



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Bachelor's Thesis

Control of Robot for Inserting Detectors for the Large Hadron Collider

Jakub Janoušek
Cybernetics and Robotics

May 2021
Supervisor: Vladimír Smutný



BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Janoušek Jakub** Personal ID number: **483444**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Control of Robot for Inserting Detectors for the Large Hadron Collider

Bachelor's thesis title in Czech:

Řízení robotu pro zakládání křemíkových detektorů urychlovače LHC

Guidelines:

1. Research the requirements for controlling of the robot being designed for insertion of detector assemblies.
2. Propose the HW and SW solution for control of the robot drives.
3. Implement basic functionality of the robot control.
4. Perform experiments with the robot.
5. Evaluate results, propose improvements.

Bibliography / sources:

- [1] Large Hadron Collider Atlas Detector, Inner Tracker Upgrade, <https://cds.cern.ch/record/2302625/files/ATL-ITK-SLIDE-2018-073.pdf>, Hong Kong 2018.
- [2] Franklin Gene F., J. David Powell, Abbas Emami-Naeini: Feedback Control of Dynamic Systems, Pearson Education Limited 2019.
- [3] Paul Acarnley: Stepping Motors a guide to theory and practice, 4th edition, 2002.

Name and workplace of bachelor's thesis supervisor:

Ing. Vladimír Smutný, Ph.D., Robotic Perception, CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **06.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Vladimír Smutný, Ph.D.
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgement / Declaration

I would like to thank my supervisor Vladimír Smutný for all his advice, patience and support during the preparation of this thesis. I would also like to thank the entire CTU team which I can be part of for the support, especially Martin Janda for precious advices and suggestions during the development. Advice and mentoring provided by Václav Vacek was also greatly appreciated.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 21, 2021

.....

Abstrakt / Abstract

Experiment ATLAS, který je součástí urychlovače LHC v CERNu akutálně prochází přípravou na další fázi provozu LHC s vyšší energií svazků. Kvůli této vyšší energii je nutné nahradit stávající vnitřní detektor Inner Detector novějším detektorem Inner Tracker. Bylo nutné navrhnout robota, který bude instalovat senzory do tohoto nového vnitřního detektoru. Vývoj tohoto robota má na starosti tým z ČVUT.

Cílem této práce je navrhnout a implementovat HW a SW řešení pro tohoto robota. Pro jednotlivé krokové motory na robota byly vyvinuty jejich controllery, které jsou propojeny do komunikační sítě RS-485 společně s centrální řídicí jednotkou. Pro komunikaci přes tuto síť mezi jednotlivými řídicí a centrální řídicí jednotkou byl navržen komunikační protokol. Celé řízení robota společně se synchronizací všech krokových motorů má na starosti navržená centrální řídicí jednotka. Celý řídicí systém robota byl úspěšně postaven, otestován a je plně funkční.

Klíčová slova: ATLAS, řídicí systém, robot, krokový motor

Překlad titulu: Řízení robota pro zakládání křemíkových detektorů urychlovače LHC

The ATLAS experiment at the Large Hadron Collider in CERN is currently being upgraded for the high-luminosity phase of the LHC, which require replacement of the Inner Detector with a new Inner Tracker. A robotic assembly is designed by a CTU team to install sensors into the Inner Tracker.

The goal of this thesis is to design and implement HW and SW solution to the control of this robot. The controllers for all the stepper motors were developed. They are connected via a RS-485 network together with a central control unit. A communication protocol was implemented for the connection of the stepper motor controllers and the central control unit. The central control unit is responsible for the synchronization of the stepper motors and control of the whole system. The control system was successfully built, tested and it is fully functional.

Keywords: ATLAS, control system, robot, stepper motor

/ Contents

1 Introduction	1
2 Problem Statement	2
2.1 Description of the degrees of freedom	4
2.1.1 Rotation around the z-axis	5
2.1.2 Radial movement	6
2.1.3 Tilting of the staves	6
2.1.4 Z-indexing	7
2.1.5 Stave insertion	8
2.2 Motors of the robot	9
3 Proposed Solution	10
3.1 Hardware	10
3.1.1 Communication network	10
3.1.2 Central control unit	11
3.1.3 Stepper motor controller	12
3.2 Software	14
3.2.1 Communication protocol	14
3.2.2 Stepper motor controller	17
3.2.3 Central control unit	19
4 Implementation	21
4.1 User Manual	21
4.1.1 Connecting the Raspberry Pi to run the control software	21
4.1.2 Preparing an external computer to run the control software	22
4.1.3 Using the control software	23
4.2 Technical manual	25
4.2.1 Stepper Motor Controller	25
4.2.2 Central Control Unit	27
4.3 Testing the system	28
4.4 Future considered improvements	30
5 Conclusion	32
References	33

Tables / Figures

2.1. Details of the motors used for the robot drives	9	2.1. Cross section of the outer cylinder	2
3.1. Types of the messages	17	2.2. Model of a stave.....	3
4.1. Names of degrees of freedom inside the control script	24	2.3. Model of the robot	3
4.2. List of debugging commands for the controller	26	2.4. Model of the main arm of the robot	4
4.3. List of functions provided for the control of the stepper motor controller	27	2.5. Schematic description of the robot in side view	5
4.4. List of functions provided for the control of the robot	28	2.6. Example of the robot movement around the z-axis	5
		2.7. Example of the radial movement	6
		2.8. Example of the tilting of the arm	7
		2.9. Example of the z-indexing.....	8
		2.10. Example of the stave insertion ..	8
		3.1. Schematic diagram of RS-485 Network.....	11
		3.2. Circuit diagram of RS-485 circuit	12
		3.3. Schematic diagram of controller with TMC2130 driver ..	13
		3.4. Schematic diagram of the controller with an external stepper motor driver.	14
		3.5. Structure of the command packet.....	15
		3.6. Structure of the response packet.....	16
		3.7. Flowchart of the main loop of the controller	18
		3.8. Flowchart of the central control unit software.....	20
		4.5. Photos of the first stave insertion sequence	29
		4.6. Photos of the second stave insertion sequence	30

Chapter 1

Introduction

The use of robotics in manufacturing processes is becoming increasingly necessary. Many manufacturing processes require high precision and reliability, which is almost impossible without the use of robotics. The robots also allow handling heavier objects than would normally be possible with such precision. There are many types of robots based on its intended purpose. Many of the manufacturing processes contain similar tasks, so there are standard robotics solutions to choose from. The robots can also be specifically designed for uncommon tasks, such as mounting components in tight or otherwise hardly accessible places.

ATLAS is one of the four major detectors at the Large Hadron Collider (LHC) at CERN. The LHC collides two beams of protons together at very high energy, which can produce variety of different particles, from which some can be previously unknown. The particle collisions occur in the center of the ATLAS detector. The goal of the ATLAS detector is to measure very broad range of signals coming from the collisions to ensure that any new particle, regardless of its form or nature, will be detected. The ATLAS detector consists of multiple layers, where each layer has its own purpose.[1]

The LHC is currently being upgraded to provide even higher energies of the collisions. This requires an upgrade of the ATLAS Inner Tracker, which is the most inner part of the whole detector. This upgrade requires installation of new sensors, which will be done via a robotic assembly designed by the team from the Department of Designing and Machine Components of the Faculty of Mechanical Engineering at CTU. The team members are Jan Brajer, Daniel Hadraba, Martin Janda, Jakub Janoušek and František Lopot.

The goal of this thesis is to design and implement HW and SW to control this robot.

Chapter 2

Problem Statement

The ATLAS experiment at the Large Hadron Collider in CERN is preparing for a major upgrade of the accelerator for the high luminosity phase (HL-LHC). The increased luminosity results in higher radiation damage to the detectors and is well beyond that for which was ATLAS designed and it requires replacement of the Inner Detector with an all-silicon Inner Tracker (ITk). The goal of ITk is to track particles created during the particle collisions inside of ATLAS detector.

ITk is composed of two layers, inner Pixel Detector and Strip Detector, which is built around the Pixel Detector. Cross section of the outer cylinder that accommodates the strip barrel is shown in Fig. 2.1. The Strip Detector is then composed of four barrel layers and divided to two symmetrical sections around the collision point where $z = 0$, each 1400 mm long and both containing total of 196 staves on all four barrel layers.

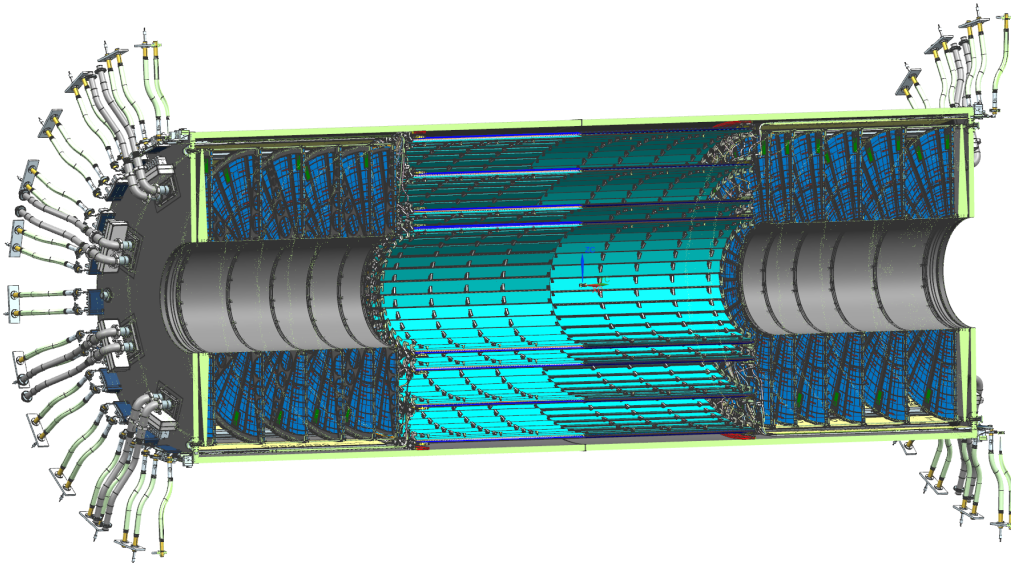


Figure 2.1. Cross section of the outer cylinder that accommodates the strip barrel (light blue) and end-cap structures.[2]

The stave is one of the basic mechanical building blocks of the barrel and consists of a core to provide mechanical rigidity and support for the electrical modules and houses common services, such as electrical or cooling. Each stave is populated with 28 silicon-strip modules, which are responsible for tracking of the particles. There is an End of Structure Card, which provides data connection and power of all the modules on the stave to the rest of the detector electronics. Model of the stave is on the Fig. 2.2 The staves are mounted on each barrel all around the z -axis, which leads through the center of the detector and is coincident with the beam path.[2]

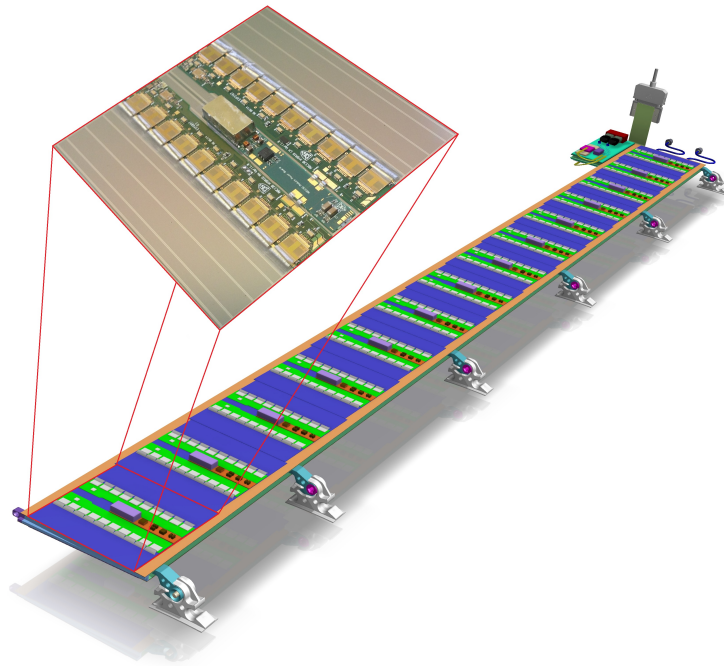


Figure 2.2. Model of the stave.[3]

The size and technical complexity of the staves requires that they cannot be installed by hand, instead an automated mechanism has to be developed in order to install them correctly. Design of this mechanism, Stave Insertion Tooling, is done by the CTU and goal of this thesis is to design the HW and SW control of the robot. Mechanical model of the robot is shown in Fig. 2.3.

