Self-Balancing Robot

Anders Latif, Boris Karavasilev and Stefan Bjerring Henriksen

12. August, 2022

1 Introduction (All)

A self-balancing robot is a robot which has an unstable equilibrium and works as an inverted pendulum, which has its center of mass placed above its pivot point. Such a robot must actively correct itself if it is not to fall over.

This report describes the design, assembly and testing of such a robot.

Both the robot and this report is a result of collaboration between all group members. The titles of the sections in this report contain names in parentheses, indicating the primary author of that section. In section 9 of the appendix we have listed which group member worked on which part of the robot.

2 Background (Stefan)

Most of the common designs of moving robots are passively balanced, meaning that their default position is inherently stable. This might be achieved by having tracks, supports or several sets of wheels/legs. Nonetheless, the use cases of mobile robots are many and there are certain advantages to having an actively balanced design. One such advantage is that the human body works as an inverted pendulum[2] and a self-balancing robot might be uniquely suited to traverse a human-centered world.

Our design will balance through a so-called proportional–integral–derivative controller (PID Controller), which is a method commonly used in automatic control of ships, aircraft and industrial robots as well as self-balancing robots.

Constructing a self-balancing robot will allow us to apply a number of techniques that we learned in the course such as 3D modelling, 3D printing and flat sheet cutting of the structure, mechanical design, powered movement as well as electronics and micro-controller programming.

3 Requirements (All)

We set the following requirements for our implementation of the self-balancing robot:

1) The robot must balance by actively correcting itself
2) The robot must balance on two powered wheels
3) The robot must use sensors to gauge its balance
4) The robot must be able to park
5) The robot must be able to recuperate from a resting position
6) The robot must have an unstable equilibrium (when not parked)

The following points are nice to have, but outside of the expected scope of the project:

7) The robot should be able to regain its balance after being nudged
8) The robot should be able to stand up from a prone position via arms or legs
9) It should be possible to change settings relevant for balancing in real time
10) The robot should be able to balance on different surfaces and inclinations

4 Description (Boris)

The body of the robot is comprised of four threaded rods onto which U-shaped shelf holders are fastened. A shelf can be easily attached with two key-like clamps to any shelf holder, this makes it easy to switch each shelf with a different one. An exception is the bottom-most shelf that needs to be attached more rigidly in order to provide enough support to the motors. A technical drawing of the basis shelf and shelf holder is included in appendix 12.5 and a drawing of the bottom shelf which holds the stepper motors is included in appendix 12.6.

Our robot has shelves for: a microcontroller, a battery, stepper and servo motors and the balance tuning mechanism with a display.

The legs of the robot prevent the robot from falling over during tuning and are also intended for pushing the robot into an upright position so that it can park itself or recover from a loss of balance. A render of the assembly along with a picture of the finished robot is included in Figure 1. An overview drawing of the full assembly is included in appendix section 12.1.

4.1 Mechanics

4.1.1 Materials (Stefan)

The basic structure of the robot is made up of Medium-density fibreboard (MDF) attached to four metallic threaded

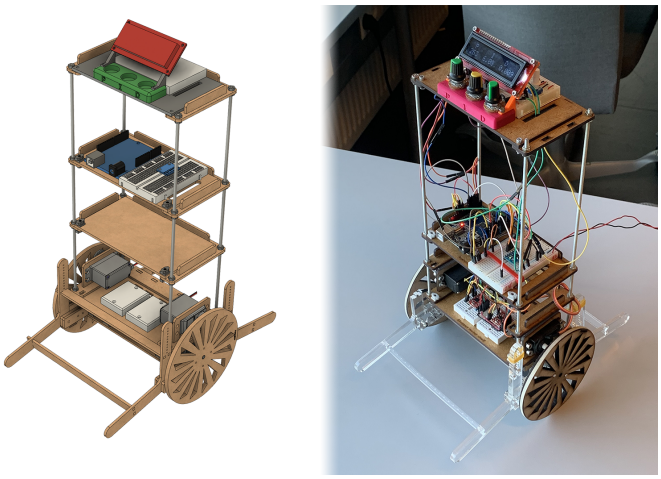


Fig. 1. Render of the robot assembly next to the finished robot.

rods. MDF is sufficiently durable and cheap which makes it a good material for prototyping. The legs for the parking mechanism requires more durability than MDF provides and is therefore made out of acrylic polymer sheet. The wheels are made of MDF with a rubber band along the edge for added friction.

The remaining parts are produced out of thermoplastic filaments via Fused Deposition Modeled 3D printing, such as the base for potentiometers, the LCD Screen holder, wheel hubs and shelf fasteners. Circuitry is made via breadboard and jumper wire.

All other parts used in our assembly have been borrowed from the ITU REAL lab and are included in the bill of materials (see appendix, section 10).

4.1.2 Shelf design (Anders)

One crucial design choice was to go with a modular design. We wanted to be able to pick apart the non-structural parts of the robot easily for faster prototyping. The modular philosophy would also aid any further development on the project and would ease adding more components like webcams, microphones and speakers.

We decided to use shelves wherever feasible. Shelves should easily slide in and out and be held by shelf holders that are fixed in place but adjustable in height. In our prototyping we tried snap-fit joints, especially cantilever joints [6]. What worked best is a lock mechanism using fasteners. The fastener enters a keyhole in both sides of the shelf and shelf holder and when turned it holds both tightly together. See appendix 12.9 for a drawing of the fastener.

A drawing of the 'naked robot' with all shelves removed is included in appendix section 12.2.

4.1.3 Legs (Boris)

The legs of the robot prevent the robot from falling over and damaging itself and they were also intended for pushing the robot into an upright position in which it can park itself or recover from a loss of balance. Appendix 12.3 contains a technical drawing of the legs.

The L-shaped legs are actuated by two servo motors. A leg together with the servo arm connected by a wire form a 4 bar linkage with 4 revolute joints, three bars and one imaginary bar (the body of the robot).

Pivoting of the legs is enabled by a hinge implemented by a screw that acts as a plain bearing and has low friction with the acrylic leg. The movement of the legs is depicted in Figure 2.

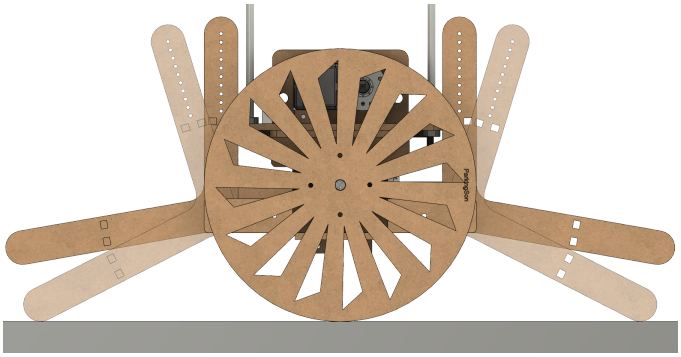


Fig. 2. Legs in both end positions.

The legs add a second degree of freedom to our robot besides the rotation of the wheels.

4.2 Electronics

4.2.1 Input (Stefan)

The robot relies on several types of sensors. The GY-521 MPU provides a 3-axis gyroscope and a 3-axis accelerometer[3], both of which serve the robot with detailed real-time updates on its location and movement which can then be used for correcting and balancing. The shelf which contains both the Arduino and the sensors need holes for wires to pass through. See appendix 12.7 for a drawing.

The robot is also fitted with potentiometers. Their main use is in the PID-tweaker which allows the user to increase or decrease the internal values of the PID controller by turning the three potentiometers. A fourth potentiometer controls the contrast of the LCD screen. A drawing of the potentiometer holder and LCD screen holder is included in appendix 12.8.

4.2.2 Output (Boris, Stefan)

Since precision was crucial we configured our stepper motor drivers (A4988)[4] to use an "eighth step" micro-stepping. This was achieved by connecting the MS1 and MS2 pins to 5V which divided each step into 8 micro-steps and significantly increased the resolution. The trade off was reduced torque and maximum speed which was not relevant to our project because we choose the Nema 17[5] stepper motors that provided sufficient torque and rotation speed to balance our robot.

The PID Tweaker contains a Sparkfun Basic 16x2 Character Black on White LCD screen[7] for displaying the current PID parameters. When the PID parameter values are adjusted by turning the potentiometers the LCD screen is updated to reflect this.

4.3 Circuits

A table showing arduino pin usage along with schematics for the circuitry of stepper motors and stepper motor

drivers, the GY-512, the PID Tweaker and LCD screen and the servo motors is found in appendix 13.

4.3.1 Power (Anders)

Different components of the robot have different power needs, as shown in the table below.

Component	Voltage
Gyroscope & accelerometer	3.3 V
Arduno Stepper motor drivers Potentiometers (PID Tweaker) LCD Screen	5 V
Stepper motors	12 V

For prototyping, we use a power supply which provide ∼12Vs for the stepper motors and the Arduino. The Arduino distributes power to the subsystems. Our aim is to have a self-balancing robot that is self-sufficient in its power needs. We currently possess a 5V 15Ah lithium battery [1]. This means that we would need several lithium batteries to provide enough power.

The 5V pin of the Arduino does not provide sufficient current for running all subsystems when the 2 servos are also connected. When prototyping and testing the servos we had to add a second channel from the power supply that delivers 5V.

5 Programming

There are many facets to our microcontroller code, but the most relevant part of it is the PID algorithm, which is described in detail in section 5.1.2. The full code, `main.ino`, is listed in appendix 14.2.

5.1 PID Control (Anders)

PID Control solves the common problem in microcontroller programming of constantly sensing input and outputting adjustments accordingly. In this way we achieve a feedback loop that is useful for our self-balancing.

5.1.1 PID Control - Sensors

We used the GY-521 with an embedded MPU-6050 that provide an accelerometer, gyroscope and temperature measurements. It does not boast of high precision which is something we had to take into account when reading the data.

We also have a calibration process that assumes that the robot starts out forward parked and then subtracts the target angle which provides the robot with its desired angle.

We can not solely rely on the accelerometer because the horizontal acceleration of the robot would affect the readings. Using a gyroscope would cause problems with drift which happens over time.

A more precise measurement of the tilt is therefore a combination of both with a filtering algorithm. We decided on the simpler ”complimentary filter” rather than the ”Kalman filter” because of hardware restrictions and the clock cycles (20MHz) of the Arduino. The complimentary filter applies a low-pass filter to the angle (accelerometer) and a high-pass filter to the angular velocity (gyroscope). This produces a more balanced reading.

5.1.2 PID Control - Algorithm

Our PID (**P**ropotional, **I**ntegral, **D**erivative) control algorithm outputs a negative or positive value that represents how much the robot should drive forward or backward to balance. Each of these terms are multiplied by a constant (denoted by K) factor. To arrive at these values which are unique to our design we had to manually tweak the values. We have included a diagram of the formula in the appendix section 8.

Purely proportional control is not enough to balance the robot because it does not correct the robot enough when it is close to the equilibrium or results in rapid oscillations if it is made too sensitive.

The integral term accumulates errors over time which makes even small errors add up to significant values over time and helps correct the robot’s tilt around the equilibrium.

To prevent overshoots the derivative term is introduced. It takes the difference between the previous angle and the current one and subtracts it from the PID formula. In addition, good derivative values allows the robot to more readily recover from outside disturbances, such as being pushed.

An example of a recorded error over time is seen in figure 3. We see the values oscillating around 0.0 which is the target angle (upright position). In the beginning the proportional term overshoots wildly around 0.0 but the integral term helps correct it.

