

Production, assembly and programming of a RoboCrane

Lukas Janssens

Martijn Spaas

Bachelor's thesis in electromechanics

Bergen, Norway 2023



Høgskulen
på Vestlandet

Production, assembly and programming of a RoboCrane

Lukas Janssens

Martijn Spaas

Department of Mechanical- and Marine Engineering

Western Norway University of Applied Sciences

NO-5063 Bergen, Norway

Preface

This thesis was for the most part written at the department of Mechanical- and marine engineering at the Western University of Applied Sciences (HVL) in Bergen, Norway in conjunction with Thomas More Kempen located in Geel, Belgium.

This thesis was supervised by Thorstein Ravneberg Rykkje, a doctoral student. We would like to thank Erwin Van de Put for supporting us in this project. We also would like to express our gratitude to the following engineers: Harald Moen, Frode Wessel Jansen, and Nafez Ardestani for helping us in ordering parts, machine custom parts, but also for teaching us how to TIG weld aluminium and for assisting us in general with our project.

Abstract

In this thesis we continue to work on a self-levelling platform that has been designed and from which small prototypes have been built. The platform is part of a bigger project, being a scale model self-levelling crane based on the Offshore Passenger Transport System (OPTS) from the company Palfinger.

Lukas Janssens worked mostly on the electrical part of the thesis, such as programming the gyroscope, the base with the first joint, and the movement of the arm. This is all programmed on Arduino(C++) and PLC(Tia Portal). A communication between Arduino and PLC has been established in order for this to work. After programming and testing everything, it was time to solder everything for a good and sturdy connection between the electronic components.

Martijn Spaas worked mostly on the mechanical side of the project which included researching and designing a new levelling and rotating base. The platform is designed in Autodesk Inventor 2021. Some parts have been 3D-printed in PLA and others are made of aluminium. He redesigned and printed other parts for better tolerances and optimizing the actuator's stroke length. This was necessary to reduce play and friction in the crane. There were also some problems with dimensions which needed to be fixed. Martijn Spaas also took care of programming the steppers, servo, second, third and fourth joint and he also cleaned up the PLC program at the end.

We worked together and:

- Welded and assembled the crane.
- Learned about the previous designs.
- Looked up missing information about electrical components.
- Read the Arduino and PLC programs that were already made.

Sammendrag

I dette prosjektet fortsetter vi arbeidet med en automatisk nivå regulerende plattform som har blitt designet og bygget. Plattformen er en del av et større prosjekt med mål om å bygge en skalert modell av en automatisk nivå regulerende kran basert på Palfinger's «Offshore Passenger Transport System (OPTS)».

Lukas Janssens har arbeidet med det elektriske delen av prosjektet. Programmeringen av gyroskopet, basen med det første leddet og bevegelsen av armen. Alt er programmert i Arduino (C++) og PLC (Tia Portal). Kommunikasjon har blitt etablert mellom Arduino og PLC for at dette skal fungere. Etter programmering og testing ble alle koblinger loddet sammen for god og solid kontakt mellom elektroniske komponenter.

Martijn Spaas har jobbet med den mekaniske delen av prosjektet, noe som inkluderer undersøkelser og design av en ny nivå-regulerende or roterende base. Plattformen ble designet i Autodesk Inventor 2021. Noen deler er 3D-printet i PLA og andre er laget av aluminium. Noen deler ble redesignet for å minske friksjon og slark i kranen, i tillegg til noen dimensjoner som måtte fikses. Martijn Spaas programmerte stepper og servo motorene i tillegg til å renske opp i PLC programmet.

Vi jobbet sammen og:

- Sveiste og monterte kranen
- Lærte om det tidligere designet
- Fant manglende informasjon om elektriske komponenter
- Leste de tidligere Arduino og PLC programmene som var laget.

Table of contents

Preface.....	V
Abstract	VII
Sammendrag.....	IX
Nomenclature	XV
1. Introduction	1
1.1 Background	1
1.2 Problem statement	1
2. Method	3
2.1 Empathize.....	3
2.2 Define	3
2.3 Ideate	3
2.4 Prototype	3
2.5 Test.....	3
3. Mechanical aspect	4
3.1 Introduction	4
3.2 Summary crane parts	4
3.2.1 Base design.....	5
3.2.2 Angled base design.....	5
3.2.3 Crane design	6
3.3 Turning mechanism.....	7
3.3.1 Choice for design.....	7
3.3.2 Realisation	10
3.3.3 Gears.....	11
3.4 Base design.....	12
3.5 Remaking truss 2.....	18
3.6 Remaking the second joint	19
3.7 New truss 2.....	20

3.8	New wrist, fifth joint and basket	21
3.9	Telescopic arm	21
4.	Electrical aspect.....	22
4.1	Components.....	22
4.1.1	Gyroscope.....	22
4.1.2	Linear actuator base (Ewellix).....	23
4.1.3	H-bridge.....	23
4.1.4	Stepper motor with driver.....	24
4.1.5	ADAC-Click module.....	25
4.1.6	Power supplies.....	26
4.1.7	Linear actuators crane (Actuonix)	27
4.1.8	Arduino.....	28
4.1.9	PLC.....	28
4.2	Electrical wiring diagram	30
4.2.1	Base and first joint.....	30
4.2.2	Gyroscope.....	31
4.2.3	Steppers + servo basket	32
4.2.4	Linear actuators arm	33
4.3	Programming aspect.....	34
4.3.1	Arduino.....	34
4.3.2	Base and first joint.....	34
4.3.3	Gyroscope.....	34
4.3.4	Steppers + servo basket	35
4.3.5	Linear actuators arm	35
4.3.6	PLC.....	36
5.	Conclusion.....	40
6.	References	41
	List of tables	45

Attachments.....	46
Drawings plasma cutter.....	46
Drawing 1.....	46
Drawing 2.....	47
Drawing 3.....	48
Drawing 4.....	49
Drawing 5.....	50
Drawing 6.....	51
Drawing 7.....	52
Drawing 8.....	53
Drawing 9.....	54
Drawing 10.....	55
Drawing 11.....	56
Drawing 12.....	57
Drawing 13.....	58
Drawing machined.....	59
Drawing assembly crane.....	60
Drawing assembly base.....	63
Crane limbs measurements.....	65
Arduino programs.....	66
Arduino 1: Base and first joint.....	66
Arduino 2: Gyroscope.....	68
Arduino 3: Steppers + servo.....	70
Arduino 4: Linear actuators arm.....	72
Table wiring connections.....	74

Nomenclature

HVL *Western Norway University of Applied Sciences*

OPTS *Offshore Passenger Transfer System*

FL *Front Left*

FR *Front Right*

BL *Back Left*

BR *Back Right*

VDC *Volts (direct current)*

FB *Function Block*

FC *Function*

OB *Organization Block*

PWM *Pulse width Modulation*

DAC *Digital-Analog-Converter*

ADC *Analog-Digital-Converter*

TIG *Tungsten Inert Gas*

PLC *Programmable Logic Controller*

1. Introduction

1.1 Background

In this Bachelor's thesis, we built a scale model, named 'RoboCrane', of the Offshore Passenger Transfer System, OPTS for short. Palfinger took over the project from Lift2Work and they are now solving some flaws. The OPTS consists of two major parts, the base and the crane itself. The base is self-levelling and can counteract the pitch and the roll of the vessel. The crane is mounted on the base and counteracts the yaw, heave, surge, and sway. Although a lot of big ships have dynamic positioning nowadays, meaning they do not have to compensate for surge, sway, and yaw. (1) If they turn on the 'compensation mode', the OPTS can keep the endpoint still within 10 centimetres of accuracy (use Figure 1 - Ship movements to comprehend the movements we discussed). There has already been research for this project in the previous year by Jasper Gielen and Bram Deboel (3). In their Bachelor's thesis, they analysed the best way to make the base. They came up with an end solution. This solution was produced in the summer later that year. The result was a base with a few centimetres of play and a lot of information for components. We invested a lot of effort into learning and understanding the project and developments from the previous year. With the OPTS they can transfer passengers and cargo efficiently and safely from a moving ship to fixed or semi-floating offshore installations.

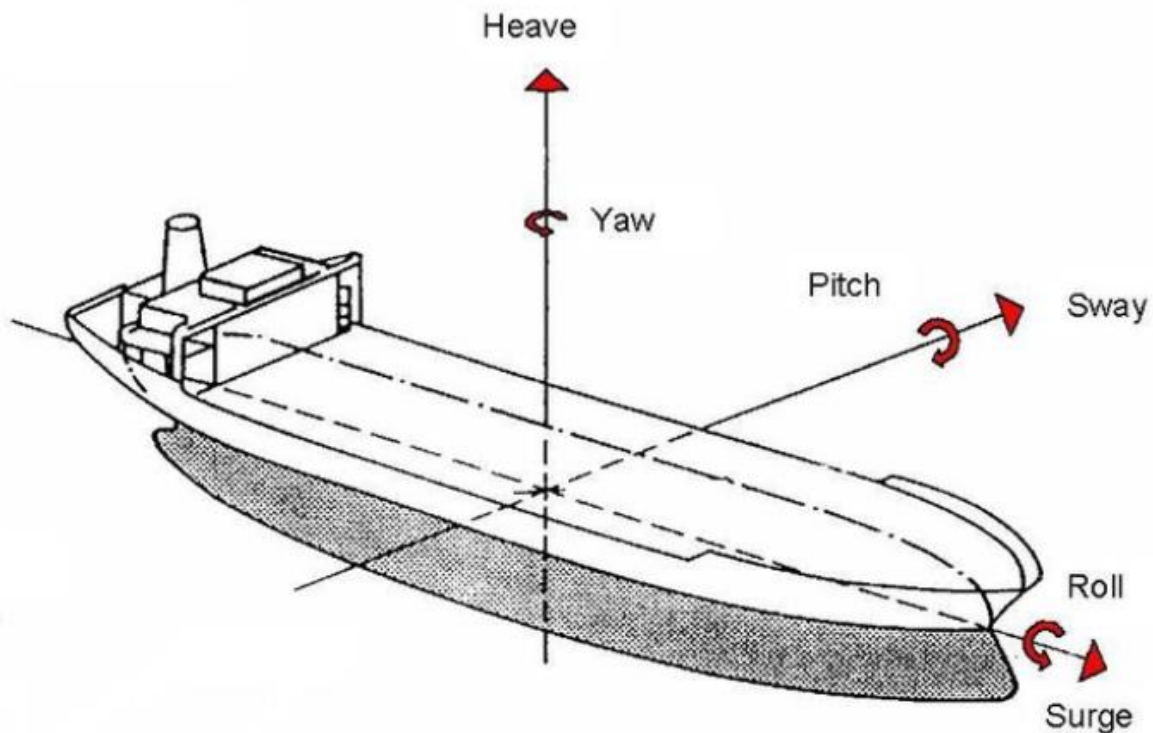


Figure 1 - Ship movements (2)

1.2 Problem statement

A scale model of the crane with platform is available. However, there is a bit (too much) play on the platform that needs to be adjusted. Parts need to be made/adjusted to let the crane and platform move properly. If there is too much play, we cannot calculate the endpoint of the crane. The crane and platform are not yet connected and tested. As minimum delivery, we will have a crane with platform that works with switches. We can let the actuators move and read the data from the sensors properly. It is possible to let the crane move manually. As an extra, we would like to implement the dynamics equations,

decrease delay and reduce play as much as possible. This would mean that the OPTS could calculate and compensate for itself.

2. Method

In this thesis a certain overall method is followed, named design thinking which consists of five steps (empathize, define, ideate, prototype, and test). These steps were not followed in a certain order as we went back and forth with the different steps throughout the thesis. Below is a more detailed description of examples on how we implemented these steps in the thesis (4).

2.1 Empathize

During the entirety of this Bachelor's thesis, we tried to communicate and think about the actual needs of the RoboCrane. Back then it was said that the crane had to be sturdy with not much play anymore. The crane should be manually controllable and the place of each joint should be trackable at any time. While getting into the project, the principles of safety, user-friendliness, and writing clean programs,... also came to mind.

2.2 Define

While thinking about the needs for the RoboCrane, more practical ideas came to mind. Here are some examples: The RoboCrane needs to be sturdy since the previous base had 10 centimetres of play. It would have been too complicated and taken too much time if we tried to fix this. Therefore, a whole new base had to be designed and produced. The crane had to be manually controllable, so buttons or switches had to be implemented. The place of each joint should be trackable, so feedback from every actuator was needed. To control all aforementioned requirements we needed a controller that we could program, like Arduino, PLC,... For safety reasons, the crane should be locked when an error occurs, to avoid hitting people standing around it. These are the big user statements that were important throughout the thesis.

2.3 Ideate

In this part of the five-step method, all the ideas that came to mind were located. We discussed how to make the base, which type of bearing to make for the turning mechanism, how to remake the parts, reprogram everything or build on what was built before. There were too many ideas throughout the thesis to discuss everything.

2.4 Prototype

After the initial theoretical thinking and internal discussion, the first digital and physical prototypes of the different crane parts(e.g. the base and the arm) were constructed. These prototypes were tiny steps in the right direction. If the subsequent testing of the prototypes failed, the prototypes needed to be adjusted accordingly.

2.5 Test

We tested the prototypes and if they were not good enough, adjustments were made. This loop kept on going till the prototype was good enough to meet the standards that were set in the 'define' step. These tests included theoretical calculations and practical testing.

3. Mechanical aspect

3.1 Introduction

Although the entire crane was designed and 3D-printed almost every part needed to be remade. There was too much play in the joints, they could not support their weight, or the dimensions were off. We noticed that all designs were scaled-down versions of the big crane with some added features. Editing dimensions in the designs was not easy as they were made by combining solid shapes derived from the original full-scale crane designs. This made adjustments difficult, as we could not easily alter the different dimensions. Editing the parts was sort of a cumbersome process.

3.2 Summary crane parts

In Figure 2 - RoboCrane: you can see the completed crane.



Figure 2 - RoboCrane

Below are the names for the base axes (Figure 3 - Base axes) and crane parts (Figure 5 - Crane with crane parts) which are used throughout this thesis. The measurements from joint to joint are in “Crane limbs measurements”.

3.2.1 Base design

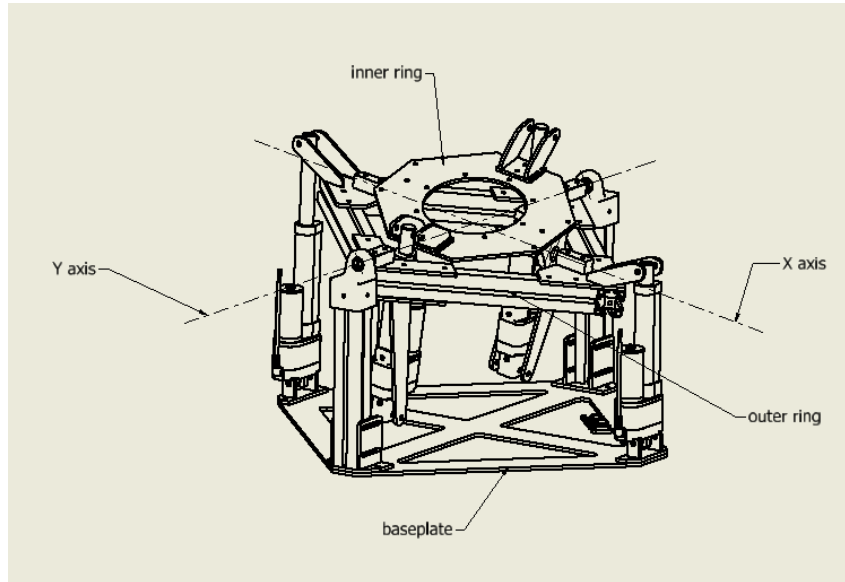


Figure 3 - Base axes

In Figure 3 - Base axes, the axes of the base are represented. We use an outer ring which moves in relation to the baseplate around an axis Y and an inner ring which moves in relation to the outer ring around the X axis. These movements can compensate for the sway and roll movements of the ship (Figure 1 - Ship movements).

3.2.2 Angled base design

The angled base compensates for the yaw of the ship (Figure 1 - Ship movements), especially since this joint needs to be stable given the whole crane relies on it. In Figure 4 - Angled base, we can see the angled base with also a corner section view.

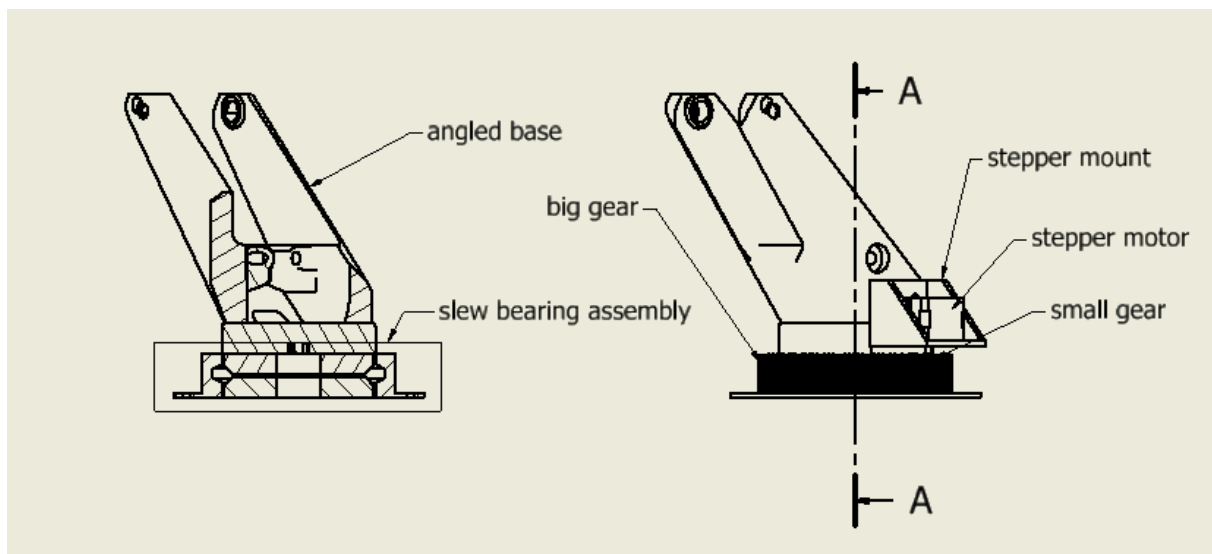


Figure 4 - Angled base

3.2.3 Crane design

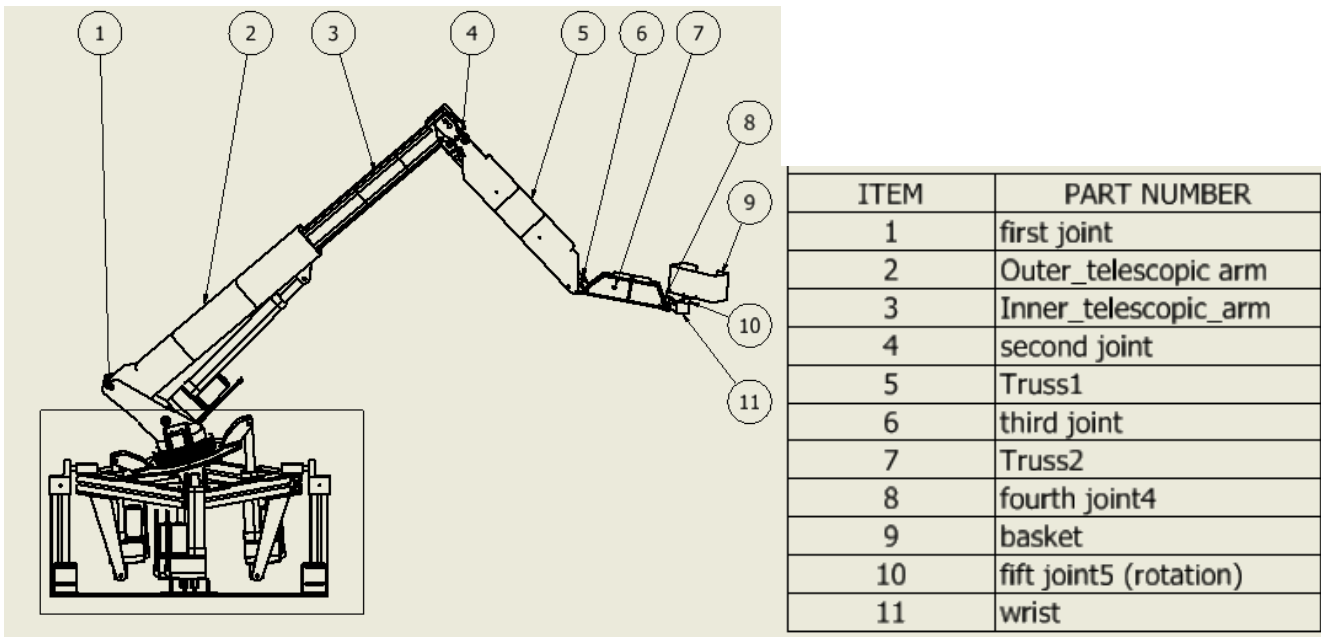


Figure 5 - Crane with crane parts

In Figure 5 - Crane with crane parts, you can see the completed crane. Number one refers to the base and rotating base. We will continue referring to different parts of this crane with these names.

3.3 Turning mechanism

3.3.1 Choice for design

During the inspection of the crane, a significant amount of play was observed at the turning mechanism. A collective decision was made to initiate research into alternative options and reconstruct this particular part of the crane.

Option one: slew bearing

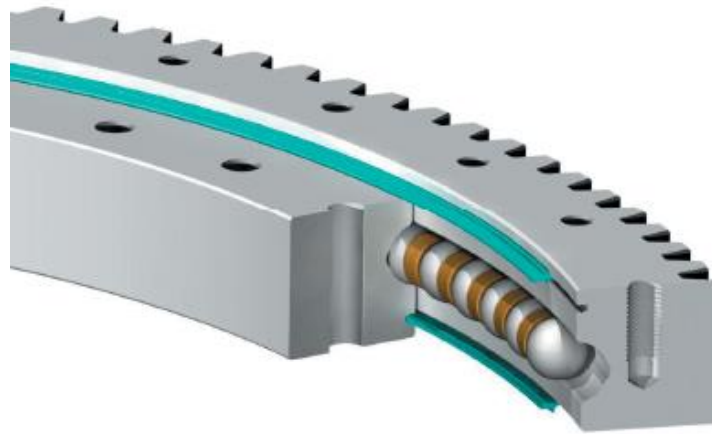


Figure 6 - Slew bearing (34)

The slew bearing is the general go-to solution for turning mechanisms of big structures. The big outer ring gives two points of contact to the steel balls in the channels and the smaller inner rings each give one point of contact. The inner plates are sandwiched together with bolts to keep the whole assembly together. The four points of contact ensure a big resistance to radial and axial forces. Reducing the price tag can be achieved by 3D printing the rings. This would increase friction and play and it will require some more design work.

Option two: radial bearing



Figure 7 - Radial bearing (35)

This design of the radial bearing places the steel balls inside two rings. It can withstand great radial forces but no axial forces. If we want to use a radial bearing we will need to buy a bearing big enough to support the axial forces. To reduce cost, we can 3D print the rings which will increase friction and play.

Option three: special assembly

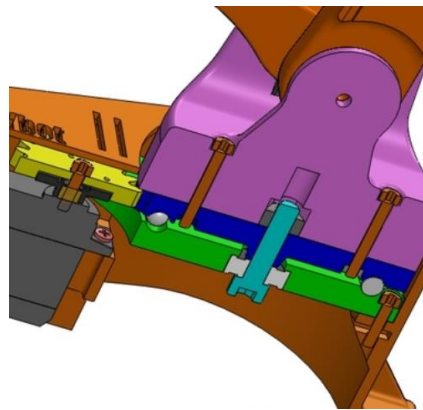


Figure 8 - Special assembly

In Figure 8 - Special assembly, you can see an example of such an assembly. This one has a small metal bearing on the bottom, a bolt to squeeze two 3D-printed plates together, and balls in between. The head of the bolt rotates freely inside the bearing. This design is more complex and needs more design work and more material to work properly.

Choice matrix

Table 1 - Choice matrix rotating base

	Slew bearing (metal)	Slew bearing (3D-printed)	Radial bearing (metal)	Radial bearing (3D-printed)	Special assembly
Price	1	5	2	5	4
tolerance	5	4	4	1	4
friction	5	4	4	2	4
complexity	5	4	5	5	2
Own opinion	3	5	2	2	3
total	19	22	17	15	17

Choice matrix Table 1, shows us that the 3D-printed slew bearing is the best option. The metal slew bearing gives the best movement but comes with a heavy price tag. Therefore we will 3D print the outer and inner rings and use metal balls in these rings. 3D printing the balls of the bearing would not be convenient since as 3D printing would not be dimensionally accurate, and they would not roll as smoothly as metal ones. This widely adopted design is commonly utilized in turning mechanisms for lifting solutions, establishing its reliability and effectiveness.

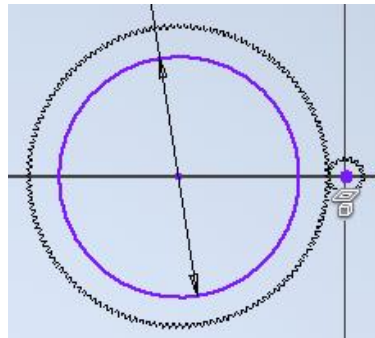


Figure 9 - Generated gears (5)

Design choices

The first step is including the outside gear in the design which is generated using an online tool to get the correct pressure angle (5). This can be seen in Figure 9 - Generated gears . The inside diameter is derived from the existing part: “angled base”, so as to not further change the design. With these measurements and knowing there are at least 17 teeth needed for the smallest gear (to not allow undercut gear tooth). The tooth count of the outer gear is calculated. For the big gear, the corresponding teeth number is 145. The holding torque of the stepper motor is 0,36 Nm. With these numbers, the calculated torque is: $0,36 \cdot 145 / 17 = 3,07 \text{ Nm}$. To get the force exerted at the teeth we multiply this by the pitch radius of the big gear: $3,07 \text{ Nm} \cdot 0,072 \text{ m} = 0,221 \text{ N}$. This works because this assembly is sitting on the base which keeps this part level. Theoretically, the only force that this stepper motor needs to overcome is friction from the slew bearing. The tolerances of the design were chosen using 3D printing tolerances obtained from an online article (6).

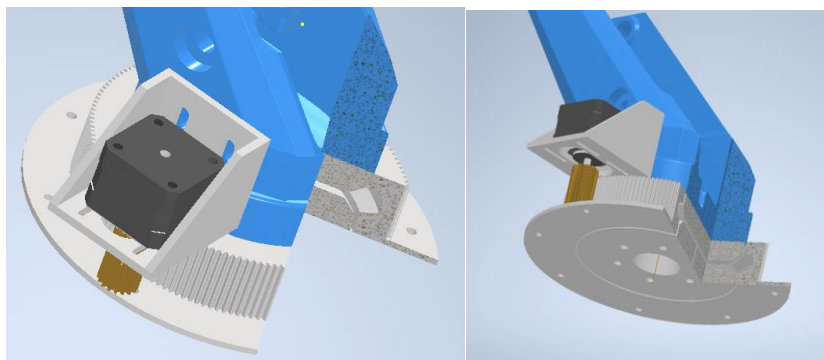


Figure 10 - Slew bearing angled base

Next up is designing the rest of this assembly represented in Figure 10. With the correct gear shape the sketch has been modified. The processing time needed when altering the involute sketch is remarkable. After designing the gear a slot is cut out as part of the bearing. These dimensions continue to the inner plates. A centre hole is added for future cables. The next task involves the motor mount, which is intended to be adjustable, enabling the assembly to fine-tune the play and height of the smaller gear in relation to the larger gear. Once all the parts were manufactured, the tolerance of the bearing was measured, and the inner plates were repeatedly 3D-printed. As a result, there is now virtually no play, allowing the assembly to rotate freely.

