Design of a Portable Control Platform for Rod-Driven Continuum Parallel Robots

Josué Licona Mármol

A graduation project submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Mechanical Engineering

Supervisor
Jonathan Camargo Leyva, M.Sc.

Universidad de los Andes
Facultad de Ingeniería

Department of Mechanical Engineering
Universidad de los Andes
Bogotá D.C., Colombia
June XX, 2024

To my mother, Gildes Ester.

Success is the sum of small efforts, repeated day in, and day out.

— Robert Collier

Abstract

This project focuses on designing and implementing a portable, modular, and scalable platform alongside a control system tailored for rod-driven continuum parallel robots. The main goal was to develop a versatile mounting solution facilitating efficient robot movement across diverse applications. The project involved the design and construction of the mounting platform, as well as the development of the control interface. A rod-driven continuum parallel robot was designed and implemented, with a focus on miniaturization and optimization of the structure. Additionally, the project aimed to implement a customized language specification for programming the linear actuators, enabling tailored control and operational flexibility. Furthermore, an application was developed to interface with the control system, utilizing the created protocol to ensure seamless communication and precise control.

Acknowledgements

I want to thanks to God...

Contents

List of Figures

List of Tables

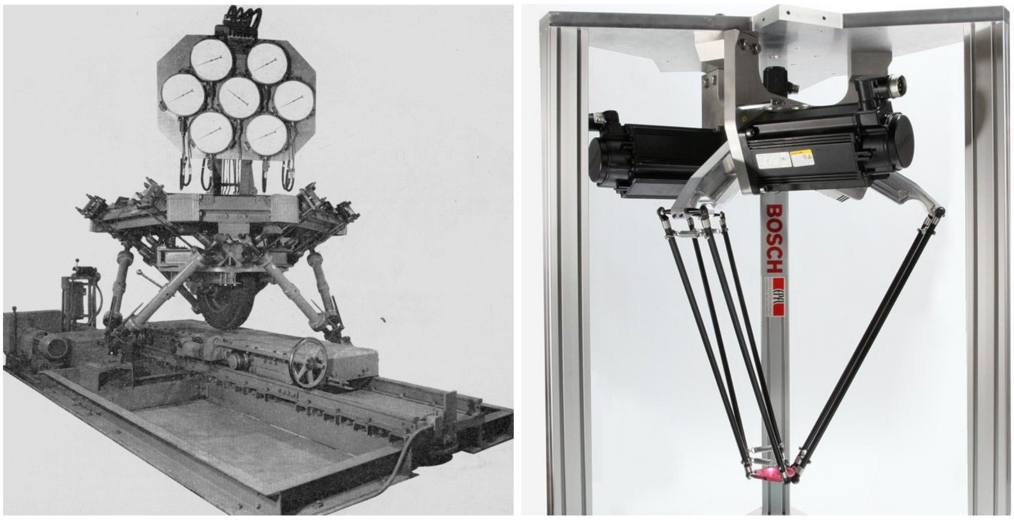
Chapter 1

Introduction

Robots in general can be categorized into serial and parallel types. Serial robots are characterized by a series of linked joints while parallel robots have multiple axes that move in parallel, usually working together to support a single platform.

Parallel robots represent a significant branch in the field of robotics due to their precision and versatility in various applications such as pick-and-place manipulation, simulators, manufacturing, and tooling, among others. As described in [1], these applications span both industry and medicine, leveraging the high rigidity, precision, and speed of these robots to compensate for the performance limitations of serial robots.

The origin of parallel robots dates back to the 1960s with the development of the Gough-Stewart [2] parallel mechanism (Figure 1.1a), which has become one of the most iconic in the field of parallel robotics. Later, in the 1980s, Reymond Clavel designed a robust parallel structure with three translational degrees of freedom and one rotational degree of freedom, as illustrated in Figure 1.1b. This robot, known as the Delta Robot, has become one of the most significant examples in the field of parallel robotics.



(a). Gough-Stewart Platform
Source: Adapted from [2]

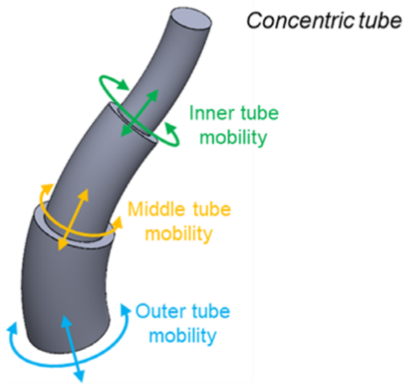
(b). Clavel's Delta Robot
Source: Adapted from [3]

Figure 1.1: Traditional parallel robots

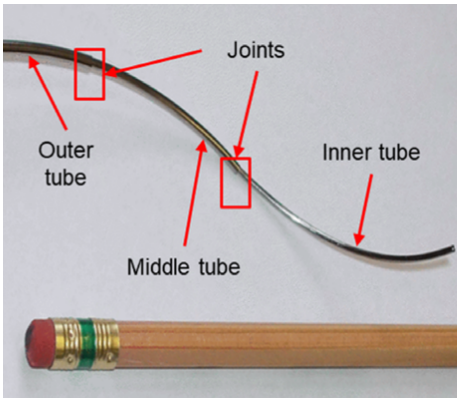
Another distinguishing feature of these robots is the presence of multiple closed kinematic chains that connect and move the mobile platforms, in contrast to serial robots, which operate with an open kinematic chain and move in a linear sequence. However, according to Briot and Kahrs [4] the fundamental questions about this class of robots have been addressed. As a result, in recent years, interest in parallel robots has shifted towards exploring new alternatives.

Current research in robotics transcends conventional definitions, pushing the boundaries of this field and fostering new possibilities and innovations. Briot and Kahrs [4] illustrate how parallel manipulators have been integrated into various novel robot types, such as continuum robots, flying robots, cable-driven robots, underactuated robots, multi-fingered hands, and microscale parallel robots. However, this emerging class of robots presents significant scientific challenges in design, modeling, and control. Among these categories, aerial robots, parallel robots, and continuum or soft robots are particularly noteworthy for their ability to venture into new areas due to their lightness and flexibility.

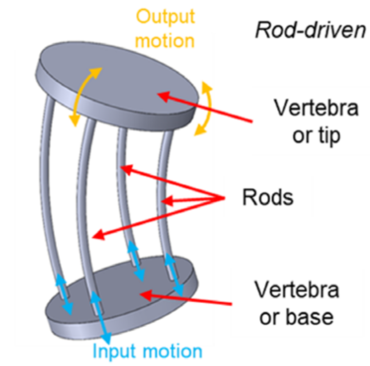
Expanding on these insights, Russo et al. [5] offer a comprehensive review



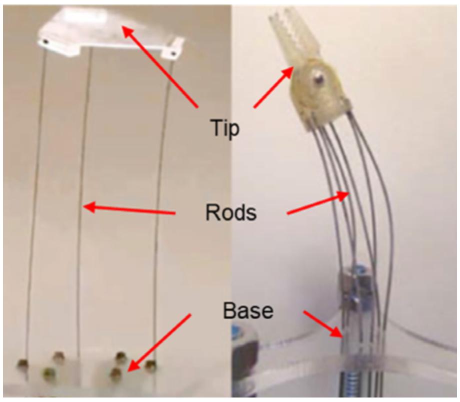
(a). Concentric tube robot mobility



(b). Concentric tube prototype



(c). Rod-driven continuum robot conceptual scheme



(d). Parallel continuum rod-driven robot prototype

Figure 1.2: Continuum robots with extrinsic actuation

Source: Adapted from [5]

focusing on recent advancements, current limitations, and ongoing challenges in the design, modeling, and control of continuum robots. They classify continuum robots based on their design, distinguishing them by their extrinsic or intrinsic actuation methods. Extrinsic actuation (Figure 1.2) involves transmitting motion from the robot's base along its structure, categorized into three main families depending on the transmission elements used: tendon-driven, concentric tube (Figures 1.2a, 1.2b), and rod-driven robots (Figures 1.2c, 1.2d). Furthermore, robots with intrinsic actuation employ actuators integrated into their structure to generate movement, meaning the actuation occurs within the robot's body.

Current challenges in designing and modeling parallel continuum robots include improving performance through miniaturization of actuators, integrating with rigid robots, exploring smart materials, precise environmental modeling, and implementing proprioception with new sensors [5]. Modeling efforts also focus on representing interaction environments and improving real-time implementations, alongside standardizing simulation environments. Control challenges involve ensuring precise manipulation and adapting to dynamic environments using advanced sensor technology and adaptive strategies

This project aims to address the efficiency challenges of continuum robots through the miniaturization of their parallel linear actuators. Specifically, it focuses on implementing adaptive control strategies for diverse applications using an extrinsic actuation platform for rod-driven continuum robots. This research endeavors to enhance affordability and user-friendliness, aiming to broaden adoption and usability in practical contexts.

1.1 Objectives

1.1.1 General Objective

To design and develop a portable, modular, and scalable platform with a control system for the actuation of rod-driven continuum parallel robots.

1.1.2 Specific Objectives

- Adapt an existing design of a rod-driven continuum parallel robot, optimizing it for reduced size while maintaining functionality and performance.
- Design linear actuators that provide precise control and enable dexterous movements, ensuring high accuracy and reliability.
- Fabricate all necessary components and assemble a fully functional physical

prototype of the platform, adhering to the design specifications.

- Implement a customized language specification for programming the linear actuators, allowing for tailored control and flexibility in operations.
- Develop an application to interface with the control system, utilizing the created protocol to facilitate seamless communication and control.
- Conduct comprehensive experimental testing and validation of the system, ensuring it meets all performance criteria and operational standards.

Chapter 2

Target Robot Structure

2.1 Reference Model

2.2 Rod Material

$L \gg r$

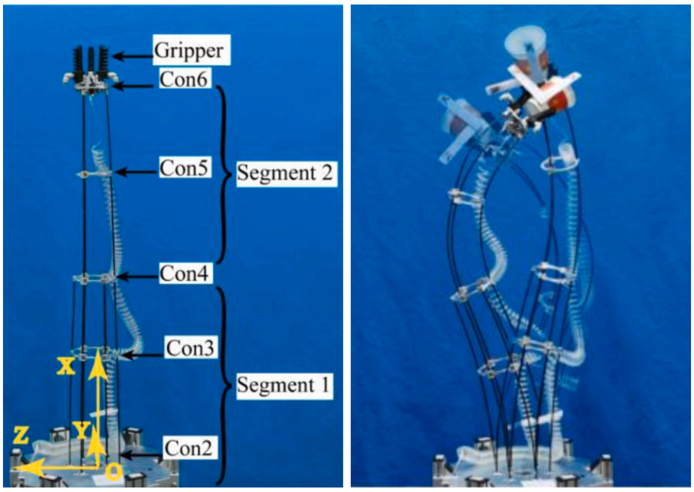
Maximum deflection Euler-Bernoulli beam

$$\delta_{max} = \frac{PL^3}{3EI} \tag{2.1}$$

$$I = \frac{\pi D^4}{64} \tag{2.2}$$

Table 2.1: Fiberglass and AISI 302 steel properties

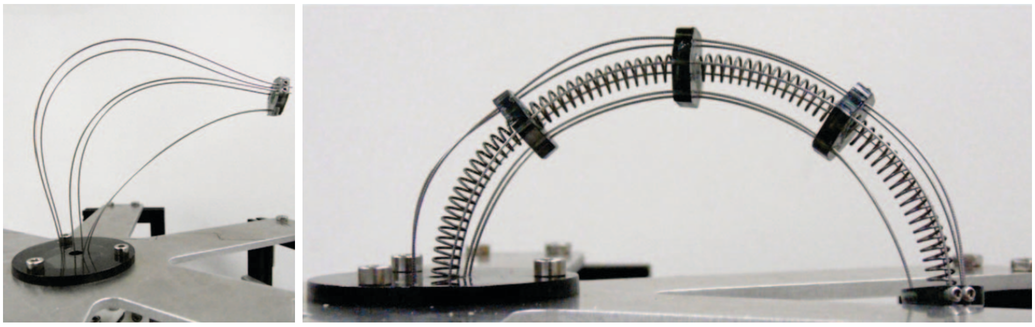
Symbol	Property	Fiberglass	AISI 302
ρ	Density [g/cm^3]	2.6	8.0
E	Young's Modulus [GPa]	85	187.5
G	Shearing Modulus [GPa]	36	70.3