The robot will be used only for a single task, which is the installation of the staves. The usage will be time-limited, and the robot will not be reused for anything else, so the design does not focus on long term operation and reliability. Instead, the design focus is the cost effectiveness and customizability. The robot will be working only in laboratory conditions and it will not encounter any radiation.

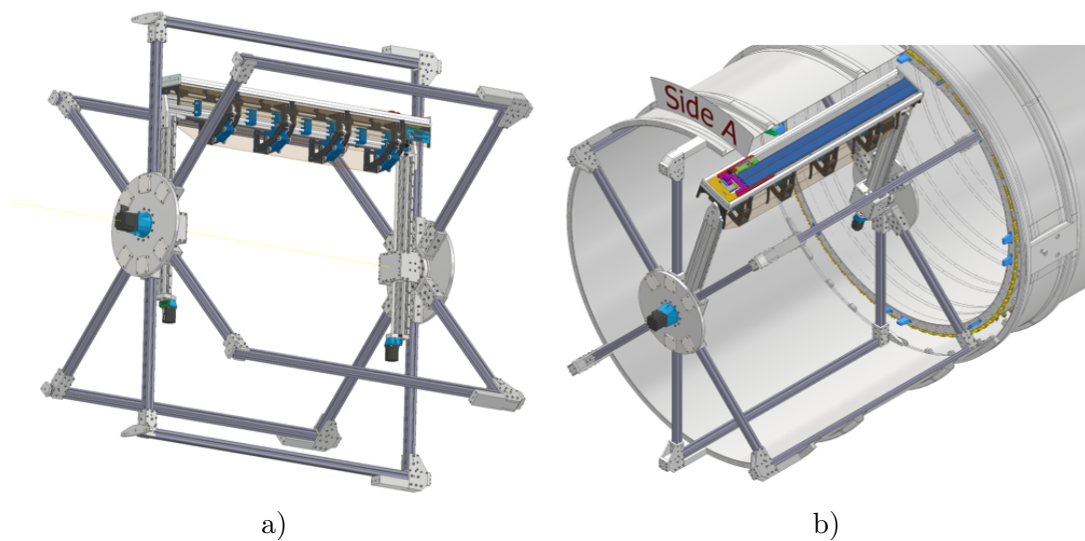


Figure 2.3. Model of the robot, on the left is the whole mechanical assembly, on the right the robot is installed inside the barrel and ready for stave insertion.

The model of the main arm of the robot is on Fig. 2.4. It is part of the model shown on Fig. 2.3. The direction of the stave insertion is represented by the green arrow. The central control unit for the control of the whole robot is mounted on the arm in the front section. The main arm has three degrees of freedom (DoF), which will be later described in the section 2.1.

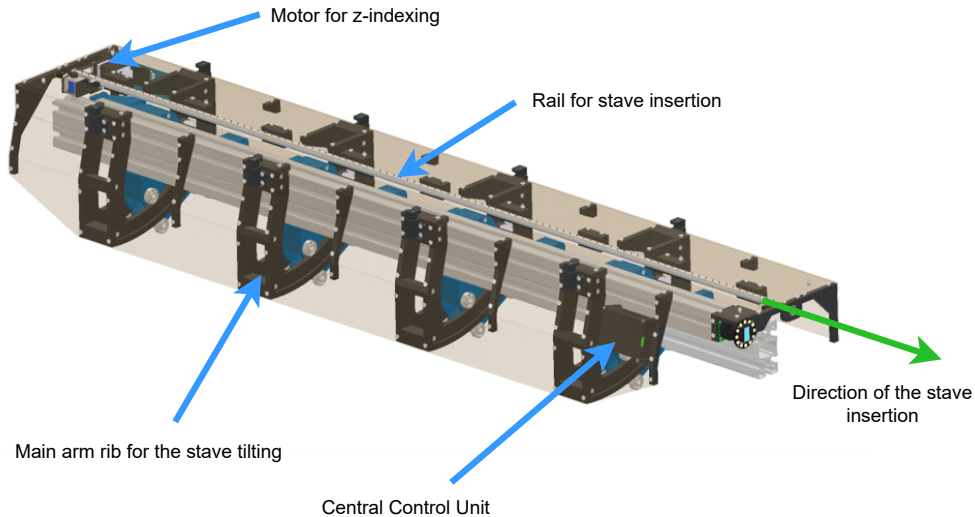


Figure 2.4. Model of the main arm of the robot

2.1 Description of the degrees of freedom

The schematic description of the robot in side view is shown in Fig. 2.5. Each joint on the figure is numbered, where the first digit corresponds to the degree of freedom and the number after dot is unique to the joint within the axis. There is also an arrow next to each of the joints which indicate its movement.

The robot has to be able to install all of the 196 staves in both sections, so it has to be able to rotate around the whole z-axis and move to all of the four barrels. All of the stave positions on each barrel are tilted and the amount is different for each of the strip barrel layer. Then the stave has to be inserted into its final position. First, the box with the stave inside is extended outwards to the beginning of the strip barrel and this movement is called z-indexing. Then, the stave is inserted into the final position inside on the strip barrel.

The whole robot is installed inside a large composite barrel and can be connected to it only on certain points inside of the barrel. This is the reason for the legs shown on the Fig. 2.3. There are at most 6 legs present, from which some of them must be removed as some of the staves are placed behind them, and also for the accessibility of the stave and the box.

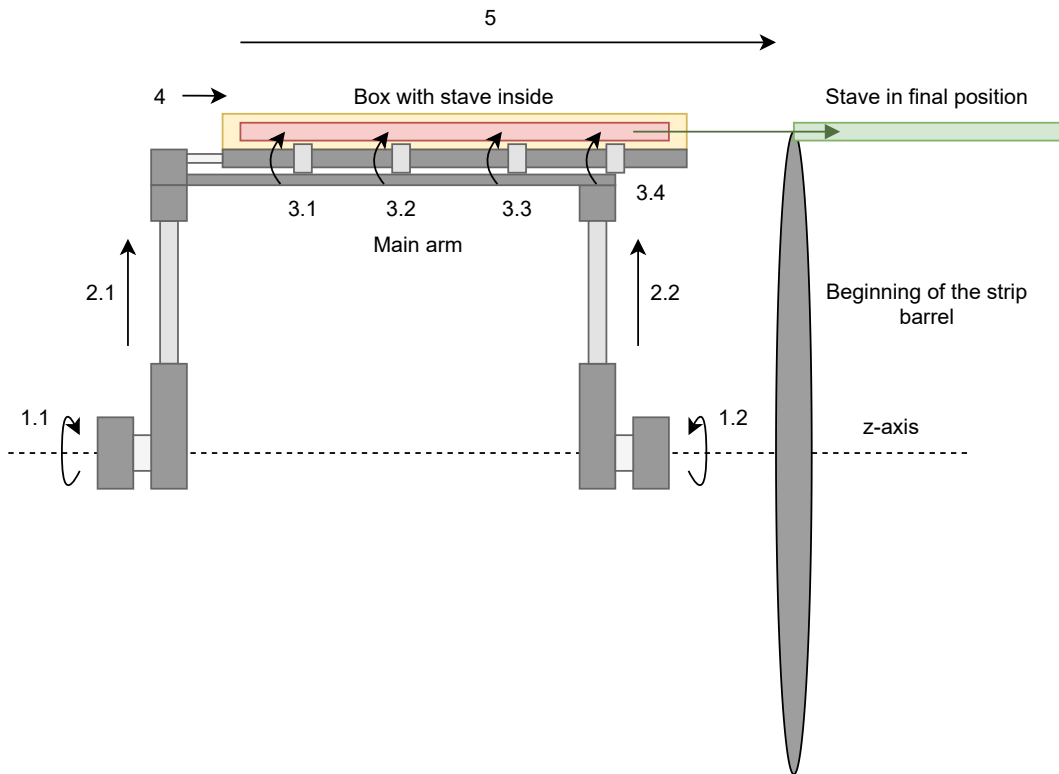


Figure 2.5. Side view diagram of the Stave Insertion Tooling mechanical layout. Robot links are in dark grey, robot joints are in light grey and their movement is illustrated by their respective arrows. Each motor has its own number corresponding to its axis.

2.1.1 Rotation around the z-axis

The rotation around the z-axis is done by two parallel joints 1.1 and 1.2. Both joints are realized as stepper motors connected to its own harmonic gearbox with a ratio of 1:100. The example of the robot movement of this DoF is on Fig. 2.6.

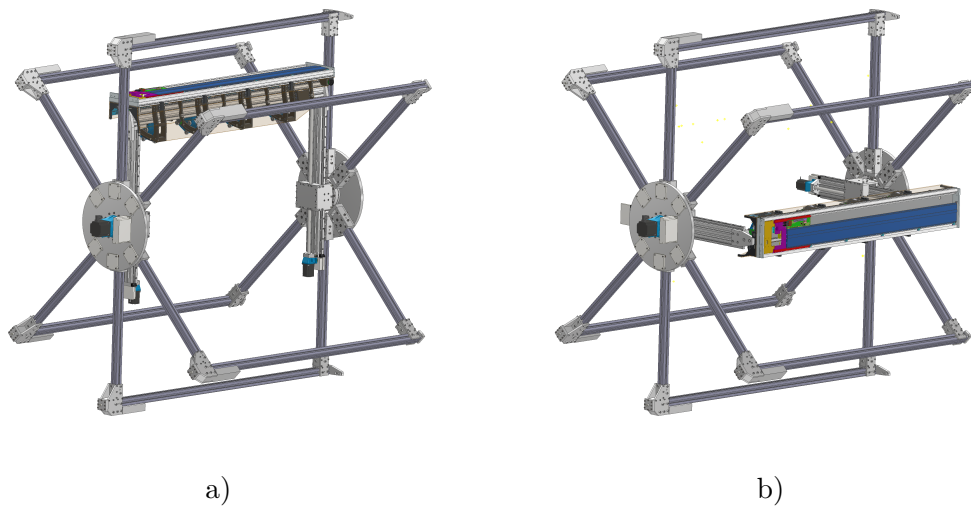


Figure 2.6. Example of the robot movement around the z-axis. On the figure a) is the robot in its default position. On the figure b) is the robot rotated -90 degrees to the home position.

A electromagnetic brake is connected to each motor and is engaged in its default (unpowered) state. This is to prevent any movement of the robot in case of a sudden unexpected loss of power or an emergency stop. This DoF is the only one with the danger of unwanted movement while powered off, due to the arm not being balanced around the axis of rotation and using a gearbox, not a lead screw. Any unwanted movement in this axis would be fatal if it would happen during the stave insertion, as the stave would be only partially inserted and the movement could seriously damage the stave. Also, there is potential risk of damaging the robot itself and anyone near it.

The reason for multiple parallel joints is increased rigidity of the system. Only one motor present to move in the rotational DoF would mean higher possible deformation of the robot, which is unacceptable because the stave has to be installed precisely. The force would be also much higher in case of one motor only, as the mechanical force would not be shared between all of the motors and a sturdier and heavier construction would be required.

2.1.2 Radial movement

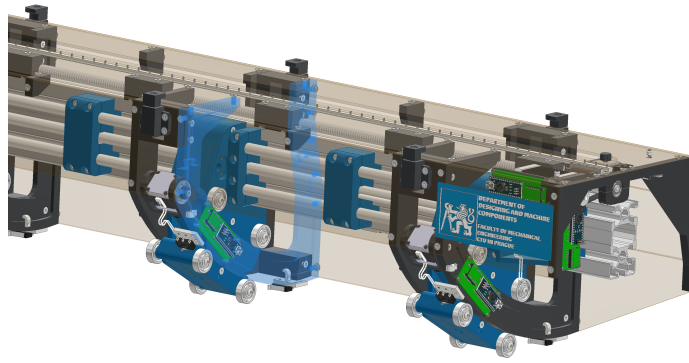
The radial movement needed to reach all four of the barrels and has two parallel joints 2.1 and 2.2. Both joints are translational and the linear motion is done with a lead screw with 4 mm per revolution. There are limit switches on both ends of the motion range to provide a reference point for the software. The example of the robot movement of this DoF is on Fig. 2.7.



