Home-Owned Robotic Arm

Emanuel Alvarez, Hani Bdeir, Hailey Gorak, Jackson Jacques III, Nkunu Nuglozeh

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — This paper presents and details the components, core concepts, and overall design of the Home-Owned Robotic Arm project in a concise and readable fashion. This project aims to use a mix of positional and current-tracking hardware, CAN bus communication, and tele-operative control to present a uniquely designed robotic arm. With the addition of force-feedback in the system, which is implemented via sensors and real-time communication systems, this design presents a safer solution to automation, giving more control to the human operators and allowing the arm to detect collisions based on position and current spikes.**

***Index Terms* — Automation, Electric motors, Embedded systems, Micronctrollers, Motor drives, Remote Control, Robots, Servo systems.**

I. Introduction

The Home-Owned Robotic Arm is a robotic arm designed with safety towards its human operators and amount of force used during operation in mind. Using a mix of CAN communication, 3D simulated environment, and tele-operative control, our project aims to give human engineers in the automation industry more safeguards and control over their own safety during robotic operation. Using a tele-operative joystick device to drive each node of our robotic arm, the human operator has much more control over the speed, positioning, and pathway of the arm. Alongside remote operation, our design uses a 3D robotic simulator (ROS 2) to assist in exact and safe movements. The design consists of three joints, each one driving a different part of our arm on an x/y/z plane. A single motor is attached to each joint, one for forward/backward movement, one for up/down movement, and one for 360 degree rotation. Each motor has its own encoder and current sensor attached, which then sends data to that node's ESP32 microcontroller. These microcontrollers then take the data sent to them and send it to our Raspberry Pi computer over a CAN bus communication network. System feedback is passed between the computer and this network of microcontrollers to update and correct positioning based on data sent from the attached encoders and current sensors. This allows the system to collect and parse data for both simulation and path correction during movement. Data processed on the computer end of our network will then be passed back to the MCUs to correct motor position, if applicable during operation. The flow of information passed through the system is cyclical and always starts with the operator at the teleo-perative device. Input is given through four axes, two per joystick, all of which send commands through our computer, to our simulator, and then to our microcontrollers to drive each motor that has been triggered. There are three buttons that allow the operator to control behavior of the design's motors and end effector. The first of our buttons' functionalities is to record and replay movement, which saves each motor's movement while active, then replays this movement when triggered again. The second button functions as an emergency stop for the system, to be used in situations that could cause harm to a body, object, or the arm itself. The final button acts as a trigger for the attached end-effector. In our case, we are using a gripper end-effector, and our third button will trigger it to open and close around objects, letting the arm grip and pick up items. All information that has been passed from our joystick and to our computer is then sent to our microcontrollers via a CAN bus communication system, which uses a decentralized framework of nodes to pass data along a shared bus line. This system of communication is primarily used in automotive and automation industries, and allows all devices to communicate simultaneously, rather than wait around for a line or node to open up, allowing for faster and more reliable communication time. The overall focus of our project is real-time response and operator control.

II. System Components

This section details the exact physical modules that were used to create this design. A majority of the components listed were personally designed by our team, however some components were able to be purchased and integrated without customization. Reading this section should result in a deeper understanding of how this design was integrated. Further detail on this hardware and its functionality within our design can be found in section IV, Hardware Functionality.

A. Tele-Operative Device

The tele-operative control device for this project is a custom PCB based on the ATmega32u4 microcontroller, specifically the ATmega32U4-AU model. This microcontroller was selected due to its built-in Human Interface Device (HID) support, which allows it to communicate directly over USB as a standard input device. The board's micro-USB connector provides both power and data transmission capabilities to the Raspberry Pi, ensuring low-latency signal transmission necessary for real-time control.

Two 10kΩ potentiometer-based analog joysticks are mounted on the board, each capable of measuring X and Y axis positional changes to control the robotic arm's movement in multiple dimensions. Additionally, the board includes three push buttons configured for auxiliary functions. The board integrates two LEDs: a power LED connected to the USB power line for indicating active power and an operational status LED tied to a GPIO pin.