(a). Sections and structure **(b).** Some end poses

Figure 2.1: Reference robot structure

Source: Adapted from [6]



(a). No constraints **(b).** Extended range with intermediate constraints

Figure 2.2: Constraints and range of motion

Source: Adapted from [7]

Table 2.2: Comparison between Wu & Shi (2022) model and custom test model

Parameter	Wu & Shi model [6]	Custom test model
Maximum length	~ 860 mm	~ 450 mm
Minimum length	~ 400 mm	~ 250 mm
Diameters of constraints	[90, 80, 70, 60] mm	[55, 50, 45] mm
Base constraint diameter	100 mm	70 mm
Number of sections	2	2
Number of rods	6	6
Rod cross-section diameter	3 mm	0.8 mm
Rod material	Fiberglass	AISI 302

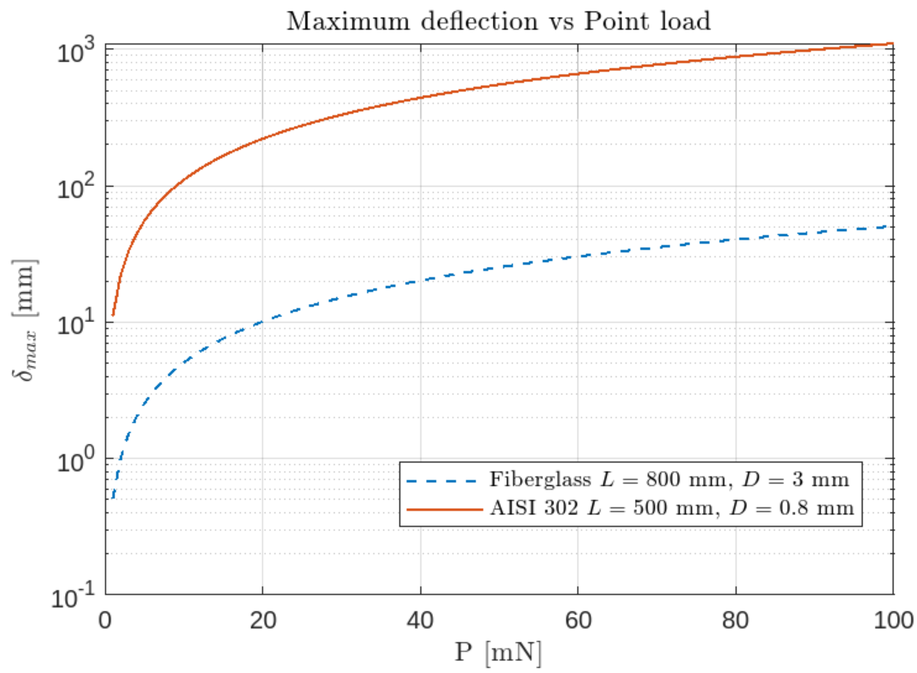


Figure 2.3: Maximum deflection and point load

2.3 Design and Dimensions

2.4 Ring Constraints

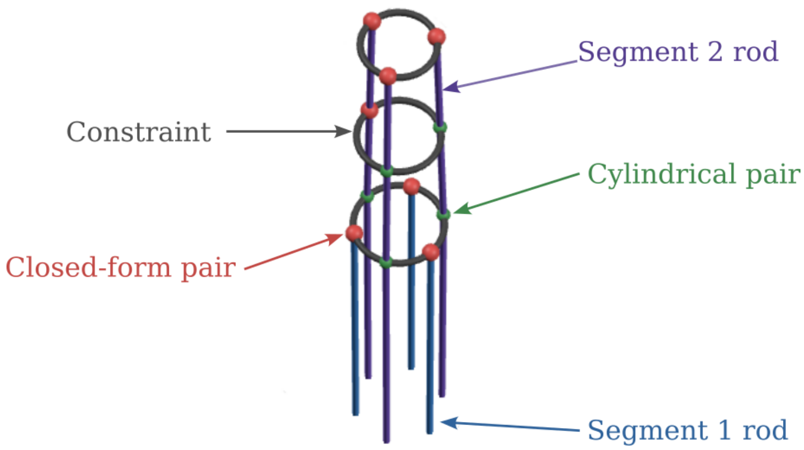
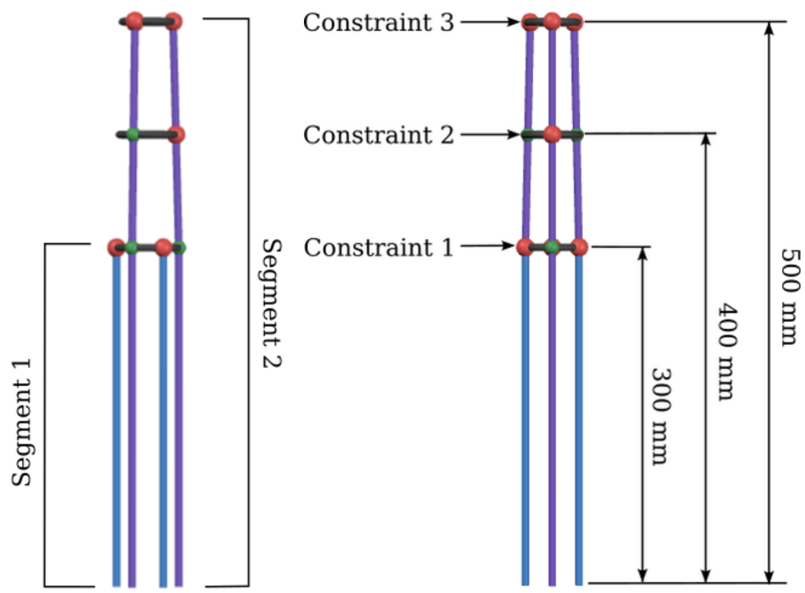
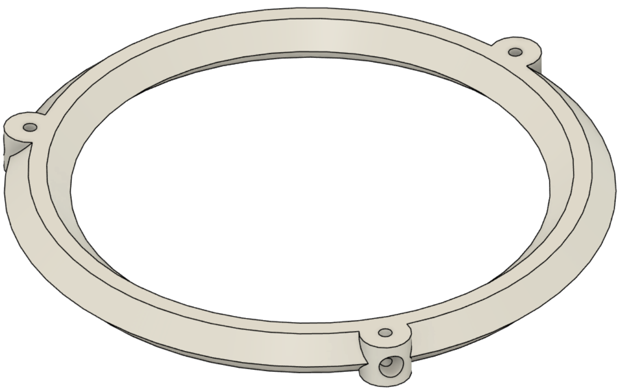


Figure 2.4: Robot body constraints and parts

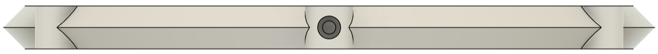


(a). Left view, and robot body segments **(b).** Frontal view, and constraints heights

Figure 2.5: Robot body lateral views



(a). General view



(b). Lateral view

Figure 2.6: Ring constraint

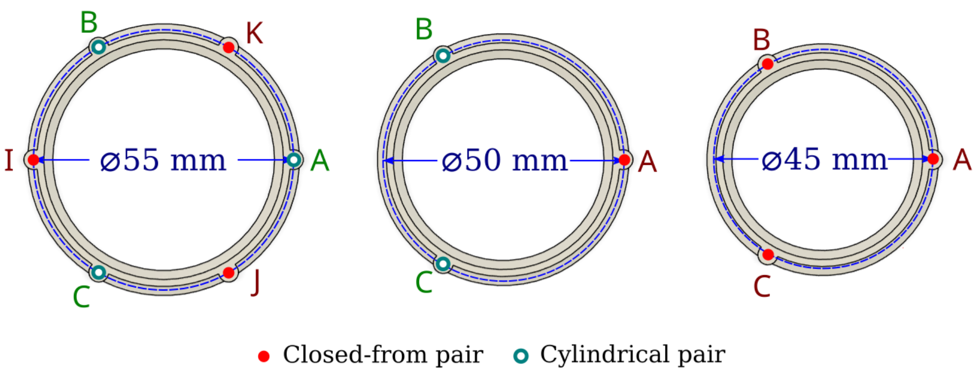
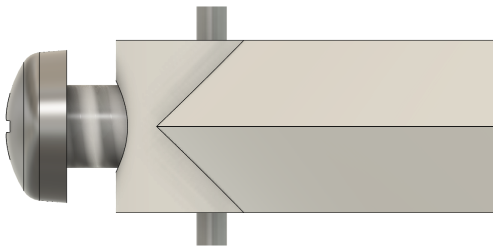
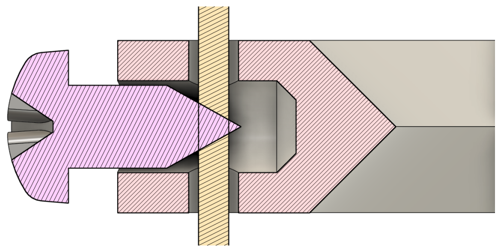


Figure 2.7: Ring constraint sizes

In *blue*, the diameters of the circumferences enclosing the rods are shown. The rods are labeled *A*, *B*, *C*, *I*, *J*, *K*. The *green* open circle indicates a cylindrical pair constraint between the ring and the rod, while the *red* indicates a closed-form pair constraint involving both.



(a). Physical closed-form pair constraint



(b). Section view of closed-form pair constraint

Figure 2.8: Closed-form pair constraint

Chapter 3

Linear Actuator

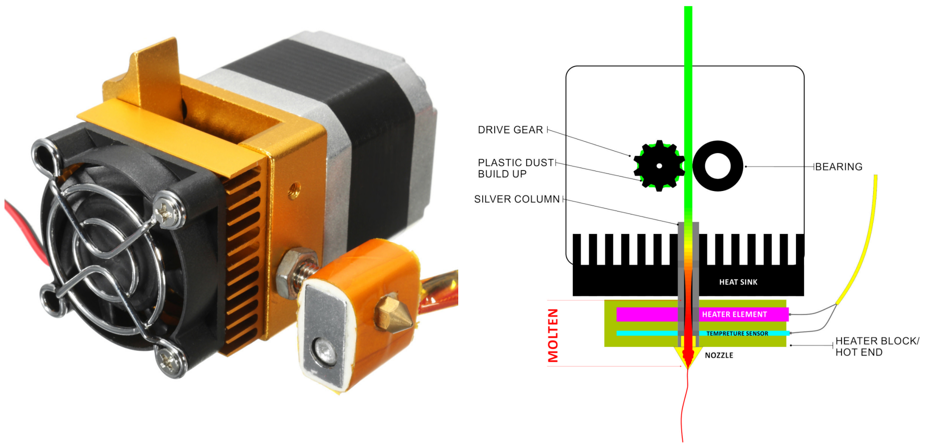
$R_{min} = 3.5$ cm, $c = 0.5$ cm, $W \approx 3.5$ cm

