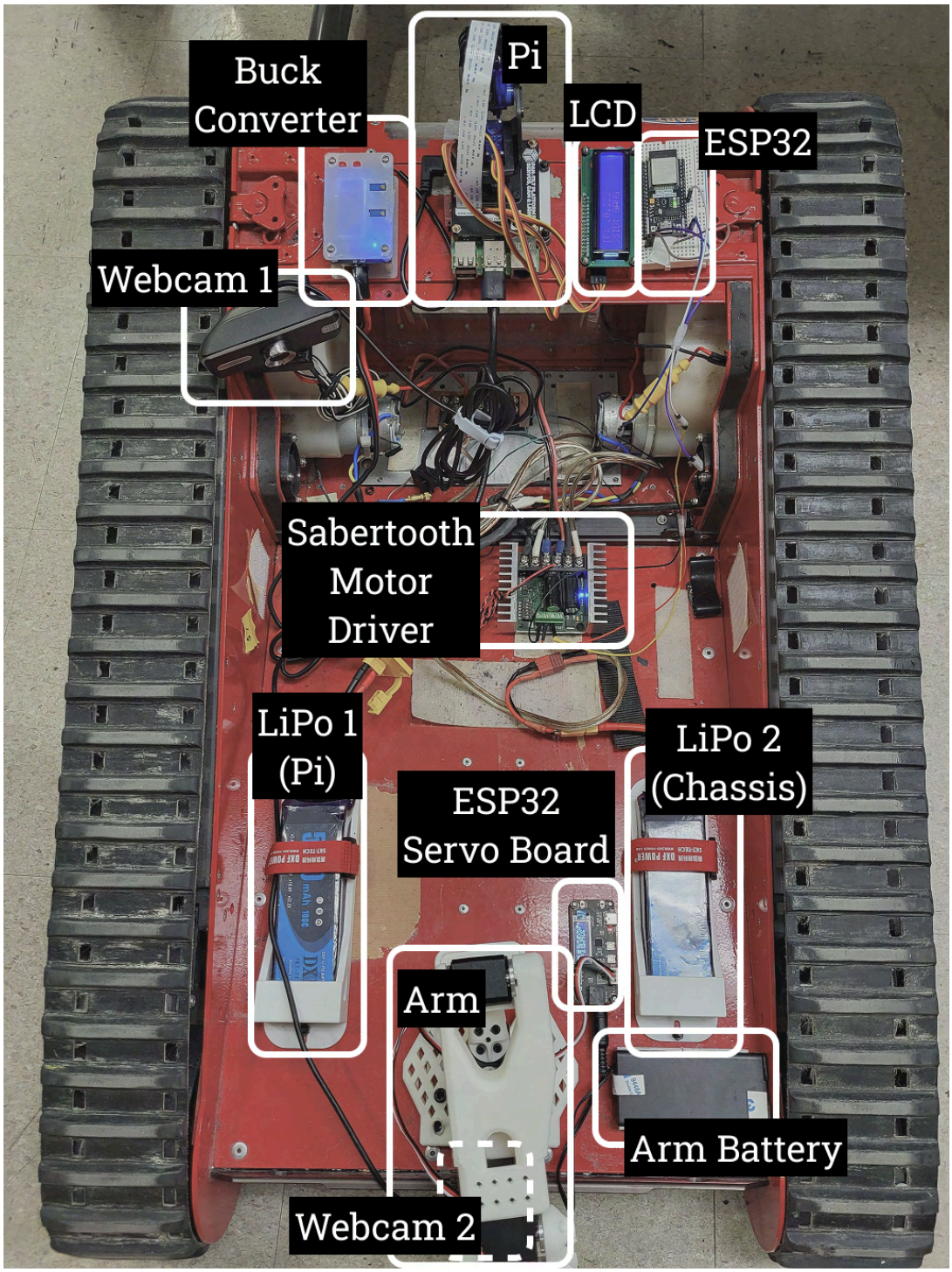
MATILDA

User Guide

1. Getting Started

1.1. Legend



Matilda has several mounted components:

- 1x DROK Buck converter
- 1x Raspberry Pi
- 1x LCD I2C Display
- 1x ESP32 Microcontroller
- 1x ESP32 Servo Driver Expansion Board
- 2x USB Webcams
- 1x Sabertooth Motor Driver
- 2x 5200 mAh 3-cell LiPo batteries
- 1x 12V 3000mAh Lithium-Ion battery bank

What is required for general usage:

- A client with a web browser (Desktop Firefox recommended)
- Program files (found on the Pi)
- A gamepad controller

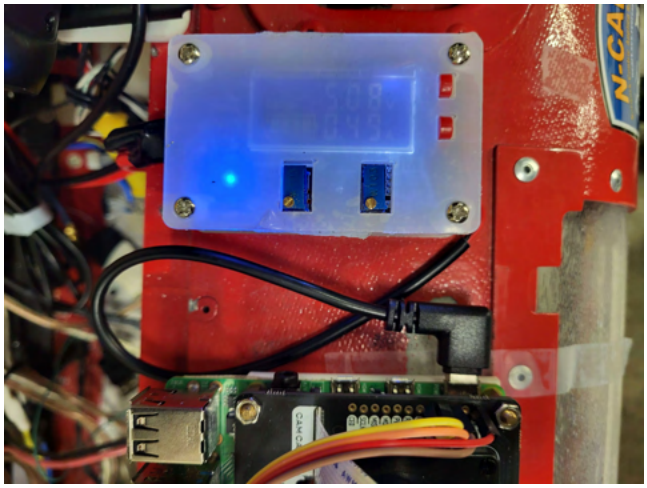
2. Construction

2.1. DROK Buck Converter

The DROK Buck Converter is attached to the chassis with 4x M3 screws. It currently intakes bare wires coming from a LiPo battery adapter (which are soldered in) and outputs to a bare-wire-to-USB-C adapter. It currently outputs ~5.1V for power supply to the Raspberry Pi.

The buck converter has two potentiometers, the left always directly controlling the output voltage, and the right controlling the current when configured in constant-load mode, which is not currently on. The LCD display outputs the current voltage and current.

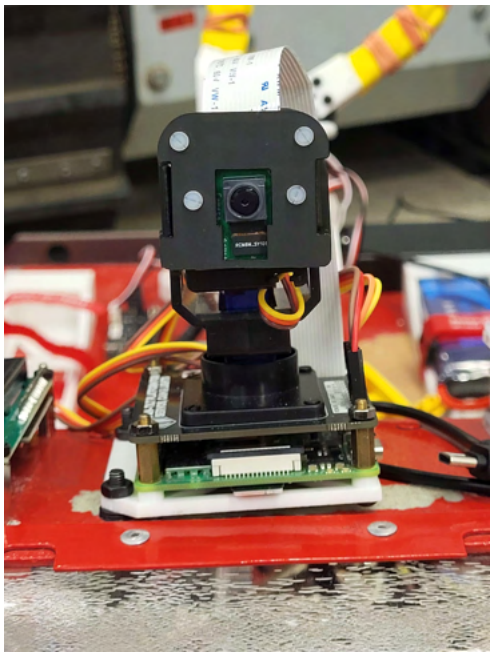
For more information, search for DROK Buck Converter 180078/180080 manual or refer to the component manual stored with Matilda.

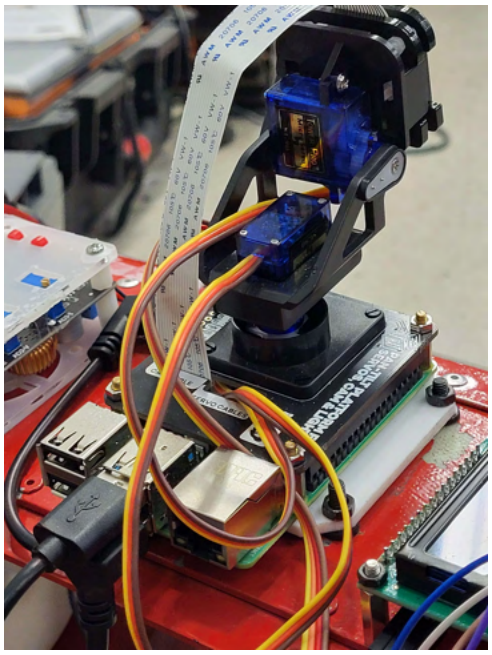


2.2. Raspberry Pi, Pi Camera 2, Pan-Tilt-Hat

The Raspberry Pi (4B, 8GB) is mounted on Matilda with 2x M4 screws via a 3D-printed mount bracket (4x M3 screws). The pan-tilt-hat complete with a Raspberry Pi Camera V2 and 2x SG90 servos is attached to the Pi via GPIO pins and secured with 4x M3 standoffs.

