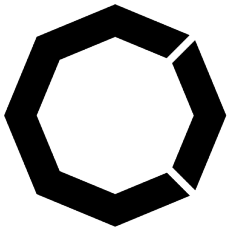


CalPolyPomona

College of Engineering

Senior Design Project

Robotic Arm Control with Computer Vision

Noah King
Junhee Lee

Anas Salah Eddin
EGR-4830
Spring 2025

Department of Electrical and Computer Engineering

May 27, 2025

Abstract

This paper presents the design and implementation of a computer vision based robotic arm system for real-time person tracking. The system has the potential to enhance workplace safety in hazardous environments, particularly in heavy industries like steel manufacturing and petroleum refining, where accurate personnel tracking is critical for accident prevention. By autonomously detecting and tracking human presence, the system can contribute to improved operational awareness and accident prevention in high-risk settings. In light industries, such as film production, such an arm can be used on prefabricated movie sets in order to track actors/actresses as they move across the film set. Additionally, in consumer settings, a robotic arm can be used to track a person's face so one could perform other tasks while moving across the room during a video conference.

Regarding the mechanical design of a robotic arm, a number of different designs have been successfully used in a variety of industries. Two-degree-of-freedom (2-DOF) pan-tilt systems have been successfully implemented security camera systems, while three-degree-of-freedom (3-DOF) designs are widely used in manufacturing, particularly in camera-guided "pick and place" robotic arms on assembly lines. This paper focuses on the development and implementation of a custom 3-DOF robotic arm.

These type of tracking systems are typically implemented using an application processor (to run the detection algorithm), and a micro-controller (to actuate motors based on data from the detection algorithm). We opted for a straightforward implementation utilizing standard inter-board communication protocols such as UART and USB. The system operates in a master-slave configuration, with a microprocessor handling high-level algorithms while motor control is delegated to a microcontroller equipped with the necessary I/O interfaces as well as dedicated integrated circuits to support this.

We found that classical computer vision algorithms such as Viola–Jones were well-suited for the task of detection, offering significantly lower computational requirements compared to modern machine learning-based methods. This makes Viola–Jones particularly advantageous for deployment on resource-constrained embedded Linux systems, where running more demanding machine learning models like "You Only Look Once" (YOLO) or "Single Shot Detector" (SSD) may not be feasible. Additionally, our motor control system demonstrated strong performance at reasonable ranges, such as ranges typically used in video conferencing. While our implementation of motor movement did not dynamically adjust speed based on the distance between the detected face and the center of the frame, it still provided consistent and predictable tracking behavior. Additionally, although the 3D printed robotic arm design exhibited some mechanical vulnerability under rough handling, resulting in minor slack that slightly diminished the smoothness of movement. In light of the mechanical vulnerabilities, we do not believe the system is suitable for rougher industrial settings. Despite this, it remained functional and well-suited for controlled environments. Overall, the system demonstrated strong potential for applications such as facial tracking in consumer video conferencing, where conditions are less demanding.

Contents

1 Introduction

This paper will detail the design and implementation of a three-degree-of-freedom (3-DOF) robotic arm controlled by the "Viola & Jones" Haar Cascade Detection algorithm. Put simply, the robotic arm will track the users face as he moves across the camera frame. In more detail, the system first identifies the face closest to the center of the camera frame then actuates the motors to align the face with the frame's center. Figure 1 illustrates this behavior, where D is the pixel distance of the nearest face from the screen center.

The algorithm begins by capturing a single frame from the camera's frame buffer. This image is then processed using OpenCV's implementation of the Viola & Jones object detection algorithm, which efficiently identifies human faces within the frame. The output of this process is a list of detected faces, each represented by its position in Cartesian coordinates as an x, y pair, corresponding to its location within the image.

Next, the algorithm computes the distance from each detected face to the center of the frame. This allows the system to determine which face is most centrally located. All other detected faces are disregarded to ensure the robotic arm focuses on a single target. If the closest face is not within a defined threshold—specifically, 20 pixels from the frame center—the system proceeds to initiate a repositioning operation.

The application processor calculates the direction of movement required to re-center the target face. This is determined by examining the sign of the x and y offsets from the center. Based on this analysis, a command is constructed and transmitted via the UART interface to the STM32 microcontroller. Upon receiving the command, the microcontroller actuates the appropriate motors to adjust the position of the robotic arm, thereby aligning the camera so that the face remains centered in the frame.

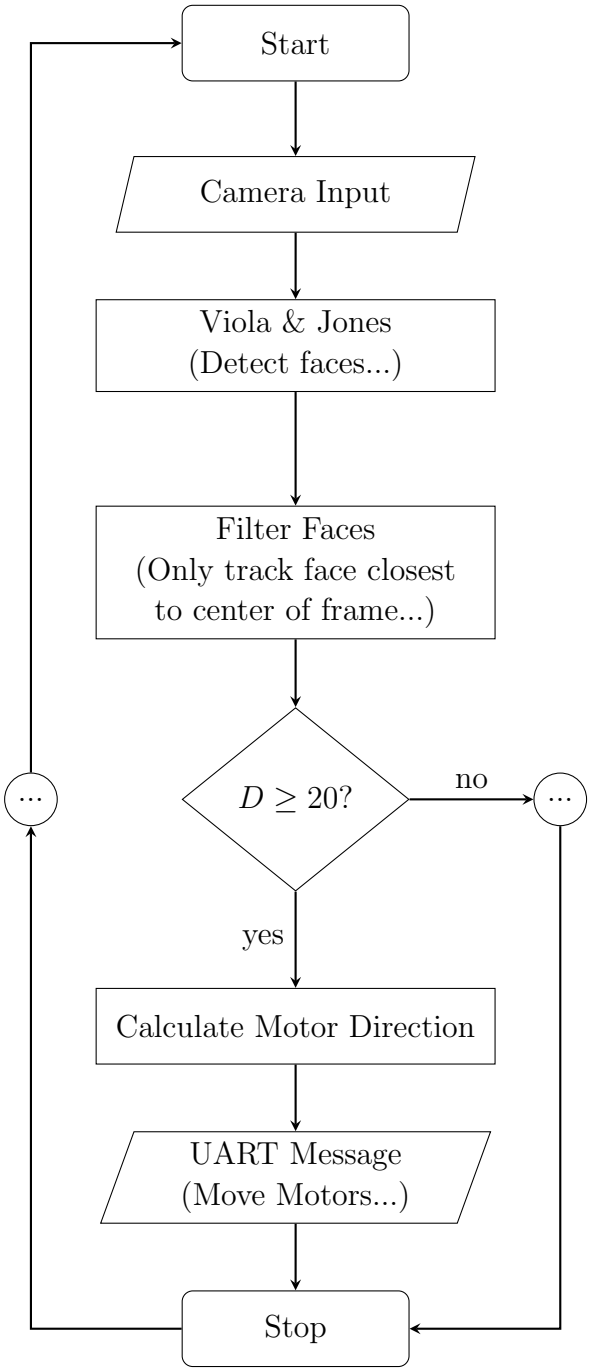


Figure 1: Face Detection Logic

2 Background

2.1 Applications of Robotic Arms & Computer Vision

Robotic arms have played a pivotal role in automating the manufacturing sector, especially in heavy industries such as automotive production. In practice, these robotic arms are generally not human operated, but controlled by advanced algorithms that interface with a variety of sensors in order to perform the job of a human on a Henry Ford style assembly line. The proliferation of high quality computer vision algorithms, such as those provided by the OpenCV project, has allowed these robotic arms to mount cameras to perform even more functions besides assembly, such as quality control procedures, equipment failure detection, workplace safety violations, etc. In the context of workplace safety violations, computer vision can be an inexpensive way to provide a method for robotic arms to track personnel as they move across a work area and intelligently notify dangerous equipment, or the person themselves, in order to prevent workplace accidents. Furthermore, computer vision face tracking can be used to detect the presence of requirement equipment such as safety glasses, respirators, and hard hats for highly toxic industrial environments. Additionally, computer vision face tracking or person tracking can be used for consumer applications such as person tracking in the context of video conferencing.

With sufficient context established, the aim of this project is to develop custom electronics hardware and control software to drive a robotic arm using computer vision. This endeavor will serve as an exercise in cross-disciplinary skills, encompassing PCB design, SMT assembly, embedded firmware programming, and computer vision programming, all within the context of low-volume manufacturing and cost optimization, taking into account existing economic factors and market dynamics.

3 Requirements and Specifications

In our design of the robotic arm, we had a set of guidelines to adhere to when designing our system.

1. The Mechanical Parts for the Robotic Arm should be readily available in reasonable quantities from vendors.
    - Must be Standardized Metric Parts
2. The electronic components for the Motor Control Board of the robotic arm should be readily available in reasonable quantities from vendors.
    - The circuit design for the Motor Control Board must be optimized to fit within the dimensions of 200mm x 150mm x 2mm.
    - The circuit design should provide a UART interface, either through a USB-UART bridge or direct UART connection.
    - The PCB design should use standard 2.54mm pitch pin headers or sockets, use standard 2.54mm screw terminal connectors, and a micro-usb connection.
    - Must be Standardized Metric Parts
3. Excluding the price of manufacturing tooling, the price of an overall complete robotic arm should not exceed $150.00.
4. The robotic arm should be able to mount a common USB webcam.
5. The robotic arm should be able to controlled through a computer vision face tracking system or through a manual control input system.

4 Design

4.1 Top Level Overview

The robotic arm tracking system consists of an application processor that communicates with a microcontroller on a PCB[1]. This microcontroller controls motor driver ICs, which, in turn, send electrical signals to the stepper motors. The complete project can be found at the GitHub link provided in entry [4] in the references section of this report. Figure 2 depicts the software architecture for our robotic arm control system. At the core, there are two processing entities communicating over a serial (UART) interface:

1. **STM32 Microcontroller (Firmware Layer)**: Responsible for receiving simple motion commands (e.g., move left, move right, move up, move down) and translating these commands into step pulses for the stepper motor drivers. It also handles endstop detection, homing sequences, and microstepping configuration.
2. **Application Processor (Host Software Layer)**: Runs a Python-based application. This application uses OpenCV to detect and track faces in real-time (or capture joystick input for manual control). The processed position data is converted into high-level "MOVE" commands, which are then sent to the STM32 over UART.

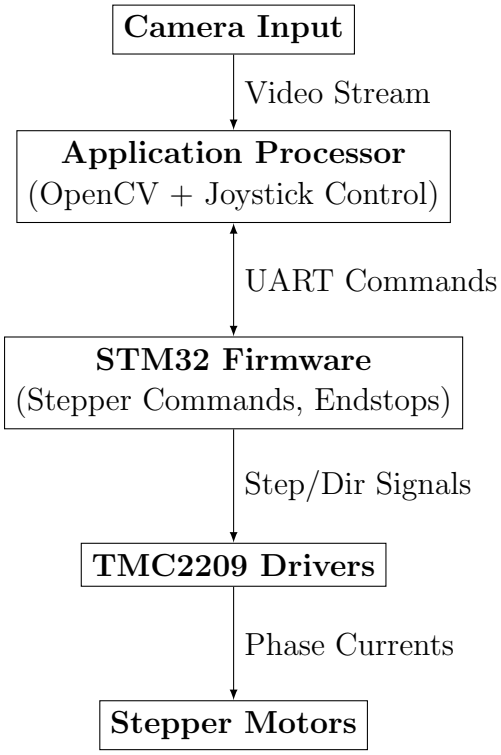


Figure 2: System Block Diagram

The above division of tasks depicted in Figure 2 ensures that the computationally heavier tasks (e.g., computer vision or real-time user input processing) are offloaded to the Application Processor, while the embedded microcontroller strictly handles lower-level motor control.

4.2 Electronic Component Selection

The final mainboard's electronics are divided into analog and digital sections. The analog portion consists of power regulation circuitry that steps down a 12V, 5A input for the stepper motors to generate a 3.3V rail for the digital logic. The digital subsystem includes an STM32 microcontroller, FTDI USB-to-UART bridge IC, and TMC2209 stepper motor driver for control, communication, and motor actuation. The power regulation circuitry is isolated on a daughter board soldered to the mainboard, which is a custom two-layer PCB designed to minimize fabrication costs. This section outlines the rationale behind the selection of key components. The complete Bill of Materials can be found in Appendix B.

Component	**Description**
STM32C071KBT3	Microcontroller
TMC2209	Stepper motor driver
FT230XS	USB-to-UART bridge
MINI-560	Step-down buck converter

Table 1: Summary of Key Components[1]

4.2.1 Microcontroller

Practical Considerations. The STM32C071KBT3 was selected for its optimal balance of features and cost, aligning well with the project's requirements. Initial development on a more capable STM32F446RE Nucleo board, featuring a Cortex-M4 core, revealed that only ≈20 GPIOs and a single UART peripheral were necessary. This insight justified a transition to the entry-level STM32C0 series, based on the Cortex-M0 core. The C071KBT3 variant met all functional requirements—sufficient I/O, required communication interfaces, and memory-while significantly reducing complexity and cost.

Institutional Considerations. The choice of an ARM-based microcontroller reflects industry-standard practices. ARM's well-documented instruction set architecture (ISA) and widespread adoption simplify onboarding, maintenance, and long-term support. For institutional projects with potential lifespans exceeding the tenure of individual developers, such considerations are critical to ensuring continuity and ease of future development.

Manufacturer Considerations. STMicroelectronics was chosen for its robust development ecosystem, including STM32CubeIDE and the STM32 HAL, which streamline peripheral integration and reduce boilerplate—ideal for rapid prototyping and production with an ≈8 month development cycle. Additionally, entry-level chips, like the one used in this project, are reliably in stock from US vendors like Digikey and Mouser, unlike competitors' offerings.