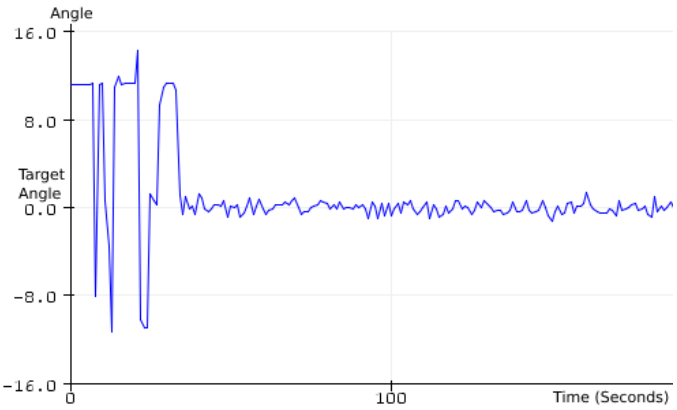


Fig. 3. Input error to the PID algorithm.

6 Challenges

6.1 Modular Design (Boris)

We made our robot modular because we had the philosophy: ”If it is easy to modify or add a new part to the robot it will encourage us to innovate”. Our final design choice was to have fixed shelf holders that are adjustable in height and shelves that could slide in and be clamped down to the holders.

One downside of this configuration was the necessity to level each of the shelves individually to be parallel to the ground (especially the shelf with the sensor measuring the tilt of the robot). The current configuration also restricts the height of the robot to the length of the threaded rods which is not important for our project but might be important

to consider if this robot was to be extended to hold more modules.

6.2 DC vs. Stepper Motors (Stefan)

We considered two kinds of motors for the robot and while both of them have advantages and are valid candidates we settled on stepper motors. The stepper motors have higher precision and better torque at low speeds which might be useful when making small corrections.

6.3 Arms vs. Legs (Boris)

One of the requirements that we defined for our robot is to be able to park and recover from the loss of balance. To fulfill this requirement we considered two types of mechanisms.

The first type of mechanism is similar to a human arm and would allow the robot to push itself up from a lying position on the floor. Although, this would require a very strong motor that we did not have access to.

The second type of mechanism is similar to a leg. We decided to design and manufacture the leg resembling mechanism because it would allow us to use a hinge and leverage close to the ground that would allow us to push the robot from a tilted position back into a balanced position. These legs have also a very important function as they serve as training wheels for the robot and prevent the robot from falling all the way to the ground and damaging itself. The details of the design are explained in the section 4.1.3.

6.4 Materials (Boris)

The choice of materials played a critical role in manufacturing many of the robot's parts, for instance, the wheels. Initially, we laser cut them out of a 6 mm MDF board and press-fitted them onto the shaft of the stepper motors. The wheels became loose during repeated detachments and re-attachments. Therefore, we designed 3D printed hubs that were attached to the wheels with screws. Another screw through the hub presses onto the flat section of the motor's shaft to secure it. Due to the anisotropic properties of the 3D printed hub, we had to redesign the hub to make it strong enough. The wheel assembly is depicted in the drawing in appendix 12.4

Another part that we improved by choosing a different material for manufacturing was the legs. At first, we laser cut them out of 3 mm MDF board but they broke during the tuning of the robot. We made the design of the legs more rigid and also laser cut them out of a 6 mm acrylic board that was much sturdier than the MDF version.

6.5 3D printing and laser cutting the same design (Stefan

Because we wanted to rapidly prototype the robot and try different approaches, we wanted our CAD designs to be able to be laser-cut as well as 3D printed wherever relevant. But this introduced a challenge with the dimensions of the designs. 3D printing might produce parts that are slightly larger than the design and laser cutting removes some small amount of material by kerfing, resulting in a smaller final product.

We overcame this challenge by introducing a global parameter called bias in a master sketch in our CAD program. For all relevant dimensions such as screw holes and finger joints we made sure our sketches would inherit this bias parameter which we could change in one place before producing the part. Because of this global parameter, increasing or decreasing important measurements became very simple.

6.6 Malfunctioning components (Boris)

We were aware that some components from the lab might be defective. In order to reduce the time spent working with malfunctioning components, we tested parts before adding them to the robot. This consideration paid off when we needed a pair of stepper motor drivers, where we managed to discard some that did not work.

Several of the servo motors we tried exhibited unexpected behavior. They were not moving at all, would not move precisely enough to the specified position, or would continue to spin far beyond $180°$. Whether all the servos were malfunctioning or we introduced errors ourselves via circuitry or code is unknown. Since we were pressed for time, we left the completion of the parking mechanism for future work.

6.7 ESP Microcontroller (Anders)

In the beginning we spent a lot of time with the XC4411 board in order to be able to send move commands over Wi-Fi. The XC4411 is a dual board that has both an Arduino and an ESP8266. We were able to run a server on the Arduino and send commands to it over the browser when connected to the hotspot. But during the development it became apparent how constantly reading sensor data and updating the motor power is crucial which Serial reading would interfere with. This side-project was thus abandoned.

6.8 PID implementation (Anders)

For a long time our bot struggled to self-balance because the output of the PID algorithm was insufficient to accelerate fast enough to self-correct. We discovered that we had to multiply the integral output (error sum) with a factor that we had to test to approximate. Through research we found out that this is a widely known problem called the integral windup[8] which is when the factor makes the error sum grow too large over time. We solved this by constraining the values.

6.9 PID Tuning (Boris)

Finding the correct PID values manually is a lengthy process and requires many iterative changes to the constants and tests of the robot's behavior. We accelerated this process by building a shelf with a display and potentiometers on our robot to make the tweaking of the constants very easy and avoid re-uploading new code to the microcontroller in every iteration.

7 Results and Analysis (All)

A video of our robot in action can be seen through the following link: https://youtu.be/ohNkID8hXRA

7.1 Test framework (Stefan)

To determine how well our robot fulfils the requirements laid out in section 3 we subject it to several tests.

To prepare for testing the target angle was calculated from a forward resting position and we manually tweaked the three PID values until we were satisfied with how well the robot could self-balance. We let the robot go and timed how long it managed to keep its balance. We denote this metric as *Time to fall* and if the robot balances for more than 60 seconds we determine that it balances and stop the time.

Unless otherwise specified, all tests are done on a flat floor by turning on the robot, letting it zero the accelerometer and gyroscope values in its resting position and then holding it steady in an upright position, letting it go and measuring Time to fall. If the robot immediately fall over, we deem that it was not balanced and retry.

The baseline PID values we found via manual tweaking were:

$$P = 240, I = 6, D = 0.01$$

To test the baseline, we measured Time to fall ten times. Test two had Time to fall of 18 seconds, all other managed 60+.

7.1.1 PID ranges test

In order to learn how much impact varying the PID values have on the robots ability to self-balance, we measured Time to fall relating to changes in PID values from baseline.

Test	P	I	D	Time To Fall
Baseline	240	6	0.01	60+ s
off	(no active balance)			1.1 s
Low P	200	6	0.01	6 s
High P	280	6	0.01	60+ s
Low I	240	5	0.01	30 s
High I	240	9	0.01	60+ s
Very high I	240	14	0.01	60+ s
Low D	240	6	0.006	7 s
High D	240	6	0.02	60+ s

7.1.2 Push test

We tested the robots ability to withstand being pushed by getting it to balance and the giving it either a light nudge, a medium push or a hard shove and noting if it managed to regain its balance. Each test was repeated five times.

Test	1	2	3	4	5
Nudge	✗	✓	✓	✓	✗
Push	✗	✗	✗	✗	✗
Shove	Did not test				

7.1.3 Surface friction test

To test how well to robot could balance on a different surface, we placed it on a somewhat firm couch with a fabric cover and attempted to balance it.

With baseline PID values the robot had a Time to fall of:

$$1.5s, 1.2s, 1.1s$$

7.1.4 Tilt test

In order to test how well the robot balanced on an un-even surface we placed a large stiff wooden board in a 1° angle and attempted to balance the robot on it. We tested three times with the robot facing both forward and backwards.

With baseline PID values the robot had a time to fall of:

Test	1	2	3
Forward	1.5 s	1.5 s	1.3 s
Backward	1.4 s	1.2 s	1.2 s

7.1.5 Weight distribution test

Our robot has its center of gravity located toward the bottom of the robot. To see how changing this would affect the balance we added weights and tried to balance the robot.

We made the robot top-heavy by placing a 680 gram object just below the top shelf. The top heavy robot had a Time to fall of:

$$3s, 3.2s, 7.8s$$

We made the robot slightly more bottom-heavy by placing a 320 gram object between the stepper motors. The robot now had a Time to fall of:

$$4.1s, 3.5s, 3.8s$$

Note: Adding these weights might have affected the sideways balance of the robot, and we should have recalibrated the gyroscope and accelerometer before testing Time to fall.

8 Discussion (All)

Our robot manages to self-balance under standard conditions, but several factors make it struggle. It does not handle changes in inclination or other types of surfaces. It is also sensitive to outside forces, such as being pushed or driving into obstacles.

There are many steps we could take to improve our design. We could improve the systematic testing of our PID ranges in section 7.1.1 and find more optimal values, which would stabilize the robot and make it able to withstand larger disturbances and pushes.

We could also tweak the robots center of gravity during testing more than we did. Our tests only showed that changing the center of gravity affected the stability to a point where we could not get the robot to balance. But testing more and recalibrating the robot after adding weights might've allowed us to make a more stable design.

The requirements that the robot should be able to park and get up from a prone position (optionally using arms or legs to do so) were not met because we decided to spend our time elsewhere. The linkage between the servo and legs has not been designed and the servo positions have not been calibrated though we already prepared the code for it. But we are confident that with more work, the parking mechanism would be successful. Being able to go from a parked state to a balanced state without human support is a more delicate task, which would require more testing and tweaking.

We also never implemented the battery properly. This was not a requirement, but just something we wanted to do.

Due to safety concerns and the power needed to power the stepper motors as well as dealing with discharge we tested the robot with the power supply attached which limited the operational range.

This robot is only a prototype and there are several of the design decisions we would like to change. We would like to implement a printed circuit board (PCB), perhaps as an Arduino shield. This is not appropriate while in the prototyping stage but once a final design is in place it will be much smaller than the breadboards and the jumper wires will not detach during transport and testing.

We would also add new features, such a control (maybe wireless steering), obstacle detection and maybe even an internal map and navigation.

If we were to mass produce this robot[1], there are several changes we would make. Mainly, we would do away with the thread rods and shelves. Instead, we could produce the main body in a single piece by plastic injection molding, thus requiring less manual assembly. We would use a PCB instead of a breadboard and choose cost-effective versions of the components such as cheaper motors and microcontroller.

This course gave our group the chance to learn a lot about prototyping and design. We feel that the hands-on approach and the project taught us a lot. Namely that most facets of prototyping take longer than we expected and that many of the seemingly straight forward tasks may introduce unseen challenges and problems. But with the right tools and knowledge one can overcome these and produce novel solutions to problems as they occur. An open mind coupled with the ability to step back and reexamine a design are very valuable qualities when iterating over a design.

8.1 Conclusion (All)

The final design of the robot is able to balance by actively correcting itself on two powered wheels. While not able to park or recuperate from a resting position, the final version of the robot lives up to most of the requirements laid out in the beginning of this report.

Furthermore, the robot lives up to the course requirements of having a mechanism (motors and legs), electronics (motor drivers, PID tweaker) and microcontroller programming (PID controller, motor control).

Lastly, the group is satisfied with our work and the final product and is proud that we managed to get the robot to balance.

1. There might be a market for selling this robot as a toy

REFERENCES

[1] ANSMANN. *ANSMANN Powerbank 15.8*. URL: https://www.ansmann.de/uploads/Import/docs/manuals/1700-0096_Powerbank-15.8-Type-C_Manual.pdf (visited on 08/05/2022).