The Raspberry Pi hosts the web application that communicates video from the Raspberry Pi Camera 2 and any connected USB webcams (natively up to 4) to a single other client connected to the local network. It is possible to move the servos attached to the Pi Camera 2 via the web app.





2.3. LCD I2C Display

The LCD I2C display is attached to Matilda via 4x M3 screws and 4x M3 standoffs. It is connected to the Raspberry Pi's GND, VCC, SDA, SCL terminals.

Wires from the Raspberry Pi should be connected to the LCD I2C display in the following order, from left to right:

1. Brown wire: GND
2. Red wire: 5V VCC
3. Orange wire: I2C SDA
4. Yellow wire: I2C SCL

The LCD I2C display outputs the real-time WLAN IPv4 and SSID of the Raspberry Pi (with a scrolling buffer in cases of overflow).

2.4. ESP32

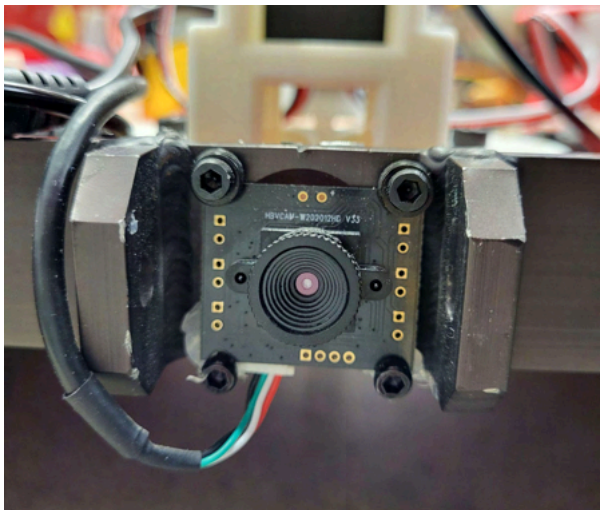
The ESP32 responsible for controlling the chassis via the Sabertooth Motor Driver is mounted into a breadboard that is attached to Matilda via double-sided tape.

Wires from the Sabertooth Motor Driver should be connected to the ESP32 in the following configuration:

1. Auxiliary GND -> ESP32 GND
2. Auxiliary VCC -> ESP32 5V
3. PWM S1 -> ESP32 Terminal 17

2.5. USB Webcams

There are two USB webcams attached to Matilda. The first is at the top-level at the chassis, attached via 2x M4 screws. The second is located at the lower-back edge of the chassis and is attached via 4x M3 screws. Both provide a rear-view of the bot to provide context when picking up objects and going down stairs.

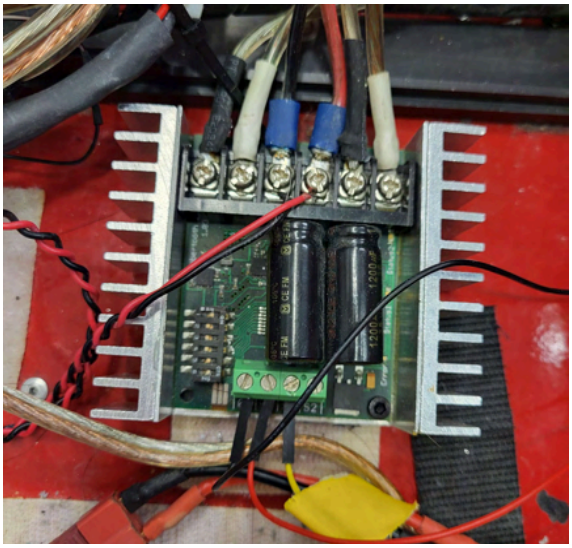


2.6. Sabertooth Motor Driver

The Sabertooth Motor Driver connects a LiPo battery to the motors of Matilda and allows control through an auxiliary module such as a radio receiver or microcontroller. There is a SWITCH that controls this power flow (O = off, I = on).

