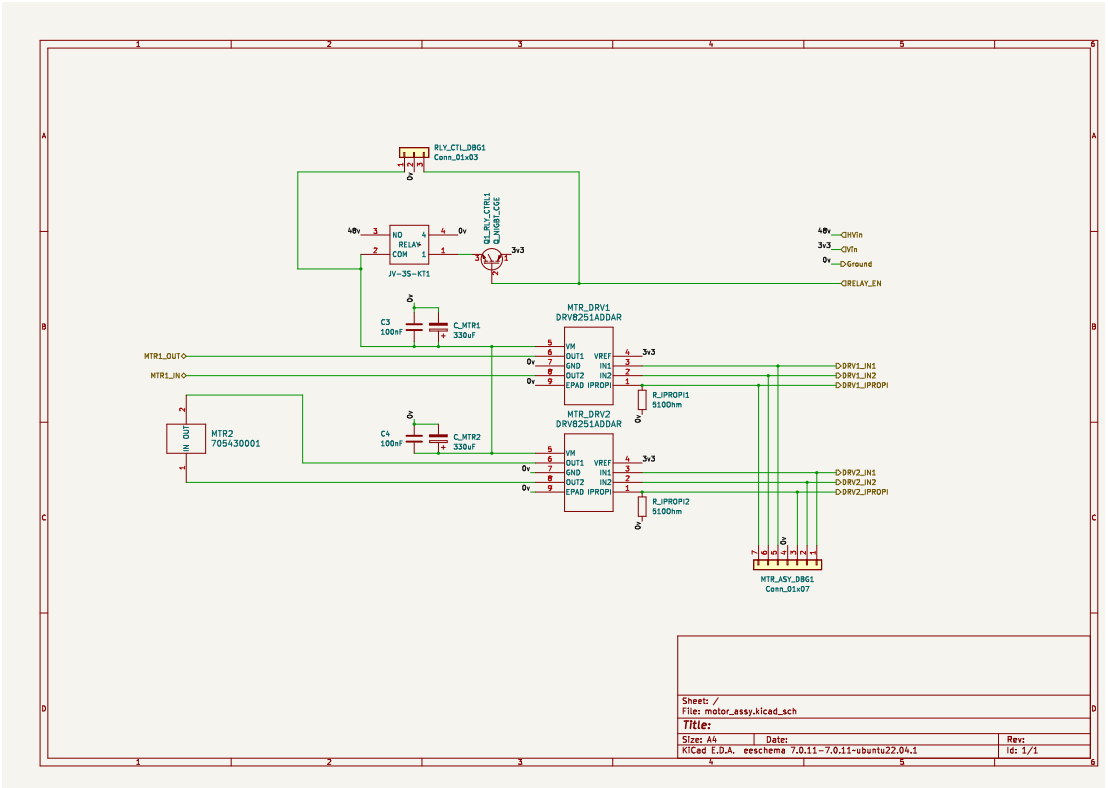


Figure 25: Circuit diagram of the Mk1.1 motor driver assembly, present in the control units