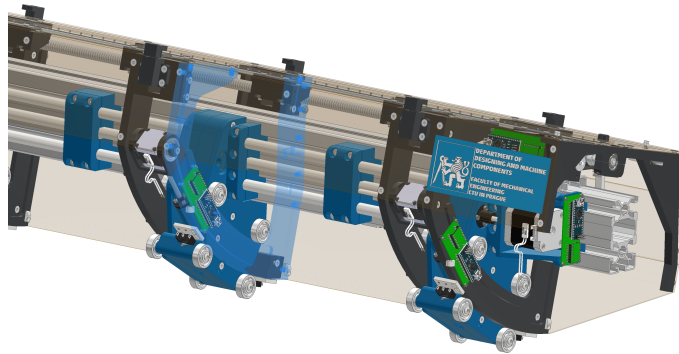
Figure 2.7. Example of the robot movement in the radial direction. On the figure a) is the robot in its default position. On the figure b) is the robot arm fully extended.

2.1.3 Tilting of the staves

The tilt of the stave has to match final position of the stave and the amount is different for each of the strip barrel layer, which is at maximum 14° . The stave is tilted together with its transport box, which is clamped to the top of the arm. There are four parallel joints synchronized together denoted in the schema as 3.1 to 3.4. The top part of the arm with the box and stave attached is connected to the main part by four sliders which are part of the joints. Then these sliders have each connected a motor with a lead screw with 0.635 mm per revolution. Both home and end switches are present at the limits of the movement. The example of the robot movement of this DoF is on Fig. 2.8.



a)

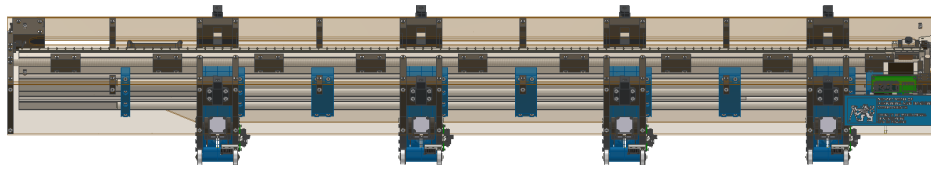


b)

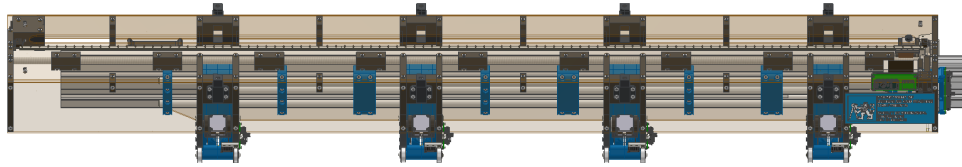
Figure 2.8. Example of the tilting of the arm. On the figure a) is the robot in its default position. On the figure b) is the robot arm fully tilted.

■ 2.1.4 Z-indexing

The whole robot with the box has to move around in the entire space of the barrel and there are supports of the robot between the box and the final stave position, so there is space between the box and beginning of the strip barrel. The robot must move the box with the stave right next to the beginning of the strip barrel, as it could not extend the stave beyond the box and this movement is called z-indexing. The movement is done by the joint 4 and it is a motor with a lead screw with 2 mm per revolution. This movement can be only done after the rotation, radial movement and tilting is done. The example of the robot movement of this DoF is on Fig. 2.9.



a)

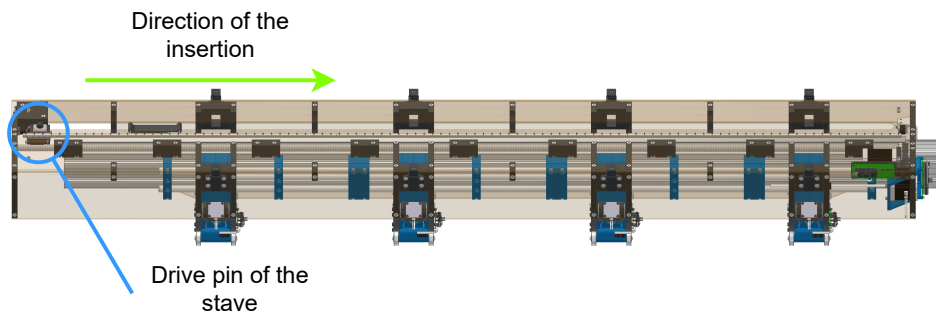


b)

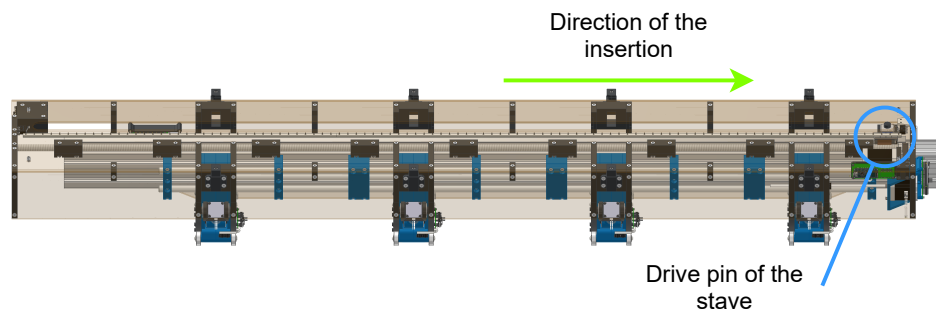
Figure 2.9. Example of the z-indexing movement. On the figure a) is the robot in its default position. On the figure b) is then the z-indexing fully extended.

2.1.5 Stave insertion

After the box with the stave inside is extended to in contact with the strip barrel, the stave has to be inserted into the guide rails on the strip barrel. There is retractable pin on the top of the arm of the robot, which locks into a slot on the end of the stave through a cut out on the bottom side of the box. The pin is then linearly driven by a lead screw with 0.635 mm per revolution connected to a motor. This enables the stave to be inserted to its final position in the guide rails on the strip barrel. The example of the robot movement of this DoF is on Fig. 2.10.



a)



b)

Figure 2.10. Example of the z-indexing movement. On the figure a) is the robot in its default position and fully tilted. On the figure b) is then the z-indexing fully extended.

2.2 Motors of the robot

Stepper motors were chosen for all of the degrees of freedom (DoF). They are more cost effective than servo motors and there is lower complexity of their control as they don't require closed loop control system for their operation. There are in total 10 stepper motors total driving all of the five DoF. Only the two motors of the joints 1.1 and 1.2 responsible for the rotation around the z-axis are used to generate rotational motion. Rest of the motors in the remaining four DoF have integrated lead screw to form a linear motion.

Parameters of all motors used can be seen in the table 2.1.

Table 2.1. Details of the motors used for the robot drives.

DoF	Type	Step angle (°)	Holding torque (Nm)
1. Rotation	34HE31-6004S	1.8	4.8
2. Radial movement	23HP30-2804S	1.8	1.85
3. Tilting	11LS13-0754E	1.8	0.08
4. Z-indexing	17LS19-1684E	1.8	0.44
5. Stave insertion	11HS20-0674S	1.8	0.12

Chapter 3

Proposed Solution

Hardware and software solution is proposed in this chapter for the control of the robot, which is described in chapter 2. The control system was designed as a distributed system with one central control unit and each motor is driven by a separate controller. The section 3.1 describes the used communication network, central control unit and the controllers. Then the section 3.2 describes the control software of the central control unit, the firmware of the controllers and the communication protocol used.

3.1 Hardware

A lightweight and small computer Raspberry Pi 4 was chosen as the central control unit, as it has the capability of running Linux operating system and many other useful features. The controllers are based on Arduino Nano Every with the microcontroller ATmega4809, as it enables rapid prototyping of the overall design and is easily programmable. Each controller has a stepper motor driver connected to it and the used type of the driver depends on the used motor. The central control unit and the controllers are together connected by RS-485, which is an industrial standard and provides reliable connection even in an electromagnetically noisy environment and over large distances.

3.1.1 Communication network

The robot needs to be able to operate in industrial environment with higher electromagnetic interference. There is also the possibility of ground potential differences due to the operation of the stepper motors. The communication standard also needs to support multiple connected devices.

There are multiple possible network types. The first possible options would be the I²C and SPI interfaces, because they are present both in the Raspberry Pi and the ATmega4809. The usage of either of these networks would mean no additional hardware required, which would reduce the cost and complexity of the system. The SPI is already used for other communication with the stepper motor drivers and thus it cannot be used for this purpose. The I²C is capable of connecting multiple devices and there is a possibility for multi-master communication, but it is only intended for short range communication within the circuit board. The total length of the network is around 10 m and with possible higher electromagnetic interference, so the I²C is not usable.

The communication standard TIA/EIA-485 (also known as RS-485) was selected as it is designed for operation in such conditions with high reliability. It is a simple and cost-efficient solution compared to other possible industrial standards such as CAN bus. The RS-485 is a serial communication standard supporting multidrop communication networks using differential signaling over twisted pair cable.

The recommended topology is the bus topology, as other types of topologies can introduce unwanted signal reflections. The ends of the cable should have a termination resistor connected across the two wires to eliminate signal reflections. The value of each

resistor should be equal to the characteristic impedance of the cable. The terminations of the cable should also include pull-up and pull-down resistors to drive the data line to a known state in case no device is currently transmitting. [4]

The schematic diagram of the RS-485 network on the robot is shown on Fig. 3.1. The bus topology is used and the termination points are in the rotational joints. Then the bus continues to the radial movement and then finally to the main arm of the robot, where the z-indexing, tilting and stave insertion is connected. The used twisted pair cable is the UTP CAT.5e.

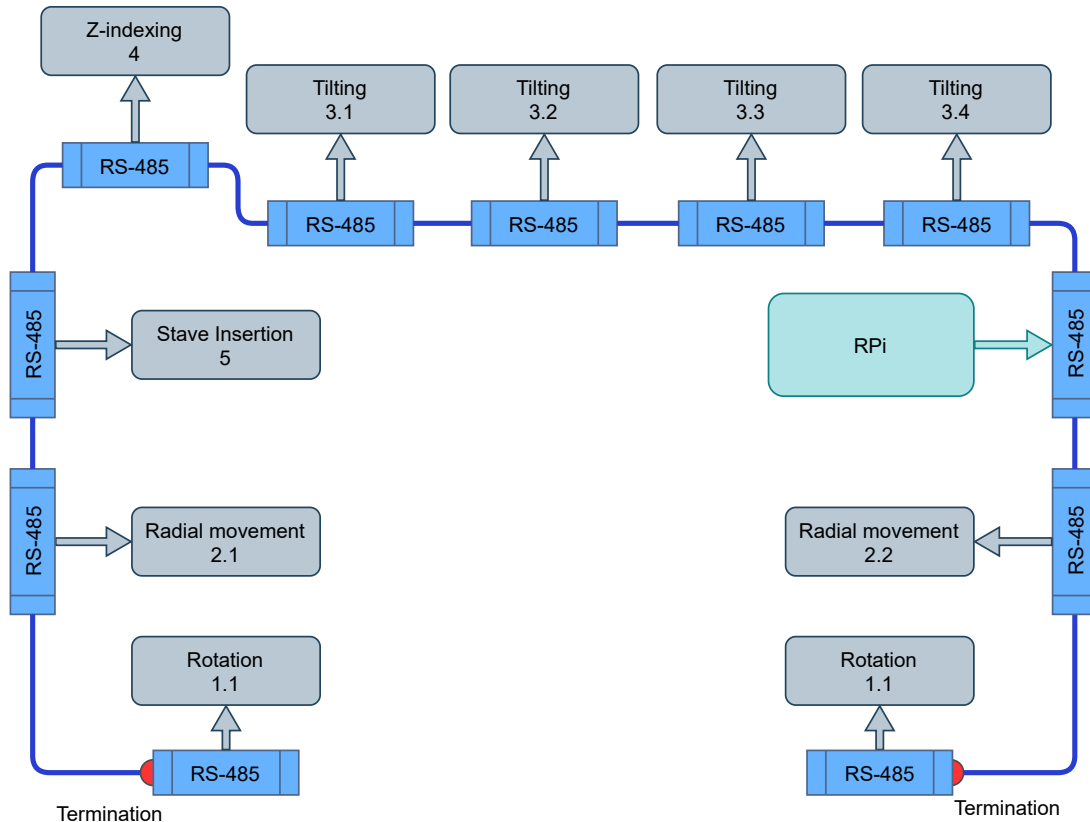


Figure 3.1. Schematic diagram of RS-485 Network

3.1.2 Central control unit

The central control unit is a Raspberry Pi 4 computer, which is equipped with Broadcom BCM2711 1.5 GHz ARM Cortex-A72 quad-core processor accompanied with 4 GB RAM and 128 GB SD card storage. The operating system is Raspberry Pi OS, which is based on Debian. Other relevant features are four USB ports, Wi-Fi, MIPI CSI connector for camera connection and 40-pin GPIO with I²C capability. It is connected to the RS-485 network via a USB to RS-485 converter, which provide the necessary logic level translation and timing. [5]

The computer is mounted on the robot arm, which is the reason for it's lightweight and small size requirement. There is also a camera directly connected to the Raspberry Pi mounted on the arm intended for the fine-tuning of the robot arm position to ensure precise insertion of the stave and this feature will be added in the future improvements of the robot.