Once again, control wires from the Sabertooth Motor Driver should be connected to the ESP32 in the following configuration in the case of our ESP32:

1. Auxiliary GND -> ESP32 GND
2. Auxiliary VCC -> ESP32 5V
3. PWM S1 -> ESP32 Terminal 17



The Sabertooth Motor Driver has six switches located near the bottom-left corner that determine its configuration. In the case of our ESP32, the motor driver has the following configuration:

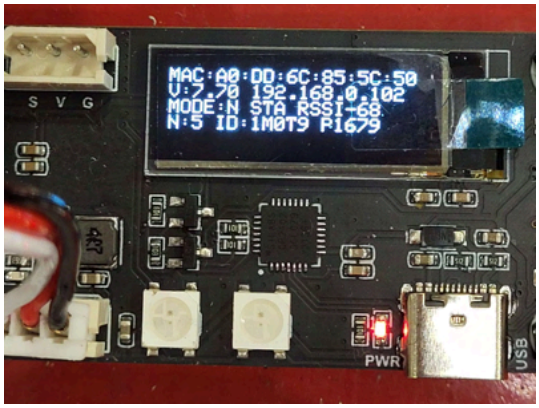
Switch	Position	Effect
1	UP	Analog input mode to be controlled by a microcontroller (not a radio receiver)
2	DOWN	
3	DOWN	Lithium cutoff since we use LiPo batteries
4	DOWN	Independent steering (not differential)
5	UP	Linear response steering (not exponential)
6	UP	Normal sensitivity (not 4x)

For more information, refer to the Sabertooth Motor Driver datasheet:
https://www.dimensionengineering.com/datasheets/Sabertooth2x25.pdf

2.7. ESP32 Servo Driver Expansion Board

The second ESP32 is expanded to control servo motors within the robotic arm. It connects to a local network to receive packets with servo positions, then sends those positions to the arm's servos. It is mounted with 4x M3 screws and standoffs. The battery bank is connected via a power connector, and the robotic arm is connected via a GND-VCC-SIGNAL cable (serial communication).

Once powered, an on-board LCD screen displays the MAC and local IPv4 addresses.

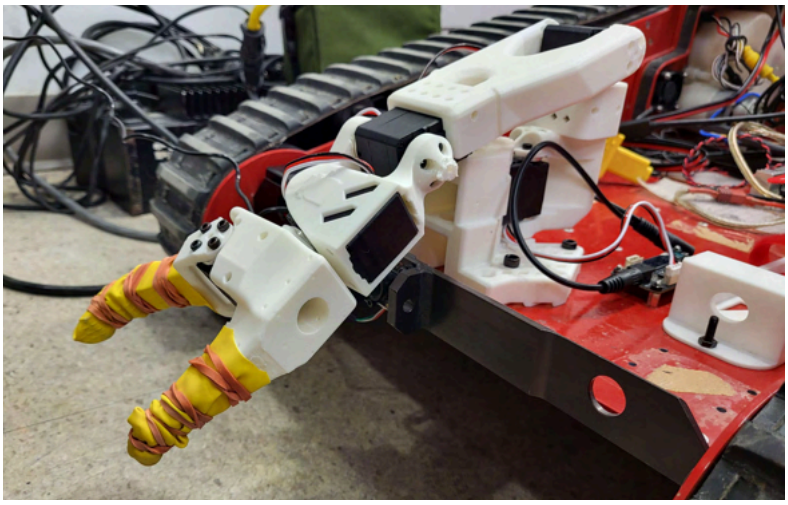


2.8. Robotic Arm

The robotic arm is made of 3D-printed joints and 12V servo motors. It has 3 degrees of freedom:

1. Claw finger, opening and closing (90 degrees)
2. Claw wrist, rotating clockwise and anticlockwise (270 degrees)
3. Elbow, lifting up and down (200 degrees)

