Read the Docs Template Documentation

Release 1.0

Read the Docs

Jan 08, 2021

Contents

1 About Faze4 **3**
- 1.1 General info . . . 3
- 1.2 Table of contents . . . 3
- 1.3 Check the arm in action ! . . . 6
- 1.4 Support the project . . . 6
- 1.5 Thanks to the: . . . 6

2 Design decisions **7**
- 2.1 Number of axis . . . 7
- 2.2 Spherical wrist . . . 7
- 2.3 Cycloidal reducer . . . 7
- 2.4 Motor position . . . 12
- 2.5 Reduction ratios: . . . 13
- 2.6 Belts . . . 13

3 Building **15**
- 3.1 BOM . . . 15
- 3.2 Actuator . . . 15

4 Printing **17**
- 4.1 Filament . . . 17
- 4.2 Printer . . . 17
- 4.3 Print settings and tips . . . 17

5 Electronics, PCB and wiring **19**
- 5.1 Faze4 distribution PCB V2 . . . 19
- 5.2 Board use guide . . . 21
- 5.3 PCB V1 (old, not recommended) . . . 21

6 Low level code **31**
- 6.1 Tennsy environment setup . . . 31
- 6.2 Teensy low level code for testing in Matlab(live script) . . . 32
- 6.3 Teensy low level code for Matlab trajectory planning . . . 34
- 6.4 Teensy low level code for ROS . . . 34

7 High level code **35**
- 7.1 Setting up Matlab . . . 35

7.2 Matlab code for testing in live script 35
7.3 Matlab trajectory planning 36
7.4 ROS 36

8 Startup and setup **37**

9 Robot data **39**
9.1 General info 39
9.2 Dimensions 39
9.3 Reduction ratios: 39
9.4 Belts 39
9.5 Stepper data 41
9.6 PSU 41

Contents:

CHAPTER 1

About Faze4

1.1 General info

Faze4 robotic arm came to be when i made a working 3d printable cycloidal gearbox and saw some potential in it. After that i designed whole arm around it. It uses 6 stepper motors, so it is a 6 axis robot arm. While designing this arm i focused on low backlash and good design of arm. All wires are hidden and arm is really rigid even tho it is 3d printed. Total cost of the arm is around 1000 - 1500 dollars.

Note: This project is still work in progress. Part that is work in progress is control cabinet, that means that the arm can be built and it will work without the cabinet.

Esthetic design was inspired by FANUC's LR Mate 200iD. Goal was also to hide all wires in the arm like most industrial arms do. Only visible wires (or pipes) would be ones for gripper.

Weight of the arm is around 14-15 kg , but it can be reduced by printing with less infill.

Low level control for the arm runs on teesny 3.5, while high level (inverse kinematics, simulations) are only on Matlab code atm, but I am working on ROS implementation and porting my Matlab code to Python.

1.2 Table of contents

On this read the docs page you will find:

- How is arm desinged and desing decisions
- Printing tips
- Electronics guide and PCB
- Building instructions for the arm
- All code and programs used
- How to test and troubleshoot

- Sources
- Where to buy parts

1.3 Check the arm in action !

1.4 Support the project

This project is completely Open source and free to all and I would like to keep it that way, so any help in terms of donations or advice is really appreciated. Thank you!

Donate

1.5 Thanks to the:

- Peter Corke for his amazing robotics toolbox for Matlab
- Maximilian_Schommer for his cycloidal drive generator

CHAPTER 2

Design decisions

2.1 Number of axis

I went with 6 axis because with 6+ axis we can reach same dot in space with different orientations. Both pictures show arm at position x=0.3m , y=0.3m and z=0.2m but as you can see orientations are different.

2.2 Spherical wrist

This arm uses so called spherical wrist. Spherical wrist is configuration of joints where axes of rotation for joints 4,5,6 intersect. You can see that in picture below. It is complicated mechanical system that in turn simplifies solving inverse kinematics for robot arm. You will see this configuration in a lot industrial robot arm.

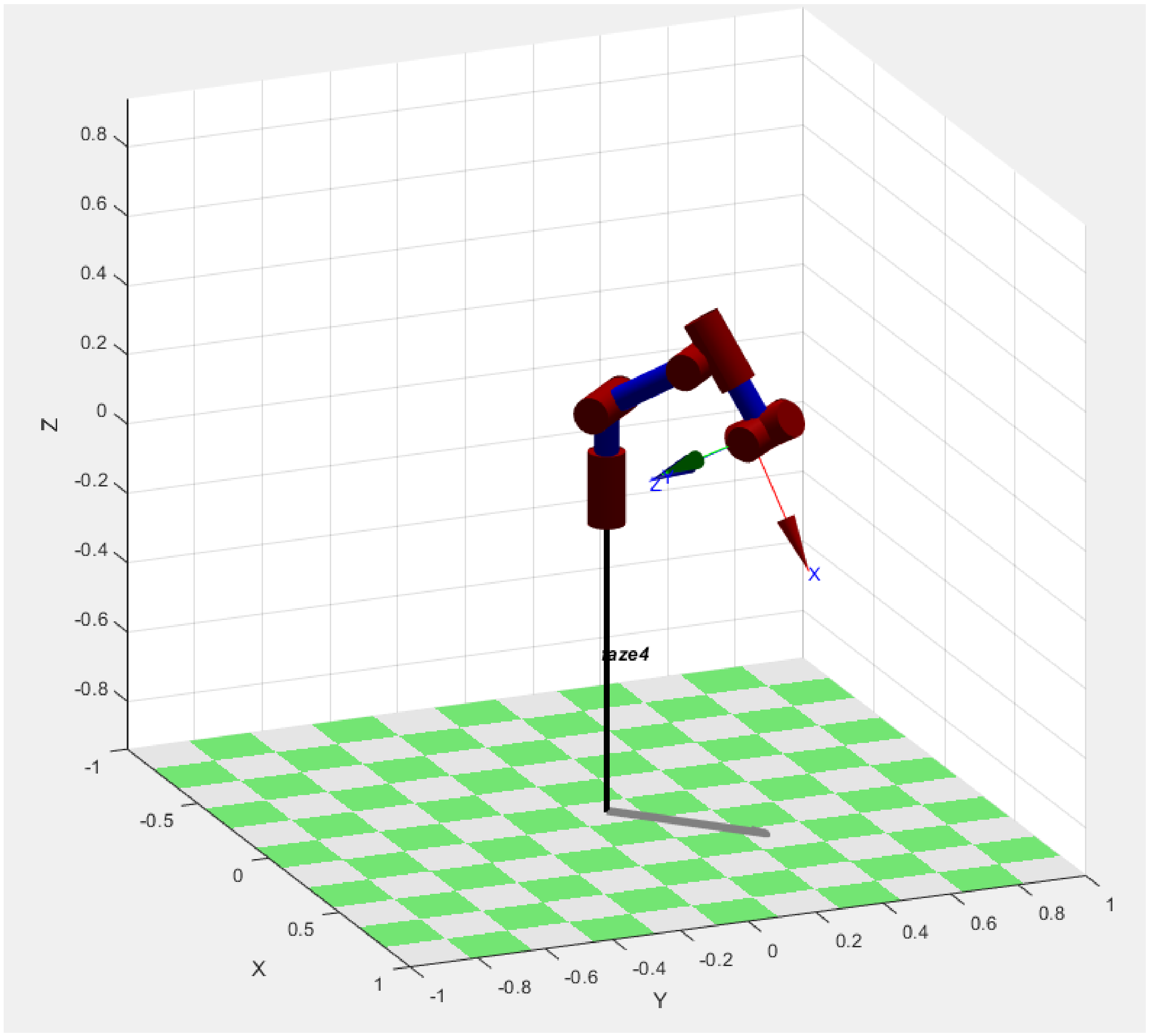
All wires are routed thru the middle of the arm and cant be seen.

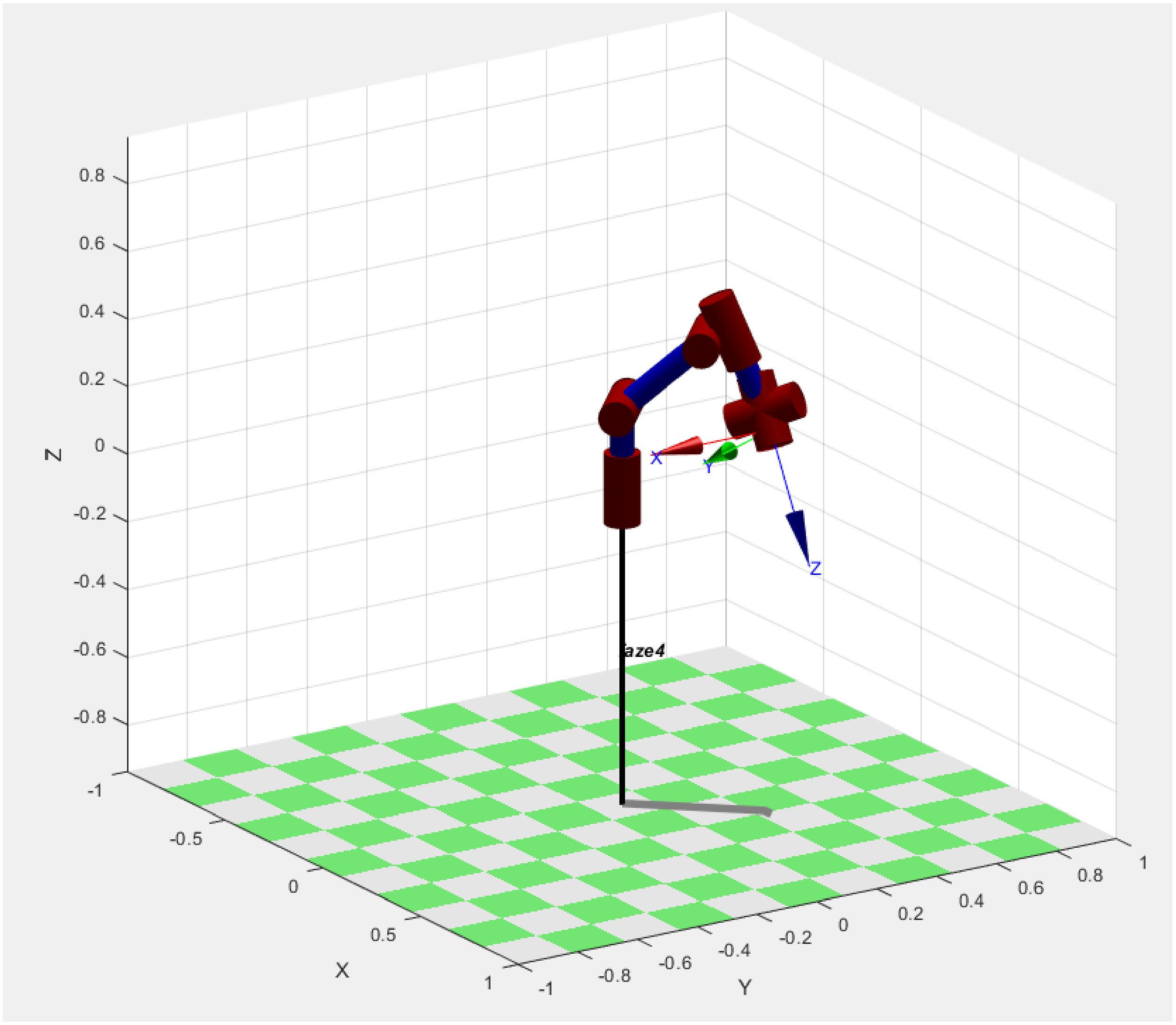
2.3 Cycloidal reducer

Why cycloidal reducers?