3.3.2 Realisation

For 3D Printing the gears (Figure 12 - Big gear slicer settings) a Prusa mk3 printer (7) is used. The printer has a printing volume of 250mm x 210mm x 210mm and has a nozzle that can reach temperatures up to 300°C which makes it capable of printing materials including ABS, PETG, PLA, etc. For this project, PLA will be used. PLA is strong enough, it is also one of the easiest materials to work with. The previous experiences we have with CURA (8) (3D print slicer software), made us decide to slice our 3D prints with CURA. We also have access to an adventurer 4 3D printer from FlashForge (9) with a build volume of 200mm x 220mm x 250mm. For this printer, a custom printer profile in CURA is required. A quick Google search led to the website of FlashForge (9). They give a step-by-step setup to use their printers with other 3D printing slices. By only using CURA, the learning curve associated with using 3 different slicer software is reduced. The profile used for the adventurer 4 is in Figure 11 - 3D printer profile below. Note that there is a need to use a bit of startup G-code and end G-code. This is a specific code for the printer. This code initializes the printer, sets the correct position of the printhead with respect to the print bed, and safely shuts down the printer when a print is finished. These settings need to be corrected. If not, they could result in bad prints or at worst, the printer breaking itself. For example, because the machine thinks the build plate is located 20cm lower and thus the printhead assembly would collide with the print bed.

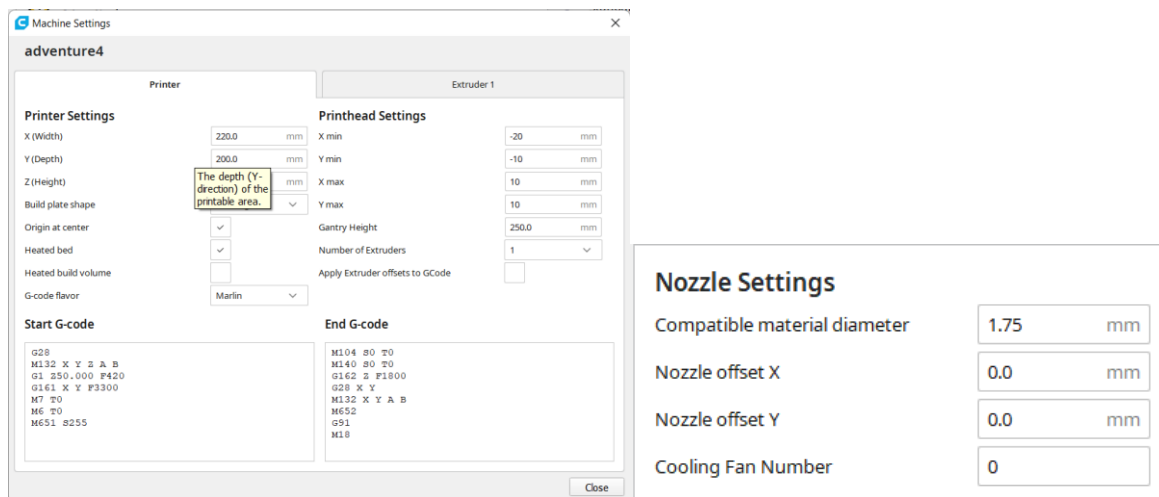


Figure 11 - 3D printer profile

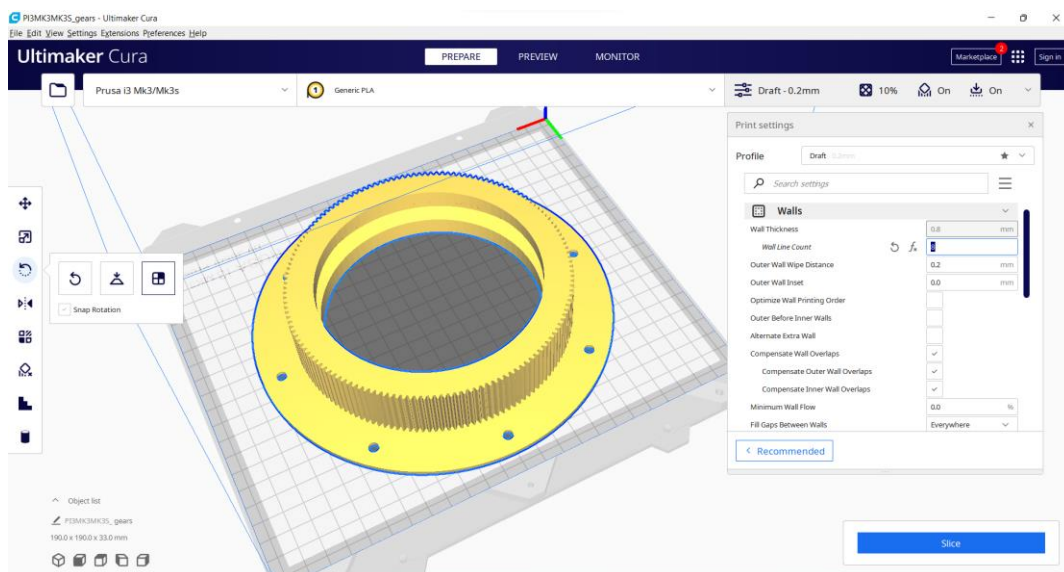


Figure 12 - Big gear slicer settings

3.3.3 Gears

The gears are printed with increased wall line counts, this influences the wall thickness which in turn greatly increases the strength of the part and makes sure there will not be any teeth breaking off. The inside discs are printed with the same settings. For the angled base, an adjustment was made. To make sure the long braces will not break next settings were altered: increased temperature (for better layer adhesion), increase infill density, infill line count, and wall line count. The end result, represented in Figure 13 - 3D printing the toothed ring with end result, meets our standards as everything moves with very little play and friction.

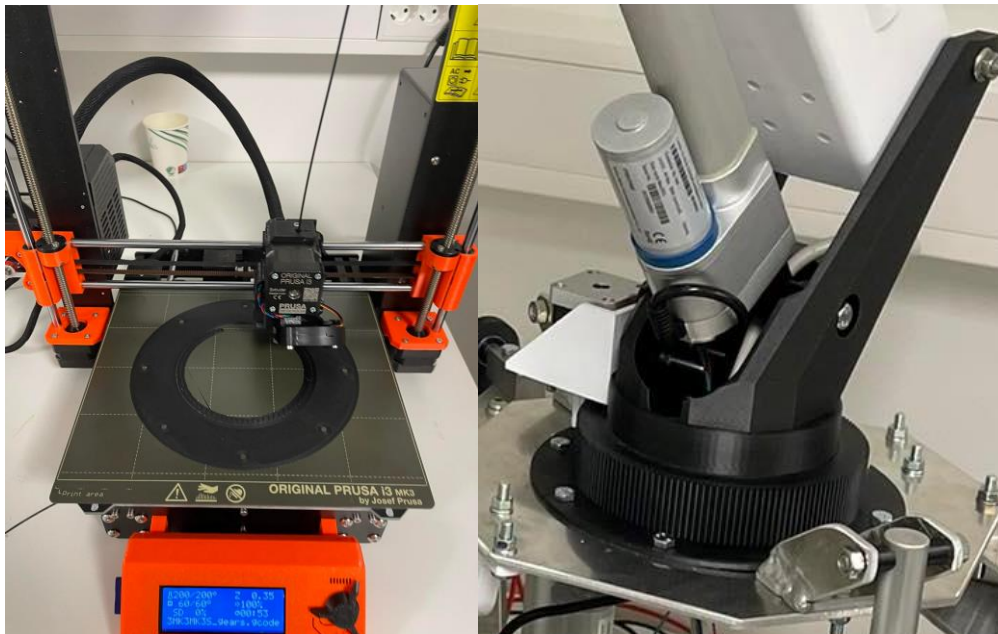


Figure 13 - 3D printing the toothed ring with end result

3.4 Base design

After taking a closer look, an excessive amount of play is observed in the base. Upon closer inspection, we discover that the base design is flawed and has too many ball joints, which gives too many degrees of freedom. The design of our scale model also does not match the design from Palfinger. We decide to start designing our own, more rigid base platform. Our preferred design is a similar design as the big crane from Palfinger. 3D-printed plastics in this part of the crane would not be efficient, thus the decision was made to use aluminium extrusions with aluminium reinforcing corners and welded subassemblies. This decision required welding aluminium which entails a challenge on its own. The design of the base starts with the connection of the rotation base. This gives some measurements to start with. The next step is measuring all the other actuators (Figure 14 - Linear actuator), these have a stroke length of 100 millimetres and a retracted length of 243 millimetres. All other measurements can be found in the corresponding datasheet (10).

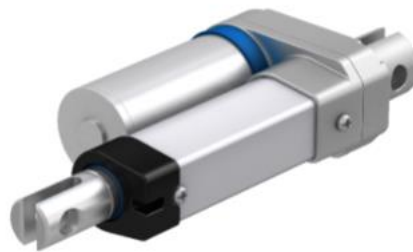


Figure 14 - Linear actuator

While designing, the choice was made to utilize all four actuators simultaneously, resulting in a closer resemblance to a real crane. The new design incorporates a platform-mounted assembly that facilitates the movement of an outer ring (left side Figure 15 - Outer and inner ring), subsequently rotating the inner ring (Right side Figure 15 - Outer and inner ring). This approach ensures that each joint possesses only one degree of freedom. The design of the base is progressing while taking into account the dimensions of the actuators. Throughout this process, each component is placed within a large assembly to enhance visualization and simplify collision-free design. The extrusions utilized are obtained from Alu flex (11). The CAD file was conveniently acquired from their online CAD filer and seamlessly imported into the drawing.

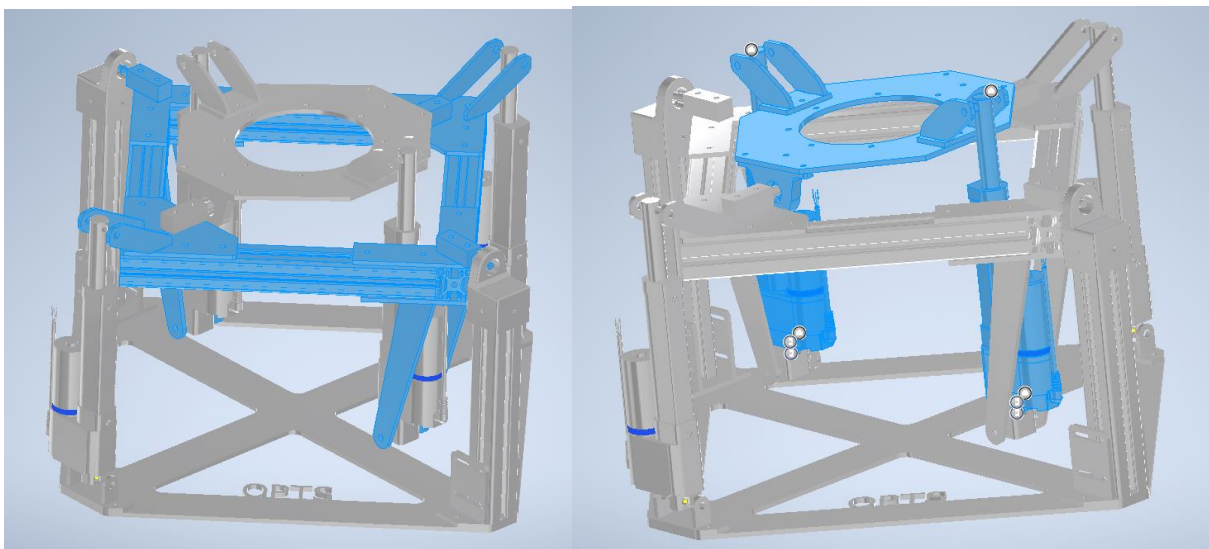


Figure 15 - Outer and inner ring

Initial standards (left side Figure 16 - Standards and actuator mount) were established at the beginning of the design process. Tabs (right side Figure 16 - Standards and actuator mount) were incorporated to ensure proper alignment with other components. These tabs are all made with the same dimensions except the “tail”. The “tail”, with a measurement of 20 millimetres, keeps linking the subassemblies with the corresponding actuator mount easier. They also give a clear physical aid for aligning the assemblies for welding. Their dimensions can be seen in Figure 16 - Standards and actuator mount.

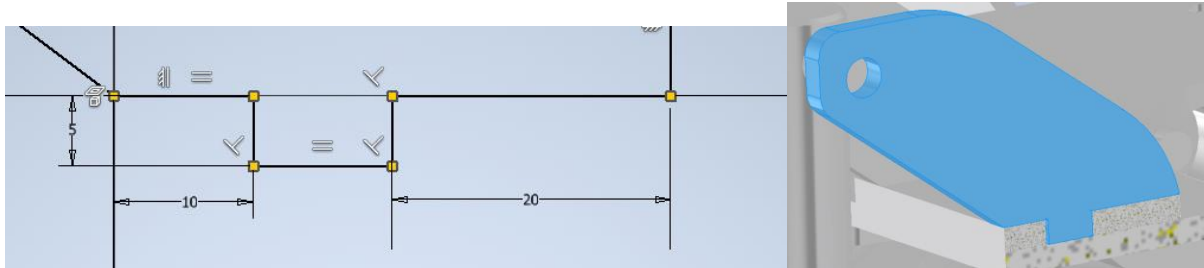


Figure 16 - Standards and actuator mount

The options for how the axis will support the base are:

- Bolts, which would cause the introduction of a lot of friction;
- 3D printing bearings, plastic bearings which would introduce a lot of play and friction;
- 3D printing the axis and hole, which would cause a lot of friction;
- Using a metal axis and standard metal bearings.

The last option was the most preferable. The strength of steel is needed in this part of the base without a lot of play. Therefore all the other options would not work. The axis (Figure 17 - Axis) is designed with a square body to fasten to the rest of the assembly. The axis also includes a shoulder that presses against the bearing and ensures no axial play. A 3D-printed pillow block (Figure 18 - Pillow block) is made to press fit the bearings into. To make sure it is a press fit, the hole is designed on exactly the same size as the bearing’s outer diameter. The axis is mounted in a slot to give the assembly adjustability and reduce play, allowing us to jam the bearing and therefore the assembly tightens by clamping both axes closer together. The other axis is not mounted in a slot but the adjustability comes from the mount at the bottom which does allow movement. To try and make it a little easier to program, the x- and y-axis are placed at the same height. This gives a point in the centre around which everything will rotate. This is an easy thing to do in this part of the project but reduces complexity later on.

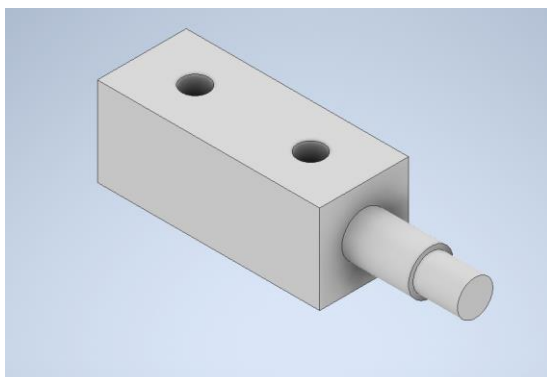


Figure 17 - Axis

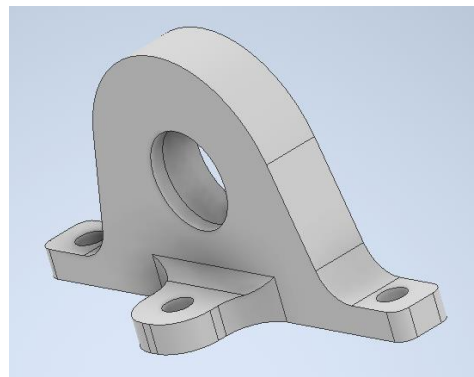


Figure 18 - Pillow block

While asking for some assistance with plasma cutting the design, we got some good advice. By making welded subassembly there will not be the chance of warping the main big plate which needs to be dimensionally accurate. The designs are edited to incorporate subassemblies (Figure 19) with bolted connections. This takes longer than expected because of the need to edit all the mounts, recalculate their lengths and remake the assembly. The recalculating consists of subtracting the length of tabs from the thickness of the aluminium plate which is 5mm. Before plasma cutting, everything is placed back into the assembly and checked for collisions and if it is possible to make/assemble everything.

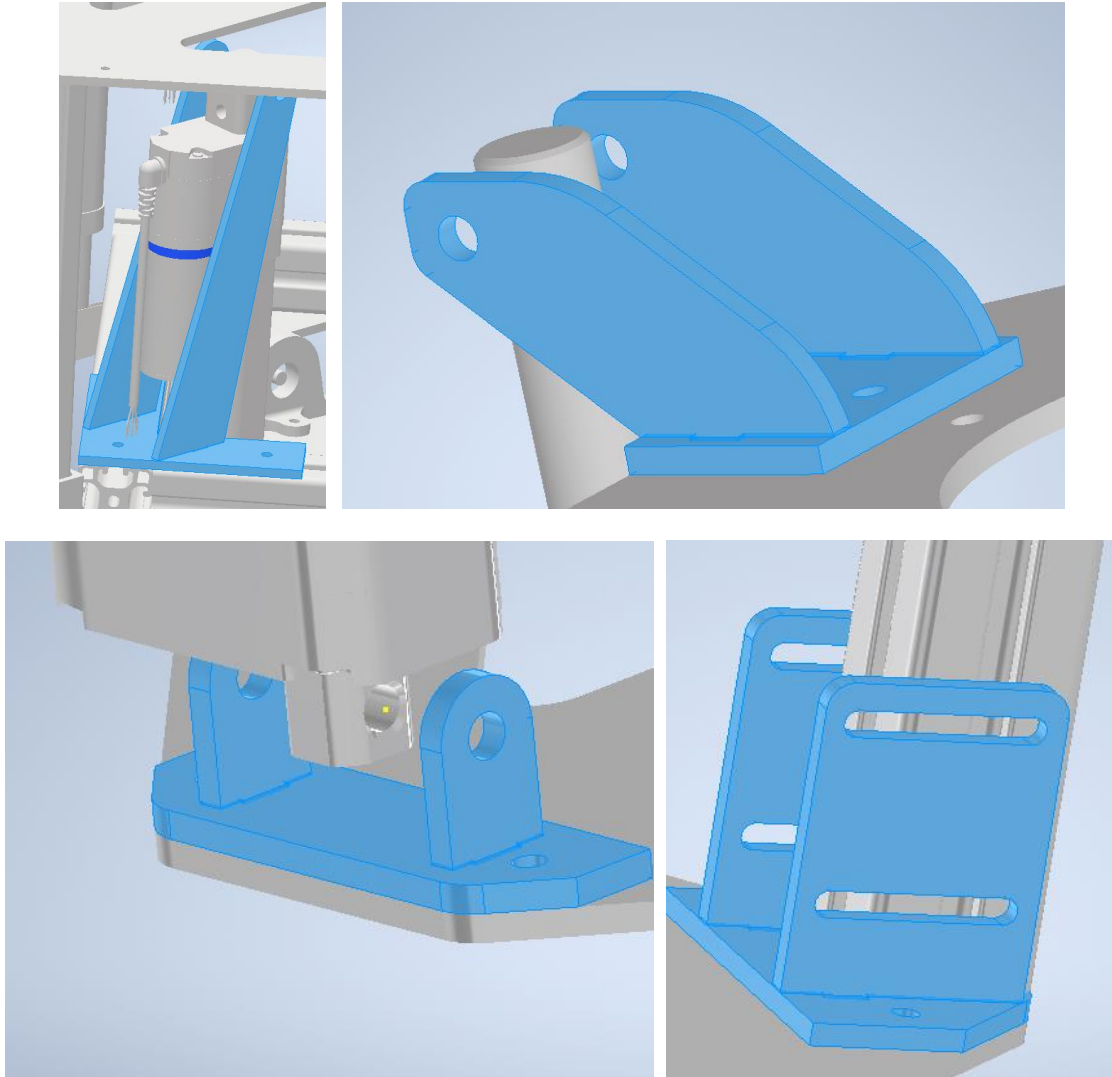
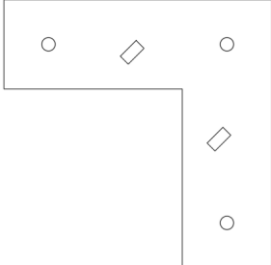


Figure 19 - Welded assemblies

Realisation

Cutting parts.

Plasma cutting the parts required a DXF file format (left side Figure 20). After exporting all needed faces in Inventor to a DXF file format, the number of parts was added behind each part name. This made ordering all parts in the Plasma cutter software a lot easier and clearer. Due to the structure, we only had to cut once. This can be seen in Figure 20.



actuator mount inside corner x2.dxf	13/03/2023 15:07	AutoCAD Drawing...	263 kB
actuator welded mount long x 4.dxf	13/03/2023 15:12	AutoCAD Drawing...	262 kB
actuator welded mount outside x4.dxf	13/03/2023 15:15	AutoCAD Drawing...	263 kB
actuator welded mount x4.dxf	13/03/2023 15:10	AutoCAD Drawing...	262 kB
base post sub assembly x2.dxf	14/03/2023 11:07	AutoCAD Drawing...	264 kB
base post support x4.dxf	13/03/2023 15:13	AutoCAD Drawing...	263 kB
baseplate x1.dxf	14/03/2023 11:27	AutoCAD Drawing...	297 kB
centerplate x1.dxf	14/03/2023 10:33	AutoCAD Drawing...	264 kB
corner axis+outside mount x2.dxf	14/03/2023 10:26	AutoCAD Drawing...	265 kB
outside bearing corner x2.dxf	13/03/2023 15:11	AutoCAD Drawing...	262 kB
outside actuator base mount x4.dxf	13/03/2023 15:14	AutoCAD Drawing...	262 kB
pivot base plate x2.dxf	14/03/2023 11:28	AutoCAD Drawing...	263 kB
welded mount inside plate x2.dxf	14/03/2023 10:34	AutoCAD Drawing...	263 kB

Figure 20 - Plasma DXF file

The same plate thickness is used throughout the design. This reduces complexity and leads to easier designing. The 5mm thick aluminium plate is cut using the plasma cutter at HVL. It is an old machine therefore we assume that the cuts are not that accurate. The machine also centre punches the places where we need to drill holes. After restarting the machine a few times, all parts were cut (Figure 21).

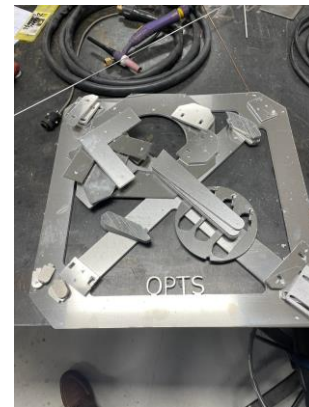


Figure 21 - Realisation with plasma cutter

Cleaning parts

The next step is cleaning all the parts. For this process, a belt grinder and file were used. The belt grinder speeds up the process, but some small pieces still need to be cleaned up by hand. When all the parts are cleaned the holes get drilled. After everything is cleaned up and drilled, we prepare for welding.

Welding

After asking about welding the aluminium, we got the chance to do it ourselves. Without hesitation, we took this chance to learn how to TIG weld aluminium. TIG refers to the tungsten electrode used in the torch and the inert shielding gas used that protects the weld. Unlike welding metal, you need a rounded electrode tip instead of a sharp one. The material also needs a wire brush before starting the weld, since this process gets rid of the tough aluminium oxide on the surface and makes the process easier. We got the machine set up with the correct current, learned about the TIG welding technique with the filler rod, and started practicing. Welding aluminium turned out to be more difficult than welding steel. The conductivity of aluminium makes sure the whole piece heats up and once everything is hot you, can drop the current or speed up. After some practice (Figure 23 - First welds), we got the hang of it and started with the real deal. To ensure alignment and no warping two steel blocks were used (one of 40 millimetres and one of 50 millimetres). These act as spacers and give a way to fix the assembly together without warping or moving.



Figure 23 - First welds



Figure 22 - Welding process

Assembling

Assembling everything starts with fixing the subassemblies to their corresponding bigger plate (Figure 24 - Base with outer ring). M6 bolts are used to accomplish this but are not fully tightened. After this, the aluminium extrusions are assembled in a rectangle with the correct corner pieces (Figure 25 - Base with welded assembly). The aluminium extrusions used are the 40 x 40 millimetre profile 8 from Alu flex (11). There were small pieces of aluminium used from different projects. There was enough for our project, this way we did not need to order any new extrusions. Using the slots from the extrusions to fasten everything together makes for a strong and ridged platform. T-slot nuts and M8 bolts are used in this part of the assembly.

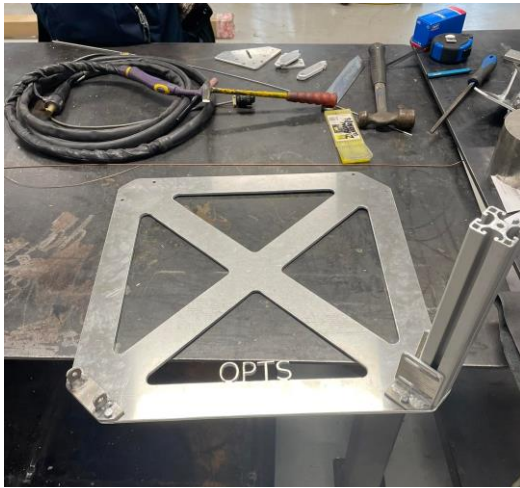


Figure 24 - Base with outer ring

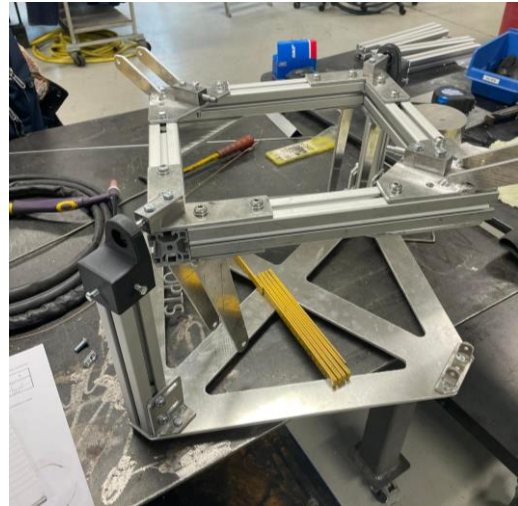


Figure 25 - Base with welded assembly

Next up is the centrepiece (Figure 26 - Centrepiece). This piece connects the rest of the crane with the base (Figure 27 - Crane with base). The base was assembled using the standard bolt sizes available to us and a few cut to size (again using M6 bolts). The tolerance of the pillow block and bearing is just right to be pressed together by hand which ensures a nice friction fit. When fitting the axes and bearings the axes are not up to spec and do not fit. When asked, a lab engineer used a belt grinder to remove excess material. While doing so, there was a small bevel introduced in the axis which actually increased the tightness. The axes are now not only clamped into each other but also wedged into the bearings. After installing the actuators, the space next to each joint was measured. With this information the spacers are made, which fixes the actuators in place and limits the side-to-side movement.

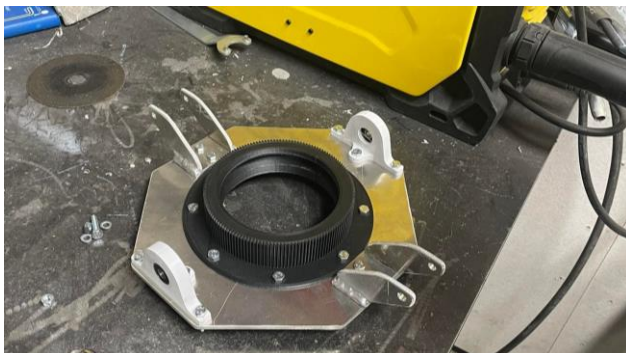


Figure 26 - Centrepiece

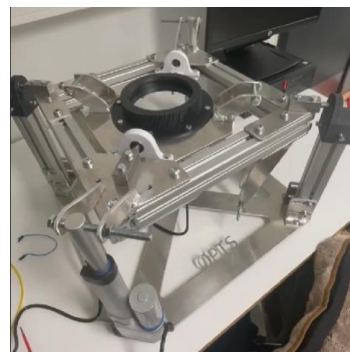


Figure 27 - Crane with base

3.5 Remaking truss 2

Upon further inspection, additional problems arose. The first of which was truss 2 as this 3D-printed part was broken due to the scaling process and print orientation. The material around the pivot point was not strong enough. This can be fixed with a few alterations:

1. Editing the design with more material around the pivot point. This is the most obvious solution, putting more material where the break happened. This option is limited to the other parts of the joint, which does not allow a lot of extra mass.
2. Printing the part in a horizontal configuration. This solution will result in better print quality and a more optimal print efficiency. The need for a good layer adhesion will be less important as the whole outline of the wall will be printed in one layer. Printing the part in this orientation will increase the support desired.
3. Using a different material. ABS and PETG are two materials that have better mechanical properties. The downside will be the 3D printer itself, since the Prusa printer we have access to, is technically capable of printing with these materials but will struggle to achieve a good result. We could outsource this part to be made somewhere else or use a lot of time getting the correct settings for this printer with these materials.