[2] Nadav; Zhao Sherry Caulley Desmond; Nehoran. *ECE 4760: Final Project, Self-Balancing Robot*. URL: https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2015/dc686_nn233_hz263/final_project_webpage_v2/dc686_nn233_hz263/index.html (visited on 07/29/2022).

[3] Electrosome. *Interfacing MPU-6050 / GY-521 board with Arduino Uno*. URL: https://electrosome.com/interfacing-mpu-6050-gy-521-arduino-uno/ (visited on 08/08/2022).

[4] Last Minute Engineers. *A4988 Driver Module Arduino*. URL: https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/ (visited on 08/08/2022).

[5] Oyostepper. *Nema17 Stepper Motor Datasheet*. URL: https://www.oyostepper.com/images/upload/File/17HS13-0404S.pdf (visited on 08/08/2022).

[6] Bayer Material Science. *Snap-fit joints for plastic*. URL: https://fab.cba.mit.edu/classes/S62.12/people/vernelle.noel/Plastic_Snap_fit_design.pdf (visited on 08/05/2022).

[7] Sparkfun. *Basic Character LCD Hookup Guide*. URL: https://learn.sparkfun.com/tutorials/basic-character-lcd-hookup-guide (visited on 08/08/2022).

[8] Wikipedia. *Integral Windup*. URL: https://en.wikipedia.org/wiki/Integral_windup (visited on 08/05/2022).

9 DISTRIBUTION OF WORK

9.1 Mechanics

All >General design, CAD
Anders >Wheels, wheel holders v1, lock mechanism, 3D printing, laser cutting and assembly.
Boris >Design and 3D modeling of the legs and the stepper motor shelf, laser cutting and assembly.
Stefan >Shelf holder design, wheel holders v2, PID Tweaker (design, laser cut, 3D print), legs (laser cut and assembly).

9.2 Electronics

Anders >GY-521, XC4411[2]
Boris >Designing the circuit connecting the stepper motors to the drivers in Fritzing + building the physical circuit.
Stefan >PID Tweaker and LCD circuitry.

9.3 Programming

Anders >PID, ESP8266[2], Servos
Boris >Control of the stepper motors with and without a library + improvement of the PID implementation.
Stefan >PID Tweaker/LCD Screen.

9.4 Tests

All >Tests have been peer reviewed by all members of the group.

10 BILL OF MATERIALS

Component	Quantity	Cost
Stepper Motor: Nema 17HS13-0404s https://www.oyostepper.com/goods-34-Nema-17-Stepper-Motor-Bipolar-18-deg-26Ncm-368ozin-04A-12V-42x42x34mm-4-Wires.html	2	86.60 DKK*
Stepper Motor Drivers: A4988 https://arduinotech.dk/shop/geeetech-stepstick-a4988-stepper-motor-driver/	2	58 DKK
Servo Motor: TowerPro SG-5010 https://www.adafruit.com/product/155	2	89.43 DKK*
Arduino Uno https://arduinotech.dk/shop/kuongshun-uno-r3/	1	84.00 DKK
Breadboard https://arduinotech.dk/shop/mellem-breadboard/	2	36.00 DKK
Mini Breadboards https://arduinotech.dk/shop/mini-prototype-breadboard/	2	16.00 DKK
Potentiometers https://arduinotech.dk/shop/potentiometer/	3	15.00 DKK
Potentiometer Knobs https://arduinotech.dk/shop/potentiometer-knob/	3	12.00 DKK
Plexiglass (Acrylic plate), 5 mm https://www.silvan.dk/plexiglas-akrylplade-transparent-5-x-500-x-750-mm-5-mm?id=9240-5035141	0.26	72.79 DKK
MDF Wooden Board, 6 mm https://www.bauhaus.dk/dlh-6-mm-mdf-plade-2440-x-1220-mm	0.019	11.58 DKK
Steel rods, 4x https://dk.rs-online.com/web/p/metalstaenger-runde-og-firkantede/6614674	1**	22.58 DKK
Miscellaneous additional costs (roughly estimated) Includes screws, nuts, wires, zip ties, PLA filament, rubber bands.		30 DKK
Total		517.98 DKK

* At the exchange rate from Danske Bank on the 8th of August. https://www.danskebank.dk/Documents/currency/currency-rates3.html?c_par=gsProdukt=INF&gsNextObj=Valuta&gsNextAkt=VFList0S&gsSprog=DA&gsBrand=DB&FixingListegsCurItem=Init&FixingListegsCurItem2=DADKK

2. Though not evident in the final product quite some time was spent on using the the dual Arduino and ESP board, XC4411, to provide a Wi-Fi hotspot. We were able to receive Serial commands on the Arduino coming from the ESP server through a HTML page served through the browser. Reading Serial data on the Arduino proved detrimental to the execution of the self-balancing code however and this side-project was not pursued further.

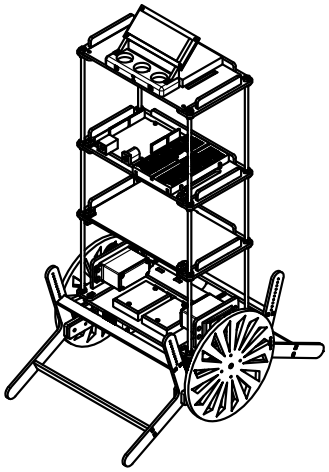
11 THIRD PARTY COMPONENTS

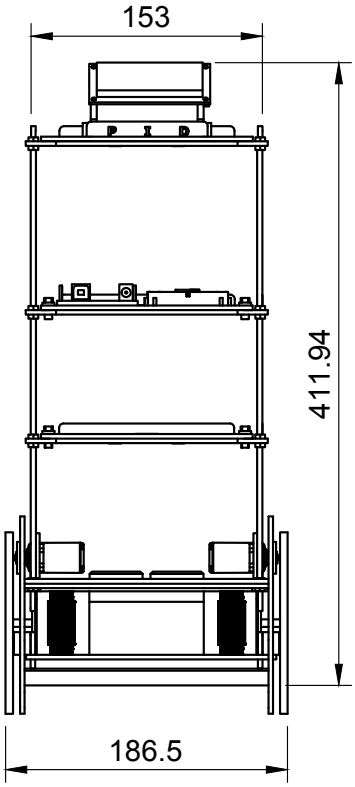
Some CAD designs were designed by third parties, which is listed in the following table. Any component not listed is designed by us.

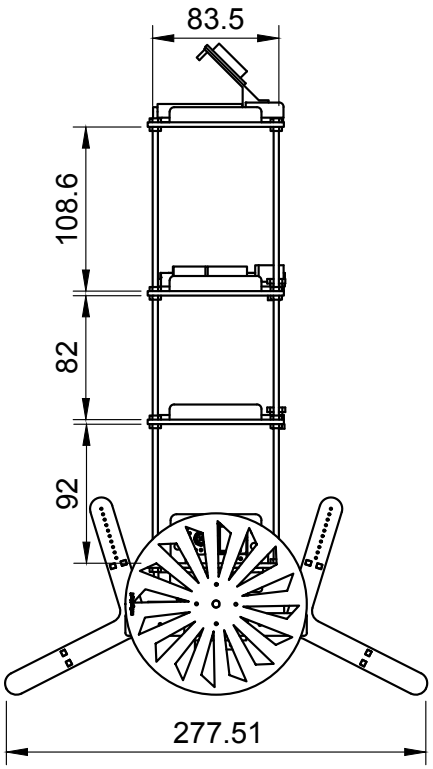
Component	Thanks
Stepper Motor	Sammy Johansson https://gallery.autodesk.com/fusion360/projects/stepper-motor
Servo Motor	Parallax https://www.parallax.com/package/parallax-servo-cad-file/
GY-512	Akshaya Simha https://grabcad.com/library/gyroscopic-sensor-1
Arduino	Anders & Boris, Mandatory Assignment 1
Breadboard	Anders & Boris, Mandatory Assignment 1

