Voluntary Project

SyArm Mk1

A simple robotic arm

By

Samuel Nösslböck

Started

April 2022

Credits

At this point I want to personally express my thanks to all people who made this thing possible. Due to a lot of time and effort having been done in school, I want to thank all teachers who actively supported me or did not force me to fully participate in their lessons.

Special thanks belong to my machine elements teacher Mr. Dipl.-Ing. Manuel Leibetseder, Mr. Dipl.-Ing. Gottfried Preuer and Mr. Dipl.-Ing. Peter Rachinger. First one mentioned even answered calls on the weekend, even though he had more important things to take care about. He always had helpful advice for most of my problems.

Furthermore, I want to mention my incredible friends, as they always listen to my personal problems which arise during such a project. Some of them even tried to listen to some details and always showed interest.

Again, I want to point out a few people: First Tobias Niedermayr, a very talented young man who is way smarter he wants to admit he is. He always showed a high interest in my projects and gave me the motivation to continue when mine was more than low.

Second there is my best friend Fabio Muratore. He just knows how to make me laugh, have a good time and really get my head out of my projects from time to time, which is a very important thing as I have learned in the process.

When working on a voluntary project, motivation is the key for progress. Problem is, such motivation is not always easy to keep up all the time. Therefore, special thanks belong to Samuel Klancnik, a brilliant and very creative artist who inspired me in the way he does his things and who also cared very much about me. He has become a very special person to this project and me in the process.

Finally, my dear friend Rene Schwarz, my classmate. He is the one who always cheers me up and who makes school much more bearable. I have the honour to work on a project with him in the near future, which makes use of his great abilities with AI.

Abstract

Many manufacturing tasks require a series of complex work or transport processes, for which it can be difficult to build a machine for. Robotic arms are very flexible types of robots which can perform a lot of those complex tasks and can be re-programmed for every new application needed.

The SyArm is a simple and compact version of such a robot, not built for heavy loads or accuracy. Its materials are cheap, simple and were mainly chosen by the factor of how easy it is to acquire them.

The whole project can be seen as a kind of learning process, that is the reason why you will find version numbers of three and above. Many of the hand-drawn sketches have very early concepts drawn onto them.

This project description focuses a lot on the theoretical aspects, especially of the software, that have been worked out in the process. Therefore, a lot of formulas and other can be found in the following pages.

Contents

1. Introduction

1.1. Motivation

Robots always fascinated me in many ways: The way they are built, the way the software is made or simply the enormous tasks they can complete nowadays.

I personally believe automation and especially robots are one of the most effective ways to fight poverty. When used right, they can get people out of jobs with miserable working conditions and accelerate technical development especially in low-wage countries.

Building this robot was kind of a first step towards helping people with the field of research I love.

1.2. Goals

The goal is to build a fully functional robotic arm (see section “Construction”) that can equip multiple different tools for tasks like drawing, lifting things around and so on. Other things a high value was put on are the stability against oscillations that will cause an inaccuracy in all movements and paths.

Flexibility is the overall term this construction was designed for, in terms of hardware and software as well. The software has been packed into libraries and separated from this specific robot as much as possible, so it can be used of other types of robots too.

1.3. Structure

The naming and versioning system with “Mk X” makes it easy to differentiate complete reworks of the same idea from each other with its basic principle being the same as for other versioning systems.

Version: X.Y.Z => Mk X, Version Y.Z

The GitHub repository contains all data used for this project. Notice that the software projects have been included as submodules, for further information, read the “Appendix” section.

2. Construction

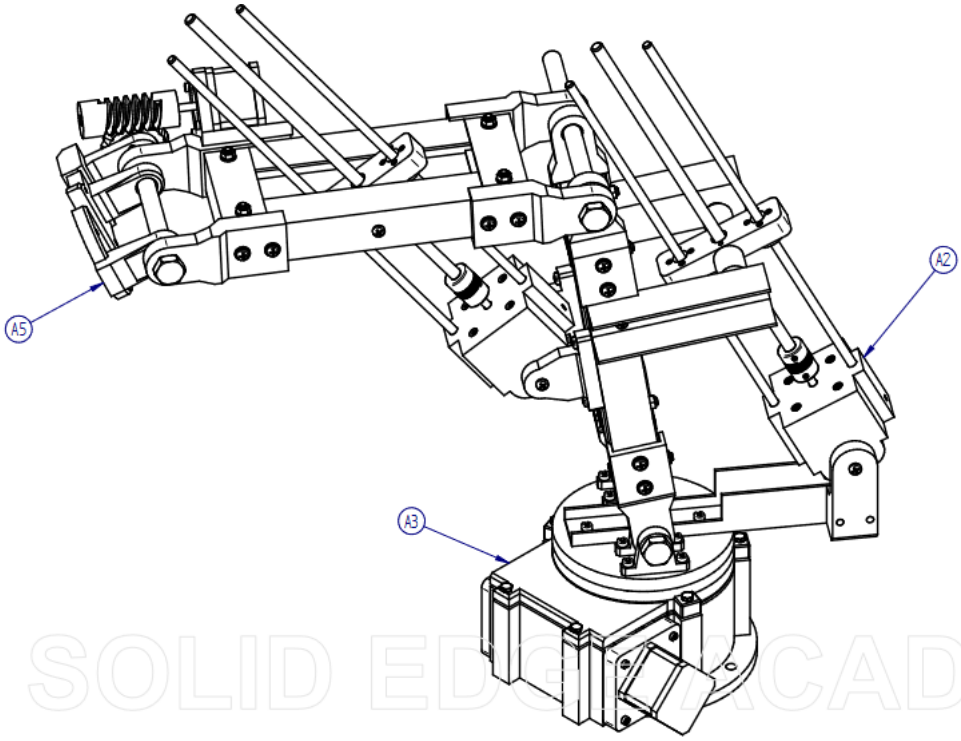

Figure 2.0.0.1

The entire construction is separated into five main parts listed in the following chapters.