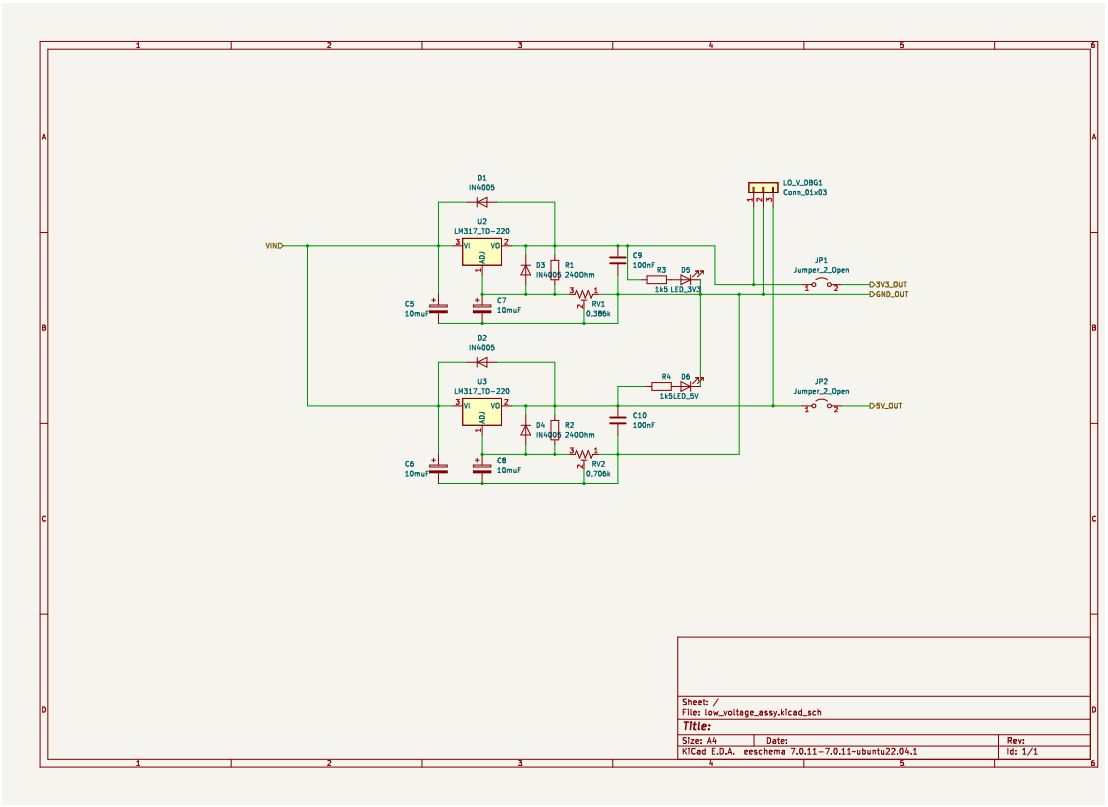


Figure 26: Circuit diagram of the Mk1.1 voltage regulator assembly, present in the control units

8 Peripheral configuration

This section presents the configuration of peripherals and certain core functions of the STM32F303RE microprocessor. Peripheral usage lays the foundation for the software architecture presented in section 9, and configuration was done in STM32CubeMX, see section 8.2.

8.1 STM32F303 overview

The STM32F303RE microprocessor unit ("the MCU") couples an Arm Cortex M4 core with several peripheral units such as ADCs/DACs, multipurpose timers and communication interfaces, as well as advanced memory functions such as direct memory access (DMA) and memory area protection. It has a nested vectored interrupt controller (NVIC) with 73 channels, enabling most peripherals to be associated with at least one interrupt channel. With 512KB of flash memory, it could also fit a basic operating system such as FreeRTOS without memory expansions. It operates at a maximum frequency of 72MHz.

This project does not utilise the F303 to its full extent, and only functionality relevant to the project is presented here. Primary sources for the configuration and usage of the STM32F303 have been the MCU datasheet[16], reference manual RM0316 for the STMF3 family[14] and user manual UM1786 for the Harware Abstraction Layer (HAL) software library[15].

8.2 STM32CubeMX

As mentioned in section 6.1, ST provides a GUI tool for the configuration of their microprocessors called STM32CubeMX ("CubeMX"). The tool generates initialisation functions in C for most functionality in the processors, with some exceptions – for instance, CAN bus message filters must

be configured in code. Upon code generation, CubeMX translates the chosen settings to HAL function arguments, preprocessor definitions et cetera and outputs .c/.h files in a folder structure with a user specified root folder. All code generated by CubeMX could have been written manually, but it was generally found to save much time which would otherwise have been spent scouring RM0316 and UM1786 for how to flip which bits in which registers. CubeMX is not open source, but is free to download upon registration with ST.

The folder structure generated by CubeMX lays the foundation for project management, and is illustrated below. The root folder is `stm_config_official`, a user chosen name. The `Core` folder contains .h and .c files for peripheral initialisation, including `main.c`. `Drivers` and `Middlewares` contain various library functions necessary for more advanced functionality such as USB, and may be configured to contain code examples[8]. The `USB_DEVICE` folder contains library functions unique to the USB peripheral. Other peripherals do not have unique folders dedicated to them.

```
EXAMPLE GENERATED FOLDER STRUCTURE
|- stm_config_official
    |- .mxproject
    |- Core
        |- Inc
        |- Src
    |- Makefile
    |- Drivers
    |- Middlewares
    |- USB_DEVICE
```

```
EXAMPLE GENERATED CODE STRUCTURE
  /* USER CODE BEGIN TIM15_Init 0 */
  /* USER CODE END TIM15_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM15_Init 1 */
  /* USER CODE END TIM15_Init 1 */
  htim15.Instance = TIM15;
  htim15.Init.Prescaler = 0;
  htim15.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim15.Init.Period = 2880;
```

Files generated by CubeMX may be edited. However, CubeMX must be configured to "keep user code when re-generating", and code written outside the `USER CODE BEGIN/END` tags may be deleted regardless. In the above example, snipped from a generated `Core` file called `tim.c`, CubeMX sets certain initialisation variables for a timer. Some of these are mirrored in the lower middle square of27, while others are opaque. All generated files have several sections dedicated to user code.