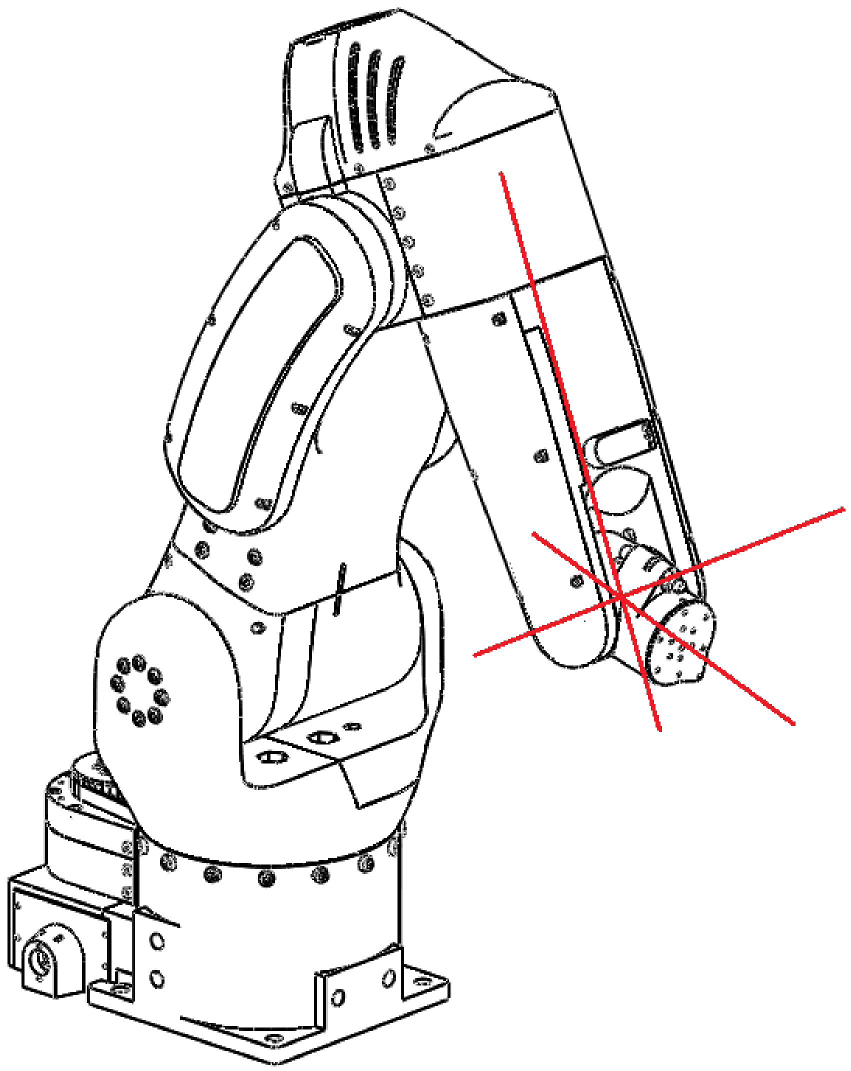
- They are really easy to 3d print
- can have large reduction ratio
- cheap
- low backlash even when 3d printed

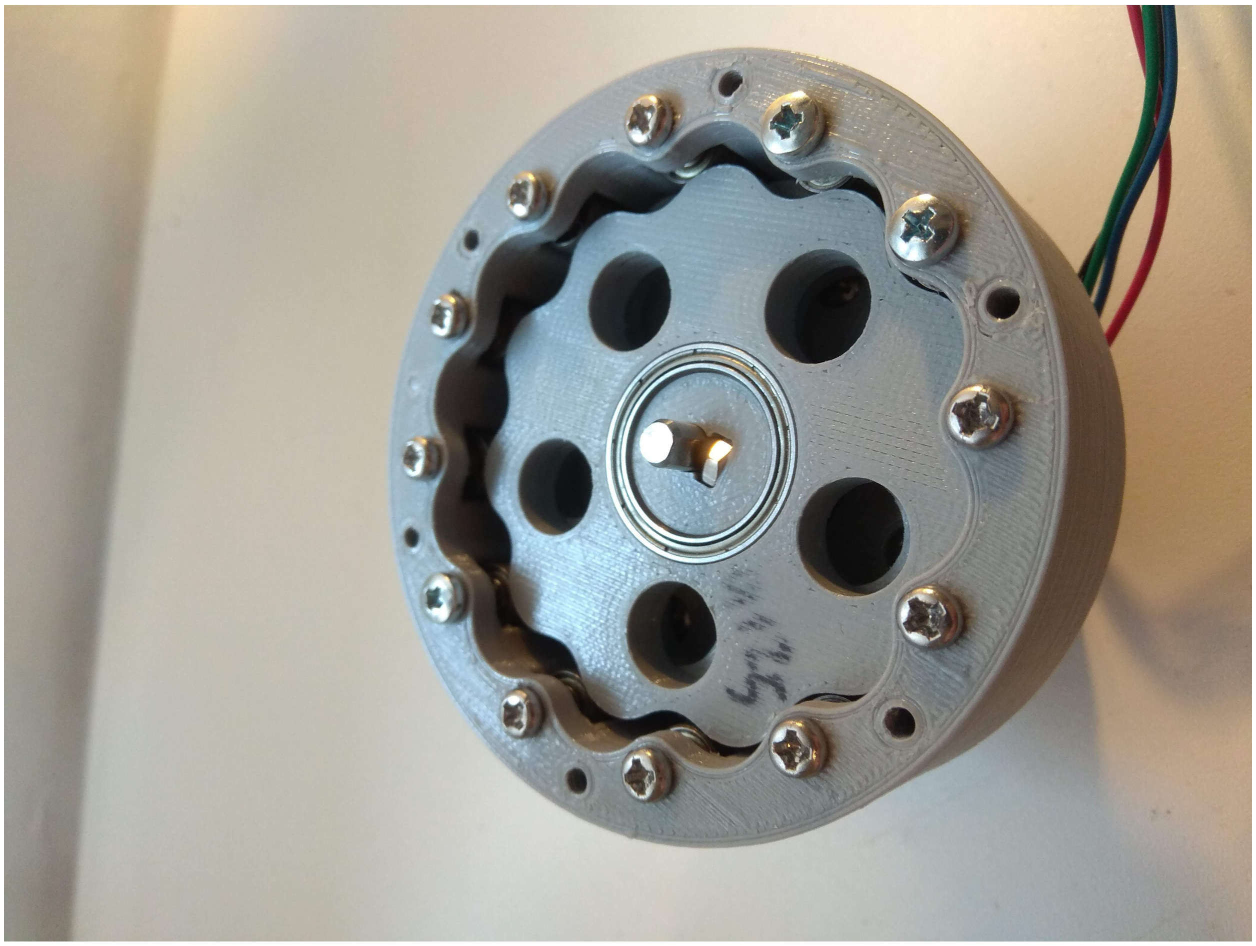
Note: When 3d printing them there are few things to keep in mind but that is discussed in sections troubleshooting and printing.


0.8
0.6
0.4
0.2
Z
0
-0.2
-0.4
-0.6
-0.8
-1
-0.5
0
0.5
X
1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
Y
Z
Y
X
aze4



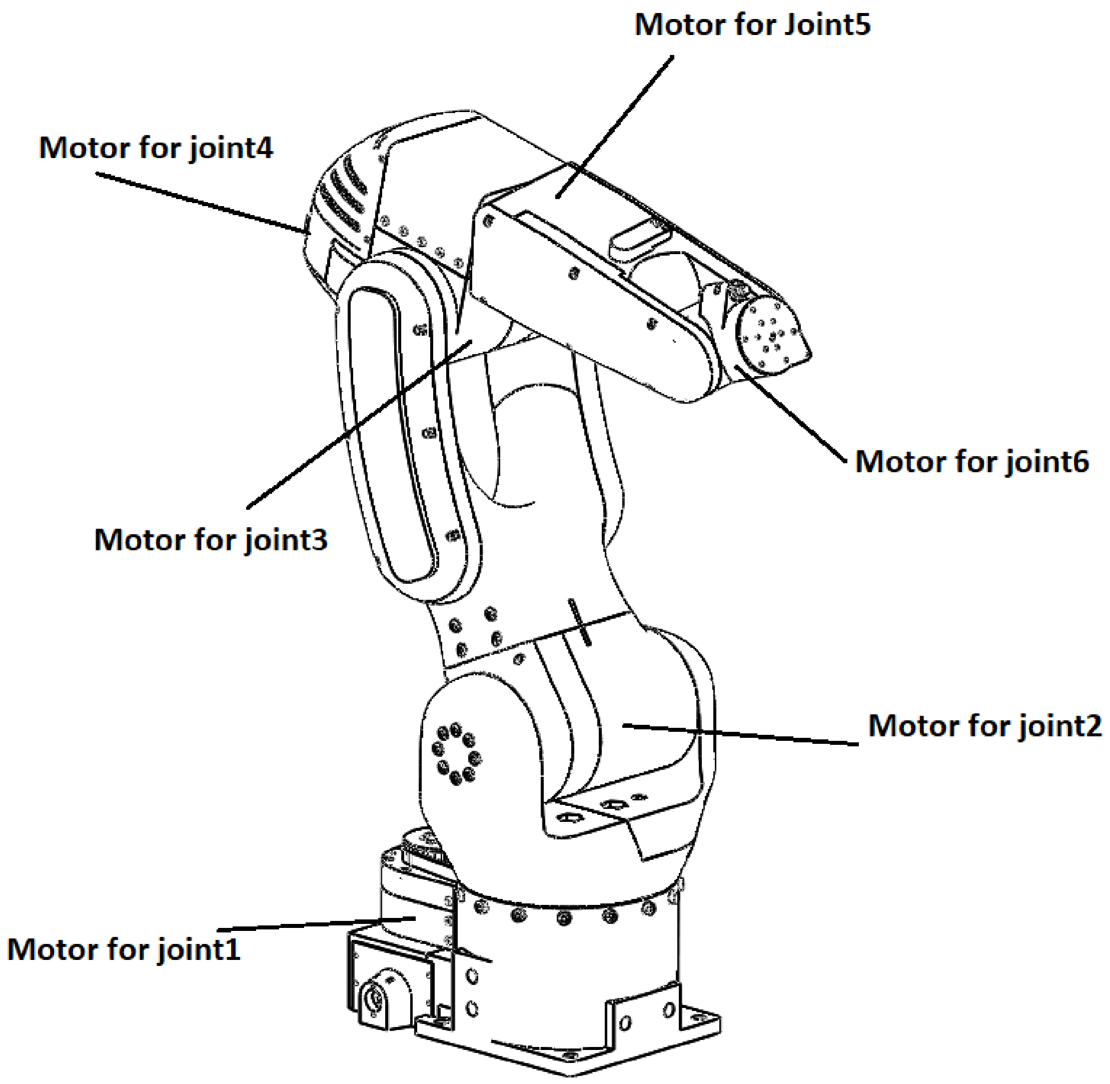
Z
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-0.5
0
0.5
1
X
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
Y
aze4
X
Y
Z