Supporting components include decoupling capacitors positioned near the ATmega32u4 to stabilize power input and minimize electrical noise, diodes and fuse for circuit protection, external pull-down resistors for each button to stabilize signal interpretation, and a reset button for reinitializing the microcontroller. Component layout on the PCB was configured for efficient space utilization within the physical constraints of the design.

B. Motors

The Home-Owned Robotic Arm utilizes NEMA 23 stepper motors. These motors operate at 24V with a peak current of 4A and provide a step angle of 1.8° per step, with a holding torque of 2.4 Nm. This model, widely recognized for its stability and precise positional control, is ideal for robotic applications requiring reliable and consistent movement.

C. Current Sensors

Each motor in the robotic arm is monitored by an ACS712ELCTR-05B-T current sensor, abbreviated here as the ACS712-05B. This sensor was chosen for its high sensitivity and suitability for the project's operating current range. The ACS712-05B can handle currents up to 5 amps, allowing it to accurately capture the load drawn by each motor, which remains within this range during operation. The sensor is connected in series with each motor and its respective motor controller, enabling continuous current measurement. The analog output from each sensor is routed to the ESP32 microcontroller on the motor's PCB, supplying real-time data for monitoring motor performance.

D. Position Sensor

The arm employs AS5600 magnetic absolute encoders, chosen for their high-resolution position tracking. These encoders provide absolute angular position feedback directly mounted on the output of each motor. The AS5600 communicates over I2C, offering a compact design with high accuracy and reliability.

E. Microcontrollers (ESP32-Wroom)

The ESP32-Wrooms were chosen for their compatibility with most components available, as they are highly versatile embedded systems which have a plethora of options for communication buses, including I2C, SPI, and Analog-to-Digital (ADC), all of which are necessary for our components. The SPI connections within these microcontrollers make the CAN communication system possible. They come with dual-core processors, at least 4MB of flash memory, and 26 GPIO pins, allowing us to communicate with every peripheral, node, and CAN bus efficiently without running into the issue of memory shortage. These devices operate at 3.3V and are not 5V tolerant.

F. CAN Bus Communication

CAN Bus systems are traditionally built via a controller node to control the system and a transceiver node to pass data. To achieve this, we designed a system that utilized the MCP25x family chips to talk with our ESP32s and be our main communication pathway within the bus. These systems consist of a single MCP2515 controller chip, and a single MCP2551 transceiver for each board. The MCP2515 can run at 3.3V or 5V, while the MCP2551 relies only on 5V. Using the MCP2515 and a current divider between the MCP RXD lines, we are able to interface our 5V intolerant MCUs with our CAN bus system and apply the necessary voltage by separating our voltage lines between both MCP chips.

G. CANable.io USB Adapter

The CANable.io USB adapter was bought directly from the CANable website, and is an open-source solution to CAN communication between devices that require a USB adapter. While it is possible to connect our computer with our CAN devices via wires, the CANable USB adapter works in a much more organized manner and allows us to set a CAN network up directly from our computer's terminal. This device connects to our CAN devices via the CANH and CANL wires, which allow all MCUs and our computer to pass and receive data from each other.

H. Computer (Raspberry Pi 5)

The Raspberry Pi 5 was selected for its computational capabilities, supporting complex control tasks for the robotic arm. This model features high processing power suitable for running kinematic calculations and integrates smoothly with a wide array of components via USB, I2C, and CAN bus communication, making it an effective bridge between user input and motor control.

I. Power Distribution

DC power is supplied by a Mean Well 24 volt 600 watt AC to DC converting power supply connected to a wall outlet for AC power. It was chosen for its compact size, high efficiency, and low cost. This power supply can sustain a load of up to 600 watts at 24 volts and up to 25 amps. Connected to the power supply is a TPS54308DDCR 24 to 3.3 volt buck regulator, and connected to the buck regulator is a TPS613222ADBVR 3.3 to 5 volt boost regulator. The motor controllers are connected to the power supply and other components are connected to our regulators as their voltage demands require.