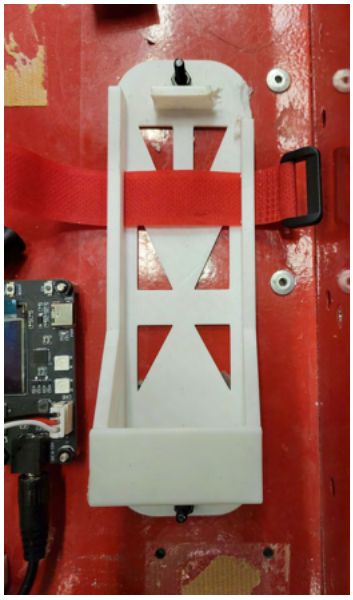
It is designed to clasp and lift loads of up to 1 pound, such as water bottles. It is mounted via 4x M5 screws (25 mm). Additionally, there are rubber gloves and elastics around the fingers of the claw to enhance grip on any held object. It connects to the ESP32 Servo Driver Expansion Board via a GND-VCC-SIGNAL cable for serial input.



2.9. LiPo Batteries

The two 11.1V 5200mAh 3S LiPo batteries are mounted via 3D-printed holders which are attached to the chassis via 2x M4 screws each.

The left LiPo connects to a XT60-to-bare-wire adapter which supplies power to the buck converter for the Raspberry Pi. The right LiPo connects to the XT60 connector which powers the Sabertooth Motor Driver and the motors.



2.10. 12V Battery Pack

The 12V 3000mAh Lithium-Ion battery pack is mounted via double-sided tape. It provides power to the ESP32 Servo Driver Expansion Board and thereby the robotic arm.

3. Software

3.1. Raspberry Pi

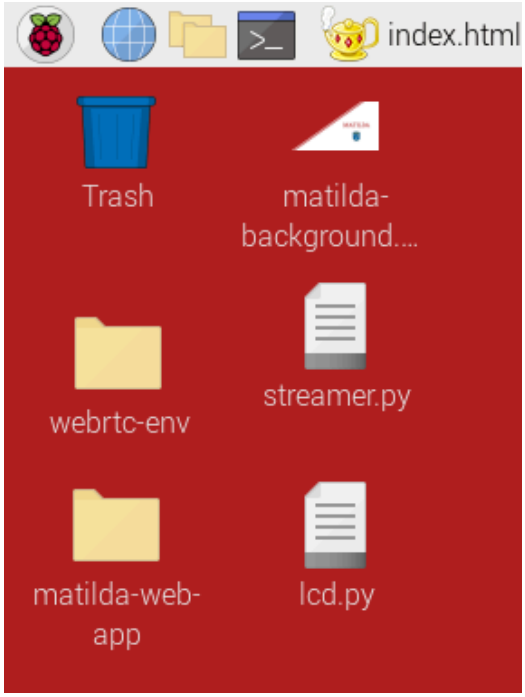
The Raspberry Pi 4B 8GB runs on Raspbian OS Bookworm and a 32 GB microSD card. It has built-in Wi-Fi scanning capabilities and connects to and remembers a chosen network (this can be done via the GUI or the `sudo raspi-config` command.

It is configured to run the `startup-tasks.service` service defined at `/etc/systemd/system/startup-tasks.service`. This runs the bash script located at `/home/cps040/startup.sh` which does the following:

1. Go to the Node.js web application folder/environment `~/Desktop/matilda-web-app`
2. Run the Node.js web application `app.js`

3. Sleep for 5 seconds to give server a chance to start
4. Go to `~/Desktop` and activate the Python virtual environment required to properly run the following scripts
5. Run `lcd.py` which displays the local IPv4 address and SSID of the current WLAN
6. Run `streamer.py`which connects to the web app and streams video + controls to a single client

All code relevant to the Raspberry Pi's operations is located on the Desktop.



3.2. ESP32 (Chassis)

The ESP32 runs on C and Python code. The scripts define the wireless network to which to connect and the IP it wants to be designated, and receive UDP packets that it then translates into PWM signals on physical terminal 17 (which should be connected to the Sabertooth Motor Driver).

The code that is currently hosted on the ESP32 has a copy on the Raspberry Pi at `~/Desktop/matilda-web-app/public/software.zip` (`esp-chassis` folder within the archive). This also means it can be downloaded over the local network when the Pi is powered and running the web app (`http://[PI-LOCAL-IP]:3000/software.zip`). New code is flashed to the ESP32 via the USB-C port when connected to a client with the Arduino Developer Environment.