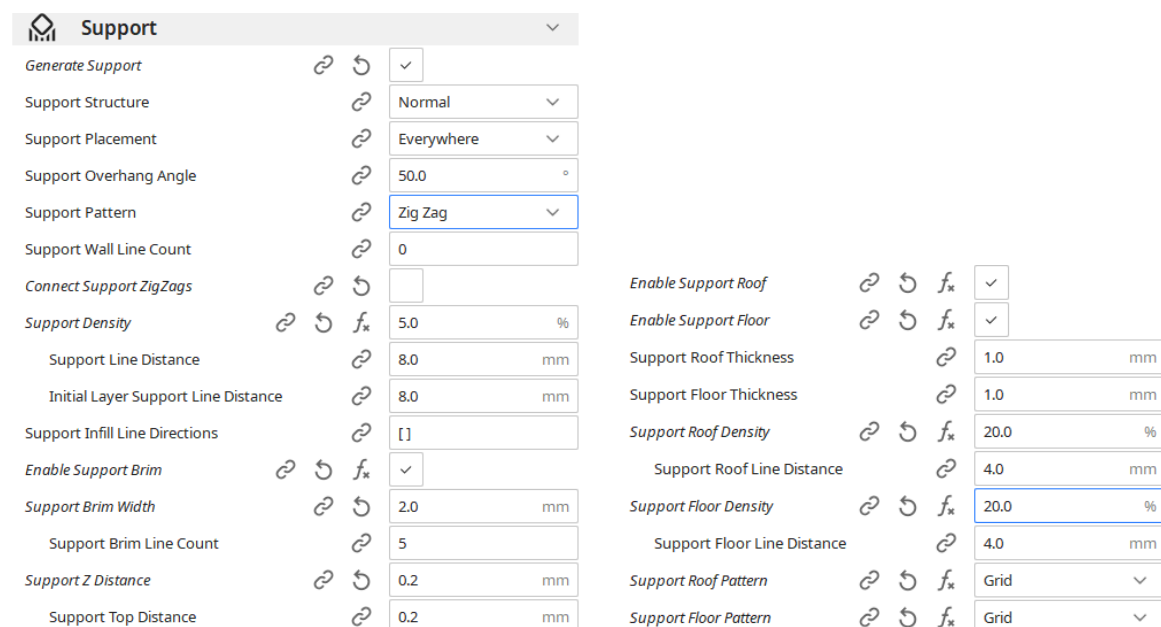


Figure 28 - Cura settings

In the end, the part was 3D-printed with PLA in the horizontal configuration with added material around the pivot area. A support roof and base were used with 5% support density to speed up the cleaning process and support material used. The wall line count and infill were also altered: infill was increased by 10% and wall line count was increased from two to six, as this change gives more material at the very bottom of the part where compression is the highest, and at the very top where tension will be the highest. The rest of the settings are strong enough for this part. The added material in the top and bottom results in a structure with a mass distribution close to an I-beam. This will be more optimal considering strength and weight in one direction. Changed support settings can be seen in Figure 28 - Cura settings.

3.6 Remaking the second joint

Joint 2 (bottom of Figure 29 - Second joint) also has a lot of play. After closer inspection, approximately 50° of movement was observed. When the crane is loaded, the joint will normally be at the lowest point due to gravity. The problem is more prevalent when side-loading this joint. The up-and-down movement can be plotted with the pot meter input. There are a few options for solving this problem:

1. Making a new joint design. This would require us to redesign the joint from scratch, which would result in more work and a lot of PLA waste.
2. Fully adjusting the old design and reprinting. This would mean adjusting hole and pin tolerances and reprinting the second joint and truss2. The needed design time would be less compared to adjusting the old one. The downside would be the way previous designs are made and the used materials.
3. Drilling out the holes and printing bigger pins where possible.
4. Remodelling the truss connections and plastic welding the pins to the other connection. Plastic welding would result in a strong connection. The downside would be that we cannot disassemble the joint.

The truss connection is 3D-printed with smaller hole sizes (top of Figure 29 - Second joint). This way the pins can be reused. 3D printing only the connections will not take a long time and will only use a little bit of material. In the top holes of the inner boom, tip bushings are used to adjust the hole size. They press fit in the holes and make for a play-free connection. They are printed with increased bottom/top thickness and wall line count to increase the strength of these small components.

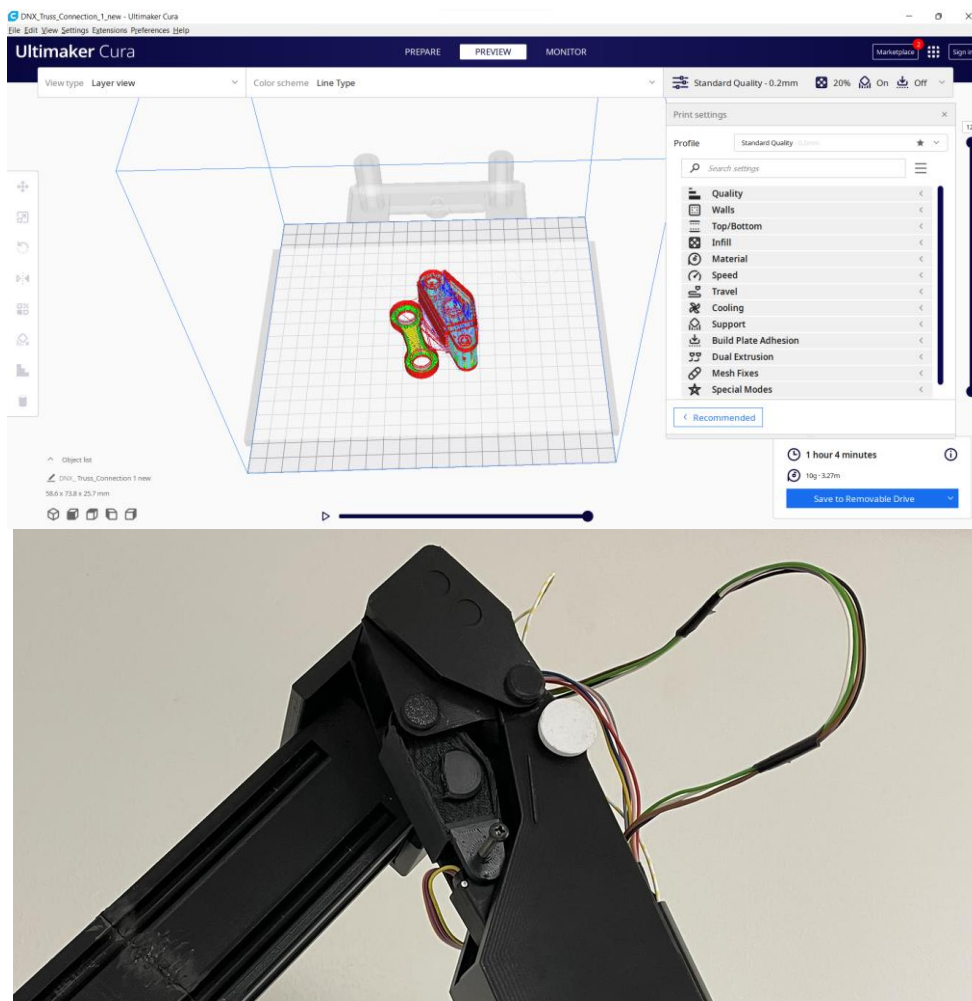


Figure 29 - Second joint

3.7 New truss 2

Previous designs did not use the measurements of the actuators correctly. Therefore the range of motion of the third and fourth joints was limited. The different options for solving these problems are:

1. Redesigning both joints from scratch.
2. Drilling new attachment holes inside truss 1 and truss 2, which is not preferable. We need the exact measurements of the crane, in order to use these in the program codes of the crane.
3. Moving the joint connections. This will allow only a small amount of correction.

Both problems are fixed by altering truss 2. To fix the motion of the third joint the connection hole shown on the left side of Figure 31 was moved down with the same distance from the pivot point. This moves the joint $43,4^\circ$ from the original position. A cut-out was made at the bottom (middle Figure 30 - New truss 2) to allow truss connection two enough space to move freely. The third joint is fixed by extending the entire truss (left image of Figure 30 - New truss 2). This solution was not easy, because of the way the old design was made, the old design just scales down the original drawings from Palfinger and adds some shapes. To extend truss 2, the truss is cut in half, one part moved and then a new extrusion was made to fill in the gap (using the function “extend” did not work with this design).



Figure 30 - New truss 2

3.8 New wrist, fifth joint and basket

Originally the fifth joint was moved with a stepper motor. When inspecting the design, there were only technical drawings and no electrical components or code for it. Making the fifth joint with the small stepper motor would introduce a lot of weight to the furthest joint of the crane. An additional encoder and driver would also be needed, which would increase the cost and would also increase the complexity of the code. A preferable option would be redesigning this joint and using a small sg90 micro servo. These servo motors can be controlled straight from an Arduino and only weighs 9 grams, given this will be the final joint, it needs to be lightweight. The basket (left of Figure 31 - Basket and fifth joint) will be placed straight on top of the servo's axis and printed with a special profile. The profile used is 'spiralize outer contour'. This setting spiralizes the print. Spiralizing the print means it is printed in one continuous line. This way the print will be light and strong. The bottom thickness is also increased. With these settings, the part will be only 40 grams and will be printed in only 3 hours and 16 minutes (right side Figure 31 - Basket and fifth joint).

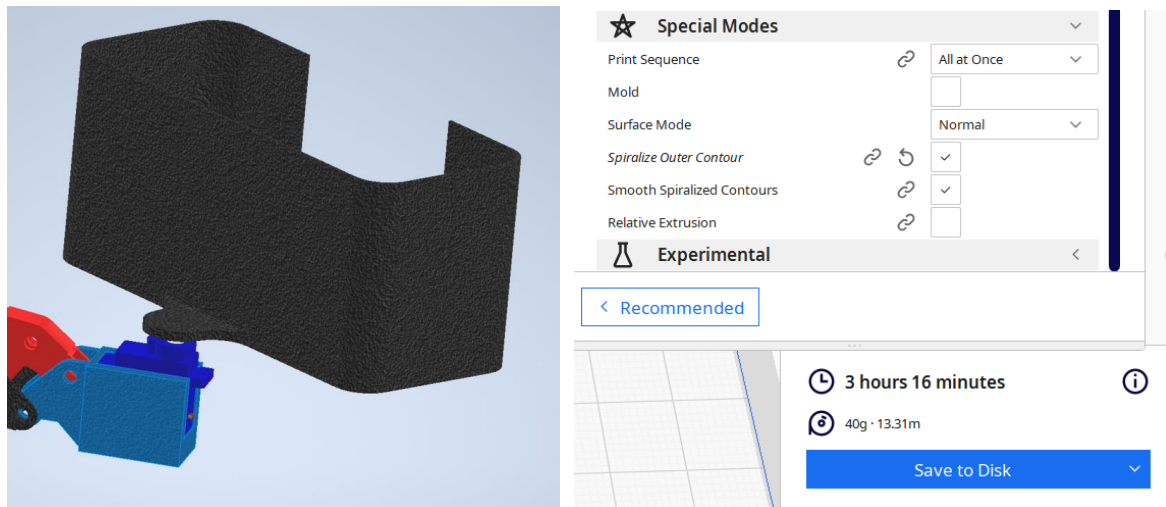


Figure 31 - Basket and fifth joint

3.9 Telescopic arm

The telescopic arm did not work due to friction, bad tolerances, and an overall bad design. The design had bad tolerances and a bad finish on the 3D-printed parts. The inner telescopic arm bends and is not straight while the outer telescopic arm is. There is also no practical way to assemble the stepper motor.

4. Electrical aspect

All datasheets can be found on our github page (12)

4.1 Components

The results from the Bachelor's thesis from the previous year (3) show us a lot of components that we can use. To use these components, we have to do research about them and get to know them thoroughly. This is what we did in our first weeks of research. Here you will see the components that we received from the previous year together with the components that we ordered this year. Sometimes we use other components than those that were initially at our disposal. There are different reasons for this, which will all be explained later on in this research.

4.1.1 Gyroscope

The gyroscope gives information about the angle and acceleration of the boat at a specific time. The information from the gyroscope is important. First, the gyroscope gives information about how many degrees the crane has to compensate for the tilt of the ship (roll, pitch, and yaw, Figure 1 - Ship movements). Secondly, it sends information about how much the crane has to go up and down (heave). In February, gyroscope type 'BNO055' (left side Figure 32 - Gyroscopes BNO055 and LSM9DS1) from the previous year was been used, but it was already a few years old and did not work properly. After testing, the conclusion was that the magnetometer and the accelerometer on the z-axis worked, but the x- and y-axis accelerometers did not work. With this issue, a new gyroscope was ordered: 'LSM9DS1' on the right side Figure 32 - Gyroscopes BNO055 and LSM9DS1. While testing the new gyroscope, it was clear that there were still some flaws. Thankfully the accelerometers worked, but there were still some problems with the magnetometer. Another problem is accuracy. If the gyroscope is laying still, the PLC gives values from -10° to $+10^{\circ}$. This leads to a deviation of 20° . The transformation of the data from the Arduino to the PLC is probably the cause. Due to lack of time, this issue unfortunately has not been resolved.

In Figure 33 - Gyroscope signal you can see the graph that shows the values of each direction (x, y, z) of the gyroscope 'BNO055'. At that moment the gyroscope was tilted in each direction, but only the z-axis reacted. The red and blue lines at the bottom are the x- and y-axis. There is a little offset between those two. The dipping of the signal from the z-axis to the ground is also due to the fact that the gyroscope is not working properly anymore.

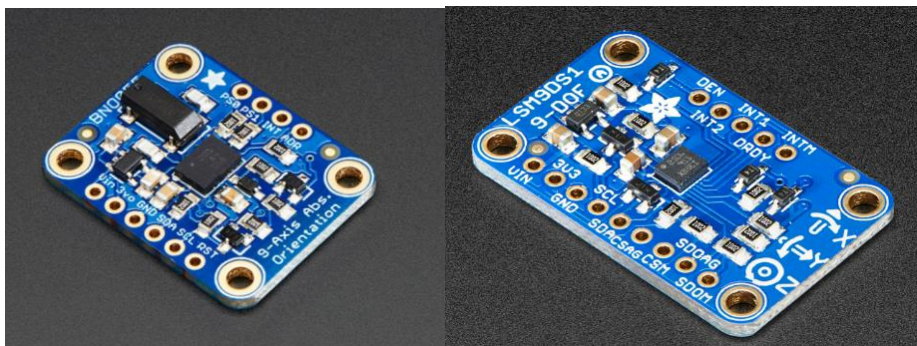


Figure 32 - Gyroscopes BNO055 and LSM9DS1

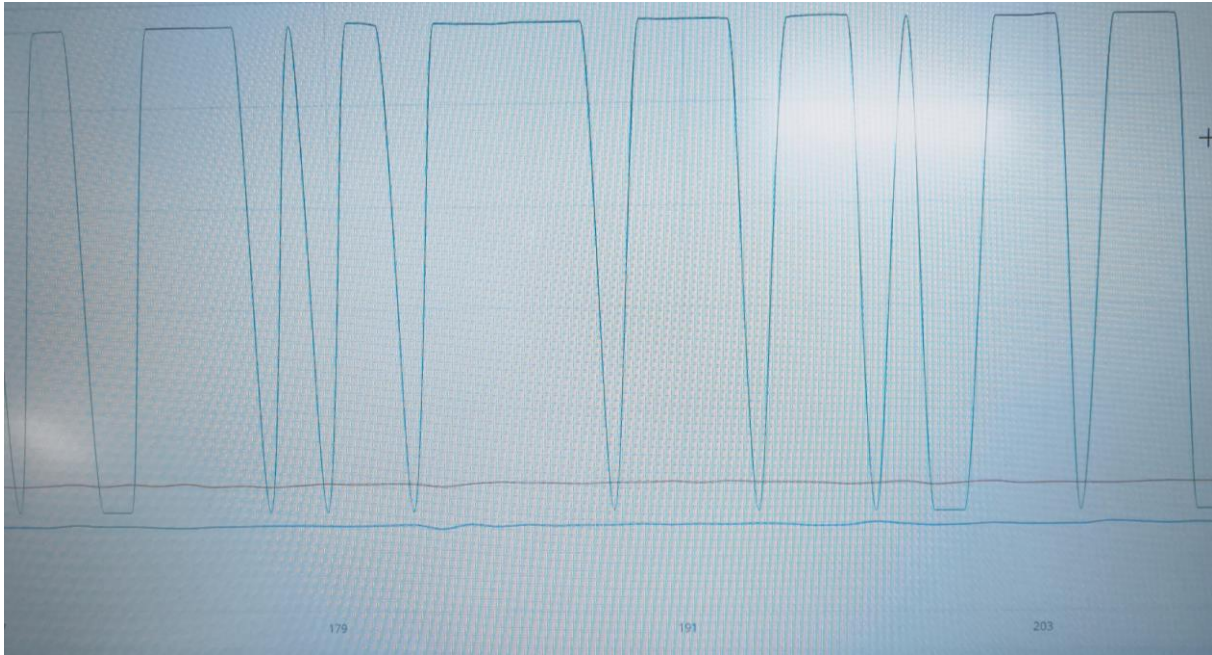


Figure 33 - Gyroscope signal

4.1.2 Linear actuator base (Ewellix)

To tilt the base, we need actuators. These actuators are very sturdy and stable and therefore perfectly fit for this job. Inside is a powerful DC motor with an absolute analog output, which is used to find the position of the motor at all times. You can compare the absolute analog output with the output that you get from a potentiometer.

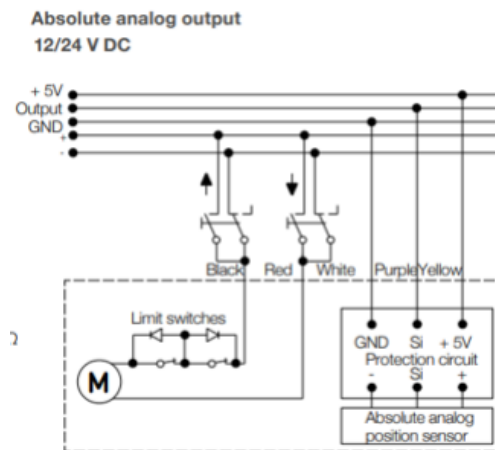


Figure 34 - Wiring diagram absolute analog output (10)

4.1.3 H-bridge

To move the linear actuators, the DC motors need to be controlled with an H-bridge. An H-bridge is an electronic component that can switch the polarity of the applied voltage. In this thesis, there are two types of H-bridges. The first one is the HW-039. It is built for 24VDC and it is used for the actuators of the base and the first joint. The other type is the L298N, which is a dual H-bridge. This means that it can control two motors at the same time. This dual H-bridge is built for 12VDC and is used for the 2nd, 3rd, and 4th joints. These linear actuators need 12VDC. An important thing to know is that there is a maximum PWM frequency that the H-bridges can handle. The maximum PWM frequency can be looked up in the data sheets (13) (14). The two different types of H-bridges do not work totally the same. HW-039 can send different PWM signals for the in-and-out movement of the actuators. The L298N H-Bridge

has to change its PWM signal in order to accomplish the same result. The HW-039 can work with the same program as the L298N if we connect the PWM signals to one output on the Arduino. The L298N cannot work with dual PWM output.

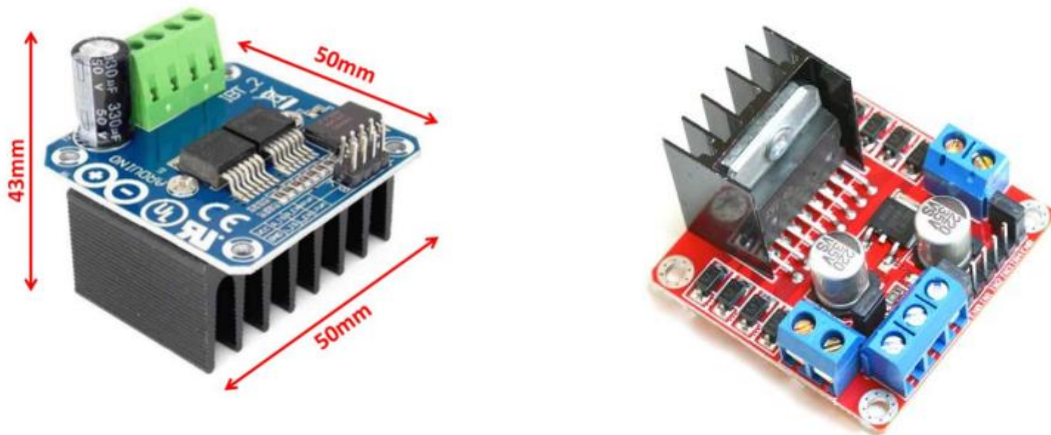


Figure 35 - H-Bridge HW-039 (13) and L298N (15)

4.1.4 Stepper motor with driver

In this project, we need two stepper motors. One will be used for the telescopic movement of the arm and the other one is for the rotation of the crane around the 'heave axis' from Figure 1 - Ship movements. The drivers that are used are the TB6600 and the geckodrive G201X. The driver TB6600 is used because it was already at our disposal. The Geckodrive was ordered later on because the TB6600 could not be ordered.



Figure 36 - Stepper motor (16)

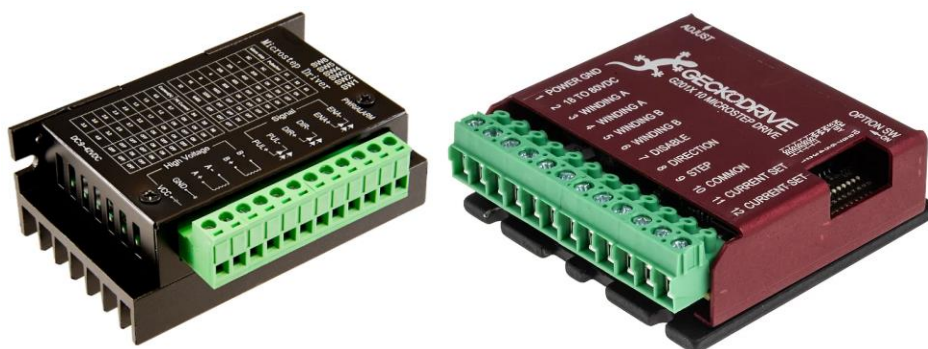


Figure 37 - Driver, TB 6600 + Geckodrive G201X38 (17) (18)

4.1.5 ADAC-Click module

The ADAC-Click module is made to convert analog signals to digital signals or the other way around. With the ADAC Click module, we can convert the data of the gyroscope to an analog signal and connect it to an input of the PLC. This way, we can read the data on the PLC. Its job is to convert the signal from I²C to analog, hence (A)DAC. This stands for Digital to Analog Converter. In the end, we could just connect the Arduino directly to the PLC and we did not need this ADAC module. The calculations show that using the ADAC is faster.



Figure 39 - ADAC Click (19)

Calculations:

Is the ADAC click really the best solution to convert the data from I²C to an analog signal? We did some tests with the ADAC click and saw that we always got a dip back to ground, the PLC took averages from these values, making it incorrect. This is an issue since the values were corrupted.

Resolution

- The gyroscope has 32 slots of 16-bit data for each of the gyroscope's three output channels (yaw, pitch, and roll). The ADAC Click gives an analog signal that has a resolution of 12 bits. The PLC can read an analog value of 16-bits. This makes the ADAC Click the 'bottleneck' for the resolution. The 12-bit is still better than the 8-bit of the 'AnalogWrite' function embedded in the Arduino.

Speed

- The function 'AnalogWrite' needs 3.4 milliseconds. That is about 290 times per second. (20)
- I²C can send 100k bits/s, meaning it can send 1 bit about 100 000 times per second. However, take into account that the signal has to go through the ADAC click now. The ADAC click has a conversion time of 2 microseconds. This means that it takes 2 microseconds to transform a digital signal from the I²C to a 12-bit analog signal. So the gyroscope sends its 16-bit data to the ADAC and then it converts the signal with 2 microseconds latency.

First get the number of packages of data it can send without delay:

$$\frac{100k \text{ bits/s}}{16 \text{ bit}} = \frac{6250}{s}$$

This is how many times the PLC could theoretically get a value from the gyroscope per second, but there is also a two microsecond latency after each time the ADAC click converts data. Then you get:

$$0.000002 \text{ s} * 6250 = 0.0125s.$$

This is the maximum latency of the ADAC click per second.

The total latency is then:

$$\frac{6250}{1.0125} = \frac{6172}{s}$$

This method is faster than the AnalogWrite function. It comes down to 6172 times per second. In comparison with the AnalogWrite, 290 times per second. (21).

Conclusion:

The ADAC click is 21,3 times faster and yet we choose not to use it.

Do we really need so much data? No, we do not.

- 1) It is a component less to worry about for the wiring and for possible failures.
- 2) The ADAC click gives a high signal but goes back to ground after the signal because the integrated reference is turned off by default. The PLC is fast enough to keep up and so it takes the average of the signal. It is certain that you can work around this, but due to lack of time we could not do it.

4.1.6 Power supplies

In total, we have three power supplies. The power supplies convert the 230V to voltages that we need for the electronics. We have:

- 5 VDC for the control circuit and communication with the PLC;
- 12 VDC for the linear actuators of the crane from Actuonix;
- 24 VDC for the big linear actuators of the base and the big arm (first joint) of the crane. It is also used for the drivers of the stepper motors.



Figure 40 - 24V, 12V and 5V power supply (22) (23) (24)

Table 2 - Power supplies

Power supply input(V)	Power supply output(V)	Current(A)	Nominal Power(W)
230V	5V	3	15
230V	12V	5	60
230V	24V	10.42(or 16.7 with fan)	250(or 400 with fan)

4.1.7 Linear actuators crane (Actuonix)

For tilting the three last joints of the crane, these actuators are used. The 3 last joints are the 2nd, 3rd, and 4th joints from Figure 41 - Linear actuator Actuonix. These actuators are crucial for the compensation of the waves. They help primarily with the compensation of the up and down movement and a little bit for the tilt, as the base already compensates for a lot of tilt. In the linear actuator is a DC motor that provides the power. The actuator is a p-type and has built-in limit switches. The P stands for 'potentiometer', the potentiometer will get us the position of the actuator at any time.

The characteristics of Figure 42 - L12 specifications (gear ratio 100, 12V) show that the speed will drop when there is more force acting on the actuator. Notice how the current drops when the applied voltage is doubled. After some tests on the second joint with this actuator, it was noticeable that it could not hold the weight. There was also an actuator missing in the beginning. When new parts were ordered, a linear actuator with a higher gear ratio was chosen. Now there is an actuator with a 150:1 gear ratio instead of 100:1.