Original extruder $W = 5.2$ cm, $c = 0.7$ cm, $R_{min} = 5.2$ cm

$$r = R_{min} - c \tag{3.1}$$

$$W = \frac{2}{\sqrt{3}} r \tag{3.2}$$

Figure 3.1: Linear motion systems



(a). 3D printer extruder **(b).** Functionality of extruder

Figure 3.2: 3D printer extruder

Figure 3.3: Extruder without heater

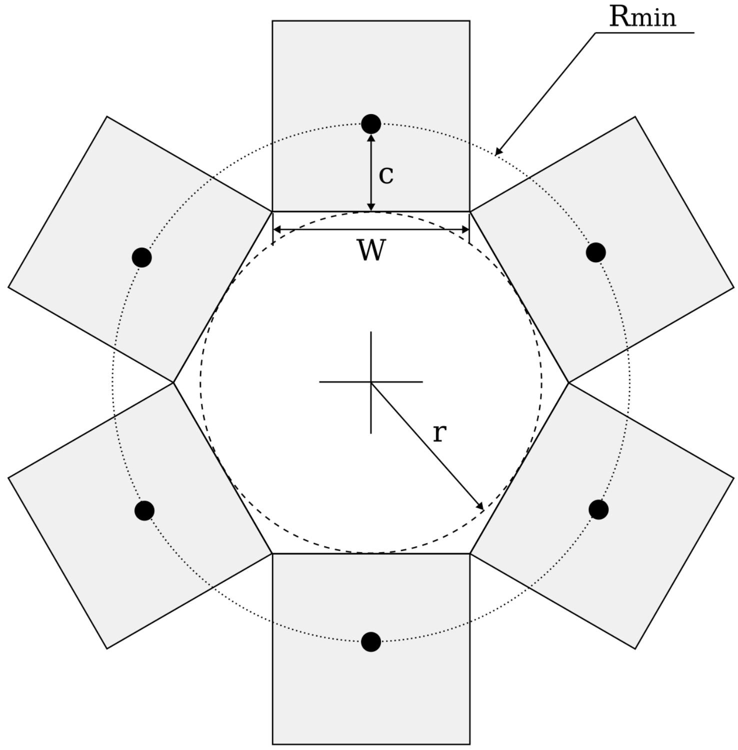


Figure 3.4: Actuators arrange

$$W = \frac{2}{\sqrt{3}}(R_{min} - c) \tag{3.3}$$

$$R_{min} = \frac{\sqrt{3}}{2}W + c \tag{3.4}$$

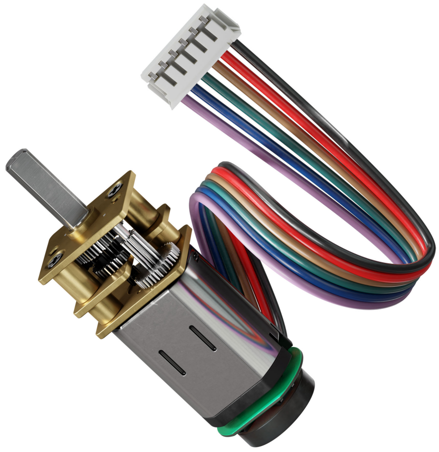


Figure 3.5: Gear motor N20 with encoder

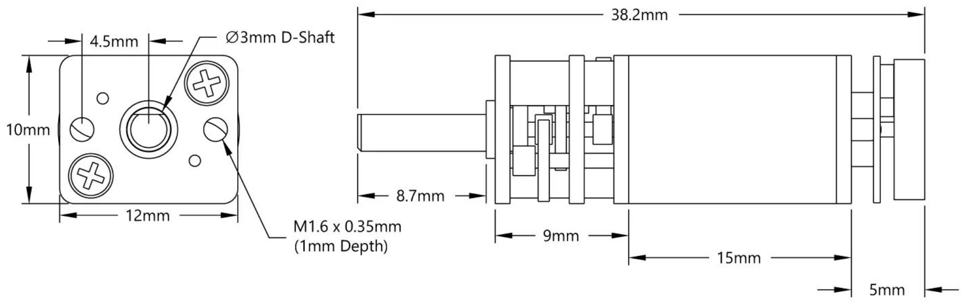
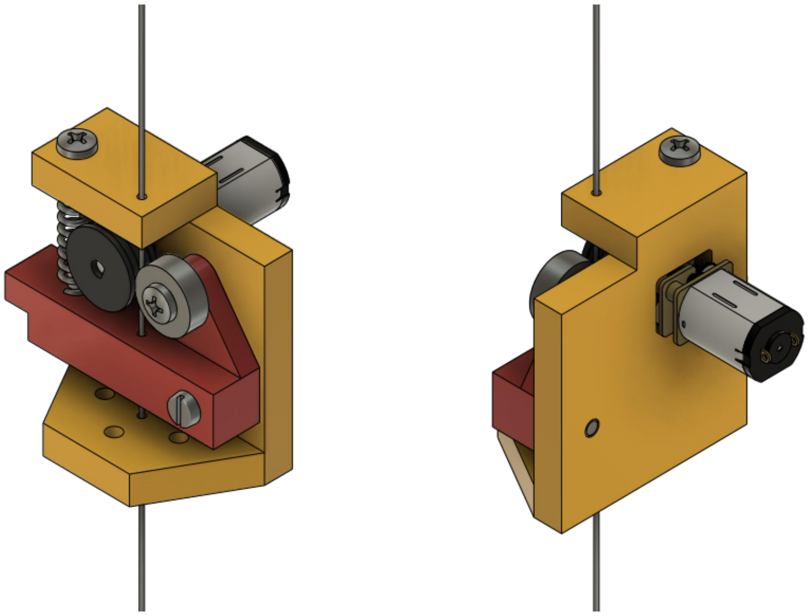


Figure 3.6: Dimensions of gear motor N20 with encoder

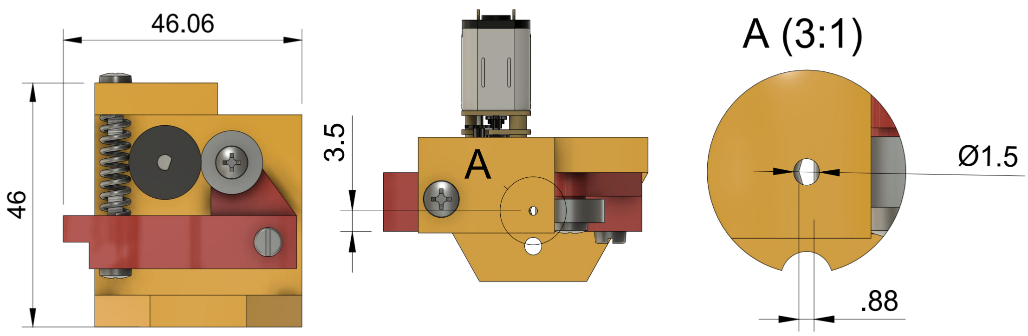
Table 3.1: Gear motor specifications

Property	Value
Output shaft style	D-Shaft
Voltage range	6 – 12V
Speed (no load @ 6VDC)	70 rpm
Rated torque	0.65 kg·cm
Stall torque	4 kg·cm
Gear ratio	210:1
Weight	15g
Encoder: cycles per revolution (motor shaft)	3
Encoder sensor type	Magnetic (Hall Effect)
Hall response frequency	100 kHz



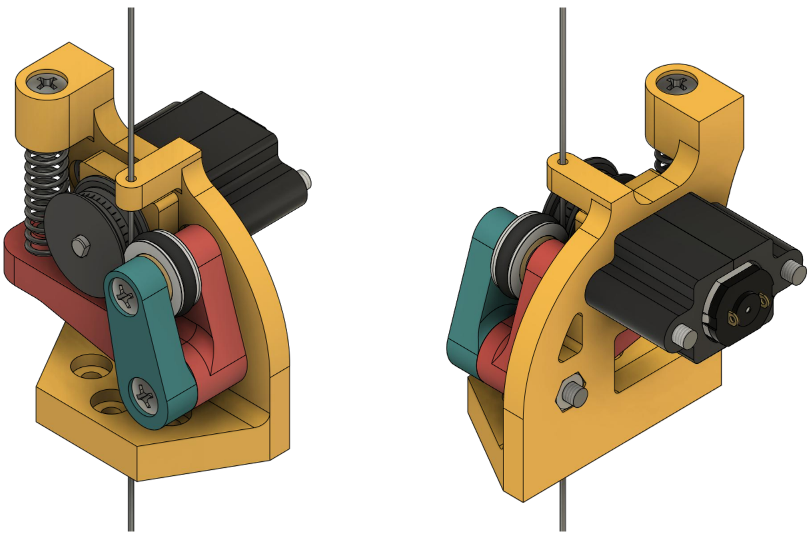
(a). Front isometric view **(b).** Back isometric view

Figure 3.7: First model isometric view



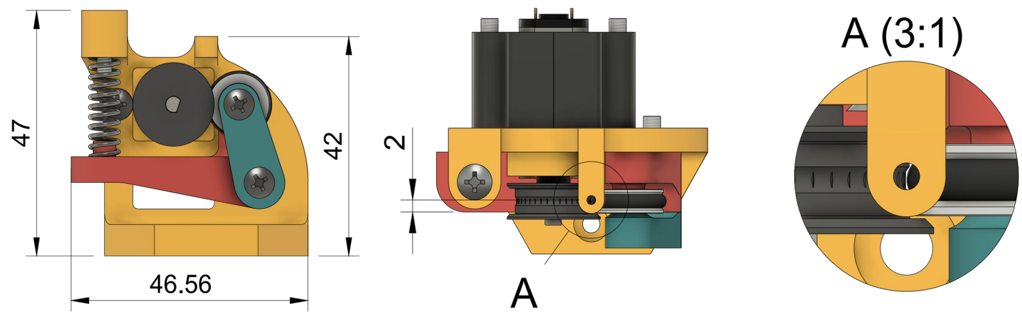
(a). Front view **(b).** Top view

Figure 3.8: First model dimensions



(a). Front isometric view **(b).** Back isometric view

Figure 3.9: Final model isometric view



(a). Front view **(b).** Top view

Figure 3.10: Final model dimensions

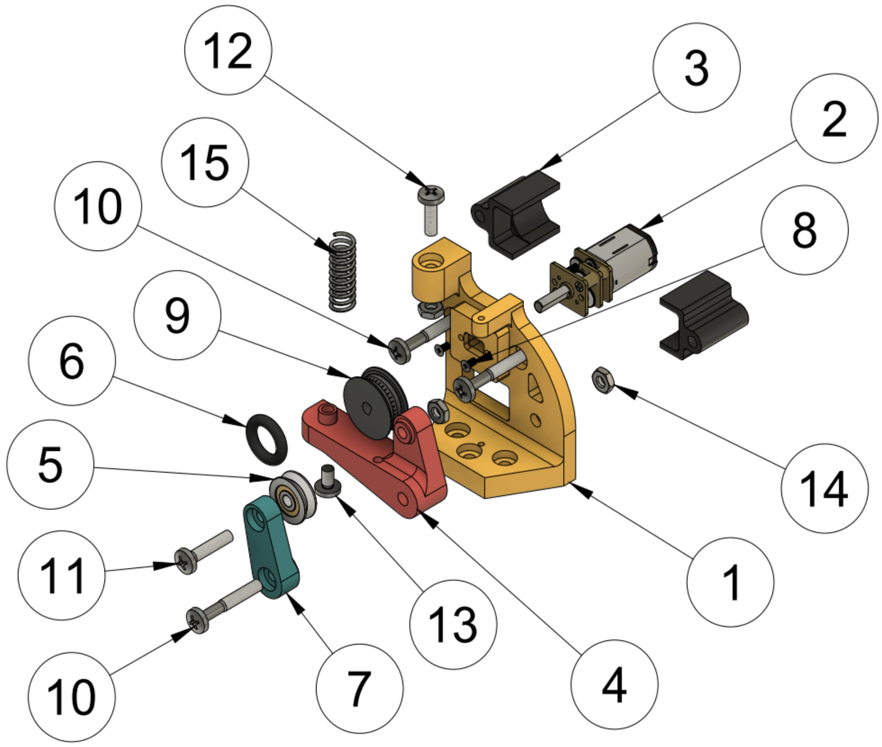


Figure 3.11: Actuator assembly exploded-view

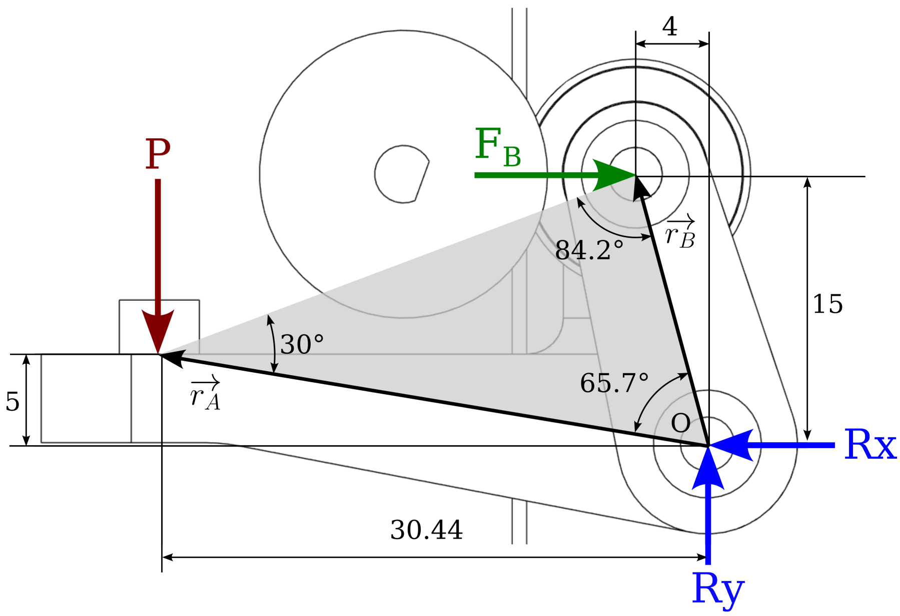


Figure 3.12: Forces in actuator arm

Table 3.2: Actuator assembly parts list

Item	Qty	Part Name / Description	Source
1	1	Base	3D Printed
2	1	Motor GA12 N20 (must be with encoder)	Commercial
3	2	Motor Support	3D Printed
4	1	Arm	3D printed
5	1	V623ZZ Groove Pulley	Commercial
6	1	O-Ring - W2.62 x DI7.59 x DE2.83	Commercial
7	1	Arm Support	3D Printed
8	2	Phillips Countersunk Screw - DIN 965H M1.6x3	Commercial
9	1	Pulley	3D Printed
10	3	Binding Head Screw JIS B 1111 - M3x20	Commercial
11	1	Binding Head Screw JIS B 1111 - M3x14	Commercial
12	1	Binding Head Screw JIS B 1111 - M3x10	Commercial
13	1	Binding Head Screw JIS B 1111 - M3x5	Commercial
14	3	Hexagon Thin Nut DIN 439-2 - M3x0.5	Commercial
15	1	Spring - D5.5x20mm	Manufactured