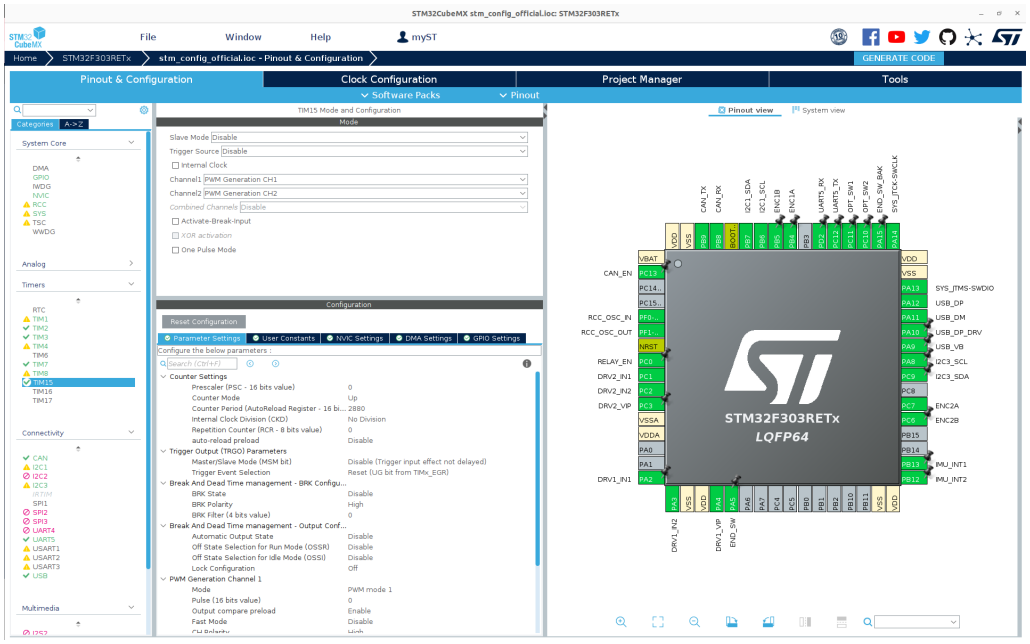


Figure 27: An example of peripheral configuration in STMCubeMX. Peripherals are selected in the left column, and configured in the middle. Mapping to pins may be selected from the MCU representation on the right, where relevant.

[8] About 2GB of code examples. Be sure to opt out!

8.3 ADC

ADCs are used to measure motor current. Channel 9 on ADC1 is pulled to pin PC1 and coupled with motor driver 2, while channel 1 on ADC2 is pulled to pin PA4 and coupled with motor driver 1, see figs 17 and 25. The ADC peripheral is described in chapter 15 of RM0316[14].

8.3.1 Parameters

Parameters not mentioned are left to their default values.

- Channels: Single-ended (measurement relative to ground)
- Resolution: ADCs are set to their maximum resolution of 12 bits
- Continuous conversion mode: Enabled
- Analog watchdog 1: Enabled
- Analog watchdog channel: Channel 9 for ADC1, channel 1 for ADC2
- Watchdog high threshold: 4000
- Watchdog interrupt mode: Enabled

8.3.2 Configuration description

The choice of channels was given by compatibility with the chosen MCU pinout, and resolution set to maximum because no advantage could be found in choosing a lower setting.

Continuous conversion mode makes the ADC start a conversion as soon as the previous has finished – "fire and forget". As soon as the ADC has been started in the program main function, it will continue working until stopped. This saves complexity in software, as the associated driver (see section 10.1) only needs to access the latest conversion result rather than triggering and waiting for a conversion.

The watchdog is part of the implementation of the safety function mentioned in section 4.3. It will trigger an interrupt if the ADC conversion result is above 4000, corresponding to a motor current draw of approximately 4A (see section 5.5), which then may be acted upon.

8.4 CAN

CAN bus is the communication backbone of this system, and CubeMX only covers transmission setup. CAN message filtering and filter configuration is covered in sections 5.6 and 9.4. The CAN bus peripheral is described in chapter 31 of RM0316.

8.4.1 Parameters

Parameters not mentioned are left to their default values

- Prescaler: 9
- Time quanta in bit segment 1: 2 times
- NVIC settings: `CAN_RX0` interrupts: Enabled

8.4.2 Configuration description

Prescaler and time quanta values were chosen to obtain a baud of 1Mb/s, which is the maximum attainable transmission rate for the MCU's CAN peripheral. The CAN peripheral has two configurable message reception FIFO queues, and an interrupt has been enabled for the first of the two, `CAN_RX_FIFO0`. As not using an interrupt for the handling of incoming messages would be impractical, this implies that FIFO1 will not be in use.