12 DRAWINGS

12.1 Full Assembly

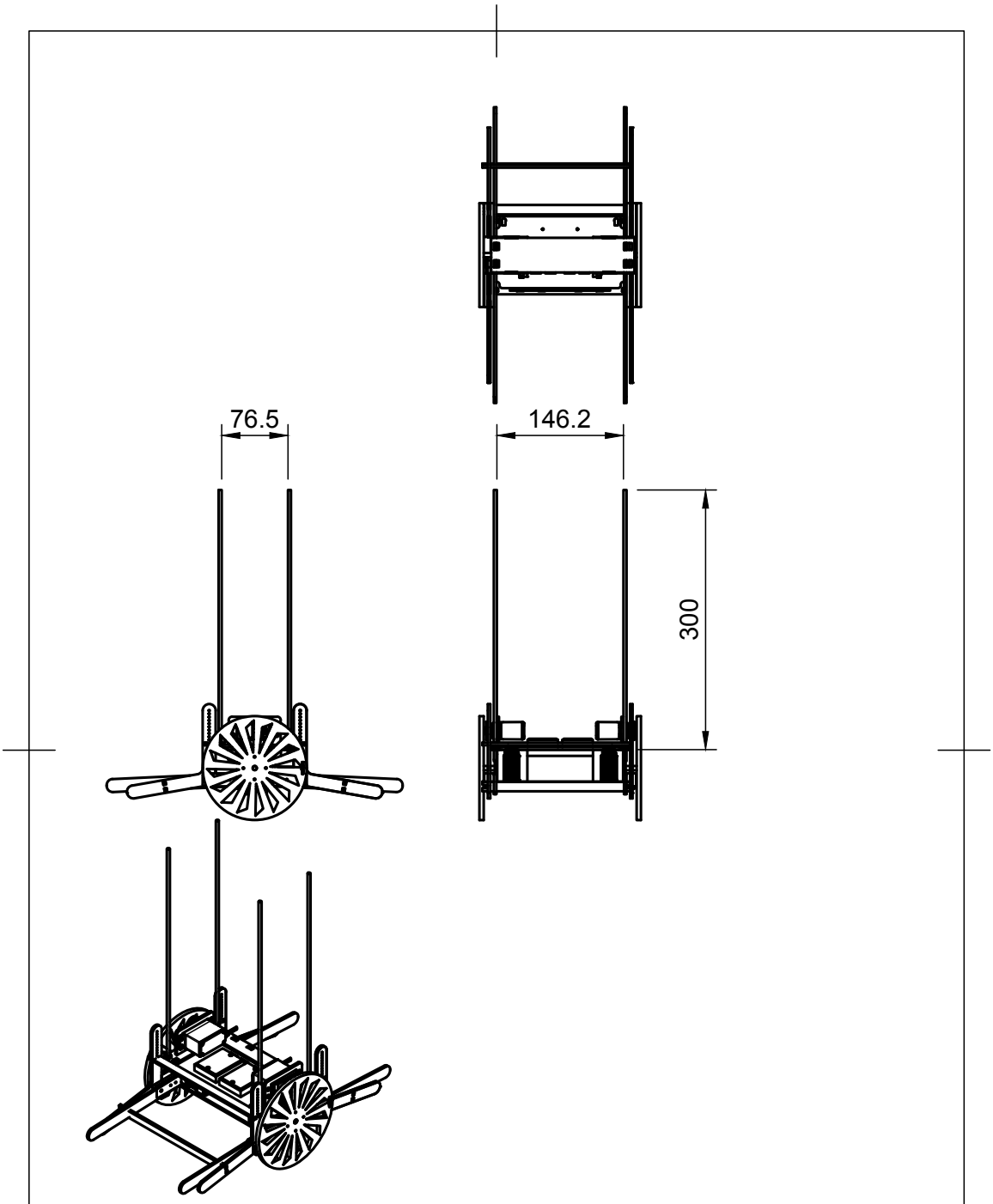






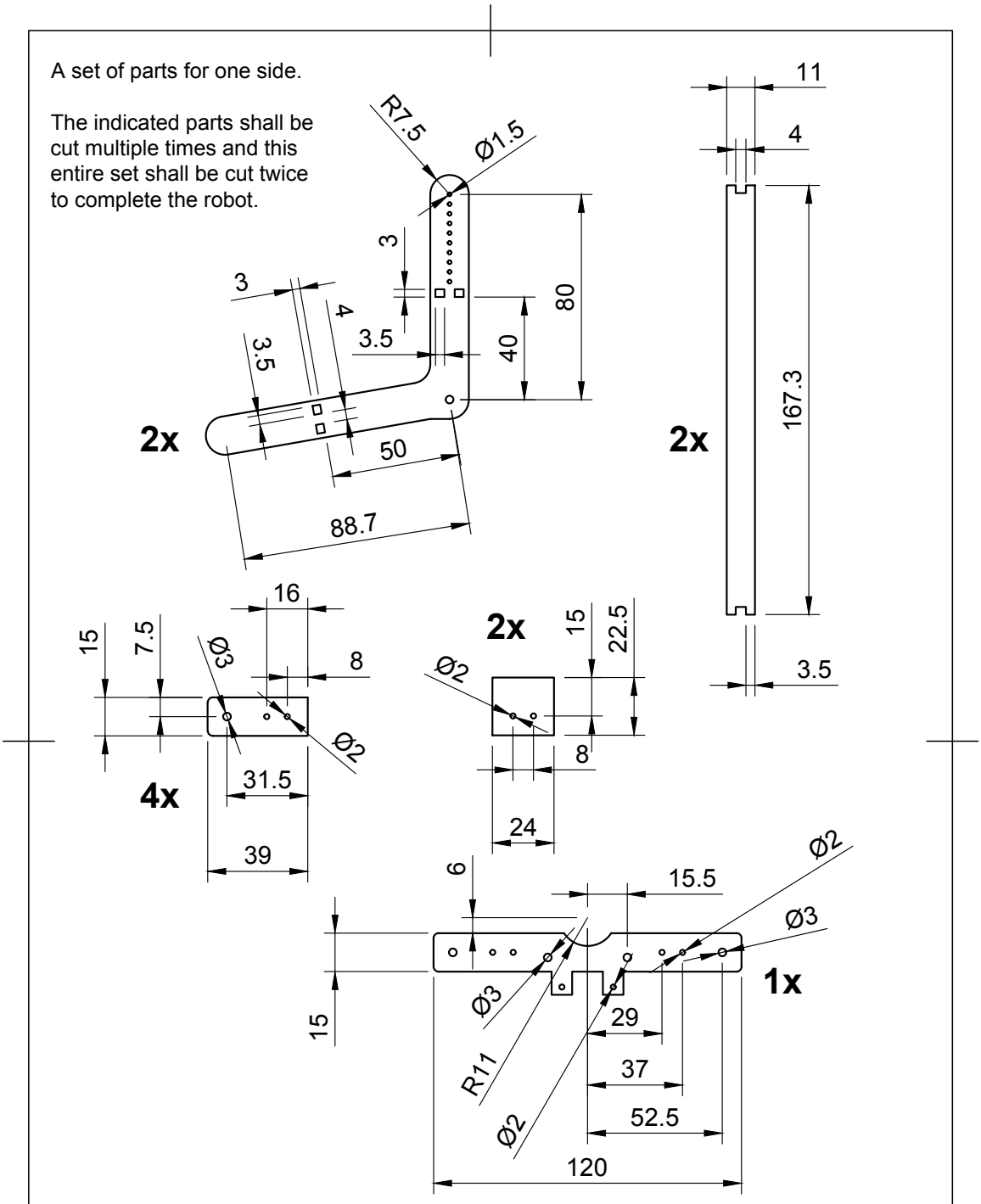
Dept.	Technical reference	Created by: Stefan Henriksen 11/08/2022	Approved by	
		Document type	Document status	
		Title: Self-Balancing Robot Full assembly	DWG No.	
			Rev. / Date of issue	Sheet: 1/1

12.2 Naked Robot



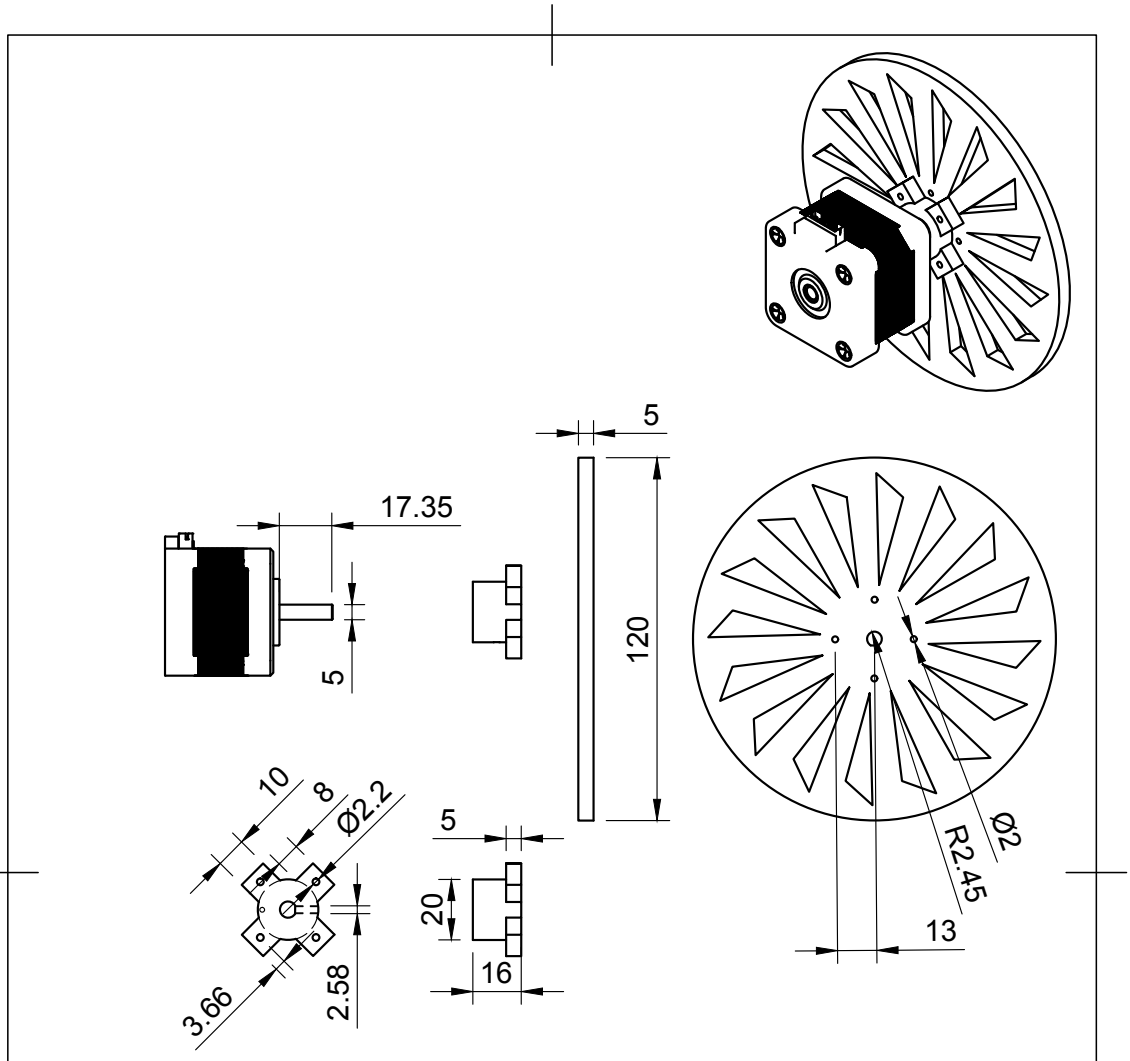
Dept.	Technical reference	Created by Anders Latif 08/08/2022	Approved by
		Document type	Document status
		Title Balancing Robot Naked Assembly	DWG No.
			Rev. / Date of issue / Sheet 1/1

12.3 Legs



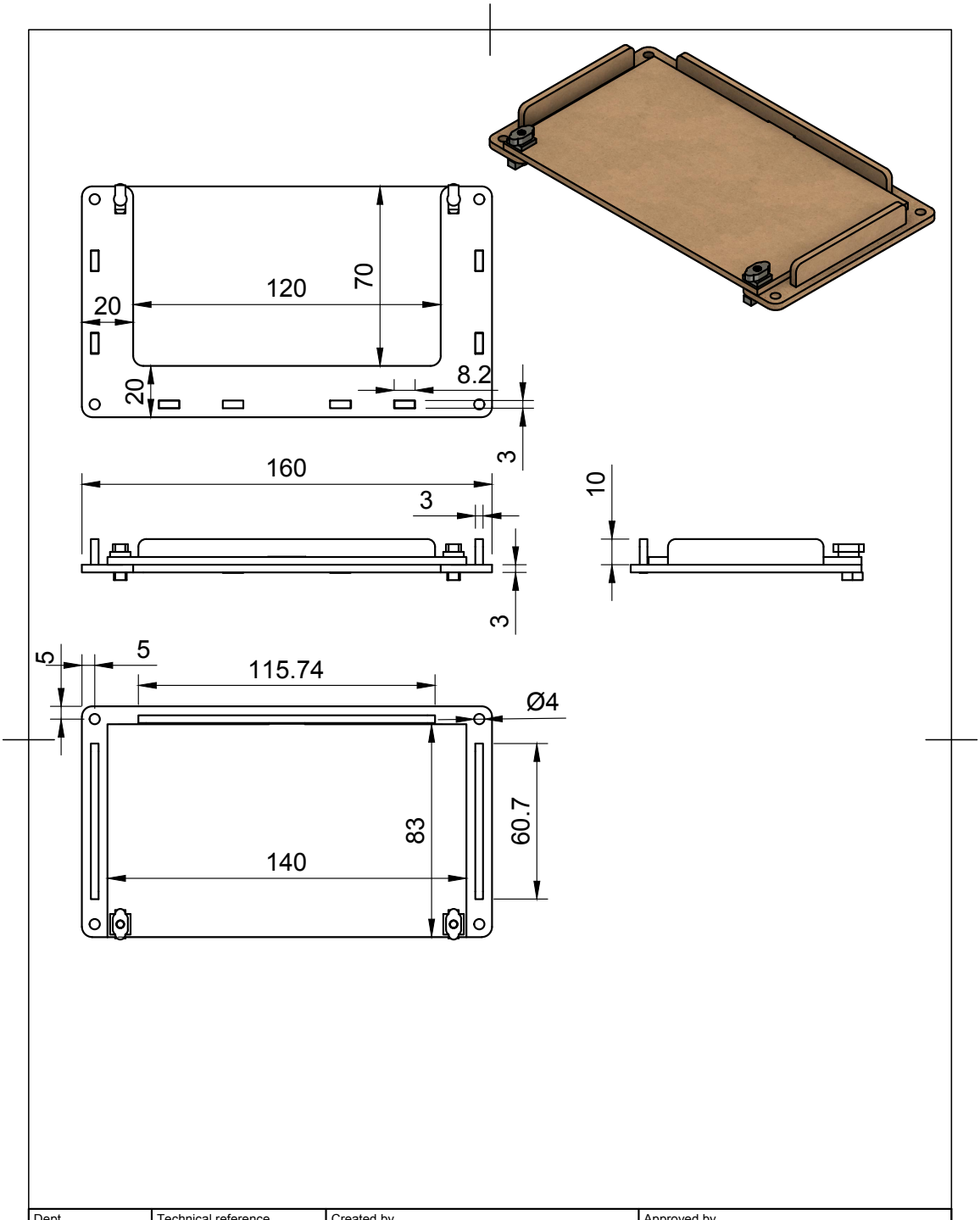
Dept.	Technical reference	Created by: Boris Karavasilev 8/10/2022	Approved by
		Document type	Document status
		Title: Stepper motor shelf - Leg	DWG No.
			Rev. / Date of issue / Sheet: 1/1