3.1 Motor

3.2 First model

3.3 Final model

3.4 Specifications

$$\sum \overrightarrow{F} = 0 \quad \therefore \quad \overrightarrow{P} + \overrightarrow{R_y} + \overrightarrow{F_B} + \overrightarrow{R_x} = 0 \tag{3.5}$$

$$\sum \overrightarrow{M_O} = 0 \quad \therefore \quad \overrightarrow{r_A} \times \overrightarrow{P} + \overrightarrow{r_B} \times \overrightarrow{F_B} = 0 \tag{3.6}$$

$$P = -k\Delta y \tag{3.7}$$

$$F_B = \frac{r_{A,x}}{r_{B,y}} P = \frac{30.44}{15} P = -2.0293k\Delta y \tag{3.8}$$

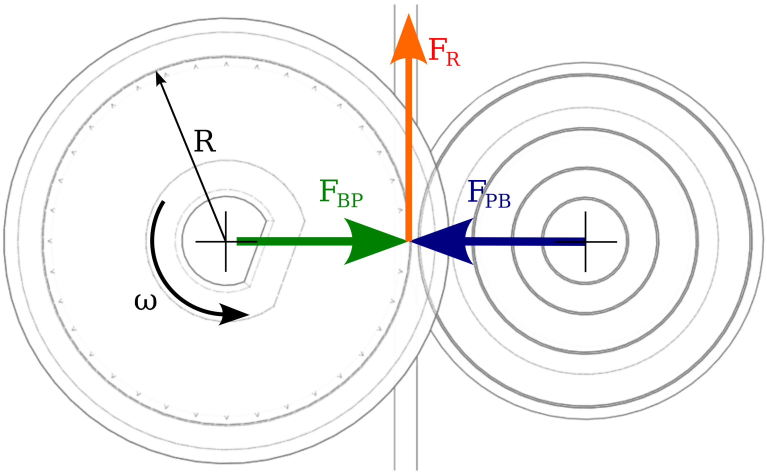


Figure 3.13: Friction force in rod

$$F_{BP} = F_B \tag{3.9}$$

$$F_R = \mu_s F_{BP} = -2.0293 \mu_s k \Delta y \tag{3.10}$$

Table 3.3: Actuator specification

Property	Value
Pulley ratio (R)	5.3 mm
Maximum angular velocity (ω_{max})	7.3304 rad/s
Maximum linear velocity (v_{max})	38.85 mm/s

Chapter 4

Platform Assembly

4.1 Electronics Case

4.1.1 Circuit Schematic

Table 4.1: Platform assembly parts list

Item	Qty	Part Name / Description	Source
1	1	Actuators Base	3D Printed
2	2	Support Base	Acrylic Laser Cut
3	6	Support Union	Acrylic Laser Cut
4	1	Support Front	Acrylic Laser Cut
5	2	Support Lateral	Acrylic Laser Cut
6	1	Electronics Assembly	
7	4	Binding Head Screw JIS B 1111 - M3x40	Commercial
8	24	Binding Head Screw JIS B 1111 - M3x10	Commercial
9	6	Actuator Assembly	

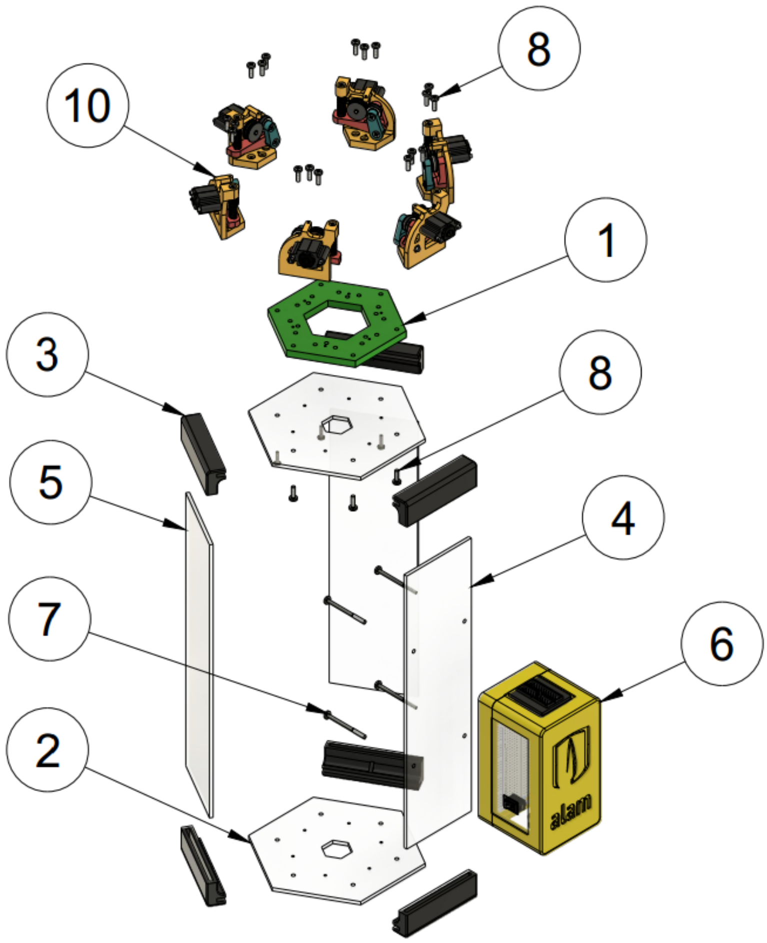


Figure 4.1: Platform assembly exploded-view

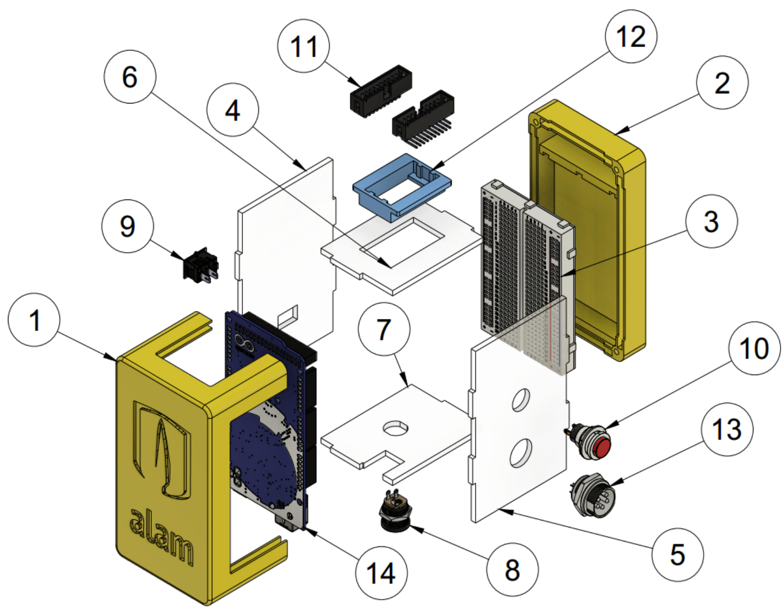


Figure 4.2: Electronics assembly exploded-view

Table 4.2: Electronics assembly parts list

Item	Qty	Part Name / Description	Source
1	1	Arduino Case	3D Printed
2	1	Breadboard Base	3D Printed
3	1	Half-Size Breadboard	Commercial
4	1	Left Wall	Acrylic Laser Cut
5	1	Right Wall	Acrylic Laser Cut
6	1	Top Wall	Acrylic Laser Cut
7	1	Bottom Wall	Acrylic Laser Cut
8	1	DC Power Input Jack - DS-223B	Commercial
9	1	Power Switch - KCD1-11-2P	Commercial
10	1	Red Push Button - DS 212	Commercial
11	2	IDC 3020-20-0200-00 - 20P 2.54MM	Commercial
12	1	IDC Support	3D Printed
13	1	MX M12 5-Pin Male MIC Connector Plug	Commercial
14	1	Arduino MEGA 2650	Commercial