[1]See Appendix B for full Bill of Materials

4.2.2 Physical Communication Interfaces

USB to UART Bridge IC. Our final design utilizes a USB Micro-B connector which connects the USB differential pair to an FTDI FT-230XS USB to UART Bridge, which then communicates to the microcontroller's on-chip UART peripheral. Initially, our design intended to utilize the native USB functionality of the STM32C071KBT3 to simulate a console environment using ST Micro's USB VCOM implementation found in the Hardware Abstraction Layer (HAL). Due to layout constraints when routing the PCB, to simplify development, we opted for a more classical solution utilizing a USB to UART bridge to facilitate communication between the microcontroller and external application processor. This concession was made to make the placement of the physical connectors on the mainboard more ergonomic for technicians to wire the board to external motors and end-stop switches. The inclusion of this IC increased the per-unit cost of the assembled board by $2.26; however, this cost can be significantly reduced through economies of scale, a consideration typically addressed in Engineering Economics (however this is irrelevant to the low volume nature of our project).

4.2.3 Power Electronics

MINI560 Step-Down Buck Converter. The motor control mainboard utilizes the following daughter board, the "MINI-560" Step-Down Buck Converter in order to step down a 12V rail used for motor power, to a 3.3V rail capable of safely powering the digital subsystem consisting of the microcontroller, USB to UART Bridge IC, and 3.3V logic of the Mixed-Signal Stepper Motor Driver ICs. A Step-Down topology was chosen as the voltage difference between 12V and 3.3V was too large for a Linear Dropoff Regulator (LDO), which would generate significant heat and eventually burn out if used in a design such as ours. The particular MINI-560 module was used as it was a module we could use early in development when we were prototyping using breadboards. We originally planned to design our own circuit around a buck-converter IC, but realized we lacked a hot-air station to mount a QFN-EP (exposed pad) package IC to the board we already had on hand that utilized through-hole technology for board mounting. As such we opted for the existing daughter board. We understand this is less than ideal for actual production for any serious organization as it's much easier to source ICs and passive components than it is to rely on a third party to have a supply of pre-made daughter boards at an affordable price.

Generic 12V, 5A Power Brick with Barrel Jack. We opted to use a readily available off the shelf power brick to power our project. The amperage was chosen based on our current draw experiments in the prototyping stage of our project which revealed that the digital subsystem of our motor control board draws 300mA when no motors are active. It draws around 2A - 4A when all three motors are active and stalling, well below the 5A limit of our external DC power supply.

4.2.4 Motor Driver Integrated Circuits

The Trinamic TMC-2209 stepper motor driver was selected over competitors like the Allegro A4988 due to its more advanced feature set. It supports 1/64th microstepping—used in our project—which enables smoother and more precise motor movement. Additionally, its StealthChop2 technology greatly reduces operational noise, making it well-suited for applications where quiet performance is important.

4.2.5 Connectors and Wires

Electrical Connectors. A 12V Barrel Jack rated for 5A was chosen for reasons detailed in the previous section. Jumper caps were used to isolate sections of the power circuitry for smoke testing. After populating the PCB, we first powered the 9V rail to verify functionality. Once confirmed, a jumper cap was applied to enable the 3.3V MINI-560 switching regulator, powering the logic-level components. We utilized generic 2.54mm pitch pin headers and sockets as well as 22AWG wire capable of sustaining currents of up to 7A. For the wires attached to the rotating arm assembly, we opted for stranded wire, as stranded wire is more flexible and less prone to snapping when jerked around compared to solid core wire. For the rotating arm assembly, we selected stranded 22AWG wire due to its superior flexibility and resistance to mechanical stress compared to solid core wire. Similarly, 2.54mm 1x2 screw terminal blocks were used for the end-stop switches to allow for easy servicing and to avoid the reliability issues associated with soldered or Dupont-style connectors in high-movement areas.

4.3 Circuit Design

4.3.1 Notes on Revisions

Two circuit schematics were developed: one for the prototype mainboard (REVISION 0) and another for the final design (REVISION 1). There are three primary differences between them. First, REVISION 1 includes a USB-to-UART bridge IC, which REVISION 0 lacks. Second, the pinouts differ, as REVISION 0 was designed for a more costly four-layer PCB process, while REVISION 1 was optimized for a more economical two-layer fabrication. Lastly, REVISION 1 features minor layout changes, including a slightly larger board footprint and repositioned connectors. This paper will only go over the design of REVISION 1, now referred to as "REV 1", as it is the most capable and efficient of the two designs. The full circuit schematic for REVISION 1 is given in Appendix A.

4.3.2 Power Supply Circuit

The circuit shown below in Figure 3 represents the power management block responsible for supplying the required voltage rails for the project. A power brick connects to the barrel jack, delivering 12V directly to the stepper motors. This 12V rail is routed through an SPDT toggle switch (SW1), followed by pins 1 and 2 of the pin header block (J1), which must be shorted using a jumper cap. The voltage then proceeds to the MINI-560 step-down buck converter which supples 3.3V to the control circuitry. The inclusion of the J1 header block facilitates "smoke testing" during initial assembly, allowing for straightforward circuit isolation and debugging post-assembly.

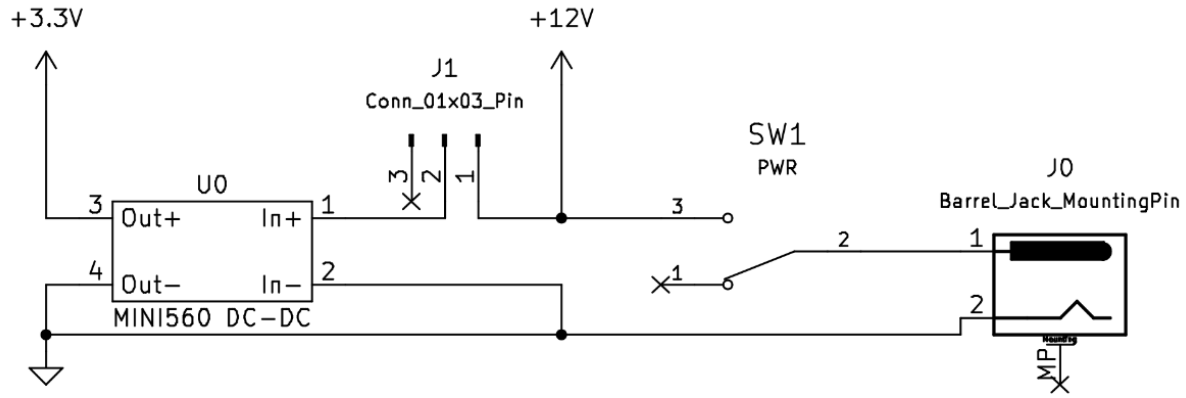


Figure 3: Power Supply Sub-Circuit [1]

[1] See Appendix A for complete schematic.

4.3.3 USB to UART Bridge Circuit

The USB-to-UART bridge circuit was implemented in accordance with the reference design outlined in the FTDI FT230XS datasheet. Nonetheless, a brief overview of the key design considerations is provided here for clarity. Series termination resistors and filtering capacitors are placed on the USB D+ and D− differential lines to mitigate signal reflections and suppress high-frequency noise, thereby preserving signal integrity. A capacitor is connected between the RESET# pin and ground to filter out transient noise and prevent unintended resets. On the VCC rail, capacitors C17 and C18 serve as decoupling capacitors, providing local charge storage and filtering high-frequency power supply noise. A voltage divider composed of resistors R4 and R5 interfaces with CBUS2, which is configured as VBUS_SENSE, allowing the FT230XS to detect the presence of VBUS as specified in the datasheet. Finally, the shield of the Micro-B USB connector is directly tied to ground to reduce potential interference.

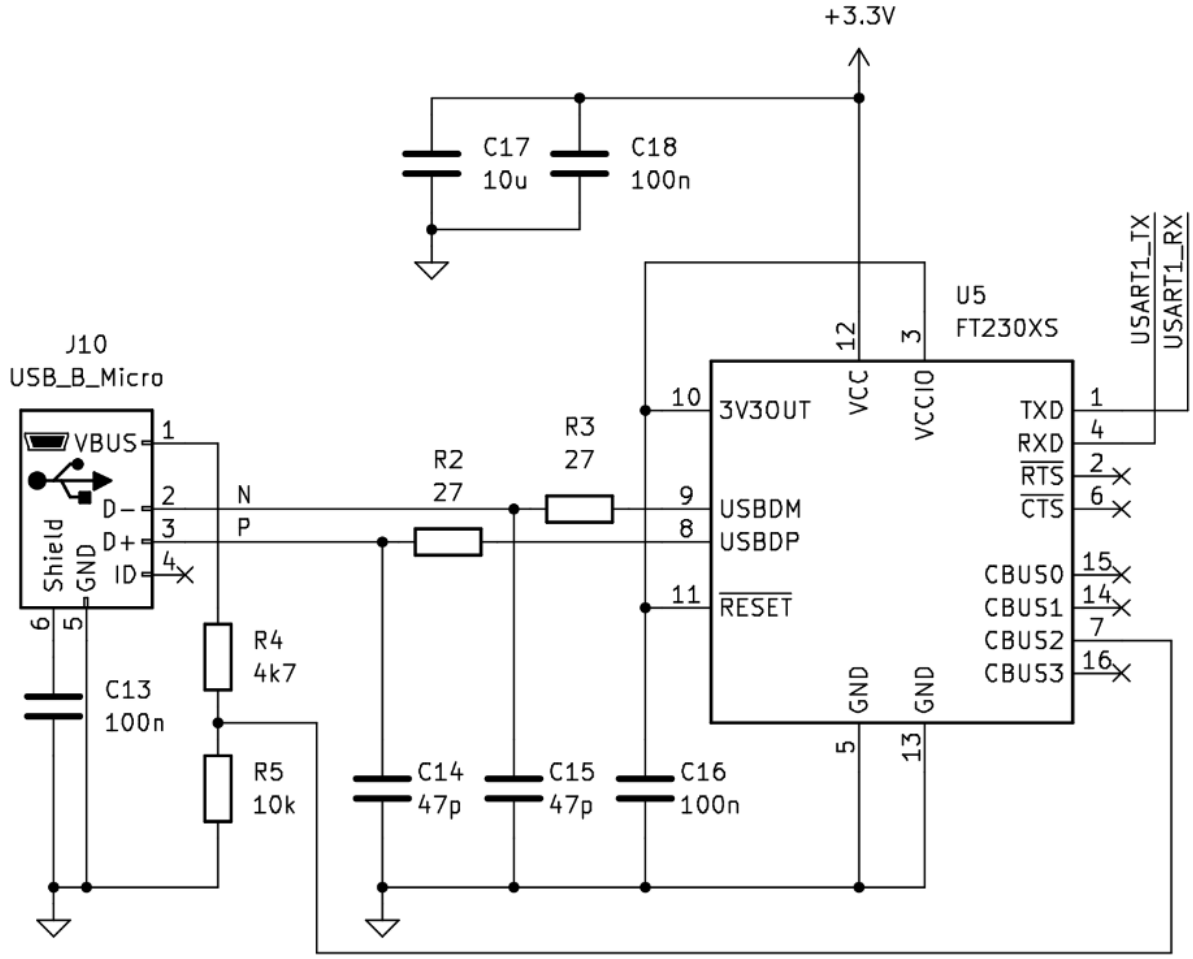


Figure 4: USB to UART Bridge Sub-Circuit[1]

[1]See Appendix A for complete schematic.

4.3.4 Microcontroller Circuit (SWD + External Clock)

The microcontroller sub-circuit, depicted in Figure 5, integrates key elements required for system operation, including power conditioning, an external clock source, a Serial Wire Debug (SWD) interface, and I/O connections to both onboard and off-board subsystems. Power rails are stabilized using a combination of small and large decoupling capacitors positioned near the supply pins. The external clock signal is sourced from an ECS-2520MV integrated oscillator, selected over a discrete crystal and Pierce oscillator configuration to conserve Pin 3 for other functions. Communication interfaces are provisioned through multiple USART peripherals. Pins 7 and 8 are configured as the USART1 interface, which connects to the USB-to-UART bridge IC, enabling serial communication with a host system. Pins 9 and 10 are allocated to USART2, which is routed directly to a two-pin header for auxiliary serial interfacing. The SWD programming interface enables firmware upload and debugging functionality. Pin 17 drives a status/debug LED for visual feedback during operation and development. Finally, physical pins 18–31 are connected to three onboard TMC-2209 stepper motor drivers, while pins 1, 30, and 31 are routed to screw terminals J7, J8, and J9, which interface with off-board end-stop switches mounted on the robotic arm assembly.

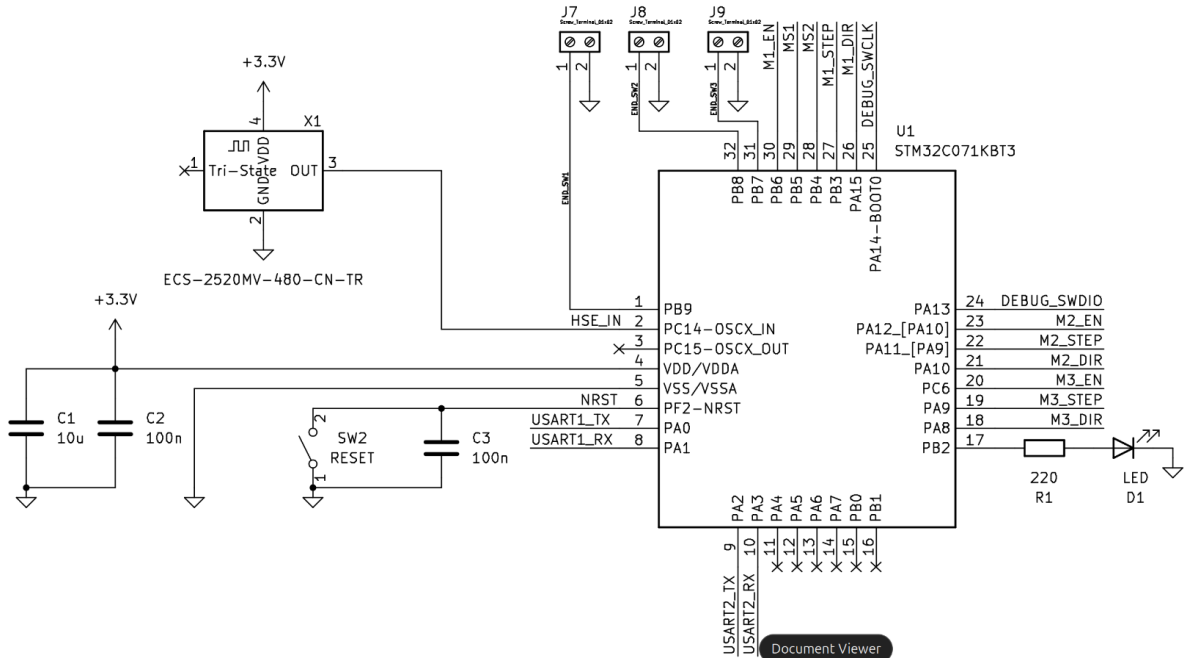


Figure 5: Microcontroller Sub-Circuit (SWD + External Clock)[1]

[1]See Appendix A for complete schematic.