J. Software

The software consisted of using the Robotic Operating System 2: Humble distribution (Shortened to ROS), and uniquely written code for the design. The ROS library gave us plenty of open-source, pre-built packages to use when simulating our robotic arm within our computer node. Our uniquely written code consisted of functions that drive the motors, passed encoder/sensor data, provided functionality for each tele-operative button to interact with the system, and sorted through CAN bus data by frame. Detailed explanation of the software is discussed in the more appropriate section V, Software Detail.

III. System Functionality and Diagrams

To completely understand the way in which this design functions, this section provides physical diagrams and flowcharts to visually represent our robotic arm.

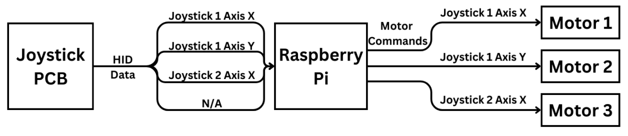


Fig. 1. A block diagram illustrating the communication pathway between the joystick PCB, the Raspberry Pi, and each motor.

The Joystick PCB, equipped with two analog joysticks, sends HID data to the Raspberry Pi, where each joystick axis is mapped to specific motor commands. The Raspberry Pi processes the Joystick 1 X Axis, Joystick 1 Y Axis, and Joystick 2 X Axis inputs, converting them into motor commands for Motor 1, Motor 2, and Motor 3, respectively. This setup provides a direct control link, allowing the operator to adjust each motor's position in real time based on the joystick inputs.

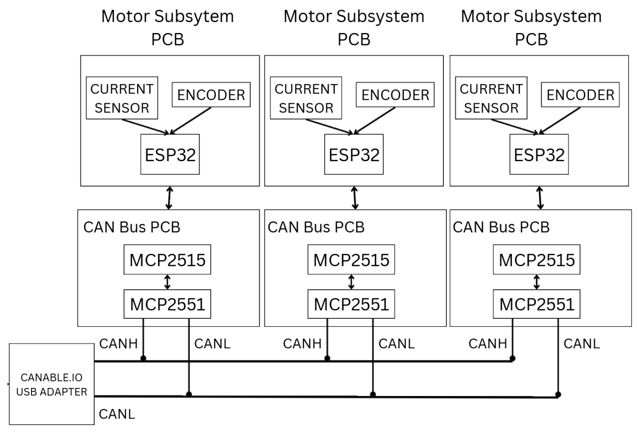


Fig. 2. A block diagram of the connections and layout between the motor driver PCBs and our CAN Bus PCBs.

All of the motor driver PCBs will have our MCUs, encoders, and current sensors attached to them. The MCUs will then allow us to connect our motor driver PCBs and our CAN Bus PCBs. This gives us the ability to pass data from our encoders, sensors, MCUs, and computer.

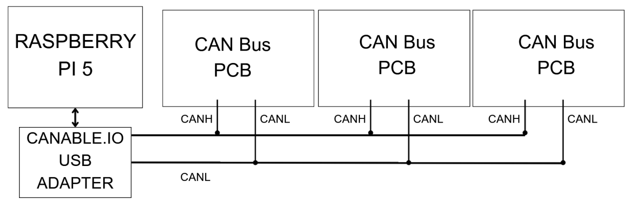


Fig. 3. A block diagram of the CAN bus system communication network as it connects through the CANable.io USB adapter to the Raspberry Pi.

The CAN bus communication system must allow data to flow between the microcontrollers and our computer. Understanding how each component connects is crucial in understanding the design. The above figure demonstrates how each MCP component is connected with the MCUs, and is in turn connected with the Raspberry Pi via CANH and CANL wires. These data lines allow us to pass information from our Raspberry Pi, to our CANable.io USB adapter, to our CAN Bus PCBs, thus driving our motors.