12.4 Wheel Assembly



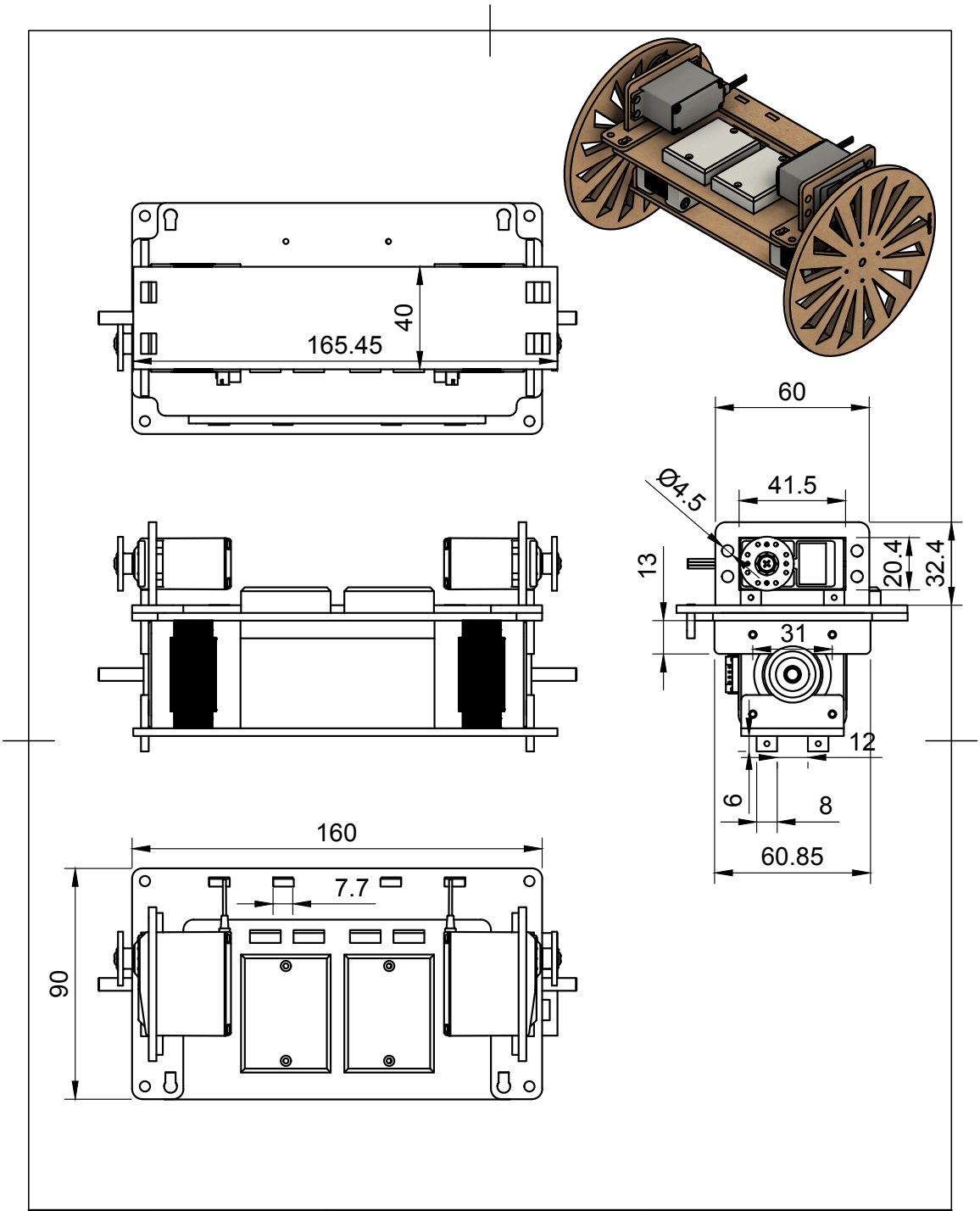
Dept.	Technical reference	Created by Stefan Henriksen 11/08/2022	Approved by	
		Document type	Document status	
		Title Wheel Assembly Exploded View	DWG No.	
			Rev. / Date of issue	Sheet 1/1

12.5 Base Shelf Design



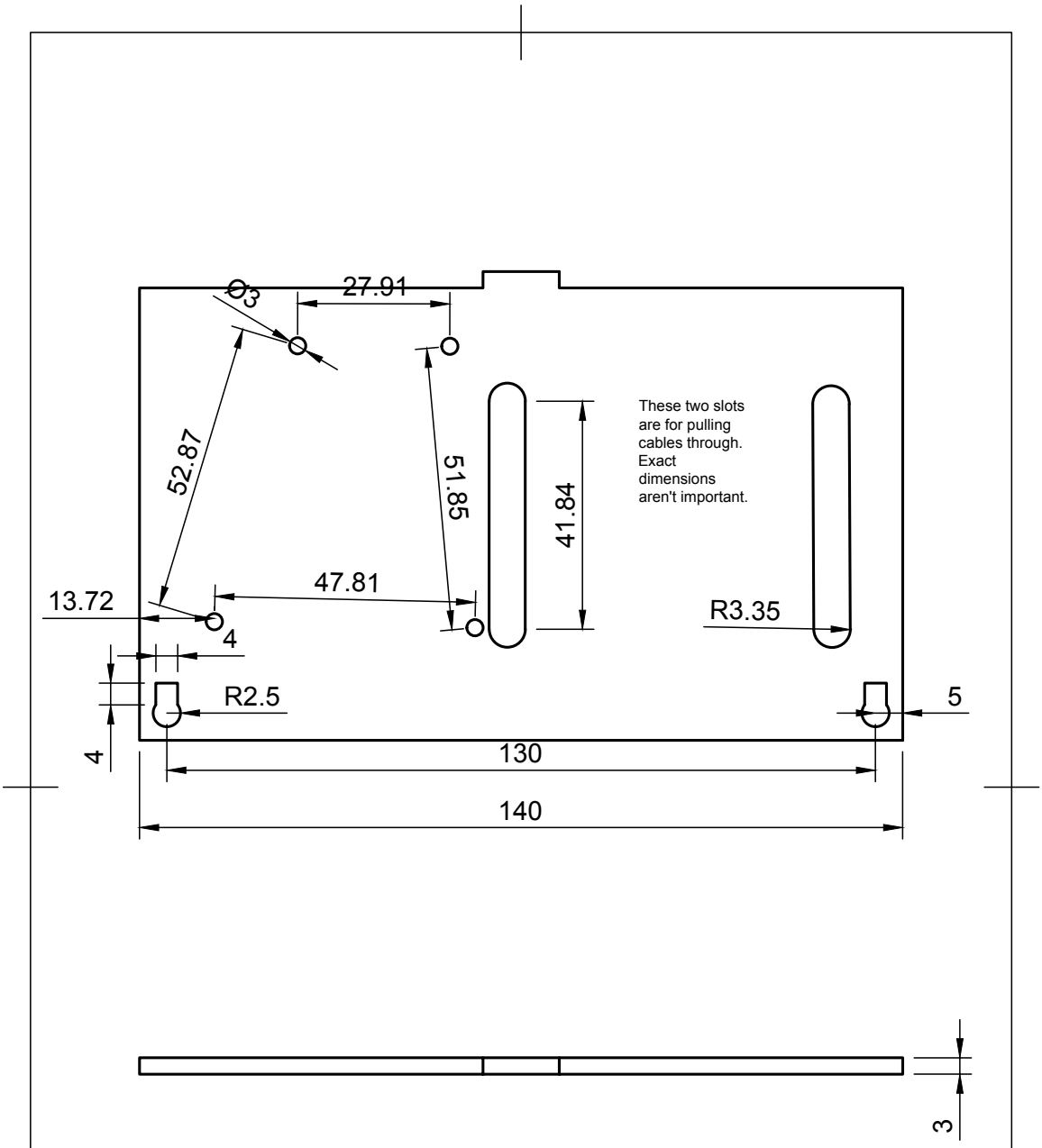
Dept.	Technical reference	Created by Stefan Henriksen 11/08/2022	Approved by	
		Document type	Document status	
		Title Base Shelf and Shelf Holder Assembly	DWG No.	
			Rev. / Date of issue	Sheet 1/1

12.6 Bottom Shelf



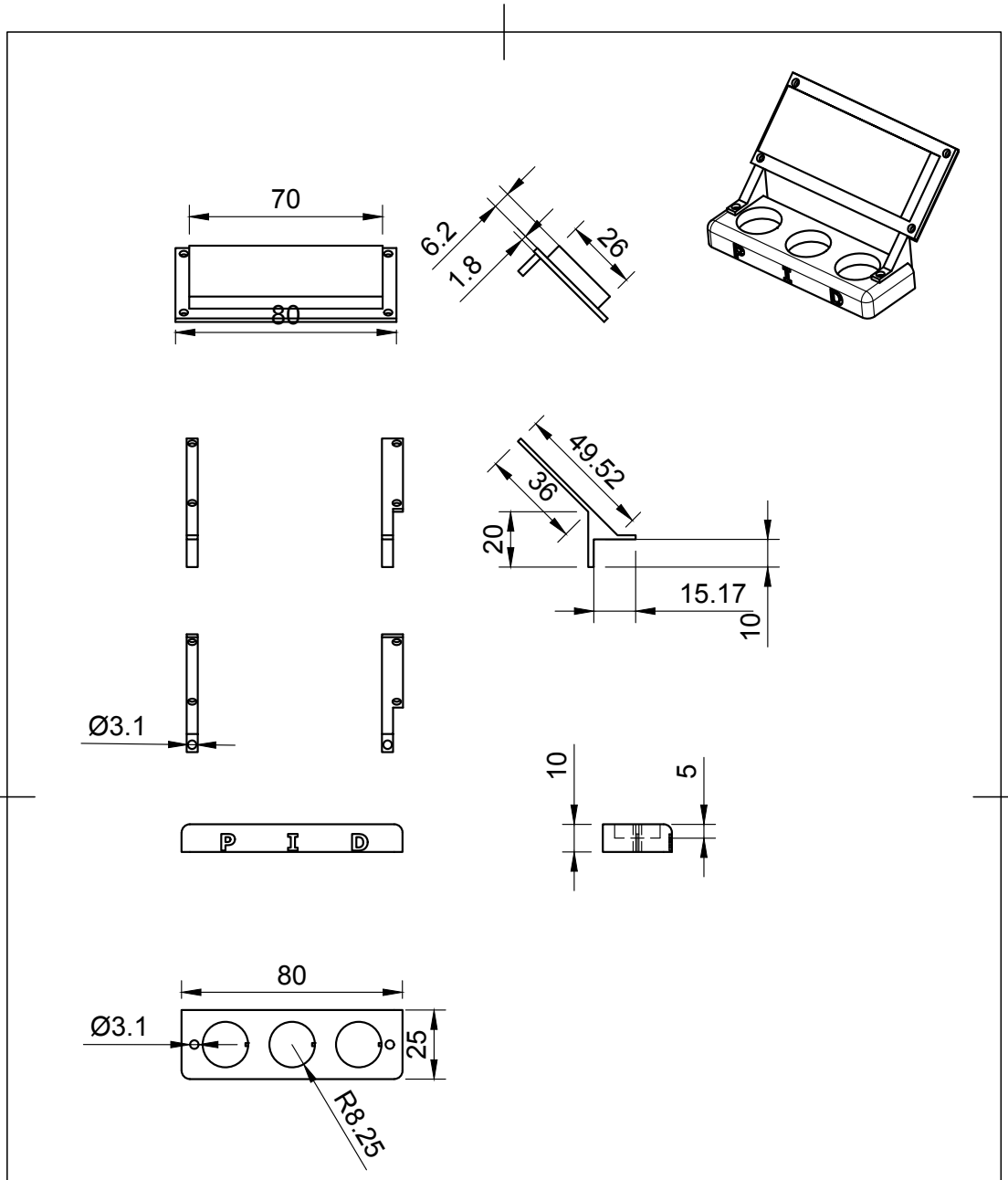
Dept.	Technical reference	Created by Stefan Henriksen 11/08/2022	Approved by	
		Document type	Document status	
		Title Stepper-Motor shelf Assembly	DWG No.	
			Rev. / Date of issue	Sheet 1/1