Figure 41 - Linear actuator Actuonix (25)

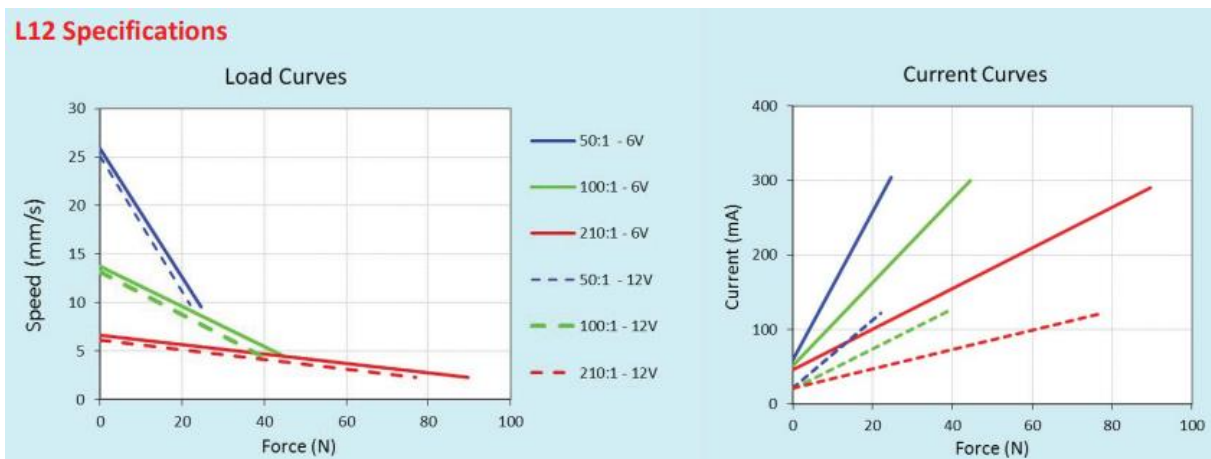


Figure 42 - L12 specifications (gear ratio 100, 12V)

Table 3 - Actuonix specs

	Stroke length(mm)	Gear ratio	Max load(N)	No load speed (mm/s)
2 nd joint (type L16)	50	150:1	200	8
3 rd and 4 th joint (type L12)	50	100:1	42	13

4.1.8 Arduino

The Arduino uno and Arduino uno wifi rev 2 were used. These are the Arduinos that were ordered last year. An Arduino is a microcontroller that can be used for a lot of things. In our case, we use it as a translator to send PWM signals to the actuators and to transform the data of the gyroscope into something readable for the PLC. The PLC can also send PWM signals, but the frequency is not fast enough for the motor controllers. (3) There are PLC modules that can produce a quicker PWM signal, but the ones from school cannot. This is the reason that the decision is made for using Arduinos as a translator. You could say that in a way the Arduino functions as a slave of the PLC. We use a normal Arduino uno for the stepper motors because the library for the stepper doesn't work so well on the wifi rev 2.

4.1.9 PLC

The PLC (Programmable Logic Controller) is a computer that is built for the industry. We use the PLC to run the main program. It is a very robust computer that can handle a lot of data. We have a learning PLC at our disposal. That means that there are some restrictions on it. For example, this PLC doesn't use all of its IO's: only half of the IO's are usable. After communicating back and forth, we had 3 options.

- The first one is to take a second learning PLC and connect them via an Ethernet cable.
- The second option is to get a few modules from the first PLC and put them on the other one.
- The third option is to buy screw connection modules that can be implemented. After communicating back and forth, it was not allowed to take apart the PLC, so the second option is not a possibility.

The choice is pretty clear to us. The third option is the best one because we do not have to try to let the two PLC's communicate via Ethernet. You also would not need to carry 2 PLC's while moving the RoboCrane. Fewer components make it easier.



Figure 43 - PLC S7-1500 (26)

Voltage Divider

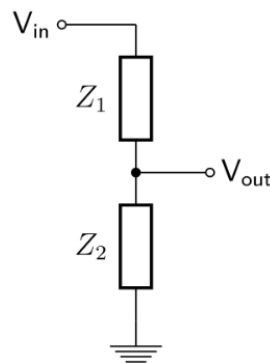


Figure 44 - Voltage Divider

The digital output signal of the PLC is 24VDC. The Arduino works on 5VDC, so a voltage divider is needed. Due to lack of space on the soldering boards, it is best to just have 2 resistors and not a few resistors in series. The standard resistors that were on disposal were the E12 resistors. This means that 1.0, 1.2, 1.5, 1.8, 2.2, 2.7, 3.3, 3.9, 4.7, 5.6, 6.8, and 8.2 times 10, 100 and 1000 ohm were available in the lab. To minimize the quantity of the resistors, the formula is best used with a fixed value of one of the above. Afterwards, the result can be compared to the available resistors. They also need to be big enough to limit the amount of current flowing from the PLC. A simple example of a voltage divider is two resistors connected in series, with the input voltage applied across the resistor pair and the output voltage emerging from the connection between them. The formula that is been used is based on Ohm's law (27).

- Z_1 Value of the first resistor $[\Omega]$
- Z_2 Value of the second resistor $[\Omega]$
- V_{in} Voltage input $[V]$
- V_{out} Voltage output $[V]$

$$Z_1 = \frac{V_{in} * Z_2}{V_{out}} - Z_2 \quad (1)$$

Fill $Z_2 = 1000\Omega$ into (1):

$$Z_1 = \frac{24 * 1000}{5} - 1000 = 3800\Omega$$

To get 5VDC, Z_1 needs to be 3800 Ω . This resistor is not available at the lab, the closest value is 3900 Ω . Let's see how much V_{out} is when Z_1 is 3900 Ω .

$$V_{out} = \frac{Z_2}{Z_1 + Z_2} * V_{in} \quad (2)$$

Fill $Z_1 = 3900\Omega$ and $Z_2 = 1000\Omega$ into (2):

$$V_{out} = \frac{1000}{1000 + 3900} * 24 = 4,897V$$

Conclusion, this value is high enough for the Arduino to detect it and not too high to break the digital pin.

4.2 Electrical wiring diagram

For the exact connections between PLC and Arduino we refer to Table wiring connections

Legend:

- **Red:** power supplies
- **Black:** Ground
- **Blue:** Communication from and to PLC
- **Yellow:** Data communication from Arduino

4.2.1 Base and first joint

In this wiring diagram, the first Arduino is visible, together with the three H-bridges and the five linear actuators. The power supply of the 24VDC is directly connected to the H-bridges. The ground of the 24VDC supply is not connected to the ground of the 5VDC circuit. There are two analog inputs coming from the PLC, which are the switches for the base. The other two digital inputs are the two switches for moving the first joint.

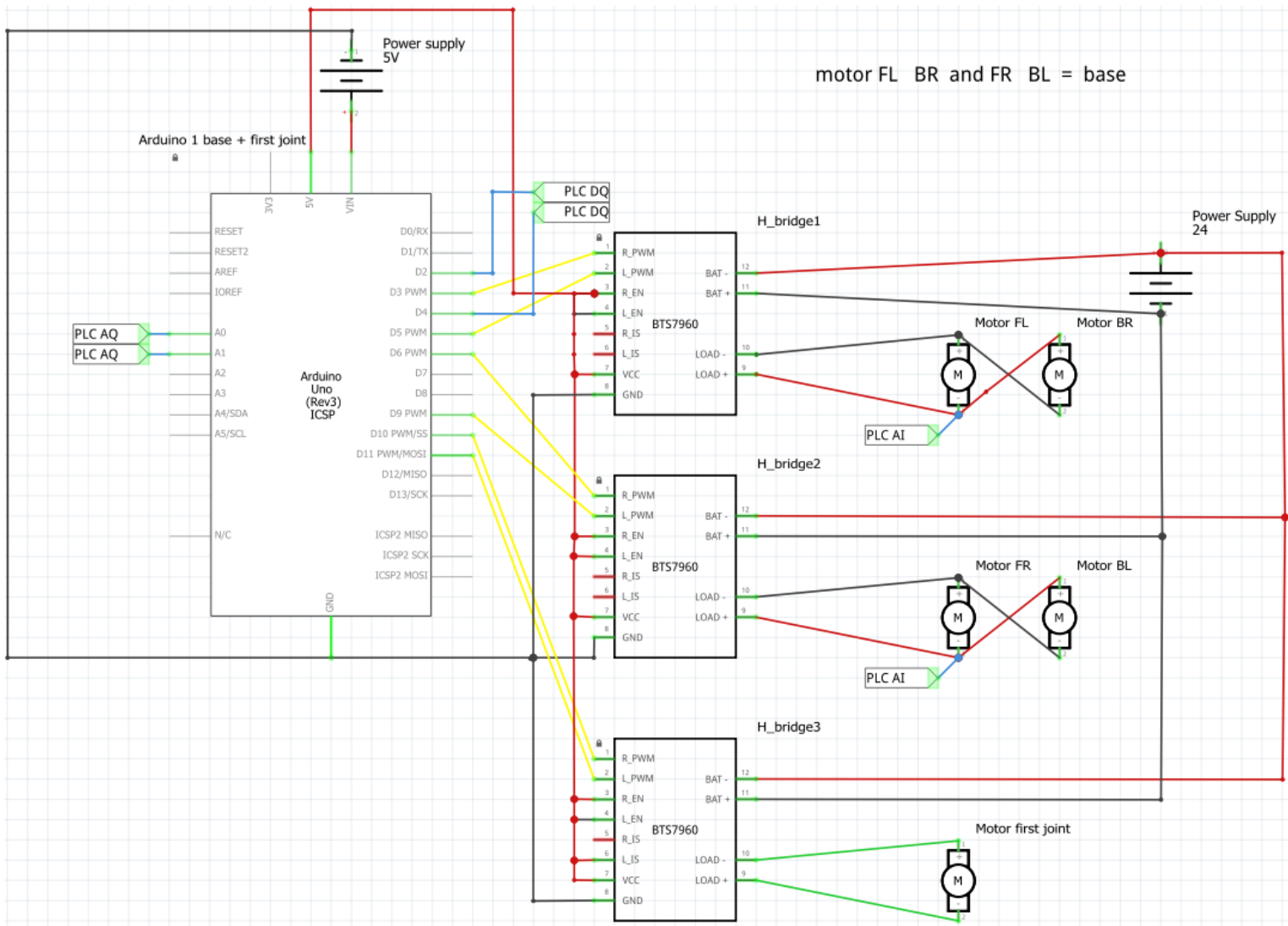


Figure 45 - Electrical diagram base and first joint

4.2.2 Gyroscope

The second Arduino is visible together with the gyroscope (LSM9DS1). The gyroscope needs 3,3V or 5VDC and two cables for the data and scale of the I²C. There are three analog outputs from the Arduino to the PLC. One for each direction, namely the x, y, and z-axis.

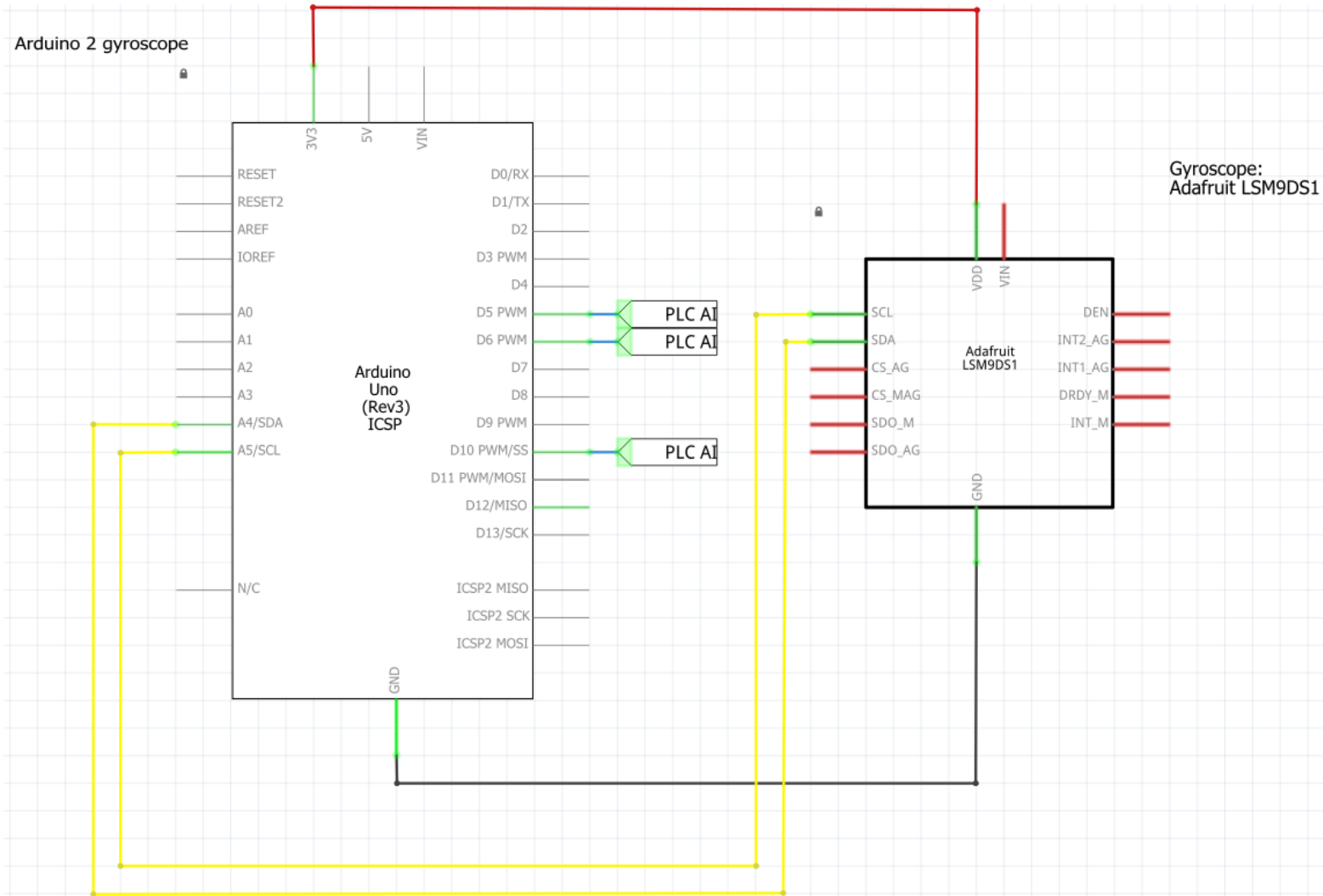


Figure 46 - Electrical diagram Gyroscope

4.2.3 Steppers + servo basket

The electrical diagram (Figure 47 - Electrical diagram steppers + servo basket), which is made for the 'steppers + servo basket' program, shows the third Arduino, together with two drivers, two steppers, and a servo. There is a 24VDC power supply needed for the stepper motors. The power supply is not connected to the control circuit of 5VDC. The 5VDC is connected to the 'enable' pin of the drivers. This is because of security, this causes the stepper motor to stay powered and stand still. This is important because, if the base is tilted, the end of the arm is leaning down due to gravity. This force needs to be held back by the stepper. The same goes for the other stepper motor. The telescopic arm can't go in when the stepper motor stops rotating.

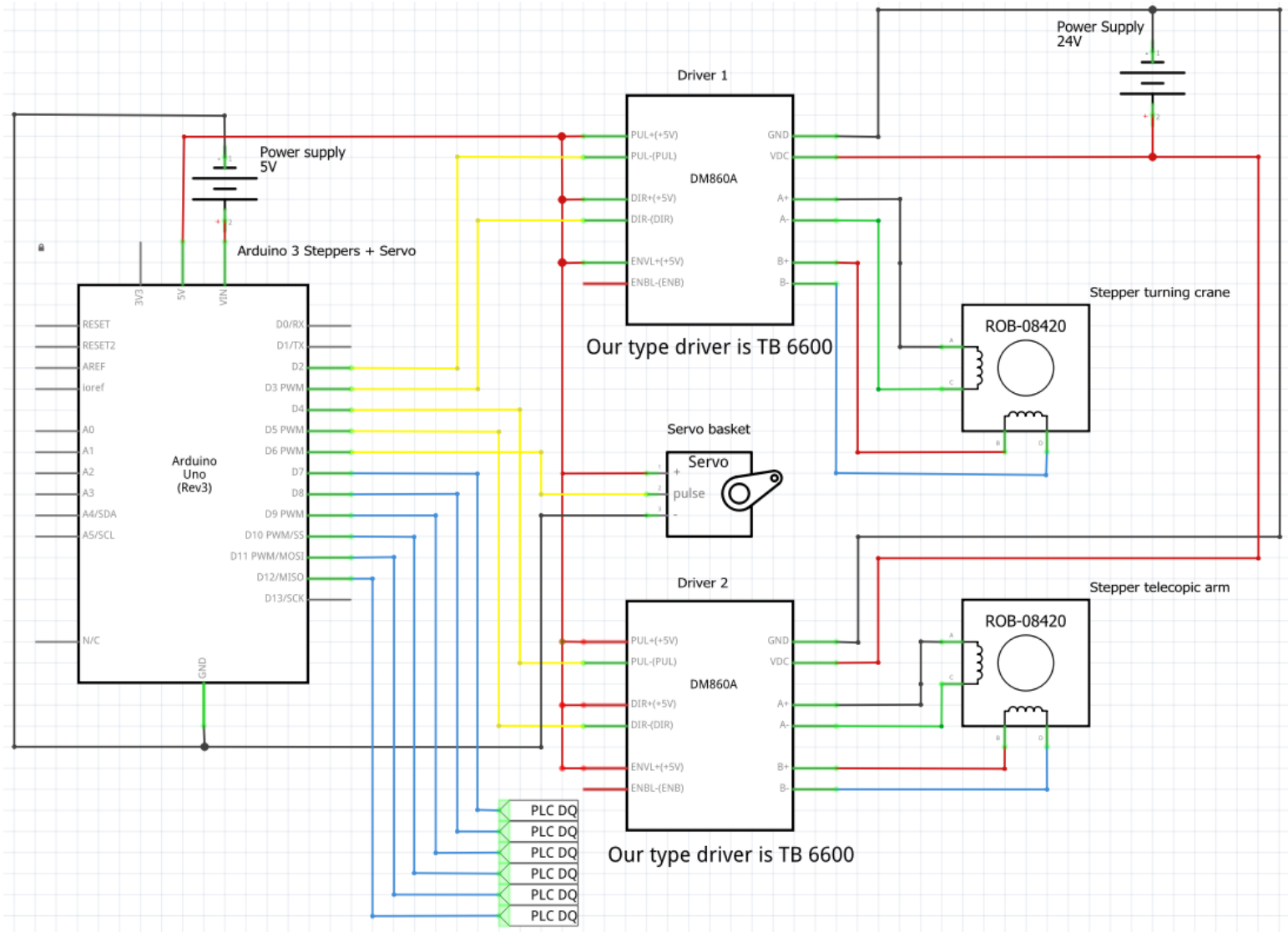


Figure 47 - Electrical diagram steppers + servo basket

4.2.4 Linear actuators arm

In this last diagram, a fourth Arduino is used, together with two dual H-bridges. The BTS7960 on the diagram is not the right one. It is not possible to put the dual H-bridge (L298N) into the diagram, so for simplification, a third H-bridge is used in the diagram. The difference is in the PWM inputs of the dual H-bridge. The dual H-bridge only has one PWM input for every motor whereas the single H-bridge has one for each direction. The third and the fourth linear actuator are together on one dual H-bridge, while the second has its own. The second and third will be controlled with an analog input while the third will be controlled with two digital ones.

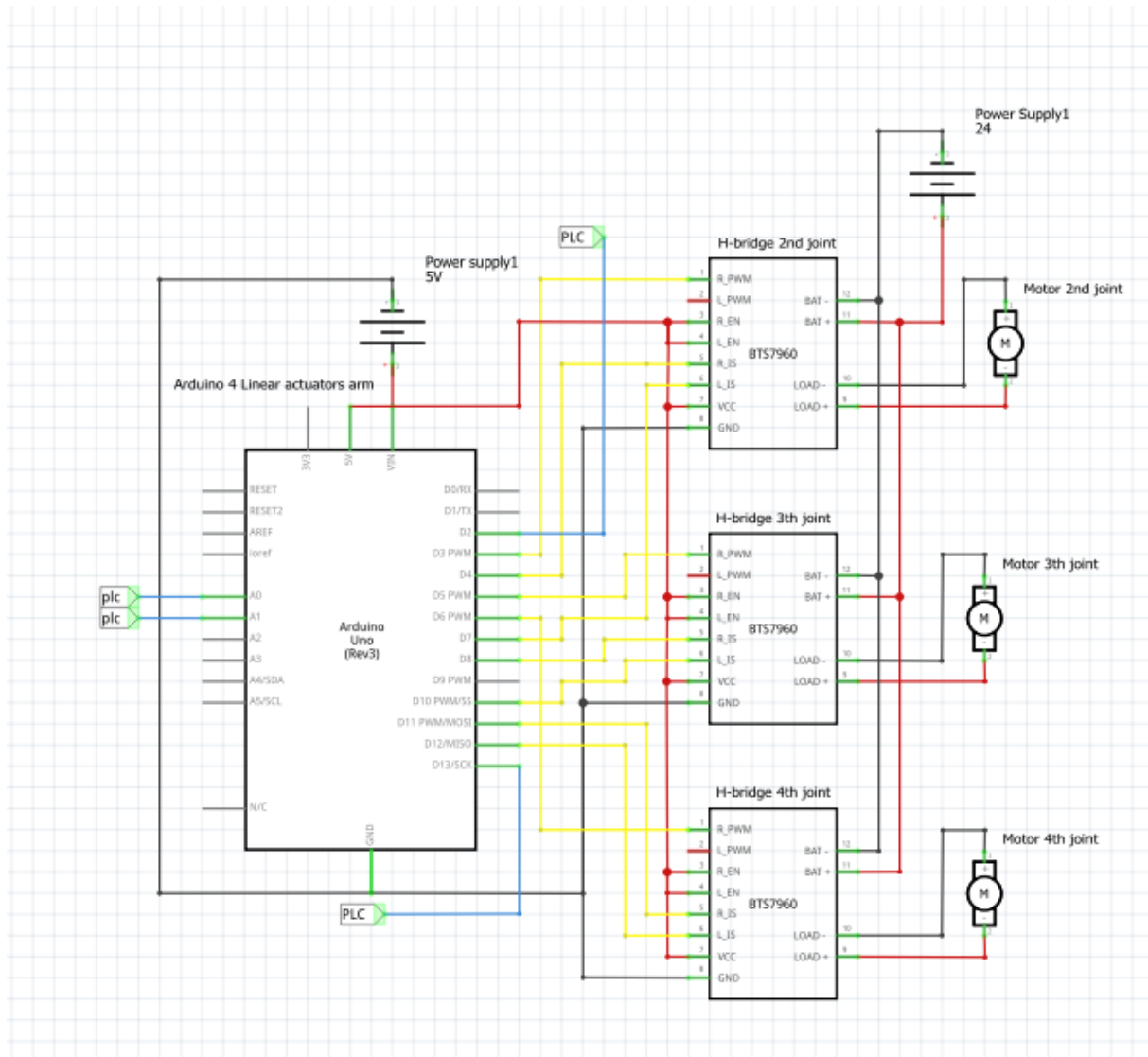


Figure 48 - Electrical diagram linear actuators arm

4.3 Programming aspect

All drafts can be found on our GitHub page. Final programs are attached, and on the GitHub page.

4.3.1 Arduino

As you can see from Figure 45 - Electrical diagram base and first joint - Figure 48 - Electrical diagram linear actuators arm there are four Arduinos used in this project. Each one of the Arduinos with its own unique code. The code is made to control the machine manually with feedback from the sensors. With this feedback, it is possible to calculate where the endpoint of the arm is. It is also possible to read this data from the gyroscope.

4.3.2 Base and first joint

The process

At the beginning of the process of writing the end program, a program was written to only move one actuator with an H-bridge. The 24VDC supply of the PLC was used to move the motors. After connecting a second motor in reverse to the H-bridge, it was clear that the PLC supply could not handle it. The 24VDC supply is meant for electrical steering and not for electrical power. Afterwards, the power supply was connected and programmed into the PLC for testing. When this was done, the program of the first joint was written into the Arduino and PLC.

End result

The Arduino reads the analog and digital data that comes from the PLC. The data is simply the input of the switches from the PLC. There are switches for moving every joint. In this program, only the switches for moving the base and the first joint are read by the Arduino. The two switches for the base are analog signals and the two switches for the first joint are digital signals. The reason that the switches of the base are analog is to minimize the number of switches. If it was a digital signal, four switches were needed, two for each direction. Now there are two analog signals, which are each separated into two parts. For the up-and-down movement of the first joint, there are two switches, one to go up and one to go down. The only thing that this Arduino does, is read the data from the PLC and then send a PWM signal to the correct motors. At the same time, the PLC reads the position of the actuators of the base and first joint (reference Arduino 1: Base and first joint).

4.3.3 Gyroscope

The process

In the process of getting the right angles of the gyroscope, we had a few milestones. In the beginning, we had the gyroscope type 'BNO055' at our disposal. A lot of time went into researching how to program the gyroscope. The temperature was the first element that got displayed, which was just for testing and getting to know the library. After this, the accelerometer and the magnetometer were programmed. The accelerometer did not work properly, but the magnetometer worked. The decision was made to buy a new one. It was another type, the 'LSM9DS1'. This type had other libraries, so we had to start over for the biggest part. In the end, the Arduino could read the values of the accelerometer and the magnetometer in the end. We followed the research from the previous year (3), and used the ADAC Click to convert the signal from I²C to digital for the PLC. After testing, the problem was that the signal always dipped to the ground. The graph showed that there were spikes for every signal, but after every signal, it dipped back to the ground. After testing with sending the data over the analog port of the Arduino, the decision was made to send the data through the analog port. When the connection

with the PLC was made, it was noticeable that the signal fluctuated a lot. It came down to 20° of deviation. Due to lack of time, we didn't come to a conclusion and therefore we have no solution.

