CONTENTS

LIST OF USED ABBREVIATIONS

ACD – Actuator Control Device

ADC – Analog-to-Digital Converter

BLDC – Brushless DC motor

CAN – Controller Area Network

DC – Direct Current

DMA – Direct Memory Access

GDP – Gross Domestic Product

GPIO – General Purpose Input/Output

HAL – Hardware Abstraction Layer

I2C – Serial asynchronous bus

LED – Light Emitting Diode

MC – Microcontroller

MCU – Microcontroller Unit

MOSFET – Metal-Oxide-Semiconductor Field-Effect Transistor

PID – Proportional-Integral-Derivative controller

PLL – Phase-Locked Loop

PWM – Pulse Width Modulation

SCD – Strategic Control Device

SPI – Serial Peripheral Interface

SSH – Secure Shell

SVPWM – Space Vector Pulse Width Modulation

TCD – Tactical Control Device

UART – Universal Asynchronous Receiver/Transmitter

UVLO – Under Voltage Lock Out

kOPS – Thousand Operations Per Second

INTRODUCTION

The active dissemination of robots after the mid-20th century led to the automation of nearly all aspects of production processes, including assembly, welding, painting, quality control, and packaging. This also had a significant impact on logistics, warehousing, and distribution, where robotic systems are used for sorting, moving, and storing goods. Advances in mechanics, electronics, and digital devices led to the creation of a variety of devices.

The use of existing universal mini manipulator robots is an effective and rational solution, due to their small size and parameters suitable for working with small parts and structures. However, despite the advantages of mini manipulator robots, they have disadvantages. The main disadvantage is the danger of robots working together with humans. To organize the work of these robots, it is necessary to create special enclosed work areas to minimize potential risk, and working together with a human is deemed impossible. A mini robot cannot become collaborative if power or force is limited for each link. To make the robot collaborative, changes need to be made to the robot's design. The size of mini manipulator robots creates limitations for all system elements, and installing torque sensors is not the best solution, as they take up additional space in each link of the robot and complicate the mini robot's design.

Collaboration is provided by a combination of information devices, motors, and control program algorithms. Among the listed set, motors play an important role. The most suitable type of electric motor is the brushless DC motor, among the many options of brushless DC motors, Gimbal motors are characterized by high power with a small size.

The goal of the work is: Development of a control system for a collaborative mini manipulator robot to demonstrate the capabilities of using Gimbal-type motors as the robot's actuating device.

The work contains 5 sections. The first section analyzes mini manipulator robots available on the market, various motor options are analyzed, and a choice of motor type is made, defining functions and technical parameters for the mini robot control system. The second section develops a structural diagram of the mini robot control system based on a hierarchical robot control paradigm, and an algorithm for its operation is created. The third section selects functional elements and analyzes microcontrollers for developing the control system, creates a functional diagram, and provides its description. The fourth section develops the principal circuit with its description and calculation of circuit elements. The fifth part develops control system algorithms and software implementation on a microcontroller.

1 RELEVANCE OF GIMBAL MOTORS IN COLLABORATIVE MINI ROBOTS

1.1. Overview of Mini Robot Features

The active spread of robots after the mid-20th century led to the automation of almost all aspects of production processes, including assembly, welding, painting, quality control, and packaging. This also had a significant impact on logistics, warehousing, and distribution, where robotic systems are used for sorting, moving, and storing goods. Progress in mechanics, electronics, and digital devices led to the creation of various devices. This, in turn, contributed to progress, improving the quality and capabilities of previously developed solutions. One of the improvements in recent developments can be considered the emergence of mini manipulator robots. The use of existing universal mini manipulator robots is an effective and rational solution due to their small size and parameters suitable for working with small parts and structures.

However, it should be noted that most of the work related to the assembly of small-sized parts is still performed by humans.

Improving the characteristics and parameters of mini manipulator robots is very relevant. Mini manipulator robots are presented on the market, and developments are being carried out in this direction (Li et al., 2022). Examples of such robots include the ”MotoMini” models from ”Yaskawa”, ”Meca500” (Figure 1.1) from ”Mecademic”, and others. The topic of mini robots is relevant, but it should be noted that the amount of information currently available on robots of this class is extremely small.

Figure 1.1. Meca500 Robot Arm

Thanks to the small size of the entire structure, it is possible to create entire production lines right on the table. This opens up new possibilities for small-scale production and prototype development, where high precision and flexibility of processes are required. However, despite the advantages of mini manipulator robots, they have disadvantages. The main disadvantage is the danger of robots working together with humans. To organize the work of these robots, it is necessary to create special enclosed work areas to minimize potential risk, and working together with a human is deemed impossible. This problem limits the application of mini robots, making them unsuitable for tasks requiring close interaction between humans and machines, especially when working with small objects that require operator participation. This also reduces the flexibility of production systems, as changing the configuration of the line may require significant time and resources to reconfigure the work area. The lack of collaboration is the main problem with these types of robots.

1.2. Features of Collaborative Manipulator Robots

A collaborative robot, or cobot, is an industrial type of robot equipped with a set of sensors and locking devices that help to avoid collisions with humans and various obstacles with high probability, even in cases of software malfunction. These devices are designed to work in close proximity to humans, not requiring placement in isolated work areas. They are also characterized by ease of programming, making them accessible for use by personnel without specialized skills.

According to the international standard ISO 10218 (parts 1 and 2) (ISO, 2011), four main types of collaborative robots (Bélanger-Barrette, 2015) are defined, which can be used for collaborative work with humans.

- **With a safety-rated monitored stop.** Such collaborative robots can work autonomously according to the input control program, but when a human appears, the robot stops working. The robot's operation continues when the person leaves the work area;
- **With manual control.** In the presence of a person, such a collaborative robot performs his commands, given by hand. For example, a command can be given by touching a certain part of the body or recognizing hand movements at a distance. In the absence of a person, the collaborative robot can work autonomously if the necessary control program is entered;
- **With "computer vision".** Special sensors track the movement of people. When a person enters the work area, a reduced, safe speed is established. If he gets too close, the collaborative robot stops working;
- **With limited force.** These cobots do not sweep everything in their path when moving. If they encounter an obstacle, they stop, including when colliding with a person. Such collaborative robots can be used in close proximity to people.

ISO/TS 15066:2016 is the first of its kind document (ISO, 2016), establishing safety criteria for the use of collaborative robots. The development of this technical standard was carried out by the international ISO committee, which included specialists from 24 countries and representatives of leading robotics manufacturers, starting in 2010. The standard supplements the existing safety standards for industrial robots, recorded in ISO 10218-1 and ISO 10218-2, and sets specific requirements for safe interaction between industrial robots and humans in the work environment. ISO/TS 15066 provides methodological recommendations for conducting risk assessments related to collaborative use of robots and humans.

Safety-rated monitored stop. This method is used when the robot mainly works autonomously, but sometimes human intervention in the work area is required. For example, the robot may be processing a part, but at a certain point in the process, human intervention is needed to perform a task unavailable to the robot. In the event of a person entering the established safety zone, the robot stops, while the motors remain energized and ready to resume work. The advantage of the method is that after the person leaves the safety zone, the robot immediately continues its activity, eliminating the need for a full program reload, which could occur in the event of a complete stop. This method eliminates downtime in the robot's work, which would occur if people regularly moved through its work area;

- The stop is ensured without loss of motor power (pause);
- The operator can interact with the robot;
- Automatic operation can resume when the person leaves the workspace;
- At any one time, either a person or a robot can move;
- Can be used with conventional industrial robots, but light safety barriers (laser rangefinders, photodetectors) need to be added.

Hand guiding. This collaboration method is used for performing complex manipulations with heavy objects and can be adapted for use with traditional industrial robots by adding a special device. This device, usually a torque sensor installed on the robot's flange, allows sensing the force applied by the operator to the manipulator, thus providing the possibility of more precise control over the robot's actions.

- The operator is in direct contact with the robot.
- The robot is under manual control.
- Both the human and the robot can move simultaneously (movements are controlled by the human).
- Conventional industrial robots can be used.
- Additional equipment is required (torque sensor).

Speed and separation monitoring. In this approach to collaborative work, the robot's workspace

is limited by light safety barriers, similar to the first type of collaboration, which monitor the location of people around. The main difference lies in the goal: while in the first case, the robot's main task is to stop when a person approaches, in this scenario, the goal is to allow parallel work of the person and the robot. The robot adapts its behavior according to pre-defined zones in its program: as the distance to a person decreases, the robot gradually reduces the speed of its movements and stops at critical proximity to prevent collision. When the person moves away, the robot resumes work and increases speed.

- The robot's speed decreases as the person approaches.
- The robot stops when a collision with a person could occur.
- Both the human and the robot can move simultaneously.
- Can be used with conventional industrial robots, but light safety barriers (laser rangefinders, photodetectors) need to be added.
- Used for operations requiring frequent staff presence.

Power and force limiting. Torque sensors are located in the robot's "joints", which can detect a collision with a person. If the sensors detect forces exceeding permissible limits, the robot stops. These robots are also designed to dissipate forces over a wide surface in case of impact – this is why their body parts are often made with rounded shapes. Power and force limiting functionality is typically included in the standard software.

All the above forms of collaborative work can be used with conventional industrial robots, provided that additional devices are present. Such solutions are called collaborative robotic systems. The difference with a collaborative robot is that it may not use additional (external) safety devices – they are already built into it. Another important difference between collaborative robotic systems and collaborative robots is that in collaborative work, contact with a human is excluded. Meanwhile, a collaborative robot can contact the human body without harm to it. This is achieved by regulating power and force.

1.3. Problem with Existing Mini Manipulator Robots

A mini robot cannot become collaborative if power or force is limited for each link. To make the robot collaborative, changes need to be made to the robot's design. The size of mini manipulator robots creates limitations for all system elements, and installing torque sensors is not the best solution, as they take up additional space in each link of the robot and complicate the mini robot's design. Other functional devices of the robot, which would ensure the features of a collaborative robot, need to be considered. The main part of the robot that provides the movement of links is the drive. Drive mechanisms are connected to the kinematic elements of the manipulator and perform movements. The drive of any manipulator robot consists of:

- An energy converter
- An electric motor
- A transmission mechanism

An electric motor is an electrotechnical device whose main function is to convert electrical energy into mechanical energy. To ensure the functions of a collaborative robot, it is necessary to choose a suitable type of electric motor.

The most commonly used motors in manipulator robots are DC motors, stepper motors, and servomotors.

- DC motors come in brushed and brushless types. It is generally accepted that brushless DC motors have displaced brushed ones. A brushless DC motor is quieter, efficient at low speeds, and can maintain constant maximum torque, but this type of motor is complex to control (Group, 2020).
- Stepper motors can control precise movement, have maximum torque at low speeds, and are simple to control. However, they have several drawbacks: they are noisy and relatively inefficient, and they overheat since they constantly consume maximum current, skip steps under high loads, and create additional vibration at low speeds of rotation (Gorupec, 2014).
- Servomotors provide extremely precise movement. The disadvantages of using servomotors include the possibility of jitter when responding to feedback and the need for complex control logic, thus taking up a lot of space and adding complexity to the robot's design. Servomotors also assume the installation of a gearbox with a high reduction ratio, which also affects the dimensions and complexity of the robot itself.

The most suitable type of electric motor for a manipulator robot, as seen from table 1.1, is the brushless DC motor due to its high torque at low speeds. Gimbal motors, which are brushless DC motors designed for use in camera gimbals, have very high torque at low speeds among this subtype of DC motors. This feature simplifies the choice of gearbox, as a high reduction ratio is not required, which simplifies the gearbox design and makes it more compact. A small reduction ratio creates the possibility of reverse control of the drive (Matsuki et al., 2019).

Table 1.1

Table of characteristics of motors of various types

Name	Type	Power (W)	Torque (Nm)	Weight (g)	RPM	Reduction Ratio to 60 RPM	Torque at 60 RPM (Nm)
GM8112	Gimbal	80	0,89	428	420	7	2,45
GM110-8T	Gimbal	50	0.35	249	209	4	1,4
EC 45 Flat	BLDC	50	0.09	110	2360	39	3,53
EC 45 Flat	BLDC	70	0.134	140	2540	42	5,67
Tarot 4008 330kv	BLDC	460	0.63	80	7920	132	83,16
Nema 8 x 38mm	Step	-	0.03	80	1200	20	0,6
Nema 17 x 34mm	Step	-	0.26	230	1200	20	5,2
Nema 23 x 76	Step	-	1.9	1200	1200	20	38
TSM4102	Servo	50	0.159	400	3000	50	7,92

r

Also, according to the trends of recent research (Sakama et al., 2022) fig 1.2, the specific power of brushless DC motors has increased more than tenfold over 20 years, and now brushless DC motors are electric motors with the highest specific power. The specific power of these motors increased after the appearance of permanent magnets with high magnetic energy. In particular, the appearance of the neodymium magnet.

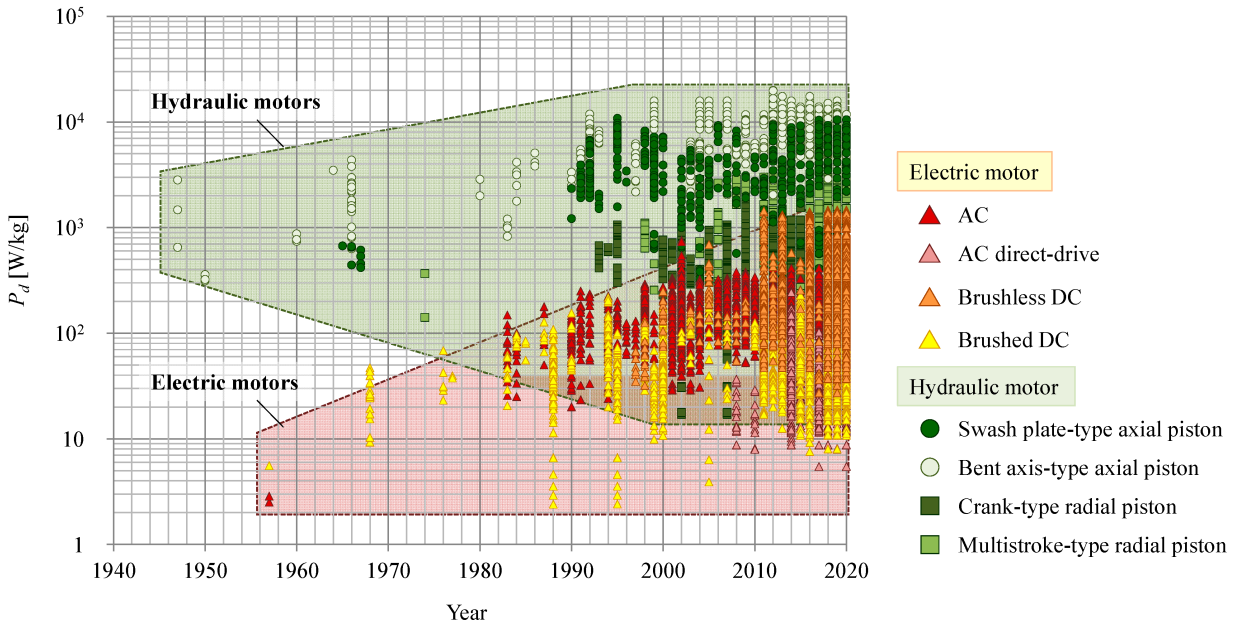

Figure 1.2. Changes in the specific power of electric motors over time (Sakama et al., 2022)

1.4. Requirements for Electric Motor Control

To ensure all the features of a collaborative robot, the most suitable type of electric motor, based on section 1.3, is the brushless DC motor. Regarding the mini robot, the Gimbal motors BLDC type is the most suitable. However, Gimbal motors have disadvantages inherent in all BLDC motors: the complexity of their control. Options for controlling brushless DC motors:

- Control with six-step commutation;
- Pulse Width Modulation (PWM);
- Control with vector modulation (FOC);
- Sensor and sensorless control.

When choosing a control method, it was necessary to consider the need for smooth control of speed, acceleration, and motor torque. Control with vector modulation is the most suitable method for controlling the electric motor axis of a mini manipulator robot. FOC provides high efficiency and smooth operation by independently controlling the magnetic flux and torque of the motor. This is achieved by converting the three-phase signal into a two-phase one in a rotating coordinate system, which is used for further control. Vector regulation requires determining the current position of the motor rotor using angle sensors.

When developing a control system for a collaborative mini robot, it is necessary to consider not only the differences in drive control but the control system as a whole. Therefore, it is necessary to make changes to the entire control system of the mini robot.

1.5. Defining Functions and Technical Specifications for the Control System of a Collaborative Mini Robot

As mentioned earlier, any manipulator robot consists of various systems: electronic, mechanical, computational, sensory, and others. The robot's control system must organize work between them and ensure the operation of the manipulator robot according to the following requirements:

- Precise movement control, smooth control of the robot's movements with high accuracy, including the synchronization of all joints and mechanisms;
  vbnet
- Implementation of the given algorithms and tasks, includes tasks of the sequence of movements and operations of the robot's actions;
- Processing of sensory data, accurately process data from sensors to adapt to changes in the environment;
- Interaction with the operator, possibilities for manual control and programming;
- Integration with other systems, the robot to be not only one element in production, used together with conveyor line systems.

The control system of a collaborative mini robot does not differ much from the requirements of a regular industrial manipulator, but it must ensure functions:

- Calculation of the motion trajectory, solving kinematic tasks;
- Precise and smooth movement control of the motor using a vector control algorithm with feed-

back;

- Control of speed, acceleration, motor torque;
- Limitation of minimum and maximum parameters of speed, acceleration, and position, current voltage;
- Torque control;
- Real-time processing of data from various sensors;
- Visual display of the robot's state;
- Possibility of integration with other systems;
- Monitoring of temperature and device shutdown at high temperatures.

Developing a control system for a mini manipulator robot is impossible without considering the design of the robot itself. Based on the data performed in section 1.1. It was determined that the control system of a collaborative mini robot manipulator using Gimbal motors should have technical parameters corresponding to the characteristics of existing mini robots. The robot design is a 6-axis manipulator robot: six degrees of freedom are necessary to approximate the possible movements of the human hand. For the development of the control system of the mini manipulator robot, the most commonly encountered kinematic structure is shown in figure 1.3.

Parameters of the kinematic structure:

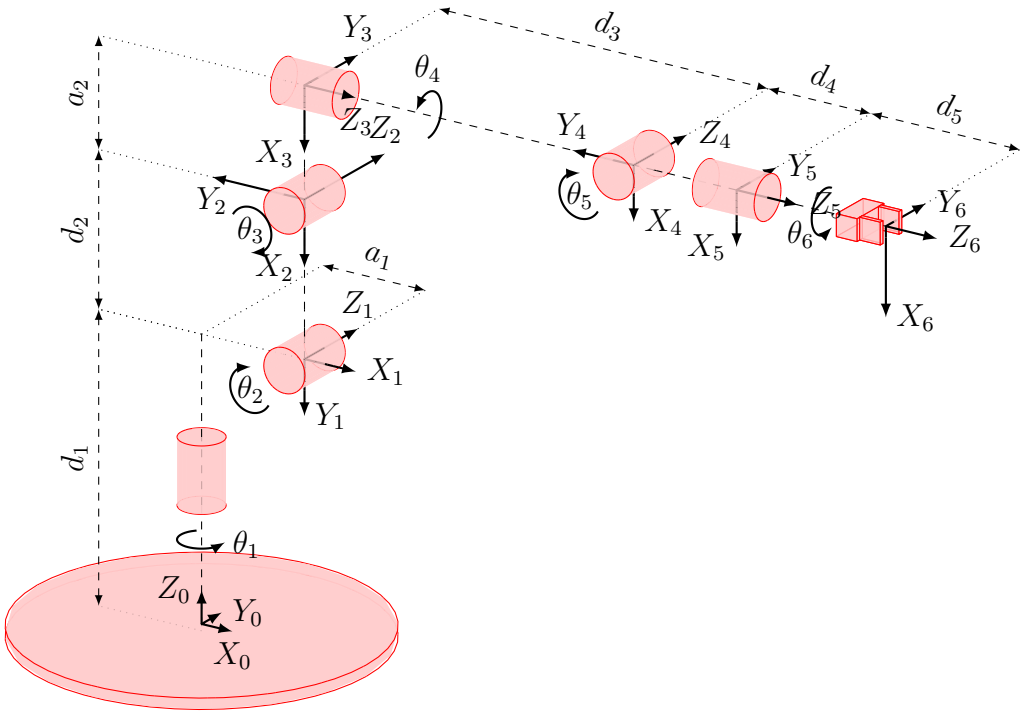

Figure 1.3. Kinematic structure of a mini robot

- d1, d2, d3, d4, d5 - distances between links;
- θ1, θ2, θ3, θ4, θ5, θ6 - angles of the links;

- a1, a2 - lengths of the links;
- X0, Y0, Z0 - base coordinates;
- X6, Y6, Z6 - tool coordinates.

The existing parameters from the mini manipulator robot MotoMini from ”Yaskawa” were taken as the basis for the implementation of the control system: Technical specifications:

- Number of axes – 6 axes;
- Repeatability 0.02 mm;
- Maximum operating temperature limit: 80 °C;
- Limiting the maximum motor current to 25 A;
- The size of the control system should not exceed 300x300mm;
- The mass of the control system should not exceed 5 kg;
- Minimum axis rotation speed not less than 150 °/s;
- Average power consumption <200W;
- Operating voltage 24V.
- Angle limitations:
    - Axis 1 from -180° to +180°;
    - Axis 2 from -90° to +90°;
    - Axis 3 from -90° to +90°;
    - Axis 4 from -180° to +180°;
    - Axis 5 from -180° to +180°;
    - Axis 6 from -360° to +360°;

2 STRUCTURAL DIAGRAM OF THE CONTROL SYSTEM FOR A MINI ROBOT

2.1. Feature of the hierarchical structure

The control system for a collaborative mini manipulator robot includes other systems, such as motor control, internal and external peripherals, and others, which are interdependent. Therefore, to develop the structural diagram of the control system for a collaborative mini manipulator robot, a hierarchical paradigm of control system architecture for robots was used. According to this paradigm, the hierarchical architecture of the robot control system (Fig. 2.1), in general, contains four levels of control: intelligent level, strategic level, tactical level, and executive level (Khatib et al., 1997).