3.1.3 Stepper motor controller

Stepper motor controller is based on the Arduino Nano Every, which is built around a ATmega4809[6] chip. It is responsible for the communication with the central control unit, controlling the position and velocity of the motor and monitoring the limiting switches on the joints.

The connection of the controller to the RS-485 network is done by MAX485 chip. The MAX485 serves as transceiver for the RS-485 communication and translates data from the RS-485 to UART. It is a half-duplex device, meaning that the the controller must switch the mode of the transceiver to either receiving or transmitting. [7]

Schematic diagram of the circuit for the RS-485 communication is on Fig. 3.2. The termination resistor together with the pull-up and pull-down resistors are present. This is only the case for the two controllers on the ends of the cable. On the rest of the controllers these resistors are not present.

The left side of the chip is the UART side, with Rx and Tx signals for transmitting and receiving the data. There is also the signal XDIR connected to the pins RE and DE of the MAX485 chip, which controls the direction of the communication. The MAX485 is transmitting when the XDIR signal is high and the DE (Drive Output Enable) pin is high. Otherwise, the MAX485 is receiving when the XDIR signal is low, as the RE (Receive Output Enable) pin is low.

There are pull-up resistors on both the Rx and Tx pins to ensure valid logic levels coming in the ATmega4809 and the MAX485. There is also a pull-down resistor on the XDIR signal to prevent any collision in case the XDIR pin is unconnected. In case of a collision high currents could flow across the cables, which could potentially damage the MAX485 chips.

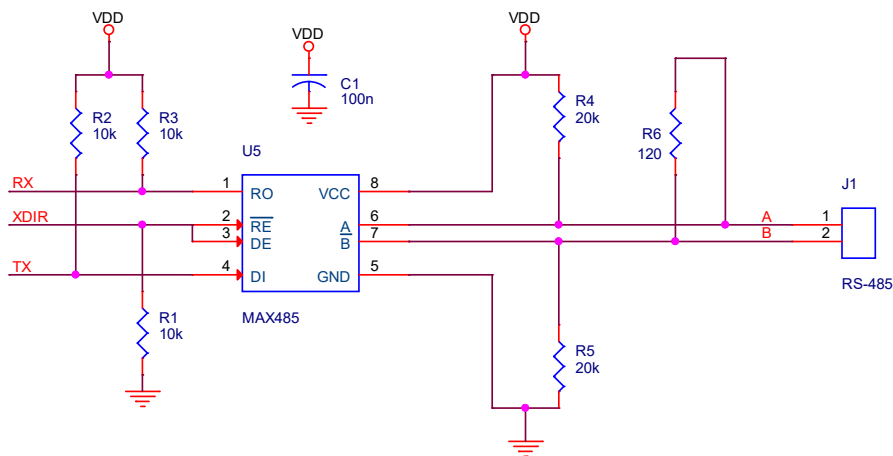


Figure 3.2. Circuit diagram of the MAX485. On the left side of the MAX485 there are signals going to the ATmega4809, on the right side is the power and the connection to the RS-485 network.

There are two variants of the controller depending on the size of the motors. The smaller stepper motors with amps per phase up to 1.2 A are driven by the TMC2130 stepper motor drivers. The TMC2130 is highly customizable and allows its parameters

to be set via a SPI interface. The driver allows for up to 256 microsteps per motor step and its function StealthChop allows for smooth motion without vibrations.

Stepper motor drivers chosen for the robot are controlled by a STEP/DIR interface. The logic state of the DIR signal is telling the controller in which to turn the stepper motor. Then the change of logic level on the STEP signal tells the driver to move the stepper motor one step forward in the direction specified by the DIR signal. If the driver has enabled microstepping, then the step command only moves the stepper motor fraction of the motor step, depending on the microstepping setting.

The driver can be sensitive only to some changes of the logic level on the STEP signal, usually it is the rising edge. This is the case for the controller variant with the external stepper driver. The TMC2130 can detect both the rising and falling edge, which is beneficial for higher possible speed of the stepper motor with the same base clock frequency.

The diagram of the controller with the TMC2130 is shown on Fig. 3.3. The whole controller is powered via a 12 V power line. The 12 V power line is used by the TMC2130 to power the stepper motor and it is also converted to 5V to power the ATmega4809 and the MAX485 via an internal DC converter of the Arduino Nano Every. The TMC2130 also needs the 5V power for its internal logic, so it is connected to the 5V output from the Arduino Nano Every.

There are two communication interfaces used on the ATmega4809, which is the UART and the SPI. The UART is used for communication between the ATmega4809 and the MAX485. The SPI interface is then used for the communication with the TMC2130 for its setup and to read the status of the driver. There are also two signals for the STEP/DIR instructions to the stepper motor driver.

The controller has two connections for limit switches. The limit switches are mechanical microswitches with a simple RC filter for switch debouncing. A bimetal thermostat is also connected to prevent the motors from overheating.

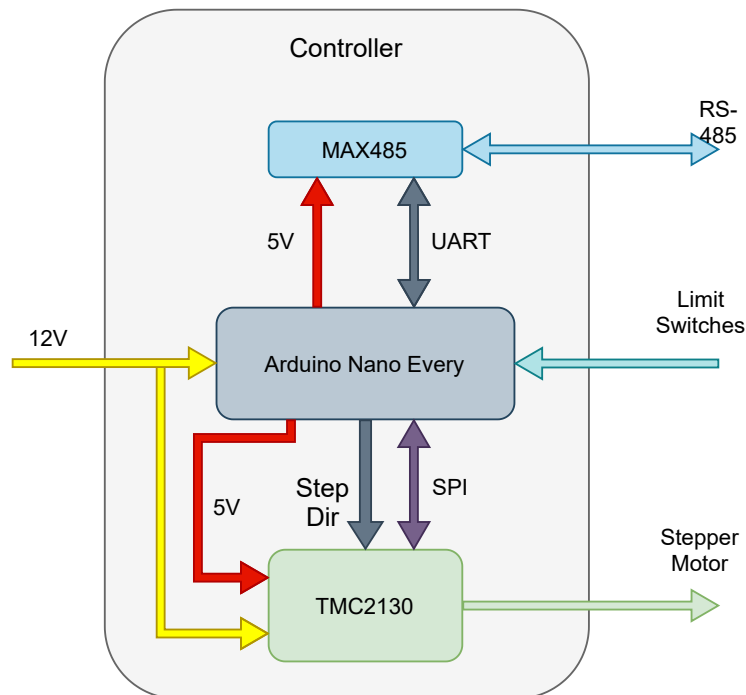


Figure 3.3. Schematic diagram of the controller with TMC2130 stepper motor driver.

The diagram of the controller with an external stepper motor driver is shown on the Fig. 3.4. The basic structure is the same as the controller with the TMC2130. The only difference is that the external stepper motor drivers cannot be controlled via the SPI interface, only interface with them are the STEP/DIR signals.

This solution can be used together with more powerful stepper motor drivers required for the motors responsible for the rotational and radial movements of the robot. The driver used for the rotation around the z-axis is the M880A, which allows up to 5.6 A RMS of current. The driver used for the radial movement is the M542, which allows up to 3 A RMS of current. Both of these drivers support microstepping up to 128 microsteps per motor step. The microstepping value chosen for both of the motors is 16, where this microstepping resolution significantly decreased vibrations of the motor.

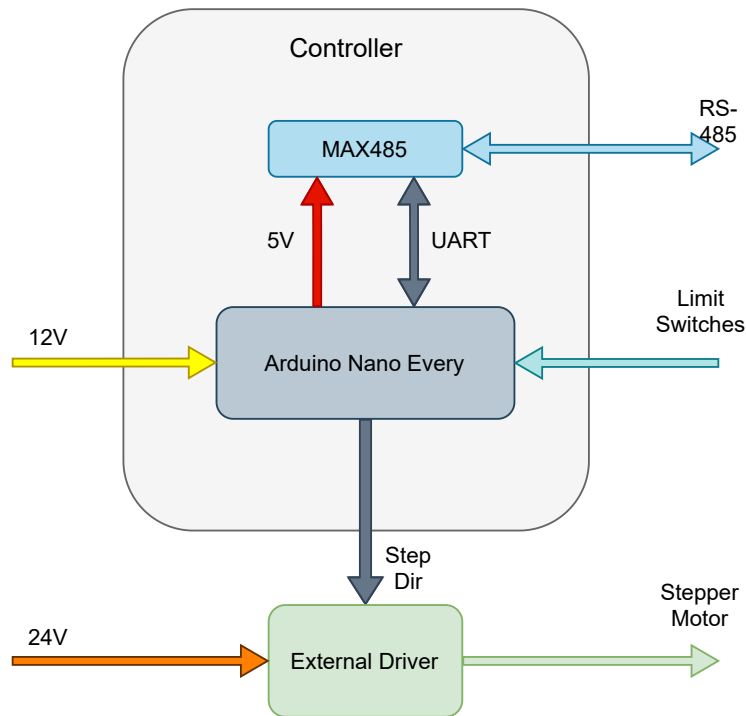


Figure 3.4. Schematic diagram of the controller with an external stepper motor driver.

3.2 Software

The robot control is based on two main parts, the central control unit and the controllers. Each controller is responsible only for the motion of a single motor, but three of the DoF have two or even four parallel motors driving them. The controllers don't know the overall status of the system and the status of other controllers, so there is need for a central authority to keep everything synchronized. The system is based on the master-slave communication control to prevent any collisions on the bus. The master is the central control unit and the slaves are the individual controllers.

3.2.1 Communication protocol

The communication protocol is based on data packets that are sent over the physical layer, the RS-485. Every packet has a fixed length of 7 bytes. Every device connected to the communication network see every transmitted packet. The device needs to know what packet is relevant for it, so every device in the network is given unique address in form of one byte.

The device needs to know if the message it receives is correctly delivered and that no error during transmission occurred. The error detection is done via a checksum, which is at the end of every packet. The checksum c is calculated as

$$c = 255 - \left(\left(\sum b \right) \bmod 256 \right), \quad (1)$$

where b are the bytes of the message without the checksum. If there is an error during the transmission, such as wrong bit, the checksum will be different and the message is not used.

Communication between the master and the slaves is always initiated from the master. The master sends a command packet to the slave. The structure of the command packet is on Fig. 3.5. The byte 0 is the address of the slave. The byte 1 is the command type, which tells the slave how to interpret the data and if there is a response expected. The bytes 2-5 are for data transmitted to the slave and the data are stored in a big-endian manner. Then the last byte 6 is the checksum.