3.3. ESP32 Servo Driver Expansion Board (Arm)

The ESP32 Servo Driver Expansion Board runs on C code. The scripts define the wireless network to which to connect, and receive UDP packets that define the servo positions for the arm.

The code that is currently hosted on the ESP32 Servo Driver Expansion Board has a copy on the Raspberry Pi at `~/Desktop/matilda-web-app/public/software.zip` and is within the `esp-arm` folder within the archive. This also means it can be downloaded over the local network when the Pi is powered and running the web app (`http://[PI-LOCAL-IP]:3000/software.zip`).
New code is flashed to the ESP32 via the USB-C port when connected to a client with the Arduino Developer Environment.

4. Operation

4.1. Network Setup

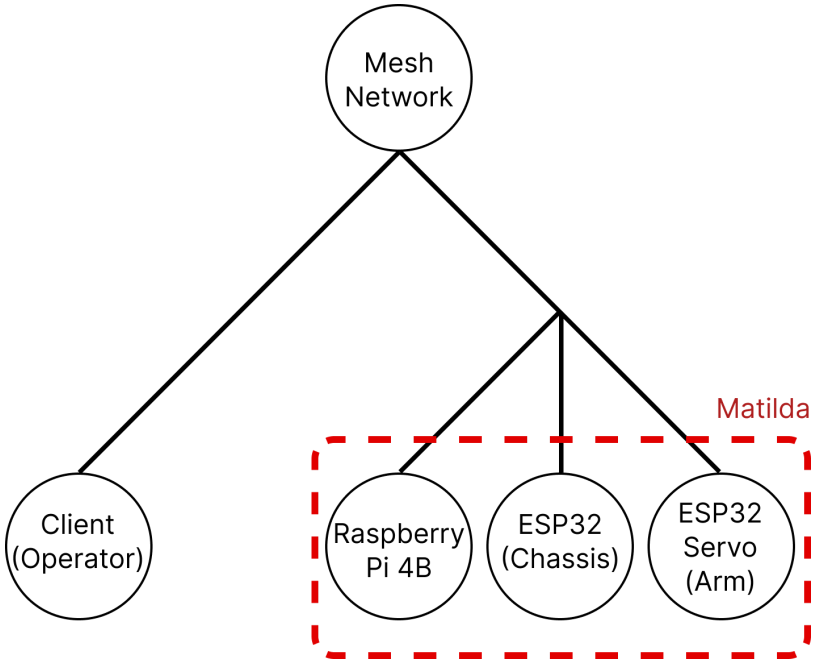
The Matilda N-CART kit operates on a local network. Each on-board computer must be connected to the same local network:

1. Raspberry Pi
    a. Connect Raspberry Pi to a monitor and access the GUI desktop or CLI
    b. Use the Network configurator via the GUI or `sudo raspi-config` command to select and connect to your desired local network.
    c. The Pi's IP is displayed on the LCD when powered and connected on startup (see Section 2.3. LCD I2C Display)
2. ESP32
    a. Obtain a copy of the current code (see Section 3.2. ESP32 (Chassis))
    b. Edit `char* ssid`, `char* password` and `IPAddress local_IP(X,X,X,X)`
    c. Flash the new code via a USB-C connection and the Arduino Develop Environment
3. ESP32 Servo Driver Expansion Board
    a. Obtain a copy of the current code (see Section 3.3. ESP32 Servo Driver Expansion Board (Arm))
    b. Edit the variables for the desired SSID and password (`STA_SSID`, `STA_PWD`)
    c. The ESP32's IP is displayed on the LCD when powered and connected on startup (see Section 2.7. ESP32 Servo Driver Expansion Board)

NOTE: Please ensure you have administrative control over the local network to which you are connecting. The network must support local clients accessing each other via IP protocols (UDP, TCP). To ensure smooth operation, access the settings of the local network and assign each device a specific static IP based on their MAC addresses:

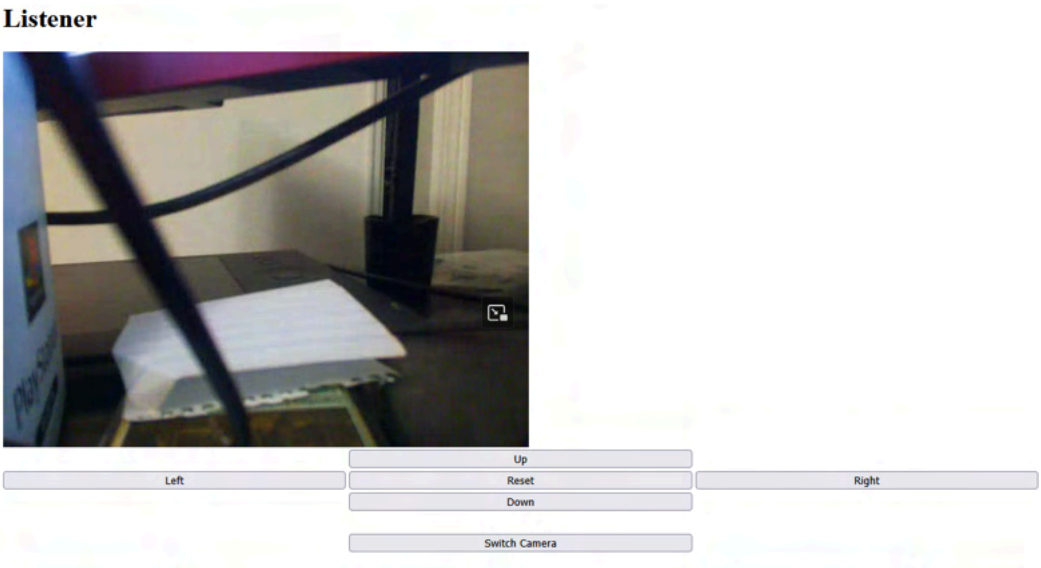
Device	MAC Address
Raspberry Pi 4B 8GB	D8-3A-DD-45-BD-35
ESP32 (Chassis)	EC-C9-FF-E1-8B-84
ESP32 Servo Driver Expansion Board (Arm)	A0-DD-6C-85-5C-50

Take note of all IP's manually or automatically assigned to each device.



4.2. Video Streaming Application

Once the Pi is powered and connected to the local network, with its IP displayed on the LCD display, use another locally-connected device (e.g. a phone, laptop, desktop) with a web browser to visit the web application at `http://[PI-LOCAL-IP]:3000`.



Controls:

- Up/Down/Left/Right/Reset for Raspberry Pi Camera 2 panning/tilting
- Switch Camera to loop through the camera array (Pi Camera 2 + USB webcams)

Here are recommendations for smooth operation:

- Use a device with a medium-to-large screen to effectively see the video stream and camera controls at once.
- Use Firefox and not a Chromium-based browser
- Once the website loads once, wait patiently for a WebRTC connection to be established and for the gray image to disappear
- If the gray image does not disappear after 30 seconds, the website may have loaded too early or incorrectly, so try refreshing
- Press the `Reset` button in the camera controls after the video loads to ensure proper positioning
- Occasionally press the `Reset` button in the camera controls during operation, as the camera can become offset during rough movement
- When using the `Switch Camera` function, only press it once and patiently wait for the stream to change. Do not spam this button.
- Use `Switch Camera` to change to rear-view cameras when picking up objects
- If the stream becomes too choppy, consider reloading the webpage (This will also let you know if the connection to the Pi as a whole has been lost)
- Only use the camera controls when viewing the Pi Camera 2 stream and not a USB webcam as it does not apply to them

4.3. Chassis & Arm Control

The ESP32 should be powered, connected to the Sabertooth Motor Driver (see Section 2.4. ESP32 and Section 2.6. Sabertooth Motor Driver), and connected to the local network (see Section 3.2. ESP32 (Chassis) and Section 4.1. Network Setup).

The ESP32 Servo Driver Expansion Board should be powered, connected to the Robotic Arm (see Section 2.7. ESP32 Servo Driver Expansion Board and Section 2.8. Robotic Arm), and connected to the local network (see Section 3.3. ESP32 Servo Driver Expansion Board (Arm) and Section 4.1. Network Setup).

On an external, locally-connected laptop or desktop, obtain a copy of the `client-controller` code from `http://[PI-LOCAL-IP]:3000/software.zip` (if the Pi is running and connected to the local network) or `~/Desktop/matilda-web-app/public/software/client-controller` directly.