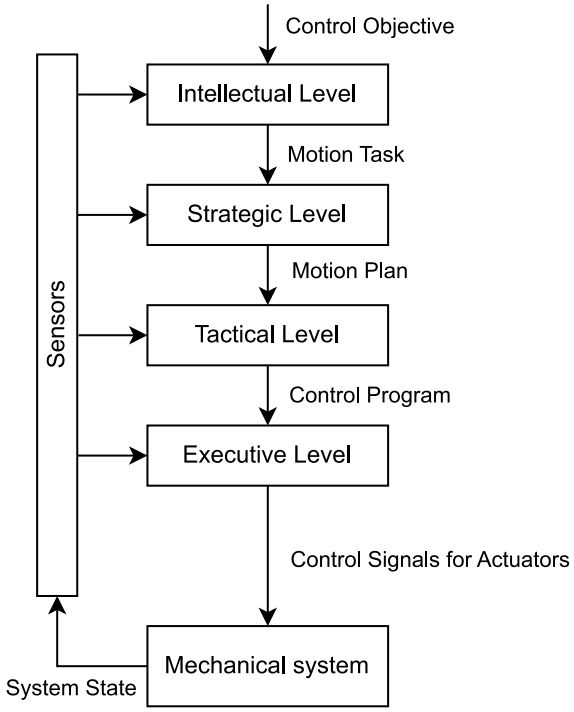

Figure 2.1. Hierarchical architecture of the robot control system

In the hierarchical architecture of the manipulator robot control system, the levels of control ensure the performance of specific robot control tasks:

- At the intelligent level, decisions are made (formulated) about the actions performed by the manipulator robot under conditions of incomplete information about the external environment and the objects the robot interacts with (for example, moving an object from one point in space to another, processing a part, etc.).
- The strategic control level is intended for planning the movements of the manipulator robot: dividing the action task, developed at the intelligent level, into a sequence of temporally coordinated elementary actions (movements) and formalizing the control objectives for each of these actions. The strategic level issues to the tactical level a motion plan in the form of motion con-

trol commands. For example, moving the robot's end-effector (gripping device) to a specified position, grasping an object, test movement to determine the reaction forces from the object, moving the object, and returning the robot to its original position.

- At the tactical level, the motion control commands coming from the strategic control level are transformed into a control program that defines the laws of coordinated motion over time of all the robot's links, considering the technical characteristics of the drive unit (primarily limitations on generalized velocities, accelerations, and forces). To execute the position control command for the manipulator robot at the tactical level, it is necessary to determine the generalized coordinates of the manipulator that correspond to the desired Cartesian coordinates of the characteristic point of the manipulator robot's gripping device. For this purpose, the inverse kinematic problem of the manipulator's position at a given point of the motion trajectory is solved.
- The executive control level is designed to calculate and issue control signals to the manipulator robot's drive unit in accordance with the control program created at the tactical level, considering the technical characteristics of the actuating devices.

2.2. Structure diagram of the mini robot control system

Figure 2.2 illustrates the structural diagram of the mini robot control system. The control system for the mini manipulator robot has a hierarchical structure that corresponds to the hierarchical architecture of the robot control system (Fig. 2.1), as the developed structural diagram includes devices corresponding to the functions of control levels in (Fig.2.1):

1. Strategic Control Device (SCD)
2. Tactical Control Device (TCD)
3. Actuator Control Devices (ACD)

The Tactical Control Device (TCD) of the mini robot coordinates the operation of all devices in the mini robot control system, ensuring the efficient functioning of the entire robot mechanism. It processes control commands from the Strategic Control Device (SCD) and is responsible for forming output signals to Actuator Control Devices (ACD), ensuring the robot's behavior corresponds to the given commands.

The Actuator Control Device (ACD) is a device whose main purpose is to generate signals for precise control of a single mechanical device (electric motor), providing rotation of the manipulator robot arm. This device allows for motor control without using the Tactical Control Device (TCD). This enables complex manipulations with a high degree of precision and turning the motor axis to the required angle.

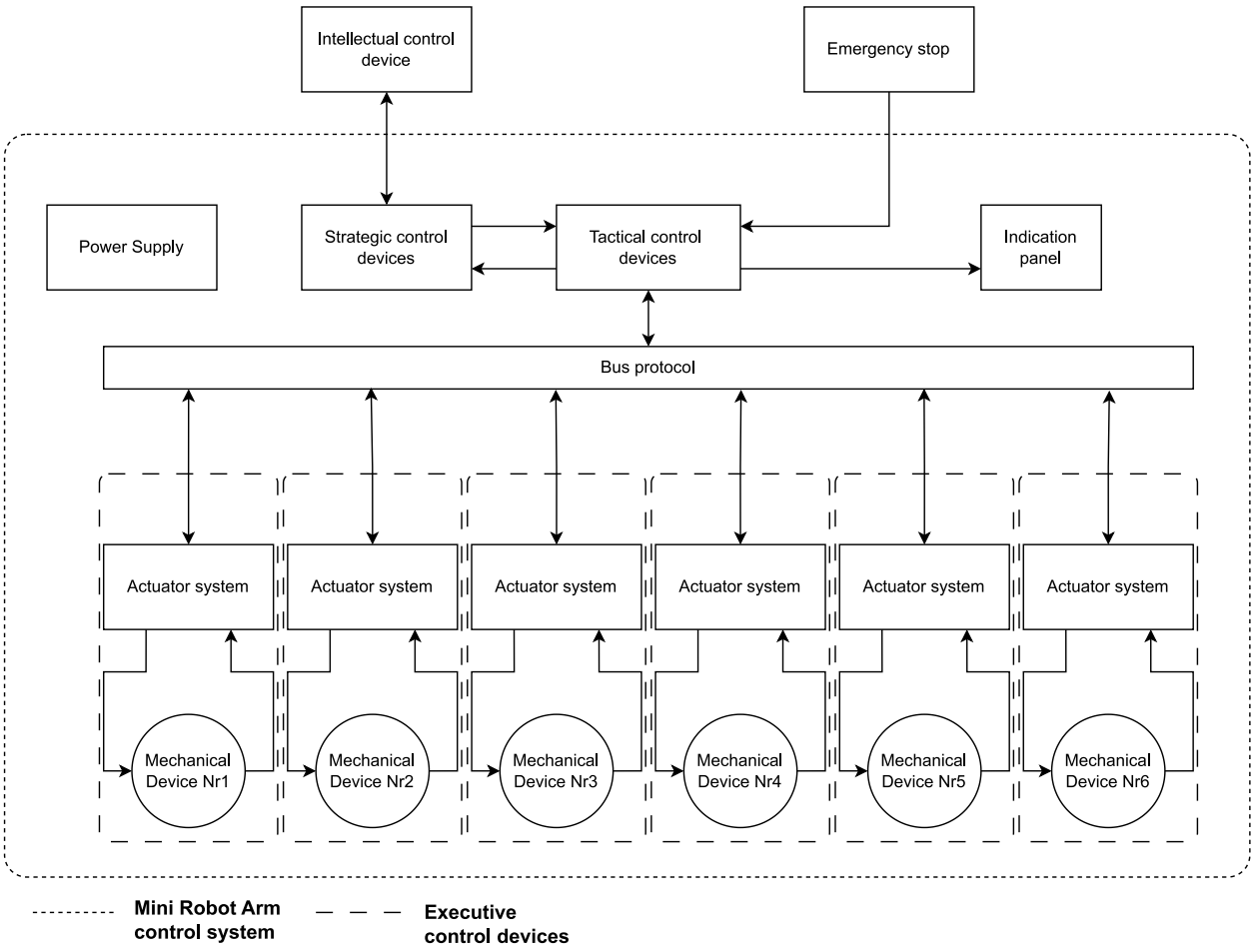

Figure 2.2. Structural diagram of the mini robot control system

After receiving data from the Strategic Control Device (SCD), the Tactical Control Device (TCD) sends data to the motor control system of each axis (ACD).

After receiving a motion command, positions and accelerations are calculated, and data is sent to the motion control system (ACD) of each axis. If the controlled parameters go beyond the range of allowed values, an emergency mode is triggered, and information is displayed on the information panel.

The indication panel provides information about current parameters and system states, as well as the robot system's state. The device facilitates monitoring the state of the manipulator robot and ensures quick problem detection by notifying the operator. When the system determines further motion of the robot is impossible, red indicates this, yellow for waiting, and green for operation.

The power supply is an integral element of the system. It converts network electricity, by forming the required current and voltage values (24V), to power all the electrical devices of the robot.

The voltage converter is necessary to change the power supply voltage levels to the required parameters, as different devices have various permissible power supply voltage and current values.

Brushless DC electric motors convert electrical energy into mechanical energy. They provide

motion to various axes of the manipulator robot.

The emergency stop device is used for the immediate stopping of the robot in emergency situations, preventing accidents and damage to structures.

The data bus ensures communication between the Tactical Control Device (TCD) and Actuator Control Devices (ACD). This communication allows efficient coordination of all motion control devices (ACD) actions among themselves, as well as simultaneous synchronization during movement.

2.3. Structural Diagram of the Actuator Control System

The structural diagram of the axis motion control systems is depicted in Figure 2.3. The purpose of this system is the direct control of the synchronous motor and monitoring its parameters. The main control elements of the system are the Actuator Control Devices (ACD). Since controlling an electric motor requires signals with frequency and phase regulation, it is the ACD that generates the controlling PWM signals.

Drivers amplify the power of the control PWM signals, thus ensuring the compatibility of the ACD signal with the output power transistors. The drivers allow for timely and safe switching of the inverter transistors.

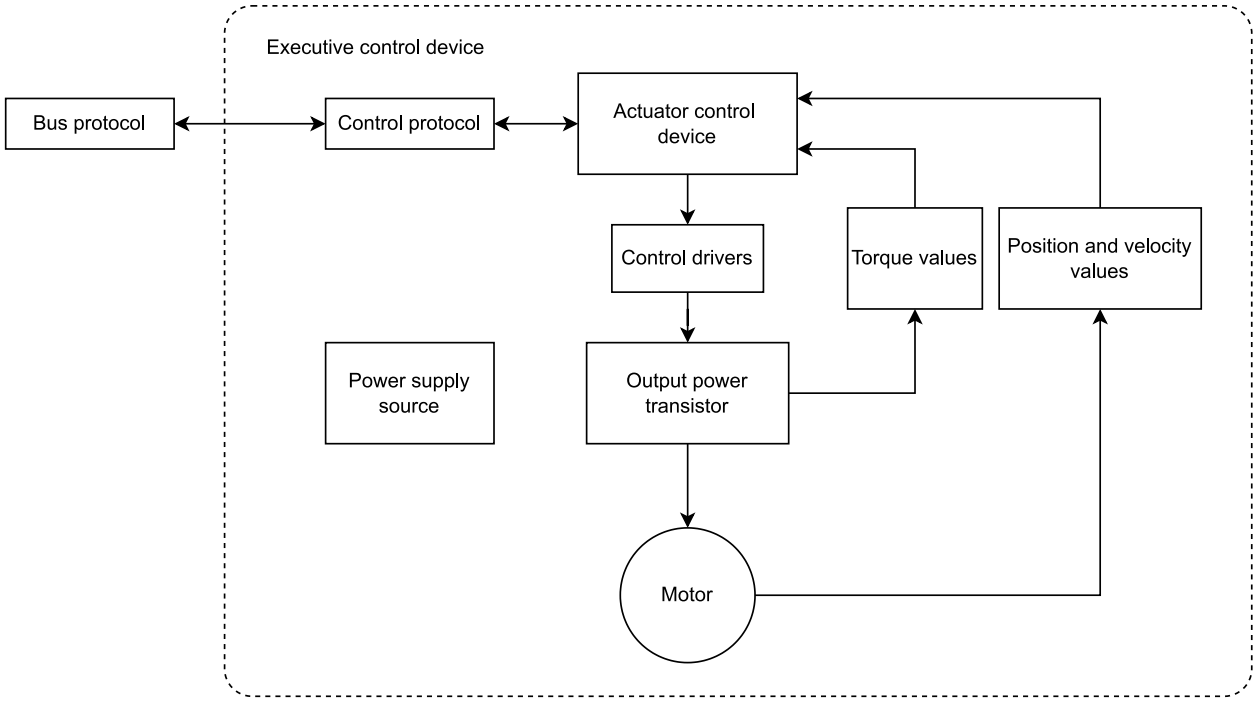

Figure 2.3. Structural diagram of the actuator control device

The signals supplied to the electric motor are controlled by power switching transistors. The electric motor needs to be regulated by changing the frequency and phase of the signal.

Power transistors form a three-phase bridge voltage inverter, and the operating mode of the

transistors is switching. In this mode, the transistors switch between fully open and fully closed states, which ensures high efficiency and minimizes transistor heating, which is especially important when creating a small-sized device.

The formation of PWM signals is carried out using the values of the current rotation angle of the axis, acceleration, and torque on the motor. By changing the width of the PWM pulses, the ACD control system can regulate the amount of energy supplied to the motor, thereby changing the speed and torque parameters.

The voltage converter provides the formation of supply voltages for various devices. The type of converter is a DC/DC converter, and the converted voltage is the 24V voltage from the common power source. The system contains various components with different levels of supply voltage, so supplying the power voltage from the outside is not an efficient method.

The data bus allows for data transmission between different devices by connecting components into a single network. Thanks to the use of the network, it becomes possible to exchange data with one or several devices simultaneously.

2.4. Algorithm of the Tactical Control Device Operation

Figure 2.4 shows the main algorithm for the Tactical Control Device operation, whose main task is to control the motor of a single robot axis. By default, when the system is turned on, the system initialization takes place, which involves checking the communication functionality between devices and verifying the non-critical state of each ACD and motor. As the final step, the indication panel parameters are initialized and set.

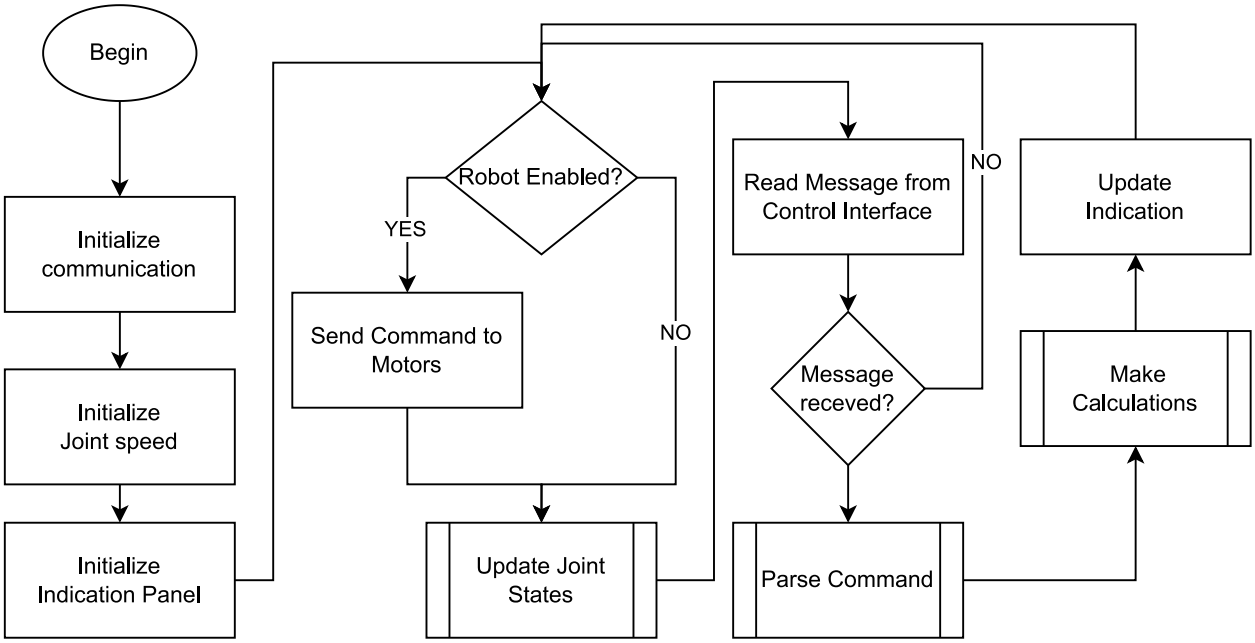

Figure 2.4. Algorithm of the Tactical Control Device operation

If the connection with the motors and the ACD interface has not been normalized, the control system will remain in standby mode until the connection is restored. After successfully connecting to all robot devices, commands are set and sent to the ACD actuator devices, setting the minimum values of speed, angle, and acceleration of rotation for each robot axis. Since the position of the robot's axes after the previous shutdown is unknown, it is necessary to calibrate all axes. Calibration occurs after receiving a permission command from the Strategic Control Device (SCD).

In the system's operation mode with the robot's powered-on flag raised, after receiving the respective command, the angles of each axis are read, and commands are sent to the ACD depending on the calculations made. Thus, the Strategic Control Device (SCD) does not participate in the path-building calculations but only sends data to the Tactical Control Device (TCD).

The "Parse command" algorithm is illustrated in Figure 2.5. After receiving a command from the (SCD), the movement type calculation is executed.

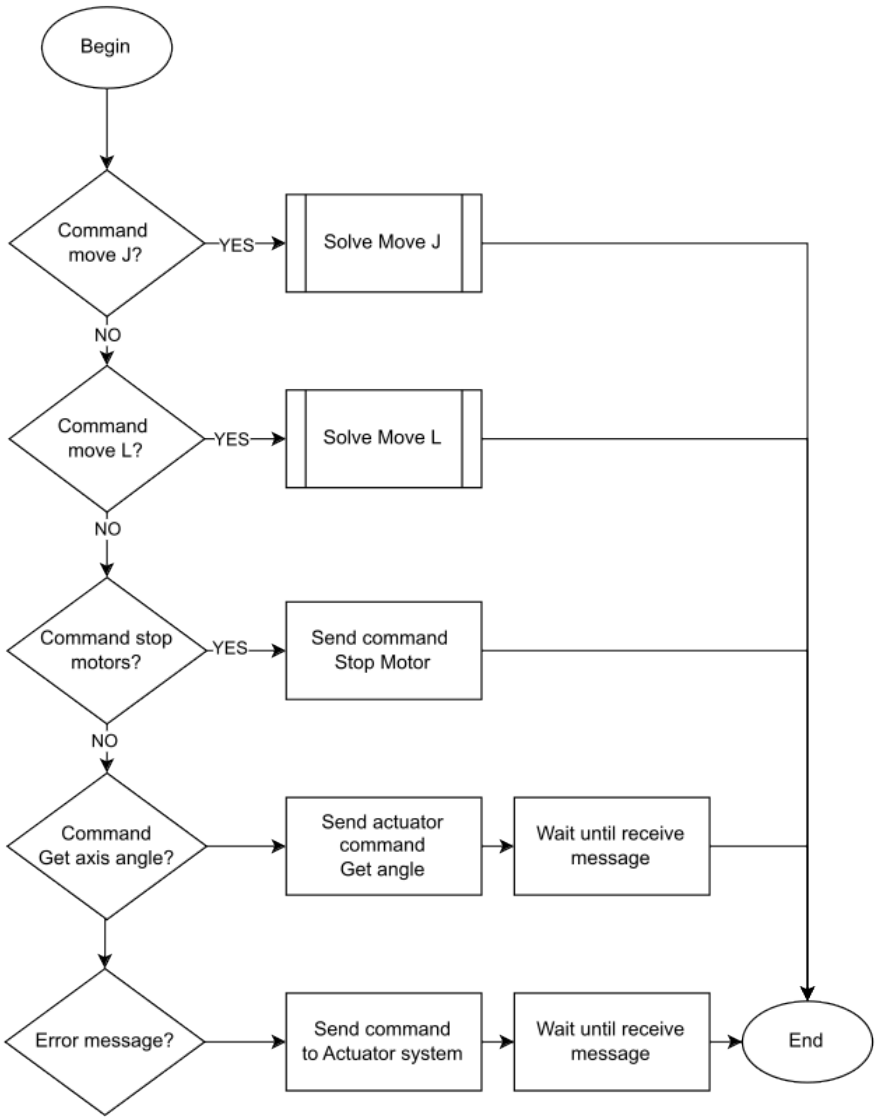

Figure 2.5. "Parse command" algorithm

The calculation includes a set of algorithms for linear (Move L) and axial (Move J) movements. In case of receiving commands related to the robot's actions, such as data on rotation angles or torque of each axis, the system sends a command to poll the ACD devices. The TCD device waits for confirmation of the sent command (ACK command). After that, the "Parse command" algorithm application ends, and the transition to "Make calculation" occurs.

The main subroutine for calculating the motion trajectory is the "Move J" command illustrated in Figure 2.6. The essence is to perform the movement to a specific position in the fastest way using angular movements. In the implementation of this function, accelerations for each robot axis are calculated, but only if the angles of the planned point are within permissible values.

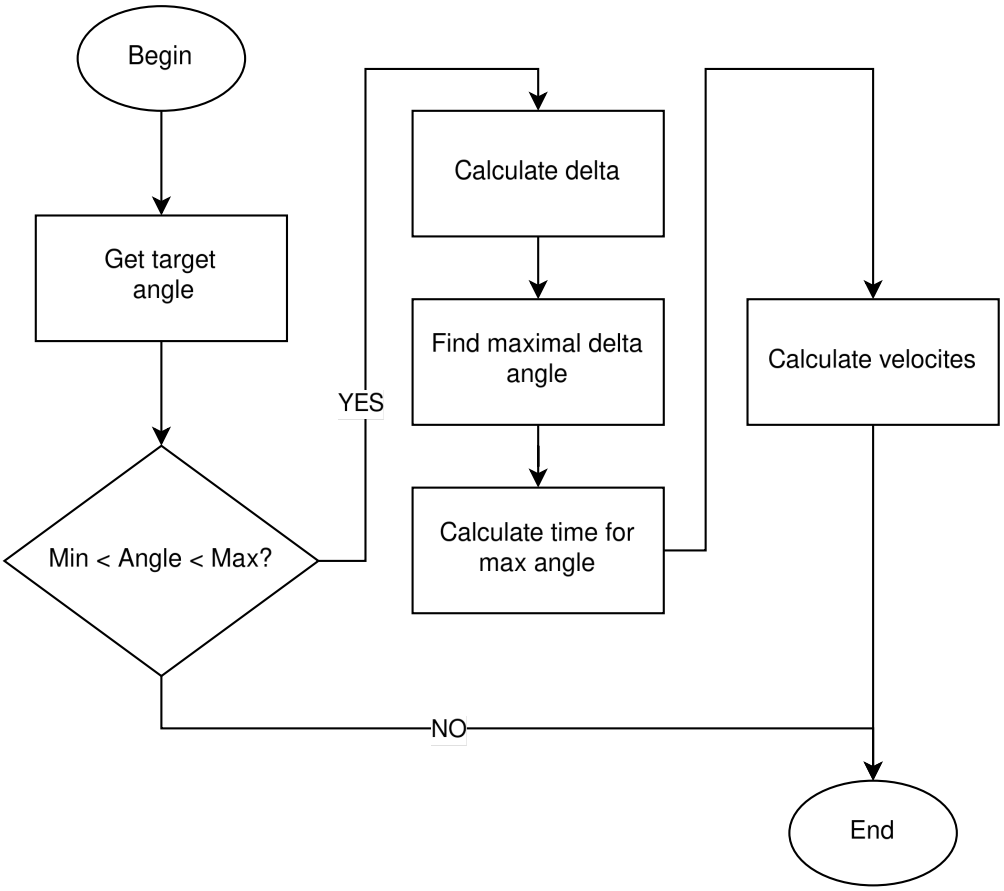

Figure 2.6. "Move J" subroutine algorithm

The subroutine for calculating the motion trajectory is the "Move L" command (Figure 2.7), for the robot's arm. The set of algorithms solves the inverse problem, the result of which is several position options, rotation angles of the robot's links. Next, a list of robot position options that do not exceed the physical limits of each robot axis is formed. Also, the calculation of the smallest angle from the initial position of the robot is required.