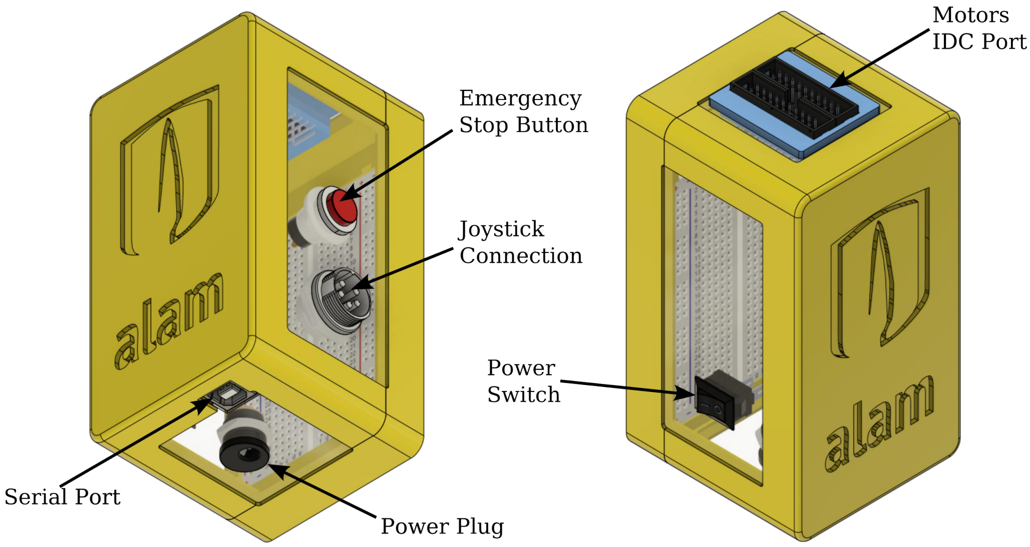


Figure 4.3: Ports and connections

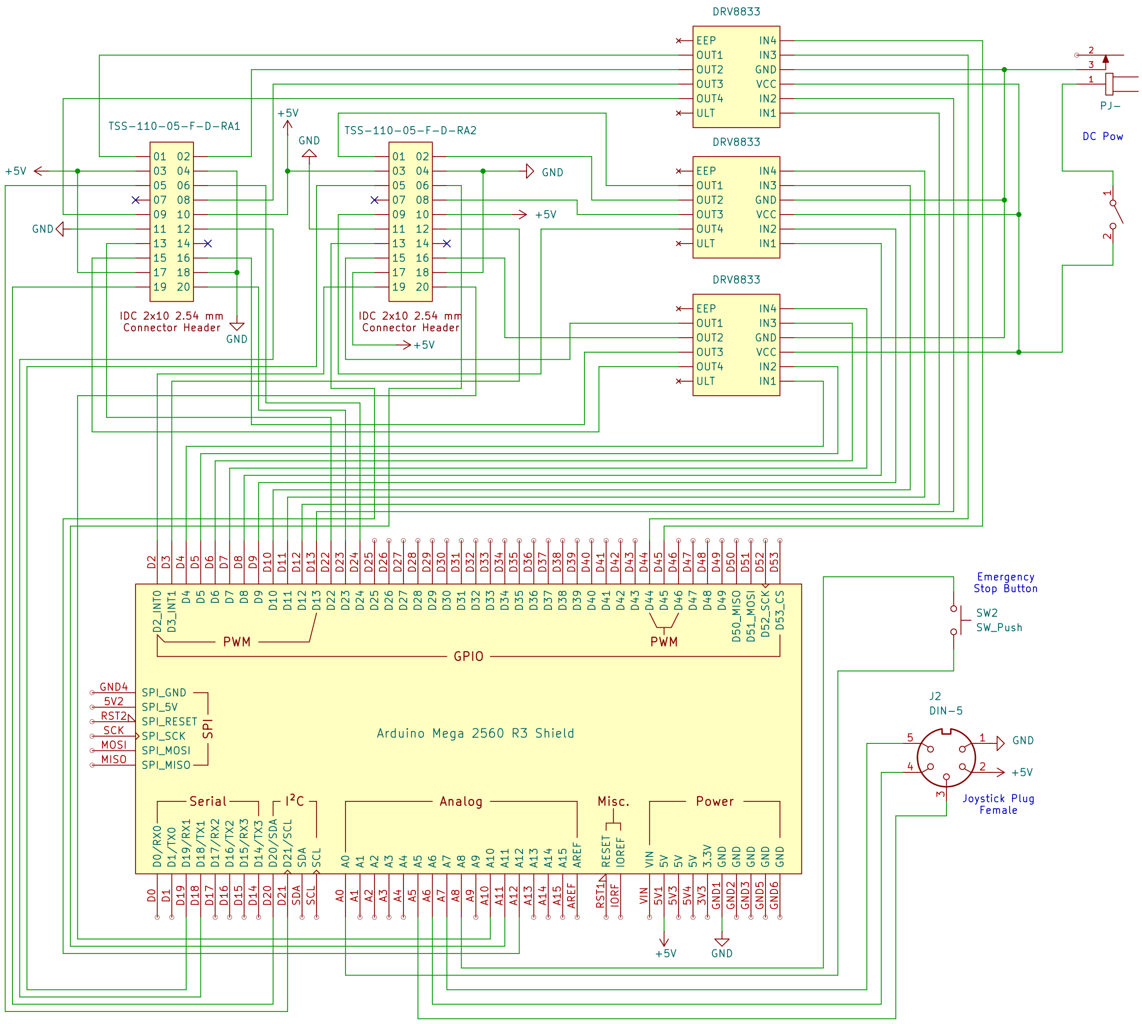


Figure 4.4: Main circuit schematic

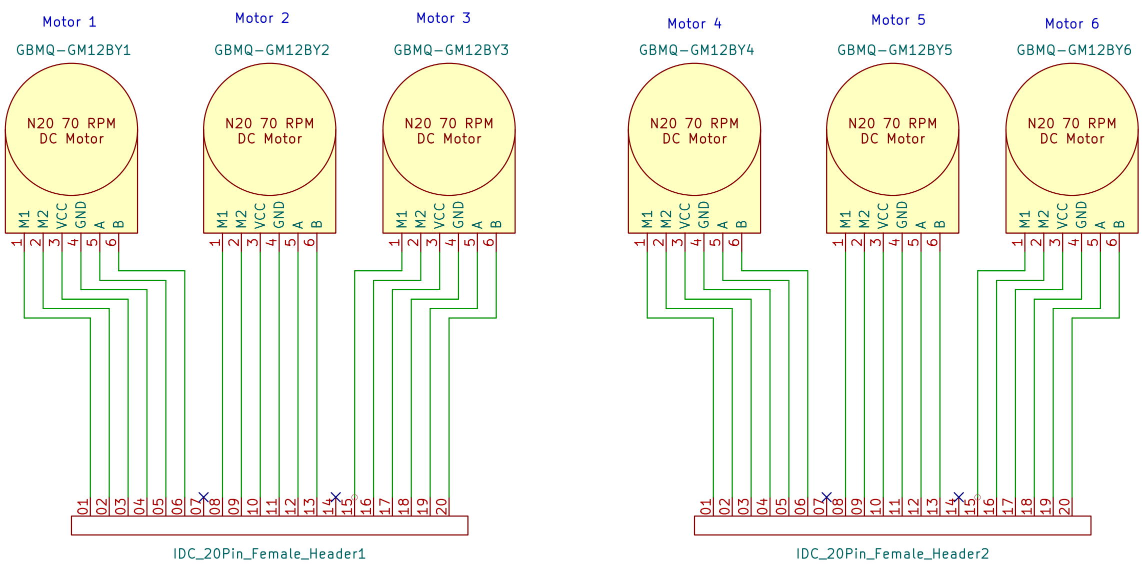


Figure 4.5: Motor circuit schematic

Chapter 5

Control System Interface

5.1 Protocol Specifications

Table 5.1: G-Code definitions

G-Code	Definition
`G1`	Move one or more actuators to specified positions at a given optional speed.
`G4`	Pause or dwell. Halts the machine for the specified duration.
`G28`	Move actuators to origin position.
`G28.1`	Move actuators to target position.
`G90`	Set absolute positioning mode. All subsequent movements are interpreted as absolute positions. This is the default mode.
`G91`	Set relative positioning mode. All subsequent movements are interpreted as displacements from the current position.
`G92`	Set origin position.
`G92.1`	Set target position. This may be a position of interest.

Table 5.2: M-Code definitions

M-Code	Definition
`M00`	Stop program
`M85`	Maximum command wait time.
`M114`	Report current position of the actuators.
`M301`	Set PID parameters.
`M303`	Initiate autotuning process for the PID parameters.
`M355`	Report current status of robot parameters.
`M500`	Save parameters to EEPROM.
`M501`	Load parameters from EEPROM.
`M502`	Reset parameters saved in EEPROM to defaults.
`M503`	Display the saved configuration in EEPROM.

5.2 Command Line Application

5.2.1 Architecture

5.2.2 Usage

5.3 PID Control and Tuning

5.3.1 Test Data

5.3.2 System Identification

$$tf = \frac{(4.808 - 4.596z^{-1}) \times 10^{-4}}{1 - 1.982z^{-1} + 0.9824z^{-2}} \tag{5.1}$$

5.3.3 PID Tuning

$$C = Kp + Ki * \frac{Ts}{z - 1} \tag{5.2}$$

Table 5.3: G-Codes usage

Code	Usage	Example
G1	G1 <id><pos> [F<speed,255>]	> G1 A100.0
G4	G4 P<millis>|S<secs>	> G4 P3000
G20	G20	> G20
G21	G21	> G21
G28	G28 [<id,A B C I J K>]	> G28 A B C
G28	G28.1 [<id,A B C I J K>]	> G28.1 J K
G90	G90	> G90
G91	G91	> G91
G92	G92 [<id,A B C I J K><pos,0>]	> G92 A-100.0 I5.0
G92.1	G92.1 [<id, A B C I J K><pos,0>]	> G92.1 B250.0

Table 5.4: M-Codes usage

Note that only commands with parameters are explained. All others can be used by simply entering the code.

Code	Usage	Example
M85	M85 P<millis,5000>|S<secs,5>	> M85 S3
M301	M301 [P<kp,0>] [I<ki,0>] [D<kd,0>]	> M301 P1000 I0.5
M500	M500 <optn>	> M500 T P
M502	M502 <optn>	> M502 I J K

Table 5.5: Code parameters

Parameter	Meaning
`<id>`	Actuator identification letter: `A`, `B`, `C`, `I`, `J` or `K`.
`<pos>`	Position number.
`<kp>`	Proportional constant.
`<ki>`	Integral constant.
`<kd>`	Derivative constant.
`<speed>`	Maximum speed of actuator as integer number from `50` to `255`.
`<millis>`	Milliseconds as integer number.
`<secs>`	Seconds as integer number.
`<optn>`	Parameter of the robot. `T` for PID tuning constants; `A`, `B`, `C`, `I`, `J` or `K` for position of given actuator; `P` for **all** actuator positions
`<...,VALUE>`	Default value.
`[...]`	Optional parameter. If omitted, default value will be used.

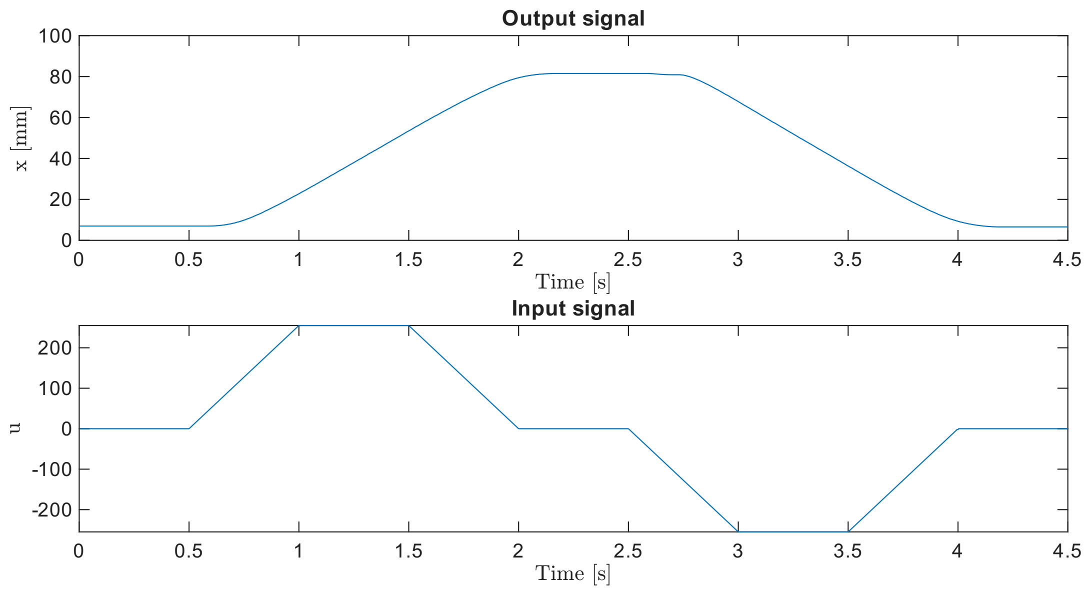


Figure 5.1: Input and output signals

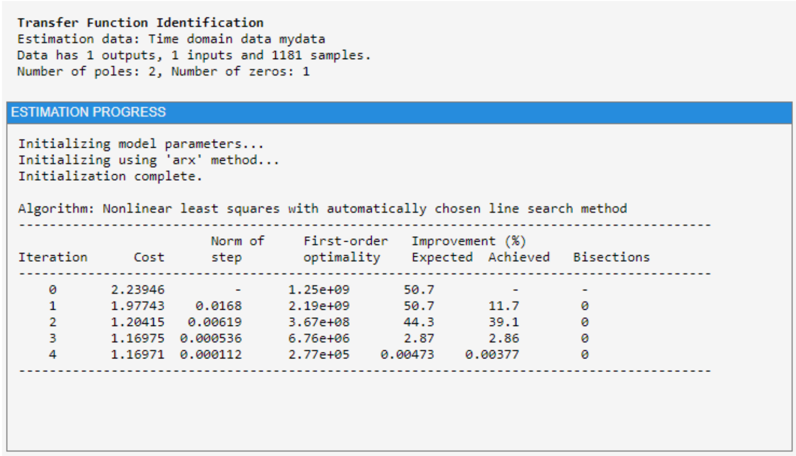


Figure 5.2: Transfer function identification in MATLAB

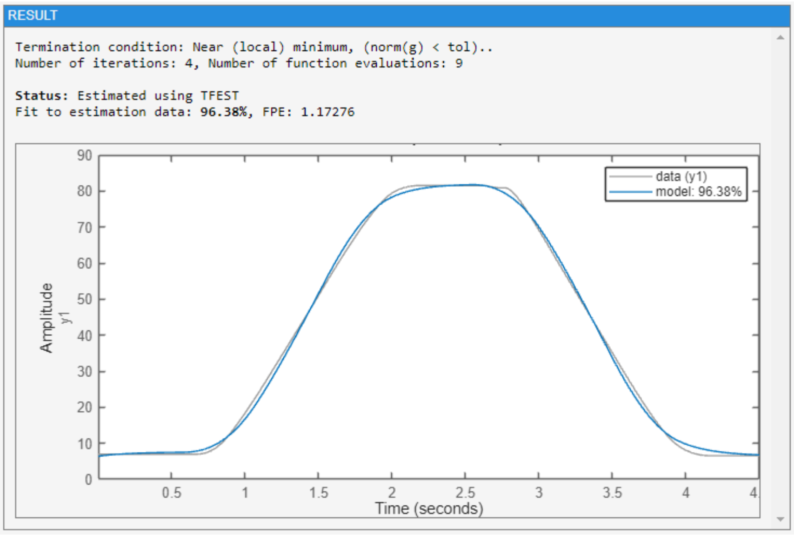


Figure 5.3: Transfer function model in MATLAB

Table 5.6: PID Tunings

Property	Value
Proportional constant, Kp	21.4
Integrative constant, Ki	1.97
Sample time, Ts	0.00381s