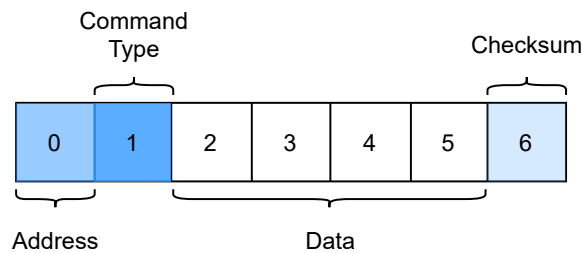


Figure 3.5. Structure of the command packet

The master can request information back from the slave. In this case the message structure of the packet is the same, and the structure of the response packet is on Fig. 3.6. The packet has similar structure to the command packet, the difference is in the byte 1. There is no need for the message type as it is an immediate response from the slave back to the master, so the byte can be used for a different purpose. The byte 1 in the response contains the status of the controller, where the bits represent logic state of the most important information.

The bit 0 should always be true and it represents the validity of the status. Then the bit 1 carries the information if the stepper motor is running. The bits 2 to 4 are for the state of the switches, where logic 1 represents the switch being active, such as the axis is in home position or the stepper motor is overheating. Last bits 5 to 7 are currently not used and are reserved for future use.

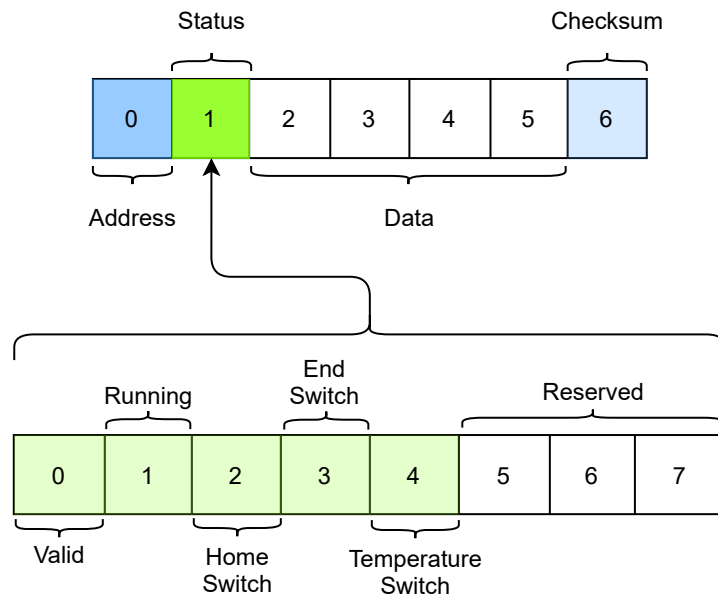


Figure 3.6. Structure of the response packet

Types of command messages are shown in table 3.1. The first type of message is the `CMD_STOP` to stop the movement if any issue would occur. The data in the case of this message type are not used and it is filled with zeros.

`CMD_MOVE` is command telling the stepper motor to go certain number of steps in a given direction. The number of steps for the stepper motor to go is stored in the data section of the packet in form of a signed 32bit integer. The sign of the integer is used for the direction of the motor rotation.

The commands `CMD_SET_SPEED`, `CMD_SET_ACCEL` and the `CMD_SET_DECEL` are used to set the parameters of the movement of the stepper motor, the maximal speed, acceleration and deceleration of the motor respectively. The values are stored in the data bytes of the packet as a unsigned 32bit integer.

The command `CMD_SET_CURR` is used to set the current going to the stepper motor from the driver, and it is only valid for the controllers equipped with the TMC2130 stepper motor driver. The current setting is the number of milliamps and it is stored as a 32bit unsigned integer.

`CMD_SET_BRAKE` is there to enable or disable the brake on the rotational DoF and it is valid only for the two controllers driving this DoF. The value is 1 for the brake to be enabled and 0 for the brake to be disabled. It is again stored as a unsigned 32bit integer to define all for bytes of the data part of the packet.

Last message type is the `CMD_GET_CURR_POS`, which is used to get the current position of the stepper motor as well as the status of the controller. The message does not send any data to to the controller and the four data bytes are filled with zeros.

The only currently returned message from the controller is the response to the `CMD_GET_CURR_POS`. The four data bytes contain current position of the motor as a signed 32bit integer.

Table 3.1. Types of the command messages used.

Name	Number	Description
CMD_STOP	0	Total stop signal for the controller
CMD_MOVE	1	Move n steps
CMD_SET_SPEED	2	Set maximal speed
CMD_SET_ACCEL	3	Set acceleration
CMD_SET_DECEL	4	Set deceleration
CMD_SET_CURR	5	Set current of the motor
CMD_SET_BRAKE	6	Set the state of the brake on the motor
CMD_GET_CURR_POS	7	Request current position of the motor

3.2.2 Stepper motor controller

The task of the stepper motor controller is to drive the stepper motor based on the instructions from the central control unit. It has to process the incoming packets, control the stepper motor and react to external switches. The software for the ATmega4809 is written in the C++ language.

The basic structure of the program is illustrated on the Fig. 3.7. The controller at the beginning initializes all the necessary values and then goes to the main loop, where it stays until power-down of the whole system. The main loop first checks if any of the switches (home switch, end switch and thermostat) is not active, which would mean stopping the stepper motor. Then the program checks for new messages, and if there is any, it tries to process it.

The processing of the message starts with checking if the message is valid and intended for the particular controller. If it is correct, the loop then does actions based on the type of the message. If the type of the message is a stop command, the motor is immediately stopped and the status of the controller is updated. In case the message is to set any parameter of the stepper motor, such as speed, acceleration or current, the current parameters are updated, but they will be used only for any future move commands. The command to set the brake will engage or disengage the brake on the motors, but only if the controller is defined as driving the rotational DoF, as they are the only ones with brakes. The request of current position will prepare and send the response packet as it is described in section 3.2.1. The move command will run the function to start the control of the stepper motor.

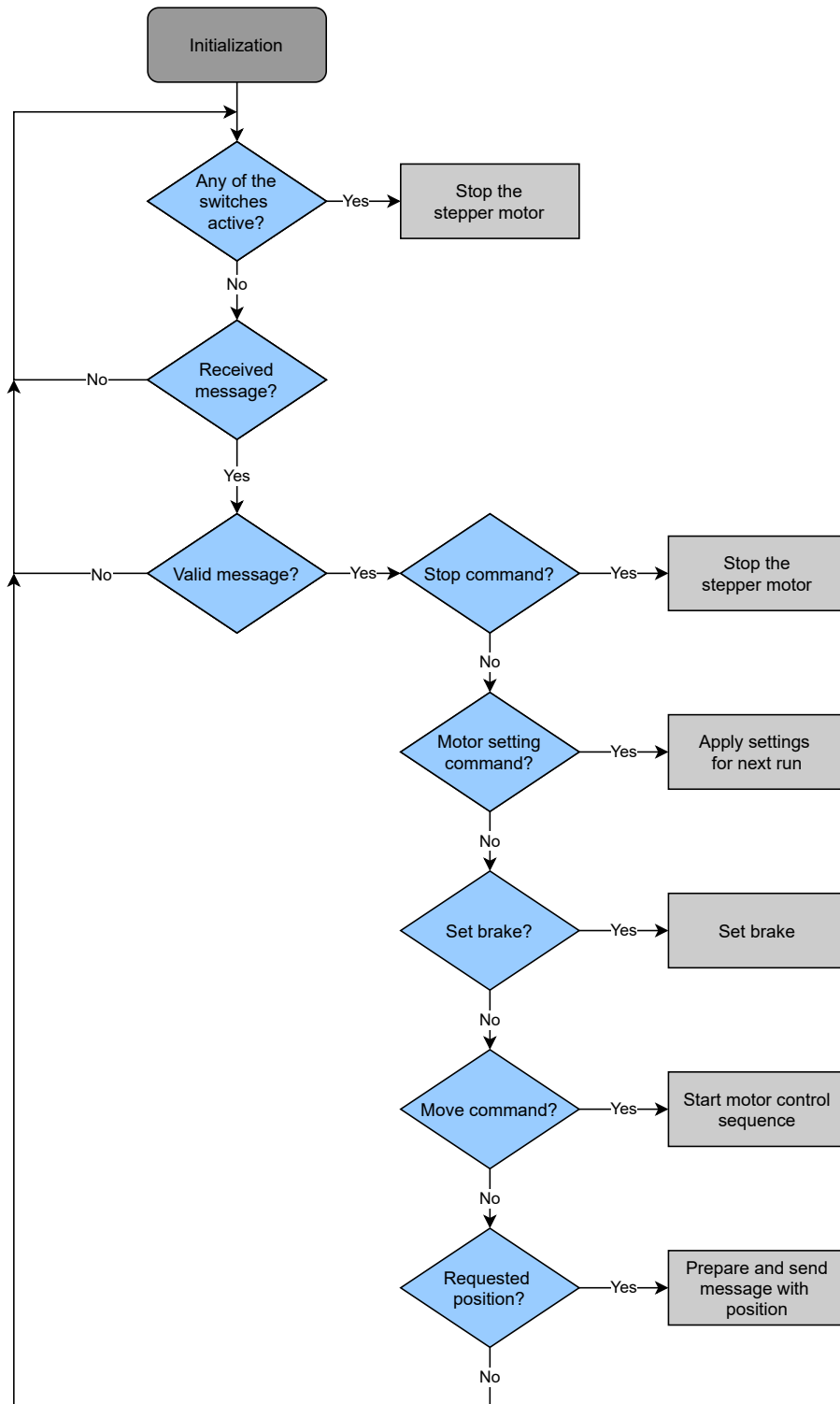


Figure 3.7. Flowchart of the main loop of the controller.

The controller must generate precisely timed pulses on the STEP signal to drive the motor with requested speed and also it is important to eliminate vibrations of the motor. The control of the stepper motor has to be running independently of the main loop, so a timer of the ATmega4809 was used together with its interrupts. The function at the beginning initializes the timer and calculates necessary parameters of movement of the stepper motor motion, such as how much steps it will take for the stepper motor

to accelerate, how much steps it will be running at the maximal speed and when to start decelerating. The timer is then used to precisely time the impulses of the motor and each impulse the parameters of the timer are updated so the overall motion will match the desired acceleration and deceleration curve. The calculation of parameters for the timer are adapted from the Linear speed control of stepper motor Application Note[8].

■ 3.2.3 Central control unit

The task of the central control unit is to control the whole robot based on the instructions from the user. The control software is written as a python script, where the user interface is designed as a console application. The software is controlled via the input arguments, which are provided from the user at the its start. The software executes the command and then exits and next command is done with the next software run. The software is divided into two main parts, where each part is in single python script.

■ 3.2.3.1 Control of a single stepper motor

The first part *stepper_controller.py* is responsible for the control of a single stepper motor controller. It contains a class *Stepper* which provides all the functions to set the stepper motor controller parameters such as acceleration, speed of current, it can command it to move and request its position and status. The class internally handles the creation, transmission and reception of the messages needed to control the stepper motor.

The structure of the messages is described in the section 3.2.1. the transmission and reception of the messages is repeated to minimize the chance of unsuccessful delivery. The transmission and reception is tried five times before reporting an error. The attempt to receive the message can be unsuccessful due to a timeout or an internal error of the bus. The repeated timeout would mean that the device is not responding and it is important to notify the overlying software.

■ 3.2.3.2 Control of the whole robot

The second part of the software *robot_control.py* is responsible for the synchronized operation of the stepper motors, formed into a single DoF. The script can be run from the command line in order to control the robot. It is also prepared as a module for future use by another software.

The main part of the script is the class *Robot* with a subclass *Axis*. The *Axis* subclass is based on the *Stepper* class, where each instance of the class is connected to a single stepper motor controller. These instances are grouped together in single instance of the *Axis* class, which then provides functions to manipulate with all the controllers at the same time.

The class *Robot* is then used to represent the whole robot. It contains multiple instances of the *Axis* subclass, where each instance corresponds to a single DoF. It also has functions to move each axis by a given distance in degrees or millimeters and to toggle the pin during the stave insertion.