4.3.5 Motor Driver Sub-Circuit

The motor driver circuit, shown below in Figure 6, interfaces the TMC-2209 StepStick module with both the onboard microcontroller and external connectors designated for the off-board stepper motors mounted on the robotic arm assembly. The module's input/output (I/O) pins are routed to the microcontroller to facilitate step and direction control, as well as UART-based configuration and diagnostics. A high-capacitance decoupling capacitor C6 is placed across the motor power input to mitigate voltage transients and suppress electrical noise generated during motor operation. On the logic voltage input, both small and large multi-layer ceramic capacitors (MLCCs) C4 and C5 are employed to filter high-frequency noise and stabilize the supply voltage. This pattern is repeated for motors M2 and M3.

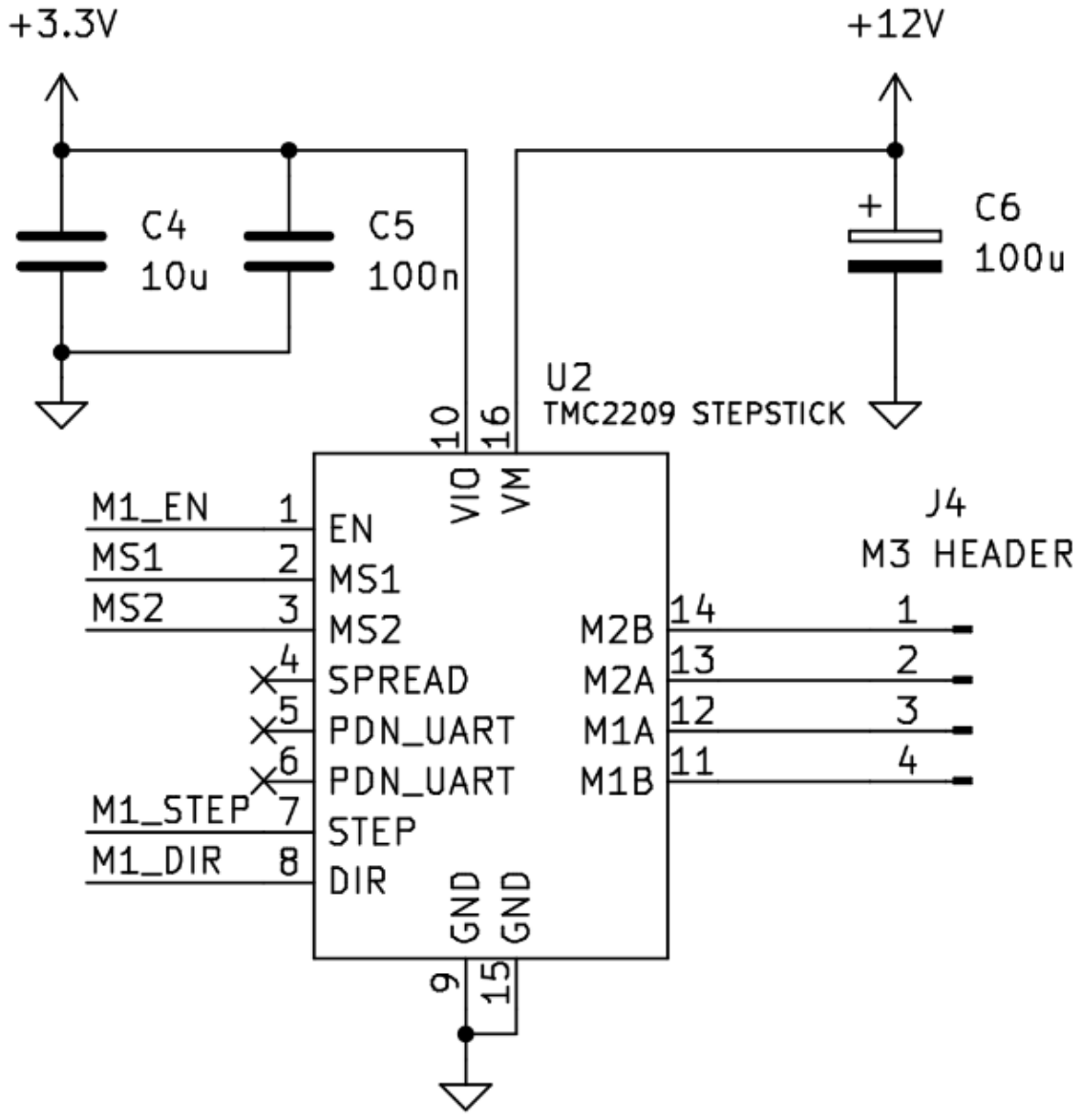


Figure 6: Motor Driver Sub-Circuit [1]

[1] See Appendix A for complete schematic.

4.4 Printed Circuit Board Design

4.4.1 PCB Stackup and EMI Mitigation

Overview. The final circuit design (REV 1) utilizes a 2-layer PCB configuration. The top layer is primarily allocated for power distribution and signal routing. The bottom layer serves as the ground plane; although it includes some signal traces, these are minimized in length and strategically routed to avoid disrupting ground continuity and to mitigate electromagnetic interference (EMI). A 2-layer PCB was selected for REV 1 to optimize cost-efficiency, as the 4-layer stackup used in REV 0 was deemed excessive relative to the signal density and routing complexity of the design.

Top Layer (Power + Signal). Below in Figure 7 is the top layer of the PCB. This layer contains the majority signal traces. Additionally, a ground pour was implemented around the USB-UART sub-circuit to provide shielding for the USB differential pair signal traces. This design choice effectively mitigates potential noise interference from the adjacent power pour, which would otherwise be present without the ground pour. Furthermore, the microcontroller was centrally positioned relative to surrounding components, and pin assignments were carefully selected to minimize signal routing complexity and trace crossings. As expected, higher current traces are thicker, lower current traces are thinner. The trace widths of 0.3mm for signal traces, 0.5mm for 3.3V power traces, and 3

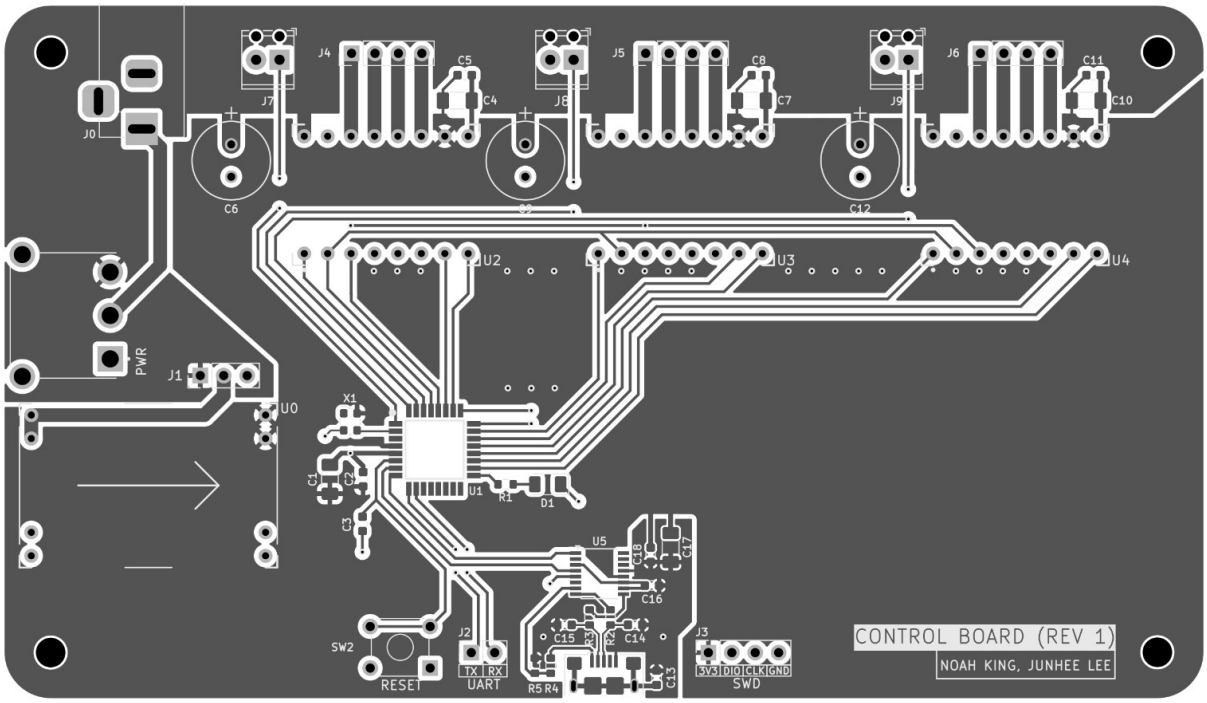


Figure 7: Top Layer of PCB (Power + Signal)

Bottom Layer (Ground + Signal). Figure 8 shows the bottom layer of the PCB. This layer primarily consists of a large ground plane implemented via a copper pour, which serves to minimize electromagnetic interference and provide low-impedance return paths for signals throughout the circuit. It also helps to suppress ground bounce and minimize voltage fluctuations across the board by stabilizing the reference voltage for digital and analog circuits. Two prominent signal traces on this layer are dedicated to the SWCLK and SWDIO lines, which are part of the Serial Wire Debug (SWD) interface for the STM32 microcontroller. These traces are routed away from high-frequency or high-current circuitry to avoid interference and preserve signal integrity along the debug interface. Additionally, shorter traces on this layer connect to stitching vias that provide inter-layer continuity for low-frequency signals. These include the "MS2" configuration lines for the TMC-2209 stepper motor drivers, as well as digital input lines connected to off-board end-stop switches. These switches interface with the PCB through screw terminals J7, J8, and J9. Careful routing and via placement ensure minimal disruption to the ground plane and optimal signal performance.

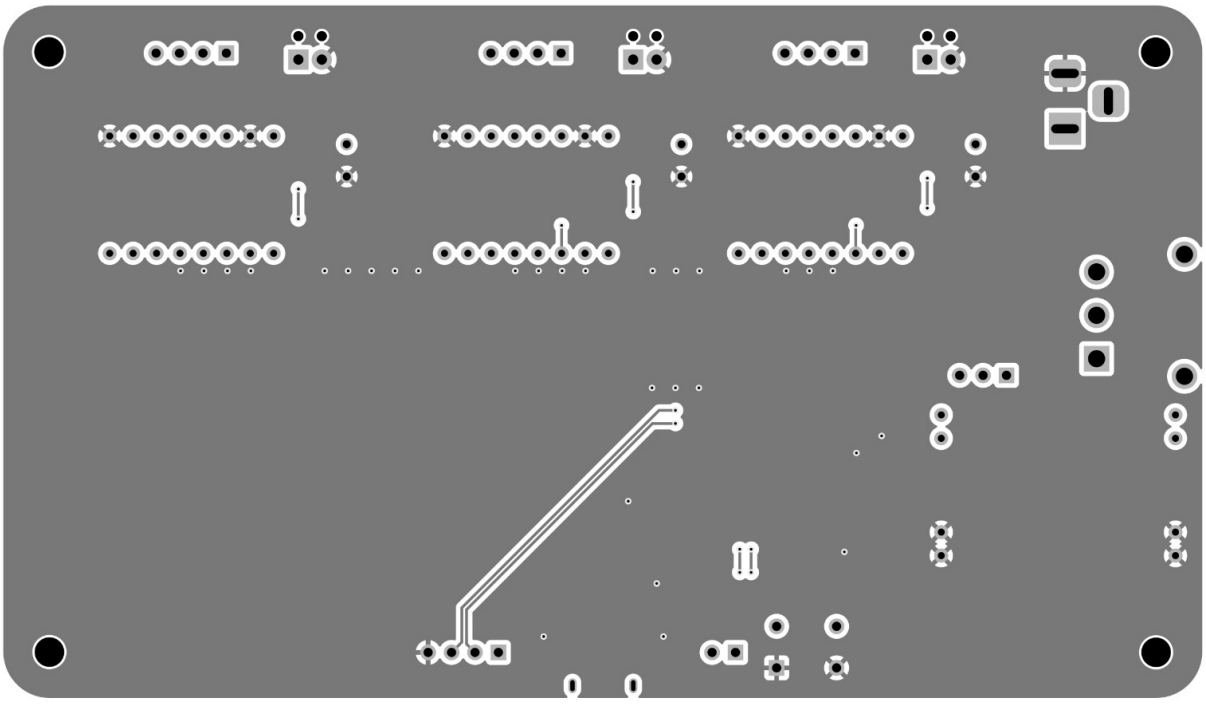


Figure 8: Bottom Layer of PCB (GND + Signal)

4.5 Firmware Design

The STM32 microcontroller firmware [4] performs several core functions critical to the operation of the robotic arm:

1. **Peripheral Initialization:** Initialization functions generated via STM32CubeMX (`MX_GPIO_Init`, `MX_USART1_UART_Init`, `MX_TIM1_Init`) configure hardware peripherals:
   - **GPIOs:** Pins such as `M1_STEP_PIN` and `M1_DIR_PIN` are set as outputs to control stepper motor drivers (TMC2209).
   - **Timers:** Configured for microsecond-level delays via `delay_us(delay)`.
   - **UART:** Enables communication with the application processor for receiving motion commands.
2. **Motor Driver Configuration:** Sets enable, direction, step, and microstepping pins for the TMC2209 stepper motor drivers.
3. **Command Handling:** Continuously listens for incoming UART commands, parsing them using a simple string tokenizer.
4. **Homing Procedure:** Executes routines to zero each motor's position, ensuring consistent and repeatable actuation.