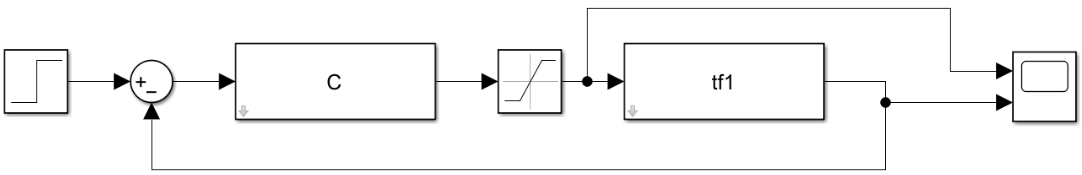


Figure 5.4: Blocks diagram of control system

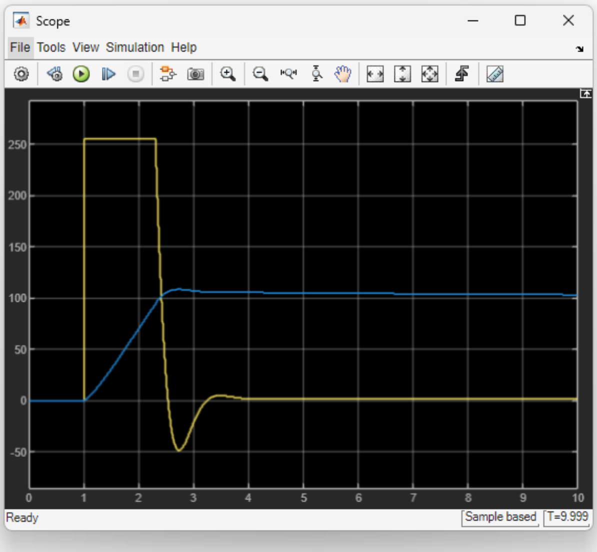


Figure 5.5: System response of simulation in Simulink

Chapter 6

Performance and Results

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Conclusions

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Future Work and Recommendations

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Bibliography

[1] Z. Shao, D. Zhang, and S. Caro, "New frontiers in parallel robots," *Machines*, vol. 11, no. 3, 2023, ISSN: 2075-1702. DOI: `10.3390/machines11030386`.

[2] D. Stewart, "A platform with six degrees of freedom," *Proceedings of the Institution of Mechanical Engineers*, vol. 180, no. 1, pp. 371–386, 1965. DOI: `10.1243/PIME_PROC_1965_180_029_02`.

[3] E. News. "The delta robot – swiss-made and fastest in the world!" (), [Online]. Available: `https://actu.epfl.ch/news/the-delta-robot-swiss-made-and-fastest-in-the-worl/` (visited on 06/14/2024).

[4] S. Briot and J. Burgner-Kahrs, "Editorial: New frontiers in parallel robotics," *Frontiers in Robotics and AI*, vol. 10, 2023, ISSN: 2296-9144. DOI: `10.3389/frobt.2023.1282798`.

[5] M. Russo, S. M. H. Sadati, X. Dong, *et al.*, "Continuum robots: An overview," *Advanced Intelligent Systems*, vol. 5, no. 5, p. 2 200 367, 2023. DOI: `https://doi.org/10.1002/aisy.202200367`.

[6] G. Wu and G. Shi, "Design, modeling, and workspace analysis of an extensible rod-driven parallel continuum robot," *Mechanism and Machine Theory*, vol. 172, p. 104 798, 2022, ISSN: 0094-114X. DOI: `https://doi.org/10.1016/j.mechmachtheory.2022.104798`.

[7] A. L. Orekhov, V. A. Aloi, and D. C. Rucker, "Modeling parallel continuum robots with general intermediate constraints," in *2017 IEEE International Con-*

ference on Robotics and Automation (ICRA), 2017, pp. 6142–6149. DOI: 10.1109/ICRA.2017.7989728.

Appendices

A Forces in Actuator Arm

```
syms Fb P rAx rAy rBx rBy

vecFb = [Fb 0 0];
vecP = [0 P 0];
rA = [rAx rAy 0];
rB = [rBx rBy 0];

eq = cross(rA, vecP) + cross(rB, vecFb);
solve(eq, Fb)
```

B PID Tuning Code

```
% Get data
dataTable = readtable("datos.csv");
t = dataTable.t/1000;  %convert time from milliseconds to seconds
PWM = dataTable.PWM;
A = dataTable.A;
startTime = t(1);
sampleTime = mean(diff(t));
mydata = iddata(A,PWM,sampleTime,"Tstart", startTime);

% Find transfer function of system
npoles = 2; nzeros = 1;
Options = tfestOptions;
Options.Display = 'on';
Options.EnforceStability = true;
tf1 = tfest(mydata, npoles, nzeros, Options, 'Ts', sampleTime, ...
    'Feedthrough', true);

% PID Tune
opts = pidtuneOptions("PhaseMargin", 90);
[C, info] = pidtune(tf1, 'PI', 5, opts);
```

C Test Code

```
;ALAM ROBOT TEST
;BY LICONAJ (c) 2024

G4 P5000  ; WAIT 3 SECONDS
;MOVE ACTUATOR A
G1 A-60
G4 P500
G1 A0
G4 P500
;MOVE ACTUATOR B
G1 B-60
G1 P500
G1 B0
G4 P500
;MOVE ACTUATOR C
G1 C-60
G1 C0
;MOVE ACTUATORS A AND B
G4 P1000
G1 A-60 B-60
G4 P500
G1 A0 B0
;MOVE ACTUATORS B AND C
G4 P1000
G1 B-60 C-60
G4 P500
G1 B0 C0
;MOVE ACTUATORS C AND A
G4 P1000
G1 C-60 A-60
G4 P500
G1 C0 A0
;MOVE ALL ACTUATORS
G4 P1000
G1 A-60 B-60 C-60
G4 P1000
;MOVE ACTUATORS A B C TO HOME
G28 A B C
```

D Command Line Interface Source Code

```python
#!/usr/bin/python3
#alamctl.py

import serial
import sys
import argparse
import time
import serial.tools.list_ports

EOC = "!EOC"
EOL = "\r\n"

def get_ports():
    ports = []
    patterns = ["COM", "ttyACM", "ttyUSB"]
```

```
    for pattern in patterns:
        ports += serial.tools.list_ports.grep(pattern)
    return ports

def list_ports():
    ports = get_ports()
    for port in ports:
        print("{0:<20} \t {1:<30}".format(port.device, port.manufacturer))

parser = argparse.ArgumentParser()
parser.add_argument("-p", "--port", help="set device port")
parser.add_argument("-f", "--file", help="open G-CODE file")
parser.add_argument("-l", "--list", help="list available devices", action="store_true")
args = parser.parse_args()

if args.list:
    list_ports()
    sys.exit()

if args.port:
    port = args.port
else:
    ports = get_ports()
    print(f"{len(ports)} device(s) found")
    if not ports:
        print("No serial ports available")
        sys.exit()
    else:
        port = ports[0].device
    print(f"Connecting to {port}...")

try:
    mcu = serial.Serial(port=port, baudrate=250000, timeout=.1)
    time.sleep(1)
    reponse = mcu.readline().decode()
    if not reponse.startswith("this is alam"):
        print("Incompatible robot")
        sys.exit()
except serial.SerialException:
    print("Serial port unavailable")
    sys.exit()

def send_command(value: str):
    mcu.write(bytes(value, "utf-8"))

def parse_command(cmd: str) -> str:
    cmd = cmd.strip().split(";")[0]
    cmd += EOL
    # print(f"Parsing command {repr(cmd)}")
    send_command(cmd)
    output = ""
    value = ""
    while not EOC in value:
        value = mcu.readline().decode()
        if not EOC in value:
            output += value
    return output

if __name__ == "__main__":
    if args.file is None:
        print("\nAlam Control Center")
        print("By Josue Licona (C) 2024\n")

        print("G to show G-CODE commands")
        print("M to show M-CODE commands")
        print("Ctrl+C or ;EXIT to quit\n")

        try:
            while True:
```

```
                cmd = input("alam> ").upper()
                if ";EXIT" in cmd:
                    break
                elif not cmd or cmd.startswith(";"):
                    continue
                print(parse_command(cmd))
        except KeyboardInterrupt:
            print("\n\nPROGRAM ENDED BY USER\n")

    else:
        time.sleep(1)
        try:
            with open(args.file, "r", encoding="utf-8") as fh:
                lines = fh.readlines()
                fh.close()
        except FileNotFoundError:
            print(f"File `{args.file}` not found")
            sys.exit()

        for cmd in lines:
            cmd = cmd.strip().upper()
            if not cmd or cmd.startswith(";"):
                continue
            print(f"> {cmd.strip()}")
            print(parse_command(cmd))
```

E Firmware Source Code

E.1 Arduino Main Entry Point

```
// AlamWare.ino

#include "Memory.h"
#include "Commands.h"

// SerialCommands Setup
char serialCommandBuffer[128];
SerialCommands serialCommands(&Serial, serialCommandBuffer, sizeof(serialCommandBuffer));

void addSerialCommands() {
    serialCommands.SetDefaultHandler(cmdUnrecognized);

    serialCommands.AddCommand(&GHelp);
    serialCommands.AddCommand(&G1);
    serialCommands.AddCommand(&G4);
    serialCommands.AddCommand(&G92);
    serialCommands.AddCommand(&G92_1);
    serialCommands.AddCommand(&G28);
    serialCommands.AddCommand(&G28_1);
    serialCommands.AddCommand(&G90);
    serialCommands.AddCommand(&G91);

    serialCommands.AddCommand(&MHelp);
    serialCommands.AddCommand(&M00);
    serialCommands.AddCommand(&M85);
    serialCommands.AddCommand(&M114);
    serialCommands.AddCommand(&M303);
    serialCommands.AddCommand(&M301);
    serialCommands.AddCommand(&M300);
}

void setup() {
```

```
    Serial.begin(250000);
    Serial.print("this is alam");
    addSerialCommands();
}

void loop() {
    if (testing) {
    updateTest();
    } else {
    serialCommands.ReadSerial();

    updateNamedActuators();
    nextCommand();
    }
}
```

E.2 Persistence Memory Management Class Header

```
// Memory.h

#ifndef MEMORY_H
#define MEMORY_H

#include <EEPROM.h>

class Memory {
 private:
  float kp;
  float kd;
  float ki;

  uint8_t kpAddress = 10;
  uint8_t kiAddress = 12;
  uint8_t kdAddress = 14;

  int lastPositions[6];
  uint8_t pAddresses[6];

 public:
  Memory();
  ~Memory();

  int GetLastPosition(uint8_t actuatorId);
  void SavePosition(uint8_t actuatorId, int position);
  void GetTunings(float* kp, float* ki, float* kd);
  void SaveTunings(float kp, float ki, float kd);
};

#endif
```