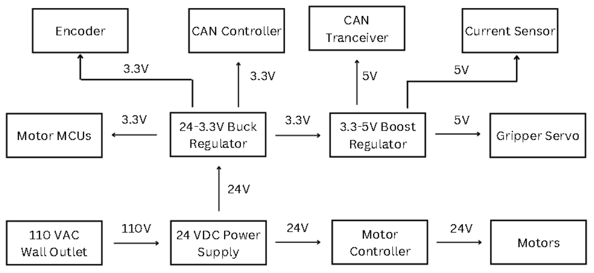


Fig. 4. A block diagram which demonstrates the power distribution of power from the main power supply to other components. .

As shown in this diagram, the main 600 watt power supply is supplied 110 volts in AC by a wall outlet and converts that to 24 volts of DC. 24 volts is directly from the power supply to the motor controller which passes 24 volts to the motors themselves. The 24 to 3.3 volt buck regulator is connected to the power supply as well and directly powers the motor MCUs, encoders, and CAN controllers at 3.3 volts. The 3.3 to 5 volt boost regulator is connected to the buck regulator. and powers the CAN transceivers, current sensors, and the servo for the gripper.

Not pictured in this diagram is the Raspberry Pi 5, which is powered separately by an off-the-shelf 5-volt DC power adapter connected to a wall outlet. This dedicated power source was chosen to ensure stable operation, as the Raspberry Pi is a delicate and valuable component in the system.

IV. Hardware Functionality

This section explores in technical detail the hardware mentioned in section II, System Components. The CANable.io USB Adapter has not been included due to it being redundant, as its functionality is included within the CAN bus communication. [This section may need certain parts removed that are not needed/are redundant. Please add that to this header in a professional manner if you remove a hardware component.]

A. Tele-Operative Devices

The tele-operative joystick PCB functions as the primary user interface, transmitting control signals from the operator to the robotic arm through the ATmega32u4's HID capabilities. Configured as an HID device, the ATmega32u4 transmits real-time joystick and button data to the Raspberry Pi over USB. The microcontroller was booted to emulate an Arduino Leonardo, allowing the Raspberry Pi to recognize it as a USB input device, which facilitates low-latency communication.

Each joystick provides analog input data along X and Y axes, which the ATmega32u4 samples and sends as HID-compliant signals. The Raspberry Pi interprets these signals as positional commands, converting them into motor instructions that are then sent to the MCUs controlling the arm's motors. This setup enables the operator to control the arm's positioning with precision.

The three auxiliary push buttons are configured on digital I/O pins with external pull-down resistors to ensure they default to a low state when not pressed. These buttons enable essential operational functions, including recording/replaying movement sequences, an emergency stop, and end-effector actuation. Button states are sent to the Raspberry Pi via HID signals, where they are interpreted as control commands.

Two LEDs on the board provide visual feedback: the power LED indicates that the board is active, while the GPIO-connected LED serves as an operational status indicator used in troubleshooting. A fuse is placed between the micro-USB connection and the rest of the circuit, offering overcurrent protection to safeguard components in case of a power surge. The board's decoupling capacitors, positioned close to the ATmega32u4, ensure stable operation by filtering voltage fluctuations. The overall design layout minimizes signal interference and supports reliable data transmission, meeting the project's requirements for responsive and stable tele-operative control.

B. Motors

The NEMA 23 stepper motors control each joint's position in the robotic arm, enabling precise movements in the x, y, and z planes. Each motor is controlled by the ESP32 microcontrollers, which receive commands from the Raspberry Pi via the CAN bus. Micro-stepping enhances the motors' precision, essential for delicate manipulations with the gripper end-effector. This setup provides the reliability and accuracy required for tele-operated and autonomous control.

C. Current Sensors

The ACS712-05B current sensors are integrated in series with each motor and its corresponding motor controller, enabling real-time current monitoring across all motor loads. Each sensor's analog output is connected directly to an ADC pin on the ESP32 microcontroller located on each motor's PCB. This configuration allows the ESP32 to sample the output voltage, which is proportional to the current drawn by the motor.