2.1. Base

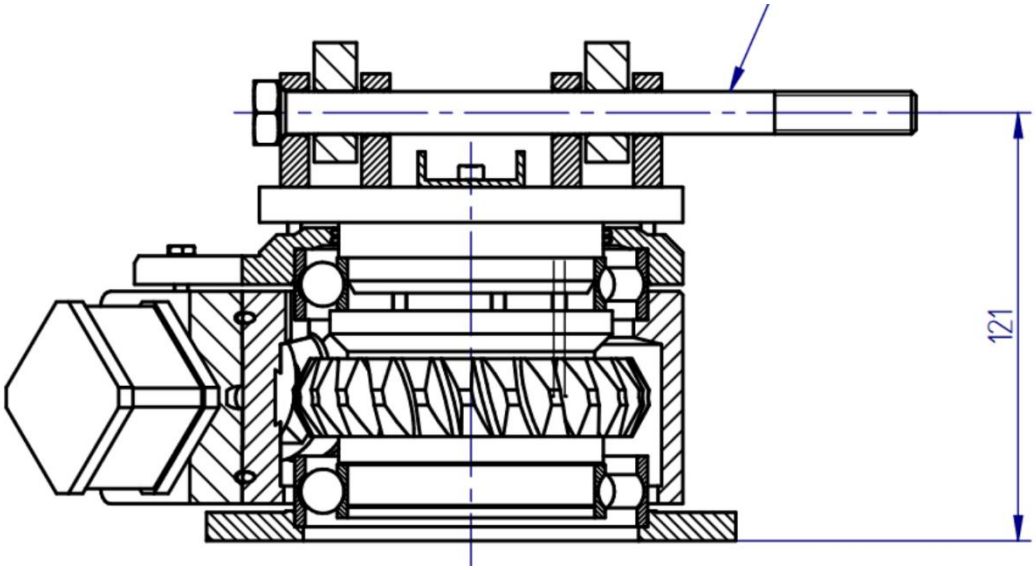

Figure 2.1.0.1

The base is the most important part of the construction when it comes to stability. Small inaccuracies in the bearings can lead to huge oscillations and imprecisions. To combat this, the base uses a combination of two bearings to grant stability even when the arm reaches far out. Between these two bearings, a worm gear transmits the rotation and torque created by the motor. Though to the high transmission ratio, a high accuracy in angles can be accessed.

2.2. Arm

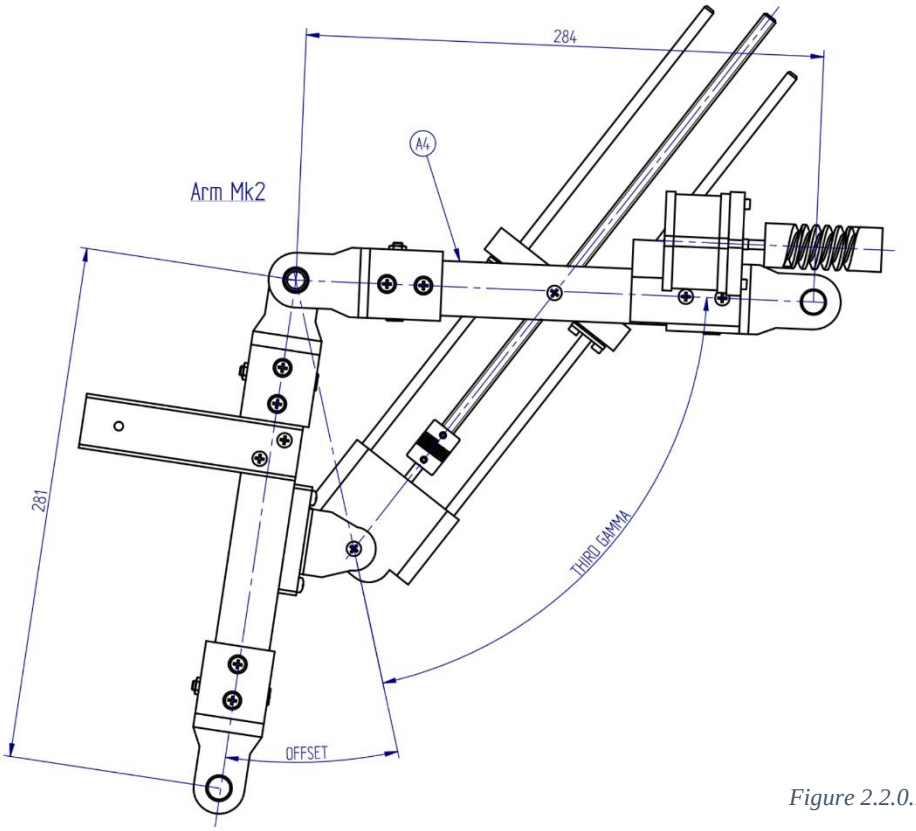

Figure 2.2.0.1

The arm consists of two segments connected by a cylinder, which determine the distance the robot can reach out together. The first segment is the lower one on the picture above, the second segment the upper one it is connected to.

The cylinder connecting these two segments is often referred to as “second cylinder” as the “first cylinder” connects the base and the arm with the aluminium segment on the far left, which is attached to the first segment.