The flowchart of the control process of the robot is on Fig. 3.8. the basic flow of the software is on the left side. First, the user parameters are processed. The user selects an DoF which he wants to control, specifies its settings and can also request its movement. Then the program checks if a correct USB to RS-485 converter is connected and establishes a connection with it. The program then sends the settings to each controller belonging to the selected DoF and then the movement is executed if requested.

The movement control function is the most critical function of the control software. The flowchart of the movement control function is on the right of the Fig. 3.8. The function needs to ensure synchronization of all of the stepper motors belonging to the same DoF. This is done by continuous checking of the state of all the stepper motors.

At first, the function checks the state of all the controllers to ensure they are ready and connected to the network. If the check is successful, the movement request is sent to all of the controllers. When the stepper motors are running, the function continuously checks the status and position of each of the controllers. The status is evaluated and if there is error the movement of all of the stepper motors is stopped. The checking stops and the function finishes successfully only after all of the motors reach the desired position.

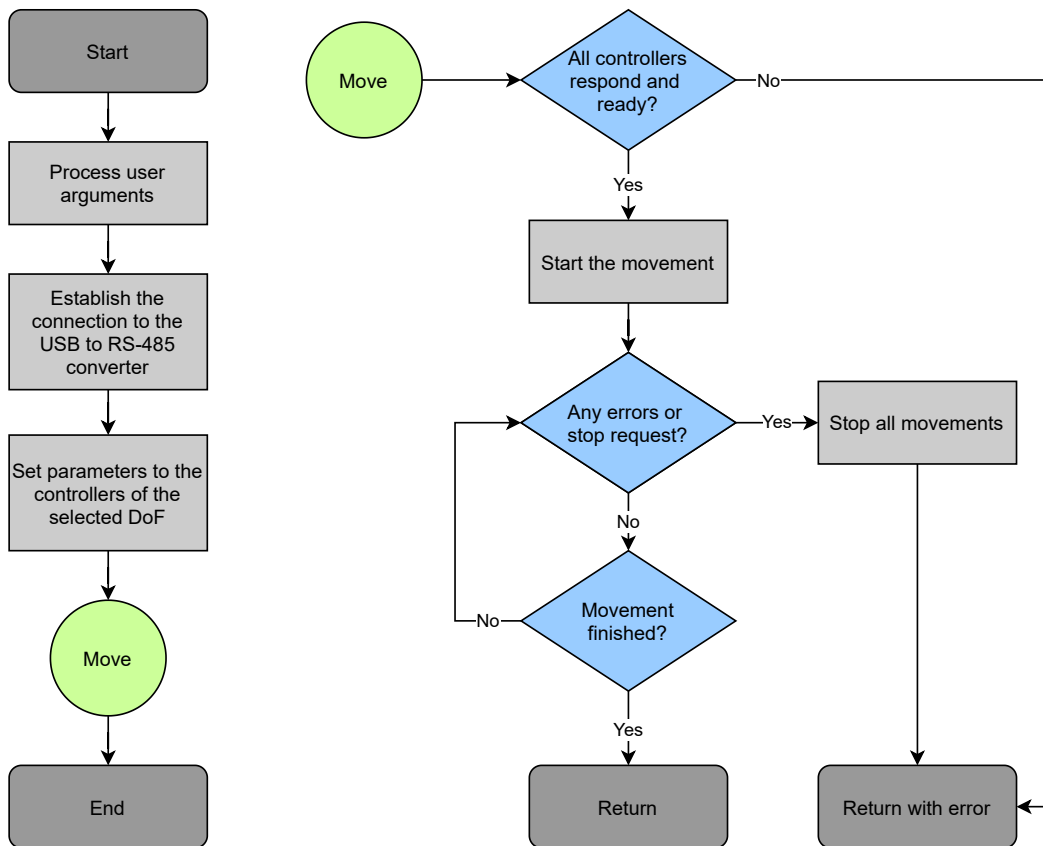


Figure 3.8. Flowchart of the central control unit software. On the left is the high level structure of the whole software. On the right is the subroutine of the stepper motor run control.

Chapter 4

Implementation

4.1 User Manual

The user manual is presented in this section. It describes two possible ways of controlling the robot. The first method of the robot control is to connect to the Raspberry Pi, which is part of the robot. The second method is to connect an external computer to the RS-485 network and run the control software from there, using a provided USB to RS-485 converter. Both of the methods described use the same scripts described in the section 3.2.3.

4.1.1 Connecting the Raspberry Pi to run the control software

This section of the user manual describes how to connect to the Raspberry Pi in order to control the robot. The Raspberry Pi creates its own Wi-Fi named *SITWifi* and the password is *StaveInsertionTooling*. After connecting to the Wi-Fi, you can remote control it via VNC.

To setup the connection in VNC, go to *File* → *New Connection* and then fill in IP address of the raspberry pi 192.168.4.1 and give it recognizable name. This can be seen on Fig. 4.1.

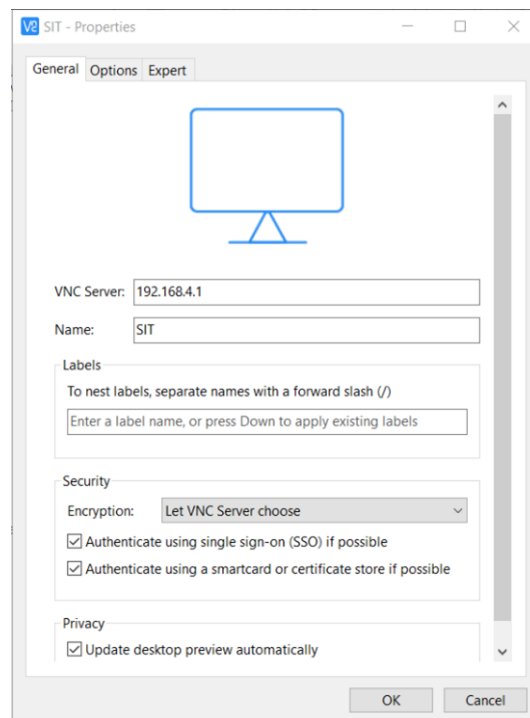


Figure 4.1. VNC connection setup screen.

After setting up the parameters the Raspberry Pi's VNC server should appear in the list. In order to connect to it, the username and password of the account inside the Raspberry Pi will be required as shown in Fig. 4.2. The username is `pi`, and the password is `StaveInsertion`. After this you will get into the Desktop screen with full access.

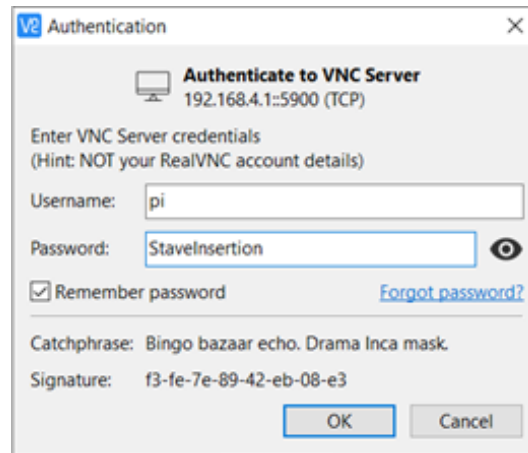


Figure 4.2. Authentication screen.

4.1.2 Preparing an external computer to run the control software

This section of the user manual describes how to prepare an external computer to run the control software. The two scripts for the robot control, `stepper_controller.py` and `robot_control.py`, require python 3.6 or newer to be installed together with libraries `serial` and `colorama`. The communication with the robot is done via a USB to RS485 converter, which require a driver to be installed.

Latest version of python can be downloaded from <https://www.python.org/downloads/>. Make sure you are using stable version 3.6 or higher. During installation, make sure you check the option to add Python to PATH. This enables python to be run directly as a command from command line, which will be needed for the control of the robot. There are two libraries, `serial` and `colorama`, required which are not included in the default python installation. The library `serial` is for the control of the serial line and the `colorama` is used for coloring of the command line interface to provide better readability. Both of the libraries can be installed with a pip command. To install the pyserial use the command:

```
pip install pyserial
```

Similarly, the command to install colorama is:

```
pip install colorama
```

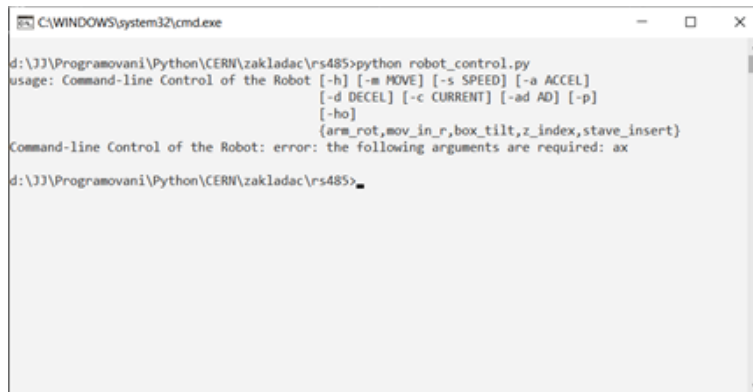
The USB to RS485 converter internally uses CH340 chip for the USB communication and its driver is not always automatically installed on every Windows version. This driver can be found for example here: https://cdn.sparkfun.com/assets/learn_tutorials/8/4/4/CH341SER.EXE.

The scripts `stepper_controller.py` and `robot_control.py` can be located in any suitable directory, but together. Both of them must be in the same directory, as the `robot_control.py` internally uses the `stepper_controller.py` for its function.

4.1.3 Using the control software

The instructions on how to run the control software are described in this section. First, it is described how to run the script. Then the usage of the script is described.

The robot is controlled by the script `python robot_control.py`. First open a command line inside of the directory with the scripts and connect the USB dongle to the PC. Type `python robot_control.py` to run the script. The result should look like on the Fig. 4.3.



```

C:\WINDOWS\system32\cmd.exe

d:\JJ\Programovani\Python\CERN\zakladac\rs485>python robot_control.py
usage: Command-line Control of the Robot [-h] [-m MOVE] [-s SPEED] [-a ACCEL]
                                         [-d DECEL] [-c CURRENT] [-ad AD] [-p]
                                         [-ho]
                                         {arm_rot,mov_in_r,box_tilt,z_index,stave_insert}
Command-line Control of the Robot: error: the following arguments are required: ax

d:\JJ\Programovani\Python\CERN\zakladac\rs485>

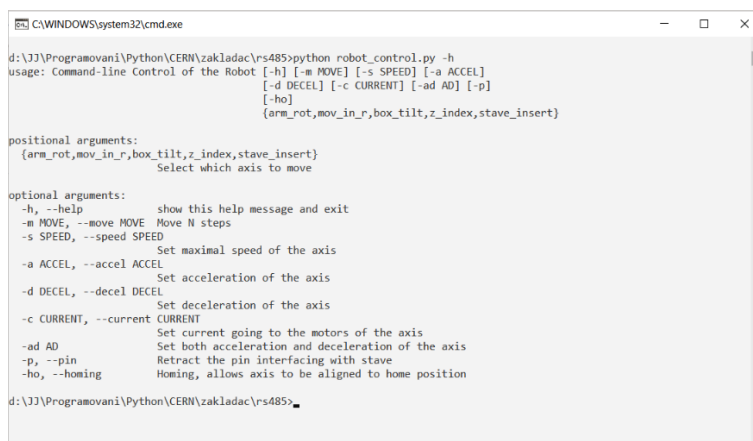
```

Figure 4.3. Program output after running the `python robot_control.py` without any arguments.