The ESP32 processes the sampled voltage data by applying a scaling factor specific to the ACS712-05B, which has a sensitivity of 0.185 V/A. This sensitivity enables the system to detect and respond to small changes in current, providing precise load monitoring. The ESP32 calculates the instantaneous current based on these readings and applies calibration offsets in software to correct for any baseline deviations.

The current data from each ACS712-05B sensor is then transmitted to the Raspberry Pi via the CAN bus, where it is used to monitor motor load and ensure the system remains within safe operating limits. This setup also enables dynamic adjustments; if an unexpected increase in current is detected—indicating a potential motor stall or mechanical obstruction—the control system can respond accordingly to prevent damage.

D. Position Sensor

Mounted directly on each motor's output, the AS5600 encoders allow for accurate positional feedback without interference from mechanical backlash. By providing absolute positioning, these encoders allow the system to track each joint's position accurately. The ESP32 microcontrollers read this data over I2C and send it via the CAN bus to the Raspberry Pi, where it's integrated into the feedback loop to ensure precise, responsive control during operation.

E. Microcontrollers (ESP32-Wroom)

One of the key components to process and send data, our microcontrollers act as a central hub of information and commands for our design. Using the ESP32-Wroom IC chips, we were able to sort and convert to useful data the information sent over from the encoders and sensors. We were also able to send drive commands to our motors, giving the operator full control over the arm's movement by using the ESP32's robust control over communication networks.

The encoders, motor controller, and CAN connections are connected via the ESP32's digital pin lines. With the exception of the encoder's SDA/SCL (I2C) lines, these pins are interchangeable. The current sensor relies on the ADC pins, and are not interchangeable. For our design, we have chosen pin 26 for our current sensor. This pin choice is arbitrary, as many other GPIO pins (with the exception of voltage, ground, and I2C lines) are considered viable ADC lines. For our encoder, we have chosen pins 21 and 22 for the SDA and SCL lines respectively. These are the default I2C lines for the ESP32, and cannot be exchanged with other pins. The motor controller's connections are set to pins 25, 32, 33 for pulse, direction, and enable. All of these are digital pin connections, and can be interchanged in the same way the pinout for our current sensor may be. Finally, the CAN bus must be connected specifically to the ESP32's SPI connections. We have attached our MOSI-MOSI (GPIO 23), MISO-MISO (GPIO 19), CS-CS (GPIO 5), 3.3V-VIN, and SCK-SCK (GPIO 18). This leaves our chip select pin at GPIO 5.

With all of these connections, our ESP32s easily pass data from our encoders and current sensors to our Raspberry Pi 5. Our CAN bus connections allow the microcontrollers to take in tele-operative input and drive the motors accordingly.

F. CAN Bus Communication

Using MCP2515 controllers and MCP2551 transceiver modules, we were able to design boards that pass and receive data across CANH and CANL data lines on each board while mitigating electromagnetic interference that most other communication lines would struggle with. The MCP2515s are the primary communicators of the CAN system, alongside our ESP32s, enabling communication pathways between our microcontrollers and our MCP2551 transceivers, which prepare the data for sending across our data lines at 5V. While the MCP2515 operates both at 3.3V and 5V, the MCP2551 only operates at 5V. Given that our ESP32s can only operate at 3.3V, our MCP2515 controllers act as a middleman, with a current divider in between the RXD lines of both components, successfully circumventing any possible current issues that could burn our more sensitive microcontrollers. This current divider consists of a 10k-Ohm resistor between the MCP2515 RXD line and the MCP2551 RXD line, and a 22k-Ohm resistor connected on the MCP2515 RXD side to ground. Without this, our RXD line on our MCP2515 could cause a part burnout, due to the way the RXD line can raise the value of VDD above 5V when in operation with different voltages between the two components.