12.7 Arduino Shelf



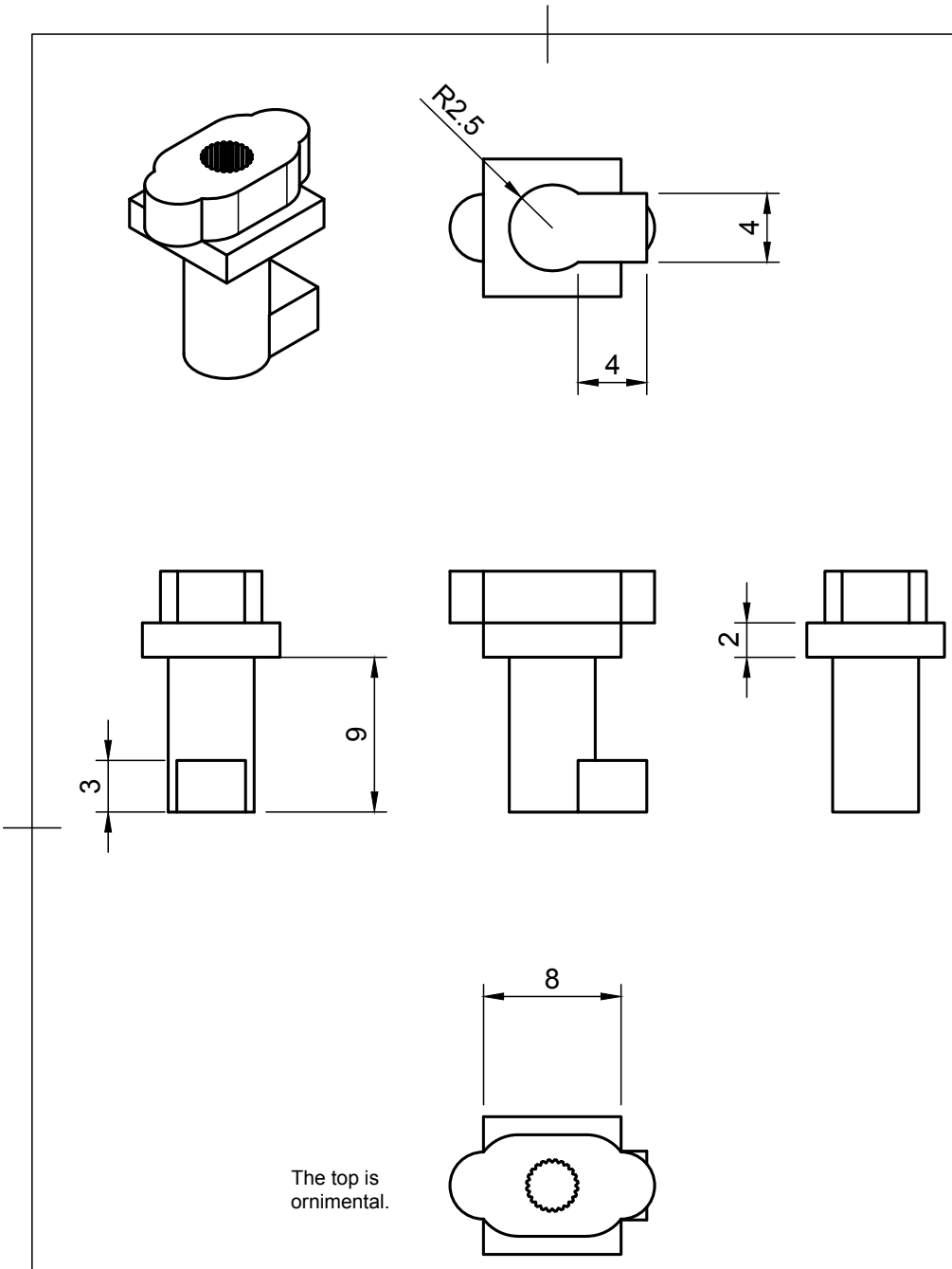
Dept.	Technical reference	Created by Anders Latif 08/08/2022	Approved by
		Document type	Document status
		Title Shelf_Arduino	DWG No.
			Rev. / Date of issue / Sheet 1/1

12.8 PID Tweaker & LCD Screen



Dept.	Technical reference	Created by Stefan Henriksen 11/08/2022	Approved by		
		Document type	Document status		
		Title PID Tweaker & LCD Screen Exploded view	DWG No.		
			Rev.	Date of issue	Sheet 1/1

12.9 Shelf Joint Key



Dept.	Technical reference	Created by Anders Latif 08/08/2022	Approved by	
		Document type	Document status	
		Title: Shelf Joint Key	DWG No.	
			Rev.	Date of issue / Sheet 1/1

13 SCHEMATICS

13.1 Arduino Pin Usage

Arduino Pin	Used by
A0	PID Tweaker potentiometer input
A1	PID Tweaker potentiometer input
A2	PID Tweaker potentiometer input
A3	
A4	MPU (SDA)
A5	MPU (SCL)
RX ← D0	
TX → D1	
D2	LCD Data
D3 (PWM)	LCD Data (PWM not used)
D4	LCD Data
D5 (PWM)	LCD Register Collect
D6 (PWM)	LCD Enable
D7	LCD Data
D8	SERVO_LEFT_LEG_PIN
D9 (PWM)	SERVO_RIGHT_LEG_PIN
D10 (PWM)	M1_STEP_PIN
D11 (PWM)	M1_DIR_PIN
D12	M2_DIR_PIN
D13	M2_STEP_PIN

13.2 Stepper motor driver schematic

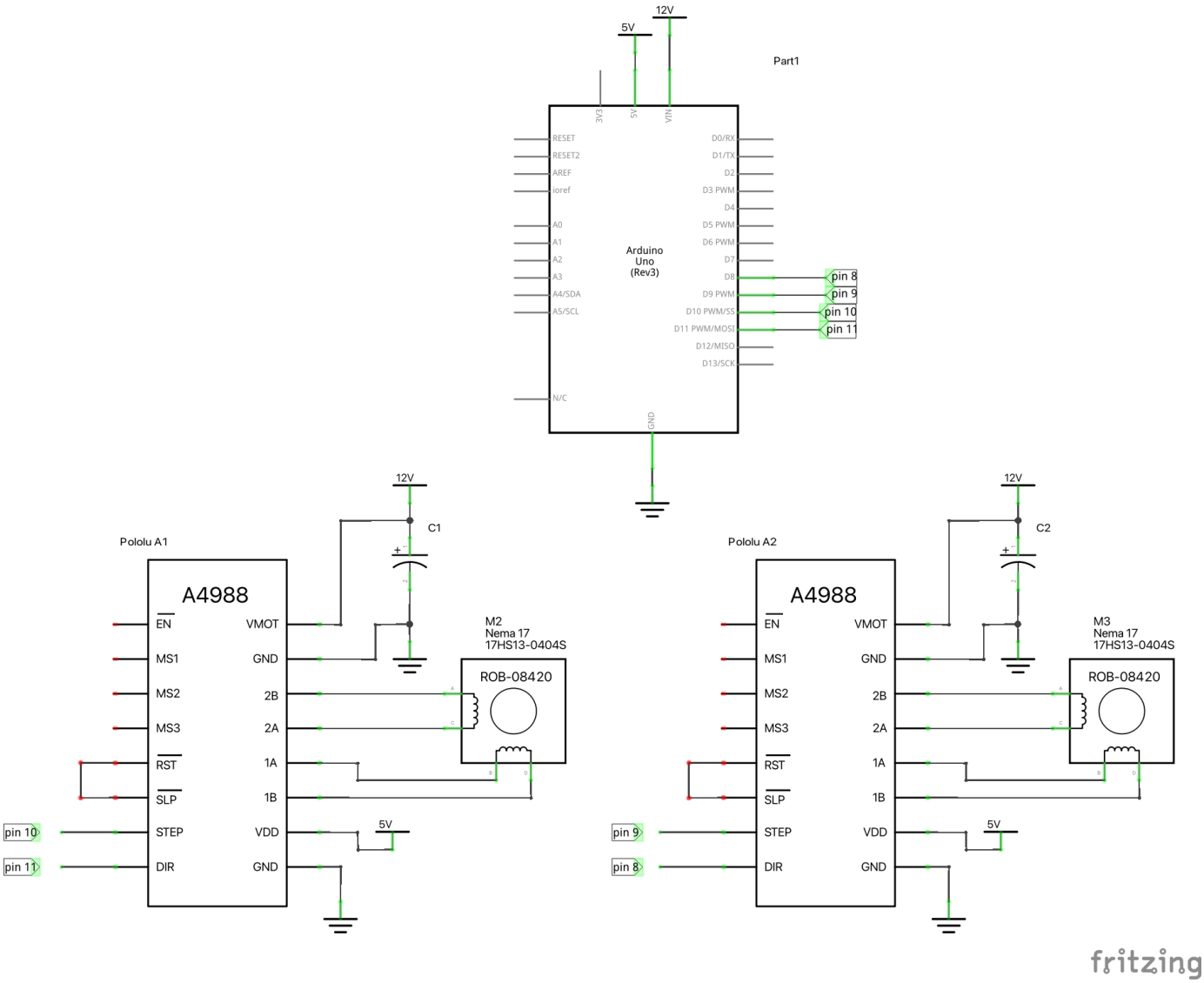


Fig. 4. The stepper motor drivers and stepper motors.