End result

The Arduino reads the data from the gyroscope and converts the signal to a 0-255 signal with a little calculation. It is possible to just use an easy map function for this. The problem is that it will round the numbers and therefore the accuracy lowers tremendously. To fix this, a function is written to calculate with floats that you can see in Figure 49 - Map float function. After this, the 0-5VDC signal that comes out of the analog output of the Arduino is sent to the PLC.

```
void mapfloatx(float x, float in_min, float in_max, float out_min, float out_max)
{
  rollv = (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Figure 49 - Map float function

4.3.4 Steppers + servo basket

Full program in Attachments : Arduino 3: Steppers + servo

The process

The process started with writing a program to control one stepper motor with analog input. The stepper drivers, in between the Arduino and the stepper motor, need a 5V direction and a pulse, which the Arduino provides. The analog input from the PLC controls direction and speed. The first program written worked for one stepper motor, it uses delays to write a pulse to the driver. This delay is in direct control over the speed. If we wanted to connect another motor, this way of programming would become a complicated mess. That is why a library was used to drive them. The library (28) uses the hardware interrupt timer, controlling the drivers in the background, leaving the sketch free for other programs. Using this library, a speed and direction are needed. There also needs to be a dead zone set in the middle of the signal, to make sure interference doesn't drive the motors. In this program a basic servo control program is added and controlled by digital inputs. We did not have enough analog outputs on our PLC, therefore we changed all the inputs to digital. We didn't have the speed control but we can alter the speed manually in the program. The program with speed control can be found on GitHub (12) and it is included in this thesis to facilitate future progress.

End result

In the end, the Arduino gets 6 digital inputs, two for every actuator. You can adjust the speed of the steppers by adjusting the variable: "speed". The speed of the servo can be changed by changing the delay in the program. The limit points of the servo are programmed in the Arduino as we do not have another analog input module for the PLC. The steppers are controlled by the drivers which are controlled by the Arduino. The Arduino gives a digital step pulse signal and a digital direction signal.

4.3.5 Linear actuators arm

Full program in Attachments: Arduino 4: Linear actuators arm

The process

For the program of the linear actuators of the arm, there was first a program written for one actuator. The L298N dual H-Bridge needed different inputs than the HW-039. The program written for the dual H-bridge used one PWM signal to control the speed of one actuator while the other H-bridge used two. With this H-bridge, we need two digital pins and one PWM pin for every motor. By writing one of the digital pins high, we set a speed. When the PWM pin is given an analog value of 0-255, then it adjusts the voltage from 0V-5V. This variable controls the speed of the motor. An analog input is linked to the direction as well as the speed. There was not enough space to put the fourth joint on the analog pins. Therefore two digital pins were used to control the fourth joint.

End result

The program uses 'if statements' to write the different direction inputs high or low. Two of the motors get controlled by an analog input, which controls direction and also speed. The third motor gets controlled by two digital inputs which only control direction.

4.3.6 PLC

Programs can be found on GitHub (12).

preparations

At the beginning of the project, a server was set up to make programming more streamlined. It started with research (29) (30) (31) (32) about how to set up and where we could run the server. The server could not be set up through OneDrive. After some more trial and error, a server was set up on a personal laptop (Figure 51). Every user needs their own account on the pc which hosts the server. After the setup is made, we need to meet some requirements:

1. Connect to the same network;
2. Host laptop can be detected on network (settings on "private");
3. Server start up: startup in "TIA Project Server V16-Configuration";
4. Login with username.

With this server, we get three different flags: blue means that you are working on this program block alone, yellow means another person is working on this, and red means you are both working on this program block (Figure 50 - Map structure). The flags appear automatically when you make a change or when you set them manually. To get rid of the flags, synchronize your program. After some tests, it worked like it should. This will make programming simultaneous more streamlined.

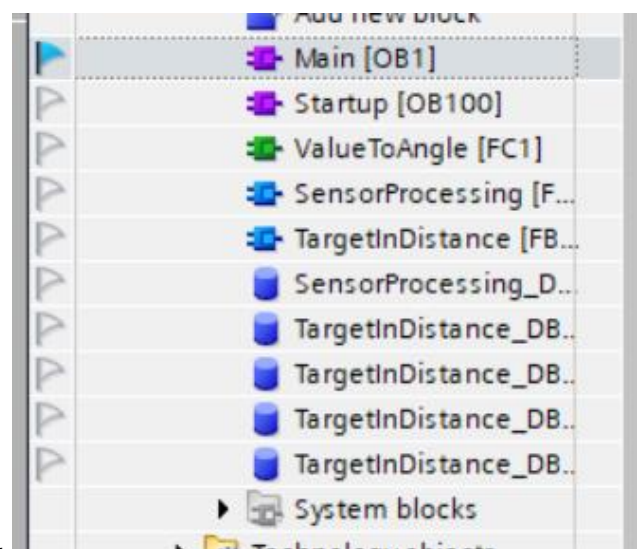
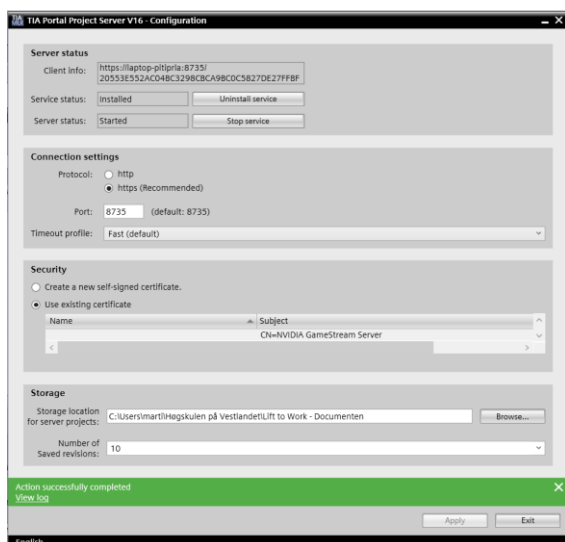


Figure 51 - Server setup

Figure 50 - Map structure

The process

The process of making the PLC program itself was very gradual. It is built parallel with designing the base and reducing the play of each joint.

Step 1:

In the beginning, the focus was more on the gyroscope. The reason that the focus was on the gyroscope is because the base was being rebuilt at the same time. It was not sure yet how the base would be designed and so it would have been a bit difficult to make a program for it at that time. When the Arduino program for the gyroscope was ready, a connection was made with the PLC. After getting the values from the gyroscope through the ADAC click module, it was clear that it would not work. The ADAC click converts the I²C to an analog signal that the PLC can read. While doing so, it dips after every signal to the ground. The PLC can read the values more quickly than the Arduino can send them, so the PLC got wrong values continuously. After this, we tried it with the analog output of the Arduino and it worked well. Another problem with the gyroscope is that the accuracy, after the readings on the PLC, was not so good. The PLC gives a deviation of 20° when the gyroscope is laying still. Due to a lack of time and putting the priority on making an overall program with readings from each joint, the gyroscope could not be finished.

Step 2:

After the gyroscope, the design of the base was ready and so a program for the base could be written. Making the base (welding, drilling, filing,...) and writing the program for it went parallel. At this moment the PLC program was as follows: There are four switches. Each switch is for one direction of the base. The PLC reads the digital input from the switch and converts this signal to an analog signal. In this case, the analog signal is a fixed value. A digital signal is a lot easier, so why did we not use a digital signal to send to the Arduino? This is because we already thought about the speed control when the machine has to compensate for itself. In this case, the machine needs to send analog values to the Arduino, so that the speed of the actuators can change, otherwise it will be a bumpy ride. On the left of Figure 52, you can see that, when you press the button to go up, it will send an analog value of 27648. When you press the button to go down, the PLC sends an analog value of 10368. At this moment the PLC could read in the position of the actuators from the base and move them while pressing the buttons. The next step was to put virtual end switches on the actuators. The actuators have 'built in' end switches, but they do not stop at the same time, which could cause the base to pull itself crooked. On the right, you see that when the actuator has reached 8000 or less, that it cannot move down anymore. In the hardware, only the connection of the front left and right are connected to the PLC. This is enough information because the other two actuators are just following the front two (due to hardware connection via the H-bridges).

After the base was programmed, the first joint needed to be programmed. This happened at the same time that the slew bearing was installed onto the base. The first joint is not a part of the automatic wave compensation and there was not enough space on the analog ports of the PLC. The decision was quickly made to put the first joint on digital signals. When you press the button to go 'up', the PLC sends a digital '1' to the Arduino. The Arduino then controls the H-bridge of the actuator. The first joint also needed virtual endpoints.

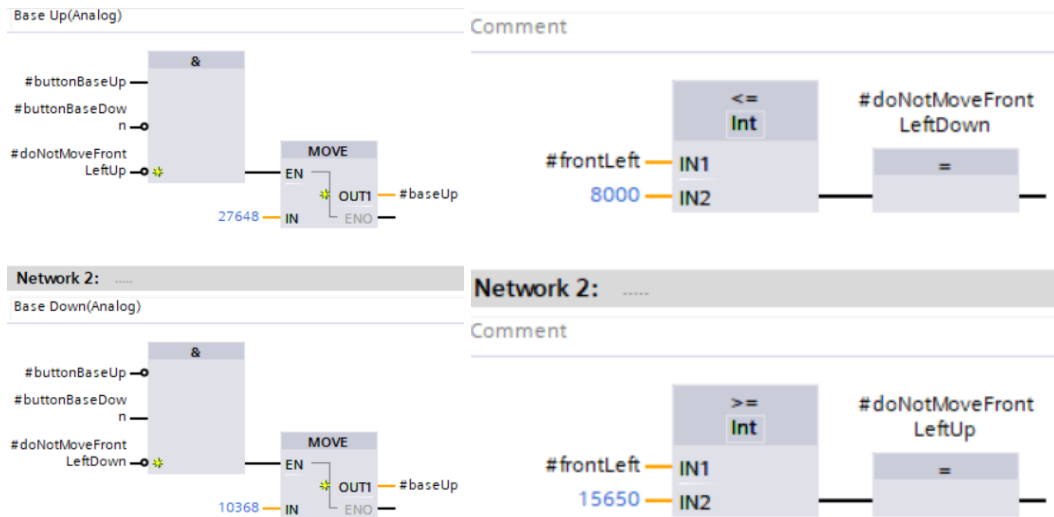


Figure 52 - 'Movement analog' and 'motor_position'

Step 3:

After the base and first joint was programmed and wired, it was time for the steppers and servo. The steppers and servo both need digital outputs on the PLC. Therefore, the same program as the first joint can be used. Cleaning up the program and giving it some more structure proved to be easier. A basic digital control program was made (Figure 53) and was placed in a FC (right side Figure 53) for each digital input that needed to be controlled. The limit variable can be connected. If we have the value, it doesn't need to be connected, so we will leave it in the program making future additions of encoders easier. For now we use the buttons to control the motor.

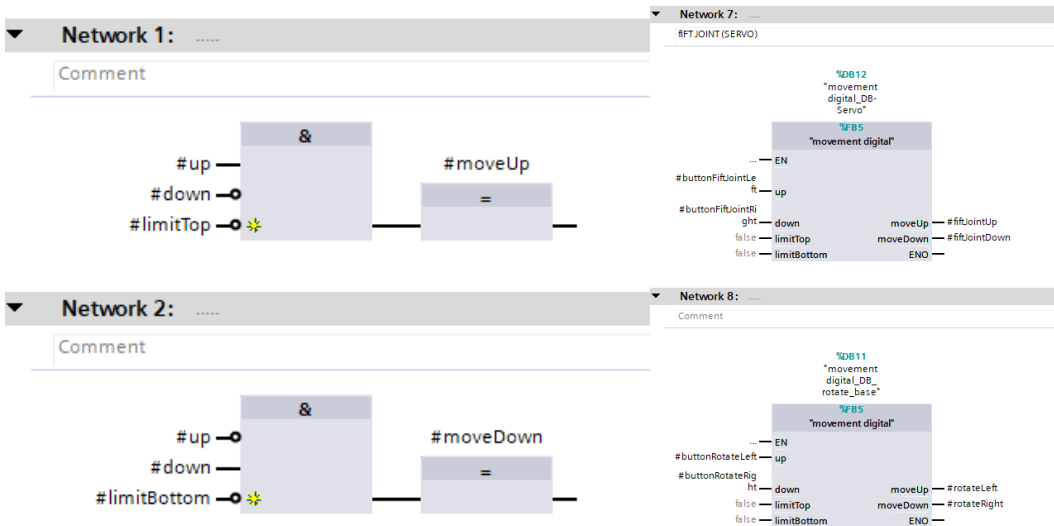


Figure 53 - 'Movement digital' and 'movement'

Step 4:

As last step, three joints of the crane needed to be programmed. Ideally the three joints should be analog (left side Figure 52), but because there is not enough place on the analog PLC ports, the last joint needed to be digital (left side Figure 53). This means that joint four can not change its speed. This means that there are two analog and one digital actuator. Which is exactly the same as the program for the base and first joint.

End result

The end result consists of cleaning up the program. Because the Arduino takes care of the “translation” we only need two types of output programs to move the entire crane. Those two programs: “movement analog” and “movement digital”, are then called upon in the FC: “movement”. In this program the links between the actuators and PLC are created. Some of the actuators give their position through a potentiometer, which will be processed in the “motor_position” FB. This function block takes care of the limit of the different actuators. Manually putting joints at their outer limits, the potentiometer values can be linked to the positions. These will be implemented in the “motor_position” FB to ensure the crane does not tear itself to pieces. The FC “movement” and FB “motor_position” get called upon in the Main OB and linked to their corresponding connections. For the connections between PLC and Arduino, we refer to the attachment Table wiring connections.

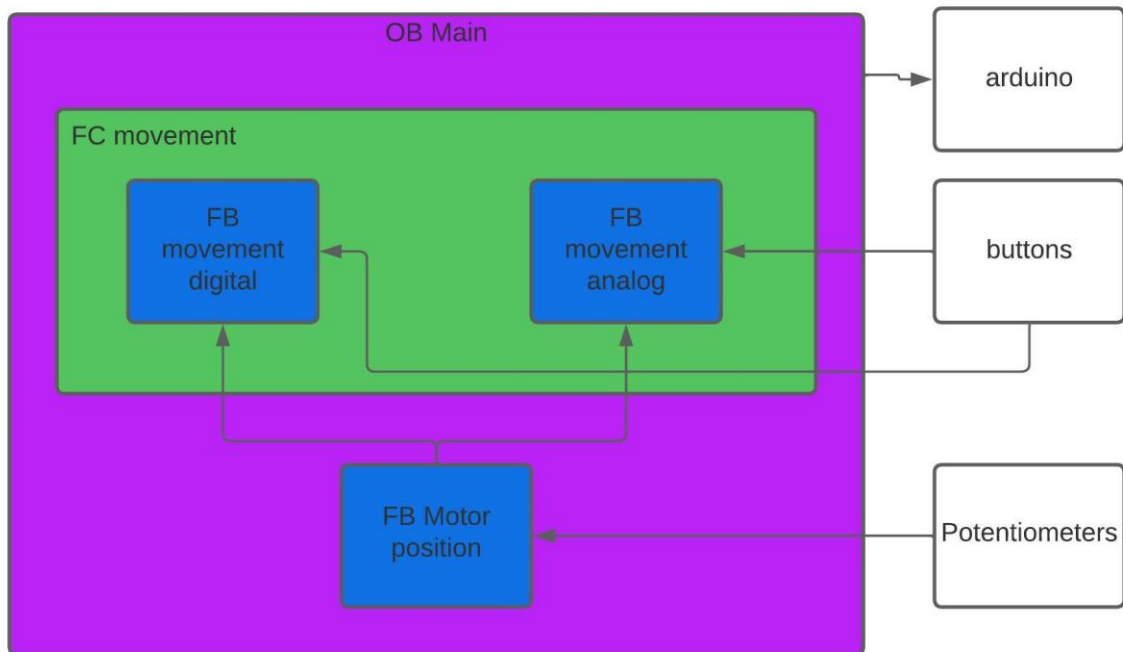


Figure 54 - PLC program diagram (33)

5. Conclusion