2.2.1. First Segment

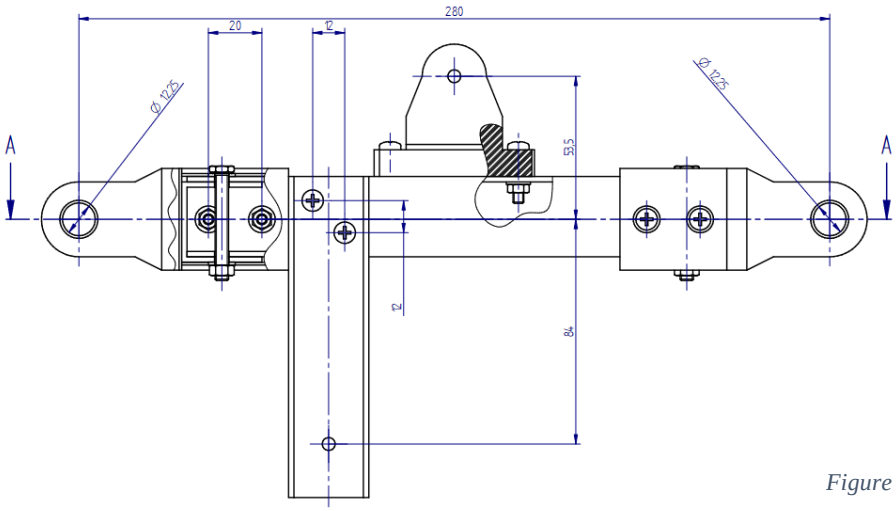

Figure 2.2.1.1

The first segment is connected to the Base with a pin and a cylinder, allowing the controls to adjust the angle. Both segments use the same type of 3D-printed connector. As the friction is low and the robot produces high torques at the pins, no additional lube must be used to guarantee correct functionality.

Furthermore, both segments use the same base frame built up of aluminium segments, called “Simple Frame”.

2.2.2. Second Segment

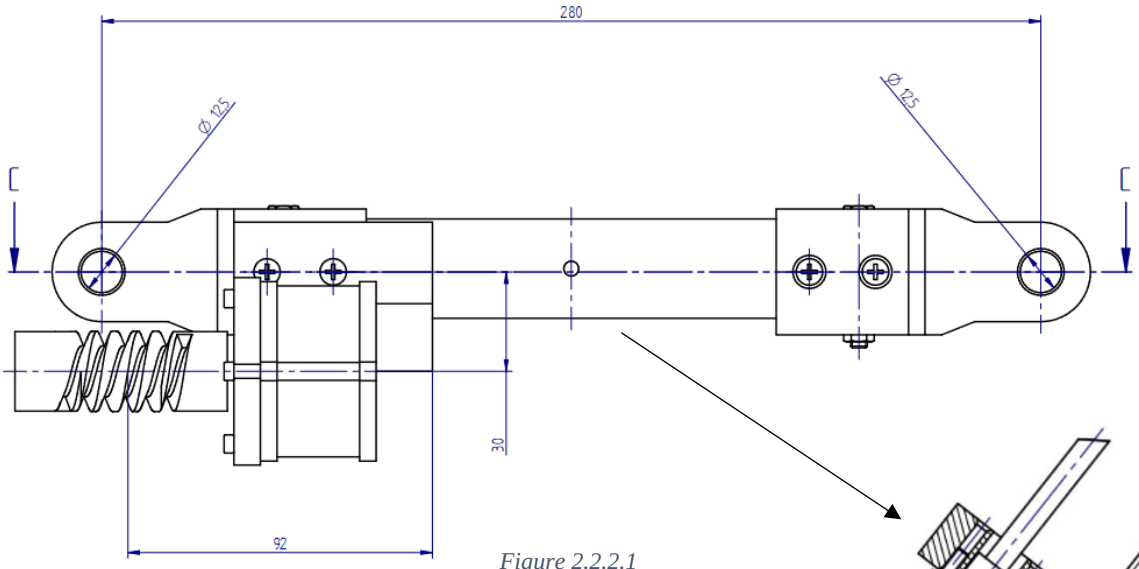

Figure 2.2.2.1

The second segment is equipped with the motor in order to control the last axis and mount for the second cylinder, as shown on the right.

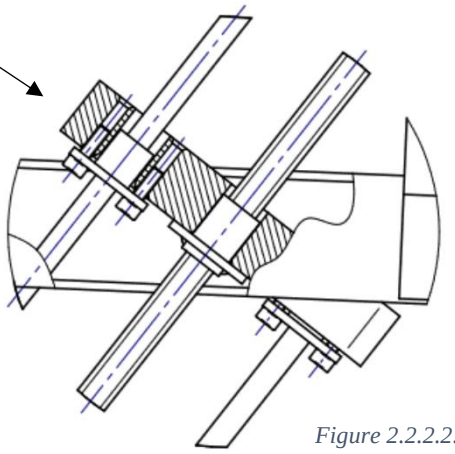

Figure 2.2.2.2.

It consists, just like the first segment, of a “Simple Frame” with four 3D-printed connectors on the edges. The mount for the last stepper motor is mounted onto one of them, as shown in the picture above.

2.3. Tool-Joint

The tool joint is the last axis, which theoretically would not be required, yet it allows the robot to pick up objects from a desired angle. The flange at the end has multiple threaded holes, where many different tools can be mounted.

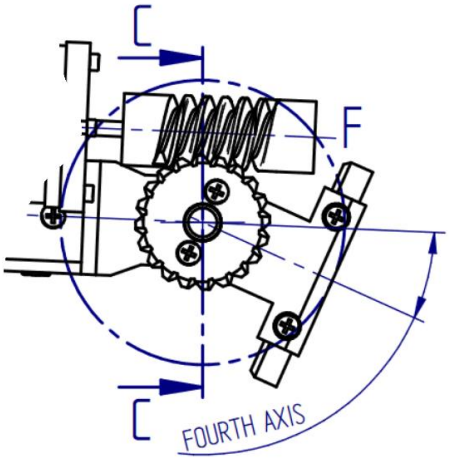

Figure 2.3.0.1

2.4. Tools

To cover the widespread uses of a robotic arm, different tools for various sizes can be used. Some tools even allow to mount yet another tool, for example the axial bearing. It allows the robot to use a simple fifth axis for rotating a tool. In combination with tongs or a screwdriver this could even be used to assemble basic parts in future, more stable and accurate versions.