The script has built-in help, where is description of every command to control the robot. The help is shown by command

```
python robot_control.py -h
```

and the result is shown on Fig. 4.4.



```

C:\WINDOWS\system32\cmd.exe

d:\JJ\Programovani\Python\CERN\zakladac\rs485>python robot_control.py -h
usage: Command-line Control of the Robot [-h] [-m MOVE] [-s SPEED] [-a ACCEL]
                                         [-d DECEL] [-c CURRENT] [-ad AD] [-p]
                                         [-ho]
                                         {arm_rot,mov_in_r,box_tilt,z_index,stave_insert}

positional arguments:
  {arm_rot,mov_in_r,box_tilt,z_index,stave_insert}
  Select which axis to move

optional arguments:
  -h, --help            show this help message and exit
  -m MOVE, --move MOVE  Move N steps
  -s SPEED, --speed SPEED
                        Set maximal speed of the axis
  -a ACCEL, --accel ACCEL
                        Set acceleration of the axis
  -d DECEL, --decel DECEL
                        Set deceleration of the axis
  -c CURRENT, --current CURRENT
                        Set current going to the motors of the axis
  -ad AD                Set both acceleration and deceleration of the axis
  -p, --pin             Retract the pin interfacing with stave
  -ho, --homing         Homing, allows axis to be aligned to home position

d:\JJ\Programovani\Python\CERN\zakladac\rs485>

```

Figure 4.4. Output of the help command of the script.

Each DoF has defined name inside of the control script so the user can choose which one to control. Names of the degrees of freedom inside the software are written in table 4.1

Table 4.1. Names of degrees of freedom inside the control script.

DoF	Name in software
1. Rotation	arm_rot
2. Radial movement	mov_in_r
3. Tilting	box_tilt
4. Z-indexing	z_index
5. Stave insertion	stave_insert

Only one DoF can be controlled at the same time for safety reasons. Basic command for the movement in the DoF is

```
python robot_control.py <DoF name> <options>
```

where *<DoF name>* should be replaced by its name and the *<options>* are optional commands. The optional commands are:

- **-m** or **--move** *<distance>*
 - Moves the DoF by number of degrees or millimeters, and the direction of the movement is given by the sign of the number.
 - Positive direction is always towards the home position of the DoF, negative towards end switch. The only exception is arm rotation, where the positive movement is counterclockwise.
- **-s** or **--speed** *<value>*
 - Set speed by which the DoF should move during the command execution. Recommended speeds are:
 - Arm rotation: 6000
 - Movement in R: 12000
 - Box Tilt 2000
 - Z-indexing: 2000
 - Stave Insertion: 2000
 - Speed for Z Indexing and Stave Insertion can be modified to lower or higher speeds as necessary but should not exceed 4000.
- **-a** or **--accel** *<value>*
 - Sets the acceleration of the DoF, higher number means higher acceleration. Values over 20000 can cause missed steps of the motors and should be avoided.
- **-d** or **--decel** *<value>*
 - Sets the deceleration of the DoF, higher number means higher deceleration. Values over 20000 can cause missed steps of the motors and should be avoided.
- **-ad** *<value>*
 - Sets both acceleration and deceleration at the same time to the given value. Values over 20000 can cause missed steps of the motors and should be avoided.
- **-c** or **--current** *<value>*

- Sets the input value as current for the motors in milliamps, higher current means higher force of the motor when holding position and when moving, but could result in overheating. Values over 1000mA are not recommended.
- **-ho or --homing**
 - Allows the DoF to be moved to default position and any possible misalignments to be removed. Homing is only effective when moving towards the home position, that is in the positive direction. After each motor hits the home switch, the movement is stopped, and the command has to be repeated until all of the motors are not able to move.
- **-p or --pin**
 - Retracts the pin interfacing stage inside the box. Available only while connected via Raspberry Pi.

4.2 Technical manual

The technical details about the robot are described in this section. This section should be useful for maintaining the control software. First the section mentions the stepper motor controller, its code structure and implementation details. Then the Central control unit details are described.

4.2.1 Stepper Motor Controller

The code of the Stepper motor controller is divided into four files. The main section of the code is the `rs485_stepper.ino`. It contains the main loop described in the section 3.2.2, the setup function to initialize all the necessary variables, functions to send and receive the messages and functions to control the stepper motor. There is also the interrupt service routine (ISR) for the timer directly controlling the stepper motor.

The next files are the `rs485_messages.h` and `rs485_messages.c` together forming the `rs485_messages` library. This library is responsible for handling the messages on the lowest level. The last file is the `defines.h` containing the constants and other definitions used in the controller software.

4.2.1.1 Description of `rs485-stepper.ino`

At the beginning of the file there are two optional defined constants, `R_MOV` and `ROT`. They define what type of motion the controller does. The `R_MOV` constant should be defined if the controller is used for the radial DoF, the `ROT` should be used for the rotational DoF and none should be defined for the rest of the degrees of freedom.

If none of the two constants are defined, the software is set to use the TMC2130 stepper motor driver. If the `ROT` or the `R_MOV` is defined, the software is not trying to set up the TMC2130 and it is expected that an external stepper motor controller is connected. The `ROT` and `R_MOV` are the same except that the `ROT` is controlling the electromagnetic brakes on the stepper motors.

The control of the TMC2130 is described in its datasheet [9] and the library used for the communication with the driver is `TMCStepper` [10]. This library uses the SPI interface of the `ATMega4809`. The settings of the TMC2130 are in the setup function, which is called at the power-up of the controller. The driver is set to 2x microstepping with the `dedge` mode on, which means that the driver recognizes both the rising and

the falling edge as a step trigger. This is chosen in order to speed up the stepper motor as the timer will need to run at half the frequency normally required.

Then the StealtChop is turned on in order to minimize vibrations from the stepper motor. Other important setting of the TMC2130 is the interpolation of the steps, which helps to reduce the vibrations by interpolating the steps of the stepper motors so a lower microstepping can be used with minimal impact on the smoothness of the motion.

To start the stepper motor the function `speed_cntr_Move` is used, which calculates values needed for acceleration, deceleration and speed of the motor and then starts the timer which then controls the STEP signal. The timer selected for this task is TCB2 and it is set to periodic interrupt mode. In each interrupt the ISR is called and a new time for the timer to wait is calculated to match the desired speed profile of the stepper motor. The calculations are discussed in the application note Linear speed control of stepper motor [8].

There are two active UART interfaces. The Serial is used for the programming of the controller as well as the debug output on the USB port of the Arduino. The Serial1 is used for communication with the RS-485 network through the MAX485 chip. Both of these UART interfaces use baud rate of 115200. The mode of the MAX485 must be controlled to set whether the chip should be transmitting or receiving. This is done via the XDIR pin, which is defined to be on all the controllers the pin 2 of the Arduino Nano Every. The XDIR pin is an output pin which is normally set to low to receive, and only when transmitting it should be set to high.

The main loop is in the function `loop`. It first checks the status of the switches, which value is stored in their respective flags, `flagHomeSwitch`, `flagEndSwitch` and `flagOverTemp`. Each switch is checked via its own ISR to ensure independency on the main loop for safety purposes. These ISRs set the flags checked in the main loop.

In each loop the incoming bytes from the RS-485 are processed and if there is a new valid message, the function will react depending on its type. The processing of the messages is described in 3.2.

The controller has a debugging output on the USB of the Arduino. Various information about the current state of the controller are sent, including current state of the motor. The state is if the stepper motor is stopped, accelerating, decelerating or running with a constant speed. The current position of the stepper motor is also displayed in steps done from the start of the controller. Last information is the RS-485 address of the controller.

The controller can also be controlled via this debugging interface even without a connection to the RS-485 network. The move command has format of $\langle \textit{number of steps} \rangle \langle \textit{direction} \rangle$, where the number of steps is an absolute number of steps to move and the direction is `d` if the direction is forward and `b` if the direction is backward. Other commands consist of one letter each, and their list can be seen in table 4.2

Table 4.2. List of debugging commands for the controller.

Command	Description
e	Stop the movement of the stepper motor
x	Enable the electromagnetic brake
y	Disable the electromagnetic brake

4.2.1.2 Description of the rs485-messages library

The library defines the messages used to communicate with the central control unit in the `rs485_messages.h`. There is a type `cmd_type` containing all the types of the messages. It is an enumerated type, which translates the names of the message type to a number, starting from zero. There is also a definition of the message structure in the struct `message`, as described in section 3.2.1.

The functions to convert an incoming series of bytes into a message and vice versa are in the file `rs485_messages.c`. The function `fill_message_buf` takes the message type and converts it into a series of bytes ready to be sent over the RS-485. The function `parse_message_buf` takes a series of bytes and tries to convert them to a meaningful message. If the message is not valid or not intended for the particular controller it reports this information to the caller.

■ 4.2.2 Central Control Unit

The control scripts of the central control unit are intended both as a standalone scripts to allow the user to control the robot via a command line and a library for future control software. The main class is the `Robot`, which uses the classes `Axis` and `Stepper`.

■ 4.2.2.1 class Stepper

The class `Stepper` is intended to control one individual stepper motor controller. Each stepper motor controller should be represented as one instance of this class. To initialize this instance, the class must be provided with necessary information.

The class needs to get the instance of a serial class from the `Serial` library in order to be able to communicate via the RS-485 interface. It also needs the address of the controller to which it is supposed to communicate and a address of the master, by default 50 in hexadecimal.

The class then provides the functions to control the stepper motor controller. The list of these functions and their description is in table 4.3

Table 4.3. List of functions provided for the control of the stepper motor controller.

Function	Parameter	Description
<code>stop</code>	None	Stop the movement of the stepper motor
<code>abs_move</code>	<code>target_pos</code>	Moves the motor to a absolute position of steps
<code>move</code>	<code>n_of_steps</code>	Moves the motor relative to its current position
<code>set_speed</code>	<code>speed</code>	Sets the maximal speed of the stepper motor
<code>set_accel</code>	<code>accel</code>	Sets the acceleration of the stepper motor
<code>set_decel</code>	<code>decel</code>	Sets the deceleration of the stepper motor
<code>set_current</code>	<code>current</code>	Sets the current of the stepper motor in milliamps
<code>set_brake</code>	<code>value</code>	Sets the brake on or off (true or false)
<code>get_curr_pos</code>	None	Gets current position and status of the controller

■ 4.2.2.2 class Axis

The class `Axis` groups together the controllers responsible for the movement of a single DoF by grouping together instances of the class `Stepper` belonging to the controllers. It provides the same functions as the class `Stepper`, which can be seen in table 4.3, but it ensures synchronization between all the controllers. The class needs to get at the initialization the the instance of a serial class from the `Serial` library in order to be able to communicate via the RS-485 interface and list of addresses of the controllers of the individual joints.

4.2.2.3 class Robot

The class `Robot` provides the control of the whole robot. It searches for the correct USB to RS-485 converter with a matching PID to the converter used in the robot. Then it opens the serial port belonging to this converter with a baud rate of 115200 and a timeout of 0.05s.

The class contains instances of the `Axis` classes belonging to all five of the DoF. These instances are then used to control the robot. Apart from accessing the instances to control the robot, there are functions to move each DoF with calculated steps to real distances of the degrees of freedom. The functions provided by this class are in the table 4.4.

Table 4.4. List of functions provided for the control of the robot.

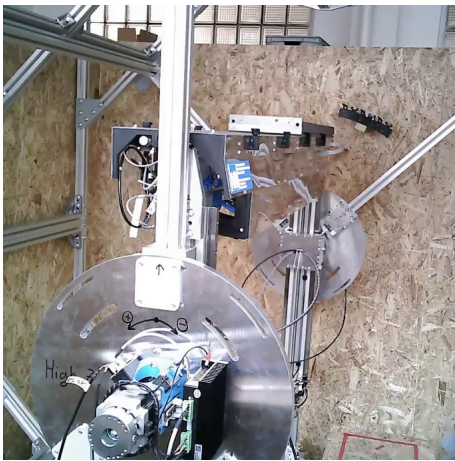
Function	Parameter	Description
<code>rot_arm</code>	angle	Rotates the robot arm by the angle in degrees
<code>mov_in_r</code>	distance	Moves the arm in the radial direction in millimeters
<code>tilt_box</code>	angle	Tilts the box with the stave by the angle in degrees
<code>index_z</code>	distance	Moves the arm in the direction of the z-indexing in millimeters
<code>insert_stave</code>	distance	Inserts the stave in millimeters

The script `robot_control` containing the `Robot` class can also be used to control the robot from a command line in a standalone manner. This usage is described in the section 4.1

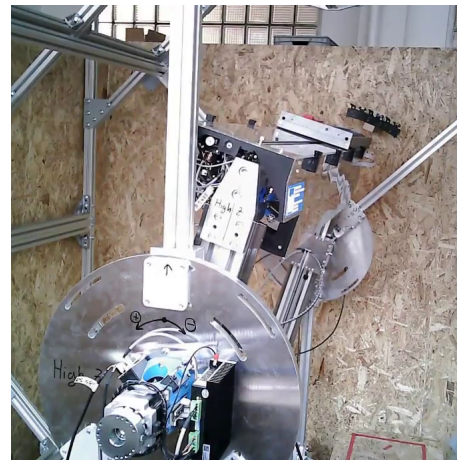
4.3 Testing the system

The system described in the chapter 3 was successfully built and made fully operational. The robot movement was tested in all five degrees of freedom individually and then a test of the full stave insertion procedure was conducted. During these tests photos and videos were taken to document the tests and the functionality of the robot. The robot control was also tested remotely by the future operators of the robot and the stave developers using the created User Manual. Further tests are planned after the delivery of the robot to the stave developers in Oxford with realistic mock-ups of the stave and its transportation box.