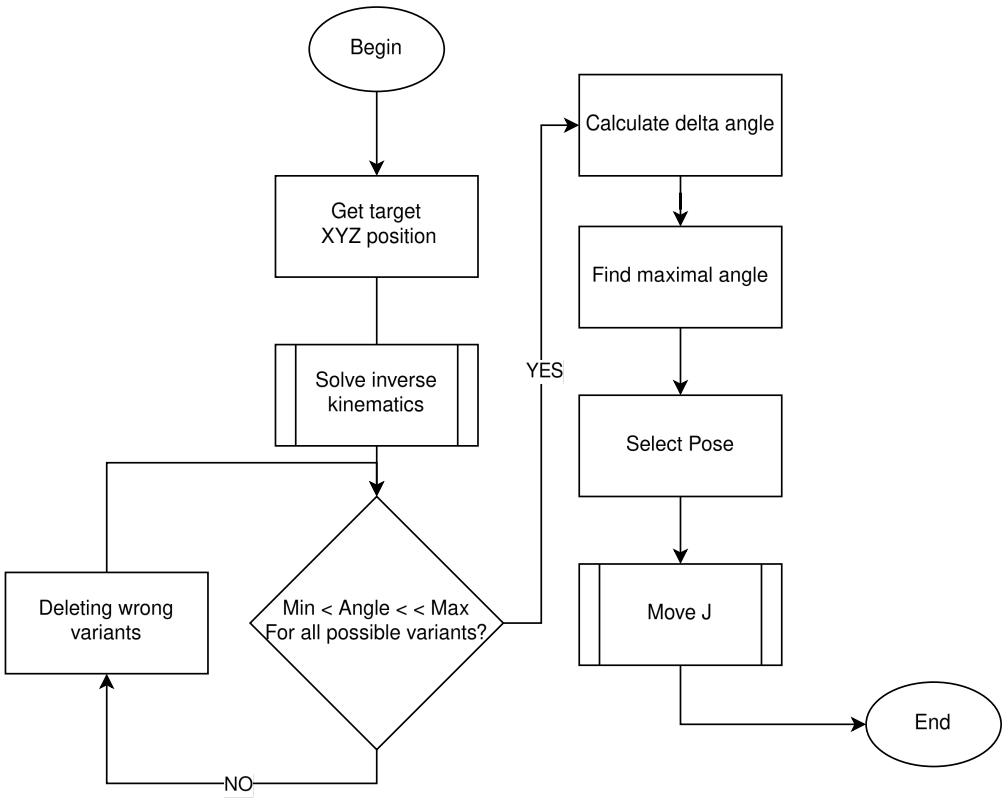

Figure 2.7. ”Move L” subroutine algorithm

2.5. Algorithm of the Actuator Control Device Operation

The algorithm for the Actuator Control Device is demonstrated in Figure 2.8.

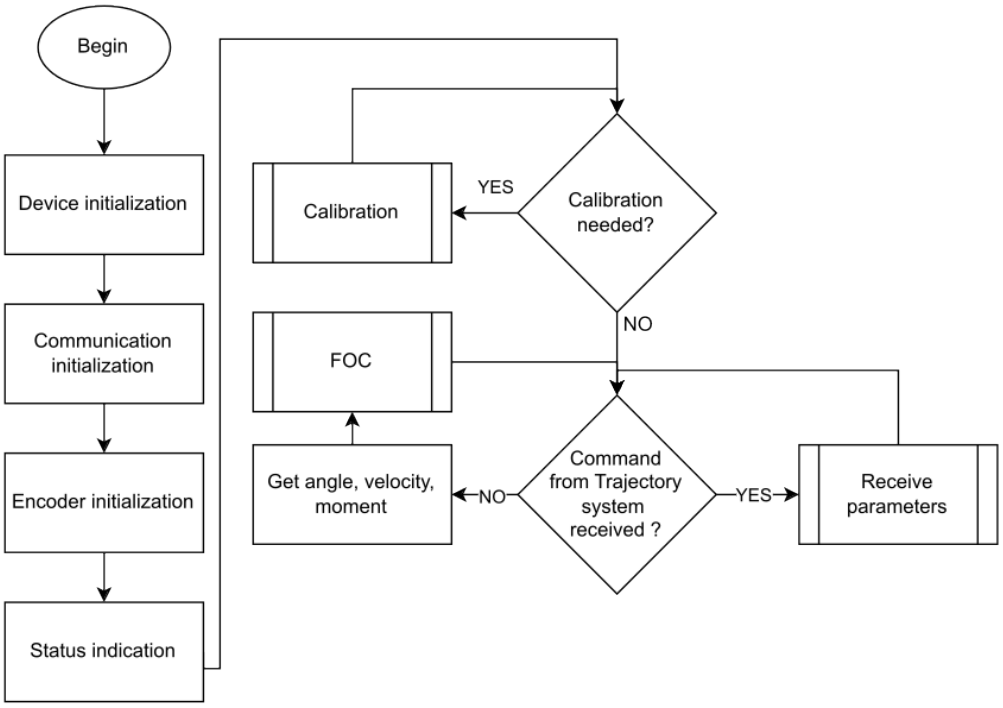

Figure 2.8. ”Actuator control device” operation algorithm

This algorithm is intended for controlling a synchronous type DC motor, responsible for rotating

the controlled robot axis to a specific angle, and monitoring its parameters. After turning on the system, the motion of the internal and external peripherals is initialized. If the initialization is successful and the connection is established, the need for motor calibration is checked. The execution of the calibration function depends on the set and values of the executed program parameters. Calibration is necessary if the robot's axis has been disassembled and the position of the angle measuring device has been changed relative to the original position of the axis.

Subsequently, encoder values are polled at a certain frequency, which are converted into motor axis rotation angles. These parameters are needed in the vector control calculation algorithm. Upon successful calculation of the vector control parameters, output signals are generated to the driver.

Figure 2.9 shows the message processing algorithm.

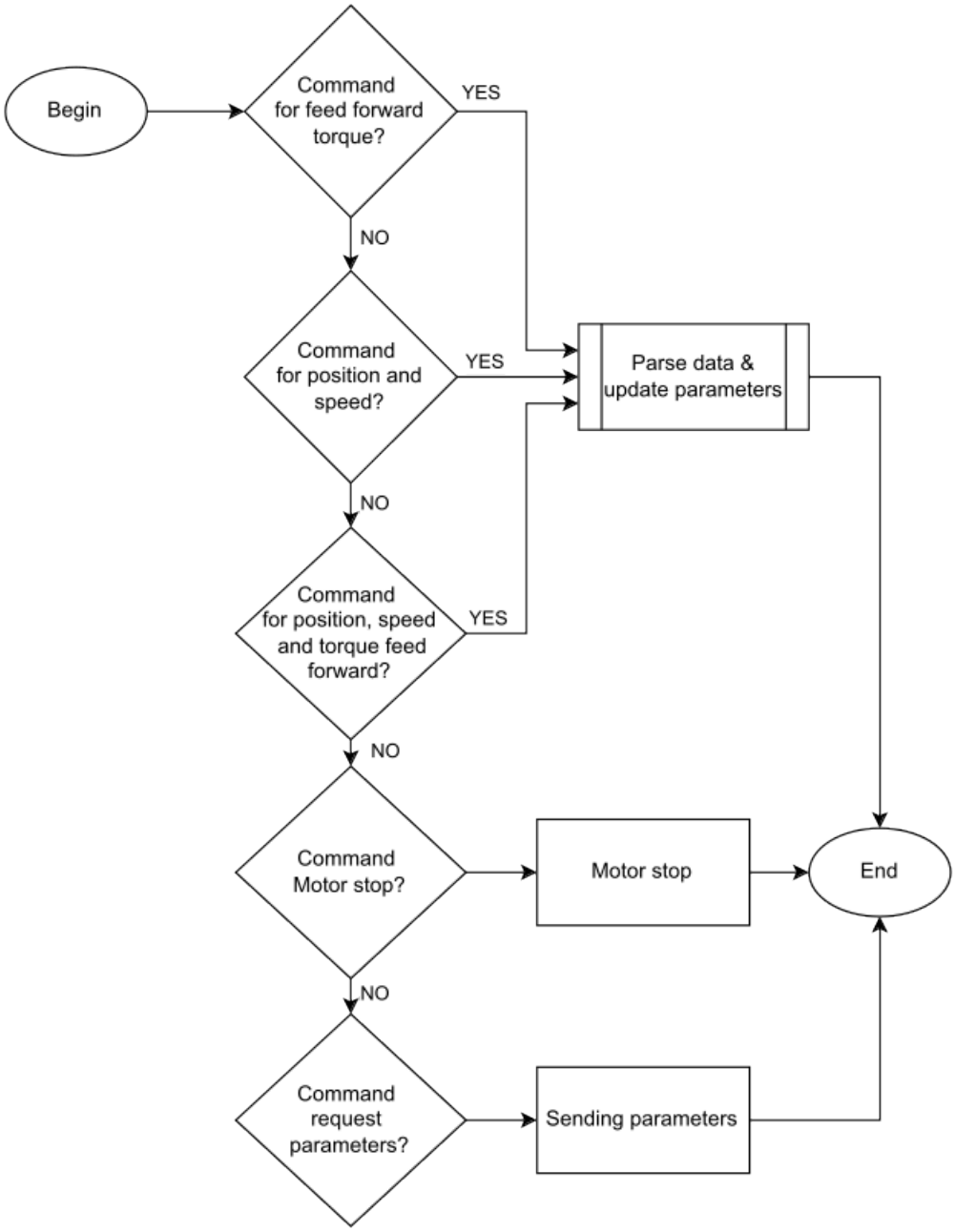

Figure 2.9. "Receive parameters" subroutine operation algorithm

If a message from the Tactical Control Device (TCD) has been received, the process of extracting data from the message, containing information about the required position, speed, and controller coefficients, is executed. These data are used to calculate the vector control parameters. In case of receiving a data request message, parameters are sent to the sending device.

3 FUNCTIONAL DIAGRAM OF THE CONTROL SYSTEM FOR A MINI ROBOT

The designed system is intended for performing manipulator robot control tasks. It is impractical to place Actuator Control Devices (ACD) in one location due to their number. Therefore, it is advisable to locate the actuator devices (ACD) near the electric motor to ensure timely and accurate response to control commands, as well as to reduce electromagnetic interference and power losses in the wires. Also, close placement facilitates maintenance simplification and enhances the reliability of the control system as a whole.

3.1. Gimbal motors

The gimbal motor in the robot is one of the components of the mechanical part of the manipulator axis. The motor transmits torque to the manipulator axis, which is used to move various parts of the robot, such as links and joints. In most cases, to increase the torque between the motor and the robot's axis, a gearbox (or transmission mechanism) is used. The gearbox reduces the rotation speed of the motor's output shaft while increasing the torque. Direct connection of the motor to the axis may not provide sufficient force or may be too fast for precise manipulations.

The load acting on each axis determines the required torque; therefore, a suitable motor and gearbox were selected for each axis. The dimensions of the motor were considered, as each subsequent link, starting from the first axis and ending with the last, must be lighter and more compact, due to the need to reduce the load on the motors and distribute the mass of the structure itself. Based on the technical requirements presented in Chapter 1, it is necessary to use motors of several models.

Synchronous electric motors must meet the following requirements:

- Internal resistance of approximately 10 Ohms, to reduce heating of the electric motor considering its dimensions;
- High torque at low speeds, Kv characteristic of less than 200 units;
- Operating voltage from 10 to 20 V;
- Low weight with a force moment of more than 1.0 [kgf·cm].

As a result of analyzing models with various parameters, it was decided to use electric motors from gimbal camera joints (Gimbal) in the robot's construction. Models of this category of motors provide an optimal balance between torque and low rotation speed. Models GBM4008H-150T, GM5208-120T, and GM3506, which are shown in Figure 3.1, were chosen. A characteristic feature of these motor models is the number of magnet pole pairs and rotor coils - more than 20 (Table 3.1), as the motor's speed is inversely proportional to the number of rotor poles, a greater torque can be achieved at a low motor rotation speed. These motors have high torque with a relatively low weight of up to

200 grams. The advantage of these electric motors is a wide operating temperature range from -20°to +60°(simplefoc, 2023).

Figure 3.1. «BLDC Gimbal motor GBM4008H-150T»

Table 3.1

Technical parameters of GBM4008H-150T, GM5208-120T and GM3506 motors

Model	Diameter [mm]	Height [mm]	Weight [g]	Kv	Poles & Magnets	Resist. [ohms]	Torque [kgf-cm]	Max power [W]
GBM4008H -150T	46±0.05	21±0.2	107±0.5	68	24N22P	6.7	1.2	40
GM5208 -120T	63±0.05	22.7±0.2	195±0.5	68	24N22P	6.7	1.9	40
GM3506	40±0.05	17.8±0.2	64±0.5	40	24N22P	5.6	1	25

3.2. Encoder

In the design of the control system of the robot arm, the precise determination of the rotation angles of its parts plays an important role. It is also necessary to determine the position of the motor rotor for smoother control. Encoders are commonly used to determine the rotation angle.

One of the most crucial parts is determining the position of each axis link. Precision criteria have special requirements. Based on the technical requirements, it is necessary to ensure a repeatability of 0.02 mm at a distance of 350mm. To achieve this precision, it is required to ensure a minimum axis rotation angle of 0.008185°. The resolution of the absolute encoder should be at least 16 bits, but encoders with such resolution are just beginning to appear and are an expensive solution for this application. It is also important to consider that for vector control, it is necessary to determine the position of the motor rotor to ensure smooth motor startup. The best solution to both problems is to install an encoder on the motor rotor, and due to the gearbox, not only the torque will increase but

also the resolving ability of the rotation angle. To ensure acceptable accuracy, it is necessary to use a gearbox with a reduction ratio of at least 11 with a 12-bit encoder resolution. However, it is essential to consider the backlash in all mechanical mechanisms and install harmonic or cycloidal zero-backlash gearboxes (Sensinger and Lipsey, 2012).

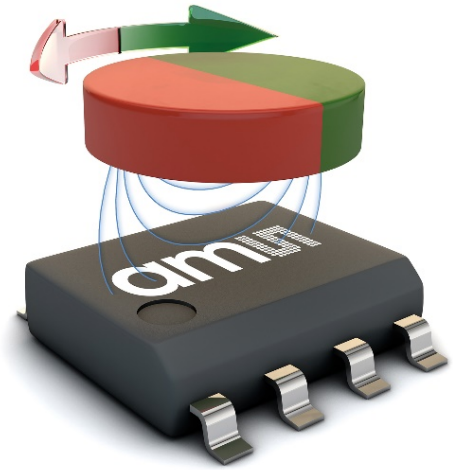

Figure 3.2. AS5600 axial position detection chip

The AS5600 chip (Fig. 3.2) with a digital output from ”AMS” was chosen as the encoder. The AS5600 magnetic position sensor is a contactless sensor for determining the absolute rotation angle, providing 12-bit accuracy (Table 3.2)in measuring angles, ranging from 0 to 360 degrees.

Table 3.2

AS5600 axial position chip parameter table

Parameters	**Value**
Resolution [bit]	12
Output	Analog out / PWM / I²C
Supply Voltage [V]	3-3,6
Temperature Range [°C]	-40 to +125
Package	SOIC-8
Max current [mA]	100
Sampling rate [µs]	150
Max RPM	20000
Position precision [°]	0.087

AS5600 combines reliability and ease of integration through the use of the I²C digital data transfer interface , making it an excellent choice for continuous and precise monitoring of the motor axis angular position in a robot (ams, 2023).

3.3. Strategic Control Device

The Strategic Control Device is essential for solving motion tasks and serves as an interface between the Tactical Control Device and the Intelligent Level Device, thereby providing control over the parameters and the robot itself. The Strategic Control Device must meet the following requirements:

- Multitasking capability;
- Compact size;
- Availability of GPIO;
- High computational power;
- Ability to use libraries and functions of the operating system;
- Low power supply voltage;
- Support for wireless communication technologies, IEEE 802.11 or 802.15 specifications.

These requirements are met by the single-board mini-computer ”Raspberry Pi Zero W 2” (Figure 3.3) from the ”Raspberry Pi Foundation”. The high computational power of the board (Table 3.3) and the advantages of the ”Linux” operating system facilitate the development and execution of complex algorithms and control tasks on the device itself. Moreover, support for a wide range of libraries and access to the 40-pin GPIO connector from the operating system significantly simplifies the development of complex control programs and integration with other systems. Unlike previous ”Raspberry Pi Zero” developments, the ”Pi Zero 2” modification has a more powerful quad-core ”ARM Cortex-A53” processor and a redesigned antenna for better wireless information transmission.

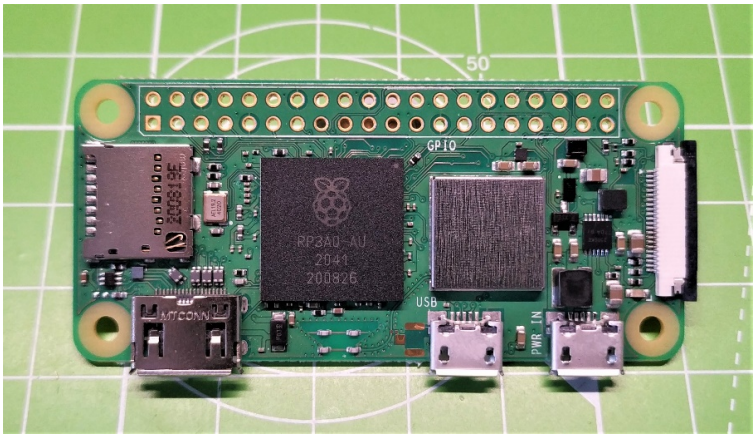

Figure 3.3. Single-board minicomputer «Raspberry Pi Zero W 2»

Table 3.3

Table of parameters of the Raspberry Pi Zero W 2 single-board mini-computer

Parameters	**Value**
Processor	Broadcom BCM2710A1, Quad-core Cortex-A53 (ARMv8) 64-bit SoC @ 1GHz
RAM	512MB LPDDR2
Wireless Connectivity	802.11 b/g/n wireless LAN, Bluetooth 4.2, BLE
Ports	Mini HDMI, USB On-The-Go port, Micro USB power
GPIO Pins	40-pin GPIO header
Power Requirement	5V/2.5A DC via micro USB connector or GPIO
Size	65mm x 30mm x 5mm

Based on the requirements to the developed strategic control device, taking into account its specifics, it was decided to use the operating system ”Raspberry Pi OS Lite”, which is based on the distribution ”Debian 12” 32-bit version. The main feature of this distribution is stability, security and wide package support.

3.4. Tactical and actutor control system controller

According to the requirements of the developed control system, the control device for tactical and executive devices must be compact; therefore, a microcontroller should be used due to its small size. The microcontroller’s tasks include reading and processing data from sensors, facilitating data exchange via the data bus, performing numerical calculations for vector control, solving direct and inverse kinematics problems, and generating PWM signals for the inverter. The minimum requirements for the microcontroller are:

- Operating power from 3V;
- More than 20 input/output lines;
- Ports for working with the CAN bus;
- Ports for working with PWM;
- Hardware support for the I2C protocol on microcontroller pins;
- Presence of an internal ADC;
- Presence of a Direct Memory Access (DMA) mechanism.

The main problem in choosing a microcontroller was the lack of information related to solving

the tasks of the tactical and executive control system. The issue was selecting the best microcontroller model for kinematic tasks. It was impossible to determine which microcontroller, among the most popular on the market, held a leading position. Microcontrollers vary significantly within a single family, and when considering models from different manufacturers, the architecture of each microcontroller is vastly different. It was also necessary to consider the toolchain, the programming tools set, which could also affect performance. Consequently, there was a need to test various types of general-purpose microcontrollers available on the market.

3.5. Testing microcontrollers

The conducted testing was necessary to find a suitable microcontroller model. Writing executable programs for each specific microcontroller is a difficult and lengthy process; it is necessary to consider the architecture of the microcontroller and optimize the algorithm for each family. Therefore, synthetic testing was conducted to choose the most suitable microcontroller.

The main selection criterion was the speed of performing specific mathematical operations. High execution speed is necessary to ensure efficient real-time system operation. This is especially important in vector control tasks, where delays in data processing lead to significant losses in accuracy and responsiveness of the entire system as a whole. Upon closer examination of tasks related to kinematics and vector control calculations (in particular, direct and inverse Clarke transformations formulas (3, 4, 5), inverse Park transformation formulas (1, 2), a large number of calculations were discovered (Huang et al., 2011):

- Multiplication and division of floating point numbers;
- Calculating trigonometric expressions, sine and cosine functions;

$$U_\alpha = U_q \sin(\theta) \tag{1}$$

$$U_\beta = U_q \cos(\theta) \tag{2}$$

- Square root calculation.

$$u_a = U_\alpha \tag{3}$$

$$u_b = \frac{-U_\alpha + \sqrt{3}U_\beta}{2} \tag{4}$$

$$u_c = \frac{-U_\alpha - \sqrt{3}U_\beta}{2} \tag{5}$$

The points mentioned above are complex for computations on the core of any microcontroller and require a large number of processor operations for their calculations. Calculating such expressions will take a significant amount of time, which will negatively affect the performance of the entire control system, as well as the operation of its individual parts (Pack and Barrett, 2008).

Microcontrollers from manufacturers such as ”STMicroelectronics”, ”Raspberry PI Foundation”, and ”Espressif Systems” were chosen for the test. The main comparison criterion was the measurement of the time spent on each type of calculation. During the testing, actions related to pausing the main loop of the executable program, such as interrupts, were not used.

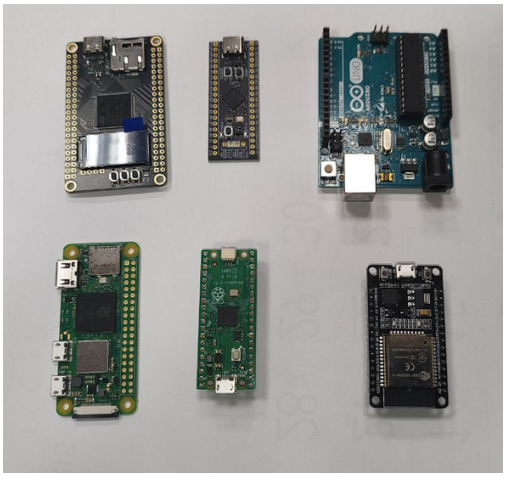

Figure 3.4. Boards that participated in the test

The 3.4 table lists the microcontrollers used, their clock speeds, and processor model. The boards that participated in the test are shown in the 3.4 figure.