2.5. Cylinders

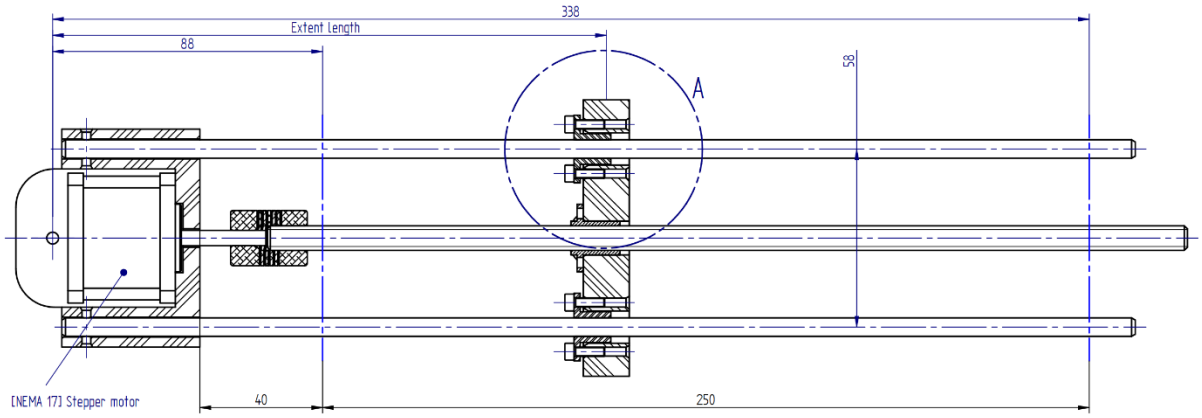

Figure 2.5.0.1

The cylinders used for the robot consist of a few simple parts: A static part, where the stepper motor is mounted, a moveable part where all the rods are passed through. Two rods and a spindle, with the latter being mounted to the motor through a jaw coupling, to compensate assembly inaccuracies.

The two rods guarantee a consistent and stable extension of the cylinder, while the spindle is between them, extending or shortening the cylinder.

2.6. Simple Frame

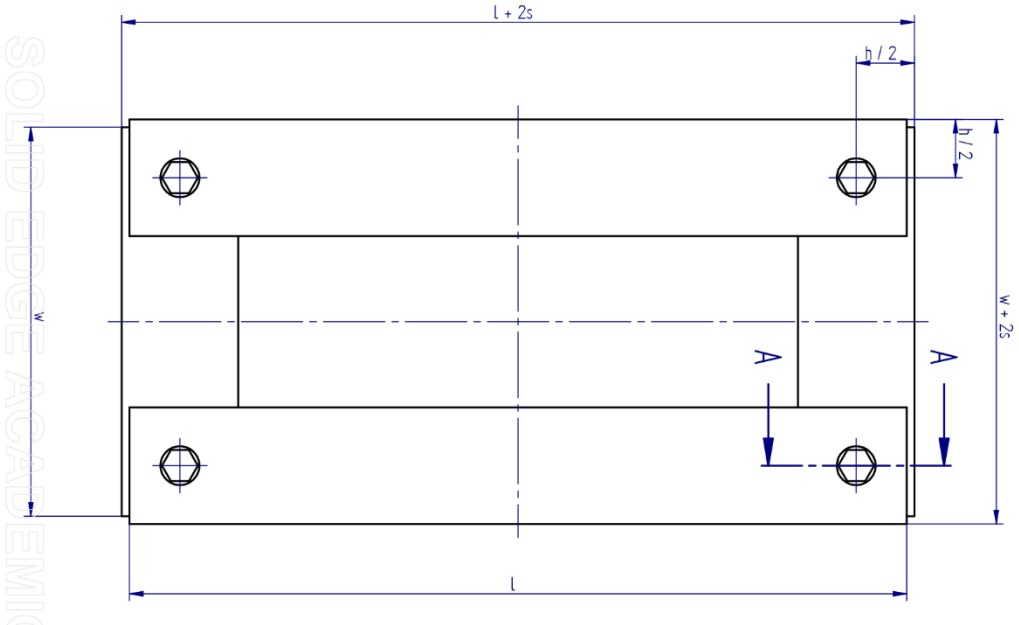

Figure 2.6.0.1

The simple frame is made from four aluminium segments, that create a stable base for the arm segments.

3. Electronics

3.1. Controller

3.1.1. Stepper motor connections

As all stepper motors used are bipolar ones with two coils, four wires need to be managed for each motor. The cables are attached to plugs that can be easily disconnected if required for transportation. Each stepper motor is connected to its own stepper controller, turning the four wires, which commonly would need to be supplied with voltage in different cycles to get different directions and movements, into two, that can both be controlled with simple low voltage logical signals. With the first one requiring pulses, one for each step to be taken, and the other being a directional switch.

3.1.2. Raspberry PI connection

The Raspberry PI controlling the robot is also connected to the controls with a few plugs, enabling easy repairs and reconfiguration. All the GPIO pins are defined as constants that can be modified in the setup process. Thus, if any other pin layout is desired, one can change it easily.

```
"name": "Base",
"type_name": "stepper_lib::comp::gear_bearing::GearBearing",
"obj": {
  "ctrl": {
    "consts": "MOT_17HE15_1504S",
    "pin_dir": 17,
    "pin_step": 26
  },
```

Figure 3.1.2.1

The different types of pins are always grouped together to one single plug, for example all the directional control pins, the step pulse pins, and the input switches are each grouped to their own plug.

3.1.3. Power Supply

The controls use two different power supplies, one capable of producing up to 42 Volts - used for the movements - and high-energy tools the producing 3.3, 5 and 12 Volts - all being used by the controls and again voltage supplies for the tools.

Ignoring the power required by the tool, the maximum power consumption is calculated as following

$$P_M = U_S \sum_i I_{Si} + P_C$$

With P_M being the total power, U_S the stepper motor supply voltage, I_{Si} the individual supply current for each stepper motor (indexed with i) and P_C the power required by the controls. Summing up to about 300 Watts of power consumption.

3.2. Measurements

As the SyArm is a considerably basic prototype of a robotic arm, a simple method of measurement is used. When measuring, the control drives the motors into the negative direction until they close a switch connected to the Raspberry PI. The motors then immediately stop driving and set the minimum endpoint at the current direction.