2.4 Motor position

Firstly i wanted to move all motors to the base of the arm to reduce weight each motor should carry. First idea was to just copy the design of this(link) kuka arm. I decided against that design and just went with probably most basic design where every motor is directly on the joint of actuation (Except for joint 5 that i moved a little bit away).This is not good idea actually and if I was redesigning this arm I would try to move at least joint 3 to the bottom of the arm, but what is done is done.



Joints 1,4 and 5 use belts in addition to cycloidal reducers. Belts are mostly used to offset the place where joint will be rotating so that we can route wires thru the body of robot. They also give some additional reduction ratio. For example Joint 1 has 15:1 cyclo but with belt our reduction ratio jumps to 25:1.

2.5 Reduction ratios:

- Joint 1 -> 15:1 cycloidal reducer and 5:3 belt reduction = combined resulting in 25:1 reduction
- Joint 2 -> 27:1 cycloidal reducer
- Joint 3 -> 15:1 cycloidal reducer
- Joint 4 -> 11:1 cycloidal reducer and 28:26 belt reduction = combined resulting in 11.846153:1 reduction
- Joint 5 -> 11:1 cycloidal reducer and 1:1 belt reduction = combined resulting in 11:1 reduction
- Joint 6 -> 19.19:1 planetary reducer

2.6 Belts

- Joint 1 belt is HTD 5M 490 mm timing belt 10 mm wide and needs to be tightened with bearings
- Joint 4 belt is HTD 5M 430 mm timing belt 10 mm wide does not need to be tightened
- Joint 5 belt is HTD 5M 285 mm timing belt 10 mm wide and needs to be tightened with bearings

CHAPTER 3

Building

Note: All parts can be bought from ebay and similar online stores. I bought all steppers,drivers and PSUs from stepper online. You can buy screws used for this arm in online stores but i sourced them locally and that was much cheaper. I also recommend to buy wires (you will need them to extend stepper wires) locally since ones on ebay and similar stores are of questionable quality.

- Full assembly instructions are here

3.1 BOM

You can find bom in this excel document and in building instructions file.

- BOM FILE

Tip: I also recommend to buy more items like bearinngs and screws than you need since you might destroy or damage some in building process.

3.2 Actuator

Watch this video on how to assemble cycloidal actuators.

CHAPTER 4

Printing

4.1 Filament

I recommend PETG since it can withstand larger temperatures then PLA and is easier to print then ABS. Since motors can heat up i think it is best choice.

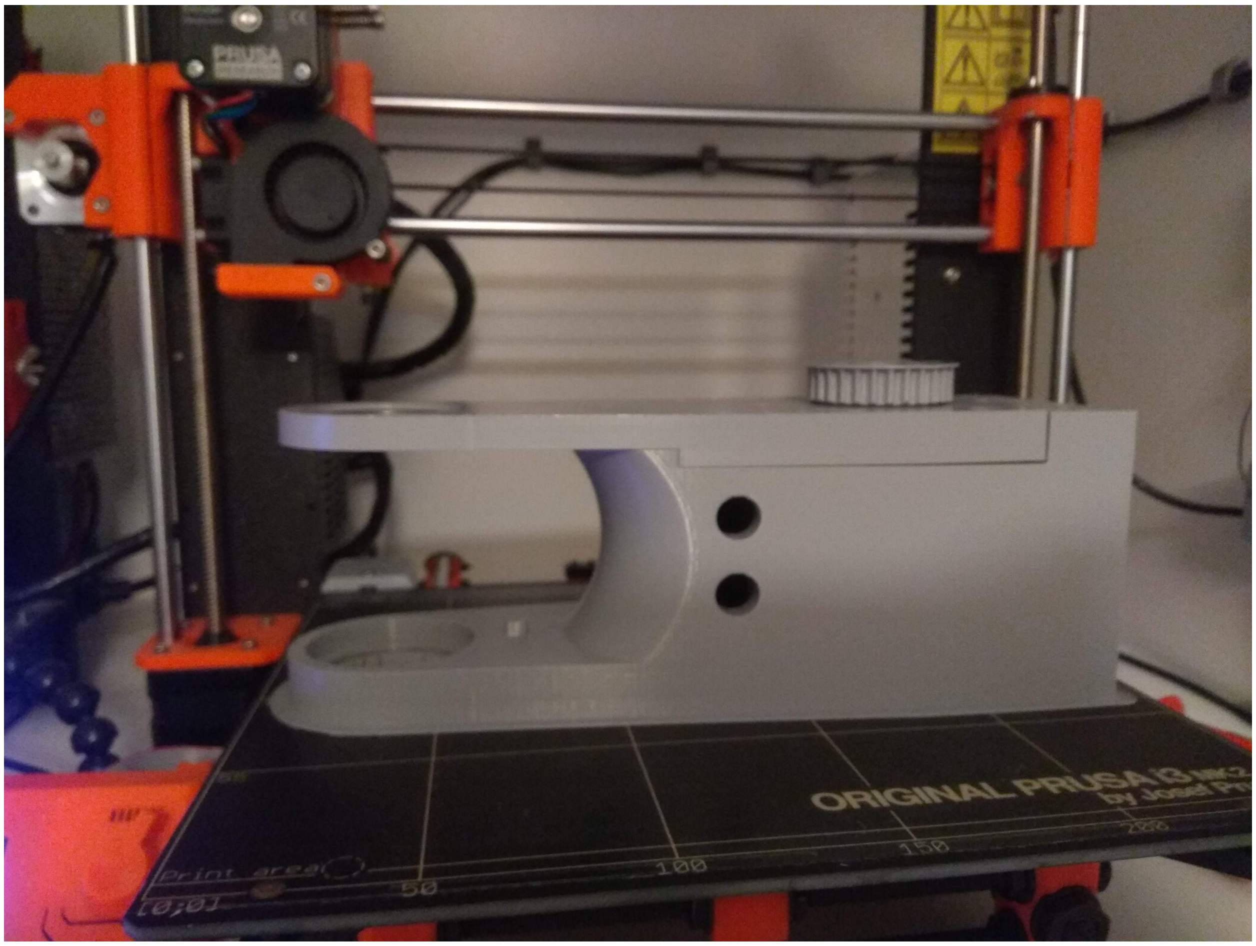
4.2 Printer

All parts are designed to be printed on Prusa MK2S, so if you have printer with build volume of 250x210x200 you are good to go.

4.3 Print settings and tips

I printed all smaller parts with 0.15 layer height and all larger with 0.35 layer height.For infill try to stay in 25%-30% range. I used 3 outer perimeters and 7 top and 5 bottom layers.

Tip: When printing larger parts like: x x x , try to print at slower speeds and with small accelerations.Since those parts are really large and heavy they tend to cause missing steps when reaching the end of 3d print due the inertia.


PRUSA
ORIGINAL PRUSA i3 MK2
Print area
[0;0]
50
100
150
200

CHAPTER 5

Electronics, PCB and wiring

All electronics can also be bought from eBay and similar sites. I used cheap stepper drivers from eBay but you can buy any stepper driver that can supply enough current for the steppers.

Check how i wired my arm in this file: FAZE4 Robotic arm electronics setup.pdf on github page: https://github.com/PCrnjak/Faze4-Robotic-arm (These instructions are for V1 version of board that is not recommended)

Tip: From experience, i found out that cheaper stepper drivers tend to make more noise, so if you want a quiet robot arm buy more expensive drivers. I foung ones from stepper online to be good.