Table 3.4

List of microcontrollers and their characteristics

Board	**MCU**	**Fcpu (Mhz)**	**CPU**
Arduino Uno (original)	ATmega328p	16	AVR
NodeMCU-ESP32	ESP32	240	Dual Xtensa LX6
NodeMcu V3	ESP8266	160	Xtensa L106
Raspberry Pi Pico	RP2040 [Arduino IDE]	133	Dual Cortex M0+
Raspberry Pi Pico	RP2040 [C++ SDK]	133	Dual Cortex M0+
Raspberry Pi Pico	RP2040 [MicroPython]	133	Dual Cortex M0+
STM32 Nucleo-32	STM32G431KB	170	Cortex M4
STM32 Nucleo-32	STM32G431 [FPU-OFF]	133	Cortex M4, **FPU-OFF**

The essence of the program in the microcontroller was to calculate the time spent using the GPIO port. Before performing the operation, all interrupts are disabled, the microcontroller sets the signal to a high state on the output, the mathematical operation is performed, the microcontroller pin is set to a low state, and the interrupts are restored. Then, using the HMO2024 oscilloscope from the manufacturer ”ROHDE & SCHWARZ”, the pulse duration was measured, and the data were recorded in Table 3.5 (the smaller the value, the better). It is worth noting that the calculations took into account the switching time of the GPIO outputs, recorded in the 2nd column of Table 3.5, and the results are shown in Graph 3.5.

Table 3.5

Table of measured time to calculate mathematical operations

MCU	Time IO [µs]	a + b [µs]	a - b [µs]	a * b [µs]	a / b [µs]	sin(a) [µs]	log(a) [µs]	sqrt(b) [µs]	pow(b, a) [µs]
ATmega328p	0,144	9,134	8,818	10,141	31,154	104,655	155,218	31,133	328,810
ESP32	0,312	0,402	0,389	0,399	0,567	0,798	1,357	0,691	3,120
ESP8266	0,508	0,942	0,959	1,214	2,296	13,393	26,611	8,454	76,808
RP2040 [Arduino IDE]	0,514	1,477	1,555	1,884	4,233	17,052	28,012	4,090	62,245
RP2040[C++ SDK]	0,008	0,750	0,783	0,650	0,819	4,634	6,610	0,709	17,288
RP2040 [MicroPython]	6,456	18,412	17,086	16,858	17,499	23,491	25,253	19,688	41,450
STM32G431	0,005	0,063	0,065	0,056	0,140	0,449	0,957	0,136	5,266
STM32G431 [FPU-OFF]	0,007	0,508	0,529	0,350	1,217	5,414	12,395	2,566	42,773

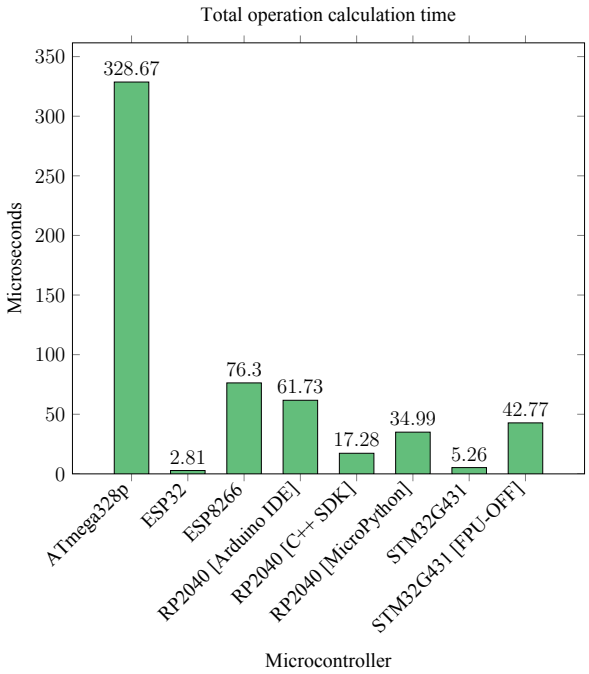

Figure 3.5. Graph comparing the calculation time of mathematical operations at different MCs

The provided data did not give clear information about the advantages of a specific controller in specific calculations. For further and more convenient analysis, the table was recalculated into the values of thousands of operations per second, kOPS, in Table 3.6, using formula (6).

$$kOPS = \frac{1000}{T_{cal} - T_{io}} \tag{6}$$

Table 3.6

Table comparing thousands of operations per second to calculate mathematical operations of each MCs

MCU	a + b	a - b	a * b	a / b	sin(a)	log(a)	sqrt(b)	pow(b, a)
ATmega328p	111,23	115,29	100,02	32,25	9,57	6,45	32,27	3,04
ESP32	11 090,82	12 961,74	11 549,13	3 918,10	2 056,26	956,62	2 641,39	356,12
ESP8266	2 305,87	2 217,75	1 416,38	559,44	77,61	38,31	125,85	13,11
RP2040 [Arduino IDE]	1 039,35	961,04	730,02	268,92	60,47	36,37	279,68	16,20
RP2040 [C++ SDK]	1 347,59	1 289,95	1 558,30	1 232,40	216,19	151,47	1 425,57	57,87
RP2040 [MicroPython]	83,64	94,07	96,13	90,56	58,70	53,20	75,57	28,58
STM32G431	17 179,17	16 702,61	19 476,01	7 383,89	2 253,15	1 050,29	7 624,36	190,09
STM32G431 [FPU-OFF]	1 996,11	1 918,50	2 917,02	826,69	184,94	80,73	390,85	23,38

The cumulative results of the recalculation of time into the values of thousands of operations per second (kOPS) are presented in Table 3.6 (higher values are better).

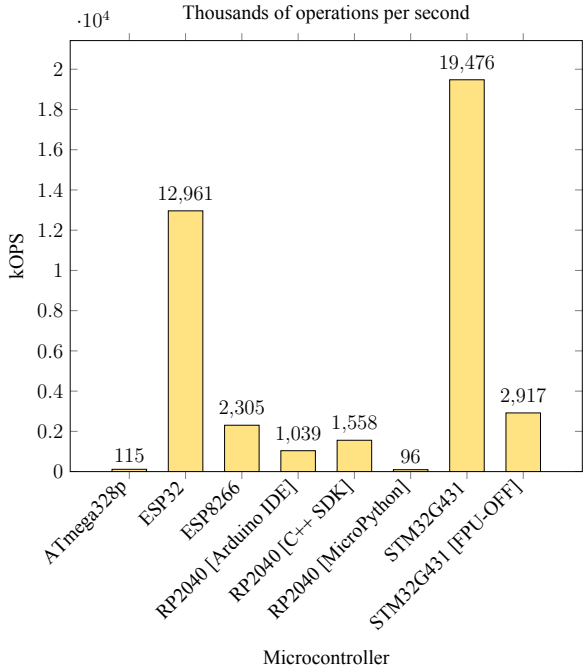

Figure 3.6. Graph of maximum values of thousands of operations per second for different MC

Figure 3.6 shows the graph of maximum values of thousands of operations per second for each microcontroller.

In the calculations provided, the microcontrollers performed operations at the maximum possible clock frequency Fcpu, but each microcontroller has a different maximum clock frequency (Table 3.5). To assess the efficiency of microcontrollers, a calculation of thousands of operations per second per megahertz (7) was conducted. The graph is shown in Figure 3.5 (higher values are better), and the table with calculations can be found in Appendix 1.

$$kOPS = \frac{K_{OPS}}{Mhz} \tag{7}$$

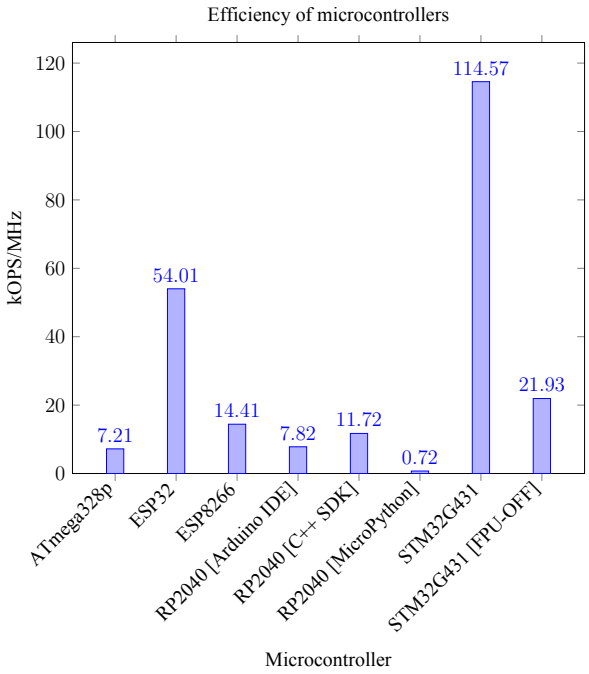

Figure 3.7. Graph of the values of thousands of operations per second per 1 megahertz of each MCs

As a result of the tests conducted, several interesting points related to microcontrollers were discovered. As can be seen in Figure 3.4, one RP2040 processor with different sets of development tools yields different results. The test favorites are the ESP32 microcontroller with a Dual Xtensa LX6 processor from the Chinese company ”Espressif Systems” and the STM32G431 microcontroller from ”STMicroelectronics” with a Cortex M4 processor. The STM chip was more efficient than the ESP32, although the clock frequency of the ESP32 is almost twice as high as that of the STM, but there is no significant increase in mathematical operations. It is worth noting that STM came out on top in terms of time, due to the presence of hardware floating-point operation units (FPU) in the processor.

Additionally, the presence of the ”CORDIC” co-processor for hardware acceleration of mathematical operations. As a result, the STM32G431 chip was chosen as the controller for the control

system, and Table 3.7 presents the main characteristics of this chip (STM, 2023).

Table 3.7

Table of main characteristics of STM32G431 microcontroller

Parameter	**Specification**
Core	ARM Cortex-M4
Operating Frequency	Up to 170 MHz
Flash Memory	Up to 512 KB
RAM	Up to 128 KB
Digital I/Os	Up to 82
Timers	Multiple, including general purpose and advanced
DAC	Yes, up to 2 channels
ADC	Yes, up to 16-bit
Interfaces	I2C, SPI, UART, USB, and others
Operating Voltage	2.0 V to 3.6 V
Temperature Range	-40°C to +85°C
CAN Bus	Yes, CAN 2.0 (A, B)
Operational Amplifier (Op-amp)	Yes

3.6. Power Transistors

Power transistor switches form a three-phase inverter for controlling the electric motor. In this case, the transistor switches are used to amplify the PWM signals coming from the MCU. The type of transistor is N-channel MOSFETs with an induced channel. Such transistors have very low channel resistance (Rds(on)). This means lower power losses when controlling the load, and the component will heat up less, which allows choosing models with smaller dimensions for the same parameter values. The power switches must meet the following requirements:

- Operating output voltage is more than 24V;
- Operating temperature range -30 to +80;
- Transistor type is MOSFET;
- Channel type – n;
- Small element size;

- Type of installation is surface mounting.

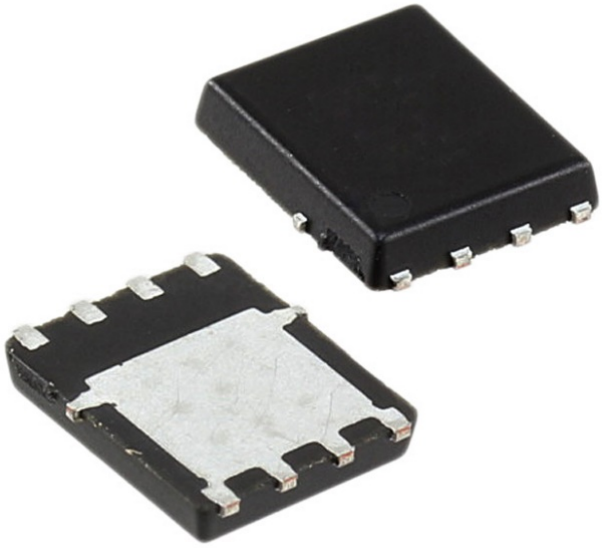

Figure 3.8. Transistor «SIR680DP»

Table 3.8

Table of main characteristics of transistor SIR680DP

Parameter	**Specification**
Type	N-Channel
Drain-Source Voltage (VDS)	80 V
Gate-Source Voltage (VGS)	± 20 V
Continuous Drain Current (ID)	100 A (at TC = 25 °C)
Pulsed Drain Current (IDM)	200 A (for 100 μs)
Power Dissipation (PD)	104 W (at TC = 25 °C)
Operating Temperature Range	-55°C to +150°C
RDS(on) max. at VGS = 10 V	0.0024 Ω
Configuration	Single
Package	PowerPAK SO-8

SIR680DP transistors (Figure 3.8) from ”Vishay Siliconix” were chosen, SIR680DP - 80V/100A n-channel MOSFET. They feature high current density and low open channel resistance (RDS(on)), around 2.4 mΩ, other parameters are in Table 3.8, which became the ideal characteristic of a suitable

element for use with Pulse Width Modulation (PWM). It is also important to note that the choice of components was dictated by the small size of the ”PowerPAK-SO-8” package.

3.7. Power Transistors Driver

To facilitate the control of the bridge transistor circuit of the inverter, drivers are used. Their task is to convert a low-power signal taken from the digital output of the MCU into a signal with a higher voltage and power level.

In this implementation of the executive system, general-purpose drivers L6385ED (Figure 3.9) produced by ”STMicroelectronics” were used. This chip is a half-bridge driver. One driver can control 2 power transistors of the upper and lower levels. The technical parameters of L6385ED are presented in Table 3.9.

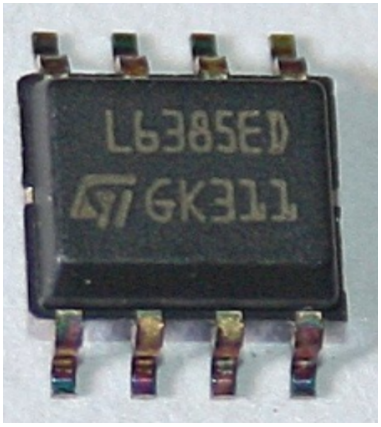

Figure 3.9. Driver L6385ED

Table 3.9

Table of L6385ED driver main characteristic

Parameter	**Specification**
Maximum Operating Voltage	580V
Supply Voltage	±50V/nsec (across full temperature range)
Driver Current Capability (Sourcing)	400mA
Switching Time (rise/fall)	50/30 nsec
Level of Input Control Signals	CMOS/TTL Schmitt trigger with hysteresis and pull down
Under-voltage Lockout	On both lower and upper sections
Bootstrap Diode	Internal

3.8. Data bus transmitter

To ensure data transfer between the executive control devices, strategic control, and other devices, it is advisable to use a data bus to reduce the number of connecting conductors. The most suitable data transfer protocol is the CAN bus, which is an industrial standard for industrial networks and has been used multiple times for implementation in manipulator robots (Megalingam et al., 2021). The main reason for using the CAN bus is its data transfer feature, where data is available to all devices connected to the network. Data transfer to one or several devices simultaneously is possible. Based on this, a synchronization option for all executive devices necessary for timely motion initiation was implemented (stm, 2020*b*). Based on the selected microcontroller model in Chapter 3.5 and technical data from Table 3.10, the hardware support by the microcontroller for the second-generation CAN bus - CAN FD, was identified.

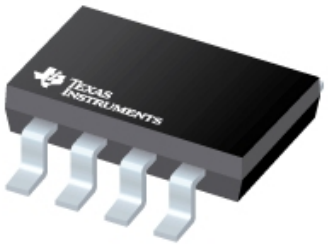

Figure 3.10. Transceiver TCAN1462DRQ1

Table 3.10

Table of main characteristics of the TCAN1462DRQ1 transmitter

Parameter	**Specification**
Package	SON-8 (3x3mm)
Operating Temperature Range	-40°C to +125°C
Supply Voltage Range	4.5V to 5.5V
Standby Current	Typically 5µA
Data Rate	Up to 5 Mbps
ISO 11898-2 Compliance	Yes

CAN-FD is the next stage in the development of the classical CAN bus, which provides a higher data transmission rate and a larger amount of transmitted data in one frame, this is possible due to the feature of transmitting field data (CAN data) at a speed multiple of the header transmission rate, which can be up to 8 Mbit/sec.

In order not to limit the possible speed of data transmission on the CAN bus, the TCAN1462DRQ1 transmitter (shown in the figure 3.10) was selected. The transmitter performs signal amplification, line protection in case of CAN bus damage, and adjusts the speed of their transmission.

3.9. Power converter

Due to the high consumption of the mini-computer, the consumption reaches up to 2.5A. According to the manufacturer's recommendation it is necessary to use a separate DC to DC converter. The power supply must meet the following requirements:

- Input voltage 24V;
- DC-DC Converter;
- Output voltage 5V,12V;
- High output current of 3A or more.

The AP64502QSP DC-DC step-down converter chip was selected, shown in Fig. **??**. The advantage of a pulse converter in high efficiency, which does not require heat dissipation, allows for compact voltage and current conversion solutions. The main advantage of this choice is the ability to convert a high current of 5A (Table **??**), without the use of external switches and the smaller number of (DIODES, 2021) elements required to operate this chip.

Table 3.11

Table of main characteristics of the AP64502QSP DC-DC converter

Parameter	**Specification**
Input Voltage Range (VIN)	3.8V to 40V
Continuous Output Current	5A
Programmable Switching Frequency	100kHz to 2.2MHz
Efficiency at Light Load	Up to 85%
Gate Driver Design	Proprietary, for EMI Reduction
Frequency Spread Spectrum (FSS)	Yes, for EMI Reduction
Low-Dropout (LDO) Mode	Yes
Precision Enable Threshold	For UVLO Adjustment
Protection Features	UVLO, OVP, Peak Current Limit, Thermal Shutdown

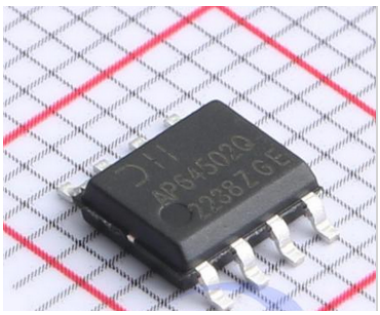

Figure 3.11. DC to DC chip AP64502QSP

3.10. Functional diagram of the mini robot control system

The functional diagram of the mini robot arm control system is shown in Fig. 3.12. The main controlling element of the tactical control system is the STM32G431 controlling microcontroller. All the necessary elements are connected to this microcontroller.

Connecting Raspberry Pi Zero W2 (as a strategic device) to the main control element STM32G431 is possible through two interfaces UART, the transfer of information in text form and through SPI, to transfer serialised data of large volumes at high speed. The Raspberry Pi device uses 5 I/O lines:

- PA7 (SPI1_MOSI) →GPIO 12 (RPI_MOSI);
- PA6 (SPI1 _MISO) →GPIO 13 (RPI_MISO);
- PA5 (SPI1 _SCLK) →GPIO 14 (RPI_SCLK);
- PA3 (UART2_RX) →RPI (RPI_TX);
- PA2 (UART2_TX) →RPI (RPI_RX).

In order to communicate between the devices of the executive system, a CAN bus transmitter (TCAN1462) is used and the transmitter uses the 2 I/O lines of port B of the STN32G431 microcontroller:

- PB9 (FDCAN_TX) →CAN Transceiver (CAN_TX);
- PB8 (FDCAN_RX) →CAN Transceiver (CAN_RX);

For external communication, an RS485 interface transmitter is used, which is connected to the 2 I/O lines of port A.

- PA10 (UART2_TX) →RS485 (UART_TX);
- PA9 (UART2_RX) →RS485 (UART_RX);

The robot status is indicated via the display panel and consists of three LEDs of different colours and occupies 3 I/O lines of port B.

- PB0 ←Indication Panel (LED_YEL);
- PB1 ←Indication Panel (LED_RED);
- PB2 ←Indication Panel (LED_GRN);

The remaining free I/O lines of the B ports are allocated to digital input and output blocks:

- **Digital inputs:**
  - PB11 ←DI (IN_1);
  - PB12 ←DI (IN_2);
  - PB13 ←DI (IN_3);
  - PB14 ←DI (IN_4);
  - PB15 ←DI (IN_5);
  - PA8 ←DI (IN_6);
- **Digital Outputs:**
  - PB7 →DO (OUT_1);
  - PB6 →DO (OUT_2);
  - PB5 →DO (OUT_3);
  - PB4 →DO (OUT_4);
  - PB3 →DO (OUT_5);
  - PA15 →DO (OUT_6);

Connection of all parts of the system is made on the electronic board by soldering the elements. The following connectors will also be used:

- connectors for operating voltage – XT30;
- CAN bus connectors – JST 1.25 PH 2;
- The pins for the rest of the peripherals – PBS.

XT30 connectors provide a reliable connection and withstand significant electrical loads without loss or risk of fire due to poor contact or overheating.

JST connectors provide easy connection and disconnection, which is convenient for maintenance and system upgrades. They prevent accidental reverse connection, which can be critical to prevent damage to components during installation.

PBS male connectors are suitable for connecting components that do not require high currents and are reusable. Pin connectors provide good contact and ease of mounting on the PCB, which is important for rapid prototyping and system setup.