8.5 GPIO

GPIO is discussed in section 7.2, and concerns the configuration of all MCU pins.

8.5.1 Parameters

Parameters not mentioned are either discussed in other sections, or left to their default values.

- NVIC EXTI line[9:5] interrupts: Enabled
- NVIC EXTI line[15:10] interrupts: Enabled

8.5.2 Configuration description

Enabling external interrupts and events controller (EXTI) lines ensures that interrupts will be enabled for PA5, PA15, PC10 and PC11, which are reserved for use with the system's optical switches. EXTI is described in chapter 14.2 of RM0316.

8.6 I2C

I2C is used for communication with the system's IMUs. I2C1 and I2C3 are enabled, chosen because they were available. The I2C peripheral is described in chapter 28 of RM0316.

8.6.1 Parameters

Parameters not mentioned are left to their default values.

- Speed mode: Fast mode
- Frequency: 400 kHz
- Primary address length selection: 7 bit

8.6.2 Configuration description

The bus frequency of 400kHz was chosen in order to maximise bus capacity. The LSM6DSM is compatible with fast mode I2C (datasheet chapter 6.4[13]), and 400kHz was successfully tested during the specialisation project. The number of address bits is also given by the slave unit, here 7.

Other configurations, including several transmission rates, were tested during the verification process described in section 7.9.8. As the one functional unit (shoulder) remained operational at the highest possible frequency (400kHz), this setting was kept.

8.7 Clock

Clock configuration is done in a separate tab of the CubeMX GUI, and lets the user set clock frequencies for most peripherals of the MCU. Clocks are described in chapter 9 of RM0316.

8.7.1 Parameters

Parameters not mentioned are left to their default values.

- HSE input frequency: 16MHz
- PLL source mux: HSE
- System clock mux: PLLCLK
- HCLK 72MHz

8.7.2 Configuration description

The high speed external clock (HSE) frequency was set to 16MHz to match that of the system's oscillator crystal. Phased-locked loop (PLL) source was set to HSE, enabling the PLL to use the external oscillator rather than the MCU's internal 8MHz oscillator (HSI). System clock mux was set to PLLCLK, enabling the system clock to reach 72MHz rather than 16MHz. HCLK was set to the maximum frequency of 72MHz, and PLL values were calculated automatically based on this choice. Other configurations, such as using the HSI, would have resulted in invalid frequencies for the USB or other peripheral clocks. Clock configuration is illustrated in figure 28.

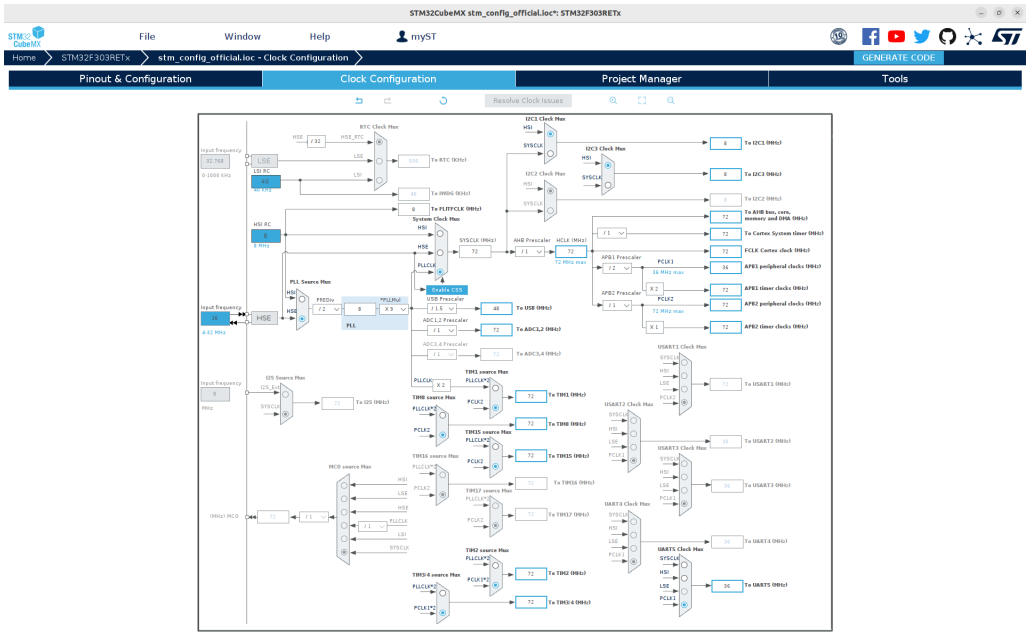


Figure 28: Clock configuration in CubeMX. The GUI presents a number of configration options to the user for tuning clock frequencies of various functions. Prescaler values are generally calculated automatically, and will yield error messages if the user attempts to set an invalid configuration.