As a conclusion of this thesis, we can say that there is a sturdy base and crane with not much play. Tests have proven that the crane with the base can be moved manually with switches from the PLC. The position of the actuators can be read on TIA Portal in real-time. This can be used to determine the end position of the crane. As a translator for the PWM signals, an Arduino is used to control the actuators. There are programs written for the Arduinos in C++ and for the PLC on TIA Portal in FBD.

As for the future, some more thought needs to be put into the gyroscope together with the dynamics equations. For the crane to counteract the waves on its own, the dynamics equations need to be implemented into the program of the PLC. The program of the gyroscope is written and implemented into the PLC program, but not yet connected.

6. References

1. Dynamic positioning. *Wikipedia*. [Online] https://en.wikipedia.org/wiki/Dynamic_positioning.
2. ResearchGate. [Online] https://www.researchgate.net/figure/Definition-and-directions-of-motions-of-ship-in-the-previously-defined-body-fixed-steady_fig8_303299722.
3. Gielen, Bram Deboel & Jasper. *Design and production of a RoboCrane*. NO-5063 Bergen, Norway : Western Norway University of Applied Sciences, 2022.
4. Design thinking. *IDEO designthinking*. [Online] <https://designthinking.ideo.com/>.
5. *evolventdesign*. [Online] <https://evolventdesign.com/pages/spur-gear-generator>.
6. 3D PRINTING TOLERANCES & FITS. *3dchimera*. [Online] 14 January 2019. <https://3dchimera.com/blogs/connecting-the-dots/3d-printing-tolerances-fits>.
7. original Prusa i3 MK3S+. *Prusa3d*. [Online] <https://www.prusa3d.com/category/original-prusa-i3-mk3s/>.
8. ultimaker Cura. *ultimaker*. [Online] https://ultimaker.com/software/ultimaker-cura/?gad=1&gclid=CjwKCAjwgqejBhBAEiwAuWHioIFbLmYihVIGE_Uin6TilUDzIqR20PYGRP1j-TMuZ2j3wT55SPnQFxoCIXwQAvD_BwE.
9. Cura Setup for Flashforge Adventurer 4 Series. *FlashForge*. [Online] <https://en.fss.flashforge.com/10000/software/22be928feb149fb5d89206912b4e6350.pdf>.
10. Datasheet Ewellix linear actuator. [Online] <https://medialibrary.ewellix.com/asset/16184>.
11. aluflex. [Online] <https://www.aluflex.no/>.
12. githubOPTS. *github*. [Online] may 2023. <https://github.com/MartijnSpaas/OPTS>.
13. Datasheet H-bridge BTS7960. [Online] <https://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf>.
14. L298N datasheet. *L298N datasheet*. [Online] <https://docs.rs-online.com/d24b/0900766b8135f847.pdf>.
15. Handson technology. *Datasheet L298N*. [Online] https://components101.com/sites/default/files/component_datasheet/L298N-Motor-Driver-Datasheet.pdf.
16. RS components. *Stepper*. [Online] <https://no.rs-online.com/web/p/stepper-motors/5350401>.
17. Farnell. *Stepper motor driver*. [Online] <https://no.farnell.com/dfrobot/dri0043/stepper-motor-driver-arduino-board/dp/3517878>.
18. Geckodrive. *Datasheet Geckodrive*. [Online] <https://www.farnell.com/datasheets/1496194.pdf>.
19. Mikroe. *ADAC Click*. [Online] <https://www.mikroe.com/adac-click-click>.
20. *RoboticsBackend*. [Online] <https://roboticsbackend.com/arduino-fast-digitalwrite/#:~:text=We%20have%20the%20answer%3A%20a,have%20a%20much%20better%20precision>.
21. *Datasheet ADAC Click*. [Online] <https://docs.rs-online.com/3c57/0900766b8161be99.pdf>.
22. Datasheet 24V supply. [Online] <https://www.meanwell-web.com/content/files/pdfs/productPdfs/MW/RPS-400/RPS-400-spec.pdf>.
23. Datasheet 5V supply. [Online] <https://docs.rs-online.com/c33b/A700000007001402.pdf>.
24. Datasheet 12V supply. [Online] <https://docs.rs-online.com/df40/0900766b816ed486.pdf>.
25. Actuonix. *Linear actuator*. [Online] <https://www.actuonix.com/116-50-150-12-s>.

26. Siemens. [Online]
<https://new.siemens.com/it/it/prodotti/automazione/systems/industrial/simatic-controller/simatic-s7-1500.html>.
27. Voltage divider. *Wikipedia*. [Online] https://en.wikipedia.org/wiki/Voltage_divider.
28. github. *hobby components / HCMotor*. [Online] july 2015.
<https://github.com/HobbyComponents/HCMotor>.
29. Setting Up a Project Server within TIA Portal V16. *YouTube*. [Online] 9 July 2020.
<https://www.youtube.com/watch?v=EvyPXRQbNYE>.
30. Tia Portal: Multiuser Projects Part1. *YouTube*. [Online]
<https://www.youtube.com/watch?v=SHhIzdR-DMw>.
31. Tia portal: multiuser Projects Part2. [Online]
<https://www.youtube.com/watch?v=1maSjWBdqY0>.
32. multiuser engineering with tia portal server. *Support.industry.siemens*. [Online] 09 08 2021.
<https://support.industry.siemens.com/cs/document/109740141/multiuser-engineering-with-tia-project-server?dti=0&lc=en-SE>.
33. lucidchart. [Online] https://lucid.app/lucidchart/810aa529-af5a-4650-93c8-d9a2802551e2/edit?invitationId=inv_024dd4ee-1327-4df0-b116-81d926326d3f&page=0_0#.
34. kaydonbearings. [Online] https://www.kaydonbearings.com/white_papers_15.htm.
35. grainger. [Online] <https://www.grainger.com/product/SKF-Radial-Ball-Bearing-6028-36MC76>.

Figure 1 - Ship movements (3).....	1
Figure 2 - RoboCrane	4
Figure 3 - Base axes	5
Figure 4 - Angled base	5
Figure 5 - Crane with crane parts	6
Figure 6 - Slew bearing (34).....	7
Figure 7 - Radial bearing (35).....	7
Figure 8 - Special assembly.....	8
Figure 9 - Generated gears (5).....	9
Figure 10 - Slew bearing angled base	9
Figure 11 - 3D printer profile	10
Figure 12 - Big gear slicer settings.....	10
Figure 13 - 3D printing the toothed ring with end result.....	11
Figure 14 - Linear actuator.....	12
Figure 15 - Outer and inner ring.....	12
Figure 16 - Standards and actuator mount.....	13
Figure 17 - Axis.....	13
Figure 18 - Pillow block.....	13
Figure 19 - Welded assemblies	14
Figure 20 - Plasma DXF file	15
Figure 21 - Realisation with plasma cutter.....	15
Figure 22 - Welding process	16
Figure 23 - First welds.....	16
Figure 24 - Base with outer ring.....	17
Figure 25 - Base with welded assembly	17
Figure 26 - Centrepiece	17
Figure 27 - Crane with base	17
Figure 28 - Cura settings	18

Figure 29 - Second joint	19
Figure 30 - New truss 2	20
Figure 31 - Basket and fifth joint	21
Figure 32 - Gyroscopes BNO055 and LSM9DS1	22
Figure 33 - Gyroscope signal	23
Figure 34 - Wiring diagram absolute analog output (10)	23
Figure 35 - H-Bridge HW-039 (13) and L298N (15).....	24
Figure 36 - Stepper motor (16).....	24
Figure 37 - Driver, TB 6600 + Geckodrive G201X38 (17) (18).....	24
Figure 39 - ADAC Click (19).....	25
Figure 40 - 24V, 12V and 5V power supply (22) (23) (24)	26
Figure 41 - Linear actuator Actuonix (25)	27
Figure 42 - L12 specifications (gear ratio 100, 12V)	27
Figure 43 - PLC S7-1500 (26).....	28
Figure 44 - Voltage Divider	29
Figure 45 - Electrical diagram base and first joint	30
Figure 46 - Electrical diagram Gyroscope.....	31
Figure 47 - Electrical diagram steppers + servo basket.....	32
Figure 48 - Electrical diagram linear actuators arm	33
Figure 49 - Map float function	35
Figure 50 - Map structure.....	36
Figure 51 - Server setup	36
Figure 52 - ‘Movement analog’ and ‘motor_position’	38
Figure 53 - ‘Movement digital’ and ‘movement’	38
Figure 54 - PLC program diagram (33).....	39

List of tables

Table 1 - Choice matrix rotating base	8
Table 2 - Power supplies	26
Table 3 - Actuonix specs	27

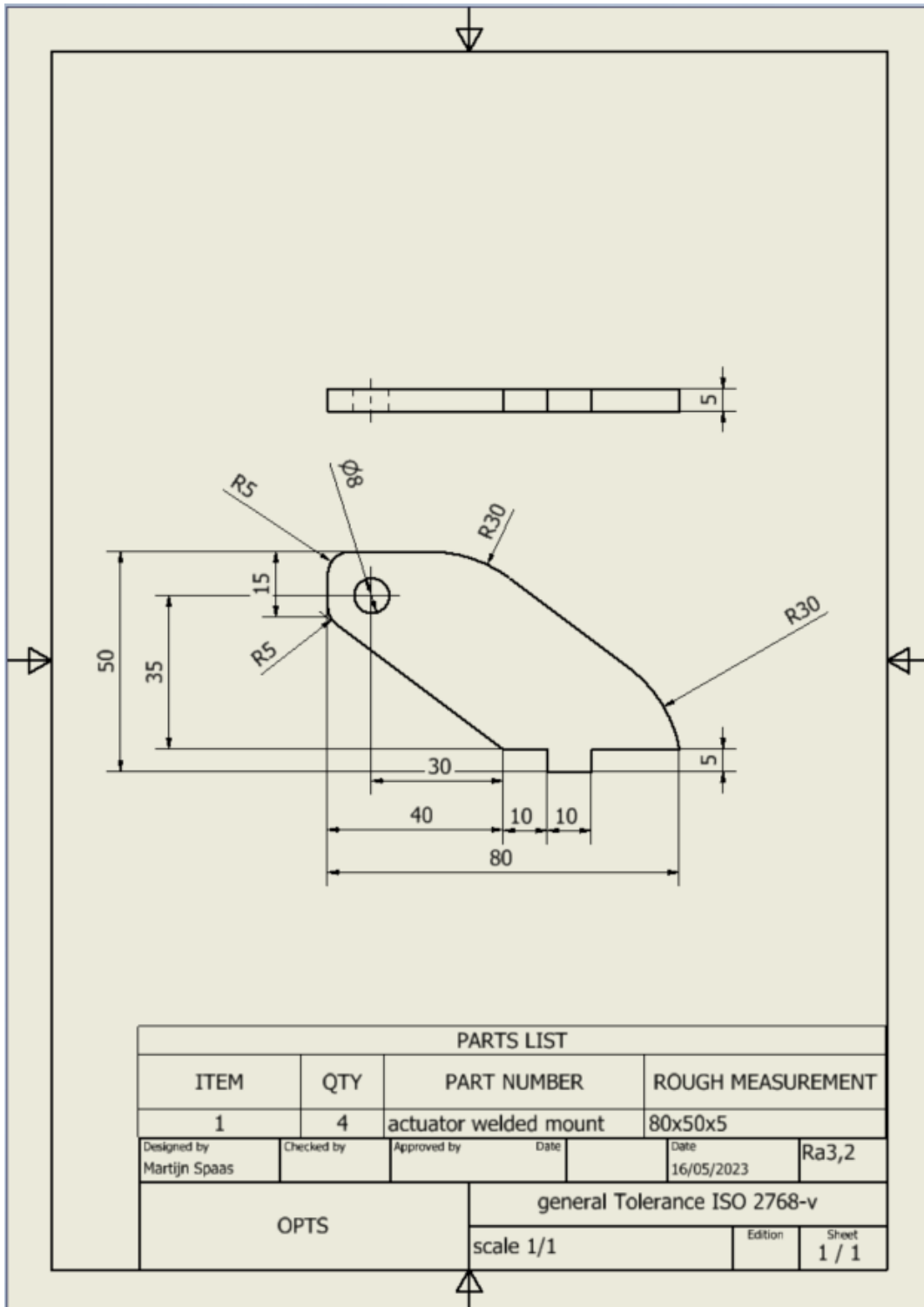
Attachments

Drawings plasma cutter

This contains the plasma cutted 5millimetre aluminium plate.

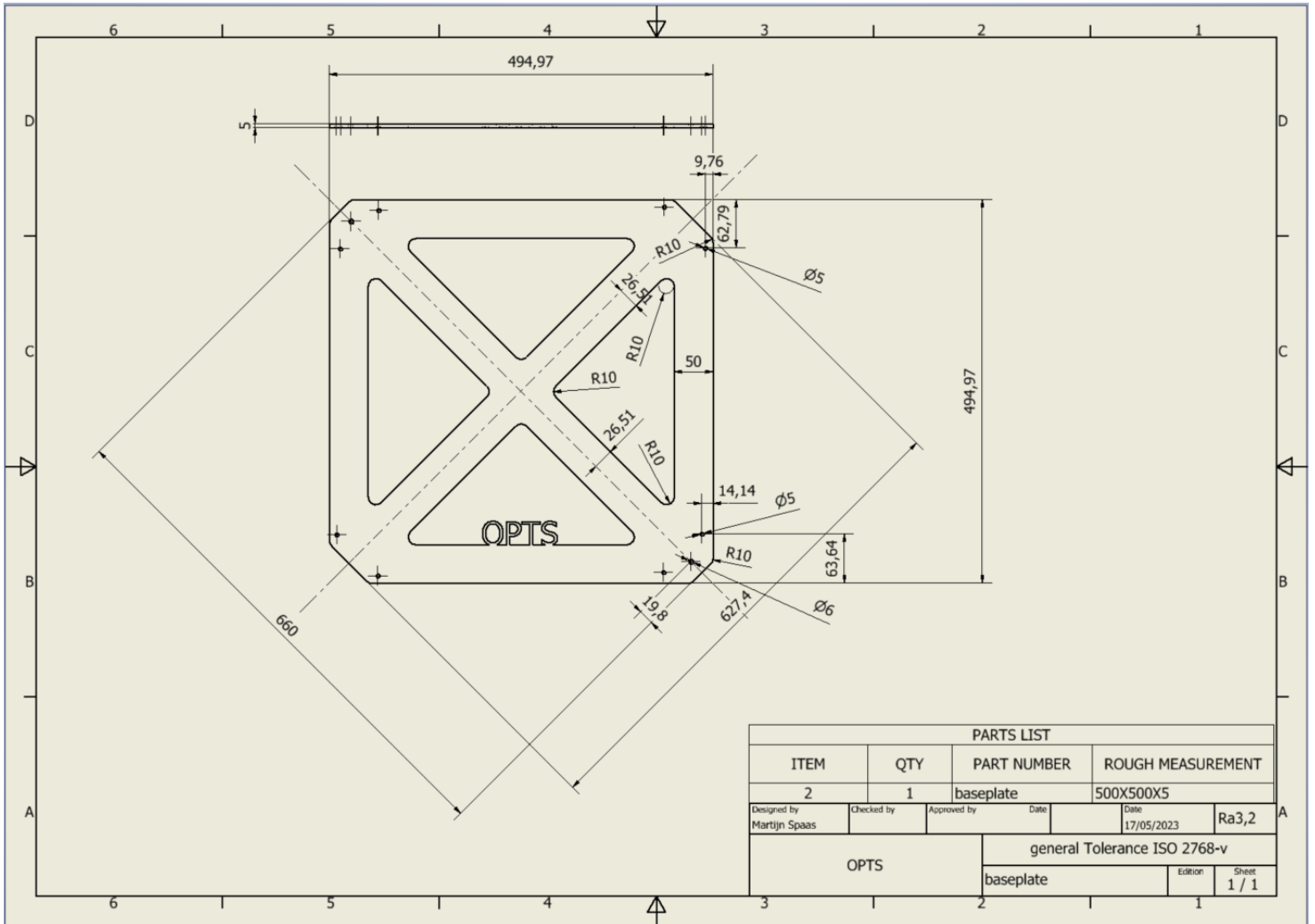
Drawing 1

Actuator welded mount. Used to move the inside ring of the base.



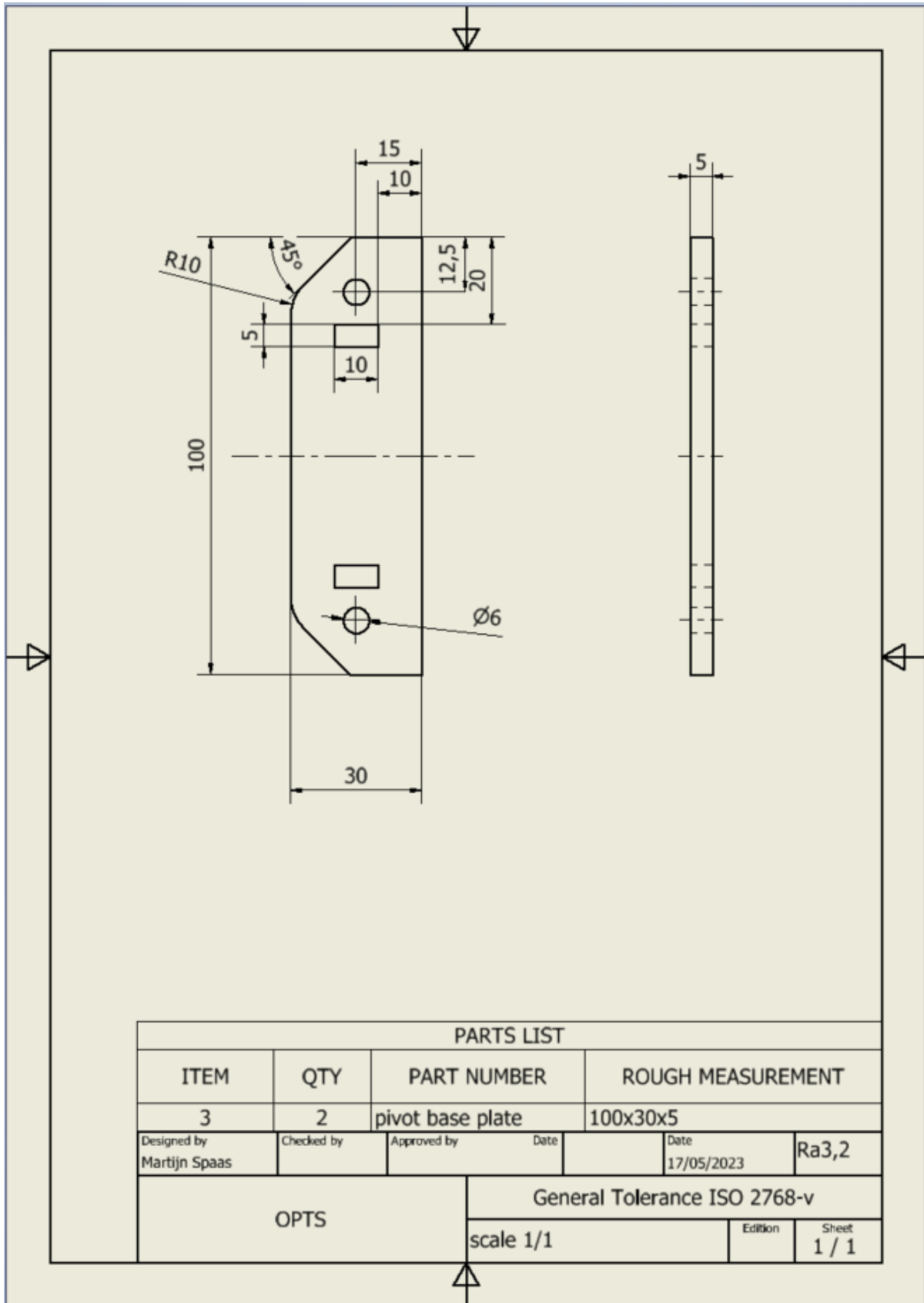
Drawing 2

Baseplate. The base of the crane.



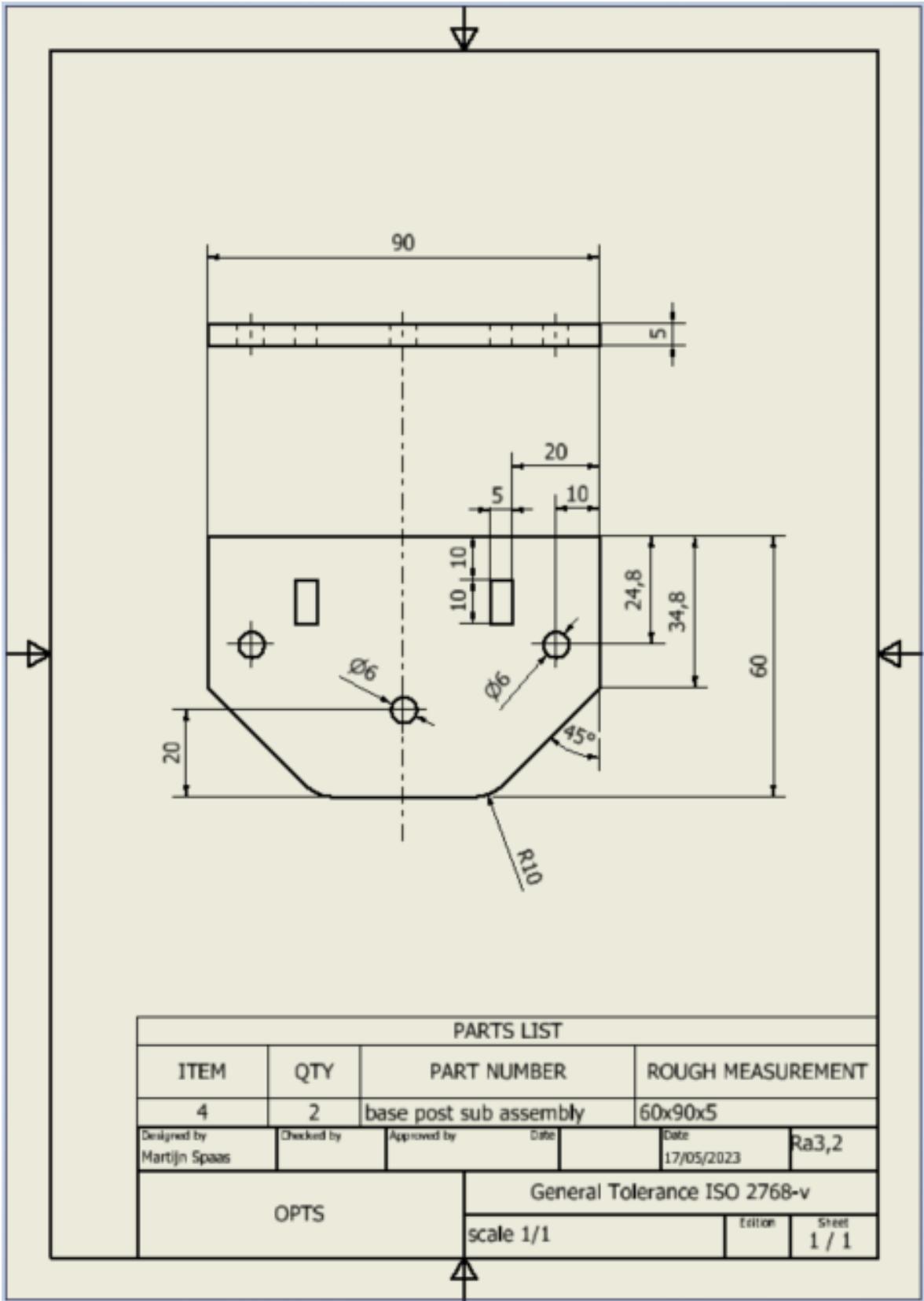
Drawing 3

Pivot base plate. Attachment for “actuator base mount” . will be bolted to the baseplate with M6 bolts



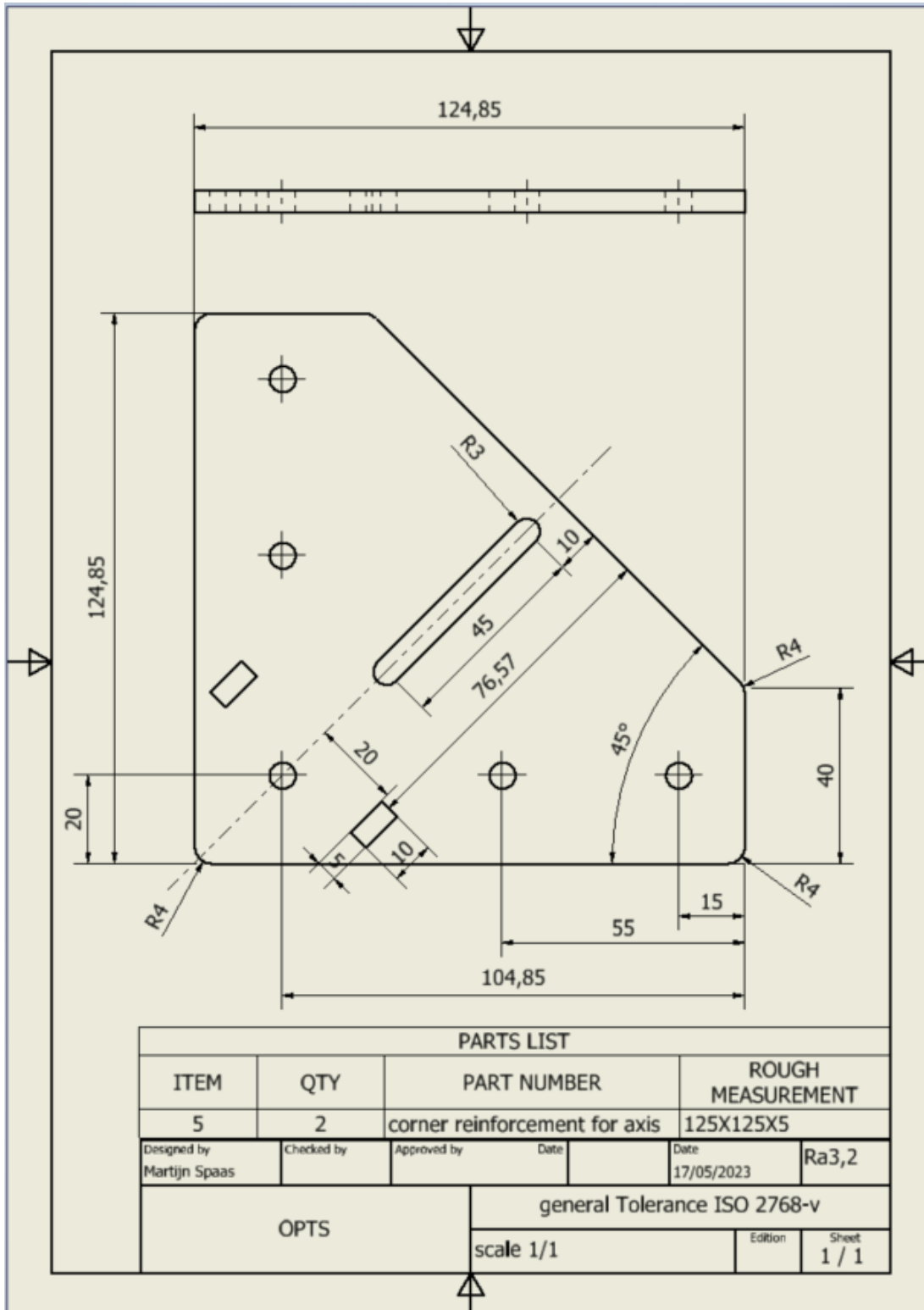
Drawing 4

Base post sub assembly. Will connect the aluminium post by welding it to “base post support”. And gets bolted to the base



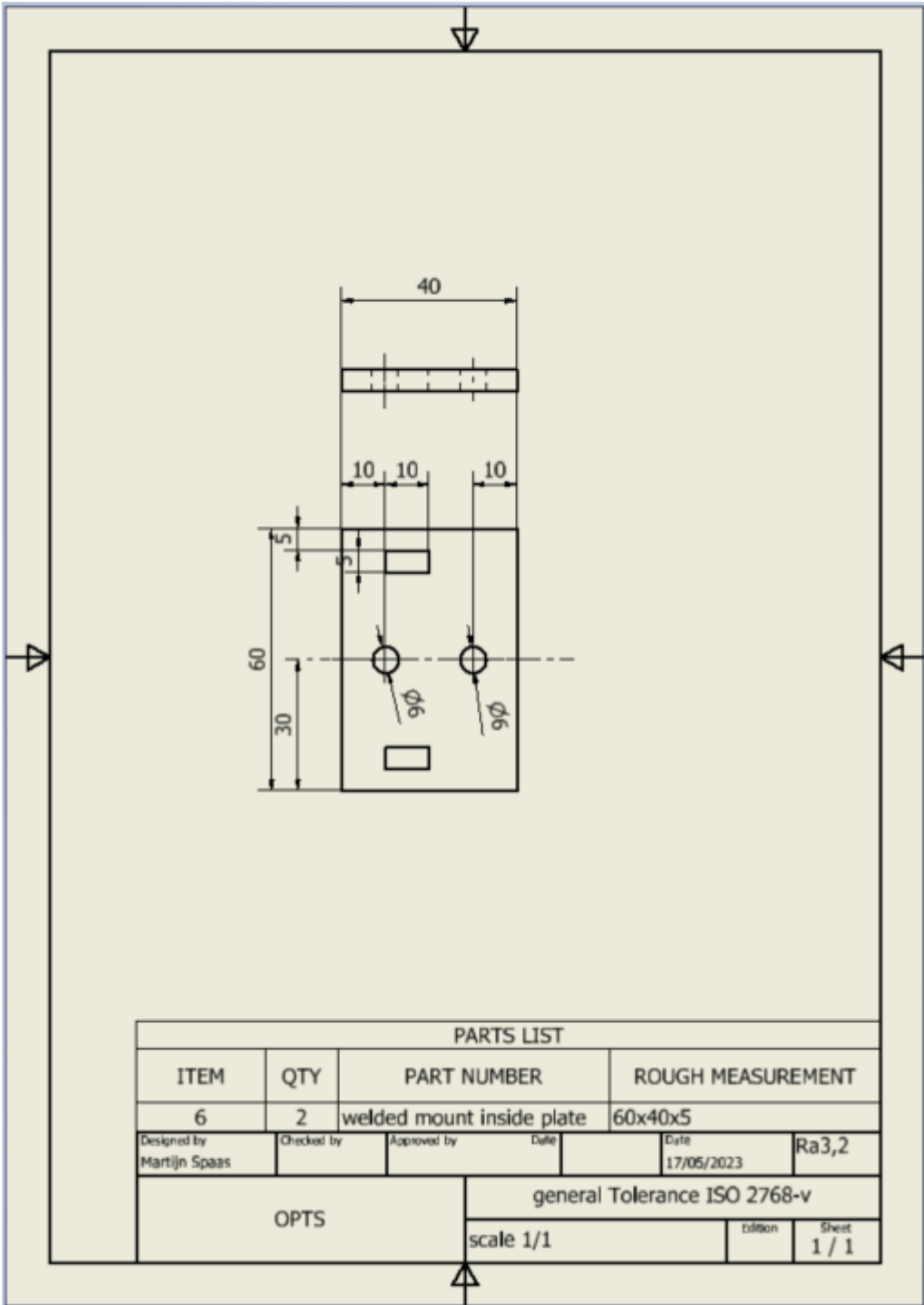
Drawing 5

Corner reinforcement for axis. This part will keep the extrusions in place with T-shape spring ball nuts and bolts. It also contains an adjustable mount for the axis. Which slides inside the slot. And rectangular slots to weld the “actuator welded mount outside” to.



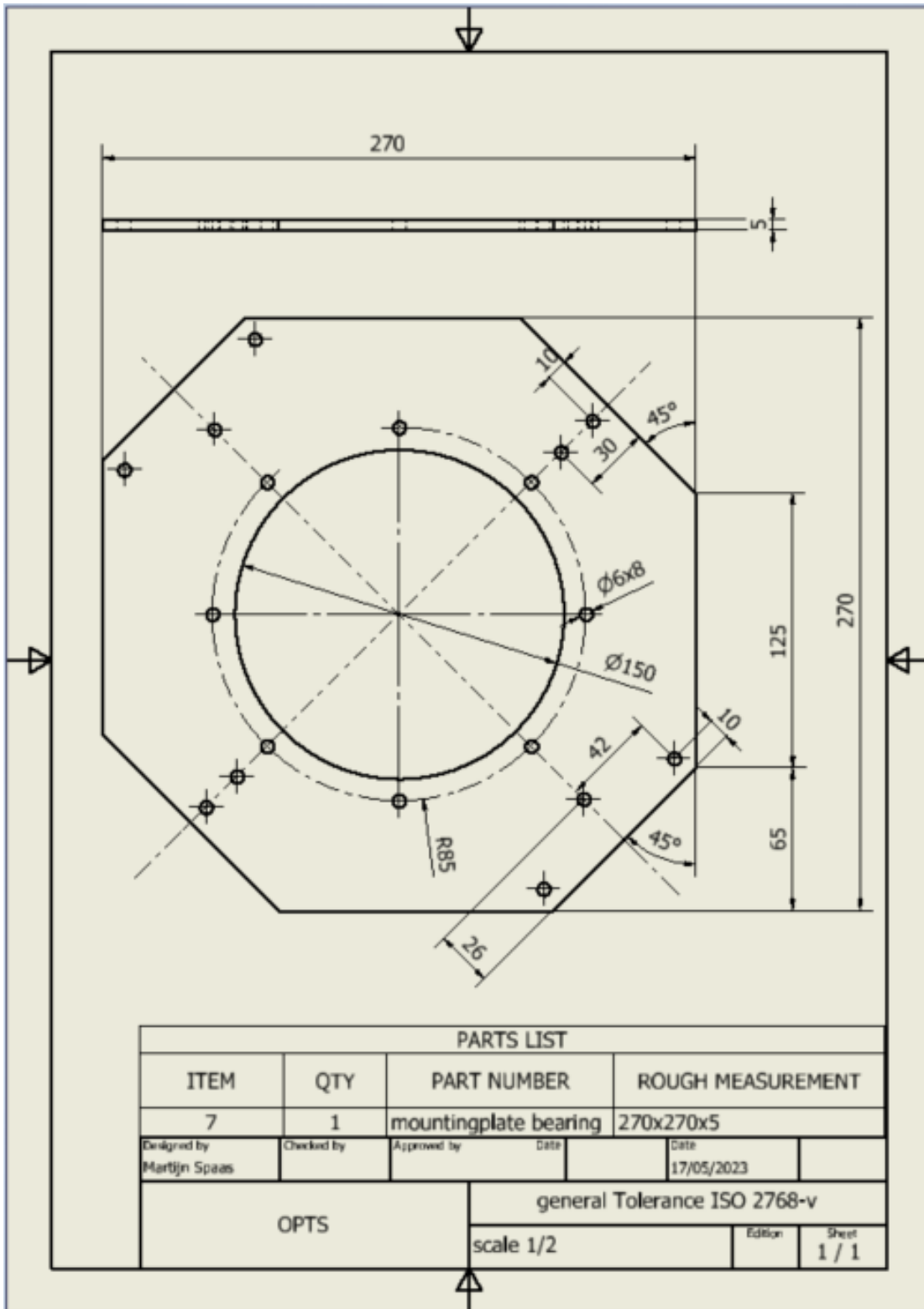
Drawing 6

Welded mount Inside plate. This piece bolts to the “mounting plate bearing” and will be welded to “actuator welded mount”.



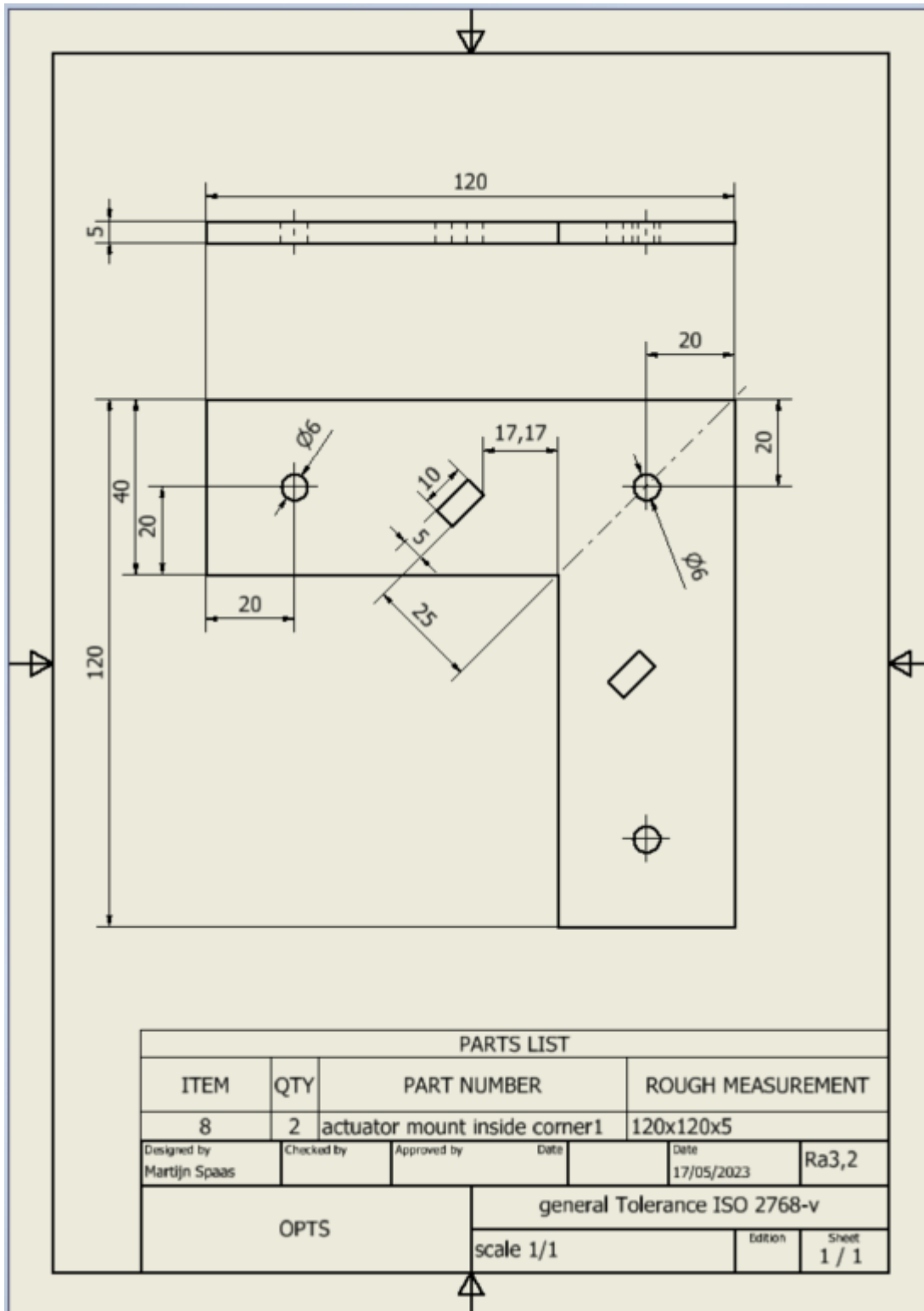
Drawing7

Mountingplate bearing. Place where the “angled base” will be mounted to, bearing “pillow block”, and bolted to welded mount inside plate.



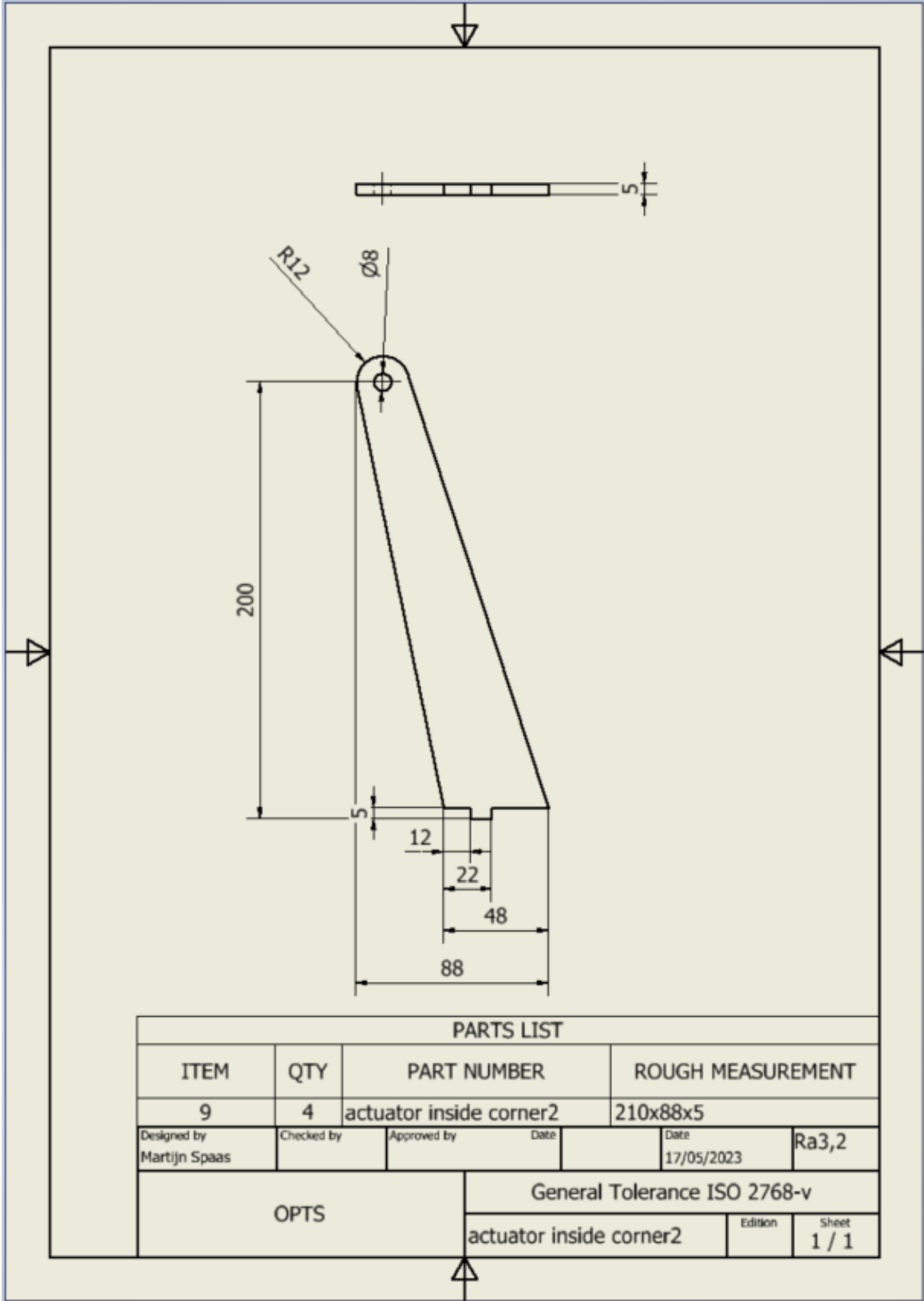
Drawing 8

Actuator mount inside corner1. Wil hold the aluminium extrusions in place like “corner reinforcement for axis”. And will be welded to “actuator inside corner2”



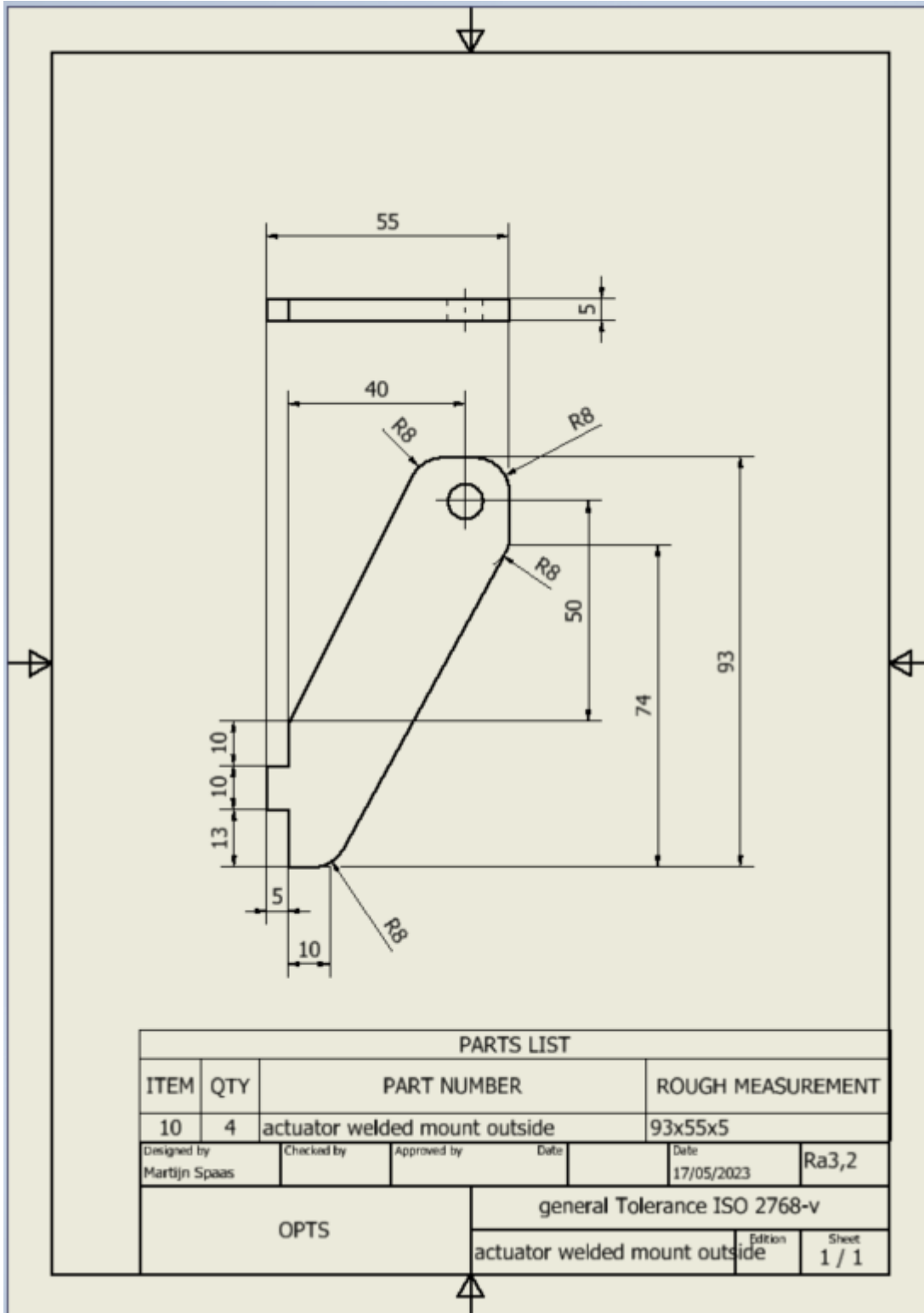
Drawing 9

Actuator inside corner 2. Will be welded to “actuator mount inside corner.



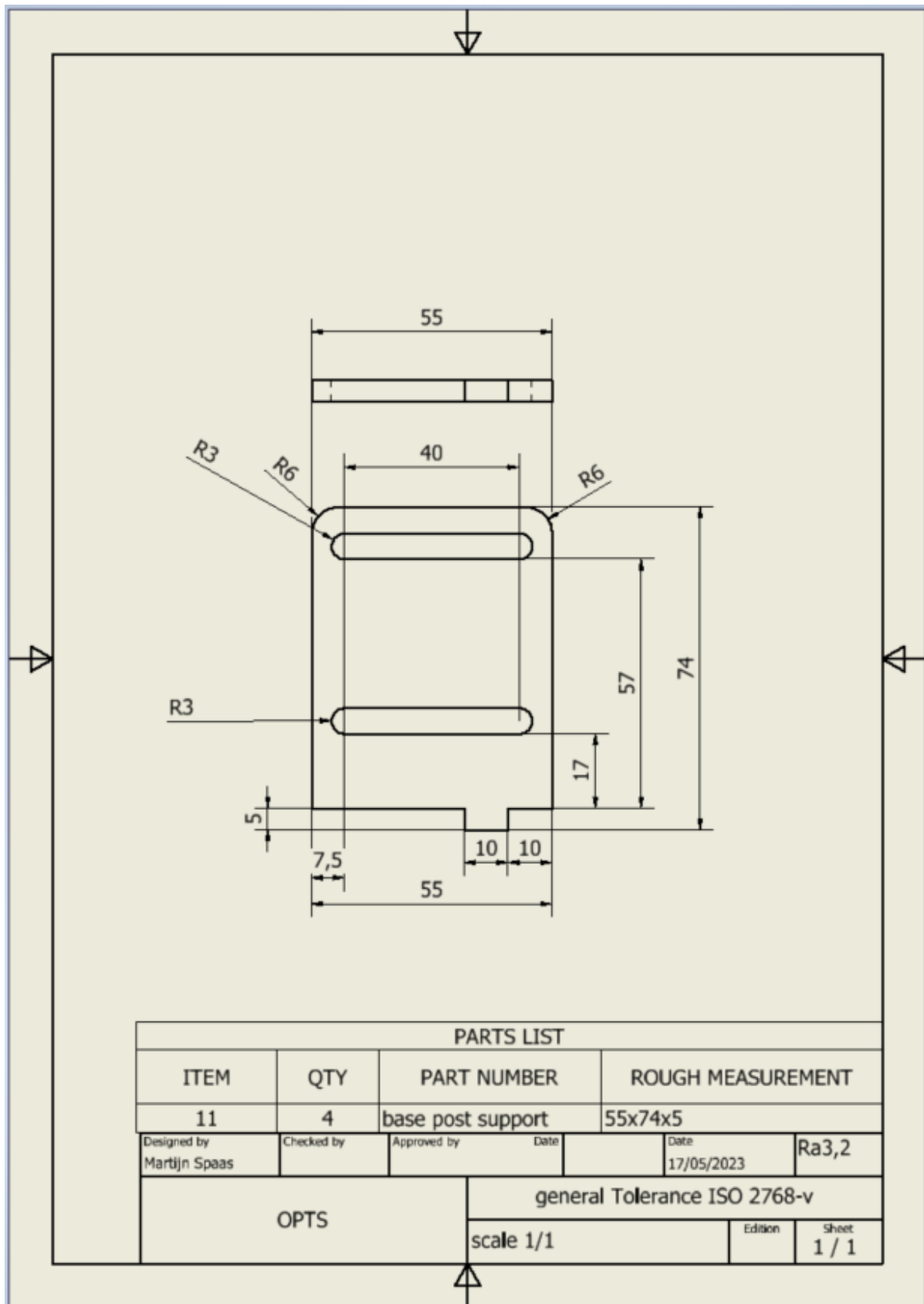
Drawing 10

Actuator welded mount outside. Will be welded to “corner reinforcement for axis”.



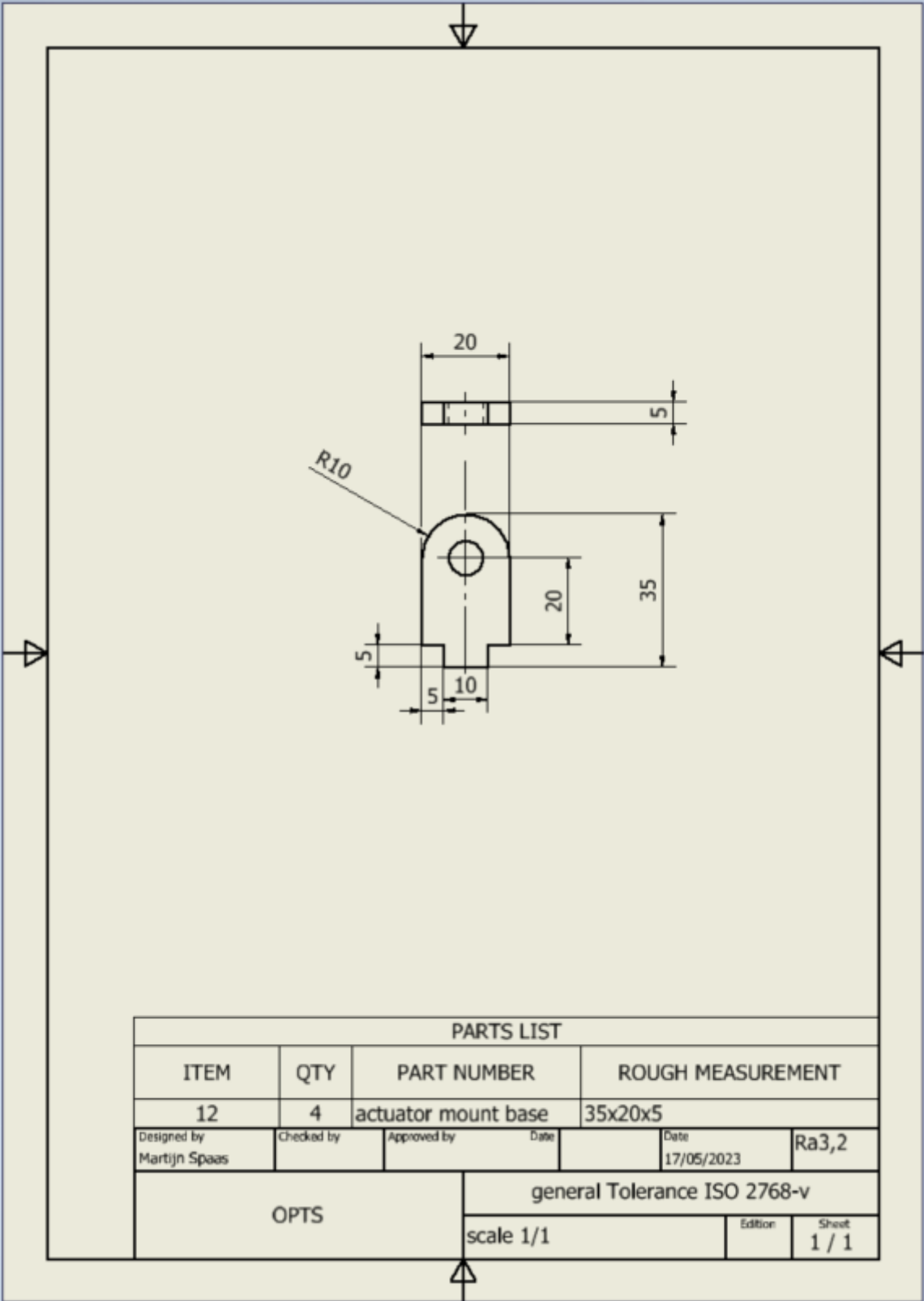
Drawing 11

Base post support. Welds to base post sub assembly. Connects the post with adjustable slots.



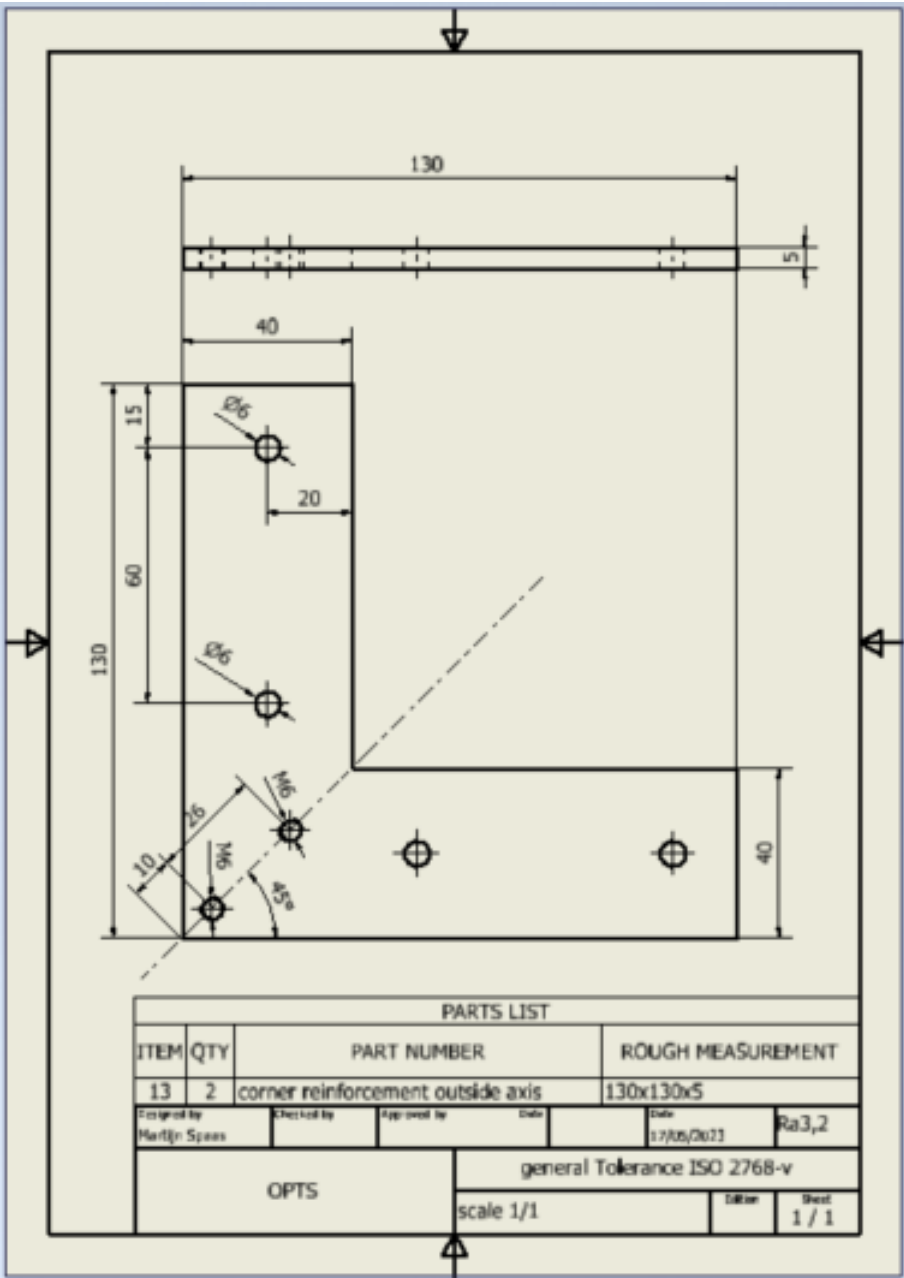
Drawing 12

Actuator base mount. Will be welded to: "Pivot base plate".



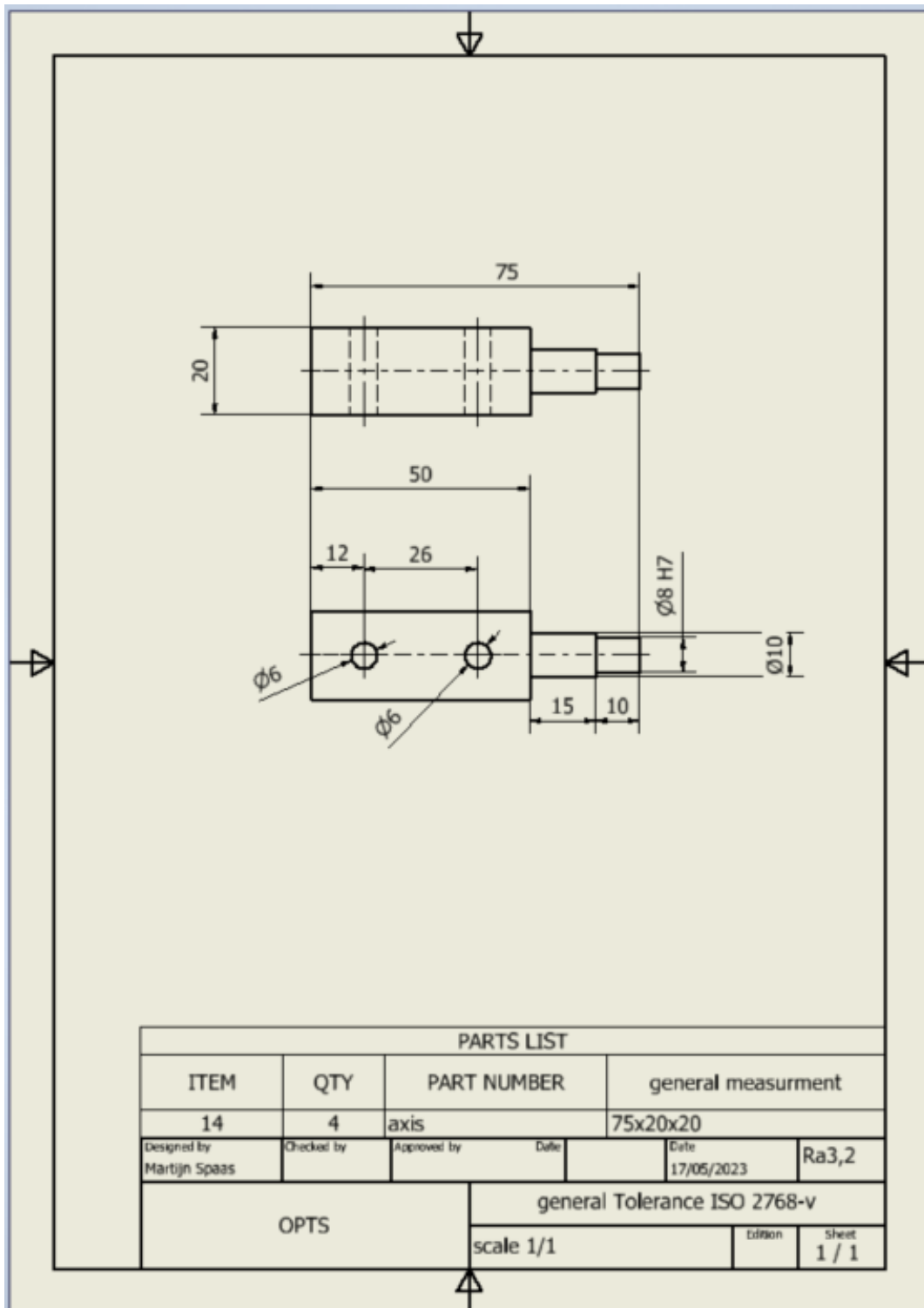
Drawing 13

Corner reinforcement outside axis. Keeps the aluminium extrusions together. Connects with “axis” through threaded holes.



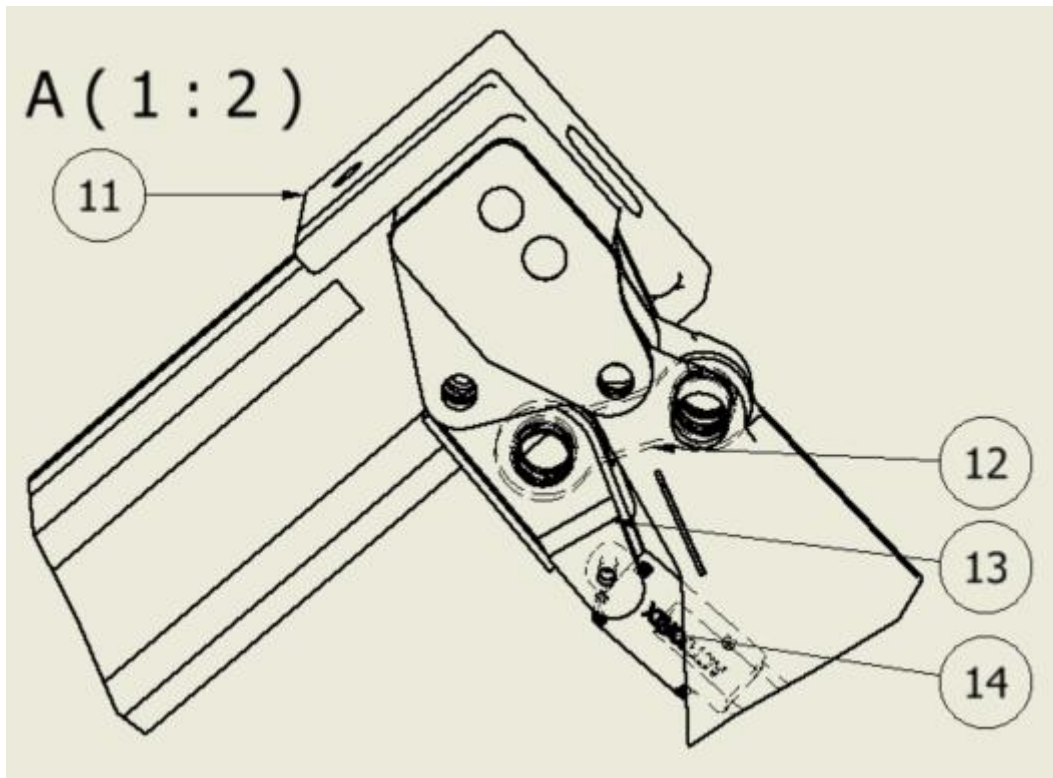
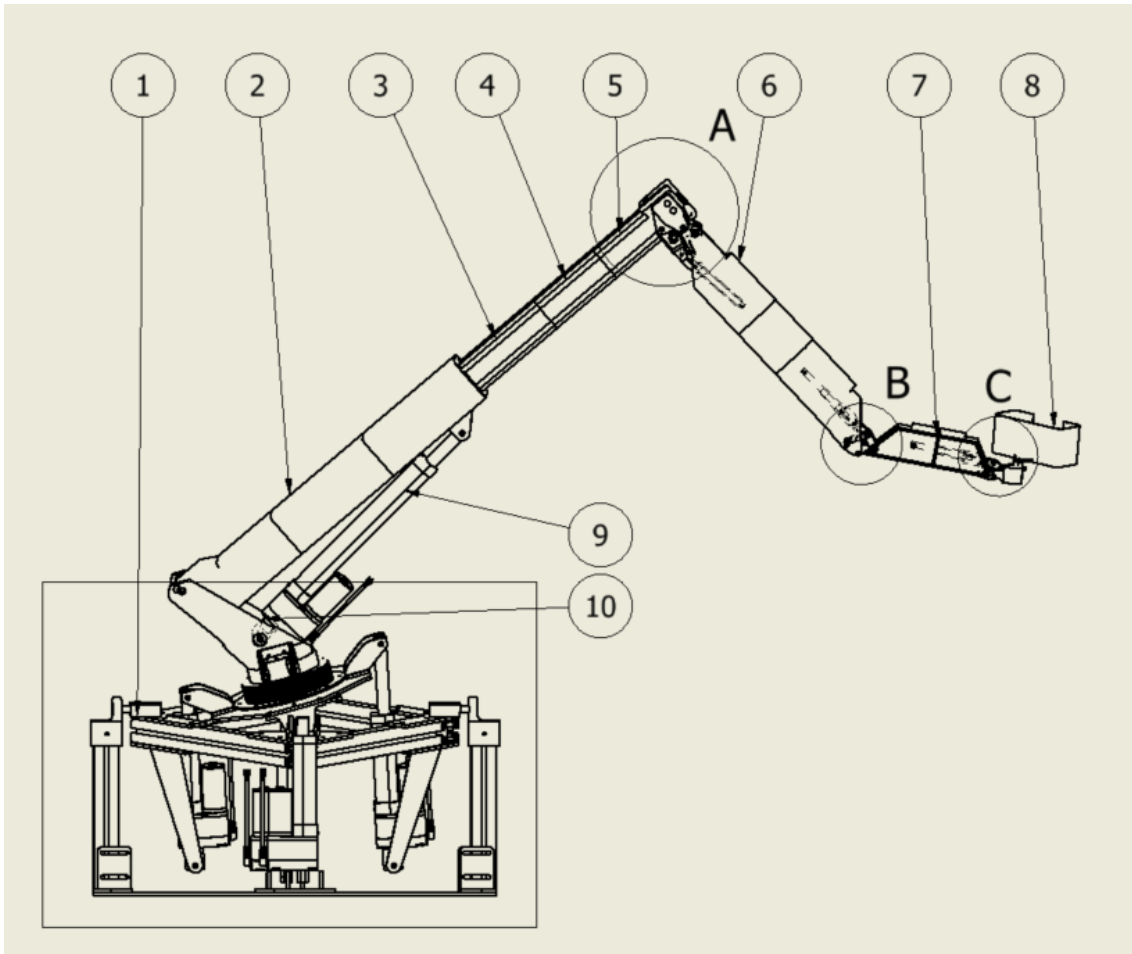
Drawing machined

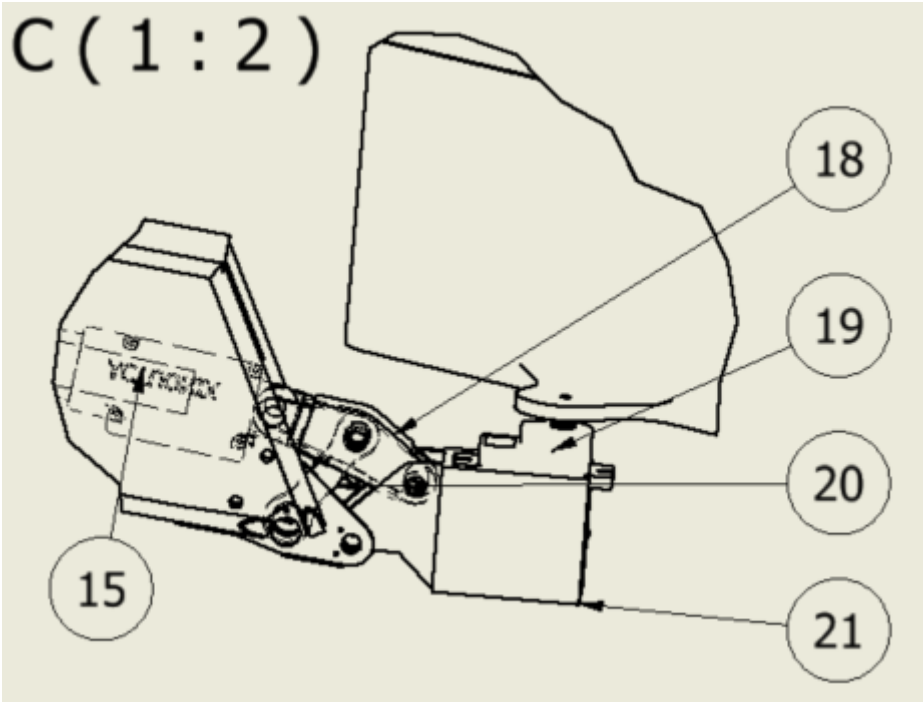
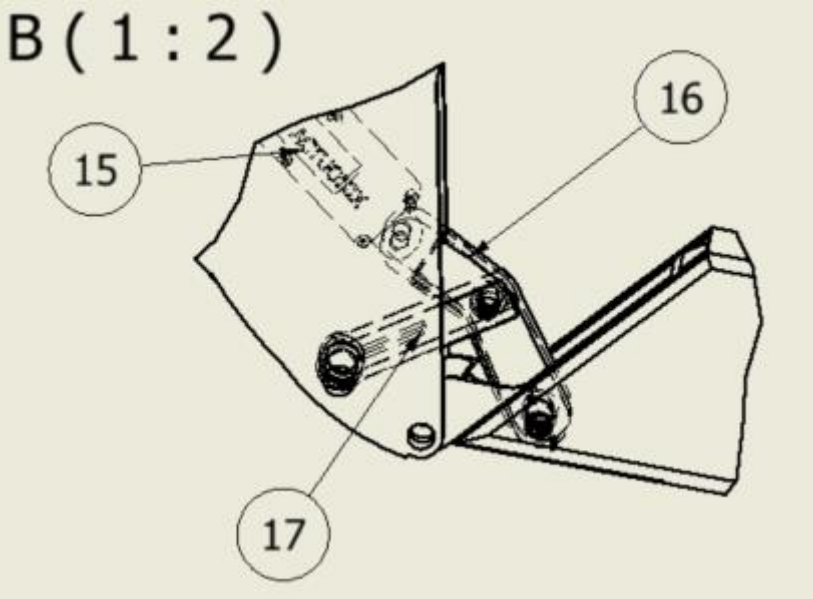
Axis. Will be used as axis together with metal ball bearings. Bolt to “Corner reinforcement outside axis” and “Corner reinforcement for axis”.



Drawing assembly crane

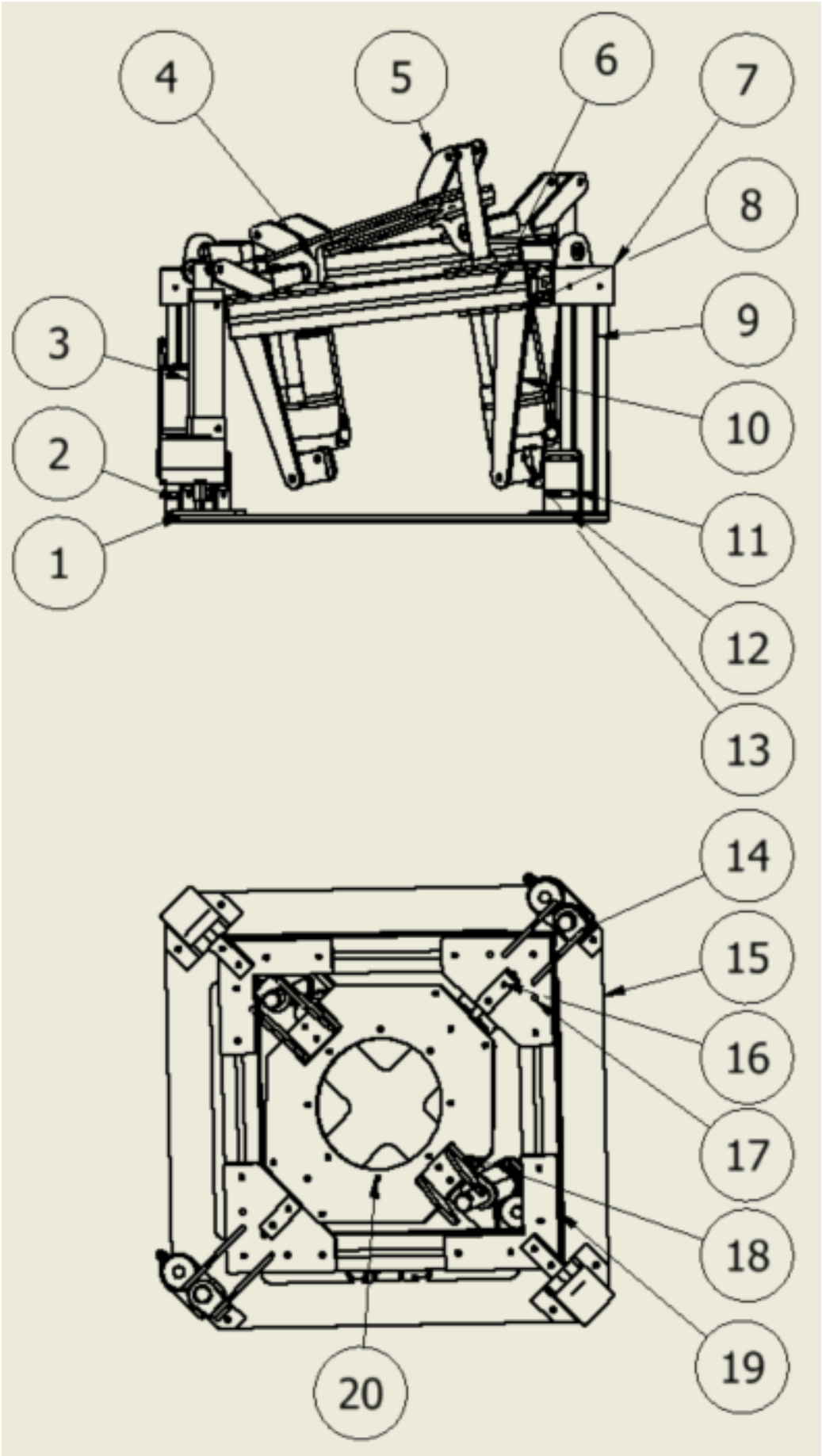
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	base with turn mechanism	
2	1	N_Outer_telescop_SPAJA	
3	1	N_Inner_telescopic_pinion_split1_SPAJA	
4	1	N_Inner_telescopic_pinion_split2_SPAJA	
5	1	N_Inner_telescopic_pinion_split3_SPAJA	
6	1	Truss1_small_wall_original_SPAJA v2	
7	1	Truss2_small_wall_extended_SPAJA	
8	1	basket	
9	1	Actuator_long_SPAJA	CAHB-10 Series-b-Linear Actuator
10	1	Long_actuator_joint2_SPAJA	
11	1	Inner_boom_tip_SPAJA_V2	
12	1	Truss_Connection 2 new	
13	1	Truss_Connection 1 new	
14	1	L12Actuators100mm_SPAJA	STEP AP242
15	2	L12Actuators50mm_SPAJA	STEP AP242
16	1	Truss_Connection2_small_SPAJA	
17	1	Truss_Connection2_1_small_SPAJA	
18	1	Truss_Connection3_1_small_SPAJA	
19	1	SG90 - Micro Servo 9g - Tower Pro.1	
20	1	Truss_Connection3_2_small_SPAJA	
21	1	basket mount	



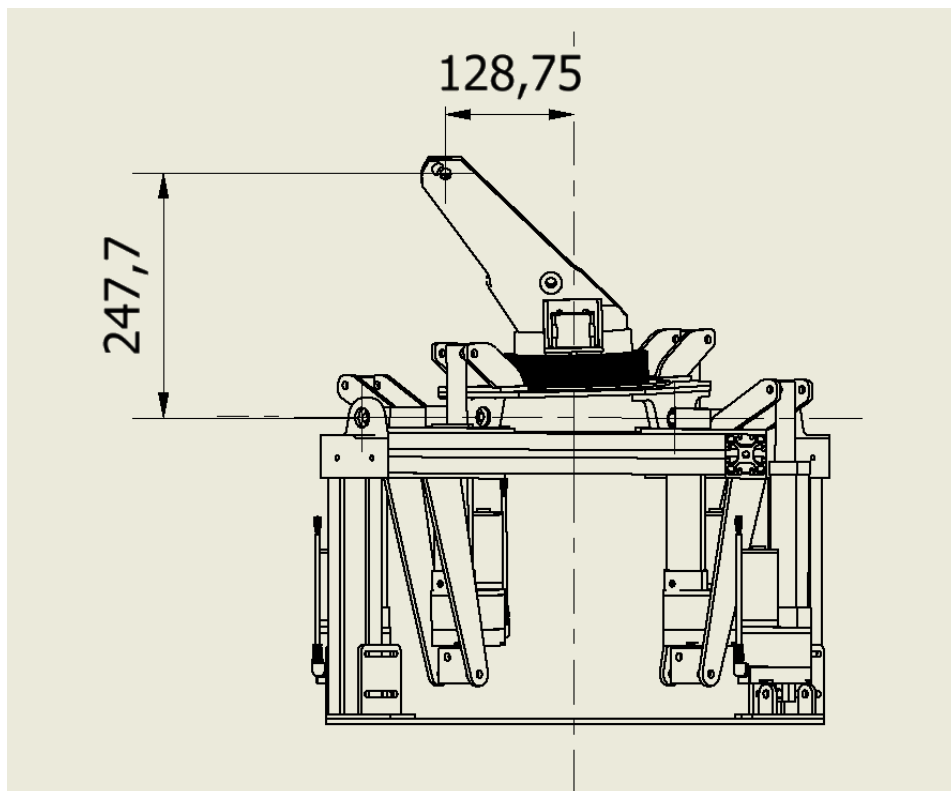
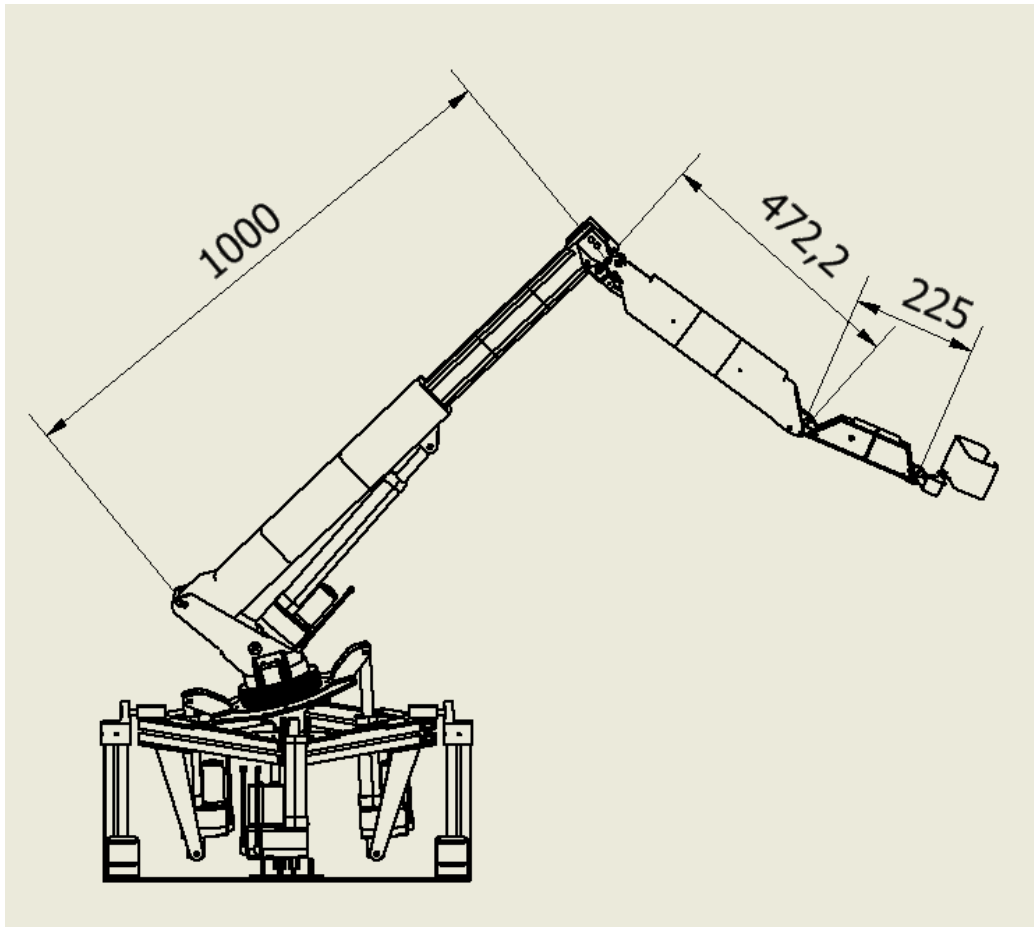


Drawing assembly base.

PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	2	pivot base plate	
2	4	actuator mount base	
3	4	actuator	CAHB-10 Series-b-Linear Actuator
4	2	bearing 608 pillow block	
5	4	actuator welded mount	
6	4	alu extrusion	
7	2	base bearing support	
8	2	actuator mount inside corner1	
9	2	extrusion base support	
10	4	actuator inside corner2	
11	4	base post support	
12	2	base post sub assembly	
13	2	adapter	
14	4	actuator welded mount outside	
15	1	baseplate	
16	4	axis	
17	2	corner reinforcement for axis	
18	2	welded mount inside plate	
19	2	corner reinforcement outside axis	
20	1	mountingplate bearing	



Crane limbs measurements



Arduino programs

Arduino 1: Base and first joint