Interrupt-Driven UART Reception. The firmware leverages `HAL_UART_Receive_IT` to handle incoming UART bytes in an interrupt-driven manner. In the `HAL_UART_RxCpltCallback` function, each received byte is accumulated in a buffer (e.g., `rx_buffer`). When a newline character (‘\n‘) is detected, the buffer is terminated, and a flag (`commandReady`) is set to indicate that the complete command (e.g., `"RIGHT,UP"`) is ready for processing:

```
if (temp_buffer[0] == '\n') {
    rx_buffer[idx - 1] = '\0'; // Terminate string
    commandReady = 1;
    // Reset buffer index for next command
    idx = 0;
}
```

This approach prevents heavy polling in the main loop and enables near real-time command processing.

Command Parsing and Stepper Control. In the main loop, when `commandReady` is true, the firmware tokenizes the received command using functions like `strtok()` and passes it to `stepper_process_command`. This function sets the direction pins based on the command and calls `stepper_move` to generate the appropriate number of step pulses:

```
if(strcmp(dx_command, "LEFT") == 0) {
    HAL_GPIO_WritePin(driver_one->DIR_PORT, driver_one->DIR_PIN, GPIO_PIN_RESET);
    move_x = true;
}
```

Homing Routine. The homing function (`stepper_home`) sequentially drives each axis until an endstop is triggered. Once an endstop is activated, the corresponding motor's position is reset to zero, establishing a known reference for all future movements.

4.6 Software Design

4.6.1 Overview

On the application processor side, we leverage OpenCV's Haar Cascade Classifiers (partially derived from the Viola–Jones algorithm) to detect and track faces. The major software blocks include:

1. **Video Capture**: Retrieves frames from a standard USB webcam or system camera using OpenCV's `VideoCapture` API.
2. **Face Detection**: Uses a frontal-face cascade (`haarcascade_frontalface_default.xml`) and optionally profile cascades.
3. **Optical Flow Tracking**: Once a face is found, we apply Lucas–Kanade optical flow to track its center between frames.
4. **Kalman Filtering**: Smooths out any noise in the tracked position, reducing jitter from momentary detection errors or sudden movements.
5. **Command Generation**: Compares face position relative to the frame center and sends high-level "move left/right/up/down" strings over UART.

The vision processing pipeline in our application processor leverages a series of complementary techniques that, when combined, yield a robust and efficient face tracking system. These techniques include real-time video capture, face detection, optical flow tracking, Kalman filtering, and command generation. Each step is optimized to work seamlessly together, ensuring high performance and accuracy even in dynamic environments. The pipeline's modularity allows for easy updates and improvements to individual components. Figure 9 depicts the face detection pipeline, showcasing how each stage contributes to the system's ability to track facial movements in real-time.

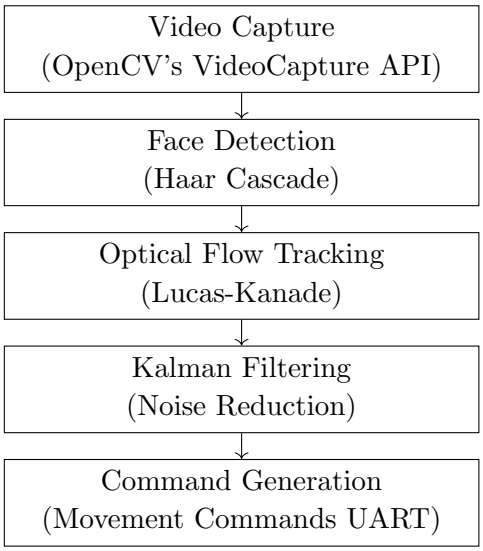


Figure 9: Face Tracking Pipeline

The rest of this section will go over the necessary background information for the individual optimization techniques shown in Figure 9 and how they contribute to the overall system. Each technique has been carefully selected for its computational efficiency and suitability for real-time processing on embedded hardware. We will discuss not only the theoretical foundations of these methods, but also their practical implementation details and integration challenges. Understanding these techniques in isolation, as well as how they interact in the full pipeline, is key to appreciating the robustness and responsiveness of our face tracking system. It is worth noting that the code for this section can be found at the repository cited in entry [4] of the references section of this report in the "software" folder of the Git repository.

4.6.2 Haar Cascade Classifiers and Multiscale Detection

Haar Cascade Classifiers are central to our face detection approach. Built upon the principles of the Viola–Jones algorithm, they rapidly analyze images by looking for simple patterns. Specifically, differences in light and dark areas across rectangular regions are looked for. These differences enable the classifier to quickly separate faces from non-face regions. By combining many of these basic features through an AdaBoost learning process, the system can accurately detect faces even in varying conditions. The classifier uses a series of weak classifiers, each focused on detecting simple features, and AdaBoost helps to optimize and combine them into a strong overall detector. This method is highly efficient, enabling real-time detection even with large image datasets. Additionally, Haar Cascade Classifiers can be trained for specific environments or variations, improving detection accuracy under different lighting conditions, poses, and partial occlusions.

- **Frontal-Face Cascade:** The primary classifier, `haarcascade_frontalface_default.xml`, is employed to detect frontally oriented faces.
- **Profile Cascade (Optional):** In scenarios where the frontal-face detector fails (e.g., when a subject turns their head), an alternative profile cascade is used as a fallback to ensure robust detection, even for individuals wearing glasses.

The detection routine is designed to handle faces of various sizes by employing a multi-scale strategy. In this approach, the original image is progressively downscaled, and the detection algorithm is applied at each scale. If a face is detected on a downscaled version, its coordinates are then scaled back to align with the original image size. This ensures that faces—whether they appear large up close or small in the distance—are reliably detected.

To further improve detection speed and robustness, we apply three lightweight preprocessing steps before running the Haar cascade:

- **Convert to Grayscale.** Faces can differ in shape and shading. Converting each frame from RGB to a single-channel grayscale image reduces memory traffic and arithmetic by a factor of three:

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

```

- **Downscale the Image.** Smaller images require fewer pixel operations. By resizing with a scale factor (e.g. 0.5), we cut the total number of pixels—and thus the cascade's workload—by roughly fourfold, while still retaining enough detail to locate faces:

```
small_gray = cv2.resize(gray, (0, 0), fx=0.5, fy=0.5)

```

- **Equalize the Histogram.** Varying lighting can cause some facial features to be too dark or too bright. Histogram equalization spreads out pixel intensities so that contrasts (especially around eyes, nose, and mouth) become more pronounced, improving cascade accuracy:

```
equalized = cv2.equalizeHist(small_gray)

```

4.6.3 Optical Flow Tracking with Lucas–Kanade

Once a face is detected in the frame, the Lucas–Kanade optical flow algorithm is used to track its position between subsequent frames. This method calculates the apparent motion of image intensities by estimating displacement vectors based on the brightness constancy assumption. These displacement vectors represent the shift in pixel positions between frames, providing a measure of movement for specific image features. Key aspects include:

- **Reduced Computation:** Instead of running a full cascade on every frame, optical flow tracking leverages a prior detection to quickly update the face position.
- **Pyramidal Approach:** Constructing image pyramids makes the algorithm robust to scale variations and larger motions:

```
next_points, status, _ = cv2.calcOpticalFlowPyrLK(prev_gray, gray, prev_points, None, **lk_params)

```

- **Local Window Search:** The search is confined to a small neighborhood (as defined by `winSize` in `lk_params`), accelerating the calculation while maintaining the precision needed for tracking.

4.6.4 Kalman Filtering for Position Smoothing

Despite robust detection and tracking methods, minor frame-to-frame variations and external noise can induce jitter in the tracked position. To compensate for this, a Kalman filter is applied:

- **State Estimation:** The filter uses a four-dimensional state vector, $[x, y, v_x, v_y]$, to model both the position and velocity of the face. This prediction helps in providing smoother transitions:

```
state = np.array([[face_center[0]], [face_center[1]], [0], [0]], np.float32)
kalman.statePre = state.copy()

```

- **Measurement Correction:** With each optical flow measurement, the Kalman filter adjusts its state estimate:

```
kalman.correct(measured)

```

- **Process Noise Covariance:** Represented by a covariance matrix, this parameter models the uncertainty in the filter's prediction. In simpler terms, it quantifies the expected random variation or error in the predicted state. Lower values favor smoother tracking by relying more on prior predictions, whereas higher values allow quicker response to new measurements at the risk of increased jitter.

4.6.5 Command Generation and UART Communication

The purpose of face tracking is to drive the robotic arm by aligning the detected face with the center of the video frame. This is achieved by:

- **Computing Offsets:** The system calculates the difference (dx, dy) between the current, Kalman-filtered face position and the frame's center. For example:

```
dx = center_x - int(prediction[0])
dy = center_y - int(prediction[1])

```

- **Thresholding:** Only if these offsets exceed a predefined tolerance are movement commands generated, preventing unnecessary micro-adjustments.
- **Direction Decisions:** Depending on the sign and magnitude of the offsets, high-level directional commands (e.g., `"LEFT,UP"`) are constructed.
- **Throttling:** A short cooldown period is enforced to avoid overwhelming the STM32 with commands.
- **Serial Transmission:** These commands are sent over UART to the microcontroller, ensuring that the motor control system remains synchronized with the tracked face position.

4.7 Mechanical Design

4.7.1 Open Source Design and Economies of Scale

The robotic arm's mechanical design is based on an open-source model by 20sffactory[1], which leverages 3D-printed components and widely available hardware common in generic Chinese 3D printers—such as NEMA 17 stepper motors, SPDT endstop switches, rubber belts, ball bearings, and standard metric fasteners. To reduce costs and development time, and to focus efforts on electronics and software, this design was chosen for its compatibility with existing supply chains and ease of maintenance, aligning with both our budget and time constraints.

4.7.2 Modifications to the Original Design

The original design opted for off-the-shelf mainboards such as the RAMPS motor control platform and the Arduino platform. Given that we opted to design our own motor control mainboard to mount the electronic components, the original housings for the boards did not fit our design. As such, it was necessary to model a housing from scratch using 3D Solid Body CAD software such as the open source FreeCAD suite. The housing we designed is pictured below. The board is housed within, and has the necessary ports to allow external connectors to mate.

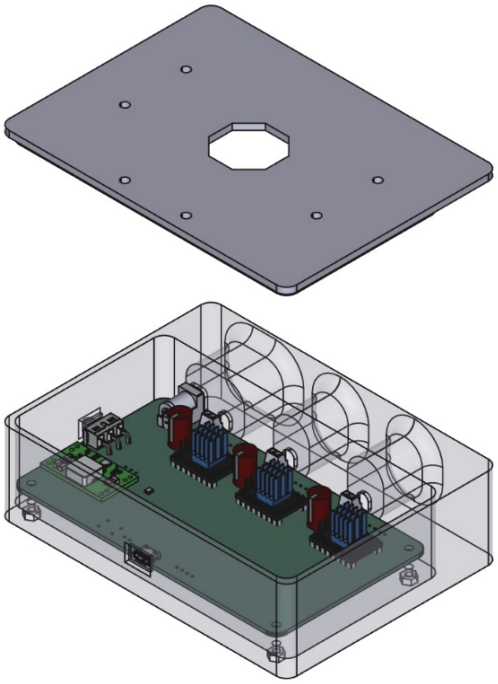


Figure 10: Custom Case for Design

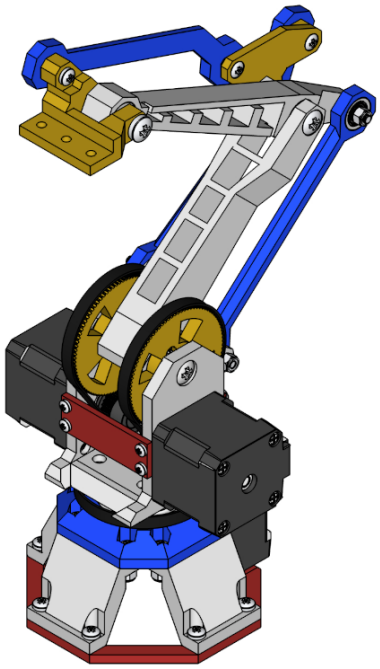


Figure 11: 20sffactory Arm Design[2]

[1]See entry [3] in references.
[2]See Appendix C for full Bill of Materials.

5 Integration and Test Results

5.1 Breadboard Testing and Homing Routine Validation

Before committing to a final PCB design, we constructed a functional prototype on a solderless breadboard to validate our hardware selection and firmware. The testing process began with isolated movement tests for each of the three stepper motors, which were not mounted from the robotic arm assembly. This allowed us to verify that basic motor control worked as expected without risking motors siezing due to the mechanical constraints of the robotic arm. Once confirmed, we evaluated the homing routine described in Section 4.5 using 3D-printed "stepper motor test mount" equipped with end-stop switches. During these tests, each motor rotated until an end-stop was triggered, signaling the firmware to halt the motor and register a reference point for positional tracking.

With the homing logic validated, we proceeded to integrate the motors and end-stop switches into the assembled robotic arm. While executing the homing routine in this full mechanical setup, we observed that the forward/backward and up/down motors occasionally stalled due to insufficient torque. To resolve this, we first loosened the bolts in the mechanical joints to reduce friction. We then adjusted the TMC-2209 driver's onboard potentiometer to increase the reference voltage to approximately 1.4V, delivering sufficient current for stable operation. All tests were performed using a STM32 Nucleo Board with an STM32F446RE, though we later transitioned to the lower-specification STM32 variant detailed in Section 4.2.1 for the final design.

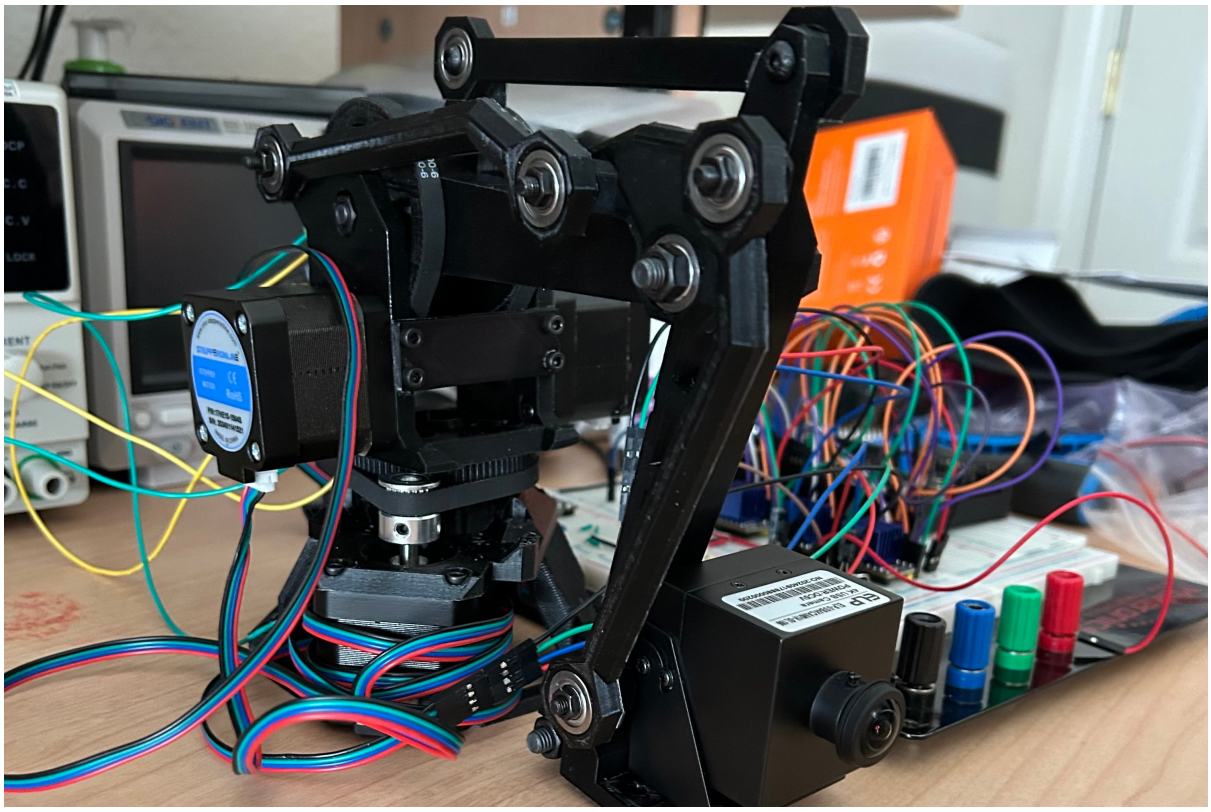


Figure 12: Robotic Arm Assembly and Electronics during Testing

5.2 PCB Smoke and Movement Tests

After receiving the PCB blanks from JLCPCB and assembling them with components sourced from Digi-Key, we conducted an initial power-on test. The 9V rail was activated first and functioned as expected, with no signs of component failure or electrical issues. We then enabled the 3.3V rail by bridging the 9V input to the buck converter using a jumper cap on the designated pin header. A minimal test firmware, programmed to blink a debug LED, confirmed proper microcontroller operation. Upon verifying normal operation, we proceeded to flash the full firmware with motor movement routines, including homing and standard motion. Motors and end-stop switches from the robotic arm were then connected to the mainboard, and USB communication was established. The system performed as expected, with the robotic arm successfully tracking faces in the camera frame.

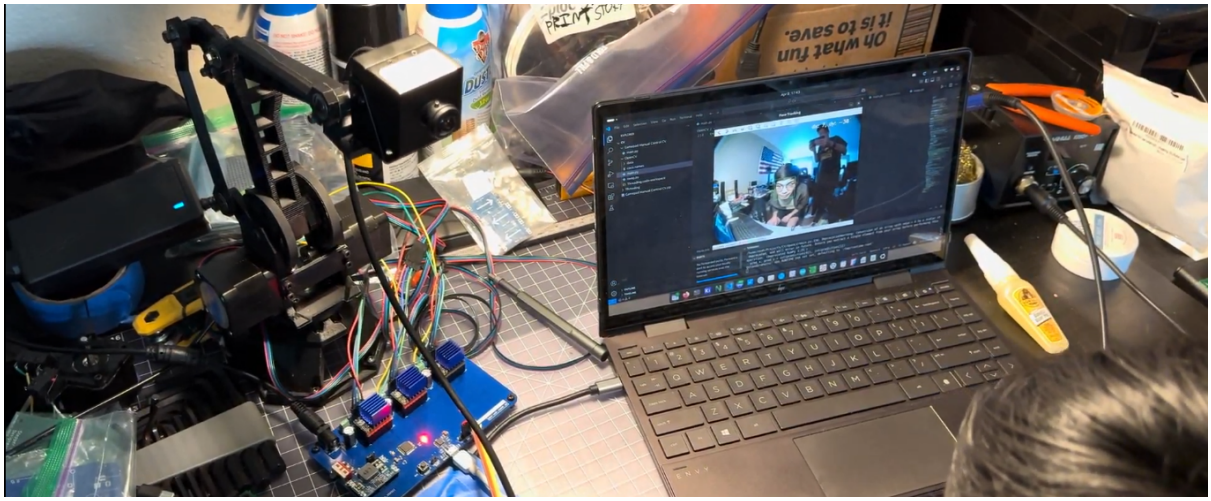


Figure 13: Assembled Circuit Prior to After Smoke Test

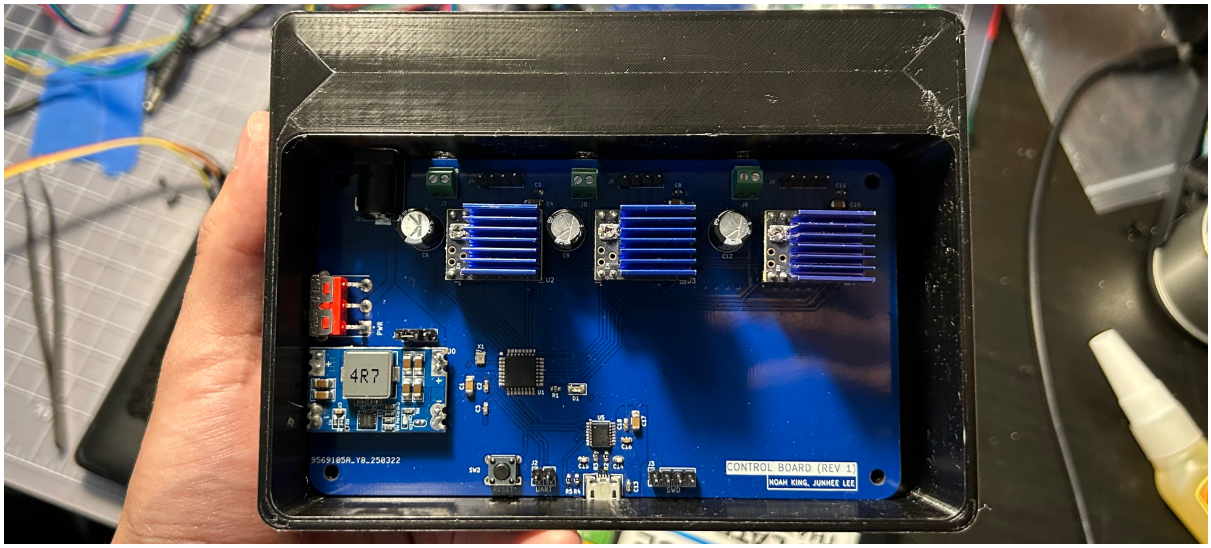


Figure 14: Assembled PCB

5.3 Manual Movement and OpenCV Tracking Tests

Following successful verification of the PCB's functionality, we conducted manual motion tests using a Python-based control program running on the application processor (an x86 laptop). The system was operated via an Xbox controller, and the resulting movement was observed to be smooth and consistent. The robotic arm used a delayed reticle aiming scheme, meaning it didn't snap instantly to the target but instead followed the reticle with a slight lag, smoothing out motion and preventing jitter. For visiblity purposes, the reticle color can be changed using user input for visibility. More importantly, reticle tracking controlling the robotic arm movement can be toggled as well. The exact controls and buttons for the different manual controls using gamepad can be found in Figure 15 below. Compared to the OpenCV tests, we found manual control to be more consistent, however, this is expected as manual control will always send a message over UART, vs OpenCV which will only send messages so long as a face is detected.

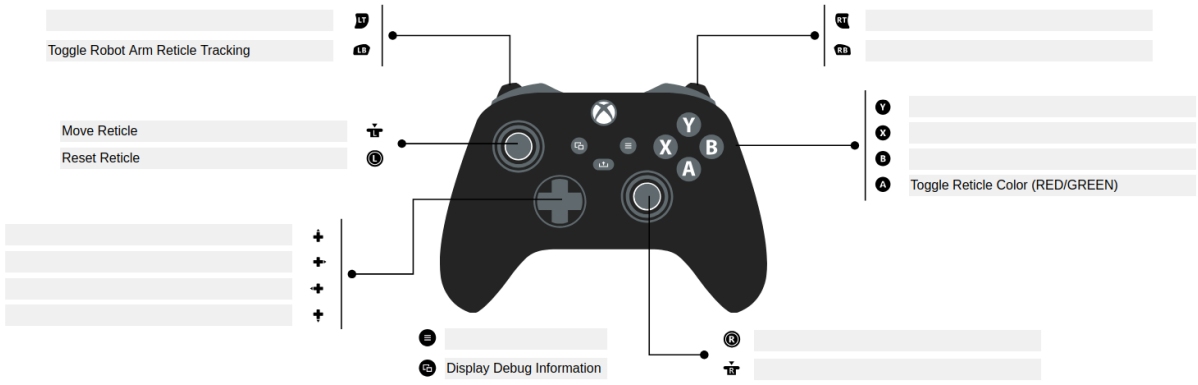


Figure 15: XBOX Controller Control Scheme

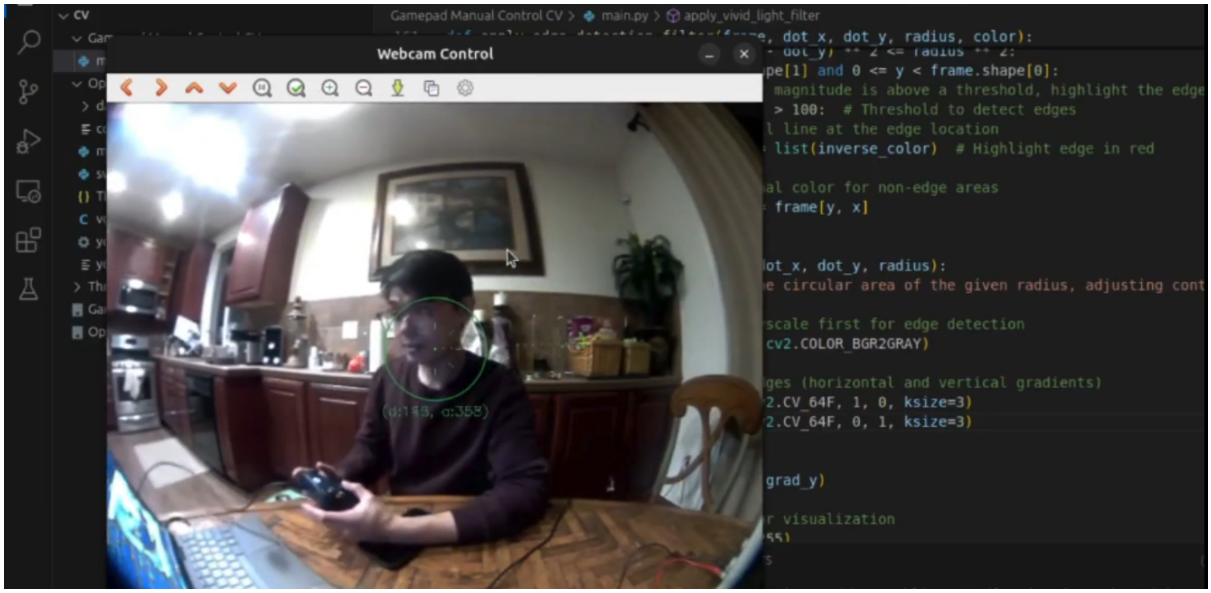


Figure 16: Movement Tests using XBOX Controller

After verifying the functionality of the manual control system, we transitioned to an automatic computer vision tracking system. Unlike the manual controls, UART were conditionally issued only when a face was detected and off-center. Initially, the system exhibited jerky movement due to unreliable face detection in low light or when the tracked subject was too far from the camera. To mitigate this, we calibrated the OpenCV face detection parameters, which resulted in smoother operation. This was achieved by adjusting the "Scale Factor" and "Number of Neighbors" variables in OpenCV's Viola and Jones implementation. By tuning the detection parameters, we achieved more stable and fluid motion from the robotic arm. Additionally, the 3D printed nature of the robotic arm assembly led to some mechanical failures when handled too roughly, prompting us to print replacement parts. Furthermore, the loose mechanical tolerances of the original design (play within the ball bearings) led to less than ideal jagged movement.

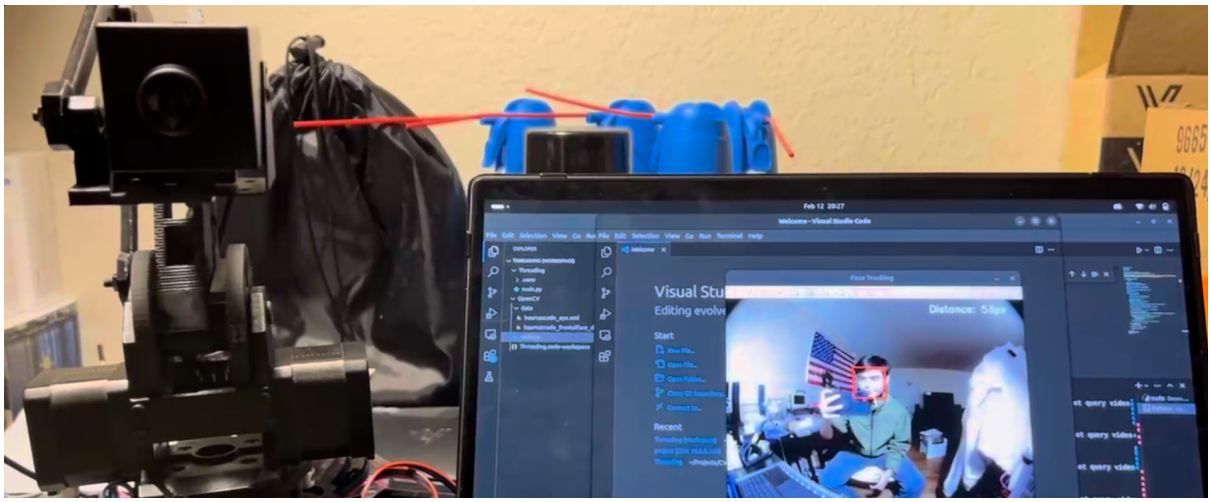


Figure 17: OpenCV Tests (Computer + Arm)

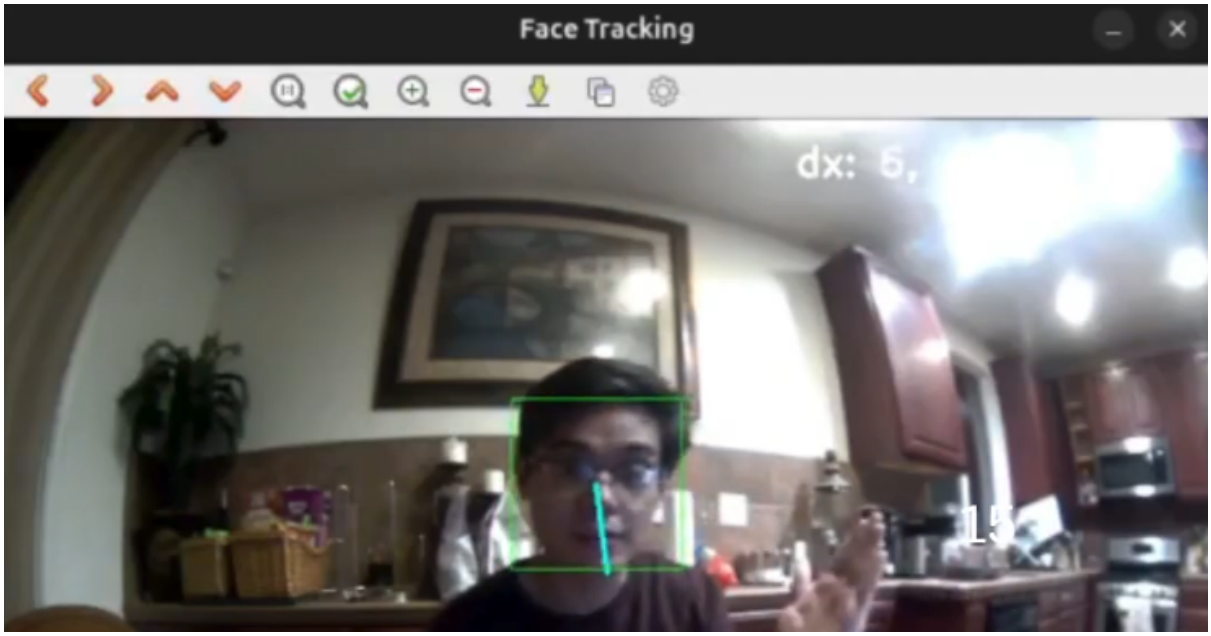


Figure 18: OpenCV Movement Test (Computer View)

6 Standards

6.1 USB

The Universal Serial Bus Physical Standard. The USB standard is a huge standard with components on all 7 layers of the Open Systems Interconnect model (OSI). The portion of the standard most relevant to our project is Layer 1, the Physical Layer of the USB standard, which defines voltage levels, signaling, and frequency, the three most relevant portions of the standard to signal integrity of printed circuit boards such as the one designed for this project. [10]

Voltage Levels and Differential Signaling. The USB standard utilizes differential signaling over two wires "D+" and "D-" respectively. This method transmits complementary signals across the pair, significantly enhancing noise immunity by allowing common-mode noise to be canceled at the receiver. The voltage levels between 0V for logic-low, and 5V for logic-high. Compared to single wire solutions, differential signaling provides superior signal integrity over equivalent distances and enables the support for higher data transfer rates. [10]

Synchronization and Packet Based Communication. USB communication is packet-based, starting with a SYNC field that allows the receiver to lock onto the sender's clock for timing recovery. This relies on clean signal transitions and precise edge timing, which can be affected by PCB layout, parasitics, and noise. Following the SYNC field, the data payload is transmitted using NRZI encoding, where a '0' causes a transition and a '1' does not. Bit-stuffing is used to ensure regular transitions for clock recovery, inserting a '0' after six consecutive '1's. Clean edges are critical, as distorted transitions can corrupt the payload or desynchronize the receiver. [10]