5.1 Faze4 distribution PCB V2

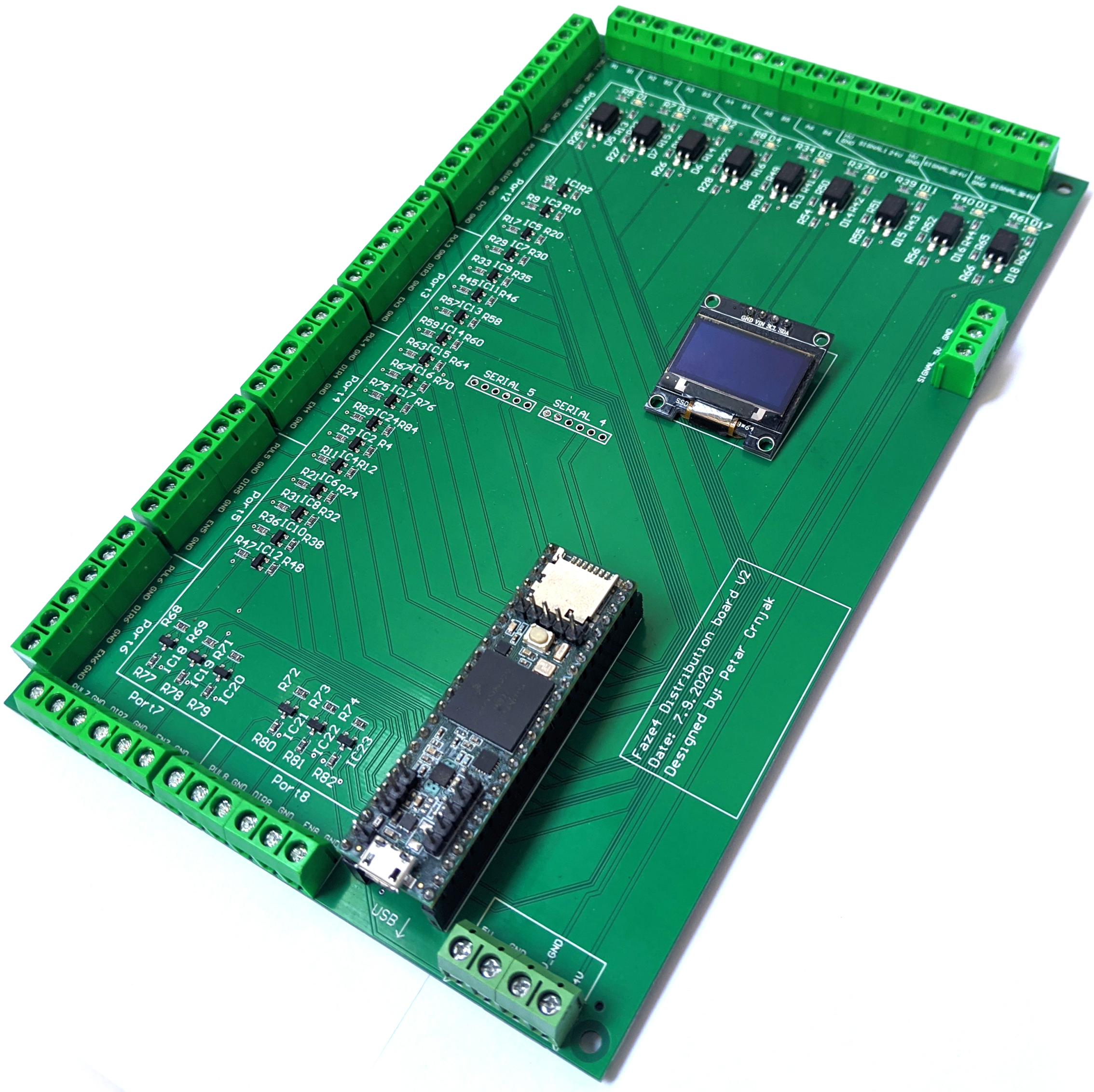
Warning: To use code seen in videos and in github repo you will need to modify some pin definitions!

- Schematic
- Board STEP files
- Board dimensions
- Test Codes

Faze4 Distribution board V2 allows you to connect stepper drivers, limit switches, sensors, displays, microcontrollers, computers, and more with each other with clean and simple wiring. It is designed for Faze4 robotic arm but you can use it for any project from CNCs, 3D printers to use in the industrial assembly line.

It is designed to use the TEENSY 3.5 microcontroller but in near future, it will support boards like with STM32 microcontrollers and Atmegas. Read more about Teensy here: https://www.pjrc.com/store/teensy35.html

Board outputs have level shifters that boost 3v3 from the microcontroller to 5V. That allows you to use a larger set of stepper drivers since they are usually designed for 5v logic. 5v is also much better for relays.


faze4 Distribution board V2
Date: 7.9.2020
Designed by: Petar Crnjak
SERIAL 5
SERIAL 4
USB

Inputs for the board support limit switches and 24V sensors (like inductive, capacitive. . .) used in the industry. Inputs are also isolated with optocouplers from a microcontroller and use current signals that ensure no voltage induction in wires can accidentally trigger on microcontroller inputs that can cause errors and damage in real life. Inputs also have LED indicators that tell you when the signal is present.

Board also has 2 UART ports routed that can extend its communication abilities.

It also supports typical 128x64 OLED displays.

When buying OLED display note the location of vcc and gnd pins.

- Order of pins should be: GND, vcc, SCL, SDA
- Check image below (LEFT ONE IS GOOD, RIGHT ONE IS BAD)

When connecting FTDI USB TTL Serial Adapter Module to PCB connect it like this. Board supports 2 modules operating at the same time!

5.2 Board use guide

Connect your teensy 3.5 to board. Once connected it uses onboard 5v regulator on teensy to level shift all 3v3 signals to 5v. If you see problems with stability of signals connect external 5V power source to the connector on the bottom right corner.

To use inputs on top of the board you NEED 24V power source. Once you connect 24V power to the bottom right connetor you can connect limit switches to the ports labled with (A1 B1, A2 B2, A3 B3 . . .) When connecting limit switches connect one end of limit switch to the A1 and other end to B1. There are total of 6 ports for limit switch inputs.

Ports labeled with HV_GND, SIGNAL, 24V are used by Industrial sensor that output 24V signal. One example is inductive sensor.

There is one port on the right side and it is used for grippers. It outputs 3v3 signal.

5.3 PCB V1 (old, not recommended)

This PCB is first version and has some errors that will be fixed in revision 2. Board was created in Altium Circuitmaker and all files can be found here:

Warning: I recommend waiting for next version since this one has alot of errors.