As the robot only uses stepper motors for positioning, no additional measurements are required, though they would be useful in assuring the correct positioning of the robot. If any movement of the segments that is not caused by the controls occurs, then the robot must be remeasured again.

Each input is connected to the raspberry with a pull-down resistor and equipped with a status LED that lightens up once closed.

3.3. Tool supplies

As previously stated, the tools require different supply voltages for different actions. Also, input pins must be provided to correctly position or maintain the tools.

Therefore, a plug has been designed that can supply many different types of tools and applications in use. If more complex measurements or other sensor data is required, the connection must be made manually.

For example, take a pencil tool for drawing figures onto paper, the tool itself would not require any additional motors, but rather a kind of switch that can recognise how far the pencil is above the ground. The easiest approach would be to just mount a switch parallel onto the same plane as the pencil points to.
The sensor can now be connected to the input pin of the plug, enabling the input to be read by the controls.

Each tool must be put into the configuration file first and then linked to a type in the library. If a completely new type of tool is required, then the software must be adjusted.

When selecting the required tool via the index in the configuration list in the GCode-interpreter, all the pins and motors required are automatically setup.

```
"tools": [
  {
    "name": "AxisBearing_Mk1",
    "type_name": "stepper_lib::comp::tool::axial_joint::AxialJoint",
    "obj": {
      "servo": {
        "consts": "MG996R",
        "pwm": 23
      },
      "length": 50.0,
      "mass": 0.1
    }
  },
  {
    "name": "AxisTongs_Mk1",
    "type_name": "stepper_lib::comp::tool::axis_tongs::AxisTongs",
    "obj": {
      "axis": {
        "servo": {
          "consts": "MG996R",
          "pwm": 23
        },
        "length": 50.0,
        "mass": 0.1
      },
      "tongs": {
        "servo": {
          "consts": "MG996R",
          "pwm": 13
        },
        "perc_open": 0.7,
        "perc_close": 0.4,
        "length": 50.0,
        "mass": 0.1
      }
    }
  },
```

Figure 3.3.0.1

```
4    ; Setup
5    G28; Measure
6    G0 X0 Y330 Z400
7    T1; Select Axisbearing tool
8    M5; Make sure that the tongs are open
```

Figure 3.3.0.2

4. Software

4.1. Stepper library

4.1.1. Movements and actions

The movements of the robot can be divided into different layers and kinds:

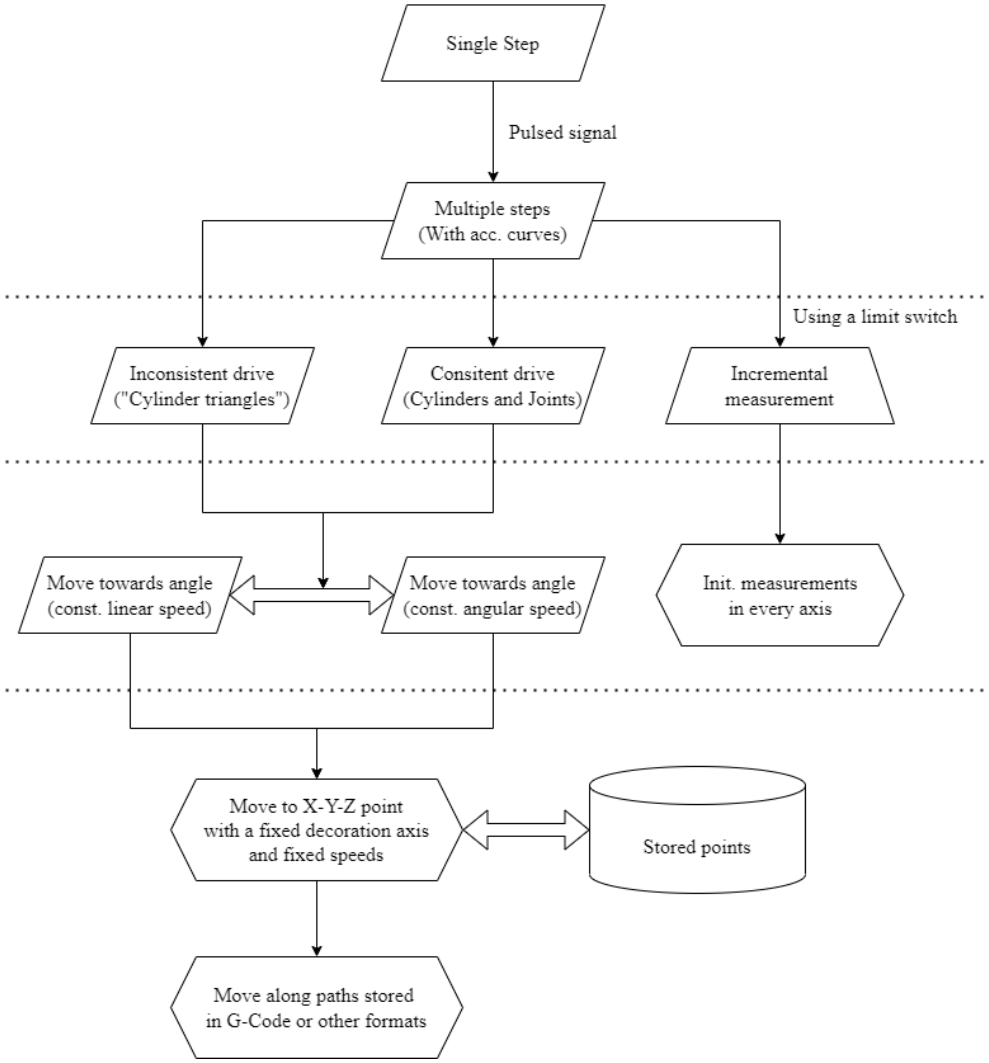

Figure 4.1.1.1

This flowchart depicts the general library structure, which features the shown actions and movements that get more and more complex the farther one goes to the bottom. Each action will now be described in detail.