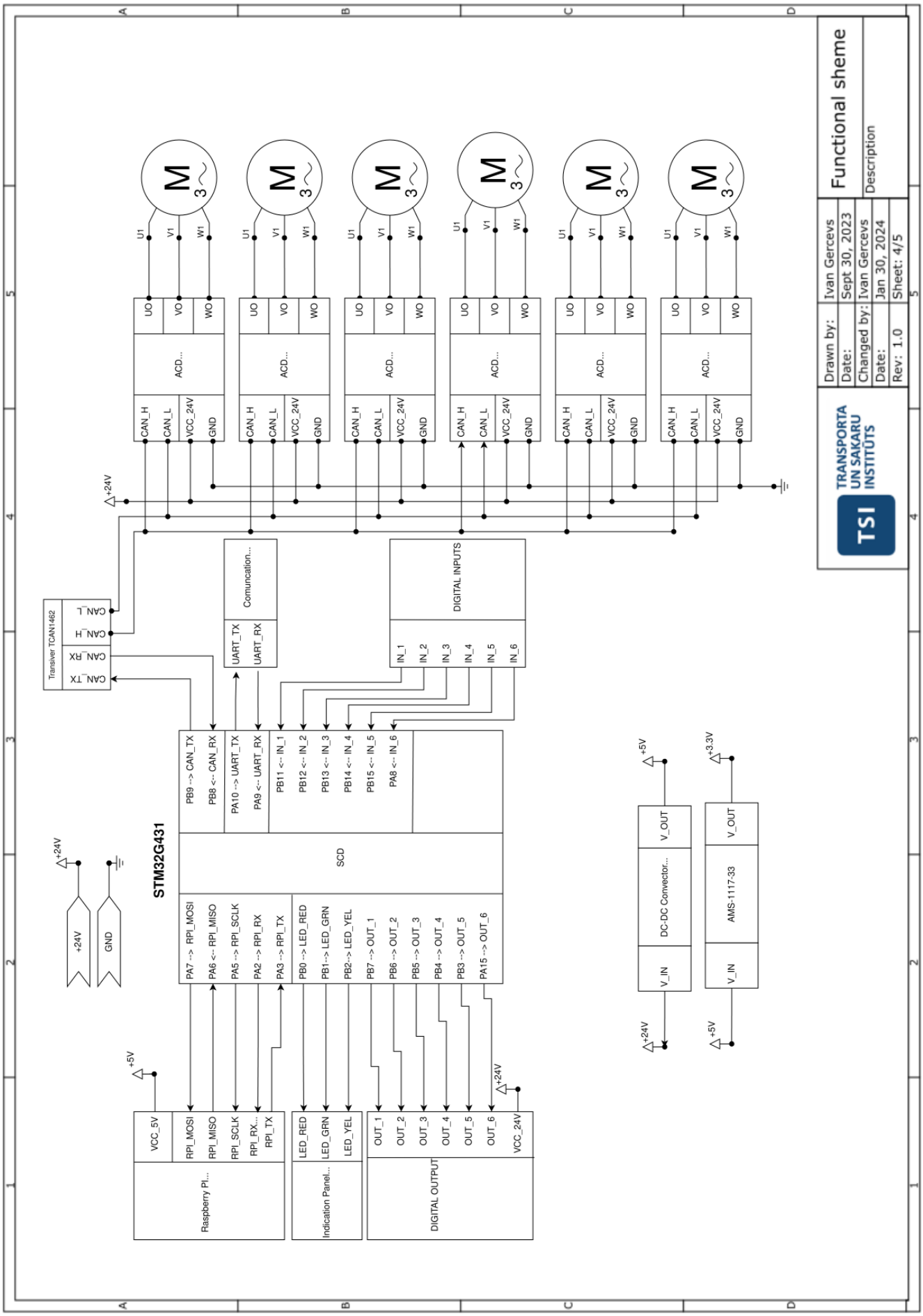

Figure 3.12. Functional diagram of the mini robot control system

3.11. Functional diagram of the actuator control system

The functional diagram is shown in the figure 3.13. The main control element of command processing and signal generation, executive control system, is the STM32G431 MCU. Generation of PWM signals is performed by the timer and output by 6 comparison channels, which are output on the lines of port A, C and B:

- PB15 →HIN1;
- PB10 →LIN1;
- PA12 →HIN2;
- PB9 →LIN2;
- PC13 →HIN3;
- PA8 →LIN3;

The drivers amplify the signal from the MCU and feed the power key transistors:

- HIN1 →HGV1;
- LIN1 →LGV1;
- HIN2 →HGV2;
- LIN2 →LGV2;
- HIN3 →HGV3;
- LIN3 →LGV3;

To measure the torque applied on the motor, there is a measurement of the current mila on each winding of the motor, the voltage drop is measured by feeding signals to the inputs of operational amplifiers which are inbuilt in the microcontroller 6 inputs:

- PA1 ←+VSHUNT_1;
- PA3 ←-VSHUNT_1;
- PA7 ←-VSHUNT_2;
- PA5 ←+VSHUNT_2;
- PB0 ←-VSHUNT_3;
- PB2 ←+VSHUNT_3;

In order to avoid breakage of the device, two input port lines are provided for connection to the ADC of the microcontroller, one is necessary for temperature measurement, the sensor located near the power transistors, so in case of overheating, the microcontroller takes action to set it to the error state. The second ADC is used to measure the incoming operating voltage, the MC goes into error mode. ADCs are located on the lines of port A and are connected:

- PA5 ←V_FED; • PA0 ←V_BUS;

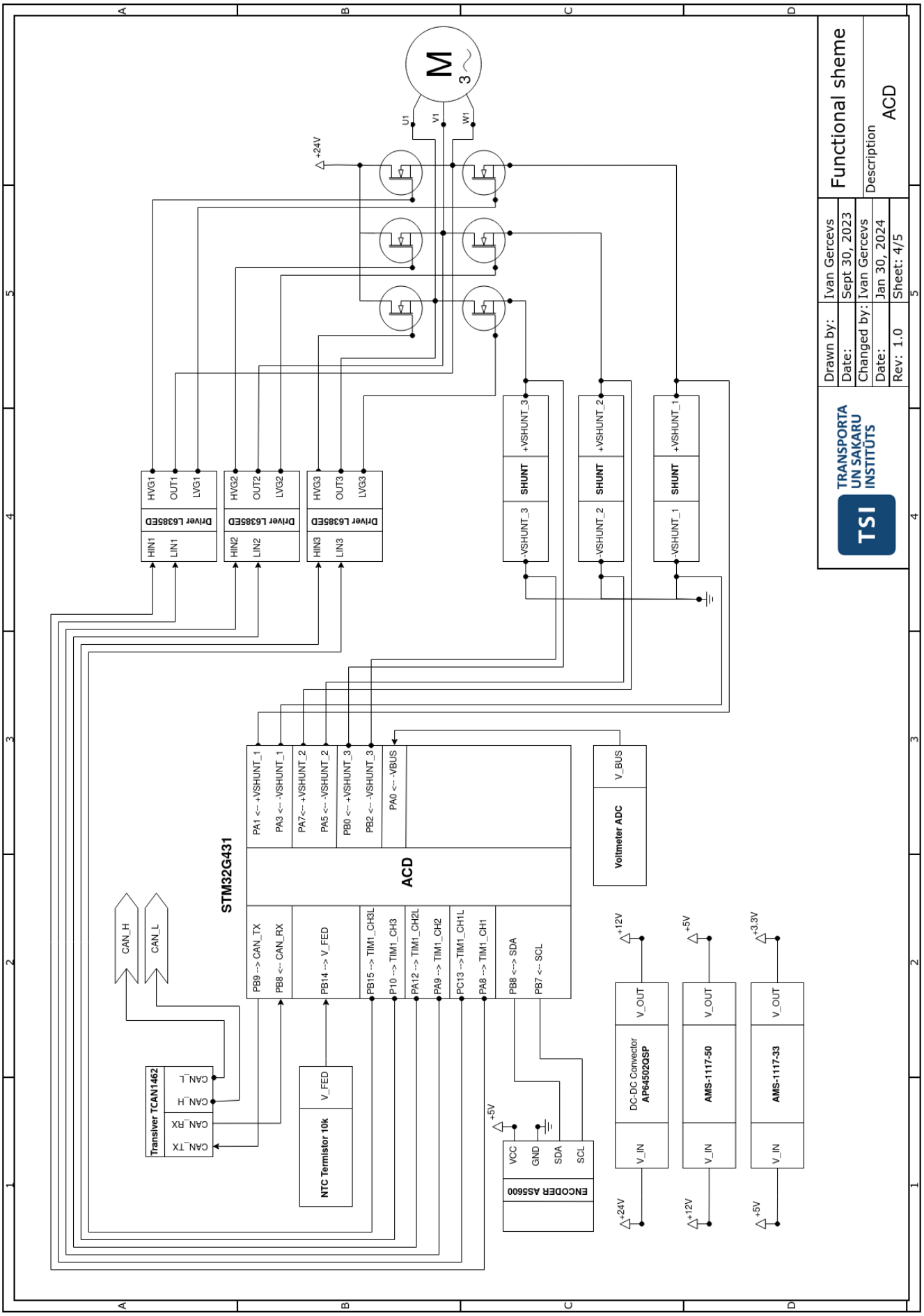

Figure 3.13. Functional diagram of the actuator control system

4 SHEMATIC DIAGRAM OF THE CONTROL SYSTEM FOR A MINI ROBOT

4.1. Schematic diagram of the actuator control device

The schematic diagram of the actuator control device for the mini manipulator robot is shown in Figure 4.1. Microcontroller U9 is the main computational component of the actuator control system. The STM32G431CBU6 chip is connected according to the standard scheme from the datasheet (STM, 2020*a*). The DC-DC converter is the main power source for the entire actuator control device, and the microcontroller works with an analog signal. To improve the stability of analog operational amplifiers and the accuracy of the ADC operation, a noise suppression filter on the analog power line VDDA and the reference voltage line VREF+ is implemented in the circuit, assembled according to the notes in AN5346 (STMicroelectronics, n.d.). The microcontroller is clocked using quartz resonators X1 and X2, which are connected according to the standard scheme with capacitors C23, C24 for X1 and C18, C19 for quartz resonator X2. Quartz resonator X1 serves as the main generator of the 8 MHz HSE clock signal for microcontroller U9. Quartz resonator X2 has a specific low-speed frequency LSE 32.768kHz, necessary for the real-time clock (RTC) operation, which is used for precise timekeeping without accumulating errors in the case of binary conversion. For programming the microcontroller, the programmer-debugger pins are led out to connector H2.

To monitor the state of the actuator device, two LEDs, LED1 and LED2, are provided. LED1 is connected through a current-limiting resistor to the power supply and lights up when power is applied, while LED2 lights up when needed from the MCU signal and serves as an indicator of the actuator device's error. An input voltage divider, consisting of resistors R16 and R18, is connected to the ADC input PA0 of the microcontroller; the resistance values are chosen to be more than 10 kOhm to prevent a large voltage drop across the resistors, with a maximum measurable voltage of 33 Volts. The second ADC input PB14 is connected to the outputs of the temperature measurement circuit, consisting of NTC resistor R15 and balancing resistance R20; capacitor C30 serves as a low-purity filter element with an upper boundary frequency of 1.5 kHz.

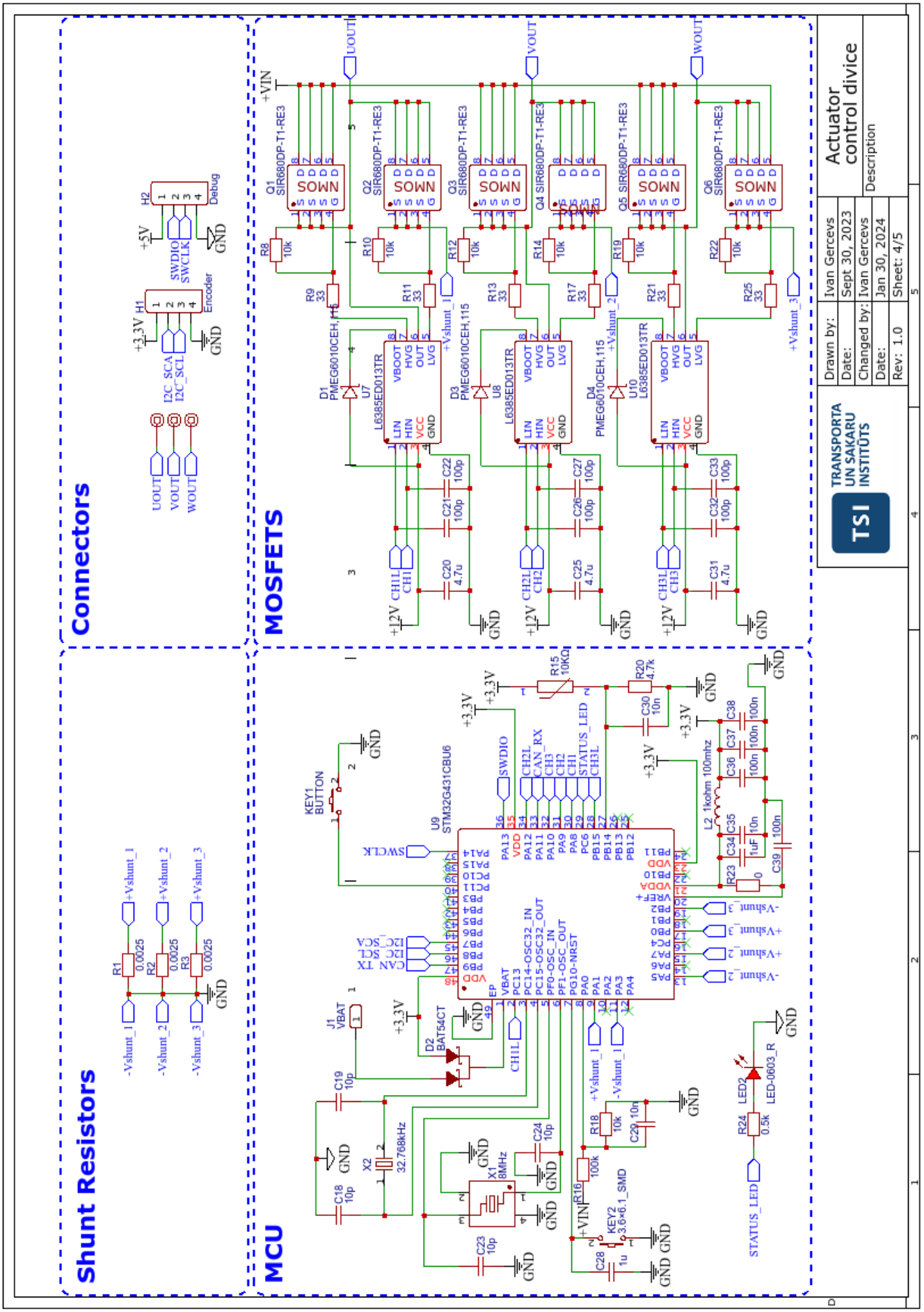

Figure 4.1. Schematic diagram of the actuator control device

Inverter drivers U7, U8, U10 are connected to the output of timer TIM1, located on the lines of port A and C. To suppress high-frequency interferences, capacitors C21, C22, C26, C27, C32, C33 with a value of 100 picofarads are connected to each of the inputs. Capacitors C20, C25, C31 on the power line are installed to prevent a sharp voltage drop when the internal power transistors are opened. Diodes D1, D3, D4 form a startup load circuit, necessary to provide power to the high-voltage section of the driver. The drivers control the power transistors from Q1 to Q6; the half-bridge driver controls the upper and lower level transistors for each motor phase. The transistors are connected through low-resistance resistors R9, R11, R13, R17, R21, R25 to eliminate ringing (parasitic oscillation) and through resistors R8, R10, R12, R14, R19, R22, which pull the gate to the source to avoid parasitic openings of the transistors.

For data bus communication, MCU lines PB9 and PA11 are connected to the CAN BUS transmitter U4, as shown in the schematic diagram 4.2. The transmitter does not require a standby mode, so the STB pin (pin 8 of the chip) is connected to the ground, and the transmitter operates from the moment it is turned on. A terminal resistor R7 is provided between the CANH and CANL lines, connected via jumper H3. For easy installation of executive control devices, connectors JP2 and JP1 form a parallel connection of two connectors, allowing the connection of devices without creating additional connections on the wires. The power connectors CN1 and CN2 are connected using the same principle.

This device implements several options for voltage and current conversion. The main converter is the DC-DC converter chip AP64502QSP, which is connected according to the standard scheme from the datasheet, as shown in the schematic diagram 4.2.

The chip converts the input voltage to the required 12V voltage for powering the power transistor drivers. To ensure voltage conversion, it was necessary to calculate the feedback divider at the FB (Feedback) line input, which is calculated (8):

$$R4 = R6 \cdot (\frac{V_{out}}{0.8V} - 1) \tag{8}$$

where R2=10 kOhm, for this variant the values are R1=140 kOhm. Capacitor C6 on the SS line of the DC-DC converter is a soft-start time setter. The standard variant 10nF was chosen, which is approximately equal to the soft start time - 4 ms(DIODES, 2021).

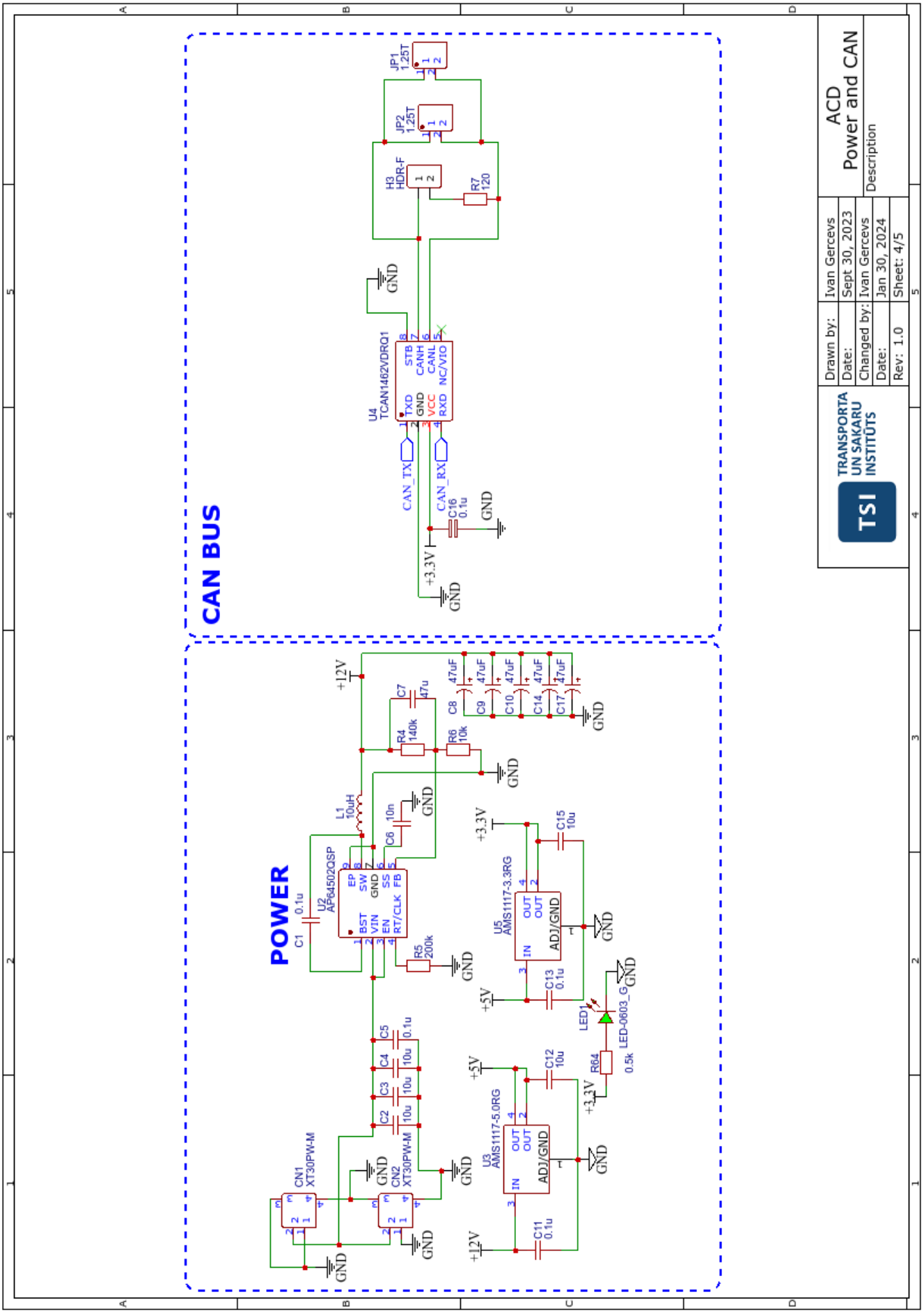

Figure 4.2. Schematic diagram of power supply elements and data bus of the actuator control system

The coil inductance is calculated for this inverter using the formula (9), where Delta I_l for this connection is calculated using the formula (10):

$$L = \frac{V_{out} \cdot (V_{in} - Vout)}{V_{in} \cdot \Delta I_l \cdot f_{sw}} \tag{9}$$

$$\Delta I_l = 0.5 \cdot 5A \tag{10}$$

The chip has a frequency control from 100kHz to 2.2MHz, the frequency is set by the resistor R5, the frequency is calculated by the formula (R5):

$$R5 = \frac{100000}{f_{sw}[kHz]} \tag{11}$$

For the conversion option, the closest in parameters inductance L = 10 μH (DIODES, 2021) was chosen. An additional row of smoothing tantalum capacitors C8, C9, C10, C14, C17 has been introduced into the standard converter scheme. Tantalum capacitors have a low internal resistance (ESR), which affects the output ripple (12):

$$V_{outRipple} = \Delta I_l \cdot (ESR + \frac{1}{8 \cdot f_{sw} \cdot C_{cout}}) \tag{12}$$

The total capacitance of the capacitors is $5 \times 47\mu F$.

Linear voltage converters U3 and U5 are used for voltage conversion for microcontrollers. The linear converters are connected to the 12V power line to reduce the heating of the converters themselves by reducing the voltage drop across them.

Torque measurement is carried out by measuring the current of the electric motor. The voltage drop across the shunts is measured, shunts R1, R2, and R3 have very low resistance to avoid the influence of the transmitted voltage and excessive heating. The shunt outputs are connected to the microcontroller inputs. After configuring the microcontroller, the shunt input lines are connected to the internal operational amplifiers and used in further conversions.

4.2. Schematic diagram of the tactical control device

The Shematic diagram of the tactical control system of the mini robot manipulator device is shown in Figure 4.3.