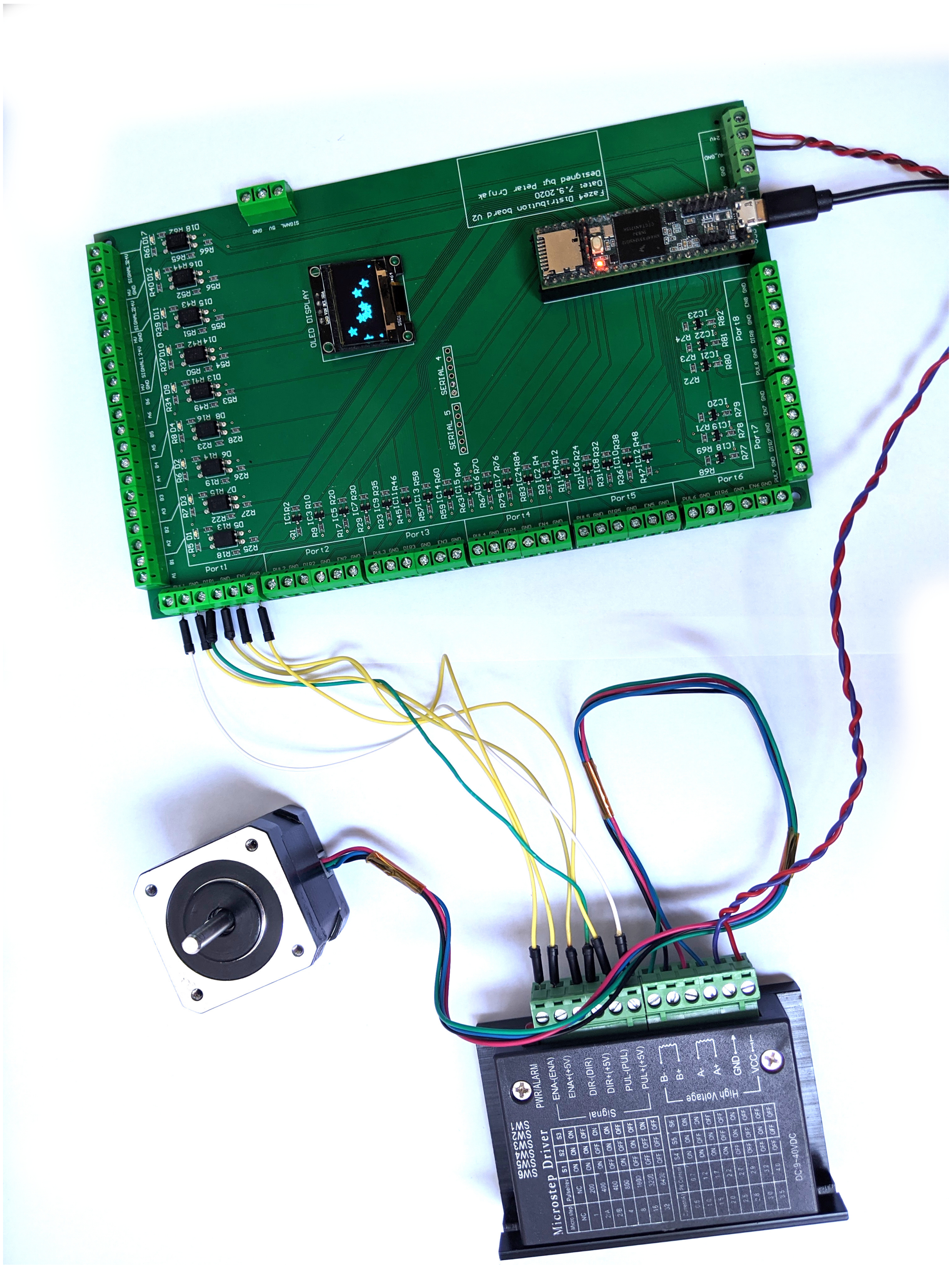
- All files for PCB

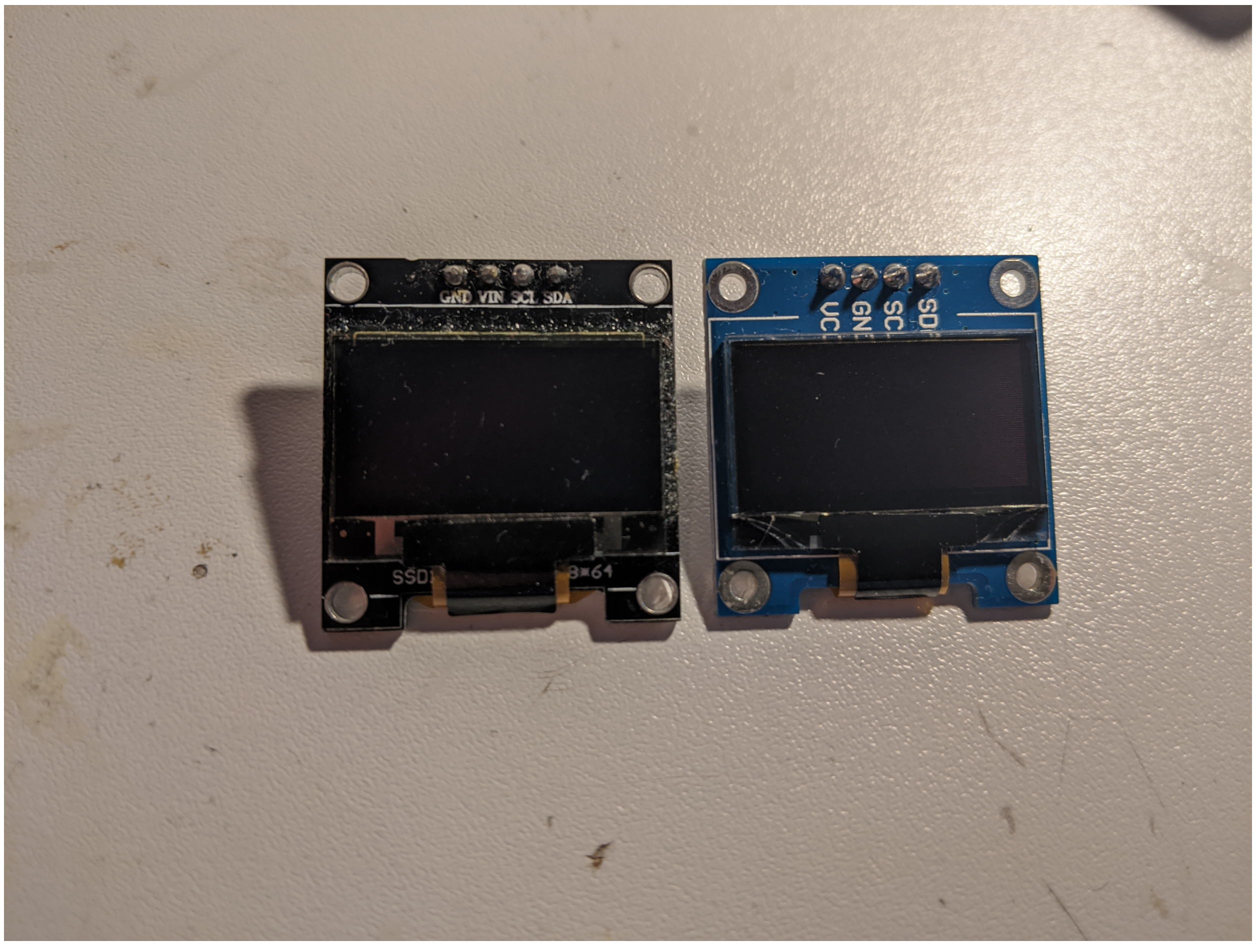
Upper section of PCB are Inputs for limit switches and inductive sensors. All inputs on top are isolated with optocouplers. I did that since all wires of the arm are going thru same spot, that means "high voltage" stepper wires were near low voltage limit switch wires. That caused problems where steppers induced enough voltage on limit switch wires to trigger interrupts on teensy. That is why limit switches and sensors are connected to 24v.

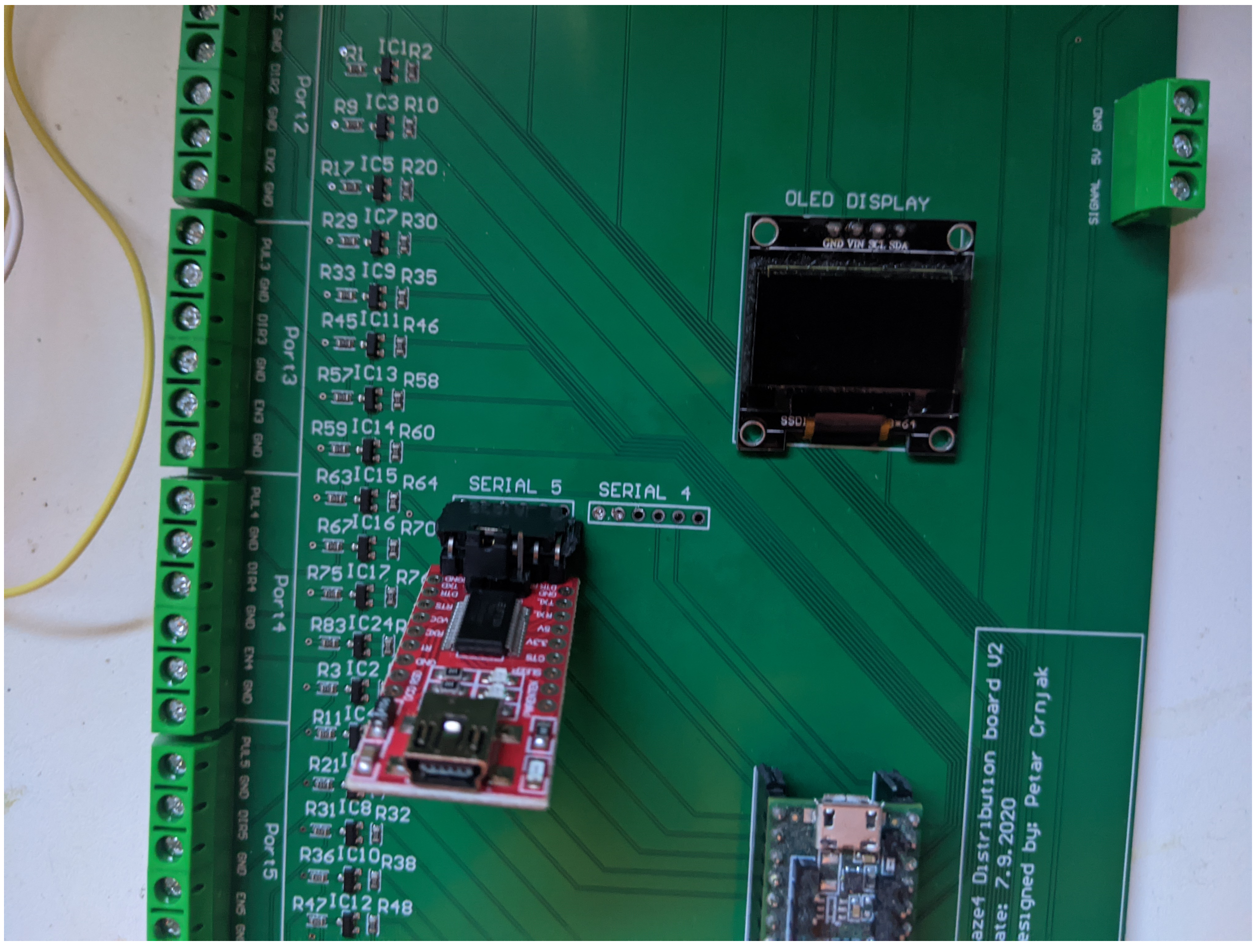
left side is for stepper outputs. There are also 2 stepper outputs on bottom right side.

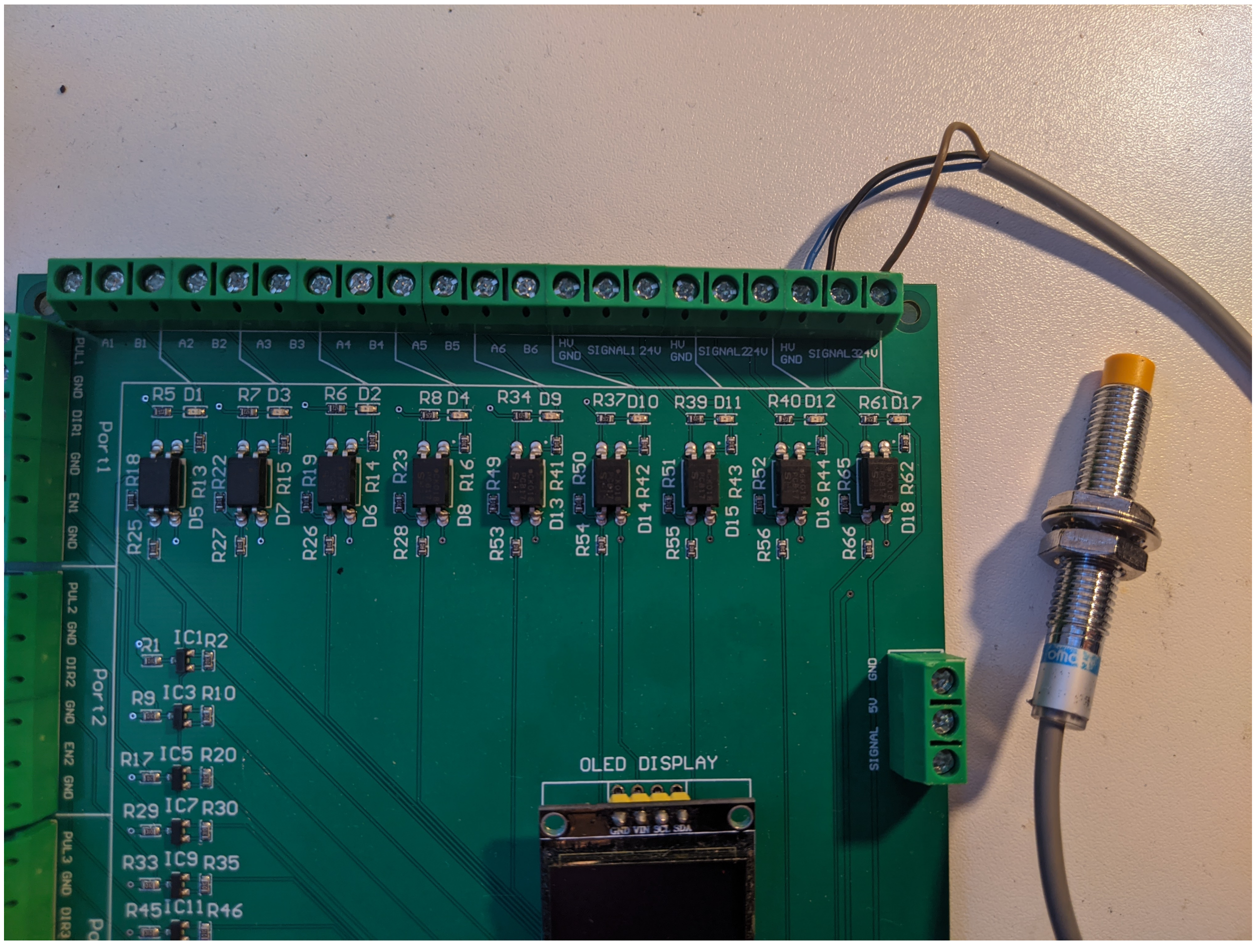
Bottom right side has 5V input and 24V input. You need both for arm to work. In the middle we have few extra serial ports and OLED display.