All CANH and CANL wires must be twisted pairs within the system to utilize the unique canceling of electromagnetic interference that the CAN bus offers. This works by using square waves for both CANH and CANL data lines, which forms a differential pair and allows enhanced immunity to noise across all bus operations. Each line is designed to have an equal impedance, which means that while transmitting bits, these methods of communication are far more stable and reliable. Within the CAN system there are many built-in ways that the system corrects bit errors and handles overflow that take the strain off of programmers to do themselves on a software-side. Having our data lines set up in this manner allows the creation of more CAN nodes extremely easy to implement, as we would only need to build another CAN board and then attach its data lines to our CANH and

CANL wires. We are only using four CAN nodes for this design, however we could easily include more if more joints for our robotic arm were necessary.

Each piece of data sent over the CAN bus is sent as an individual frame. These frames include an 11-bit identifier, data length, the data being sent, CRC bits, and an acknowledgement frame. All of these pieces allow us to create a decentralized framework, where every node acts as the master and no node has to wait for availability to begin communication. The inclusion of an identification frame makes it very accessible to organize and determine which nodes will receive which CAN messages, as well as which nodes will send which messages. This makes communication between our CAN nodes and our computer especially easy, since it is possible to use a different identifier for every node and simply send a CAN frame from our Raspberry Pi with the needed identifier.

Since all messages require an acknowledgement frame back from the receiving node, to properly create a CAN system there must be at least two nodes available, along with a proper 120-Ohm termination resistor, otherwise sending messages is not possible.

H. Computer (Raspberry Pi 5)

The Raspberry Pi 5 performs the system's inverse and forward kinematics calculations, determining the necessary motor positions to achieve the desired endpoint. Acting as the control hub, it receives joystick input from the user interface and sends precise commands to each joint via CAN bus, ensuring seamless and responsive control. The Raspberry Pi also monitors feedback from the current sensors and encoders, adjusting motor commands in real-time for accurate tele-operation and collision detection.

I. Power Distribution

Using a Nuofuwei 110VAC to 24VDC power supply, we were able to supply a maximum power load of up to 600 watts and 24 volts[2]. The motor controller boards are connected to the power supply and drive the NEMA23 stepper motors at 24 volts with a peak current of 4 amps[3]. Since there are three stepper motors for the three axis of the manipulator's movement, the peak power draw from a single motor controller is 96 watts and 288 watts across all three. The device that was selected to decrease the voltage from 24 to 3.3 volts is a TPS54308DDCR buck regulator. This device can step down from 24VDC to 3.3VDC with efficiency between 85% and 90%, with up to 3 amps of output current[4]. While this component can output between 1 and 26 volts, our 24 to 3.3 step down regulator board outputs 3.3 volts. As a result the maximum load a single one of our buck regulators can supply is 9.9 watts. Powered by the buck converter at 3.3 volts are the encoders, CAN controllers, and motor MCUs. The MA702GQ-P encoders have a typical supply current of 12mA, and a power draw of 0.0396 watts each. The MCP2515-I/SO CAN controllers have an operating current of 10mA and power draw of 0.033 watts[5]. The motor MCUs have an operating current of 80mA and a power draw of 0.264 watts[6].

The regulator that was selected to increase the voltage from 3.3 to 5 volt is a TPS613222ADBVR boost regulator. It has a fixed output voltage of 5 volts and has a maximum output current of 2.5 amps[7]. Therefore the maximum load power of the boost converter is 12.5 watts. However, the efficiency of the regulator drops off to 75% at 1. Although, it would not be able to achieve that output if it were powered by a single buck regulator. Powered by the boost regulator are the current sensors, CAN transceiver, and the end effector servo. The ACS723LLCTR-10AU-T current sensor has a max supply current of 14mA and a power draw of 0.07 watts[8]. The MCP2551-I/SN CAN transceiver has a supply current of 75mA and a power draw of 0.375 watts[9]. Our parallel gripper end effector is operated by a HS-311 servo. While stalling, the servo reaches its maximum current of 700mA for a peak power draw of 3.5 watts[10]. It's notable that only one boost regulator is powering the end effector. The other two only support a CAN transceiver and current sensor so the power drawn by them will be lower. Between all of the 5V connections, there is a maximum power draw of 3.994 watts which fits well within the maximum load of the buck regulator that supplies it. All of the 3.3V connections including the boost regulator at 75% efficiency add up to 4.9297 watts, which is well below the capacity of the buck regulator. So, even with three series of regulator connections for each stepper motor, the capacity of our power supply comfortably exceeds the power demands of our system.