13.3 GY-521 schematic

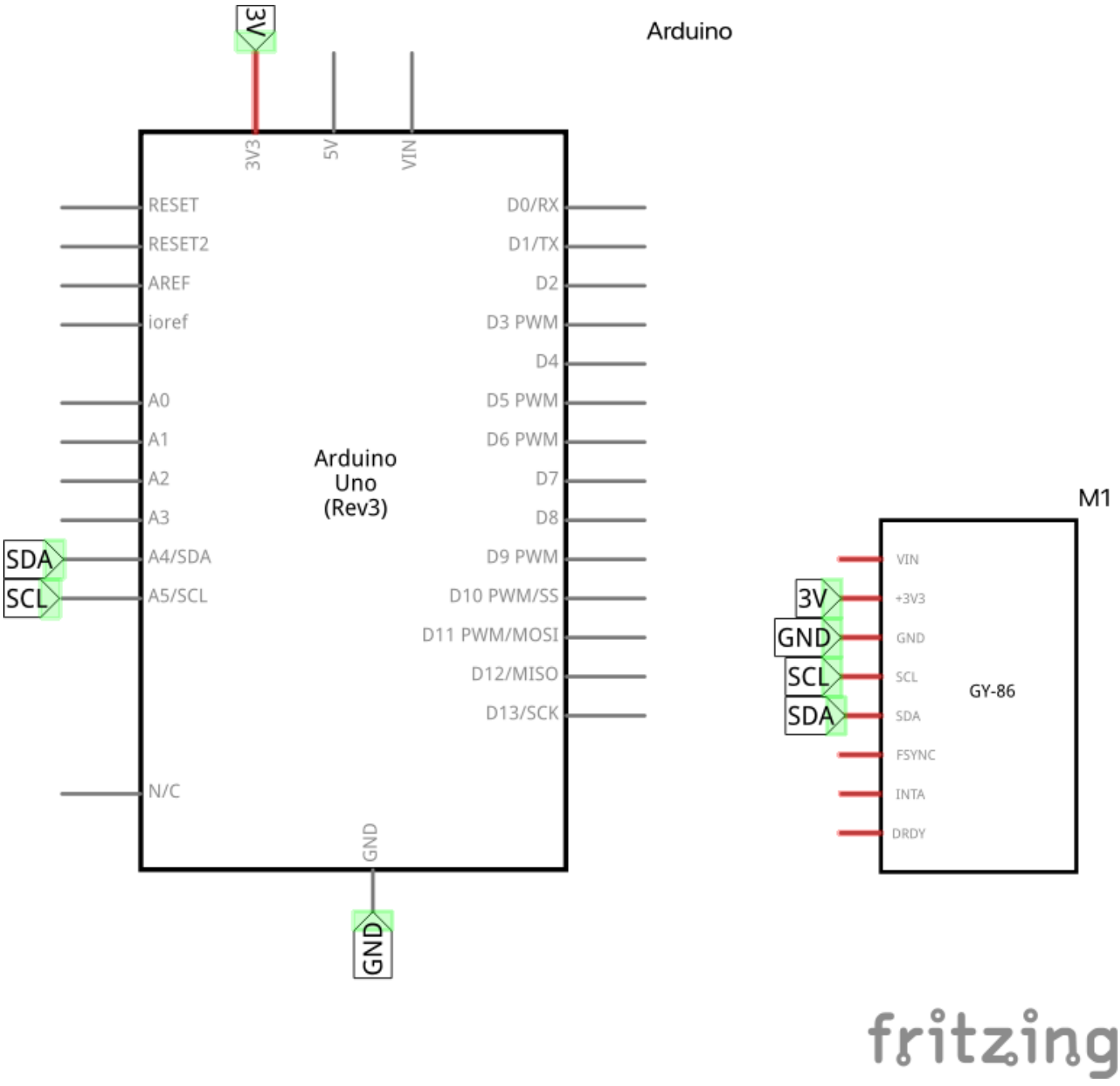


Fig. 5. The GY-521 Schematics that uses the I2C protocol.

13.4 PID Tweaker and LCD schematic

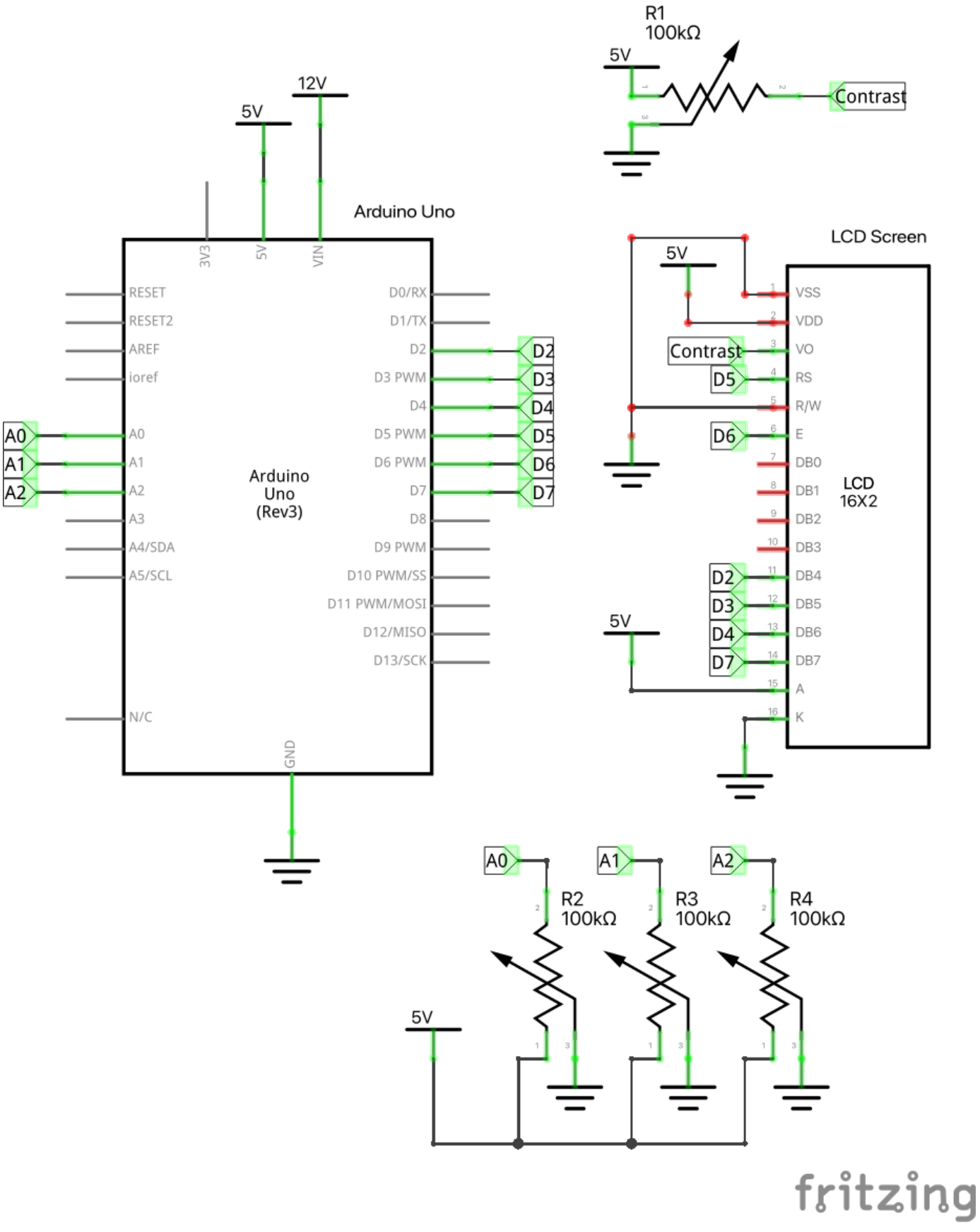


Fig. 6. Schematics for the PID Tweaker. Three potentiometers (R2-4) control the values Kp, Ki and Kd which are displayed on the LCD screen in real time. One potentiometer (R1) allows for ajusting the contrast for the LCD screen.

13.5 Servo schematic

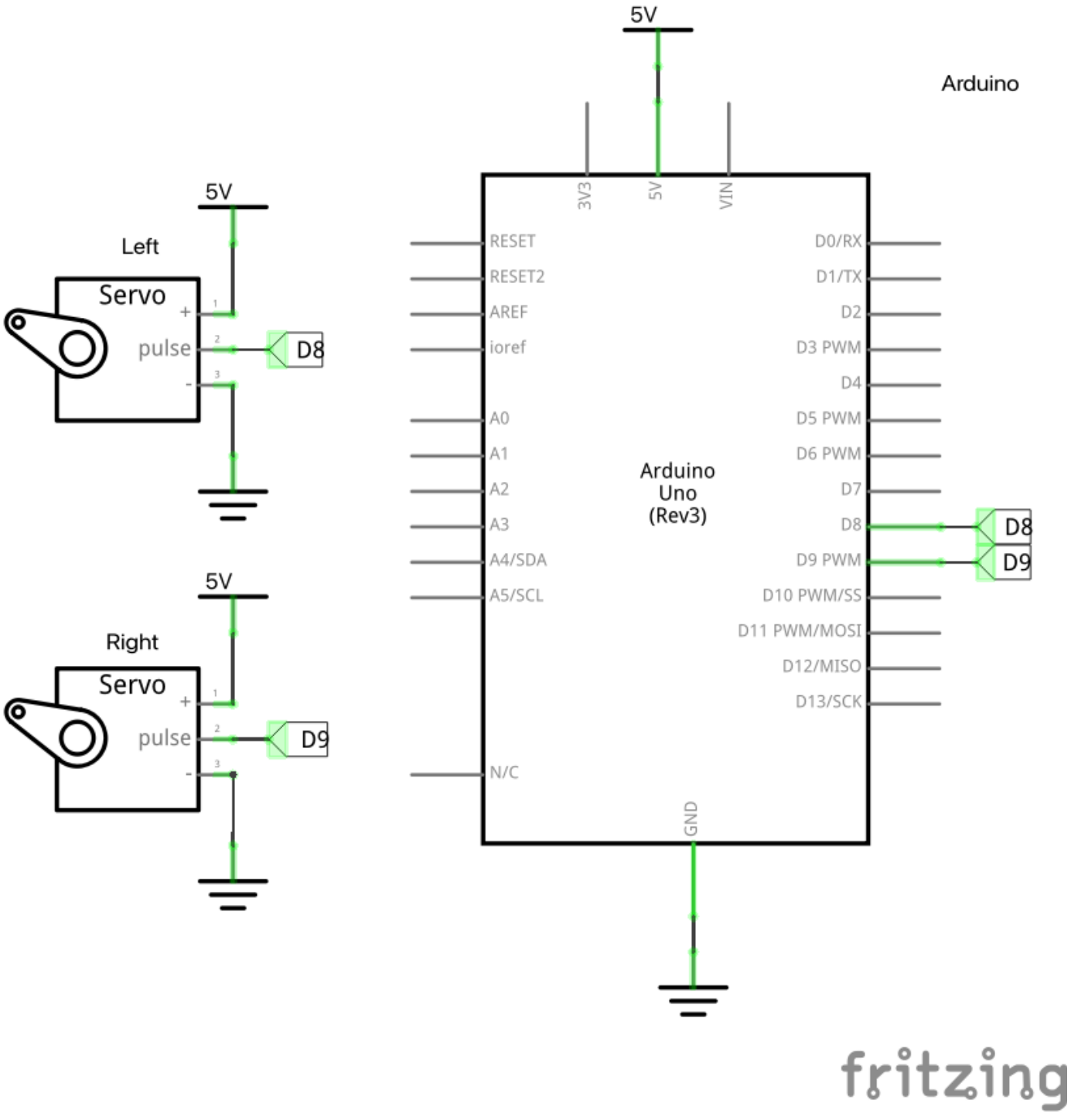


Fig. 7. Schematics for the Servos

14 PROGRAMMING

14.1 PID Control Diagram

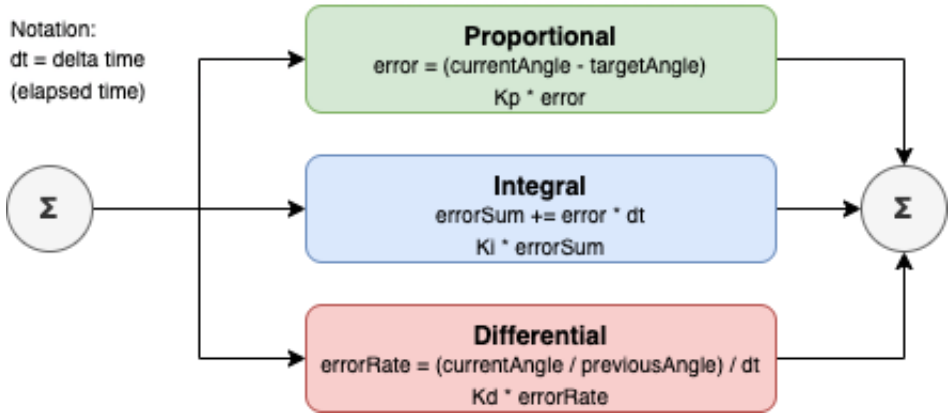


Fig. 8. A diagram that gives an overview of our implementation of PID.