Error Detection USB uses error detection mechanisms based primarily on Cyclic Redundancy Checks (CRC). Token packets and Start of Frame (SOF) packets include a 5 bit CRC (CRC5) that covers the address and endpoint fields, ensuring the correctness of control information. Data packets, which carry the actual payload, use a 16 bit CRC (CRC16) to validate the integrity of the data. Handshake packets, due to their minimal size, do not include a CRC. If a device detects a CRC error, it simply discards the packet. In most cases, the absence of an acknowledgment (ACK) signals the host to retransmit. [10]

Applications and Design Considerations. USB was selected as the communication protocol between the motor control mainboard and the external application processor (a personal computer) due to its widespread availability, reliability, and ease of integration. The system does not demand the extended range or specialized features offered by less common standards such as RS-485, as the application processor and motor control board are expected to operate in close proximity to the robotic arm. [10]

Sample Diagrams. Figures 19 and 20 illustrate the typical waveform characteristics of the D+ and D signals forming a USB differential pair. Given the variety of packet types defined at the protocol level in the USB standard, further technical details can be found in Section 8.4 of the USB 2.0 specification, as cited in entry [10] in the references section of this report.

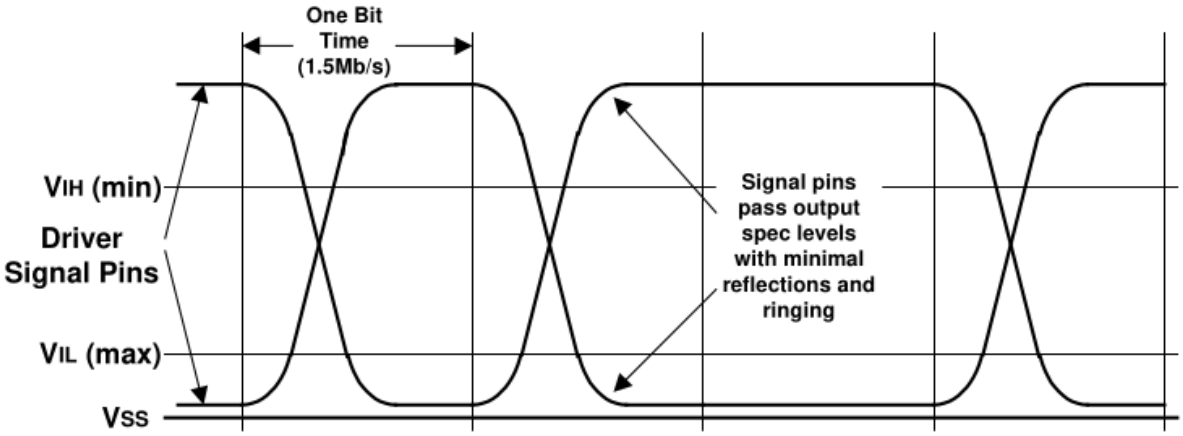


Figure 19: USB Logic Level Specifications (D+, D-) [1]

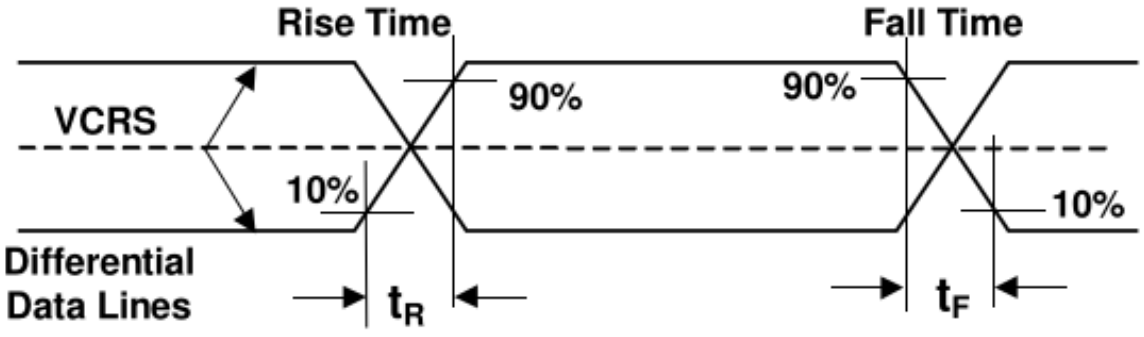


Figure 20: USB Rise/Fall Specifications (D+, D-) [1]

[1]From USB Revision 2.0 [10]

6.2 UART

Overview. The Universal Asynchronous Receiver/Transmitter (UART) is a serial-data interface that converts parallel bytes into a timed bitstream over a minimal wire set, enabling low-cost communication between microcontrollers and peripherals.

Asynchronous Operation. UART is asynchronous: there is no shared clock line. Each frame begins with a low-going start bit that tells the receiver's baud-rate generator (which typically runs at 8× or 16× the data rate) to begin timing. Data bits are then sampled at the midpoint of each bit period—where the signal is most stable—to reduce the chance of reading during a transition and tolerate small clock-rate differences [9].

Frame Structure. A UART frame consists of:

- Idle state: logic 1 (mark) when the line is inactive.
- Start bit: logic 0 (space) marking frame start.
- Data bits: 5–9 bits sent least significant bit first.
- Parity bit (optional): even, odd, or none.
- Stop bits: one or two logic 1 bits marking frame end.

Both transmitter and receiver must use identical baud rate and framing parameters—otherwise a framing error occurs when the receiver fails to detect the stop bit at the expected time, indicating a misalignment in timing or configuration [9].

Clock and Synchronization. Inside the UART, the baud-rate generator derives timing from the system clock. On reception, detection of the start-bit falling edge initiates sampling at fixed intervals (e.g., every 16 clock ticks), allowing for small clock-rate discrepancies without data corruption [9].

Electrical Signaling. UART framing is independent of voltage levels. Common signaling standards include:

- TTL-level UART: 0 V for logic 0, V_{CC} (3.3 V or 5 V) for logic 1.
- RS-232: ±3 V to ±15 V via external transceiver.
- RS-485: differential ±1.5 V to ±5 V for multi-drop networks.

Choosing the correct line driver ensures signal integrity and noise immunity [1].

Error Detection and Flow Control. UART can detect simple errors using the optional parity bit, and peripherals report framing, parity, and overrun errors via status flags. Hardware flow control with RTS/CTS lines can pause transmission to prevent buffer overruns when the receiver's FIFO is full [1].

Duplex Modes and Buffering. Most UART modules support full-duplex communication by using separate transmit and receive buffers (often implemented as 8–16 byte FIFOs) that allow data to be sent and received simultaneously. If a microcontroller lacks a dedicated UART peripheral, firmware can emulate the same framing and timing by manually toggling GPIO pins at precise intervals—a method known as "bit-banging"—to generate the start, data, parity, and stop bits in software. This software-driven approach increases CPU usage and requires very accurate timing to maintain reliable data transfer.

Application and Design Considerations. We selected UART as the communication link between the application processor and the STM32 microcontroller because it requires only two signal lines (TX and RX) plus ground, minimizing PCB complexity and pin usage. By interfacing through the FT230XS USB–UART bridge, the host computer gains plug-and-play USB connectivity without additional drivers, while the STM32's built-in USART peripheral handles incoming commands natively at 115,200 baud. This data rate is sufficient for real-time directional and control messages, and the UART's asynchronous framing plus hardware error-status flags make it easy to detect and recover from framing, parity, or overrun errors in the noisy environment of stepper-motor drivers [9].

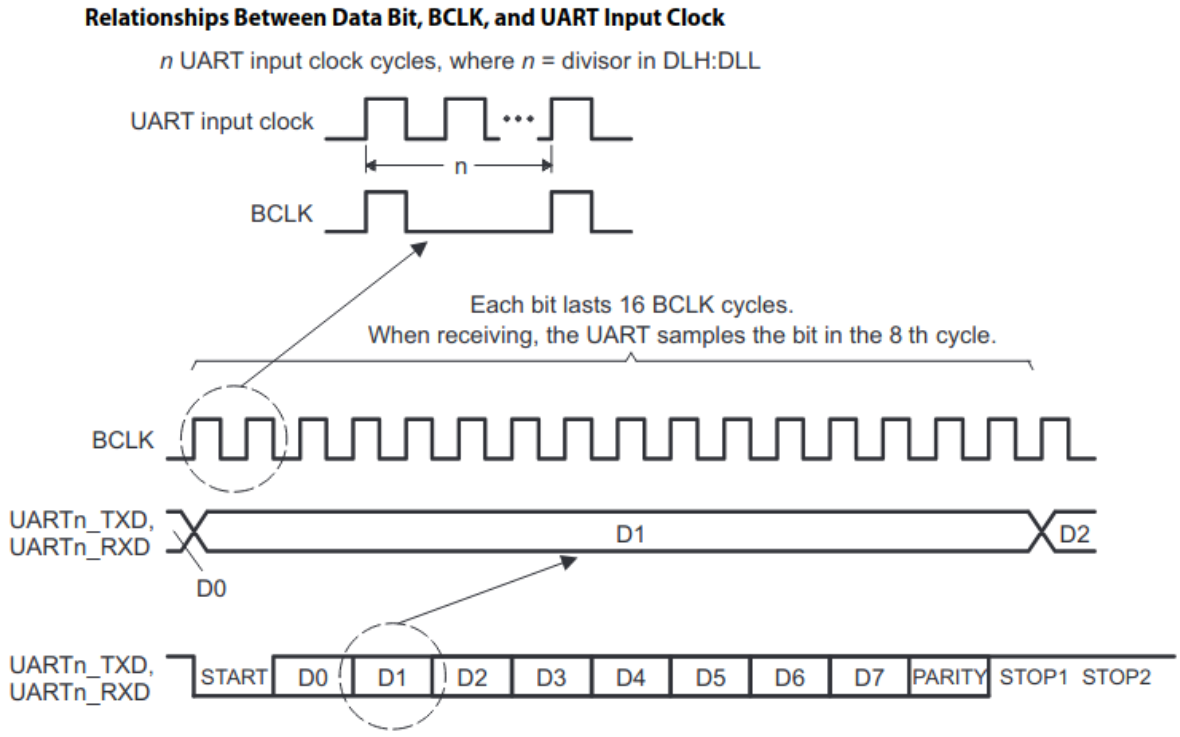


Figure 21: UART Sample Timing Diagram [1]

[1]Figure from Texas Instruments [9].

6.3 Standards for Fasteners, Wires, & Connectors

The mechanical and electrical connector parts had properties that were denominated in a variety of standards. The standards each component adheres to are detailed in the datasheets of the parts listed in the Bill of Materials for the final version of the motor control mainboard or robotic arm design.

American Wire Gauge (AWG). Our project utilized wires denominated in standard AWG denominations.

International System of Units (SI) Our project utilized mechanical fasteners such as bolts, screws, washers, and ball bearings utilizing units denominated in the international (SI) standard. Additionally, the dimensions of the pins of connectors attached to the PCB were given in SI units, the same can be said of particular surface mount components.

US Customary System "Imperial Units". A subset of the surface mount components present on the final design of the PCB have their dimensions denominated in US Customary System units. For example a 0603 surface mount capacitor is 0.06×0.03 inches.

Applications and Design Considerations. Mechanical components and electrical connectors were selected primarily based on cost-effectiveness and availability. Preference was given to parts conforming to established industry standards, as these tend to be more readily accessible and economically viable. The authors, based in the United States, selected components conforming to both SI and US Customary standards to ensure compatibility and ease of sourcing.

7 Constraints

7.1 Mainboard Design Constraints

7.1.1 PCB Layout Constraints

Constraints when Routing USB Differential Pairs. In the context of our project, the USB standard introduced a number of constraints that needed to be taken into account when designing the final version of our motor control mainboard, that being the routing of USB differential pairs. USB differential pairs should be routed short, parallel, and as closely matched in length as possible. USB differential pairs should be routed with near equal lengths, kept parallel, closely spaced, and as short as practical. This layout minimizes parasitic capacitance and inductance, which can degrade the sharp edge transitions of USB signals, leading to data corruption. To further reduce electromagnetic interference (EMI), the USB-to-UART bridge circuitry was enclosed with a ground-referenced copper pour, tied to the ground plane through multiple stitched vias for effective shielding and noise suppression.

7.1.2 PCB Manufacturing Constraints

As one would expect, a PCB design is constrained by the capabilities of the chosen manufacturing process. For our project, the chosen manufacturer was the prototyping service provider, JLCPCB. JLCPCB provides a number of design rules to designers to make sure that their design is manufacturable, this includes the amount of clearance needed between copper traces, the minimum size of vias, and the minimum size for plated holes. These variables may differ between processes (our PCB utilized the 2 layer process). To ensure compliance with the selected manufacturing process, the design rules were integrated into our EDA software, KiCAD. This enabled automatic alerts during the Design Rule Check (DRC) whenever the design deviated from the specified manufacturing parameters. Additionally, JLCPCB offers a free PCB-DFM (Design for Manufacturing) tool that notifies designers if their design falls outside the manufacturing specifications.