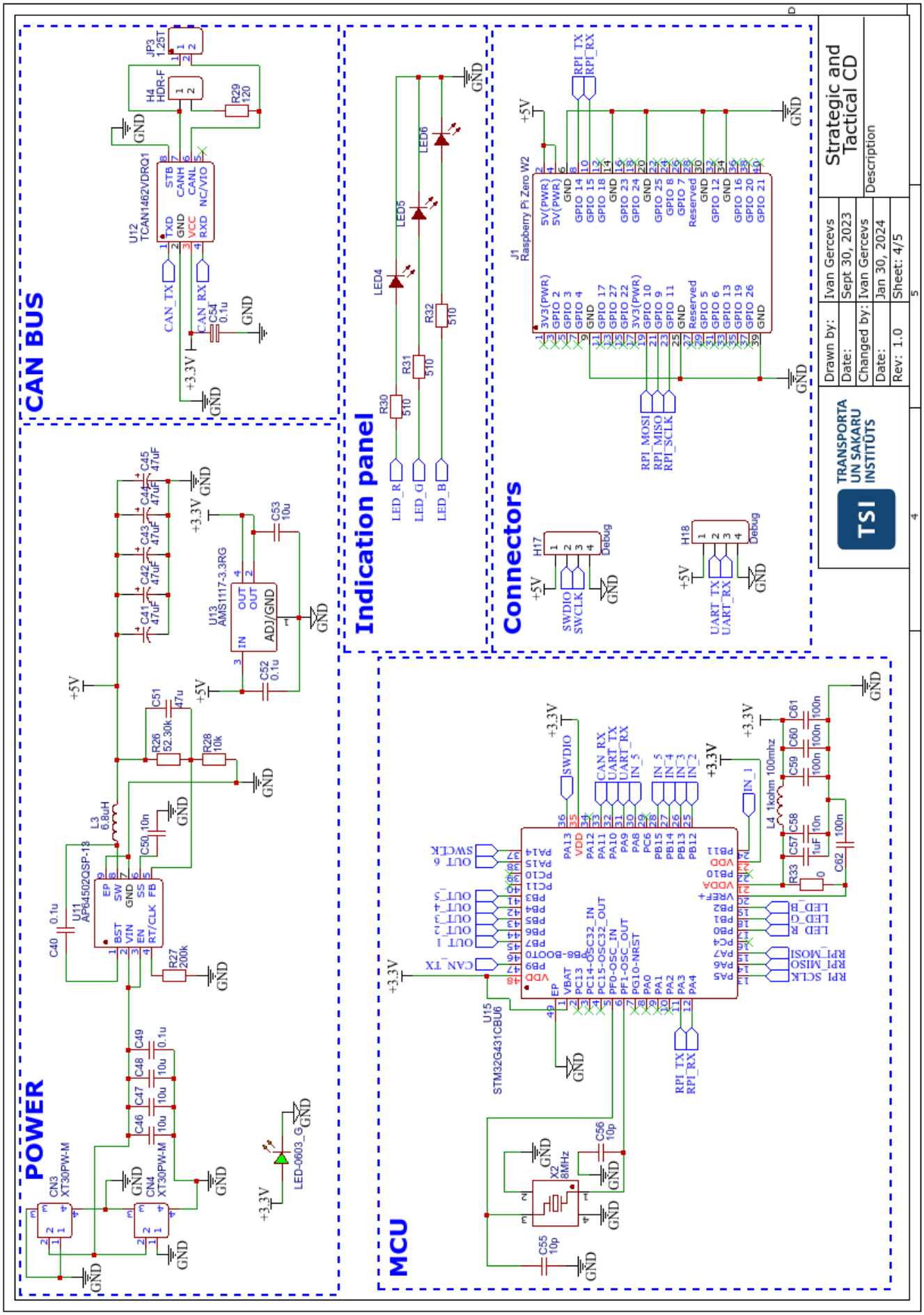

Figure 4.3. Schematic diagram of the tactical control device for minirobot manipulator

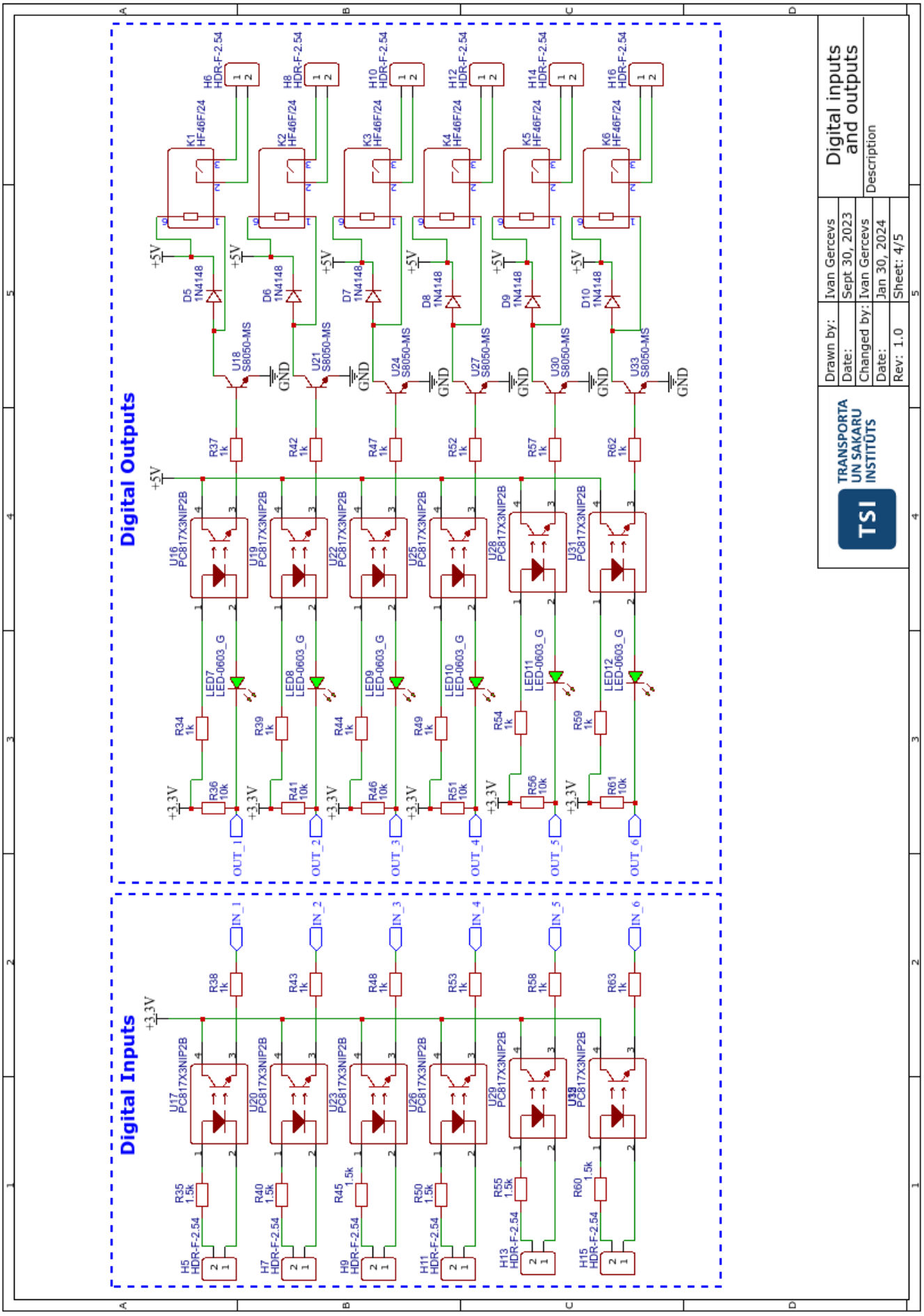

Figure 4.4. S Schematic diagram of the pin inputs for the control system of a mini-robot manipulator

The microcontroller U15 is a tactical control device assembled according to a standard scheme, as shown in the figure 4.1. The strategic control device connects through the J1 connector and utilizes two interface options: SPI and UART. The other pins of the J1 connector are not used, as they can connect to various expansion boards for the ”Raspberry PI ZERO 2”, which expand the functionality of the single-board computer.

The indicator panel consists of three LEDs of 3 colors: LED4, LED5, LED6, and they are connected to the microcontroller input. The other free inputs and outputs of the microcontroller are divided for controlling external signal inputs (6 pieces) and outputs (6 pieces) and are connected according to the standard scheme from the datasheet.

The inputs and outputs are isolated using optocouplers, shown in the figure 4.4, and the output signals are switched using relays K1, K2, K3, K4, K5, K6. The signals from the optocouplers U16, U19, U22, U25, U28, U31 are amplified by the transistors U18, U21, U24, U27, U30, U30. A resistor in the base circuit is necessary to limit the current flowing to the transistor’s base. Diodes D5, D6, D7, D8, D9, D10 are needed to shunt the relay coil terminals when the power is turned off. When switching off, a back EMF (electromotive force of self-induction of the coil) pulse is generated at the coil terminals, and this pulse can reach tens of volts and can lead to the failure of transistors U18, U21, U24, U27, U30, U30, which are not designed for such voltage.

The input digital signals are isolated using U17, U20, U23, U26, U29, U32. It is assumed that a logical ”1” corresponds to 24V voltage. To ensure the activation of the optocoupler, the resistors R35, R40, R45, R50, R60 were calculated with the condition that the current through the optodiode inside the optocoupler should be 20 mA at a voltage of 1.2 Volts, the maximum allowable voltage is 30 Volts, and the approximate resistor value was chosen from the E24 resistance value series and corresponds to 1.5 kOhm.

5 ALGORITHM AND SOFTWARE IMPLEMENTATION FOR COLLABORATIVE MINIROBOT MANIPULATOR CONTROL SYSTEM

5.1. Software implementation of the strategic management device

The strategic control device is a single-board minicomputer ”Rasp- berry Pi Zero W 2” (Figure 3.3), the system is started immediately after power supply to the device. The operating system ”Linux” starts and waits for the user to connect to the device. There are two possible ways of using the strategic device:

1. Connecting wirelessly, via SSH protocol.
2. Through a wired connection, by connecting the wires of the monitor and I/O devices.

To connect the minicomputer wirelessly, it is necessary that the microcomputer is in the local environment of the wireless network.

For further operation of the control, one of the high-level languages is used, for example, Python programming language with the use of libraries, an example is shown in the figure 5.1.

```
import numpy as np
from pybotics.geometry import vector_2_matrix
poses = np.array([
    vector_2_matrix([600, -150, 800,
                     np.deg2rad(-90), 0,
                     np.deg2rad(-90)]),
    vector_2_matrix([700, -150, 800,
                     np.deg2rad(-90), 0,
                     np.deg2rad(-90)])
])
start_end_joints = [robot.ik(p) for p in poses]
```

Figure 5.1. A fragment of using third-party libraries in Python.

Using libraries, it is possible to program a robot in simulation mode and only then generate commands for the tactical control controller. The use of a high-level language allows for the creation of complex movement strategies and the processing of sophisticated algorithms. For communication between strategic and tactical level devices, the use of a written API library is anticipated, an example of which is shown in Figure 5.2 for communication between the microcontroller and the microcomputer.

```
def send_movej_command(self, command):
    """
    Send a moveJ command to the robot through the serial connection.
    Args:
    serial_connection (serial.Serial): The serial connection to the robot.
    command (str): The moveJ command string.
    """
    with serial.Serial(self.serial_port, self.baud_rate, timeout=1) as ser:
    log("Serial connection established.")
    serial_connection.write(command.encode())
      log(f"Sent command: {command}")
    if self.ser.in_waiting:
  response = self.ser.read(ser.in_waiting).decode()
        log("Received response:", response)
```

Figure 5.2. A fragment of an API library function for use in Python.

Also another possible type of programming is the use of RoboDK software, a driver is executed on the strategic control device, which transmits data from the socket TCP/IP connection to Uart, the list of commands is considered in Table 5.1 on the tactical control controller fragment is presented in Appendix 2.

Table 5.1

List of basic commands that robotDK supports

Command	Description	Parameters	Example Usage
MOVJ	Moves the robot joints to specified positions.	Joint Positions (degrees or radians)	MOVJ 10 20 30 40 50 60 70
CJNT	Requests the current joint positions.	None	CJNT
MOVL	Moves the robot to a specified linear position.	Cartesian Coordinates (X, Y, Z, Rx, Ry, Rz)	MOVL 500 400 300 0 1.57 0
SPEED	Sets the movement speed of the robot.	Speed Value (units per minute)	SPEED 500
WAIT	Pauses the robot operation for a specified duration.	Time (seconds)	WAIT 5
STOP	Stops all robot movements immediately.	None	STOP
DISCONNECT	Terminates the connection with the robot.	None	DISCONNECT

5.2. Algorithms of the actuator control device

The main purpose of the actuator system is to smoothly control a 3-phase BLDC motor by reading and processing data from position and current sensors and generating PWM signals. Figure 5.3 shows the basic algorithm of the executive control system.

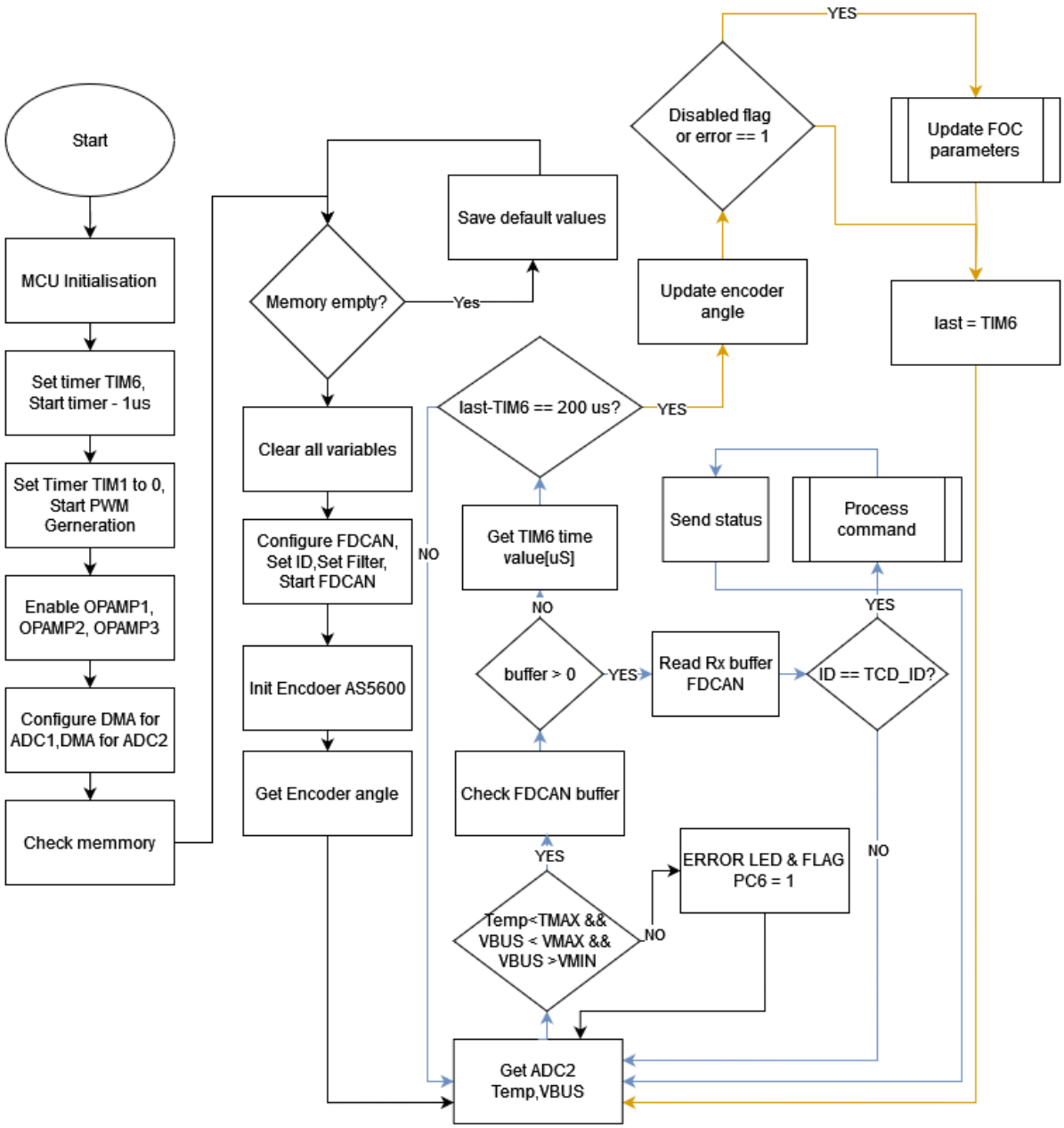

Figure 5.3. Algorithm of the executive system system.

Upon power-up, the microcontroller conducts a functionality check of all devices by reading data. The system is divided into two parts: the first part executes functions that do not require frequent execution (highlighted in yellow), and the second part operates at the highest possible frequency, tasks such as computing the vector algorithm and parsing messages are performed in this part, as shown in Figure 5.4.

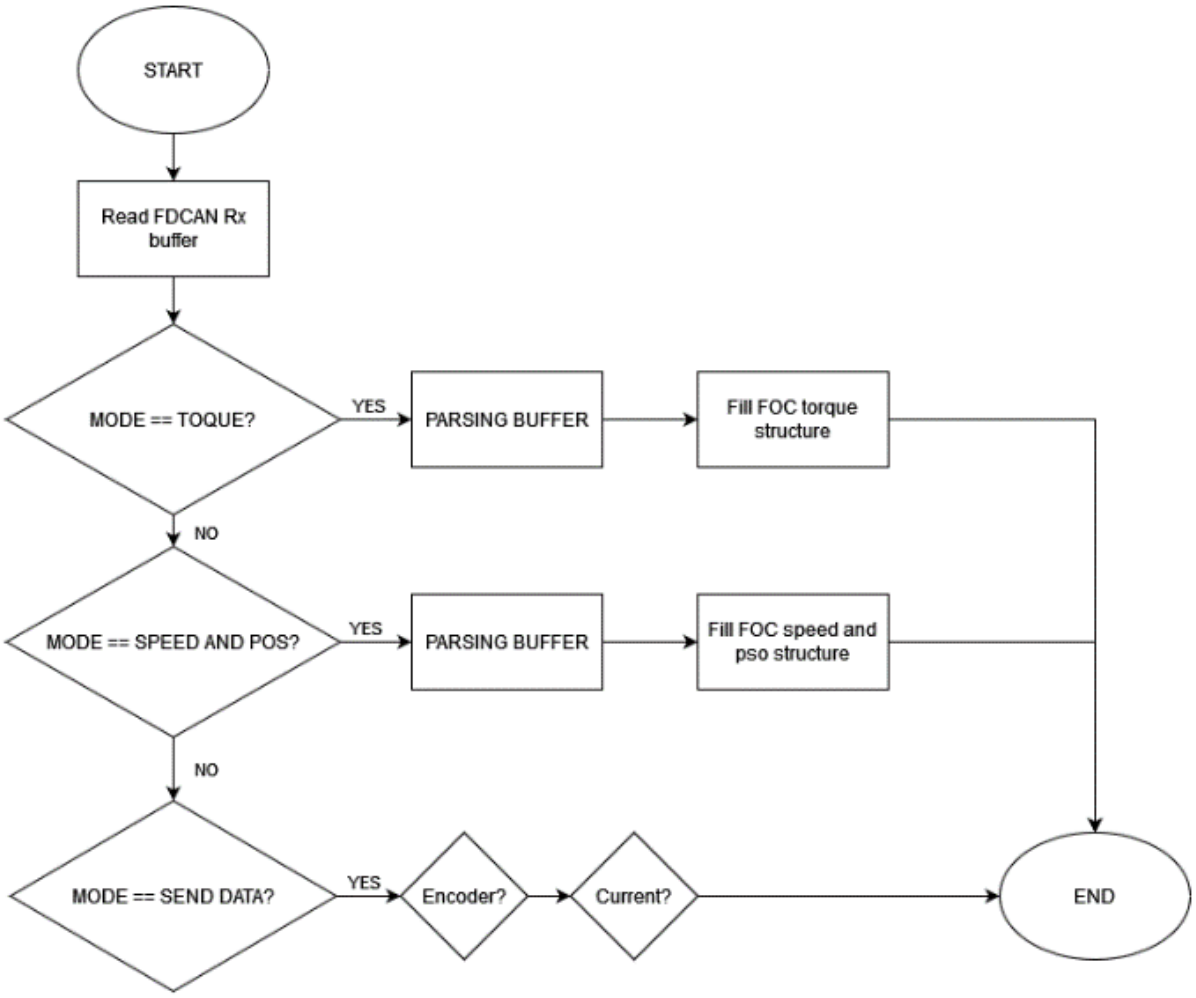

Figure 5.4. Message processing algorithm

Figure 5.8 illustrates the operation of vector control. As an example, current control with feedback is demonstrated, meaning that the motor always produces a constant torque (i.e., constant current, since torque is proportional to current).

The inputs i_q and i_d are regulated through feedback using a PID controller, which also includes several ”Park” and ”Clarke” transformation modules. The signals pass through the ”SVPWM” (Space Vector Pulse Width Modulation) block and impact the three-phase inverter to control the motor. The feedback magnitude of the PID controller represents a sampled value of the motor’s output current.

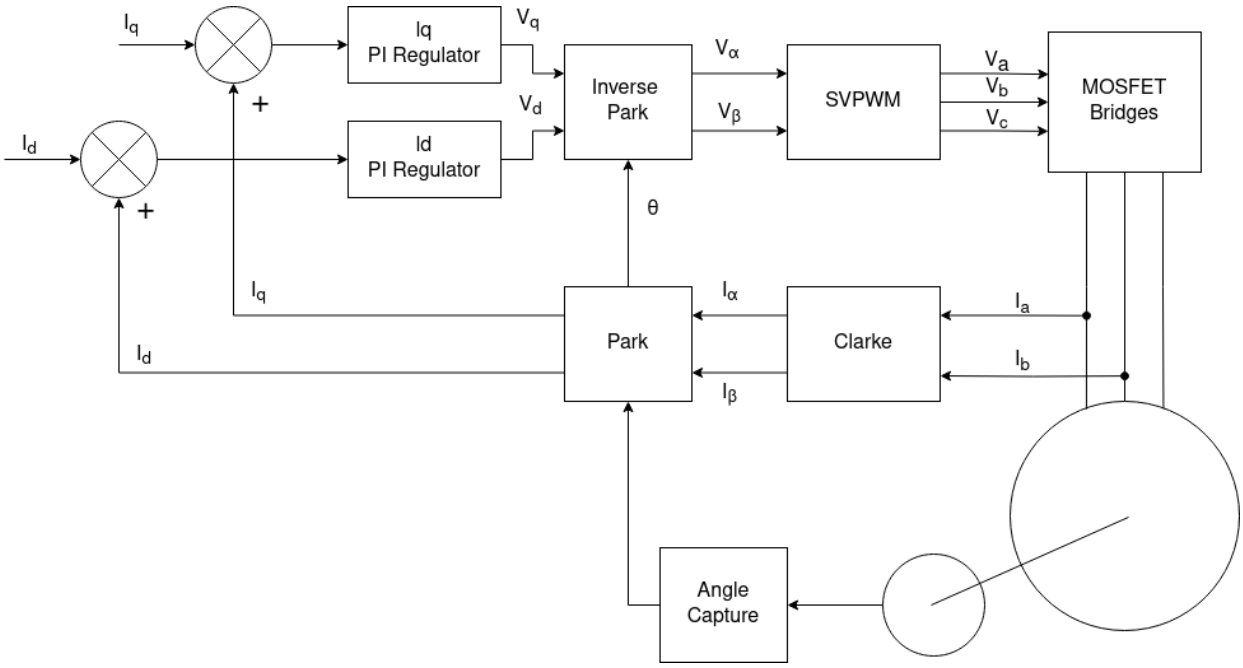

Figure 5.5. Vector motor control diagram.

The entire vector control process can be broken down into stages:

1. Acquiring data of the three-phase motor current to compute the values I_a, I_b;
2. Converting I_a, I_b, I_c using the Clarke transformation, obtaining I_α, I_β;
3. Converting I_α, I_β using the Park transformation, obtaining I_q, I_d;
4. Computing the error of $I_q, I_d$from the set values of I_q, I_dcoming from the controller;
5. Inputting the specified error into two PID controllers (only PI is used) to get the output control voltages V_q, V_d;
6. Converting V_q, V_d using the inverse Park transformation, obtaining V_α, V_β;
7. Using the space vector V_α, V_β to determine the pulse width modulation signals for the three half-bridges of the inverter;
8. Controlling the power MOSFET transistors of the three-phase inverter according to the previously derived code value for motor drive;
9. Repeating the steps mentioned above.