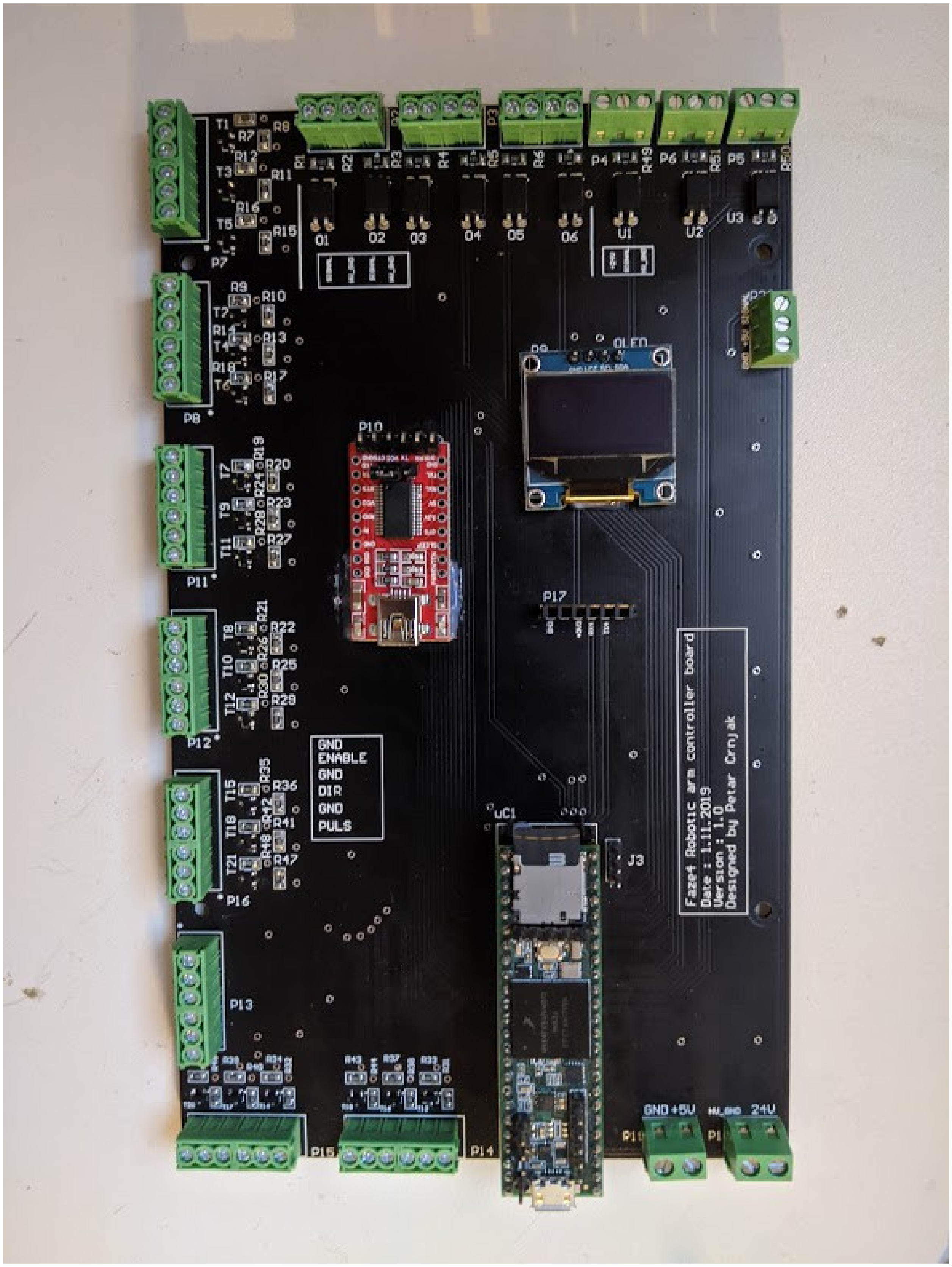
When wiring stepper drivers to the PCB i used TB6600 driver pinout as reference.I planned for all drivers to be connected like picture below. HIGH signal on ENABLE + pin would enable stepper drivers. Now i made some mistakes and not all drivers can be connected like that.

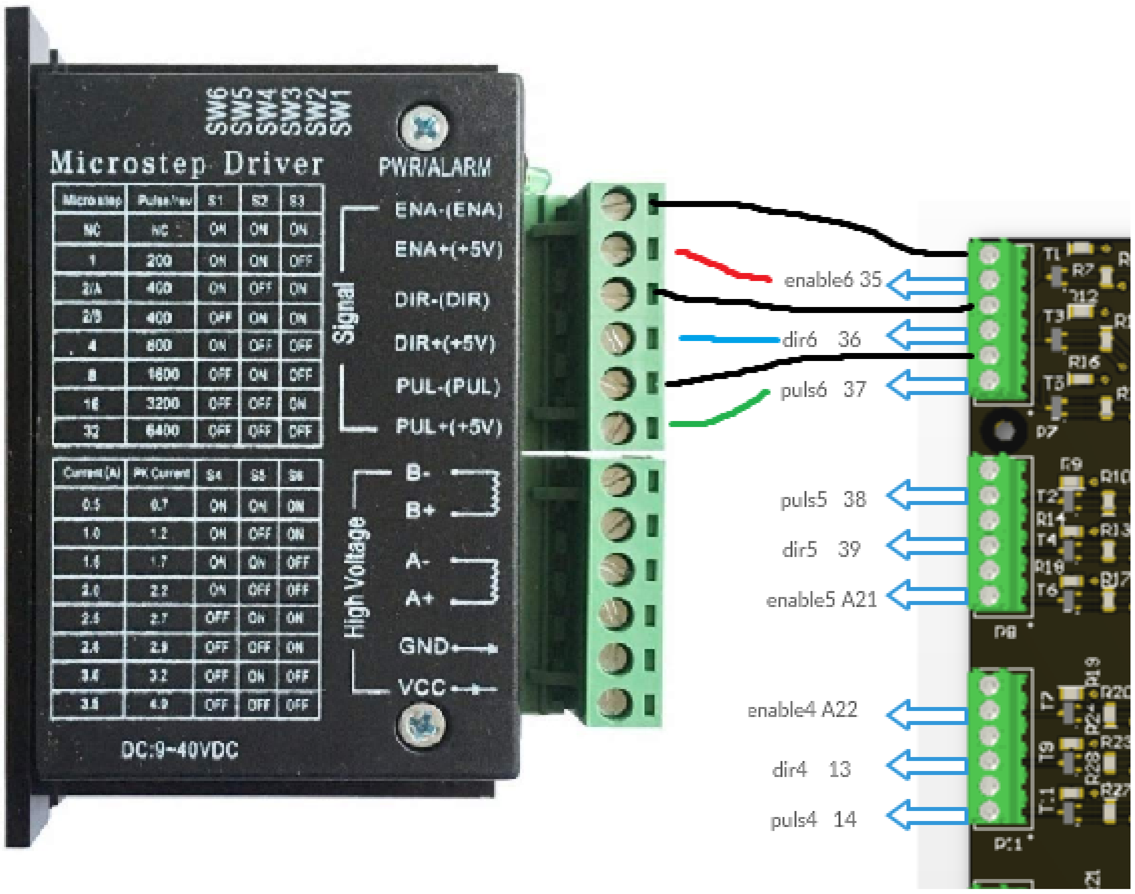

Fazet Distribution board V2
Date: 7.9.2020
Designed byr Petar Crnjak
OLED DISPLAY
SERIAL 4
SERIAL 5
Port1
Port2
Port3
Port4
Port5
Port6
Port7
Port8
Microstep Driver
PWR/ALARM
ENA-(ENA)
ENA+(+5V)
DIR-(DIR)
DIR+(+5V)
PUL-(PUL)
PUL+(+5V)
Signal
B-
B+
A-
A+
GND
VCC
High Voltage
DC9-40VDC


GND VIN SCL SDA
SSD1
8x64
VCC
GND
SCL
SDA


OLED DISPLAY
GND VIN SCL SDA
SERIAL 5
SERIAL 4
Port2
Port3
Port4
Port5
SIGNAL 5V GND
aze4 Distribution board V2
ate: 7.9.2020
esigned by: Petar Crnjak


OLED DISPLAY
SIGNAL 5V GND
Port1
Port2


GND
ENABLE
GND
DIR
GND
PULS
Faze4 Robotic arm controller board
Date : 1.11.2019
Version : 1.0
Designed by Petar Crnjak
OLED
P10
P17
uC1
J3
GND +5V
24V
P7
P8
P11
P12
P16
P13
P15
P14
O1
O2
O3
O4
O5
O6
U1
U2
U3
P4
P6
P5