8.8 Timers

Timers have multiple roles in the system. They act as encoder input channels, PWM signal generators and interrupt generators for various tasks discussed in section 9.3. The timer peripherals

are described in chapters 20, 21 and 22 of RM0316.

8.8.1 Encoder timers: TIM3, TIM8

TIM3 and TIM8 were configured for encoder input and were pulled to pins PB4 and PB5, and PC6 and PC7, respectively.

8.8.1.1 Parameters Parameters not mentioned are left to their default values.

- Combined channels: encoder mode
- Counter period: 65535

8.8.1.2 Configuration description Selecting encoder mode enables the timer to automatically update its counter register based on the state of its two input pins, connected directly to the output pins of the encoder. The counter register is 16 bits wide, and setting the counter period to 65535 makes use of the whole register. Depending on the resolution of the encoders versus the mechanical movement range of the relevant joint, it may be necessary to handle overflow/underflow of the register.

8.8.2 PWM generators: TIM1, TIM15

TIM1 and TIM15 were configured for PWM generation and pulled to pins PC1 and PC2, and PA2 and PA3, respectively. They were chosen because they are compatible with PWM generation and a pin placement practical for PCB design.

8.8.2.1 Parameters Parameters not mentioned are left to their default values.

- PWM generation: CH2 and CH3 on TIM1, CH1 and CH2 on TIM15
- Counter period: 2880

8.8.2.2 Configuration description Selecting PWM mode enables the timer to output a PWM signal on the selected channels. The PWM frequency is determined by a relation between the system clock frequency and the counter period register, while the PWM duty cycle is set in the timer's capture/compare register. The counter period of 2880 corresponds to a PWM frequency of approximately 25kHz, which is outside the human audible range of 20kHz. The counter period was initially set to 7200, corresponding to a PWM frequency of 10kHz, but this was found to be disturbing to humans nearby the arm as the motor drivers tend to emit a constant noise of the same frequency[9]. Timer frequency calculations are described in section 5.8.

8.8.3 Interrupt timers: TIM2, TIM4, TIM7

TIM2, TIM4 and TIM7 were activated to serve as periodic interrupt generators, primarily to prevent overloading of data buses.

[9]Presumably, as no appropriate testing equipment was available

8.8.3.1 Parameters

Parameters not mentioned are left to their default values.

- TIM2 prescaler: 7199
- TIM2 counter period: 29
- TIM4 prescaler: 719
- TIM4 counter period: 10
- TIM7 prescaler: 7199
- TIM7 counter period: 200
- NVIC settings: global interrupts enabled for all three timers

8.8.3.2 Configuration description

Prescaler and counter period values were calculated such that the capture/compare registers of each timer would reach the counter period value at frequencies of 345Hz, 10kHz and 50Hz respectively, with interrupts triggering on this event. Frequency calculations are discussed in section 5.8, and the reasoning behind the target frequencies is presented in 9.3.

8.9 UART

The UART peripheral is responsible for communication with the external computer, and UART5 was routed to pins PC12 and PD2. UART5 was chosen over other UART peripherals due to PC12 and PD2 being easy to trace to the UART header pin on the torso control unit (see 11). USART/UART is described in chapter 29 of RM0316.

8.9.1 Parameters

Parameters not mentioned are left to their default values.

- Mode: Asynchronous
- Baud: 115200 b/s
- Word length: 8 bits, including parity
- Parity: None
- Stop bits: 1

8.9.2 Configuration description

Bitrate was set to 115200 as this was the highest commonly used bitrate which was successfully tested. Remaining parameters are default, but are included in the interest of documentation.

8.10 USB

The USB peripheral is responsible for communication with the external computer, as an alternative to UART, and is routed to pins PA[9..12]. The USB peripheral of the STM32F303RE may act as a USB full-speed device according to the USB2.0 standard, utilising an internal USB physical interface. USB is described in chapter 31 of RM0316 as well as AN4879[12].

8.10.1 Parameters

Parameters not mentioned are left to their default values.