In the calculations, it is sufficient to use the current from only two phases of the motor, as according to Kirchhoff's current law, the third current component can be calculated since the sum of currents flowing into a node is equal to the sum of currents flowing out of the node $I_a + I_b + I_c = 0$. The vectors $I_a, I_b, I_c$are not initially orthogonal, and further work requires recalculating the projection

into the coordinate axes, formula 13.

$$\begin{cases} I_\alpha = I_a - \cos\left(\frac{2\pi}{3}\right) I_b - \cos\left(\frac{2\pi}{3}\right) I_c \\ I_\beta = \sin\left(\frac{2\pi}{3}\right) I_b - \sin\left(\frac{2\pi}{3}\right) I_c \end{cases} \tag{13}$$

After the transformation, a sinusoidal wave is obtained, but with one variable less.The values I_α, I_β can be used to control the rotation of the motor, making them correspond to the rules of signal shape change. The inverse Clarke transformation is applied to the three phases of the motor.

The Park transformation, formula 14, aims to translate the stationary system I_α, I_β into a coordinate system rotating with the rotor I_q, I_d, as the system is supposed to receive real-time motor rotation angle data from an encoder. As a result, the rotation vector becomes a fixed value in the I_α, I_β coordinate system. Subsequently, I_q, I_d are used as control objects through feedback via a PID controller. In reality, only a PI controller is used, as the differential regulator is omitted, because the transfer function of voltage and current is a first-order inertial element.

$$\begin{cases} I_d = I_\alpha \cos(\theta) + I_\beta \sin(\theta) \\ I_q = -I_\alpha \sin(\theta) + I_\beta \cos(\theta) \end{cases} \tag{14}$$

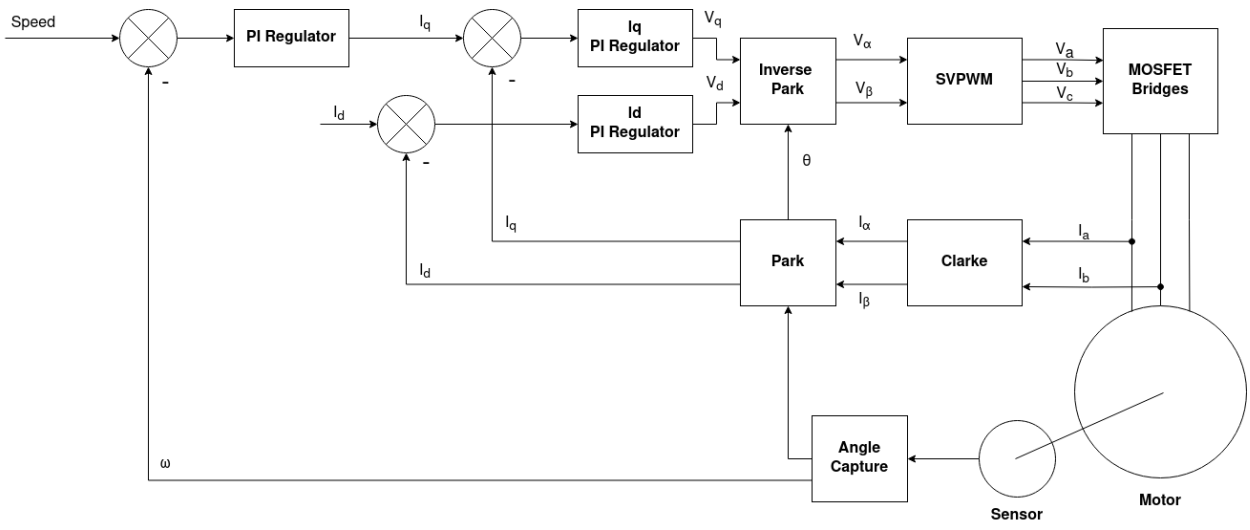

Figure 5.6. Speed vector motor control scheme

In standard vector control, three PID loops are primarily used: current loop, speed loop, and position loop. This means controlling the motor current (torque) via current feedback, then controlling the motor speed by managing the torque, and finally controlling the motor position by regulating the speed.

In the figure above 5.6, ω is the motor speed feedback, which is calculated using the motor's

encoder. It is controlled by a PI regulator.

The calculated motor speed and the speed setpoint value are used to compute the error value, which is then fed into the PI speed loop. The computed result is used as the input signal for the current loop, thus implementing dual speed-current feedback control.

The outermost level is the position loop, which controls the motor by rotating it to a precise angle and maintaining it. The control block diagram is shown in the figure 5.7.

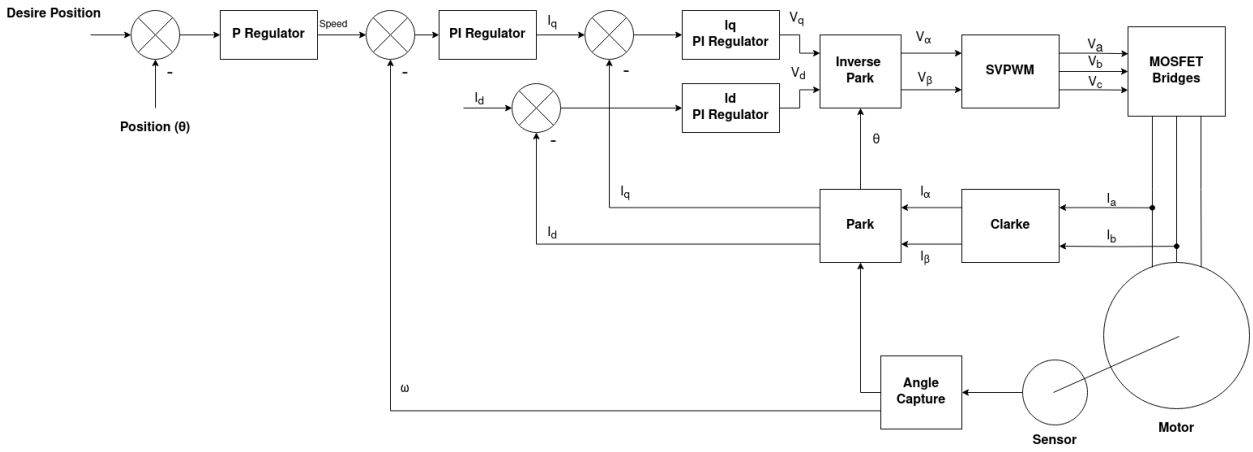

Figure 5.7. Motor position vector control scheme

But in the real-world application of the algorithm to a system, the encoder cannot directly return the motor speed. Therefore, the motor speed is calculated by determining the change in encoder value over a specific period of time (i.e., using the average speed to represent the instantaneous speed). This method is suitable when the motor speed is relatively high, but in position control mode, the motor speed will be very low (since the rotor needs to be fixed in a specific position). To avoid errors caused by speed feedback, only a dual loop consisting of position and current is used for position control. However, at this time, certain modifications need to be made to the position loop as shown in Figure 5.8.

However, it is also taken into account that with the speed loop removed, full PID control is used for the position loop, meaning a differential term is added (since the differential of position is speed, this can reduce position regulation oscillations and speed up convergence, the function of the integral term is to eliminate static error).

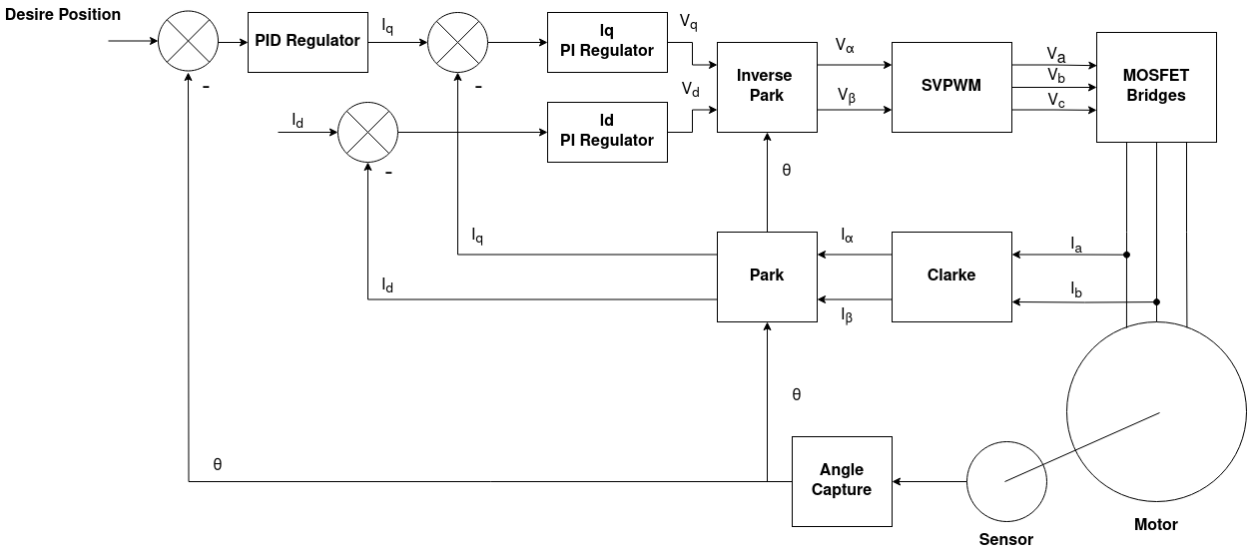

Figure 5.8. Optimal position vector motor control scheme

For direct conversion to signals supplied to the power transistors, the SVPWM (Space Vector Pulse Width Modulation) method is used, as it is more efficient than SPWM (Mirdas et al., 2023).

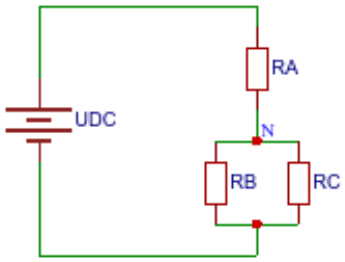

Figure 5.9. Equivalent circuit

To understand the process of signal formation, it is necessary to comprehend the concept of the space voltage vector. The connection of the motor to the voltage source can be visualized as shown in Fig.5.9.

The three-phase voltages (phase voltage is the voltage of each phase relative to the midpoint of the motor's connection) are expressed by the formula 15:

$$\begin{cases} U_a &= U_A - U_N = \frac{2}{3}U_{dc} \\ U_b &= U_B - U_N = -\frac{1}{3}U_{dc} \\ U_c &= U_C - U_N = -\frac{1}{3}U_{dc} \end{cases} \quad (15)$$

It can be envisioned that the three voltages form vectors $\vec{U_a}, \vec{U_b}, \vec{U_c}$, which in turn create a resultant vector $\vec{U}$, enabling the determination of the magnetic field vector. Given that the permanent magnet of the rotor will tend to rotate until the lines of its internal magnetic field align with the direction of the

external magnetic field, this vector can effectively represent the direction in which we want the rotor to rotate. Thus, it becomes possible to calculate the space voltage vector 16 (Instruments, 2013).

$$\begin{cases} U_A(t) = U_{dc}\cos(2\pi ft) \\ U_B(t) = U_{dc}\cos\left(2\pi ft - \frac{2\pi}{3}\right) \\ U_C(t) = U_{dc}\cos\left(2\pi ft + \frac{2\pi}{3}\right) \end{cases} \tag{16}$$

The SVPWM method synthesises the SPWM value at each time instant17.

$$\int_0^T U_{\text{ref}}\,dt = \int_0^{T_x} U_x\,dt + \int_{T_x}^{T_x+T_y} U_y\,dt + \int_{T_x+T_y}^{T} U_0^*\,dt \tag{17}$$

$$U_{\text{reference}} \cdot T = U_x \cdot T_x + U_y \cdot T_y + U_0^* \cdot T_0^* \tag{18}$$

Thus, the program in the microcontroller uses discretization, which can be simplified to 18, where $U_{\text{reference}}$ is the expected voltage vector, and T is the PWM period. The essence of the above formula lies in periodically switching between different space voltage vectors, through which an equivalent arbitrary space voltage vector can be synthesized.

5.3. Software implementation of the actuator control device

The microcontroller program was developed in C/C++ language, and to facilitate and accelerate development, the HAL (Hardware Abstraction Layer) library was utilized. The set of libraries allows for a higher level of abstraction and enables the porting of code to microcontrollers of a different family. The STM32CubeMX graphical configuration tool was used for project generation and setup. The project was generated for the STM32CubeIDE integrated development environment, which is a modified version of the Eclipse development environment for embedded systems. Figure 5.10 shows the graphical representation of the clock frequency setting and the clock frequencies of individual peripheral blocks. The microcontroller operates at its maximum frequency, considering the use of an external 8 MHz quartz crystal oscillator. The input frequency is fed into the Phase-Locked Loop (PLL) block, where it undergoes frequency multiplication and division to the output frequency, reaching a level of 94% of the maximum, i.e., 160 MHz.

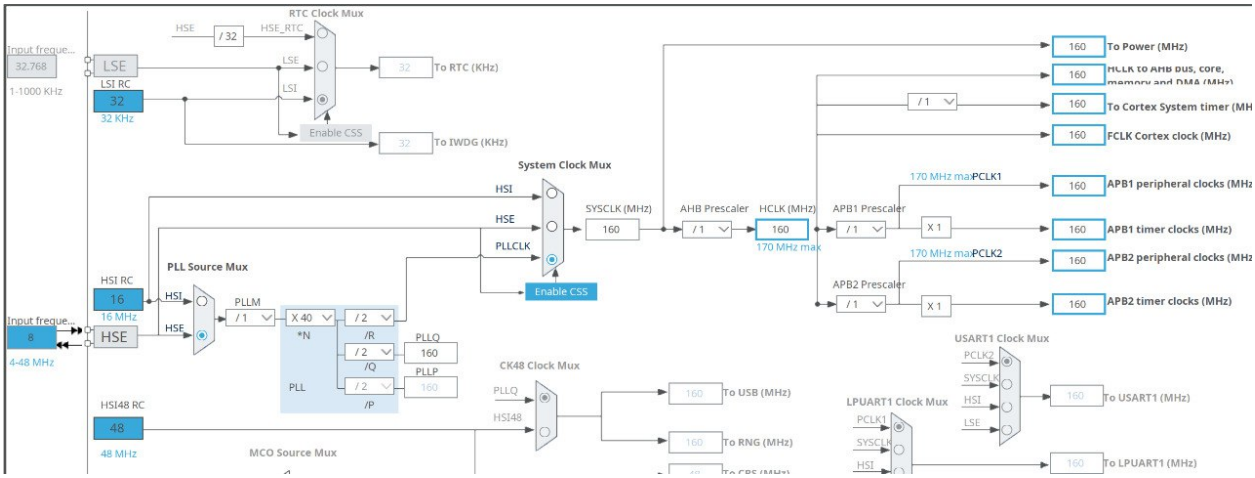

Figure 5.10. STM32G431 microcontroller clocking structure

The encoder is mounted on the motor rotor together with the gearbox, and it was necessary to solve the problem of maintaining the rotation angle. For this purpose, an algorithm for preserving the rotation angle was created for the encoder's rotation range from 0 to 360 degrees. In application 3, a test of the encoder rotation angle algorithm is implemented in Python. To ensure data reading about the rotor's position, the microcontroller sends a data request to the AS5600 encoder, after which a function fills the data structure. A snippet of the function's code is shown in Figure 5.11.

```
angle_t data;
data.raw = 0;
  if(HAL_I2C_Mem_Read(a->i2cHandle,a->adress,0x0E,I2C_MEMADD_SIZE_8BIT,
  ↪  (uint8_t*)&data.raw,2,2)!= HAL_OK)
{
  status = HAL_ERROR;
}
  *angle = ((data.bit.angleHIGH << 8) & 0xF00) | data.bit.angleLOW;
}
```

Figure 5.11. Code fragment of the encoder read function

The encoder is mounted directly on the shaft of the electric motor. The rotation of the motor exceeds the measurement range of the absolute encoder AS5600 from 0 to 360 degrees (due to the use of a gearbox), necessitating the measurement of the rotation angle in a wider range. The function that measures the current position, speed, and acceleration values is presented in Annex 4.

5.4. Software implementation of the tactical control device

During various calculations, operations are required that go beyond the standard C++ libraries. Matrix multiplication is often used in calculations, and a function for multiplying two matrices has been implemented, which is shown in Figure 5.12. The matrix multiplication function is designed to take three pointers to arrays representing two input matrices and one output matrix, as well as three integer values indicating the sizes of these matrices. The input matrices are referred to as the first and second, and the result of their multiplication is recorded in the output matrix.

To perform matrix multiplication, the function employs three nested loops, where each loop is responsible for its part of the process: the outer loop iterates through the rows of the first matrix, the middle loop through the columns of the second matrix, and the inner loop deals with the elements of the current rows and columns involved in the calculation. This allows the values for each cell of the resulting matrix to be computed sequentially.

```
void MultiplyMatrices(const float* matA, const float* matB, float*
↪  resultMat, int rowsA, int commonDim, int colsB) {
float sum;
int rowIdx, colIdx, kIdx;
// Loop through each row of the first matrix
for (rowIdx = 0; rowIdx < rowsA; rowIdx++) {
    // Loop through each column of the second matrix
    for (colIdx = 0; colIdx < colsB; colIdx++) {
        sum = 0.0f; // Temporary variable to store sum
        // Multiply elements across the common dimension
        for (kIdx = 0; kIdx < commonDim; kIdx++) {
            sum += matA[commonDim * rowIdx + kIdx] * matB[colsB * kIdx +
            ↪  colIdx];            }
        // Assign the computed sum to the result matrix
        resultMat[colsB * rowIdx + colIdx] = sum;
    }}}
```

Figure 5.12. Function for multiplication of two matrices

During the multiplication process, the dot product of the corresponding vectors is calculated for each pair of row and column in the input matrices. To do this, elements of the row from the first matrix

are successively multiplied by the corresponding elements of the column from the second matrix, and the results of these multiplications are added together to form the value in the cell of the resulting matrix.

Thus, each element of the output matrix is the sum of the products of the elements of the corresponding row and column of the input matrices, which provides the result of matrix multiplication.

The function for converting a rotation matrix to Euler angles, shown in Figure 5.13, takes two arguments: the first argument is a pointer to the input rotation matrix, and the second argument is a pointer to an array where the calculated Euler angles will be stored.

```
void ConvertRotationMatrixToEulerAngles(const float* rotationMatrix,
↪  float* angles){
    float roll, pitch, yaw, cosPitch;
    if (fabs(rotationMatrix[6]) >= 1.0 - 0.0001){//Check gimbal lock
        roll = 0.0f;  // Set roll to zero as it's indeterminate
        if (rotationMatrix[6] < 0) {// Handle the gimbal lock case
            pitch = (float) M_PI_2;  // 90 degrees
            yaw = atan2f(rotationMatrix[1], rotationMatrix[4]);
        }else{
            pitch = -(float) M_PI_2;  // -90 degrees
            yaw = -atan2f(rotationMatrix[1], rotationMatrix[4]);
        }}else{
        // Regular case, no gimbal lock
        pitch = atan2f(-rotationMatrix[6], sqrtf(rotationMatrix[0] *
        ↪  rotationMatrix[0] + rotationMatrix[3] * rotationMatrix[3]));
        cosPitch = cosf(pitch);
        roll = atan2f(rotationMatrix[3] / cosPitch, rotationMatrix[0] /
        ↪  cosPitch);
        yaw = atan2f(rotationMatrix[7] / cosPitch, rotationMatrix[8] /
        ↪  cosPitch);        }
    angles[0] = yaw;        // Assign calculated angles to output
    angles[1] = pitch;
    angles[2] = roll; }
```

Figure 5.13. Function of matrix transformation into Euler angles

At the beginning of the function, variables for the three Euler angles: yaw, pitch, and roll, as well as a variable for the cosine of the pitch, are declared. These variables are used to store intermediate calculations and the final results of the conversion.

The function then checks for the condition known as ”gimbal lock,” which occurs when one of the rotation axes becomes undefined due to the parallelism of the other two axes. The gimbal lock condition is determined by the value of an element in the rotation matrix, and if this value is close to 1 or -1, special processing is performed to calculate the Euler angles.

If the rotation matrix element is less than zero, the pitch angle is set to 90 degrees, and the roll angle is set to zero. The yaw angle is calculated using the arctangent function of two other matrix elements. If the matrix element is not less than zero, the pitch is set to -90 degrees, roll to zero, and yaw is calculated as the negative value of the angle obtained using the arctangent function.

If gimbal lock is not detected, the pitch is calculated through the arctangent of the negative value of the corresponding matrix element and the square root of the sum of the squares of the other two elements. Roll and yaw are then calculated using the cosine of the calculated pitch and the arctangent function for the corresponding matrix elements.

In the end, the calculated yaw, pitch, and roll angle values are stored in the output array in the specified order, completing the conversion of the rotation matrix to Euler angles. The function ConvertEulerAnglesToRotationMatrix is used for the inverse transformation.

The function **ConvertEulerAnglesToRotationMatrix**, Figure (**?**) to convert Euler angles to a rotation matrix takes two pointers to arrays containing the Euler angles (yaw, pitch and roll) and an array for the resulting rotation matrix. At the beginning of the function, variables for the cosines and sines of each of the Euler angles are initialised. These values are calculated to simplify subsequent calculations, as they are used repeatedly in the transformation formulas.

```
void ConvertEulerAnglesToRotationMatrix(const float* angles, float*
↪  rotationMatrix)
{
float cosRoll, cosPitch, cosYaw, sinRoll, sinPitch, sinYaw;
// Calculate sine and cosine of angles for efficiency
cosYaw = arm_cos_f32(angles[0]);
cosPitch = arm_cos_f32(angles[1]);
cosRoll = arm_cos_f32(angles[2]);
sinYaw = arm_sin_f32(angles[0]);
sinPitch = arm_sin_f32(angles[1]);
sinRoll = arm_sin_f32(angles[2]);

// Populate the rotation matrix using the sine and cosine values
rotationMatrix[0] = cosRoll * cosPitch;
rotationMatrix[1] = cosRoll * sinPitch * sinYaw - sinRoll * cosYaw;
rotationMatrix[2] = cosRoll * sinPitch * cosYaw + sinRoll * sinYaw;
rotationMatrix[3] = sinRoll * cosPitch;
rotationMatrix[4] = sinRoll * sinPitch * sinYaw + cosRoll * cosYaw;
rotationMatrix[5] = sinRoll * sinPitch * cosYaw - cosRoll * sinYaw;
rotationMatrix[6] = -sinPitch;
rotationMatrix[7] = cosPitch * sinYaw;
rotationMatrix[8] = cosPitch * cosYaw;
}
```

Figure 5.14. Function of transformation of Euler angles into a matrix

Then, using these pre-calculated cosine and sine values, the function fills in the elements of the rotation matrix. The rotation matrix that results from the function is an orthogonal matrix that describes rotation in three-dimensional space.