Single step

A single step with a single motor, defined by the steptime T. The control keeps the signal up for half the desired steptime, therefore it is low the other half of said time.

The direction is changed by inverting the directional pin of the control, not by modifying the step pulse itself.

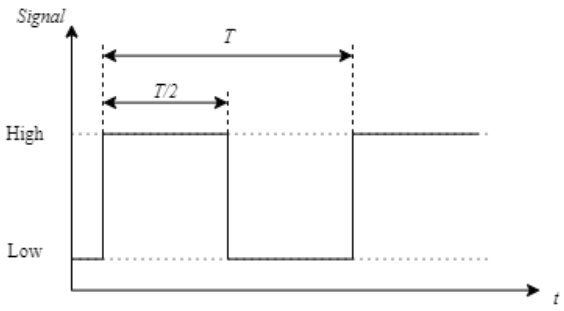

Figure 4.1.1.2

Multiple steps (with curves)

Moves a stepper motor multiple steps into the defined direction with proper acceleration and deceleration considering the inertia of the construction and motor torque. A full explanation of acceleration curves and different methods can be found in the appendix[1].

The two relevant functions are then defining a series, which look like the following:

$$f_0 = \sqrt{\frac{\alpha_{max} n_S}{4\pi}} \qquad f_n = f_0\sqrt{n}$$

The motors accelerate until the certain speed is reached or half the steps are travelled, after this distance, the control will mirror the curve at half the steps travelled.

Consistent drive (Used for cylinders or geared joints)

The consistent drive mode converts an angle into the right direction and steps that need to be travelled, using a predefined gear ratio or spindle pitch.

Inconsistent drive (Used for "cylinder triangles")

The inconsistent drive is used to keep a certain omega up when a cylinder is built in a triangular construction like the one displayed on the right. The law of cosines gives an angle or the cylinder length if either one of the variables is given.

$$c^2 = l_a^2 + l_b^2 - 2l_a l_b \cos(\gamma)$$

Considering the change in the cylinder extent is non-linear proportional to the angle γ and γ depending on the time t the following differential emerges:

$$c(\gamma), \gamma(t) \rightarrow v_c(t) = \frac{dc}{d\gamma}\big(\gamma(t)\big) \cdot \omega(t)$$

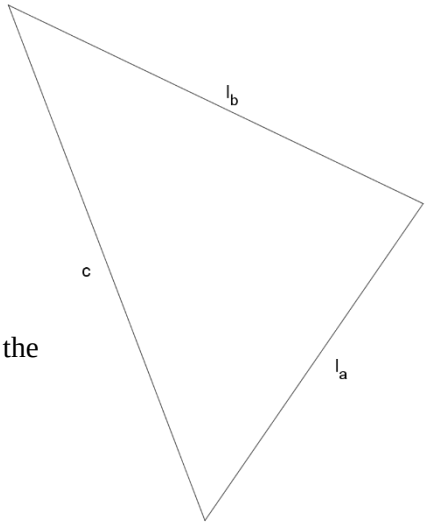

Figure 4.1.1.3

[1] Article "Stepper Motors"

When now a constant angular velocity is desired, the angular acceleration ends up at zero, which makes the second derivative simpler.

$$a_C(t) = \frac{d^2c}{dt^2} = \frac{d^2c}{d\gamma^2}\big(\gamma(t)\big) \cdot \omega^2(t) + \frac{dc}{d\gamma}\big(\gamma(t)\big) \cdot \alpha(t)$$

$$\alpha(t) = 0 \rightarrow a_C(t) = \frac{d^2c}{d\gamma^2}\big(\gamma(t)\big) \cdot \omega^2(t)$$

With the formular displayed above the velocity correction of the cylinder can be calculated, which leads to a consistent angular velocity.

Moving to a specific angle (with or without constant speed)

The certain movements are then made by taking the current position ϕ_C and calculating the difference $\Delta\phi$ to the desired angle ϕ.

$$\Delta\phi(\phi) = \phi - \phi_C$$

4.1.2. Incremental measurement

The robot should always know it's exact position, which is why a combination of stepper motors, and an initial incremental measurement system are used. When running this measurement operation, the motors move into the defined direction until the controls recognize a HIGH signal on the defined measurement input pin (one for each motor). The positional value is then set to another predefined value, the initial position value.

Additionally, the controls set a limit in the direction on the initial position value, as it would overextend and damage the construction.

4.1.3. Asynchronous movements

Moving only one motor at a time results in an unbearably low efficiency, which quickly leads to the concept of asynchronous movements.

To perform such movements, the control must be structured very differently than when normal synchronous ones are used. To understand the following descriptions in full detail, it is recommended to first read the part of the language documentations of rust dealing with their approach on sharing memory safely between threads.

The control creates a thread for each motor attached to the controls running in the background and equipped with a channel to transfer data with the main thread. If the thread receives a message with the data required for the movement, it will wake up and perform this movement.

If the thread receives a message without data in it, a Rust “None”-value of an option, it will break the loop and kill itself without sending a message back to the main thread.