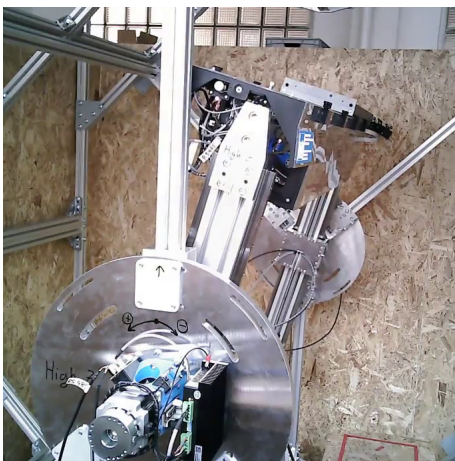
The photos from two full insertion sequences are shown in this section to show the movement of the real robot from two different angles. The first sequence is on Fig. 4.5 and the second sequence is on Fig. 4.6.



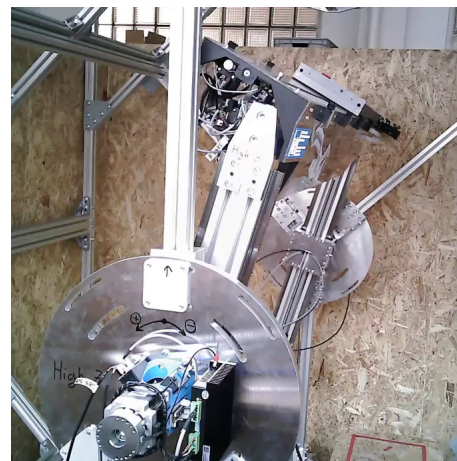
a) Home position



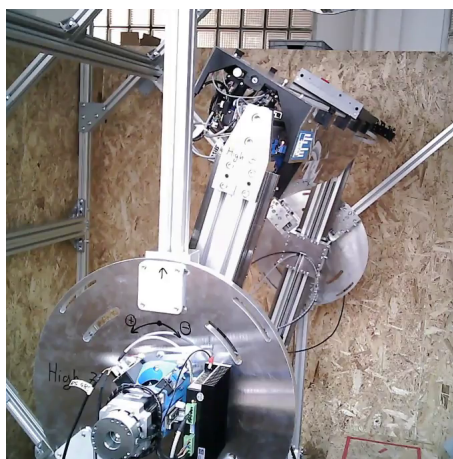
b) After rotation



c) After radial movement



d) After tilting



e) After z-indexing

Figure 4.5. Photos of the first stave insertion sequence.

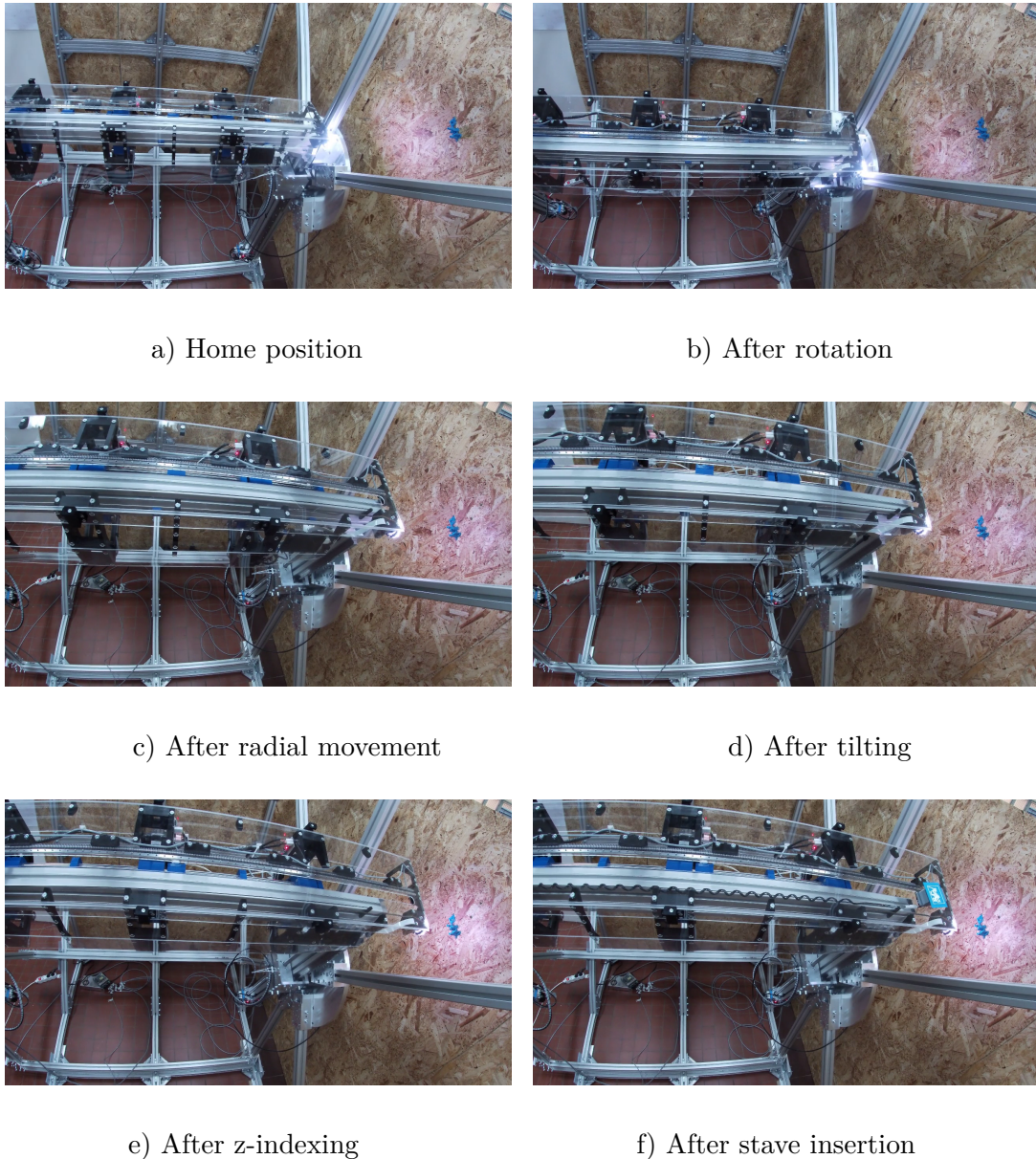


Figure 4.6. Photos of the second stave insertion sequence.

4.4 Future considered improvements

The proposed hardware and software solution of the robot control was proved to be successful and the whole robot is operational. However, during the development and later testing of the robot we identified a number of possible improvements to both the hardware and software of the robot.

In some rare cases, there is a possibility for the motor skipping steps. The controllers currently operate on the assumption that no steps are lost during the whole movement, as the motors should be stronger than any force stopping them. The addition of rotational encoders to the motors would fix this issue by providing the controllers with positional feedback and any potential skipped steps would be noticed. The reliability

of the whole system would increase, but at the cost of additional sensors and change of the design of the controllers.

The communication protocol developed for connection between the controllers and the central control unit currently requires no response from the controllers for any command message setting parameters of the controller. Adding a response packet to acknowledge the successful reception of a received message would increase the reliability of the system and there would be no need to repeatedly send the commands to the controller.

Currently the central control unit is continuously checking the status of the controllers while the motors are running. If there is any error, the stop command is transmitted to all the motors to prevent any damage. The issue could occur if a controller would stop receiving the commands from the central control unit. This could be solved by having a internal timer checking if there was a received message from the central control unit and stopping the movement if a timeout would occur.

The final adjustments of the robot position are planned to be done via a image recognition software and using the camera connected to the Raspberry Pi. This feature is planned to be developed by a team colleague and will further improve the precision and reliability of the whole system.

The control of the robot via a command line is meant to be for a basic control of the whole robot. This software is planned to be used as a library for a more sophisticated control software, where the user will be able to select the stave to be inserted and the whole operation will be fully automatic. The image recognition is a planned feature of this future software.

Chapter 5

Conclusion

This thesis presented the hardware and software solution to the control of the robot for inserting detectors in the ATLAS Inner Tracker. The solution was successfully implemented, tested and it is fully functional.

The description of the mechanical design of the robot and the requirements for the control system were described in the chapter 2. The robot has five degrees of freedom with total of ten stepper motors and multiple motors are used to drive a single DoF.

The stepper motor controllers were developed for all stepper motors and they are connected to a RS-485 network together with a central control unit based on a Raspberry Pi. There are two variants of the stepper motor controllers based on the used driver and the size of the stepper motors. The central control unit is responsible for synchronous operation of the stepper motor controllers. The communication protocol was developed for the connection between the central control unit and the stepper motor controllers.

The control software of the robot can be used as a library allowing future extensions of the robot control, or it can be directly used to control the robot by the user. The user can move the robot in each degree of freedom and set the parameters of the stepper motors, such as maximal speed, acceleration or current. The User Manual and the Technical Manual were created and they are described in the chapter 4.

The full functionality of the robot was tested and a number of test sequences of the detector insertion were conducted. The robot was remotely tested by the developers of the detectors and further tests will take place in their laboratory in Oxford. The tests proved that the robot is fully operational and some improvements were proposed to improve the reliability. The user interface is planned to be improved in the near future as this is only a first prototype of the robot. The future control software will allow the user to select the desired position via a GUI and all the movements will be fully automatic.

References

- [1] Sven Wonsak, and ATLAS Collaboration. *The ATLAS ITk Strip Detector System for the Phase-II LHC Upgrade*. 2020. CERN.
<http://cds.cern.ch/record/2706027>.
- [2] CERN. *Technical Design Report for the ATLAS Inner Tracker Strip Detector*. 2017.
<https://cds.cern.ch/record/2257755>.
- [3] Science, and Technology Facilities Council. *ATLAS Inner Tracker Upgrade (ITk)*.
<https://www.ppd.stfc.ac.uk/Pages/ATLAS-SLHC-Upgrade.aspx>. 2020. Accessed: 2021-05-08.
- [4] Analog Devices. *RS-485/RS-422 Circuit Implementation Guide*.
<https://www.analog.com/media/en/technical-documentation/application-notes/AN-960.pdf>.
- [5] Raspberry Pi Foundation. *Raspberry Pi Documentation*.
<https://www.raspberrypi.org/documentation/>. 2021.
- [6] Hen Marais, and Microchip Technology. *ATmega4808/4809 Data Sheet*.
<https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega4808-09-DataSheet-DS40002173C.pdf>.
- [7] Maxim Integrated. *MAX481/MAX483/MAX485/MAX487-MAX491/MAX1487 Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers*.
<https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf>.
- [8] Microchip Technology. *AVR446: Linear speed control of stepper motor*. 2016.
<https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en591185>.
- [9] Trinamic. *TMC2130 Datasheet*.
<https://www.trinamic.com/products/integrated-circuits/details/tmc2130/>. 2020. Rev. 1.12.
- [10] Teemu Mäntykallio. *TMCStepper*.
<https://github.com/teemuatlut/TMCStepper>. 2021.