- USB device (FS): enabled
- Middleware/usb_device class for FS IP: Communication device class (Virtual COM port)

8.10.2 Configuration description

In addition to enabling the peripheral device, a set of hardware drivers provided by ST was included via the "Middlewares" section in CubeMX. These drivers provide library functions similar to the HAL.

9 Software architecture

This section presents the structure of the software written during this project. This includes naming conventions, development patterns and other general considerations which informed the implementation of modules described in section 10.

9.1 Implementation of Architectural Requirements

Low coupling between modules to simplify maintainability. Developing software around hardware; embedded programming. Naming conventions

9.1.1 Naming conventions

9.1.1.1 Definition: Module A module is defined as a set of .c and .h files sharing the same name. A module has a specific purpose or is associated with one specific category of hardware. As the C language has no clear definition of classes/objects, modules function as a conceptual replacement. *Example:* The CAN bus module consists of the `can_driver.c` and `can_driver.h` files.

9.1.1.2 Module names Modules are named by their role in the system, and adhere to one of the following patterns:

- `<noun>_driver`: Module is responsible for interaction with the hardware type given by `<noun>`. These modules represent the abstraction level above the CubeMX generated modules, and typically make use of the HAL functions associated with that module.
- `<noun>_controller`: Module is responsible for control of the hardware type given by `<noun>`. These modules represent the abstration level above `_driver`, and may make use of several of those modules.
- `<noun>_parser`: Module is responsible for parsing input from the UART peripheral.

9.1.1.3 Function names Functions are typically named by what module they belong to and whether they are part of the module's interface, analogous with a public/private modifier in other languages.

- `<module>_interface_<verb>_<noun>`: `interface` indicates that the function may be used outside of the module, and uses a short form of the module name.
- `<module_name>_<verb>_<noun>`: Function does something with the data type indicated by `<noun>`, and uses the full name of the module. These functions may not be used outside

Verbs indicate what the function does to the data type indicated by `<noun>`, and generally fall into one of the following categories:

- `get/set/clear`: Function manipulates data in a struct associated with the module.
- `calculate`: Calculates a value indicated by `<noun>`, according to a given algorithm.
- `update`: Renew data in a struct associated with the module. Typically incorporate a call to the associated `calculate` and `set` functions.
- `handle`: Exclusive to the CAN module, indicates that the function handles an incoming CAN message of a type indicated by `<noun>`.

9.1.2 Externally accessible information

9.1.3 Development patterns

Structs as hardware representation. Lists of structs. Lists of functions.

9.2 Arm onboard

Input: UART/USB Structure: Peer-to-peer vs master/slave? MCUs are equal, but torso deals with external comms. Backbone: CAN bus Module inclusion: Preprocessor statements Folder structure: CubeMX vs TTK4900 ROS vs Terminal USB vs UART Timer based interrupts

9.3 Interrupts

9.4 CAN bus

9.4.1 CAN bus message ID structure

9.5 ROS nodes

10 Hardware drivers

Point out that these are the TTK4900 drivers, not CubeMX generated files. Not including joint controllers, can executives; these are control system domain. Defining positive and negative direction for the joints, make figures

10.1 ADC driver

11 Control system

Main loop, joint controllers, can executives, state machine

12 Calibration

Method of calibration, usage in the state machine.

13 ROS nodes

14 Tests

Message round trip times: CAN, I2C, accelerometers=(CAN+I2C)

15 Results

Regret not putting indicator LEDs on GPIO, would have made basic testing easier. The optical sensor shenanigans could have been avoided if the original sensors had been studied more carefully. State machine is whack, inconsistent calibration for twist for whatever reason Accelerometers are whack, should have had proper debug headers System works well overall when accelerometers are disabled ROS is really powerful, and can probably be expanded readily CAN bus craps out for voltages above 25V

The decision to place the upper arm IMU on the shoulder control unit PCB made it impossible to debug, and may be considered a violation of the project's second goal (or requirement number NUMBER) as the whole board will need to be replaced in order to repair/replace the IMU.

USB non-functional. Could it be due to the clamping voltage in the rail TVS? Or not implementing VBUS ESD protection properly, see an4879 2.3.

Input voltage no more than 25V, far less than the suggested 48V. Heat, motor control, CAN bus.

Low coupling between modules? Not really, look at the include graph

Were the ARs adhered to? More or less

16 Discussion

Pros and cons of using youtube as a source: bootstrap large project, making other people's mistakes. Not having access to a debugger was frustrating DMA and peripheral interconnect: could using them have increased i2c or can message rates? FreeRTOS: would be cool, and relevant with the amount of nonsynchronous stuff going on. CubeMX, despite landing solidly in the category of "corporate shitware" was easy to use and saved a lot of time - not a change I would have made it to ROS if I had had to flip all them bits.