V. Software Detail

To understand the composition of the software, it is important to understand the needed functions of the robotic arm. Using our Raspberry Pi, we implemented a single python script that logged joystick input, executed sending CAN frames, and did the heavy lifting that the MCUs were not capable of processing. The MCUs acted as a direct line of communication between all of our parts, and were responsible for the organization and execution of specific tasks.

A. Microcontrollers and Peripherals

The most important function of our microcontrollers was to receive data from our computer. This was achieved with the use of the Arduino IDE, the MCP2515 library provided by auto-pwm, and the AS5600 library by Rob Tillart for our encoder. Besides basic headers and pinouts provided by these libraries, all of the code was written from scratch. Each MCU (motor node) is given a unique CAN ID to pass off to the Raspberry Pi. This allows our computer to organize the data it receives from each node, and send the same data back to the corresponding node. Organizing the data like this made passing and parsing information efficient and clean, lowering the response time from our system.

Encoder and sensor data is checked regularly from our MCUs, and only sent through as CAN frames if a change is detected in either. The encoders allow us to detect the position of our motors, giving us the ability to reset them to default when running through our record/replay functionality, emergency stop functionality, and force-feedback functionality. The current sensors are essential in detecting when a collision occurs, as a particular spike indicates resistance and triggers our function, which in turn lowers the speed of operation until the robotic arm comes to a complete stop.

Each node sends data back based on the CAN ID of the data. Similarly, they receive specific joystick data by reading the ID of the CAN frames that they receive.

On our Raspberry Pi, we are able to continuously take input from our joystick and send it as individual frames to the correct nodes. Implementing a delay for each frame was needed to make the system function entirely as expected, otherwise it was very likely our CAN frames would run into each other. Button inputs are detected and handled accordingly by the MCUs, however the Raspberry Pi is used to poll button timing, as well as timing for frames when recording movement that the operator wishes to replay. This allows us to get multiple functionalities out of the same buttons, such as resetting recorded data based on long-presses of our record/replay button, or resetting our arm to the default position.

At its most basic, the Raspberry Pi is used to calculate complex mathematical and timing-based processes that the MCUs would otherwise not be able to handle, or would cause a significant amount of lag if offloaded to the ESP32 processors. The MCUs are used to carry out basic functions and commands, which primarily focus on the exchanging and organizing of data, as well as driving hardware such as the motors.

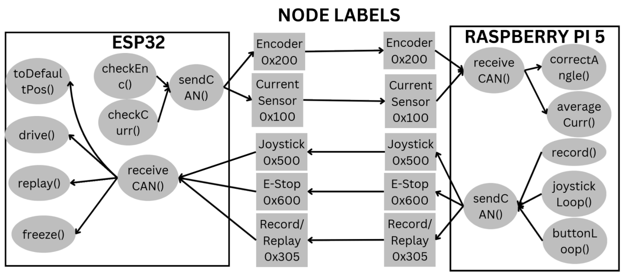


Fig. 5. A flowchart demonstrating the relationship between our MCUs, current sensors, encoders, CAN bus, computer, and joysticks.

This diagram shows the basic functions, node labels, and how each MCU communicates/handles data. The graphic also demonstrates how the Raspberry Pi handles and sends data to our MCUs, including the CAN IDs that it recognizes.