7.1.3 PCB Cost Constraints

As previously stated, a key design objective for our mainboard was cost minimization, as marketable designs often focus on delivering required functionality at the lowest price. In evaluating PCB manufacturers, U.S. and EU-based options proved uncompetitive for low-volume prototyping. In contrast, Chinese and Indian manufacturers offered significantly lower unit costs. We selected JLCPCB for its superior cost efficiency—while DigiKey Red and OSHPARK quoted around $200 for minimum quantities, JLCPCB quoted approximately $17. It should be noted that if cost were not a constraint, or if regulations prohibited overseas manufacturing, this decision would be reconsidered. Design choices are ultimately shaped by both technical requirements and external constraints.

7.2 Integrated Circuit Constraints

7.2.1 Microcontroller Constraints

Due to our choice of the STM32C071KBT3, we were constrained to the peripherals present on the chip, that being the number of I/O pins, and the layout of the I/O pins on the QFP-32 package. For example, the USART1 peripheral on this microcontroller can be mapped to physical pins 7 and 8 through appropriate configuration of the alternate function registers. Despite this, USART1 can only be initialized to the pins the hardware allows it to be assigned to.

7.2.2 Motor Driver Control Mode Constraints

Due to our use of the TMC-2209 in legacy control mode, we utilized the MS1, MS2, and EN pins to maintain compatibility with other pin-compatible drivers like the A4988. This choice precluded the use of the TMC-2209's UART control mode, which replaces those pins with a single UART line. For the robotic arm, we prioritized compatibility with off-the-shelf components to enhance sourcing flexibility. While not intended as a commercial product, this approach served as a practical exercise in engineering economics, factoring in market-driven considerations such as component availability and cost.

7.3 Mechanical Constraints

7.3.1 Range of Motion Constraints

The mechanical assembly of the robotic arm featured a limited range of motion due to physical constraints inherent in the design, such as joint angles, link lengths, and potential interference between components. As a result, the firmware had to be carefully designed to operate within these bounds, incorporating motion limits and safety checks to prevent overextension or mechanical collision. These constraints influenced the implementation of movement routines, homing sequences, and position tracking, ensuring that all motor commands remained within the safe and functional range of the physical system.

7.4 Processing/Compute Constraints

7.4.1 Support for Lower Performance Embedded Processors

Another constraint we took into account was the amount of processing power we wanted our software to consume. While a personal computer with a GPU can handle more advanced detection algorithms (YOLO), a less powerful embedded processor core such as a ARM Cortex-A9 or ARM Cortex-A35 could struggle. As mentioned previously, this is the reason we opted for classical computer vision algorithms such as Viola & Jones, Kalman Filters, and Optical Flow.

8 Project Planning and Task Definition

8.1 Initial Prototyping on the Nucleo-F446RE

The very first experiments were planned to be carried out on ST's **Nucleo-F446RE** development board. Its on-board ST-LINK debugger allowed single–step inspection of registers while driving an external `TMC2209` driver that was plugged into a solderless breadboard. During this phase we verified that:

- the `STEP` and `DIR` outputs from a 3.3 V *Cortex-M4* were fully compatible with the `TMC2209`'s logic inputs, and
- a separate 12 V rail was mandatory: the driver's internal low-dropout regulator cannot source enough current for a motor, thus a small *MINI-560* buck converter was temporarily wired in to step 12 V down to 3.3 V.

All control firmware lived in a single `main.c` that toggled GPIOs in a blocking loop. UART traffic was generated with `HAL_UART_Transmit()` calls so that a PC terminal could display debug strings such as `"Endstop triggered!\r\n"`.

8.2 Migrating to the STM32C0 Series on a Custom PCB

Once our proof-of-concept ran reliably on the Nucleo-F446RE and a solderless breadboard, we shifted our focus to planning a custom PCB. This decision arose from several considerations.

- **Robustness and Reliability.** Breadboards are great for initial wiring and testing, but loose jumper wires and inconsistent connections often lead to intermittent faults, especially when the assembly is moved around or vibrates. A PCB with soldered joints would reduce the overall time required to troubleshoot erratic behavior.
- **Signal Integrity.** The stepper control lines (UART, STEP/DIR, SWD) benefit from controlled trace widths, clear ground referencing, and minimal crosstalk. On a breadboard, long loose wires and lack of a ground plane can introduce noise; a two-layer PCB lets us pour a solid ground plane and route critical traces with controlled impedance.
- **Clear Documentation and Reproducibility.** By storing KiCad schematic and PCB layout files in a version control system (e.g., Git), we can track changes and roll back to prior revisions if necessary, and share the exact designs with manufacturers.
- **Ease of Assembly and Maintenance.** As our group will be uploading our firmware and CAD files to a GIT repo, it opens up the possibility that future students or engineers could reproduce the design quickly or build more intensive projects on top of our existing work.

Following a resource audit revealing less than 20 GPIOs, a single UART at 115200 baud, and one 16-bit timer in use, the **STM32F446RE** was replaced with the simpler and cost-effective **STM32C071KBT3** since most of the peripherals offered by the STM32F446RE

were not being utilized. All required signals mapped cleanly to the C0's pins, with spare I/Os available for end-stops. Switching to the C0 reduced the BOM by ≈$6 and enabled a simpler two-layer PCB layout in KiCad. The 32-pin LQFP package also allowed the stepper drivers, `FT230XS` USB-UART bridge, and *MINI-560* power module to fit within a compact 70mm×60mm board outline.

8.3 Modular Multi-Motor Firmware Development

In anticipation that the firmware would become harder to manage as our project would have to account for second and third motor axis, the codebase would be refactored from a single `main.c` into:

- `tmc2209.c/h` – low-level routines: enable, micro-step mode, pulse generation with `delay_us()` driven by TIM1;
- `motor_defines.h` – all pin assignments in one place;
- `main.c` – high-level parser that converts strings such as `"LEFT,UP"` into function call `stepper_process_command()`.

Interrupt-driven UART reception (`HAL_UART_RxCpltCallback`) copies a complete command into a ring buffer and sets a `commandReady` flag; the main loop merely dispatches the command, eliminating busy-waiting.

8.4 Vision Algorithm Planning

8.4.1 Why *not* YOLOv5

YOLOv5 was initially considered due to its state-of-the-art object detection performance. However, it was ultimately ruled out to avoid unnecessary AI complexity for our application, which focuses solely on tracking a single face rather than detecting multiple object classes. Deploying YOLOv5 would demand more powerful hardware such as high-end embedded processors that would increase costs. While pre-trained facial tracking models and parameters exist, they require more powerful hardware to retain the real-time responsiveness required for face tracking. Therefore, the haar cascades were chosen because they can run efficiently on low-power embedded platforms.

8.4.2 Haar Cascade Pipeline

1. **Stage 1 – UART Validation.** The first objective was to prove that the USB-UART link and interrupt-driven receive handler operated correctly. We wrote a minimal PyQt5 GUI with four buttons (`UP`, `DOWN`, `LEFT`, `RIGHT`). Each click sent the exact ASCII word to the STM32, which echoed it back.
2. **Stage 2 – Initial OpenCV Integration.** Once UART was implemented, we loaded the pre-trained frontal-face cascade (`haarcascade_frontalface_default.xml`) in a Python script. The code detected all faces in each frame, picked the bounding box whose center lay closest to the image center, and transmitted the matching

direction string over UART. To smooth out occasional mis-detections, we overlaid pyramidal Lucas–Kanade optical flow and a simple four-state Kalman filter.