```
1  const int Speed = 255; //0-255 to ajust the speed of the motors, needs to be variable in the future
2  int ReadButtonx = 0;
3  int ReadButtony = 0;
4  int Delay = 100;
5
6  bool FirstJointUp = false;
7  bool FirstJointDown = false;
8
9  void setup() {
10     analogWrite(10,0);
11     analogWrite(11,0);
12     Serial.begin(9600);
13     FirstJointDown = digitalRead(4);
14     FirstJointUp = digitalRead(2);
15     delay(1000);
16
17 }
18
19 void loop() {
20     delay(Delay);
21
22     ReadButtonx = analogRead(A0);
23
24     Serial.print("Button x =");
25     Serial.println(ReadButtonx);
26     Serial.print("Button y =");
27     Serial.println(ReadButtony);
28     Serial.print("Button First Joint =");
29     Serial.println(digitalRead(2));
30
31     //x direction
32     if (ReadButtonx >= 900) //Up,RPWM
33     {
34         analogWrite(6, Speed);
35         ReadButtony = analogRead(A1);
36     }
37     else if (ReadButtonx > 450 and ReadButtonx < 900) //Down,LPWM
38     {
39         analogWrite(9, Speed);
40         ReadButtony = analogRead(A1);
41     }
42     else
43     {
44         analogWrite(6, 0);
45         analogWrite(9,0);
46         ReadButtony = analogRead(A1);
47         //digitalWrite(R1_Enable, LOW); //Hardware HIGH
48         //digitalWrite(L1_Enable, LOW);
49     }
50
51
52     //y-direction
53     if (ReadButtony >= 900) //Up,RPWM
54     {
55         analogWrite(3, Speed);
56         ReadButtonx = analogRead(A0);
57     }
58     else if (ReadButtony > 450 and ReadButtony < 900) //Down,LPWM
59     {
60         analogWrite(5, Speed);
61         ReadButtonx = analogRead(A0);
62     }
}
```