Choosing UART5 instead of literally any other USART port may have limited communication options. Everything relating to handling of the accelerometer inData structs in the joint controller module should have been in a separate accelerometer controller module. This is a gross violation of AR1.2

17 Conclusion

18 Operations

18.1 PCB installation

18.2 MAIN connector usage

18.3 Gripper

18.3.1 Installation

18.3.2 Manufacture

18.4 Control software usage

Bibliography

[1] Adafruit. *Adafruit MMA8451 Accelerometer Breakout*. 2023.

[2] Kristian Blom. 'From hardware to control algorithms: Retrofitting a legacy robot using open source solutions'. MA thesis. The Norwegian University of Science and Technology (NTNU), 2023.

[3] Bourns. *CDSC706-0504C - Surface Mount TVS Diode Array*. 2015.

[4] TT Electronics. *Photologic® Slotted Optical Switch OPB960, OPB970, OPB980, OPB990 Series*. 2019.

[5] Escap. *escap 23LT12 graphite/copper commutation systems*. 2024.

[6] U. Ghia, K. N. Ghia and C. T. Shin. 'High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method'. In: *Journal of Computational Physics* 48 (1982), pp. 387–411.

[7] DIODES incorporated. *ZMR series: fixed 2.5, 3.3 and 5 volt miniature voltage regulators*. 2013.

[8] Pittman Metek. *Brush commutated DC motors DC030B Series*. 2024.

[9] Pittman Metek. *Brush commutated DC motors DC040B Series*. 2024.

[10] Pittman Metek. *Brush commutated DC motors DC054B Series*. 2024.

[11] NXP. *TJA1057 High-speed CAN transceiver*. 2023.

[12] ST. *AN4879: Introduction to USB hardware and PCB guidelines using STM32 MCUs*. 2023.

[13] ST. *LSM6DSM iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope*. 2017.

[14] ST. *ST RM0316: Reference manual STM32F303xB/C/D/E, STM32F303x6/8, STM32F328x8, STM32F358xC, STM32F398xE advanced Arm®-based MCUs*. 2024.

[15] ST. *ST UM1786: Description of STM32F3 HAL and low-layer drivers*. 2020.

[16] ST. *STM32F303xD STM32F303xE*. 2016.

[17] ST. *UM1724: User manual STM32 Nucleo-64 boards (MB1136)*. 2020.

[18] Avago Technologies. *HEDS-9000/9100 Two Channel Optical Incremental Encoder Modules*. 2016.

[19] TexasInstruments. *DRV8251A 4.1-A Brushed DC Motor Driver with Integrated Current Sense and Regulation*. 2022.

[20] TexasInstruments. *LMx17HV High Voltage Three-Terminal Adjustable Regulator With Overload Protection*. 2015.

Appendix

A Hello World Example

```cpp
int main {
    // This is a comment
    std::cout << "Hello World from C++!" << std::endl;
    std::cout << "I am using the default style to print this code in beautiful
    ↪  colors. Since the text is so long I have to include the 'breaklines'
    ↪  option as well" << std::endl;
    return 0;
}
```

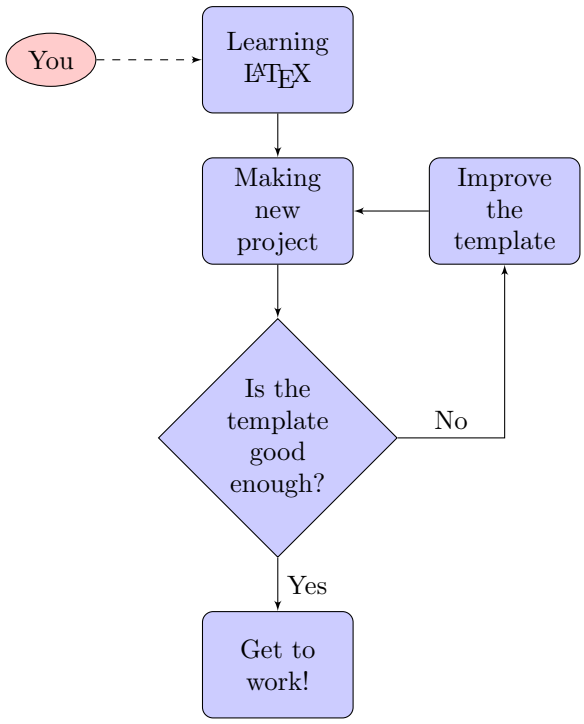
⋮

```python
# This is a comment
print('Hello world from Python!')
print('I am using the "rrt" style to print this code in beautiful colors')
```