E.3 Persistence Memory Management Class

```
// Memory.cpp
#include "Memory.h"

Memory::Memory() {
  for (int i = 0; i < 6; i++) {
    pAddresses[i] = 20 + sizeof(float) * i;
  }
}
```

```
Memory::~Memory() {
}

void Memory::SavePosition(uint8_t actuatorId, int position) {
  EEPROM.put(pAddresses[actuatorId], position);
}

int Memory::GetLastPosition(uint8_t actuatorId) {
  return lastPositions[actuatorId];
}

void Memory::SaveTunings(float kp, float ki, float kd) {
  EEPROM.put(kpAddress, kp);
  EEPROM.put(kiAddress, ki);
  EEPROM.put(kdAddress, kd);
}

void Memory::GetTunings(float* kp, float* ki, float* kd) {
  EEPROM.get(kpAddress, kp);
  EEPROM.get(kiAddress, ki);
  EEPROM.get(kdAddress, kd);
}
```

E.4 Commands Definitions

```
// Commands.h

#ifndef COMMANDS_H
#define COMMANDS_H

#include "SerialCommands.h"

#include "NamedActuators.h"
#include "RodActuator.h"

bool sendingCommand = false;
bool testing = false;
unsigned long startTimeTest;

void updateTest();
void nextCommand();

// G-Code Definitions

void cmdUnrecognized(SerialCommands *, const char *);
void cmdGetGCodes(SerialCommands *);
void cmdDelay(SerialCommands *);
void cmdMoveActuator(SerialCommands *);
void cmdSetGlobalMode(SerialCommands *);
void cmdSetRelativeMode(SerialCommands *);
void cmdSetHome(SerialCommands *);
void cmdSetTarget(SerialCommands *);
void cmdMoveToHome(SerialCommands *);
void cmdMoveToTarget(SerialCommands *);

// M-Code Definitions

void cmdGetMCodes(SerialCommands *);
void cmdStop(SerialCommands *);
void cmdSetTimeout(SerialCommands *);
void cmdGetActuatorPositions(SerialCommands *);
void cmdTestActuators(SerialCommands *);
void cmdStartAutoTuning(SerialCommands *);
void cmdSetPIDConstants(SerialCommands *);

SerialCommand GHelp("G", &cmdGetGCodes);
SerialCommand G4("G4", &cmdDelay);
SerialCommand G1("G1", &cmdMoveActuator);
SerialCommand G28("G28", &cmdMoveToHome);
SerialCommand G28_1("G28.1", &cmdMoveToTarget);
SerialCommand G90("G90", &cmdSetGlobalMode);
SerialCommand G91("G91", &cmdSetRelativeMode);
SerialCommand G92("G92", &cmdSetHome);
SerialCommand G92_1("G92.1", &cmdSetTarget);

SerialCommand MHelp("M", &cmdGetMCodes);
SerialCommand M00("M00", &cmdStop);
SerialCommand M85("M85", &cmdSetTimeout);
```

```
SerialCommand M114("M114", &cmdGetActuatorPositions);
SerialCommand M300("M300", &cmdTestActuators);
SerialCommand M303("M303", &cmdStartAutoTuning);
SerialCommand M301("M301", &cmdSetPIDConstants);

void endCmd(SerialCommands *sender) {
  sender->GetSerial()->println("!EOC");
}

void nextCommand() {
  if (!sendingCommand)
    return;
  for (NamedActuator actuator : namedActuators) {
    if (!actuator.object->Next()) {
      return;
    }
  }
  Serial.println("!EOC");
  sendingCommand = false;
}

float getvel(int t) {
  float v;
  float m = 255.0 / 500.0;
  if (t < 500 || t > 4000) {
    v = 0;
  } else if (t < 1000) {  // entre 500 y 1000 ms rampa
    v = m * (t - 500);
  } else if (t < 1500) {  // mantener velocidad máxima por 500 ms
    v = 255;
  } else if (t < 2000) {  // disminuir velocidad
    v = -m * (t - 1500) + 255;
  } else if (t < 2500) {  // mantener velocidad máxima en reversa por 500 ms
    v = 0;
  } else if (t < 3000) {
    v = -m * (t - 2500);
  } else if (t < 3500) {
    v = -255;
  } else if (t < 4000) {
    v = m * (t - 3500) - 255;
  }
  return v;
}

void updateTest() {
  unsigned long t = millis() - startTimeTest;
  Serial.print(t);
  float v = getvel(t);
  Serial.print(";");
  Serial.print(v);
  for (NamedActuator actuator : namedActuators) {
    Serial.print(";");
    actuator.object->TestVelocity(v);
    actuator.object->Update();
    Serial.print(actuator.object->GetPosition());
  }
  Serial.println("");
  if (t >= 4500) {
    testing = false;
    for (NamedActuator actuator : namedActuators) {
      actuator.object->Stop();
    }
    Serial.println("!EOC");
  }
}

void cmdUnrecognized(SerialCommands *sender, const char *cmd) {
  sendingCommand = true;

  sender->GetSerial()->print("Unrecognized command [");
  sender->GetSerial()->print(cmd);
  sender->GetSerial()->println("]");
}

void cmdGetGCodes(SerialCommands *sender) {
  sendingCommand = true;

  sender->GetSerial()->println("G: Show this help");
  sender->GetSerial()->println("G1: Move actuator");
  sender->GetSerial()->println("G4: Pause robot by a determined P<time>");
  sender->GetSerial()->println("G28: Move actuators to home position");
  sender->GetSerial()->println("G28.1: Move actuators to target position");
  sender->GetSerial()->println("G90: Set global positioning mode");
  sender->GetSerial()->println("G91: Set relative positioning mode");
  sender->GetSerial()->println("G92: Set home position");
  sender->GetSerial()->println("G92.1: Set target position");
}

void cmdGetMCodes(SerialCommands *sender) {
```

```
  sendingCommand = true;

  sender->GetSerial()->println("M: Show this help");
  sender->GetSerial()->println("M00: Stop actuators");
  sender->GetSerial()->println("M85: Set command timeout with P<milliseconds> or S<seconds>");
  sender->GetSerial()->println("M114: Report actual position of actuators");
  sender->GetSerial()->println("M300: Test actuators");
  sender->GetSerial()->println("M301: Set PID constants with P<value> I<value> D<value>");
  sender->GetSerial()->println("M303: Starts autotuning");
}

void cmdDelay(SerialCommands *sender) {
  sendingCommand = true;

  // sender->GetSerial()->println("Starting delay");
  char *arg = sender->Next();
  if (arg != NULL) {
    char parameter = arg[0];
    float value = atof(arg + 1);
    if (parameter == 'P') {
      delay(value);
    }
  }
}

void cmdMoveActuator(SerialCommands *sender) {
  sendingCommand = true;
  uint8_t speed = 255;
  char *arg;
  while ((arg = sender->Next()) != NULL) {
    char id = arg[0];
    float value = atof(arg + 1);
    if (id == 'F') {
      speed = value;
      continue;
    }
    RodActuator *actuator = getActuatorByName(id);
    actuator->Move(value);
    actuator->SetMaxSpeed(speed);
  }
}

void cmdStop(SerialCommands *sender) {
  sendingCommand = true;
  for (NamedActuator actuator : namedActuators) {
    actuator.object->Stop();
  }
}

void cmdSetTimeout(SerialCommands *sender) {
  sendingCommand = true;
  float maximumTimeout = 60e3;
  float timeout = 5000;
  char *arg = sender->Next();
  if (arg != NULL) {
    char parameter = arg[0];
    float value = atof(arg + 1);
    if (parameter == 'P') {
      timeout = value;
    } else if (parameter == 'S') {
      timeout = value * 1000;
    }
    if (timeout > maximumTimeout)
      timeout = maximumTimeout;
    for (NamedActuator actuator : namedActuators) {
      actuator.object->SetTimeout(timeout);
    }
  } else {
    timeout = namedActuators[0].object->GetTimeout();
  }
  sender->GetSerial()->print("Timeout:");
  sender->GetSerial()->println(timeout);
}

void cmdGetActuatorPositions(SerialCommands *sender) {
  sendingCommand = true;

  int decimalPlaces = 5;

  String positions = "";
  for (int i = 0; i < N_ACTUATORS; ++i) {
    String name = (String)namedActuators[i].name;
    float position = namedActuators[i].object->GetPosition();
    positions += name + ":" + String(position);
    if (i < N_ACTUATORS - 1) {
      {
        positions += " ";
      }
    }
```

```
  }
  sender->GetSerial()->println(positions);
}

void cmdStartAutoTuning(SerialCommands *sender) {
  sendingCommand = true;

  char *arg;
  while ((arg = sender->Next()) != NULL) {
    char id = arg[0];
    Serial.print("Starting auto tuning in actuator ");
    Serial.println(id);
    RodActuator *actuator = getActuatorByName(id);
    actuator->AutoTune();
  }
}

void cmdTestActuators(SerialCommands *sender) {
  testing = true;
  startTimeTest = millis();
  sender->GetSerial()->print("t");
  sender->GetSerial()->print(";PWM");
  for (NamedActuator actuator : namedActuators) {
    sender->GetSerial()->print(";");
    sender->GetSerial()->print(actuator.name);
  }
  sender->GetSerial()->println("");
}

void cmdSetPIDConstants(SerialCommands *sender) {
  sendingCommand = true;

  float kp = 0;
  float ki = 0;
  float kd = 0;

  char *arg;
  while ((arg = sender->Next()) != NULL) {
    char id = arg[0];
    float value = atof(arg + 1);
    if (id == 'P')
      kp = value;
    else if (id == 'I')
      ki = value;
    else if (id == 'D')
      kd = value;
  }

  for (NamedActuator actuator : namedActuators) {
    actuator.object->SetPIDConstants(kp, ki, kd);
  }

  sender->GetSerial()->print("Kp:");
  sender->GetSerial()->print(kp);
  sender->GetSerial()->print(" Kd:");
  sender->GetSerial()->print(kd);
  sender->GetSerial()->print(" Ki:");
  sender->GetSerial()->println(ki);
}

void cmdSetGlobalMode(SerialCommands *sender) {
  for (NamedActuator actuator : namedActuators) {
    actuator.object->SetMoveMode(actuator.object->Global);
  }
}

void cmdSetRelativeMode(SerialCommands *sender) {
  for (NamedActuator actuator : namedActuators) {
    actuator.object->SetMoveMode(actuator.object->Relative);
  }
}

void cmdSetHome(SerialCommands *sender) {
  sendingCommand = true;
  char *arg;
  while ((arg = sender->Next()) != NULL) {
    char id = arg[0];
    float value = atof(arg + 1);
    RodActuator *actuator = getActuatorByName(id);
    actuator->SetHomePosition(value);
  }
}

void cmdSetTarget(SerialCommands *sender) {
  sendingCommand = true;
  char *arg;
  while ((arg = sender->Next()) != NULL) {
    char id = arg[0];
    float value = atof(arg + 1);
```

```
    RodActuator *actuator = getActuatorByName(id);
    actuator->SetTargetPosition(value);
  }
}

void cmdMoveToHome(SerialCommands *sender) {
  sendingCommand = true;
  char *arg;
  arg = sender->Next();
  if (arg == NULL) {
    for (NamedActuator actuator : namedActuators) {
      actuator.object->MoveToHome();
    }
  } else {
    char id = arg[0];
    float value = atof(arg + 1);
    RodActuator *actuator = getActuatorByName(id);
    actuator->MoveToHome();

    while ((arg = sender->Next()) != NULL) {
      id = arg[0];
      float value = atof(arg + 1);
      actuator = getActuatorByName(id);
      actuator->MoveToHome();
    }
  }
}

void cmdMoveToTarget(SerialCommands *sender) {
  sendingCommand = true;
  char *arg;
  arg = sender->Next();
  if (arg == NULL) {
    for (NamedActuator actuator : namedActuators) {
      actuator.object->MoveToTarget();
    }
  } else {
    char id = arg[0];
    float value = atof(arg + 1);
    RodActuator *actuator = getActuatorByName(id);
    actuator->MoveToTarget();

    while ((arg = sender->Next()) != NULL) {
      id = arg[0];
      float value = atof(arg + 1);
      actuator = getActuatorByName(id);
      actuator->MoveToTarget();
    }
  }
}

#endif
```