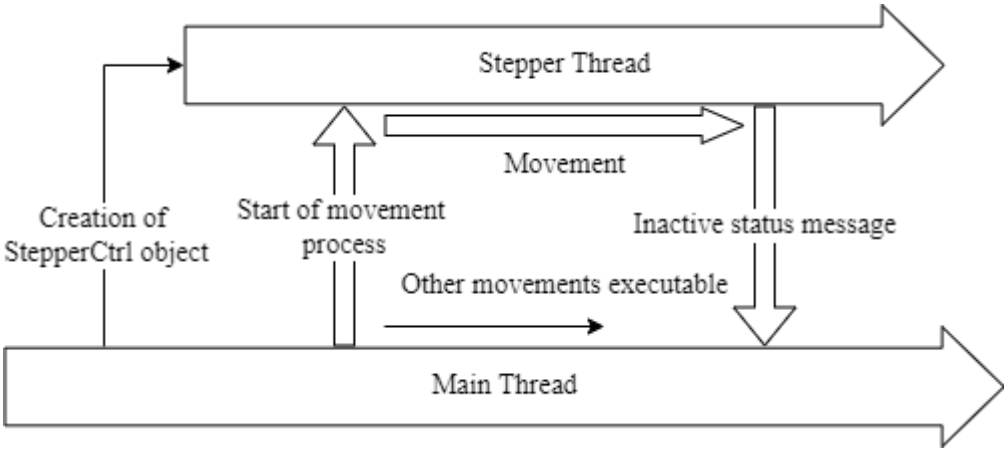

Figure 4.1.3.1

The main driver class is stored in a so called mutex, a structure that prevents multiple processes from writing to the same block of data at the same, which could cause memory safety issues. The mutex forces each process to first put a lock on the structure, preventing all other instances to write to the locked data until said process is not in need of the mutable access anymore, dropping the data and the lock afterwards.

4.1.4. Limits and failsafe

The library provides a protection against force overloads and movements out of the allowed range of a component, which can be set at any time. Before the robot executes any movements, it will check for said cases and inform the user that an invalid move would be executed.

4.1.5. Component System

Every motor, every gear and every cylinder are categorized as a *Component* in the stepper library. Components can consist of a motor and a kind of mechanical structure that transforms the movement. The only condition is that only one motor is in use, if multiple motors are required, as it is the case in most robots, a *Component-Group* has to be used. It makes coordinating and storing data of multiple motors easier. The stepper library includes some of the basic components like a geared bearing, a cylinder and so on. When requiring more complex components, simply create a new structure and implement the *SyncComp* provided by the crate.

4.2. SyBot Library

The SyBot Library is the main part of the controls. It manages all the motors using a vector model of the robot. A clear overview of the mathematical procedures will be provided and explained in the further sections.

The library is, just like the stepper library, written in Rust using cargo as package manager. The libraries, which have been used, can be seen either in the GitHub repository or at the sources section.

4.2.1. Modelling

The main purpose of modelling is calculating loads and inertias to help the motor calculate the right acceleration curves. If the model results in giving incorrect values to the controls this could lead to motors moving too slow, not in the right direction or not at all. On the other hand, it could lead to a low efficiency due to the motors not using their full potential.

Inertia

To keep things simple, every part of the robot can be seen as a rod, represented by a certain mass m, a positional vector $\overrightarrow{a_P}$ and a representing vector $\overrightarrow{a_l}$. The inertia of a single rod can now be calculated using the standard inertia of a rod and considering center distance of the rod and rotation centre.

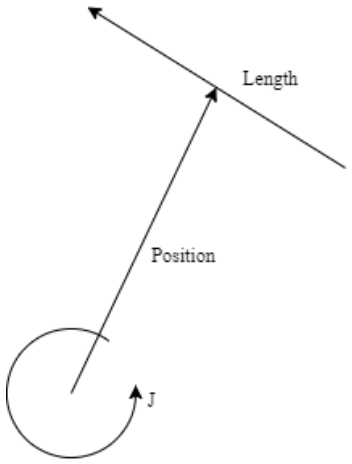

Figure 4.2.1.1

$$J = m\left(\frac{|\overrightarrow{a_l}|^2}{12} + |\overrightarrow{a_p}|^2\right)$$

$$|\overrightarrow{a_t}|^2 = \left(\sqrt{x_t^2 + y_t^2}\right)^2 = x_t^2 + y_t^2$$

Assuming the robot is described as a sequence of rod segments, the position vector can be calculated as following

$$\overrightarrow{a_{p\imath}} = \frac{\overrightarrow{a_{l\imath}}}{2} + \sum_{j=1}^{i} \overrightarrow{a_{lJ}}$$

Which results in the inertia of that sequence for a coordinate a, later being x, y, or z, as

$$J_a = \sum_{i=1}^{n} m_i \left(\frac{a_i^2}{12} + \left(\sum_{j=1}^{i-1} a_j + \frac{a_i}{2}\right)^2\right) \qquad \boldsymbol{J} = \begin{bmatrix} J_y + J_z & 0 & 0 \\ 0 & J_x + J_z & 0 \\ 0 & 0 & J_x + J_y \end{bmatrix}$$

The inertia matrix $\boldsymbol{J}$ can be used for all calculations requiring an inertia, for example

$$\overrightarrow{\mathrm{M}} = \boldsymbol{J} \cdot \vec{\alpha}$$

Reduced mass

As the construction includes cylinders with a linear motion, the concept of a reduced mass is used to calculate inertia for those linear motions.

$$m_r = \left|\boldsymbol{J} \cdot \frac{\vec{r} \times \widehat{a_0}}{|\vec{r} \times \widehat{a_0}|^3}\right|$$

Where $\boldsymbol{J}$ represents the inertia tensor, $\vec{r}$ the distance to the centre point of the inertia and $\overrightarrow{a_0}$ the direction of the velocity or acceleration input acting upon m_r.

Load forces

As the robot should be able to carry around weights later, the mathematical model should include forces and how they impact the robot, including its own weight. Every segment is loaded with a certain force $\vec{F}$ and its own weight $\overrightarrow{F_G}$. Additionally, every segment causes a force in the joint F_J it's connected to and in the cylinder or gears stabilizing the segment $\overrightarrow{F_C}$.

Each vector has got a position vector attached to it called $\vec{a}$ and whatever prefix the force it expresses has. Also, the unknown forces are expressed with a directional vector called $\hat{a}$ and again whatever prefix the force they express has. So, using the basic mechanical principles of statics

$$\sum \vec{M} = 0 \qquad\qquad \sum \vec{F} = 0$$

the two reaction forces can be calculated.

$$\overrightarrow{F_C} = \widehat{a_C}\left(\vec{M} \cdot \vec{\eta}\right) \qquad \vec{M} = \sum_i \overrightarrow{a_i} \times \vec{F_i} \qquad \vec{\eta} = \frac{\overrightarrow{a_C} \times \widehat{a_C}}{|\overrightarrow{a_C} \times \widehat{a_C}|^2}$$

As now only the joint force is left unknown, the second principal can be applied to get the following equation.

$$\overrightarrow{F_J} = \sum \overrightarrow{F_A} - \sum \overrightarrow{F_R}$$

Where the first sum stands for all action forces onto the segment and the second sum for all the reaction ones that are already known.