```
63 else
64 {
65   analogWrite(3, 0);
66   analogWrite(5,0);
67   ReadButtonx = analogRead(A0);
68   ReadButtony = analogRead(A1);
69 }
70
71
72 FirstJointUp = digitalRead(2);
73 FirstJointDown = digitalRead(4);
74
75 Serial.print(FirstJointUp);
76 Serial.println(FirstJointDown);
77 //First Joint up
78 if (FirstJointUp == true && FirstJointDown == false)
79 {
80   analogWrite(10, Speed);
81   FirstJointUp = digitalRead(2);
82   FirstJointDown = digitalRead(4);
83 }
84 //First Joint Down
85 else if (FirstJointDown == true && FirstJointUp == false)
86 {
87   analogWrite(11, Speed);
88   FirstJointUp = digitalRead(2);
89   FirstJointDown = digitalRead(4);
90 }
91 //if both are on or both are off, don't move
92 else
93 {
94   analogWrite(11,0);
95   analogWrite(10,0);
96   FirstJointUp = digitalRead(2);
97   FirstJointDown = digitalRead(4);
98 }
99
100 }
101
```

Arduino 2: Gyroscope

```

1 #include <Wire.h>
2 #include <SPI.h>
3 #include <SparkFunLSM9DS1.h>
4 #include <math.h>
5 #include "AD5593R.h"
6
7 //configuring the ADAC module
8 AD5593R AD5593R(23);
9 bool my_DACs[8] = {1,1,1,1,1,1,1,1}; //Choose which channels need to be configured as Digital Analog Convertors(1 is select)
10
11 LSM9DS1 imu;
12
13 ////////////////////////////////////////////////////
14 // Example I2C Setup //
15 ////////////////////////////////////////////////////
16 // SDO_XM and SDO_G are both pulled high, so our addresses are:
17 #define LSM9DS1_M 0x1E // Would be 0x1C if SDO_M is LOW
18 #define LSM9DS1_AG 0x6B // Would be 0x6A if SDO_AG is LOW
19 #define SAMPLERATE_DELAY_MS (300)
20
21 static unsigned long lastPrint = 0; // Keep track of print time
22 float roll = 0;
23 float pitch = 0;
24 float rollV = 0;
25 float pitchV = 0;
26
27 void mapfloatx(float x, float in_min, float in_max, float out_min, float out_max); //void for converting the signal from radians to 5V signal.
28 void mapfloaty(float y, float in_min, float in_max, float out_min, float out_max); //map function from arduino is only int, so I made a float mapping function.
29
30 void setup()
31 {
32   Serial.begin(9600);
33
34   Wire.begin();
35
36   if (imu.begin() == false) // with no arguments, this uses default addresses (AG:0x6B, M:0x1E) and i2c port (Wire).
37   {
38     Serial.println("Failed to communicate with LSM9DS1.");
39     Serial.println("Double-check wiring.");
40     Serial.println("Default settings in this sketch will " \
41     | "work for an out of the box LSM9DS1 " \
42     | "Breakout, but may need to be modified " \
43     | "if the board jumpers are.");
44     while (1);
45   }
46   delay(1000);
47
48
49   AD5593R.enable_internal_Vref();
50   AD5593R.set_DAC_max_2x_Vref(); //Set the max channel voltage to 2 times the internal voltage
51   AD5593R.configure_DACs(my_DACs); //Configure chosen channels to Digital Analog Convertors
52   //AD5593R.set_Vref(5);
53 }
54

```

```
55 void loop()
56 {
57   imu.readAccel();
58
59   roll = atan2(imu.ay, imu.az);
60   //pitch = atan2(-imu.ax, sqrt(imu.ay * imu.ay + imu.az * imu.az));
61   pitch = atan2(imu.ax, imu.az);
62   //Serial.print(imu.ax);
63
64   mapfloatx(roll, -0.45, 0.45, 0, 255); // do the map functions
65   mapfloaty(pitch, -0.45, 0.45, 0, 255);
66
67   Serial.print(rollV);Serial.print(",");
68   Serial.println(pitchV);
69
70   analogWrite(3, rollV);
71   analogWrite(5, pitchV);
72
73   //AD5593R.write_DAC(0, rollV); //with ADAC click module
74   //AD5593R.write_DAC(2, pitchV);
75
76   //delay(SAMPLERATE_DELAY_MS);
77 }
78
79 void mapfloatx(float x, float in_min, float in_max, float out_min, float out_max)
80 {
81   rollV = (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
82 }
83
84 void mapfloaty(float y, float in_min, float in_max, float out_min, float out_max)
85 {
86   pitchV = (y - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
87 }
```

Arduino 3: Steppers + servo

```
1  #include "HCMotor.h" //library stepper motors
2  #include <Servo.h>   //library servo
3
4  #define dirPin_step1 3
5  #define stepPin_step1 2
6  #define dirPin_step2 5
7  #define stepPin_step2 4
8  #define servo 6
9
10 //inputs
11 int Up1 = 7;    //stepper1
12 int Down1 = 8;
13 int Up2 = 9;    //stepper2
14 int Down2 = 10;
15 int Up3 = 11;   //servo
16 int Down3 = 12;
17
18 bool up1 = 0;   //stepper1
19 bool down1 = 0;
20 bool up2 = 0;   //stepper2
21 bool down2 = 0;
22 bool up3 = 0;   //servo
23 bool down3 = 0;
24
25 Servo servo1;   //servo speed and starting angle
26 int angle = 90;
27 int angleStep = 1;
28 HCMotor HCMotor;
29
30 void setup() {
31   Serial.begin(9600);
32   HCMotor.Init();
33   // put your setup code here, to run once:
34   HCMotor.attach(0, STEPPER, stepPin_step1, dirPin_step1);
35   HCMotor.attach(1, STEPPER, stepPin_step2, dirPin_step2);
36
37   HCMotor.Steps(0,CONTINUOUS);
38   HCMotor.Steps(1,CONTINUOUS);
39
40   pinMode(7, INPUT_PULLUP);
41   pinMode(8, INPUT_PULLUP);
42   pinMode(9, INPUT_PULLUP);
43   pinMode(10, INPUT_PULLUP);
44   pinMode(11, INPUT_PULLUP);
45   pinMode(12, INPUT_PULLUP);
46   servo1.attach(6);
47 }
48
49 void loop() {
50   Serial.print (Up2);
51   Serial.println (Down2);
52   int Speed = 1024;
53   up1 = digitalRead(Up1);
54   down1 = digitalRead(Down1);
55   up2 = digitalRead(Up2);
56   down2 = digitalRead(Down2);
57   up3 = digitalRead(Up3);
58   down3 = digitalRead(Down3);
59
```

```
60 Speed = 0;
61   servo1.write (angle);
62
63   if (up3 == 0 && down3 == 1 && angle < 180)
64   | {
65   |   angle = angle + angleStep;
66   |   delay (50);
67   | }
68   else if (up3 == 1 && down3 == 0 && angle > 0) {
69   |   angle = angle - angleStep;
70   |   delay (50);
71   | }
72   | if (up1 == HIGH && down1 == LOW) {
73   | | HCMotor.Direction(0, REVERSE);
74   | | Speed = 100;
75   | | }
76
77   | else if (up1 == LOW && down1 == HIGH) {
78   | | HCMotor.Direction(0, FORWARD);
79   | | Speed = 100;
80   | | }
81
82   | else if (up2 == HIGH && down2 == LOW) {
83   | | HCMotor.Direction(1, REVERSE);
84   | | Speed = 100;
85   | | }
86
87   | else if (up2 == LOW && down2 == HIGH) {
88   | | HCMotor.Direction(1, FORWARD);
89   | | Speed = 100;
90
91   HCMotor.DutyCycle(0, Speed);
92   HCMotor.DutyCycle(1, Speed);
93
94 }
95 }
```

Arduino 4: Linear actuators arm

```
1  int motor1pin1 = 4;
2  int motor1pin2 = 7;
3  int motor2pin1 = 8;
4  int motor2pin2 = 10;
5  int motor3pin1 = 11;
6  int motor3pin2 = 12;
7
8
9
10 int ReadButtonSecondJoint;
11 int ReadButtonThirdJoint;
12
13 bool ForthJointUp ;
14 bool ForthJointDown ;
15
16 const int Speed = 255;
17
18
19 void setup() {
20     | | // define outputs
21
22     pinMode(motor1pin1, OUTPUT);
23     pinMode(motor1pin2, OUTPUT);
24     pinMode(motor2pin1, OUTPUT);
25     pinMode(motor2pin2, OUTPUT);
26     pinMode(motor3pin1, OUTPUT);
27     pinMode(motor3pin2, OUTPUT);
28
29     pinMode(3, OUTPUT);
30     pinMode(5, OUTPUT);
31     pinMode(6, OUTPUT);
32
33     | | //define digital inputs with internal pullup resistor.
34
35     pinMode (2, INPUT_PULLUP);
36     pinMode (13, INPUT_PULLUP);
37
38     delay (1000);
39 }
40
41 void loop() {
42
43
44     | | | | | | | | | | //reading plc controls
45     ReadButtonSecondJoint = analogRead(A0); //second joint
46
47     ReadButtonThirdJoint = analogRead(A1); //third joint
48
49     ForthJointUp = digitalRead(2); //fourthjoint
50     ForthJointDown = digitalRead(13);
51
52     if (ReadButtonSecondJoint >= 900) //Up,RPWM
53     {
54         analogWrite(3, Speed); //write pwm signal to pin 3.
55         digitalWrite(motor1pin1, HIGH); //setting the direction
56         digitalWrite(motor1pin2, LOW);
57     }
58     else if (ReadButtonSecondJoint > 450 and ReadButtonSecondJoint < 900) //Down,LPWM
59     {
60         analogWrite(3, Speed);
61         digitalWrite(motor1pin1, LOW);
62         digitalWrite(motor1pin2, HIGH);
63
64     }
65     else (ReadButtonSecondJoint < 450)
66     {
67         analogWrite(3, 0);
68         digitalWrite(4, LOW);
69         digitalWrite(7, LOW);
```



```
70 |  
71 | }  
72 |  
73 | //ThirdJoint  
74 | if (ReadButtonThirdJoint >= 900)  
75 | {  
76 |     analogWrite(5, Speed);  
77 |     digitalWrite(8, HIGH);  
78 |     digitalWrite(10, LOW);  
79 | }  
80 | }  
81 | else if (ReadButtonThirdJoint > 450 and ReadButtonThirdJoint < 900)  
82 | {  
83 |     analogWrite(5, Speed);  
84 |     digitalWrite(8, LOW);  
85 |     digitalWrite(10, HIGH);  
86 | }  
87 | else  
88 | {  
89 |     analogWrite(5, 0);  
90 |     digitalWrite(8, LOW);  
91 |     digitalWrite(10, LOW);  
92 | }  
93 |  
94 | //ForthJoint  
95 |  
96 | //Forth Joint up  
97 | if (ForthJointUp == true && ForthJointDown == false)  
98 | {  
99 |     analogWrite(6, Speed);  
100 |     digitalWrite(11, HIGH);  
101 |     digitalWrite(12, LOW);  
102 | }  
103 | //Forth Joint Down  
104 | else if (ForthJointDown == true && ForthJointUp == false)  
105 | {  
106 |     analogWrite(6, Speed);  
107 |     digitalWrite(11, LOW);  
108 |     digitalWrite(12, HIGH);  
109 | }  
110 | //if both are on or both are off, don't move  
111 | else  
112 | {  
113 |     analogWrite(6,0);  
114 |     digitalWrite(11, LOW);  
115 |     digitalWrite(12, LOW);  
116 | }  
117 |  
118 |  
119 | }
```

Table wiring connections

Module	PLC	Arduino
DQ	FirstJointUp(Q10.0)	Arduino 1: pin 2
DQ	FirstJointDown(Q10.1)	Arduino 1: pin 4
DQ	StepperRotCW(Q10.2)	Arduino 3: pin 7
DQ	StepperRotCCW(Q10.3)	Arduino 3: pin 8
DQ	StepperTelescCW (10.4)	Arduino 3: pin 9
DQ	StepperTelescCCW (10.5)	Arduino 3: pin 10
DQ	ServoCW(Q10.6)	Arduino 3: pin 11
DQ	ServoCCW(Q10.7)	Arduino 3: pin 12
DQ	FourthJointUp(11.0)	Arduino 4: pin 2
DQ	FourthJointDown(11.1)	Arduino 4: pin 13
AQ	BaseUp(QW4)	Arduino 1: A0
AQ	BaseLeft(QW6)	Arduino 1: A1
AQ	SecondJoint(QW0)	Arduino 4: A0
AQ	ThirdJoint(QW2)	Arduino 4: A1
DQ	FourthJointUp()	Arduino 4: pin 2
DQ	FourthJointDown ()	Arduino 4: pin 13
AI	FL_PositionSensor(IW126)	(Directly from actuator)
AI	FR_PositionSensor(IW128)	(Directly from actuator)
AI	FirstJoint_PositionSensor(IW130)	(Directly from actuator)
AI	SecondJoint_PositionSensor(IW132)	(Directly from actuator)
AI	ThirdJoint_PositionSensor(IW134)	(Directly from actuator)
AI	FourthJoint_PositionSensor(IW136)	(Directly from actuator)
AI	SensorXValue(IW138)	Arduino 2: pin
AI	SensorYValue(IW140)	Arduino 2: pin
DI	ButtonServoCCW (I3.7)	(From manual control panel)
DI	ButtonServoCW (I3.6)	(From manual control panel)
DI	ButtonStepperRotCCW (I3.5)	(From manual control panel)

Production, assembly and programming of a RoboCrane

DI	ButtonStepperRotCW (I3.4)	From
DI	ButtonFourthJointDown (I3.3)	(From manual control panel)
DI	ButtonFourthJointUp (I3.2)	(From manual control panel)
DI	ButtonThirdJointDown (I3.1)	(From manual control panel)
DI	ButtonThirdJointUp (I3.0)	(From manual control panel)
DI	ButtonSecondJointDown (I2.7)	(From manual control panel)
DI	ButtonSecondJointUp (I2.6)	(From manual control panel)
DI	ButtonFirstJointDown (I2.5)	(From manual control panel)
DI	ButtonFirstJointUp (I2.4)	(From manual control panel)
DI	ButtonBaseRight (I2.3)	(From manual control panel)
DI	ButtonBaseLeft (I2.2)	(From manual control panel)
DI	ButtonBaseDown (I2.1)	(From manual control panel)
DI	ButtonBatoseUp (I2.0)	(From manual control panel)
DI	StepperTelescoCW (/)	(From manual control panel)
DI	StepperTelescoCCW (/)	(From manual control panel)