Edit the variables `ROBOT_IP` and `ARM_IP` in each script to reflect the local IPs of the ESP32 and the ESP32 Servo Driver Expansion Board (see Section 4.1. Network Setup).

Connect a gamepad (e.g. PS5 Dualsense, Xbox One controller) to your device and run the appropriate file with Python 3+:

Controller	File
PS4 Dualshock 4	control-DS4.py
Other	control.py

NOTE: You must have the Python library `pygame` installed (run `pip install pygame` if it is missing).

This program communicates controls to the chassis and robotic arm via UDP packets, and is controlled by the connected gamepad:

File	Controls

control-DS4.py	• Left Joystick Up/Down
	  ○ Chassis Forward/Backward
	• Right Joystick Left/Right
	  ○ Chassis Right/Left
	• D-PAD Left/Right
	  ○ Robotic Arm Down/Up
	• D-PAD Up/Down
	  ○ Robotic Arm Rotate CW/CCW

	• X-Button
	  ○ Robotic Arm Close Grip
	• O-Button
	  ○ Robotic Arm Open Grip

control.py	• Left Joystick Up/Down
	  ○ Chassis Forward/Backward
	• Right Joystick Left/Right
	  ○ Chassis Right/Left
	• D-PAD Up/Down
	  ○ Robotic Arm Up/Down
	• Left/Right Bumper
	  ○ Robotic Arm Rotate CCW/CW
	• Left/Right Trigger
	  ○ Robotic Claw Close/Open Grip

5. Switching to Radio Mode

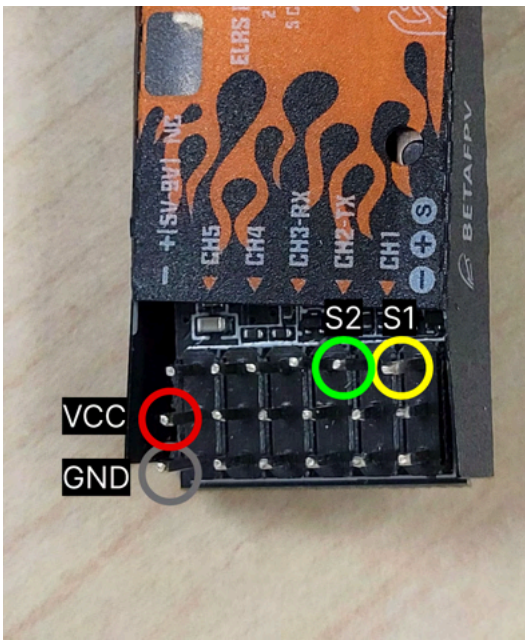
In the case that radio is the desired protocol for controlling the chassis, certain changes need to be made:

The Sabertooth Motor Driver should have the following configuration:

Switch	Position	Effect
1	DOWN	Analog input mode to be controlled by a radio receiver (not a microcontroller)
2	UP	
3	DOWN	Lithium cutoff since we use LiPo batteries
4	UP	Differential steering (not independent)
5	UP	Linear response steering (not exponential)
6	UP	Normal sensitivity (not 4x)

The ESP32 that is connected to the Sabertooth Motor Driver should be removed and replaced with the ELRS Micro Receiver:

1. Auxiliary VCC -> Top Row, 5V-9V terminal
2. Auxiliary GND -> Top Row, GND terminal
3. S1 PWM -> Bottom Row/CH1, Signal terminal
4. S2 PWM -> Second Last Row/CH2, Signal terminal



Once wired, the Radiomaster Pocket ELRS controller can be powered on. It should automatically connect with the ELRS Micro Receiver as they have already been paired. However, if it does not connect, quickly remove and restore power to the ELRS Micro Receiver three times to enter pairing mode, and then search again with the Radiomaster Pocket controller.

The Radiomaster Pocket ELRS controller should now control Matilda with the left joystick alone, using the two channels it affects. There are still three channels available to be wired on the ELRS Micro Receiver, so additional functionality can be added (e.g. robotic arm control, up to 3 DOF).

For more information, visit https://betafpv.com/products/elrs-micro-receiver and https://cdn.shopify.com/s/files/1/0609/8324/7079/files/Pocket_1.pdf?v=1736839330.