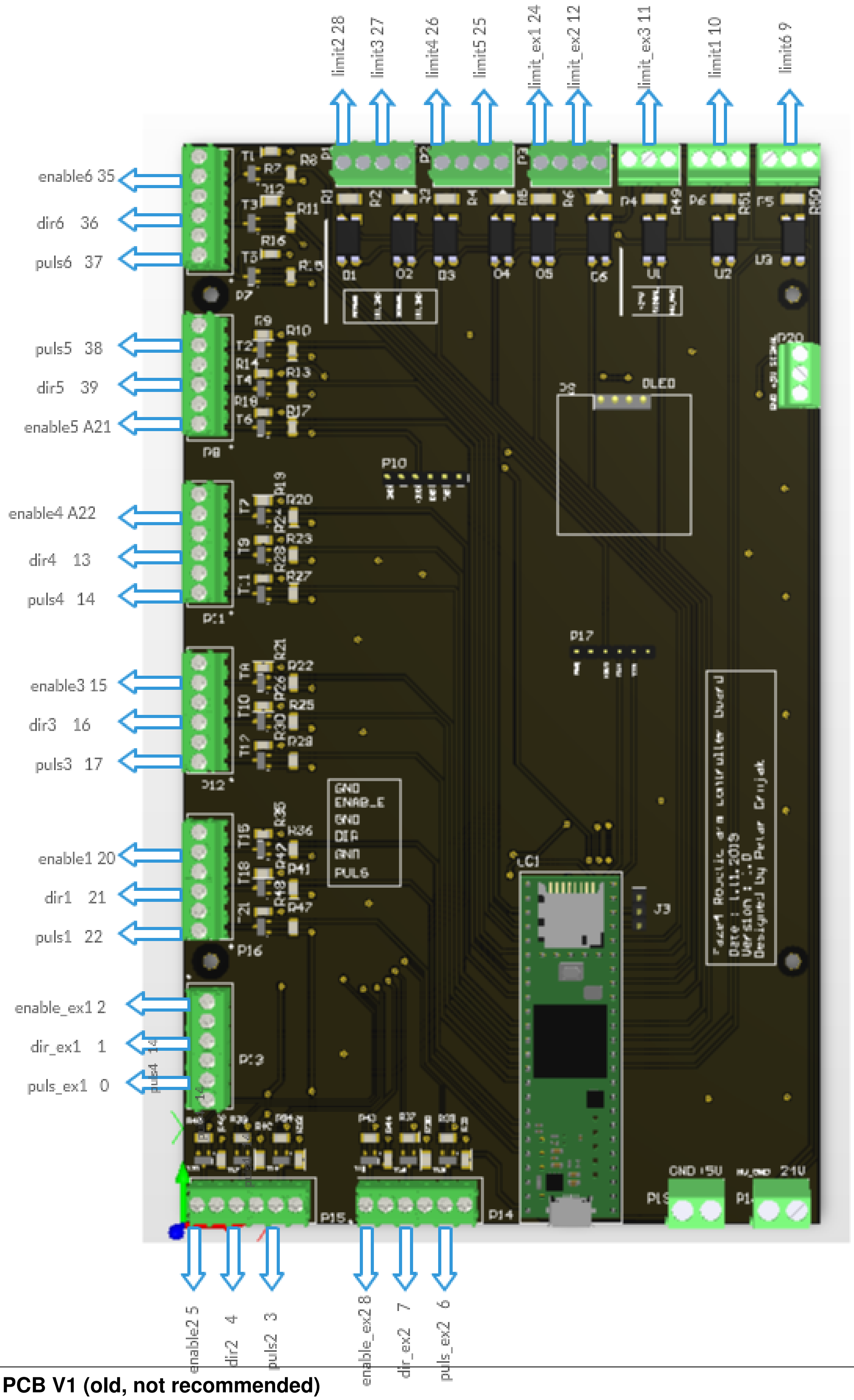


Micro step	Pulse/rev	S1	S2	S3
NC	NC	ON	ON	ON
1	200	ON	ON	OFF
2/A	400	ON	OFF	ON
2/B	400	OFF	ON	ON
4	800	ON	OFF	OFF
8	1600	OFF	ON	OFF
16	3200	OFF	OFF	ON
32	6400	OFF	OFF	OFF

Current (A)	PK Current	S4	S5	S6
0.5	0.7	ON	ON	ON
1.0	1.2	ON	OFF	ON
1.5	1.7	ON	ON	OFF
2.0	2.2	ON	OFF	OFF
2.5	2.7	OFF	ON	ON
2.8	2.9	OFF	OFF	ON
3.0	3.2	OFF	ON	OFF
3.5	4.0	OFF	OFF	OFF

Now i made some mistakes and not all drivers can be connected like that. In picture below you can see how to connect all drivers. As you can see for joint 5 (connector below top one) i switched enable and puls pins. That means that puls5 pin (pin 38 on teensy) is connected to puls pin on stepper driver. I also skipped Port that has no 5V level shifter since it didnt work well.

After wiring all dir,puls and enable pins you are left with power pins on stepper driver and A- A+ B- B+ Pins. Those are quite straightforward. Pins that have ext in name are extra pins that are now used in my version of arm.


limit2 28
limit3 27
limit4 26
limit5 25
limit_ex1 24
limit_ex2 12
limit_ex3 11
limit1 10
limit6 9
enable6 35
dir6 36
puls6 37
puls5 38
dir5 39
enable5 A21
enable4 A22
dir4 13
puls4 14
enable3 15
dir3 16
puls3 17
enable1 20
dir1 21
puls1 22
enable_ex1 2
dir_ex1 1
puls_ex1 0
enable2 5
dir2 4
puls2 3
enable_ex2 8
dir_ex2 7
puls_ex2 6
OLED
GND
ENABLE
GND
DIR
GND
PULS
GND +5V
24V

CHAPTER 6

Low level code

- Code is under MIT license: https://github.com/PCrnjak/Faze4-Robotic-arm/blob/master/LICENSE
- Matlab version used R2018a
- Robotic toolbox version: Robot-10.3.1
- Teensyduino version: Teensyduino, Version 1.53
- Teensy board used: 3.5
- Arduino IDE version: 1.8.13

Tip: You can scrap this code for parts and write much better code.

These codes are just proof of concept and must be handled with caution

Code for this robotic arm can be devided in 2 parts:

- Low level code
- High level code

Low level code is code that runs on our microcontroller (in our case teensy 3.5). Its main purpose is to receive information from high level code and transform that information into useful signals for stepper drivers. That is a gross simplification, but codes used will be explained in more detail below.

All code is in github repository.

6.1 Tennsy environment setup

Install arduino IDE: https://www.arduino.cc/en/main/software

Download Teensyduino from here: https://www.pjrc.com/teensy/td_download.html

Teensyduino is a software add-on for the Arduino software.

6.2 Teensy low level code for testing in Matlab(live script)

Tip: You can scrap this code for parts and write much better code.

File is located at: Faze4-Robotic-arm/Software1/Low_Level_Arduino/Arduino_GUI_code.ino

It works together with Matlab code located at: Faze4-Robotic-arm/Software1/High_Level_Matlab/GUI_Matlab.mlx

In first 100 lines are pin definitions. They are made for first version of faze4 distribution board that is now legacy and is replaced with second version that is also on sale here: * https://blestron.com/product/faze4-connector-boarad/

When power is applied to robot it will first perform homing routine. After all joints are homed it will go to starting/start by postion. After that it is ready to receive data from Code located in: Faze4-Robotic-arm/Software1/High_Level_Matlab/GUI_Matlab.mlx

Program is actually really simple. After homing we get data from matlab or any other source and then move robot according to that data. As you can see in code bellow we call only 2 functions in main loop.

```
void loop() {

while (error == 0) {

 get_data();
 move_all();
 }
 }
```

But before we get to main loop robot needs to home. Now if robot starts moving away from its limit switches during homing you need to stop it and do one of 2 things:

- Switch stepper motor phases
- change direction in software

After robot homes it goes to standy/start position.

In main loop move_all(); is based on functions like this:

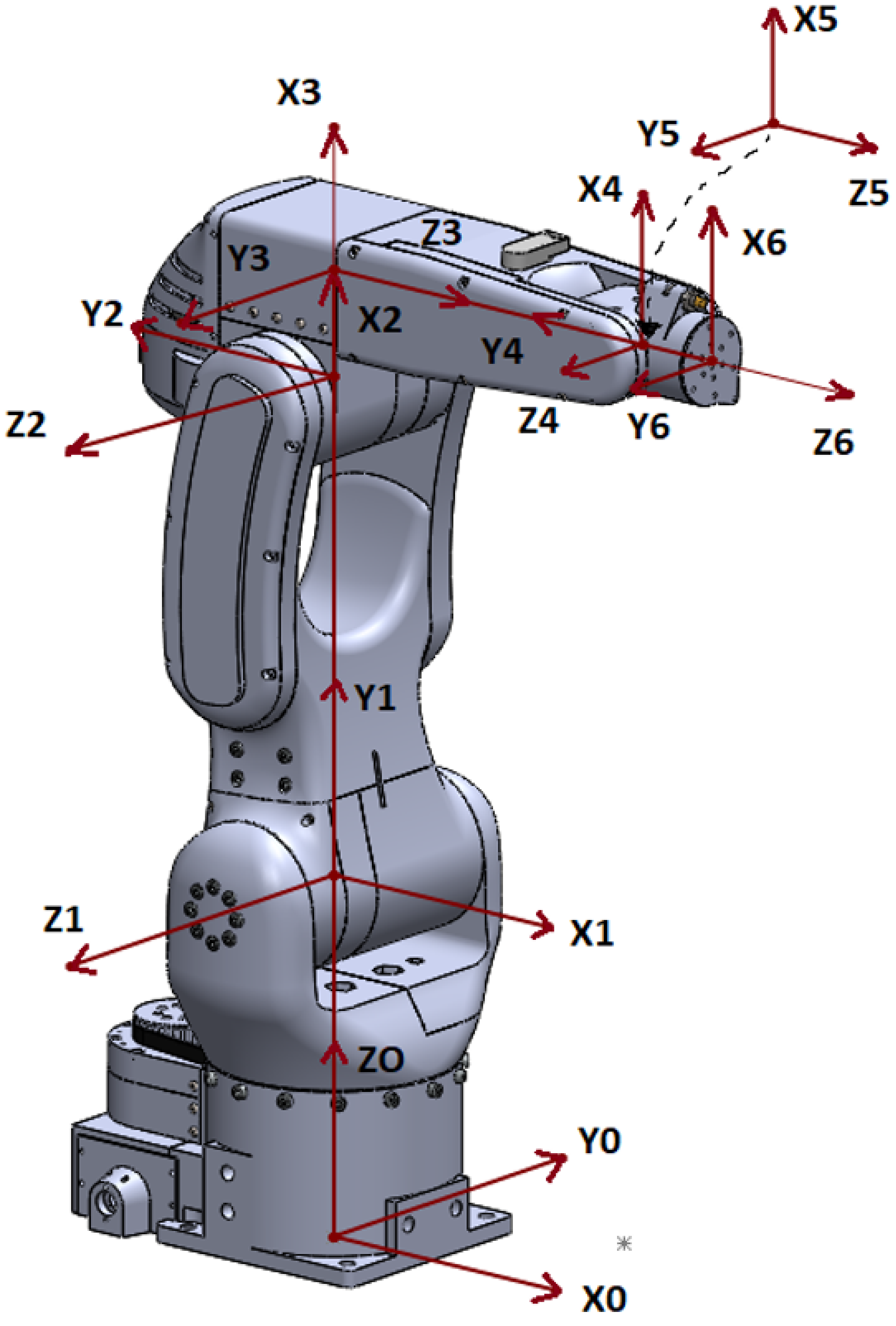
```
 if (needed_position[joint[1]] > current_position[joint[1]] ) {
 digitalWrite(dirPin[joint[1]], LOW);
 move_routine_forward(joint[1]);

 }
 else if (needed_position[joint[1]] < current_position[joint[1]] ) {
 digitalWrite(dirPin[joint[1]], HIGH);
 move_routine_backward(joint[1]);
 }
```

What this snippet does is this: Code is always looping thru this code in main loop. If needed_position is same as current_position nothing happens. If it is different we check if it is smaller or bigger and on those terms move_forward or move_backward. Then we perform same code on all joints of robot arm.