The main task of the tactical device is to calculate the trajectory of the robot, in order to provide this we have written functions to solve the forward or inverse kinematic problem for a 6-axis mini robot, the code is presented in **Annex 6**.

The function ”CalculateFK” is designed to determine the position and orientation of the final actuator of a robot with six degrees of freedom based on the given rotation angles of its links. Then, for each link, a rotation matrix is calculated based on its rotation angle and parameters specified in

the Denavit-Hartenberg table. These parameters describe the mutual arrangement of the robot links. The rotation matrix of all links is multiplied sequentially from the base of the robot to its final actuator. This multiplication gives the total rotation matrix, which describes the complete orientation and position of the final actuator relative to the robot base. By summing the position vectors of all links, the function finds the final position of the robot's final actuator. In the final step, based on the total rotation matrix, the function calculates Euler angles that describe the orientation of the final actuator in space. These angles represent rotation around three axes and allow the exact orientation of the actuator to be determined. The results of the calculations - the coordinates of the position of the actuator and its orientation - are written to the output structure **outputPose**.

The most complex function is the solution of the inverse kinematics problem, which, based on the rotation matrix, calculates three angles: yaw, pitch and roll. For yaw and pitch, the calculations are based on rotations around the vertical and horizontal axes, respectively, while roll is determined through rotation around the gaze axis. The calculations take into account the physical limitations of the joints to ensure that the resulting angles are realistic and feasible. The results, represented as yaw, pitch and roll angles, are then written to a designated array, ready to be used to control the robot's movements. The boolean value returned by the function serves as an indicator of the success of the operation: true indicates that the angles have been successfully calculated and are within acceptable values, while false indicates that the required orientation cannot be achieved due to constraints imposed by the joint design.

CONCLUSION

During the bachelor's thesis, an analysis of mini robots was conducted, as well as an examination of the characteristics of mini manipulator robots and their shortcomings. The issues of using robots in collaboration with humans, the necessity for the features of creating collaborative robots were described. Variants of the engine, the choice, and options for controlling a brushless DC motor, as well as the option of using motor control, were considered, and the method of vector control with feedback was selected.

Functions and technical specifications for the developed control system for a collaborative mini manipulator robot were formulated. A hierarchical paradigm of robot control systems was considered and adopted as the basis for separating the principal control system schemes for the mini robot. The algorithm for interaction between tactical, strategic, and executive control systems was described. Algorithms were developed based on the described hierarchical levels of architecture:

- The algorithm for the tactical control system operation;
- The algorithm for the executive control system operation.

A functional diagram of the device was developed in accordance with the selected components:

- Gimbal GBM4008H-150T, GM5208-120T, GM3506;
- Absolute Encoder AS5600;
- Strategic control device Raspberry Pi Zero W 2;
- Power transistors SIR680DP;
- Drivers for power transistors L6385ED;
- Data bus transmitter TCAN1462DRQ1;
- DC-DC converter AP64502QSP.

Mathematical operations required for solving tasks were considered, and as a result, testing and analysis of available microcontrollers for vector control tasks and kinematic tasks were conducted. Two principal device schemes were developed, with the necessary calculations of elements for tactical, strategic, and executive control systems.

- An algorithm for the control system of a mini manipulator robot was developed;
- An algorithm for the tactical control system of a mini robot;
- An algorithm for the executive control system of a mini robot.

Thus, it can be said that a control system for a collaborative mini manipulator robot has been successfully developed to demonstrate the capabilities of using Gimbal-type motors as the robot's actuating device. This topic remains relevant as components with similar characteristics but smaller in size continue to emerge.

REFERENCES

1. ams (2023), 'ams.com'. Available at https://ams.com/documents/20143/36005/AS5600_DS000365_5-00.pdf/649ee61c-8f9a-20df-9e10-43173a3eb323.

2. Bélanger-Barrette, M. (2015), 'What Does Collaborative Robot Mean ? — blog.robotiq.com'. Available at https://blog.robotiq.com/what-does-collaborative-robot-mean.

3. DIODES (2021), 'Datasheet AP64502Q'. Available at https://lv.mouser.com/datasheet/2/115/DIOD_S_A0011818538_1-2543578.pdf.

4. Gorupec, D. (2014), 'Stepper motor vibrations'. Available at http://charming.awardspace.us/stepper/Stepper_motor_vibrations.pdf.

5. Group, S. M. (2020), 'Brushed Versus Brushless DC Motors: What's the Best Choice for Medical Applications? — medicaldesignbriefs.com'. Available at https://www.medicaldesignbriefs.com/component/content/article/37231-brushed-versus-brushless-dc-motors.

6. Huang, G.-S., Tung, C.-K., Lin, H.-C. and Hsiao, S.-H. (2011), Inverse kinematics analysis trajectory planning for a robot arm, *in* '2011 8th Asian Control Conference (ASCC)', pp. 965–970.

7. Instruments, T. (2013), 'Field Oriented Control of Permanent Magnet Motors — youtu.be'. Available at https://youtu.be/cdiZUszYLiA.

8. ISO (2011), 'ISO 10218-1:2011 — iso.org'. Available at https://www.iso.org/standard/51330.html.

9. ISO (2016), 'ISO/TS 15066:2016 — iso.org'. Available at https://www.iso.org/standard/62996.html.

10. Khatib, O., Quinlan, S. and Williams, D. (1997), 'Robot planning and control', *Robotics and Autonomous Systems* **21**(3), 249–261. Available at http://dx.doi.org/10.1016/S0921-8890(96)00078-4.

11. Li, J., Guan, Y., Chen, H., Wang, B., Zhang, T., Hong, J. and Wang, D. (2022), 'Real-time normal contact force control for robotic surface processing of workpieces

without a priori geometric model', *The International Journal of Advanced Manufacturing Technology* **119**(3–4), 2537–2551. Available at http://dx.doi.org/10.1007/s00170-021-07497-2.

12. Matsuki, H., Nagano, K. and Fujimoto, Y. (2019), 'Bilateral drive gear—a highly backdrivable reduction gearbox for robotic actuators', *IEEE/ASME Transactions on Mechatronics* **24**(6), 2661–2673.

13. Megalingam, R. K., Sahajan, A., Rajendraprasad, A., Manoharan, S. K. and Reddy, C. P. K. (2021), Ros based six-dof robotic arm control through can bus interface, *in* '2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)', IEEE. Available at http://dx.doi.org/10.1109/ICICCS51141.2021.9432341.

14. Mirdas, Q. H., Yasin, N. M. and Alshamaa, N. K. (2023), Analytical comparison of spwm amp; svpwm techniques for three-phase induction motor v/f speed control, *in* 'AIP Conference Proceedings', AIP Publishing. Available at http://dx.doi.org/10.1063/5.0154520.

15. Pack, D. and Barrett, S. (2008), *Microcontroller Theory and Applications: HC12 and S12*, Pearson Prentice Hall. Available at https://books.google.lv/books?id=klZUPxiGUJ4C.

16. Sakama, S., Tanaka, Y. and Kamimura, A. (2022), 'Characteristics of hydraulic and electric servo motors', *Actuators* **11**(1), 11. Available at http://dx.doi.org/10.3390/act11010011.

17. Sensinger, J. W. and Lipsey, J. H. (2012), Cycloid vs. harmonic drives for use in high ratio, single stage robotic transmissions, *in* '2012 IEEE International Conference on Robotics and Automation', IEEE. Available at http://dx.doi.org/10.1109/ICRA.2012.6224739.

18. simplefoc (2023), 'BLDC motors'. Available at https://docs.simplefoc.com/bldc_motors.

19. STM (2020*a*), 'Datasheet STM32G431'. Available at https://www.st.com/resource/en/datasheet/stm32g431rb.pdf.

20. stm (2020*b*), 'FDCAN peripherals for STM32 product'. Available at https://www.st.com/resource/en/application_note/

an5348-introduction-to-fdcan-peripherals-for-stm32-product-classes-stmicr
pdf.

21. STM (2023), ‘How to use the CORDIC to perform mathematical functions on STM32 MCUs’. Available at https://www.st.com/resource/en/application_note/an5325-how-to-use-the-cordic-to-perform-mathematical-functions-on-stm32-m
pdf.

22. STMicroelectronics (n.d.), ‘STM32G4 ADC use tips and recommendations’. Available at https://www.st.com/resource/en/application_note/an5346-stm32g4-adc-use-tips-and-recommendations-stmicroelectronics.
pdf.

APPENDIX

Appendix 1.

Table comparing the results of thousands of operations per second at 1 MHz for each MCs

MCU	a + b	a - b	a * b	a / b	sin(a)	log(a)	sqrt(b)	pow(b, a)
ATmega328p	6,952	7,205	6,251	2,015	0,598	0,403	2,017	0,190
ESP32	46,212	54,007	48,121	16,325	8,568	3,986	11,006	1,484
ESP8266	14,412	13,861	8,852	3,497	0,485	0,239	0,787	0,082
RP2040 [Arduino IDE]	7,815	7,226	5,489	2,022	0,455	0,273	2,103	0,122
RP2040 [C++ SDK]	10,132	9,699	11,717	9,266	1,625	1,139	10,719	0,435
RP2040 [MicroPython]	0,629	0,707	0,723	0,681	0,441	0,400	0,568	0,215
STM32G431	101,054	98,251	114,565	43,435	13,254	6,178	44,849	1,118
STM32G431 [FPU-OFF]	15,008	14,425	21,932	6,216	1,391	0,607	2,939	0,176

Appendix 2.

Code fragment of the RobotDK TCP command driver code

```
import socket
import serial
import sys

TCP_IP = '127.0.0.1'  # Listen
TCP_PORT = 5005     # port

# UART settings
UART_PORT = '/dev/serial0'  # UART1 port on Rasp
BAUD_RATE =
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Setup socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

print(f"Listening for TCP connections on {TCP_IP}:{TCP_PORT}")

conn, addr = s.accept()
 try:
    while True:
        data = conn.recv(1024)  # Buffer = 1024
        if not data:
            break
        ser.write(data)  # Write data to UART
```

Appendix 3.

Software code Python simulation code for converting encoder rotation angle to total angle

```
# Initialize variables
last_encoder_value = 0
full_rotation_count = 0
total_angle_degrees = 0
cpr = 4096  # Counts per revolution for the encoder

def update_angle(encoder_value):
    global last_encoder_value, full_rotation_count, total_angle_degrees, cpr
    # Calculate the change in encoder value
    d_value = encoder_value - last_encoder_value
    if d_value > cpr / 2:
        full_rotation_count -= 1

    elif d_value < -cpr / 2:
        full_rotation_count += 1

    # Update the total angle in degrees, considering multiple rotations
    total_angle_degrees = (full_rotation_count * 360) + (encoder_value /
    ↪  cpr) * 360
    # Update last encoder value for the next call
    last_encoder_value = encoder_value
    return total_angle_degrees

# This loop simulates the encoder values changing
for i in range(10000):
    encoder_value = i % cpr  # Simulate encoder value
    angle = update_angle(encoder_value)
    print(f"Total Angle: {angle} degrees")
```

Appendix 4.

Code fragment of the encoder data processing function.

```
float d_time;float d_angle;
uint16_t angle_data;
float resolution = 4096;
uint16_t now = __HAL_TIM_GET_COUNTER(&htim6);
//Calculate the time between the current and last measurements
d_time = now <= encoder->lastUpdateTime ? 0xffff - encoder->lastUpdateTime +
↪  now : now - encoder->lastUpdateTime;
EncoderGetAngle(encoder->handle, &angle_data);
//Track the number of revolutions
d_angle = (float)(angle_data - encoder->last_angle_data);
//If detected occurs, count it as a full revolution
if(abs(d_angle) > (0.8 * resolution) ){
  if(full_rotation_offset>0{
    d_angle += 2*PI;
  }else{
    d_angle += -2*PI
  }}
//to determine if an overflow has occurred
encoder->last_angle_data = angle_data;
encoder->angle_rad = encoder->full_rotation_offset + ((float)angle_data /
↪  (float)resolution) * 2*PI;
encoder->angle_deg = RADIANS_TO_DEGREES(encoder->angle_rad);
encoder->velocity_rad = ((encoder->angle_rad - encoder->angle_prev_rad) /
↪  d_time * 1000000.0f);
encoder->velocity_deg = ((encoder->angle_deg - encoder->angle_prev_deg) /
↪  d_time * 1000000.0f);
encoder->angle_prev_rad = encoder->angle_rad; // save angle rad
encoder->angle_prev_deg = encoder->angle_deg; // save angle deg
//Update time of last measurement
encoder->lastUpdateTime = __HAL_TIM_GET_COUNTER(&htim6);
```

Appendix 5.

Code fragment for Vector Control Algorithm update

```
void FOC_Update(uint16_t time_us,float setpoint_torque_current_mA,float
↪  setpoint_flux_current_mA,float phase_synchro_offset_rad,uint32_t
↪  closed_loop,float setpoint_velocity_dps){
    float phaseCurrents[3];
    // performance monitoring
        uint16_t startTime = __HAL_TIM_GET_COUNTER(&htim6);
        for(size_t i = 0; i < 3; ++i) {
            phaseCurrents[i] = -((float)motor_current_sample_adc[i] -
            ↪  motor_current_input_adc_offset[i]) /
            ↪  motor_current_input_adc_mA[i];
        }// process cosine(theta) and sine(theta)
    float phaseOffset_rad = convert2degrees((int16_t)(MAKE_SHORT(
    ↪  regs[REG_MOTOR_SYNCHRO_L], regs[REG_MOTOR_SYNCHRO_H])));
    float polePairs = PP, direction = REVERSE;
    float theta_rad = fmodf(absolute_position_rad * polePairs * direction,
    ↪  2*PI) + phaseOffset_rad + syncOffset_rad;
    static float cos_theta = 0.0f, sin_theta = 0.0f;
    CORDIC_Processor(theta_rad, &cos_theta, &sin_theta);
        //Clarke Transformation
    float Ialpha = 2.0f / 3.0f * phaseCurrents[0] - 1.0f / 3.0f *
    ↪  (phaseCurrents[1] + phaseCurrents[2]);
    float Ibeta = 1.0f / sqrtf(3.0f) * (phaseCurrents[1] -
    ↪  phaseCurrents[2]);
    // Park Transformation
  float Id = Ialpha * cos_theta + Ibeta * sin_theta;
    float Iq = -Ialpha * sin_theta + Ibeta * cos_theta;
```

Annex 5.(continued)

```
  // (Id,Iq) filtering
float Id_filtered = 0.05f * Id + (0.05f) * present_Id_filtered;
  float Iq_filtered = 0.05f * Iq + (0.05f) * present_Iq_filtered;
    // flux controller (PI+FF)
  float setpoint_Id = fluxSet_mA;
  float Flux_Kp = (float)((int16_t)(MAKE_SHORT(
  ↪  regs[PID_FLUX_CURRENT_KP_L], regs[PID_FLUX_CURRENT_KP_H]))) /
  ↪  100000.0f;
  float error_Id = setpoint_Id - (loopStatus == 1 ? Id_filtered : 0.0f);
  float Vd = error_Id * Flux_Kp;
  float setpoint_Iq = torqueSet_mA;
  float Torque_Kp = (float)((int16_t)(MAKE_SHORT(
  ↪  regs[PID_TORQUE_CURRENT_KP_L], regs[PID_TORQUE_CURRENT_KP_H]))) /
  ↪  100000.0f;
  float error_Iq = setpoint_Iq - (loopStatus == 1 ? Iq_filtered : 0.0f);
  float Vq = error_Iq * Torque_Kp;
  // do inverse clarke and park transformation
  float Valpha = Vd * cos_theta - Vq * sin_theta;
  float Vbeta = Vq * cos_theta + Vd * sin_theta;
      // Inverse Clarke Transformation
  float Va = Valpha;
  float Vb = (-Valpha + sqrtf(3.0f) * Vbeta) / 2.0f;
  float Vc = (-Valpha - sqrtf(3.0f) * Vbeta) / 2.0f;
  float Vneutral = 0.5f * (fmaxf(fmaxf(Va, Vb), Vc) + fminf(fminf(Va, Vb),
  ↪  Vc));
    // convert (Va,Vb,Vc) to PWM duty cycles % [0.0 1.0]
float dutyA = fconstrain(((Va - Vneutral) / present_voltage_V + 1.0f) *
↪  0.5f, MIN_PWM_DUTY_CYCLE, MAX_PWM_DUTY_CYCLE);
```

Annex 5.(continued)

```
float dutyB = fconstrain(((Vb - Vneutral) / present_voltage_V + 1.0f) *
↪  0.5f, MIN_PWM_DUTY_CYCLE, MAX_PWM_DUTY_CYCLE);
float dutyC = fconstrain(((Vc - Vneutral) / present_voltage_V + 1.0f) *
↪  0.5f, MIN_PWM_DUTY_CYCLE, MAX_PWM_DUTY_CYCLE);
    // fPWM = 16KHz
    // fTIM = 160MHz
    // ARR = fTIM/(2 * fPWM) -1 => ARR = 4999
uint16_t CCRa = (uint16_t)(dutyA *
↪  (float)(__HAL_TIM_GET_AUTORELOAD(&htim1) + 1)) - 1;
uint16_t CCRb = (uint16_t)(dutyB *
↪  (float)(__HAL_TIM_GET_AUTORELOAD(&htim1) + 1)) - 1;
uint16_t CCRc = (uint16_t)(dutyC *
↪  (float)(__HAL_TIM_GET_AUTORELOAD(&htim1) + 1)) - 1;
// update TIMER CCR register
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, CCRa);
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, CCRb);
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, CCRc);
// calculate loop time
int16_t endTime = __HAL_TIM_GET_COUNTER(&htim6);
uint16_t processingTime = endTime - startTime;
static const float performanceAlpha = 0.001f;
average_processing_time_us = (1.0f - performanceAlpha) *
↪  average_processing_time_us + performanceAlpha *
↪  (float)processingTime;
++foc_cnt;
```

Appendix 6.

Function for calculating the Forward Kinematics

```
CalculateFD(const JointAngles& inputAngles, RobotPose& outputPose){
  float rawInputAngles[6];
  float adjustedAngles[6];
  float cosineAngle, sineAngle;
  float cosineAlpha, sineAlpha;
  float endEffectorPosition[6];
  float endEffectorRotation[9];
  float rotationMatrices[6][9];
  float rotationMatrixPartial[4][9];  // Only need 4 as we're multiplying
  ↪ sequentially
  float linkVectors[4][3];  // Vectors from base to shoulder, shoulder to
  ↪ elbow, etc.
  // Convert input angles from degrees to radians
  for (int i = 0; i < 6; i++)
      rawInputAngles[i] = inputAngles.values[i] / DEGREES_TO_RADIANS;
  // Compute rotation matrices based on adjusted joint angles
  for (int i = 0; i < 6; i++){
      adjustedAngles[i] = rawInputAngles[i] + DHParams[i][0];
      cosineAngle = arm_cos_f32(adjustedAngles[i]);
      sineAngle = arm_sin_f32(adjustedAngles[i]);
      cosineAlpha = arm_cos_f32(DHParams[i][3]);
      sineAlpha = arm_sin_f32(DHParams[i][3]);
      // Populate each rotation matrix for the joint
      rotationMatrices[i][0] = cosineAngle;
      rotationMatrices[i][1] = -cosineAlpha * sineAngle;
      rotationMatrices[i][2] = sineAlpha * sineAngle;
      rotationMatrices[i][3] = sineAngle;
      rotationMatrices[i][4] = cosineAlpha * cosineAngle;
      rotationMatrices[i][5] = -sineAlpha * cosineAngle;
```

Annex 6.(continued)

```
  rotationMatrices[i][6] = 0.0f;
      rotationMatrices[i][7] = sineAlpha;
      rotationMatrices[i][8] = cosineAlpha;
  }}
// Sequentially multiply the rotation matrices to get the cumulative
↪  rotation matrix
MultiplyMatrices(rotationMatrices[0], rotationMatrices[1],
↪  rotationMatrixPartial[0], 3, 3, 3);
for (int i = 1; i < 4; i++){  // Start from 1 as first multiplication
↪  already done
  MultiplyMatrices(rotationMatrixPartial[i - 1], rotationMatrices[i + 1],
  ↪  rotationMatrixPartial[i], 3, 3, 3);
}
MultiplyMatrices(rotationMatrixPartial[3], rotationMatrices[5],
↪  endEffectorRotation, 3, 3, 3);
// Calculate position vectors for each segment in the arm's kinematic
↪  chain
MultiplyMatrices(rotationMatrices[0], LinkBaseToShoulder, linkVectors[0],
↪  3, 3, 1);
MultiplyMatrices(rotationMatrixPartial[0], LinkShoulderToElbow,
↪  linkVectors[1], 3, 3, 1);
MultiplyMatrices(rotationMatrixPartial[1], LinkElbowToWrist,
↪  linkVectors[2], 3, 3, 1);
MultiplyMatrices(endEffectorRotation, LinkWristToEndEffector,
↪  linkVectors[3], 3, 3, 1);
// Sum up the position vectors to get the end effector's position
for (int i = 0; i < 3; i++)Inverse kinematics calculation function
```

Appendix 7.

Inverse kinematics calculation function

```
bool SolveIK(const Pose& targetPose, const JointStates& lastState,
↪  Solutions& solutions){
  // Initialize variables for joint angles and intermediate calculations
  float jointAngles[6], tempAngles[2], wristPos[3];
  float cosAngle, sinAngle, angle1, angle2, distSquared, dist;
  float endEffectorPos[3], endEffectorRot[9], tempRot[9], jointRot[9];
  ConvertPoseToRadians(targetPose, endEffectorPos, endEffectorRot);
  // Calculate the position of the wrist to solve for the first three
  ↪  joints
  CalculateWristPosition(endEffectorPos, endEffectorRot, wristPos);
  // Solve for the first three joints (base, shoulder, elbow)
  bool baseSolved = SolveBaseJoint(wristPos, jointAngles);
  bool armSolved = SolveArmJoints(wristPos, jointAngles, tempAngles);
  // If the base or arm cannot be solved, mark the solution as invalid
  MarkInvalidSolutions(baseSolved, armSolved, solutions);
  for (int i = 0; i < 2; i++)  // Loop through possible arm solutions
  {
      // Calculate the rotation matrix for the current arm solution
      CalculateArmRotationMatrix(jointAngles, tempAngles[i], tempRot);
      // Determine the wrist orientation relative to the base
      MultiplyMatrices(tempRot, endEffectorRot, jointRot, 3, 3, 3);
      // Solve for the wrist joints (roll, pitch, yaw)
      bool wristSolved = SolveWristJoints(jointRot, jointAngles + 3);
      // Store the solution if valid
      StoreSolution(wristSolved, jointAngles, solutions, i);
  }
  ConvertSolutionsToDegrees(solutions);
  return true;
}
```