E.5 Actuators Instances

```
// NamedActuators.h
#ifndef NAMEDACTUATORS_H
#define NAMEDACTUATORS_H

#include "RodActuator.h"

// 1.1
#define A_ENA 21
#define A_ENB 24
#define A_IN1 13
#define A_IN2 12
// 1.3
#define B_ENA 20
#define B_ENB 23
#define B_IN1 6
#define B_IN2 7
// 2.2
#define C_ENA 3
#define C_ENB 66
#define C_IN1 10
#define C_IN2 11
// 2.1
#define I_ENA 19
#define I_ENB 65
#define I_IN1 9
#define I_IN2 8
// 2.3
#define J_ENA 2
#define J_ENB 64
#define J_IN1 5
#define J_IN2 4
```

```
// 1.2
#define K_ENA 18
#define K_ENB 22
#define K_IN1 44
#define K_IN2 45

enum Actuators {
  A, B, C, I, J, K,
  N_ACTUATORS
};

struct NamedActuator {
  char name;
  RodActuator *object;
};

// Posiciones
volatile long angPosition[N_ACTUATORS] = {0, 0, 0, 0, 0, 0};

//
// Interrupt Handlers
//
static void A_interruptHandler() {
  if (digitalRead(A_ENA) != digitalRead(A_ENB))
    angPosition[A]++;
  else
    angPosition[A]--;
}
static void B_interruptHandler() {
  if (digitalRead(B_ENA) != digitalRead(B_ENB))
    angPosition[B]++;
  else
    angPosition[B]--;
}

static void C_interruptHandler() {
  if (digitalRead(C_ENA) != digitalRead(C_ENB))
    angPosition[C]++;
  else
    angPosition[C]--;
}

static void I_interruptHandler() {
  if (digitalRead(I_ENA) != digitalRead(I_ENB))
    angPosition[I]++;
  else
    angPosition[I]--;
}

static void J_interruptHandler() {
  if (digitalRead(J_ENA) != digitalRead(J_ENB))
    angPosition[J]++;
  else
    angPosition[J]--;
}

static void K_interruptHandler() {
  if (digitalRead(K_ENA) != digitalRead(K_ENB))
    angPosition[K]++;
  else
    angPosition[K]--;
}

NamedActuator namedActuators[N_ACTUATORS] = {
    {'A', new RodActuator(A_ENA, A_ENB, A_IN1, A_IN2, angPosition[A], A_interruptHandler)},
    {'B', new RodActuator(B_ENA, B_ENB, B_IN1, B_IN2, angPosition[B], B_interruptHandler)},
    {'C', new RodActuator(C_ENA, C_ENB, C_IN1, C_IN2, angPosition[C], C_interruptHandler)},
    {'I', new RodActuator(I_ENA, I_ENB, I_IN1, I_IN2, angPosition[I], I_interruptHandler)},
    {'J', new RodActuator(J_ENA, J_ENB, J_IN1, J_IN2, angPosition[J], J_interruptHandler)},
    {'K', new RodActuator(K_ENA, K_ENB, K_IN1, K_IN2, angPosition[K], K_interruptHandler)},
};

RodActuator *getActuatorByName(char name) {
  for (int i = 0; i < N_ACTUATORS; ++i) {
    if (namedActuators[i].name == name) {
      return namedActuators[i].object;
    }
  }
  return nullptr;
}

void updateNamedActuators() {
  for (NamedActuator actuator : namedActuators) {
    actuator.object->Update();
  }
}

#endif
```

E.6 Actuator Class Header

```
//RodActuator.h
#ifndef RODACTUATOR_H
#define RODACTUATOR_H

#include <Arduino.h>
#include <QuickPID.h>
#include <sTune.h>

class RodActuator {
 public:
  RodActuator(uint8_t ENA, uint8_t ENB, uint8_t IN1, uint8_t IN2,
    volatile long &angPosition, void (*interruptHandler)());

  enum MoveMode {
    Global,
    Relative
  };

  void Stop();
  void Update();
  void SetPIDConstants(float Kp, float Ki, float Kd);
  void Move(float desiredPosition);
  float GetPosition();
  int GetTimeout();
  void TestVelocity(float velocity);
  void SetLeftLimit(float leftLimit);
  void SetRightLimit(float rightLimit);
  void SetMaxSpeed(uint8_t maxSpeed);
  void SetMoveMode(MoveMode);
  void SetTimeout(float timeout);
  void MoveToHome();
  void MoveToTarget();
  void SetHomePosition(float newHomePosition);
  void SetTargetPosition(float newSecondHomePosition);
  float GetHomePosition();
  float GetTargetPosition();
  void AutoTune();
  bool Next();

 private:
  float homePosition = 0;
  float targetPosition = 0;

  bool isStable(float threshold);
  volatile long &angPosition;
  uint8_t IN1;  // Pin forward velocity
  uint8_t IN2;  // Pin backwards velocity
  uint8_t ENA;  // Pin encoder A
  uint8_t ENB;  // Pin encoder B

  float ticksRev = 7.0;      // ticks in angPosition that makes a rev
  float gearRatio = 210.0;  // reducion gear of gearRatio:1
  float R = 5.3;
  float positionFactor;

  uint8_t speed = 255;
  unsigned long timeout = 5000;
  unsigned long timer = millis();
  float position = 0;
  float lastPosition = 0;
  float desiredPosition = 0;
  float lastAngvelocity = 0;
  float angVelocity = 0;  // motor angular position

  MoveMode moveMode = Global;

  bool next = true;
  bool move = false;
  float Kp, Kd, Ki;
  float outputMin = -255;
  float outputMax = 255;
  float outputSpan;
  bool startup = false;
  QuickPID PID;  // Controlador PID

  bool test = false;
  bool tune = false;
  sTune tuner = sTune(&position, &angVelocity, tuner.NoOvershoot_PI, tuner.directIP, tuner.printALL);

  float _leftLimit = -2000;
  float _rightLimit = 2000;

  void updateVelocity();
  void updatePosition();
};

#endif
```

E.7 Actuator Class

```cpp
//RodActuator.cpp
#include "RodActuator.h"
#include <Arduino.h>
#define DEBUG false
RodActuator::RodActuator(uint8_t ENA, uint8_t ENB, uint8_t IN1, uint8_t IN2,
    volatile long &angPosition, void (*interruptHandler)()) :
    ENA(ENA), ENB(ENB), IN1(IN1), IN2(IN2), angPosition(angPosition) {
  pinMode(ENA, INPUT);
  pinMode(ENB, INPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(ENA), interruptHandler, CHANGE);

  // Calculate position factor
  // 1 Revolución en el encoder son 7 cambios
  // La relación de engranages de 235:1
  // El valor de _position está en revoluciones
  // 1/(7*235) = 6.0790273556231e-4
  positionFactor = 2.0 * R * PI / (ticksRev * gearRatio);

  // Inicializar control PID
  PID = QuickPID(&position, &angVelocity, &desiredPosition);
  PID.SetOutputLimits(outputMin, outputMax);
  PID.SetSampleTimeUs(2030);
  PID.SetDerivativeMode(QuickPID::dMode::dOnError);
  PID.SetProportionalMode(QuickPID::pMode::pOnError);
  PID.SetAntiWindupMode(QuickPID::iAwMode::iAwCondition);
  PID.SetMode(QuickPID::Control::automatic);
  PID.SetTunings(22.1, 67, 3.8632);
}

void RodActuator::Update() {
  updatePosition();

  if (tune) {
    switch (tuner.Run()) {
      case tuner.sample:
        updateVelocity();
        break;

      case tuner.tunings:
        angVelocity = 0;
        updateVelocity();
        tuner.GetAutoTunings(&Kp, &Ki, &Kd);
        SetPIDConstants(Kp, Ki, Kd);
        tune = false;
        next = true;
        break;

      default:
        break;
    }
  }

  else if (move) {
    if (startup && position > desiredPosition - 10) {
      startup = false;
      angVelocity = 0.01 * angVelocity;
      updateVelocity();
      PID.SetMode(QuickPID::Control::manual);
      PID.SetMode(QuickPID::Control::automatic);
    }
    PID.Compute();
    updateVelocity();
  }

  else if (test) {
    updateVelocity();
  }

  lastPosition = position;
  lastAngvelocity = angVelocity;

  if (isStable(1.5)) {
    next = true;
  }

  if (!next && move) {
    unsigned long currentTime = millis();
    if (currentTime - timer > timeout) {
      next = true;
    }
  }
```

```
}

void RodActuator::Stop() {
  angVelocity = 0;
  next = true;
  move = false;
  test = false;
  updateVelocity();
  updatePosition();
}

void RodActuator::Move(float desiredPosition) {
  next = false;
  timer = millis();
  move = true;
  startup = true;
  if (moveMode == MoveMode::Global) {
    RodActuator::desiredPosition = desiredPosition;
  } else if (moveMode == MoveMode::Relative) {
    RodActuator::desiredPosition += desiredPosition;
  }
}

float RodActuator::GetHomePosition() {
  return homePosition;
}

float RodActuator::GetTargetPosition() {
  return targetPosition;
}

void RodActuator::SetHomePosition(float newHome) {
  RodActuator::homePosition = newHome;
}

void RodActuator::SetTargetPosition(float newSecondHome) {
  RodActuator::targetPosition = newSecondHome;
}

void RodActuator::MoveToHome() {
  float desiredPosition = homePosition;
  if (moveMode == MoveMode::Relative) {
    desiredPosition = homePosition - desiredPosition;
  }
  Move(desiredPosition);
}

void RodActuator::MoveToTarget() {
  float desiredPosition = targetPosition;
  if (moveMode == MoveMode::Relative) {
    desiredPosition = targetPosition - desiredPosition;
  }
  Move(desiredPosition);
}

float RodActuator::GetPosition() {
  return position;
}

void RodActuator::SetPIDConstants(float Kp, float Ki, float Kd) {
  Kp = Kp;
  Ki = Ki;
  Kd = Kd;
  PID.SetTunings(Kp, Ki, Kd);
}

bool RodActuator::Next() {
  return next;
}

void RodActuator::SetMaxSpeed(uint8_t maxSpeed) {
  if (maxSpeed < 50) {
    maxSpeed = 50;
  }
  RodActuator::speed = maxSpeed;
}

void RodActuator::TestVelocity(float velocity) {
  RodActuator::angVelocity = velocity;
  test = true;
}

void RodActuator::SetTimeout(float timeout) {
  RodActuator::timeout = timeout;
}

int RodActuator::GetTimeout() {
  return timeout;
}

void RodActuator::AutoTune() {
  tune = true;
  next = false;
```

```
  tuner.Configure(1000, 200, 100, 0, 3, 1, 500);
  tuner.SetEmergencyStop(1000);
}

void RodActuator::SetMoveMode(MoveMode moveMode) {
  RodActuator::moveMode = moveMode;
}

// =========================================
// Private methods
// =========================================
bool RodActuator::isStable(float threshold = 1) {
  return abs(position - desiredPosition) <= threshold;
}

void RodActuator::updateVelocity() {
  uint8_t vel1 = 0;
  uint8_t vel2 = 0;
  float minInput = 0;
  uint8_t minPWM = 0;
  if (angVelocity > minInput) {
    // vel1 = abs((float)_angVelocity / (float)_outputMax) * (float)_speed;
    vel1 = map(abs(angVelocity), minInput, outputMax, minPWM, RodActuator::speed);
    if (vel1 < 40)
      vel1 = 0;
  } else if (angVelocity < -minInput) {
    // vel2 = abs((float)_angVelocity / (float)_outputMax) * (float)_speed;
    vel2 = map(abs(angVelocity), minInput, outputMax, minPWM, RodActuator::speed);
    if (vel2 < 40)
      vel2 = 0;
  }
  // Serial.println(vel1);
  if (DEBUG && (move && !next)) {
    Serial.print("Velocity: ");
    Serial.print(angVelocity);
    Serial.print(" ");
    Serial.print(vel1);
    Serial.print(" ");
    Serial.print(vel2);
    Serial.print("  Position: ");
    Serial.print(position);
    Serial.println();
  }
  analogWrite(IN1, vel1);
  analogWrite(IN2, vel2);
}

void RodActuator::updatePosition() {
  position = angPosition * positionFactor;
}
```