When now a constant angular velocity is desired, the angular acceleration ends up at being zero, which makes the second derivative simpler.

$$a_C(t) = \frac{d^2c}{dt^2} = \frac{d^2c}{d\gamma^2}\big(\gamma(t)\big) \cdot \omega^2(t) + \frac{dc}{d\gamma}\big(\gamma(t)\big) \cdot \alpha(t)$$

$$\alpha(t) = 0 \rightarrow a_C(t) = \frac{d^2c}{d\gamma^2}\big(\gamma(t)\big) \cdot \omega^2(t)$$

With the formular displayed above the velocity correction of the cylinder can be calculated, which leads to a consistent angular velocity.

4.2.2. Collective movements

Moving to a specific X, Y and Z coordinate point

To move to a specific position in space expressed by a X, Y and Z coordinate, the control must convert such a point into angles for the motors to move to.

$$C(X, Y, Z) \rightarrow \phi_B, \phi_1, \phi_2, \phi_3$$

The base angle ϕ_B can be determined with ease as it is the only angle that performs a rotation around the Z-Axis. Looking at the robot from top down can be used to calculate the base angle as following:

$$\phi_B = \tan\left(\frac{Y}{X}\right) - \frac{\pi}{2}$$

The base angle is shifted by ninety degrees as the base arm is orientated along the Y-axis to perform correct rotation operations around the X-axis. This base angle can now be used to rotate the point onto the Y-Z plane.

$$P_R = R_Z(\phi_B) \cdot P; \quad P = (X, Y, Z)$$

Now the decoration vector can be subtracted without applying the rotation matrix to it. To finally simplify the point into a triangle that is fully defined, the base vector must be subtracted from the rotated point P_R as well.

$$P_T = P_R - \overrightarrow{a_B} - R_B(\phi_D) \cdot \overrightarrow{a_D}$$

This point now stands for the distance the first two arm segments must reach. Their exact angles are calculated as following:

$$\phi_{h1} = arccos\left(\frac{|P_T|^2 + l_{A1}^2 - l_{A2}^2}{2|P_T|l_{A2}}\right)$$

$$\gamma_2' = \arccos\left(\frac{-|\mathrm{P_T}|^2 + \mathrm{l_{A1}^2} + \mathrm{l_{A2}^2}}{2\mathrm{l_{A1}l_{A2}}}\right)$$

$$\phi_H = arcsin\left(\frac{P_T \cdot \hat{j}}{|P_T||\hat{j}|}\right)$$

$$\phi_1 = \phi_{h1} + \phi_H$$

$$\phi_2 = \pi - \gamma_2'$$

$$\phi_3 = \phi_D - (\phi_1 - \phi_2)$$

As displayed in the formula, ϕ_3 can be expressed as the difference between the decoration angle and the angle left by the first two segments when moving to the required point P_T.

4.2.3. Data and constants

To retrieve all the data required for the controls like component mass, allowed angle ranges and speeds. All this information is packed into a single JSON-file, the *JsonConf* (".conf.json" file extension). The file is structured so that every robot built into the framework can be defined and parsed by this file.

The file is parsed into a *Machine Configuration* during runtime, if any of the syntax or values are incorrect, the program will inform the user.

4.2.4. G-Code interpreter

The robot uses the common GCode standard to dynamically describe its movements. For this robot, a GCode-interpreter binary is provided in the sybot library repository. This interpreter can also handle files, examples are also provided in the repository.

For each command the interpreter returns a JSON-Value informing about the result or success of the given instruction.

The library also contains a HTTP Webserver equipped with multiple ways to gather information or control the robot. The webserver is hosted as a public network interface and port configured via an environment variable.

The easiest way to control the robot is via the websocket interpreter interface, where GCode can be streamed directly to the robot. Of course, there are other options then the webserver for executing operations. The application has more executables and commands that can help handle the different application fields required in automation.

5. Conclusion

This robot gives a lot of insight into basic robotics and showed me what concepts to keep and which ones to improve upon. The versioning with “Mk 1”, has a purpose, as I am planning to build improved versions with more stability and one or even two more axes.

It is a great experiment for testing and improving the control libraries in rust, a programming language that I was not able to work in at the start of this project. The language showed me its qualifications for serving high level communication as hardware programming all in one. Unfortunately, its accuracy ended up being not good enough for tasks like drawing.

Many of the work done for this project, especially mentioned libraries, will be reused for my next project that I am working on right now as I am writing this (diploma project DrAI).

6. Appendix

The whole project can be found on a public GitHub repository, either search the name (“SamuelNoesslboeck/SyArm_Mk1”) or the Link:

https://github.com/SamuelNoesslboeck/SyArm_Mk1

6.1. Technical drawings

- “A_SyArm_Mk1”, Main draft
- “A1_SimpleFrame”, Frame draft
- “A2_Cylinder_Mk2”, Cylinder draft
- “A4_S1_S2_FirstSegment_SecondSegment”, Draft about arm segments

6.2. Articles and other self-made references

- “Stepper Motor”, Article about stepper motors, acceleration curves etc.
- “SyArm Positioning”, Article about positioning, conversions, and transformations
- “SyArm Mechanics”, Article about forces and loads acting upon the robot

6.3. Sketches and drawings

- See “Sketches” subfolder

6.4. GitHub-Repositories

- “SyArm_Mk1” – Main Repo
- “Stepper Library” – Rust Library for controlling stepper motors and components
- “SyBot Library” – Rust Library for controlling complex component groups and robots, includes many different ways for communication

7. Sources, references, and figures

7.1. Figures