Move routine works like this:

```
void move_routine_forward(int joint_num) {
currentMillis = micros();
#state is used to prevent triggering of this if statement twice in row
#same goes for second one , this secures perfect square wave form
```

(continues on next page)


X5
X3
Y5
X4
Z5
Z3
X6
Y3
Y2
X2
Y4
Z2
Z4
Y6
Z6
Y1
Z1
X1
ZO
YO
XO

(continued from previous page)

```
if (currentMillis - previousMillis[joint_num] >= current_pulse_widht[joint_num] and␣
↪state[joint_num] == 0) {
 previousMillis[joint_num] = currentMillis;
 digitalWrite(stepPin[joint_num], HIGH);
 state[joint_num] = 1;
 }
   else if (currentMillis - previousMillis[joint_num] >= (current_pulse_widht[joint_
↪num]) and state[joint_num] == 1) {
 previousMillis[joint_num] = currentMillis;
 digitalWrite(stepPin[joint_num], LOW);
 state[joint_num] = 0;
 current_position[joint_num] = current_position[joint_num] + 1;
}
}
```

One step of stepper motor is defined by change from high to low signal on step pin. speed is defined by length of that signals period. In this code we use micros as timer function. Lets say we want half period of our pulse to be 500 us. Once we see 500us or more passed we switch step pin to HIGH and move state variable to low. We move that variable low as an indicator that next 500 us will swtich pin to LOW. Now when we switch to LOW state goes to 0 and we increment current position +1. This proces goes until move_all(); see that current_position = needed_position.

Now in normal operation robot can never hit limit switches if it hits them error variable in main loop goes to 1 and robot locks. This is done as a simple safety feature. But it can be disabled by just removing that while (error == 0) loop.

```
void loop() {

while (error == 0) {

 get_data();
 move_all();
 }
 }
```

6.3 Teensy low level code for Matlab trajectory planning

File can be found here: Faze4-Robotic-arm/Software1/Low_Level_Arduino/Robot_Arduino_trajectory/Robot_Arduino_trajectory.ino Code is almost the same only thing that is done differently is protocol for sending and receiving data is a bit more complex.

6.4 Teensy low level code for ROS

CHAPTER 7

High level code

- Code is under MIT license: https://github.com/PCrnjak/Faze4-Robotic-arm/blob/master/LICENSE
- Matlab version used R2018a
- Robotic toolbox version: Robot-10.3.1

These codes are best used to take parts from them to develop your own code or just to test the robot. They are clunky and you must be really careful when using them on your robot.

High level code does all calculations and simulations. It will usually run on the PC since it will be doing a lot of math and matrix multiplications. It will use some kind of inverse kinematics solver that will generate joint angles and speeds. Again, gross simplification, but we will talk more about it later.

All code is in github repository.

7.1 Setting up Matlab

I used Robot-10.3.1 library version from Peter Corke. You can find it here: https://petercorke.com/resources/downloads/ You need to install that libary to use codes used in this documentation.

7.2 Matlab code for testing in live script

File is located at: Faze4-Robotic-arm/Software1/High_Level_Matlab/GUI_Matlab.mlx

It works together with C/C++ Arduino file located at: Faze4-Robotic-arm/Software1/Low_Level_Arduino/Arduino_GUI_code.ino

When you run this live script it allows you to control robot using sliders and test all joints that way. It also shows simulated robot in Matlab window.Script will give back error if connection is not made with robot arm.

```
#Here you need to change your COM port to one your arm is connected to
delete(instrfind);
x=serial('COM12','BaudRate', 9600);
fopen(x)
```

Here we Create model of robot arm. If you are using gripper on robot change Gripping point to your value.

```
gripping_point = 0.057 ;
L(1) = Link([0 0.23682 0 pi/2]);
L(2) = Link([0 0 0.32 0]);
L(3) = Link([0 0 0.0735 pi/2 ]);
L(4) = Link([0 0.2507 0 -pi/2 ]);
L(5) = Link([0 0 0 pi/2]);
L(6) = Link([0 gripping_point 0 0]);
robot = SerialLink(L);
robot.name='faze4';
```

Now if you dont want to send data to robot you can make print in code bellow = 0. You would do this if you want to adjust to your robots zero position or move all joints at once.

```
if(print == 1)
fwrite(x,Start,'char');
fwrite(x,Start,'char');
for i = 1:6
```

7.3 Matlab trajectory planning

Files are located in: Faze4-Robotic-arm/Software1/High_Level_Matlab/Trajectory_Matlab/ C/C++ Arduino files are in: Faze4-Robotic-arm/Software1/Low_Level_Arduino/Robot_Arduino_trajectory/

As seen in this video: https://www.youtube.com/watch?v=0We33vvnHSE&t=6s&ab_channel=PetarCrnjak

Code is used to make a trajectory that arm will follow. Structure of the code is devided in logical sub files:

- Robot_angular_vel.mlx
- Robot_calculate_angles.mlx
- Robot_ik_code_1.mlx
- Robot_path.mlx
- Robot_plot_angles_velocity.mlx
- Robot_sending.m
- Robot_setup.mlx
- Robot_simulation.m
- Robot_trajectory.mlx

Robot_ik_code_1.mlx is main file. From there you can remove and add what stepts to do. For example if you dont want to send data to robot just comment out Robot_sending. If you dont want to simulate robot comment out Robot_simulation…

7.4 ROS

CHAPTER 8

Startup and setup

holder

CHAPTER 9

Robot data

9.1 General info

9.2 Dimensions

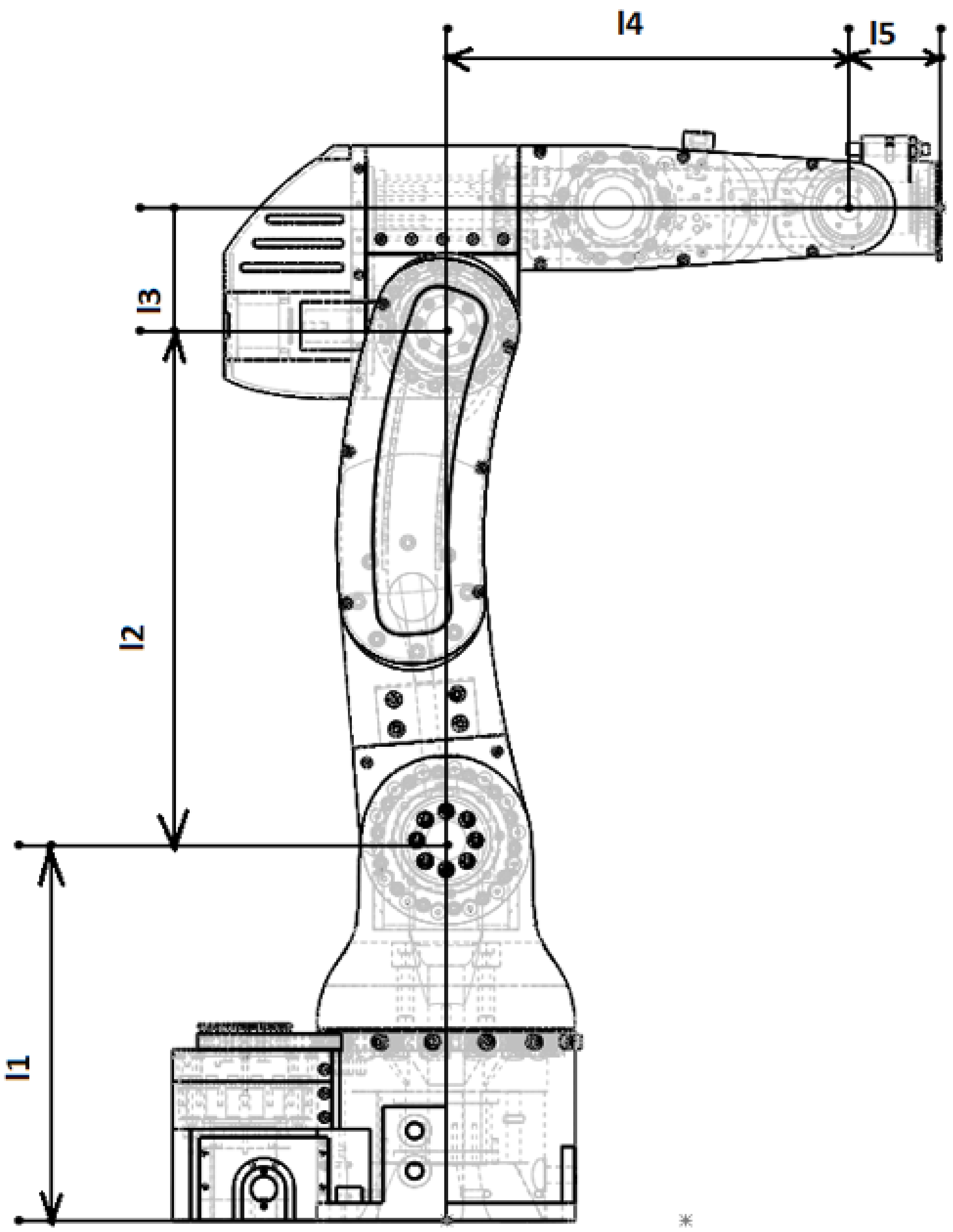
- L1 = 0.23682m
- L2 = 0.32m
- L3 = 0.0735m
- L4 = 0.2507m
- L5 = 0.057m

9.3 Reduction ratios:

- Joint 1 -> 15:1 cycloidal reducer and 5:3 belt reduction = combined resulting in 25:1 reduction
- Joint 2 -> 27:1 cycloidal reducer
- Joint 3 -> 15:1 cycloidal reducer
- Joint 4 -> 11:1 cycloidal reducer and 28:26 belt reduction = combined resulting in 11.846153:1 reduction
- Joint 5 -> 11:1 cycloidal reducer and 1:1 belt reduction = combined resulting in 11:1 reduction
- Joint 6 -> 19.19:1 planetary reducer

9.4 Belts

- Joint 1 belt is HTD 5M 490 mm timing belt 10 mm wide and needs to be tightened with bearings


l4
l5
l3
l2
l1

- Joint 4 belt is HTD 5M 430 mm timing belt 10 mm wide does not need to be tightened
- Joint 5 belt is HTD 5M 285 mm timing belt 10 mm wide and needs to be tightened with bearings

9.5 Stepper data

9.6 PSU