14.2 main.ino

```
1
2  // Standard libraries
3  #include <Wire.h>
4  #include <LiquidCrystal.h>
5  #include <Servo.h>
6
7  // Our files
8  #include "LegsStepper.h"
9
10 /* Include the AccelStepper Library
11 (search in the library manager "accelstepper")
12 Library docs: http://www.airspayce.com/mikem/arduino/AccelStepper/ */
13 #include <AccelStepper.h>
14
15 /* From I2Cdev install zip the following 2 folders under Arduino and install them through
       the IDE
16 https://github.com/jrowberg/i2cdevlib
17
18  1) Download the ZIP from GitHub
19  2) Go to "i2cdevlib-master.zip > i2cdevlib-master > Arduino"
20  3) Zip "I2Cdev" and "MPU6050" into individual zip archives
21  4) Include them one by one through the Arduino IDE
22       -->  "Sketch > Include Library > Add .ZIP Library..."
23 */
24 #include "I2Cdev.h"
25 #include "MPU6050.h"
26
27
28 //************************************************************
29 // GLOBAL VARIABLES
30 //************************************************************
31
32 /*
33  * Fine tuning the PID constants
34 1. Set Ki and Kd to zero and gradually increase Kp so that the robot starts to oscillate
       about the zero position.
35
36 2. Increase Ki so that the response of the robot is faster when it is out of balance.
37 Ki should be large enough so that the angle of inclination does not increase.
```

```
The robot should come back to zero position if it is inclined.

3. Increase Kd so as to reduce the oscillations. The overshoots should also be reduced by now.

4. Repeat the above steps by fine tuning each parameter to achieve the best result.
*/
float Kp = 150; //100;
float Ki = 5.1; // Influence of the sum of integrated errors in the final equation
float Kd = 0.005;

const float SPEED_OF_INTEGRATION = 480; // Multiplier of the integrated error
const float MAX_INTEGRATION_SUM = 70; // Floor and ceiling in order to make it easier to go back to zero sum
                                      // (to not be too far in the positive or negative numbers)


// Define how often code in inner loop should run
const int loopMax = 1000;
int loopCounter = loopMax;



//************************************************************
// Stepper motor variables
//************************************************************

// Define motor interface type
#define motorInterfaceType 1

const int MOTOR_MAX_SPEED = 1000;

// PIN CONNECTIONS
const int M1_DIR_PIN = 11;
const int M1_STEP_PIN = 10;

const int M2_DIR_PIN = 12; // 8 to 13
const int M2_STEP_PIN = 13; // 9 to 13

// LCD Data Pins
const int LCD_d_4 = 2;
const int LCD_d_5 = 3;
const int LCD_d_6 = 4;
const int LCD_d_7 = 7;

const int LCD_Register_Sel = 5;
const int LCD_Enable = 6;

// Creates an instance
AccelStepper stepper1(motorInterfaceType, M1_STEP_PIN, M1_DIR_PIN);
AccelStepper stepper2(motorInterfaceType, M2_STEP_PIN, M2_DIR_PIN);


//************************************************************
// LCD Screen
//************************************************************

// Setup LCD pins to Arduino pins
LiquidCrystal lcd(LCD_Register_Sel,
                  LCD_Enable,
```

```
                  LCD_d_4,
                  LCD_d_5,
                  LCD_d_6,
                  LCD_d_7);

// Ranges for PID tweak values
// Change these when testing
float KpMin = 100;
float KpMax = 250;

float KiMin = 0;
float KiMax = 10;

float KdMin = 0;
float KdMax = 0.01;

// Maximum analog read value for our arduino
const int PotMax = 1023;

const float KpFactor = (KpMax - KpMin) / PotMax;
const float KiFactor = (KiMax - KiMin) / PotMax;
const float KdFactor = (KdMax - KdMin) / PotMax;

// The readings from the potentiometers
int KpRead = 0;
int KiRead = 0;
int KdRead = 0;

// PID Tweaker pins
const int TWEAK_KP_PIN = A0;
const int TWEAK_KI_PIN = A1;
const int TWEAK_KD_PIN = A2;



//************************************************************
// MPU6050 variables = Accelerometer + Gyroscope
//************************************************************
MPU6050 mpu;

/* MPU */
int16_t AcY, AcZ, GyX;


//************************************************************
// PID variables
//************************************************************

/* MOTOR */
volatile int motorPower, gyroRate;

/* PID */
volatile float accAngle, gyroAngle, currentAngle, prevAngle=0, error, prevError=0,
    errorSum=0, errorRate=0;

const float targetAngleCalculated = -10.19;
const float targetAngleOffset = -1.06;
const float targetAngle = targetAngleCalculated + targetAngleOffset;
/* This is assuming that it is in a forward resting position on the support wheels. If
    leaning back then change the sign.
 *  Code for recalculating the target angle
```

```
 *    float sum = 0;
  float readings = 20000;
  for (int i = 0; i < readings; i++) {
      AcY = mpu.getAccelerationY();
      AcZ = mpu.getAccelerationZ();
      sum += atan2(AcY, AcZ)*RAD_TO_DEG;
  }
  Serial.println(sum / readings);
 */

unsigned long prevTimePID;
unsigned long currentTimePID;
float elapsedTimePIDSeconds;

//************************************************************
// Servos - Legs
//************************************************************

// One leg on each side share the same pins
const int SERVO_LEFT_LEG_PIN = 8; // todo change to the right pin
const int SERVO_RIGHT_LEG_PIN = 9; // todo change to the right pin
LegsStepper rightLeg;
LegsStepper leftLeg;

//************************************************************
// SETUP
//************************************************************
void setup() {
  Serial.begin(9600);

  // ******** Stepper Motor
  stepper1.setMaxSpeed(MOTOR_MAX_SPEED);
  stepper2.setMaxSpeed(MOTOR_MAX_SPEED);
  stepper1.setSpeed(0);
  stepper2.setSpeed(0);

  // ******** Leg Servos
  rightLeg.initialize(SERVO_RIGHT_LEG_PIN, 0);
  leftLeg.initialize(SERVO_LEFT_LEG_PIN, 180);

  // ******** LCD - Welcome Screen
  // Setup the first (unchanging) line of the display
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("Callibrating...");

  // this is because turning on the power supply jolts the motors and this interfers with
     the mpu callibration.
  delay(400);

  // ******** MPU
  mpu.initialize();
  // todo raise the value when done developing
  mpu.CalibrateAccel(8);
  mpu.CalibrateGyro(8);

  // ******** LCD - Info Screen
  lcd.setCursor(0, 0);
  lcd.print("␣P␣␣␣␣I␣␣␣␣D␣␣");
  lcd.setCursor(0, 1);
  lcd.print("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣");
```

```
215
216   // ******** PID
217   prevTimePID = millis();
218
219
220 }
221
222 //**************************************************************
223 // LOOP
224 //**************************************************************
225 void loop() {
226   AcY = mpu.getAccelerationY();
227   AcZ = mpu.getAccelerationZ();
228   GyX = mpu.getRotationX();
229
230   calculatePID();
231
232   stepper1.setSpeed(motorPower);
233   stepper2.setSpeed(-motorPower); // inverted speed because motors are facing opposite
        directions
234
235   stepper1.runSpeed();
236   stepper2.runSpeed();
237
238   if (loopCounter < loopMax) {
239       loopCounter++;
240     } else {
241       loopCounter = 0;
242       updatePIDValuesFromTweaker();
243       showPIDValuesOnScreen();
244   }
245 }
246
247
248 void calculatePID() {
249   currentTimePID = millis();
250   elapsedTimePIDSeconds = (currentTimePID - prevTimePID) / 1000.0f;
251
252   accAngle = atan2(AcY, AcZ)*RAD_TO_DEG;
253
254   // The MPU6050 outputs values in a format in a range that needs to be converted
255   gyroRate = map(GyX, -32768, 32767, -250, 250);
256   gyroAngle = (float) gyroRate * (elapsedTimePIDSeconds);
257
258
259   // even the angle out by adding to the previous angle
260   // this is the formula for the complementary filter
261   // Page 11: https://d1.amobbs.com/bbs_upload782111/files_44/ourdev_665531S2JZG6.pdf
262   currentAngle = 0.98 * (prevAngle + gyroAngle) + 0.02 * (accAngle);
263
264   error = currentAngle - targetAngle;
265   errorSum += SPEED_OF_INTEGRATION * error * elapsedTimePIDSeconds;
266
267   errorSum = constrain(errorSum, -MAX_INTEGRATION_SUM, MAX_INTEGRATION_SUM);
268
269   errorRate = (currentAngle-prevAngle)/elapsedTimePIDSeconds;
270
271   // calculate output from P, I and D values
272   motorPower = Kp*error + Ki*errorSum - Kd*errorRate;
273   motorPower = constrain(motorPower, -MOTOR_MAX_SPEED, MOTOR_MAX_SPEED);
274
```

```
275
276   prevAngle = currentAngle;
277   prevTimePID = currentTimePID;
278 }
279
280 void updatePIDValuesFromTweaker() {
281
282   // Find the proportional, update the value and write to LCD
283   KpRead = PotMax - analogRead(TWEAK_KP_PIN);
284   Kp = KpRead * KpFactor + KpMin;
285
286
287   // Find the integral, update the value and write to LCD
288   KiRead = PotMax - analogRead(TWEAK_KI_PIN);
289   Ki = KiRead * KiFactor + KiMin;
290
291
292   // Find the derivative, update the value and write to LCD
293   KdRead = PotMax - analogRead(TWEAK_KD_PIN);
294   Kd = KdRead * KdFactor + KdMin;
295
296 }
297
298 void showPIDValuesOnScreen() {
299   lcd.setCursor(0, 1);
300   lcd.print(String(Kp, 0));
301
302   lcd.setCursor(5, 1);
303   lcd.print(String(Ki, 2));
304
305   lcd.setCursor(11, 1);
306   lcd.print(String(Kd, 3));
307 }
```

14.3 LegStepper.h

```
1  #ifndef _LegsStepper_H_
2  #define _LegsStepper_H_
3
4  #include "Arduino.h"
5
6
7  class LegsStepper {
8      public:
9          LegsStepper();
10         void initialize(int pinNumber, int topIsInPosition);
11         void moveTo(int position);
12         void moveToTop();
13         void moveToBottom();
14 };
15
16 #endif
```

14.4 LegStepper.cpp

```
1  #include "LegsStepper.h"
2
3  #include <Servo.h>
4  Servo servo;
5
```

```
// this is to account for the fact that the servos are opposite each other so 0 for one is position 180 for another
int topIsInPosition = 0;

LegsStepper::LegsStepper() {
}

void LegsStepper::initialize(int pinNumber, int topIsInPosition) {
  servo.attach(pinNumber);
  topIsInPosition = topIsInPosition;
}

void moveTo(int position) {
  servo.write(position + topIsInPosition);
}


void LegsStepper::moveToTop() {
  servo.write(0 + topIsInPosition);
}

void LegsStepper::moveToBottom() {
  servo.write(180 + topIsInPosition);
}
```