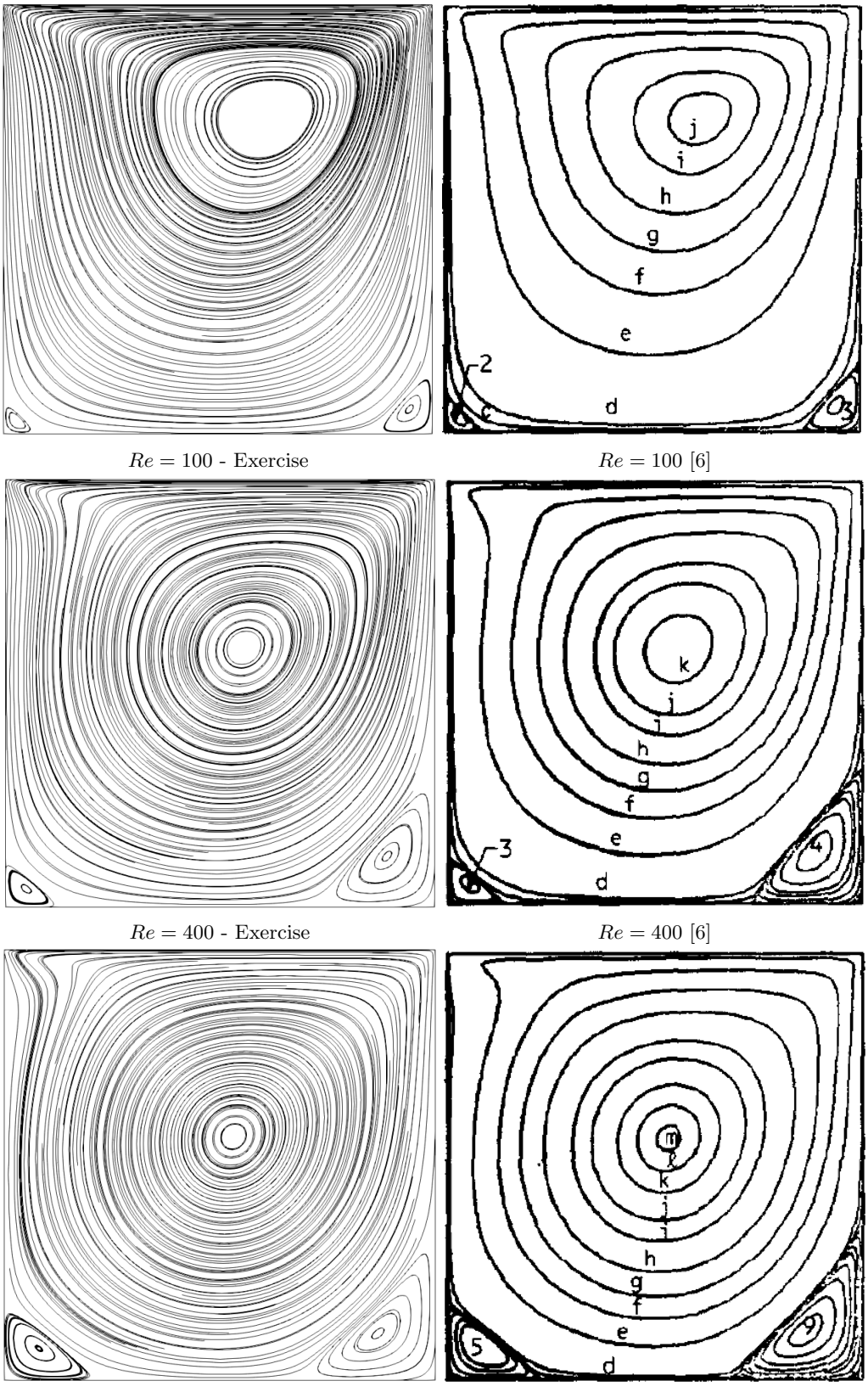
⋮

```matlab
% This is a comment
disp("Hello World from MATLAB!");
disp("I am using the "tango" style to print this code in beautiful colors");
```

B Flow Chart Example



C Sub-figures Example



$Re = 100$ - Exercise $Re = 100$ [6]

$Re = 400$ - Exercise $Re = 400$ [6]

$Re = 1000$ - Exercise $Re = 1000$ [6]

Figure 29: Streamlines for the problem of a lid-driven cavity.