VI. Board Details

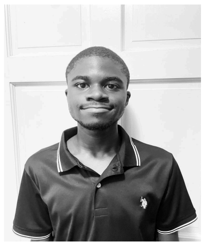
There were several PCB boards that were designed in order for the entire robot arm to operate. Firstly, we designed PCBs that included the 24-3.3V Buck regulator and the 3.3-5V Boost Regulator to give proper voltage to each component of the robot arm. Next, there were boards created that consisted of the MCU, a logic level shifter, and switch buttons. The purpose of this board is to manage motor drivers and control the servo motors that power the robot arm's joints. Moreover, we created a PCB that consisted of the CAN Bus controller and the CAN Bus transceiver, which would allow different parts of the robot arm to communicate efficiently, ensuring that messages are exchanged quickly and reliably. Finally, a joystick PCB needed to be added in order for the user to have the ability to manually control the robot arm's movements.

VII. The Engineers

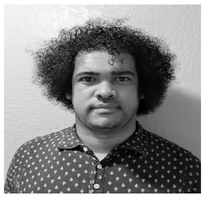
Hailey Gorak is a 23-year-old graduating Computer Engineering student at the University of Central Florida. Hailey hopes to pursue a career working with embedded systems within the medical or automotive field, and specializes in communication systems.

Emanuel Alvarez is a 22-year-old graduating Electrical Engineering student with a minor in Computer Science at the University of Central

Florida. He hopes to pursue a career in the power industry or in software development, combining his skills in both hardware and software to make a meaningful impact.



Jackson Jacques III is a 25-year old graduating Electrical Engineering student at University of Central Florida. He hopes to pursue a career working designing, developing, and testing electrical systems and equipment, such as motors, sensors, and communication systems.



Nkunu Nuglozeh is a 24-year old graduating Electrical Engineering student at the University of Central Florida specializing in power. He hopes to pursue a career in working with and developing solutions for power generation, electrical systems, and motors.

Acknowledgement

The authors wish to acknowledge the assistance and support of Dr. Chung Yong Chan, Dr. Arthur Weeks, Dr. Mike Borowczak, Dr. Varadraj Gurupur, and Mark Maddox: University of Central Florida.

References

[1] Sharma, S. (2023, November 7). *Understanding can bus: A comprehensive guide*. Wevolver. https://www.wevolver.com/article/understanding-can-bus-a-comprehensive-guide
[2] LRS-600-MEAN WELL Switching Power supply Manufacturer. (n.d.). https://www.meanwell.com/webapp/product/search.aspx?prod=LRS-600
[3]NEMA 23 Stepper Motor 2.4nm(339.87oz.in) 4A 57x57x82mm 8mm shaft 4 wires. StepperOnline. (n.d.). https://www.omc-stepperonline.com/nema-23-stepper-motor-2-4nm-339-79oz-in-4a-57x57x82mm-8mm-shaft-4-wires-23hs32-4004s?srsltid=AfmBOopD99CtNC88G69JwqaJqt0ZeOiT7RU1RfOtn1zMlZiUQCtNF_rG
[4] Texas Instruments, "TPS54308 4.5-V to 28-V Input, 3-A Output,
Synchronous 350-kHz FCCM Step-Down Converter in SOT23 Package," TPS54308 datasheet, Jun. 2017 [Revised Apr. 2021]
[5]Microchip Technology Inc. (2003). MCP2515 [Technical documentation]. https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf
[6]Espressif Systems. (2018). ESP32-WROOM-32 (ESP-WROOM-32) Datasheet.
[7]Texas Instruments Incorporated. (2024). "TPS61322 6.5-µA Quiescent current, 1.8-A switch current boost converter,"TPS61322 datasheet January 2018 [Revised Feb. 2024].
[8]High-Accuracy, Galvanically Isolated Current Sensor IC with Small Footprint SOIC8 Package. (2024). In Allegro. https://www.allegromicro.com
[9]Microchip Technology Inc. (2001). MCP2551 (pp. 1–5). https://ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/DataSheets/20001667G.pdf
[10] DFRobot. (2018). LG-NS Robot Gripper (With Servo). In DFRobot. https://www.dfrobot.com/product-1169.html