3. **Stage 3 – Performance Tuning & Final Integration.** After initial implementation of OpenCV, we planned to dedicate time to optimizing the software by reducing the CPU load and improving robustness. We switched from frame-count to a one-second timer to trigger cascade runs, down-scaled the grayscale image before detection, and applied histogram equalization for better contrast in dim lighting. A profile-face cascade was added as a fallback whenever the frontal detector failed, and the LK tracker was forced to re-initialize on any lost track. Finally, we imposed a ±20 pixel dead-zone around the frame center and a 20ms command cooldown to prevent UART flooding and ensure smooth motor operation.

8.5 Project Timeline and Milestones

- **Weeks 1-4** - The breadboard prototype was created, achieved single-axis motion on the Nucleo-F446RE, and verified basic functionality with UART `printf` diagnostics with PyQt5.
- **Weeks 5-6** - A end-stop interrupt service routine was tested and implemented. A homing procedure for the motors was created as well.
- **Weeks 7-8** - Initial designs for the KiCad layout for REV 0 were created and ordered, later to arrive. A Python script was created, implementing Haar cascades with Lucas-Kanade tracking and a Kalman filter.
- **Weeks 9-10** - The firmware was migrated to an STM32C0 while refactoring the codebase into separate `*.c` and `*.h` modules, and tuning the vision pipeline by implemention optimizations such as histogram equalization and a profile-face fallback cascade. REV 0 was also assembled and tested successfully.
- **Weeks 11-13** - REV 1 PCB was designed and ordered, then built and tested once the boards were shipped and delivered.

9 Conclusion

By integrating these modules, the system achieves real-time face tracking and precise motion control for the robotic arm. The inclusion of code snippets—such as the use of `cv2.VideoCapture`, `cv2.calcOpticalFlowPyrLK`, and the Kalman filter update routines—illustrates how key variables (e.g., `prev_points`, `lk_params`, `rx_buffer`, `commandReady`) are utilized in practice. Together, these examples help bridge the gap between the conceptual design and its implementation, providing a clear, comprehensive view of the entire system. To see specific parts of the implementation, see our Git repository cited in entry [4] in the references section of this report. The repository contains the firmware, software, and CAD files used in our design.

References

[1] Analog Devices, "UART: A Hardware Communication Protocol," *Analog Dialogue*. [Online]. Available: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html. [Accessed: 2025].

[2] Analog Devices, "TMC2209 Datasheet." [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/TMC2209_datasheet_rev1.08.pdf. [Accessed: 2024].

[3] 20sffactory, "Community Robot Arm," *GitHub* repository. [Online]. Available: https://github.com/20sffactory/community_robot_arm. [Accessed: 2025].

[4] N. King and J. Lee, "SDP-ROBOT-ARM," *GitHub* repository, 2025. [Online]. Available: https://github.com/nck2e3/SDP-ROBOT-ARM. [Accessed: Apr. 30, 2025].

[5] Future Technology Devices International Ltd., "FT230X USB to Basic UART IC Datasheet," Rev. 1.6. [Online]. Available: https://ftdichip.com/products/ft230xs/. [Accessed: 2025].

[6] OpenCV Team, "OpenCV," *GitHub* repository, 2025. [Online]. Available: https://github.com/opencv/opencv. [Accessed: Apr. 30, 2025].

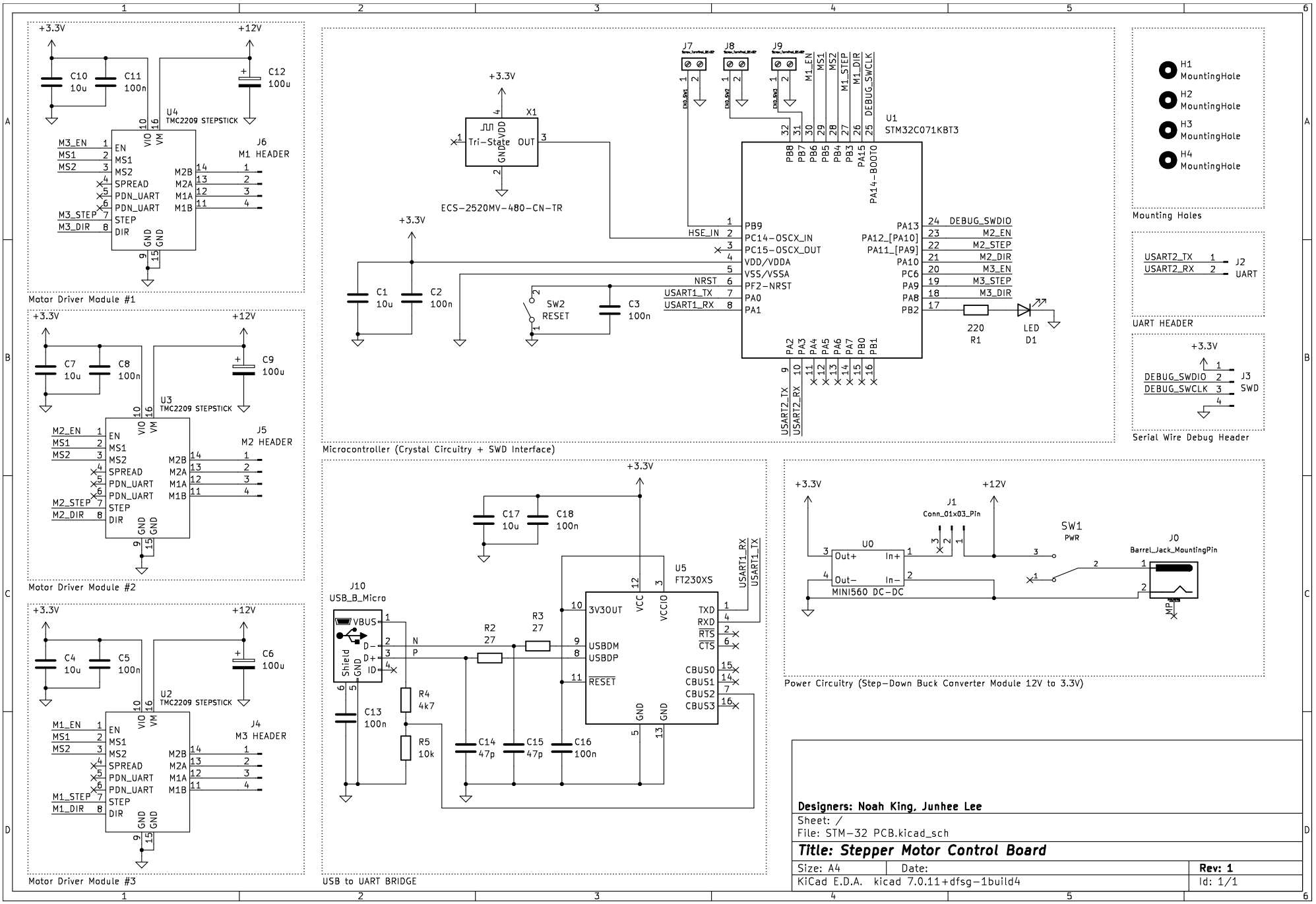
[7] Pygame Community, "Pygame," *GitHub* repository, 2025. [Online]. Available: https://github.com/pygame/pygame. [Accessed: Apr. 30, 2025].

[8] STMicroelectronics, "STM32C071x8/xB Arm Cortex-M0+ 32-bit MCU Datasheet," Rev. 1, Sep. 2024. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32c071kb.pdf. [Accessed: 2025].

[9] Texas Instruments, "Universal Asynchronous Receiver/Transmitter (UART) and Synchronous/Asynchronous Receiver/Transmitter (USART) Satellite User's Guide (SPRUGP1)." [Online]. Available: https://www.ti.com/lit/ug/sprugp1/sprugp1.pdf. [Accessed: 2025].

[10] USB Implementers Forum, "Universal Serial Bus Specification Revision 2.0," Apr. 27, 2000. [Online]. Available: https://www.usb.org/document-library/usb-20-specification. [Accessed: 2025].

Appendix A


Motor Driver Module #1
Motor Driver Module #2
Motor Driver Module #3
Microcontroller (Crystal Circuitry + SWD Interface)
USB to UART BRIDGE
Power Circuitry (Step-Down Buck Converter Module 12V to 3.3V)
Mounting Holes
UART HEADER
Serial Wire Debug Header
ECS-2520MV-480-CN-TR
STM32C071KBT3
TMC2209 STEPSTICK
FT230XS
MINI560 DC-DC
Designers: Noah King, Junhee Lee
Sheet: /
File: STM-32 PCB.kicad_sch
Title: Stepper Motor Control Board
Size: A4
Date:
Rev: 1
KiCad E.D.A. kicad 7.0.11+dfsg-1build4
Id: 1/1

Appendix B

Bill of Materials for Revision 1 PCB

#	Reference [1]	QTY	Value	Manufacturer	Vendor	Manufacturer's Part ID	Cost/Part	Parts Cost
1	C1, C4, C7, C10, C17	5	10u	Samsung EM America	Digikey	CL31B106KAHNNNE	$0.12	$0.60
2	C2, C3, C5, C8, C11, C13, C16, C18	8	100n	Kemet	Digikey	C0603C104M3RACAUTO	$0.10	$0.80
3	C6, C9, C12	3	100u	Rubycon	Digikey	50YXJ100MT78X11.5	$0.36	$1.08
4	C14, C15	2	47p	Kemet	Digikey	C0603C474K3RACAUTO	$0.28	$0.56
5	D1	1	LED	QT Brightek	Digikey	QBLP615-R	$0.32	$0.32
6	J0	1	DC Barrel Jack	Adam Tech	Digikey	ADC-002-1	$0.33	$0.33
7	J1	1	01x03 Male Pin Header	Amphenol	Digikey	G800W268018EU	$2.11	$2.11
8	J2, J3, J4, J5, J6	5	01x04 Male Pin Header	Amphenol	Digikey	G800W268018EU	$0.00	$0.00
14	U2, U3, U4	6	01x08 Female Pin Header	Adam Tech	Digikey	RS1-08-G	$0.52	$3.12
14	J7, J8, J9	3	01x02 Screw Terminal	Phoenix Contact	Digikey	1725656	$1.93	$5.79
15	J10	1	USB Micro B Port	Amphenol	Digikey	10118193-0001LF	$0.41	$0.41
16	R1	1	220	Stackpole Electronics	Digikey	RMCF1206FT220R	$0.10	$0.10
17	R2, R3	2	27	Rohm Semiconductor	Digikey	ESR03EZPJ270	$0.14	$0.28
18	R4	1	4k7	Rohm Semiconductor	Digikey	ESR03EZPJ472	$0.14	$0.14
19	R5	1	10k	Rohm Semiconductor	Digikey	ESR03EZPF1002	$0.15	$0.15
20	SW1	1	SPDT Toggle Switch	E-SWITCH	Digikey	500SSP1S1M6QEA	$3.91	$3.91
21	SW2	1	SPST Momentary Switch	Omron Electronics	Digikey	B3F-1060	$0.70	$0.70
22	U0	1	MINI-560 DC-DC	Generic	Aliexpress	N/A	$0.44	$0.44
23	U1	1	STM32C071KBT3	ST Microelectronics	Digikey	STM32C071KBT3	$1.77	$1.77
24	U2, U3, U4	3	TMC2209 STEPSTICK	Generic	Aliexpress	N/A	$1.30	$3.90
25	U5	1	FTDI USB to UART Bridge	FTDI	Digikey	FT230XS-R	$2.26	$2.26
26	X1	1	ECS 48 MHZ Oscillator	ECS	Digikey	ECS-2520MV-480-CN-TR	$1.16	$1.16
							Total Parts Cost	$29.93
							Mfg. Cost	$10.54
							Total Cost	$40.47

The above table depicts the costs associated with the Motor Control Mainboard. " Parts Cost" is the cost associated with the parts present on the mainboard. "Mfg. Cost" is the cost charged by JLCPCB to manufacture the PCB. "Total Cost" is the sum of both aforementioned quantities.

[1] Items 7 and 8 in the table are sourced as a single component.

Appendix C

Bill of Materials for Robotic Arm

#	Part [1]	QTY [2]	MOQ [3]	Vendor	Manufacturer	MOQ Parts Cost	Adjusted Cost/Part $= \frac{\text{MOQ Cost}}{MOQ}$	Adjusted Parts Cost $= (\text{QTY}) \times (Adj.Cost/Part)$
1	M2 × 10mm Bolt	6	...	...	...	...	...	...
2	M2 Nut	6	...	...	...	...	...	...
3	M3 × 6mm Bolt	36	...	...	...	...	...	...
4	M3 Nut	4	...	...	...	...	...	...
5	M4 × 10mm Bolt	6	...	...	...	...	...	...
6	M4 × 16mm Bolt	8	...	...	...	...	...	...
7	M4 × 20mm Bolt	6	...	...	...	...	...	...
8	M4 Nut	14	...	...	...	...	...	...
9	Lukaisen Assorted Bolt/Nut Kit	1	1	Amazon	Lukaisen	$20.00	$20.00	$20.00
10	M6 × 20mm Bolt	1	2	Home Depot	Everbilt	$3.75	$1.88	$1.88
11	M6 × 40mm Bolt	2	2	Home Depot	Everbilt	$3.75	$1.88	$3.75
12	M6 × 50mm Bolt	1	2	Home Depot	Everbilt	$3.75	$1.88	$1.88
13	M6 Nut	4	5	Home Depot	Everbilt	$3.75	$0.75	$3.00
14	51105 Ball Bearing	1	2	Amazon	uxcell	$7.50	$3.75	$3.75
15	F624ZZ Ball Bearing	6	10	Amazon	uxcell	$9.99	$1.00	$5.99
16	F686ZZ Ball Bearing	12	20	Amazon	uxcell	$14.99	$0.75	$8.99
17	2GT-6mm Closed Belt 200mm	3	10	Amazon	3dMan	$8.99	$0.90	$2.70
18	2GT Pulley 20 Teeth	3	5	Amazon	Reliabot	$9.99	$2.00	$5.99
19	End Stop Switch	3	30	Amazon	ThtRht	$5.97	$0.20	$0.60
20	Motor 17ME15-1504S	3	1	Amazon	Stepper Online	$6.50	$6.50	$19.50
					Total Cost	$98.93	**Adjusted Total Cost**	$78.03

The above table depicts the costs associated with the Robotic Arm Assembly. "Adjusted Total Cost" is the total cost of assembling a single robotic arm adjusted for the per-unit savings of buying the parts in bulk packages. The total cost of the project is given in the "Total Cost" column. The 2nd order variables are derived according to the equations given in the column headers.

[1] Items 1 through 8 are sourced as the kit shown in Item 9.

[2] QTY is an acronym for the quantity of a part required to assemble one robotic arm.

[3] MOQ is an acronym for "Minimum Order Quantity"