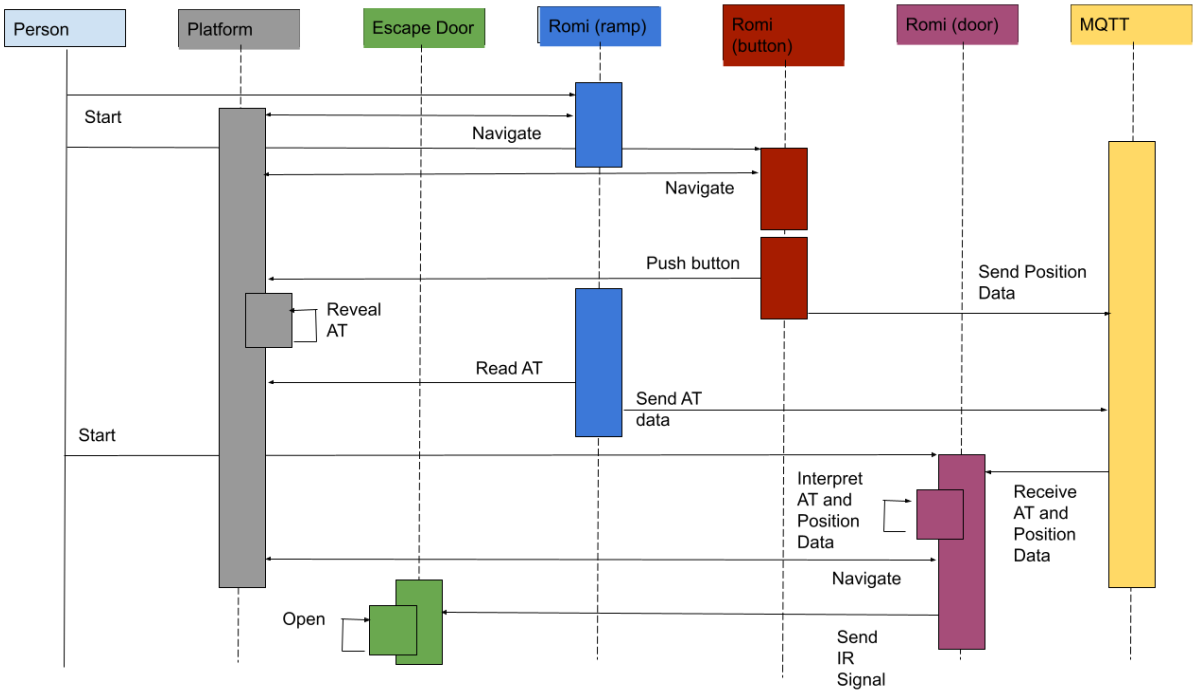
Final Report

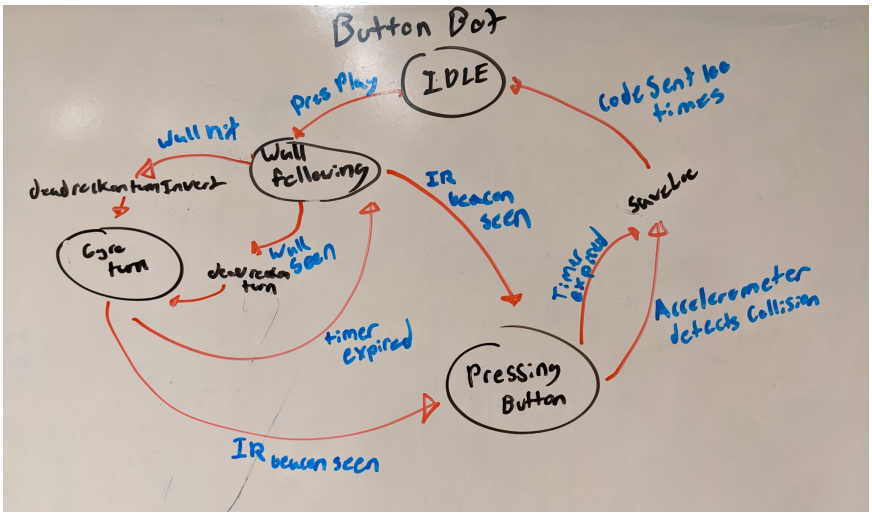
Team 15: Bryce McKinley, Aiden Higuera, Tristin Youtz

Sequence Diagram:

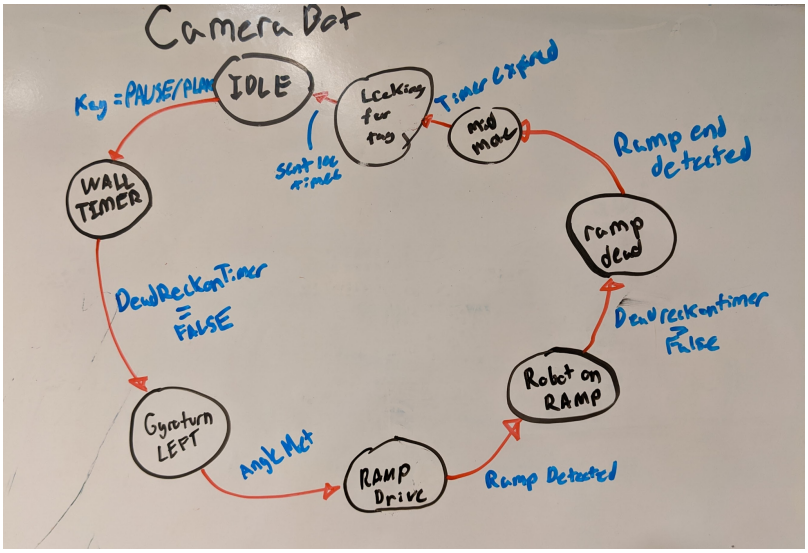


State Diagrams:

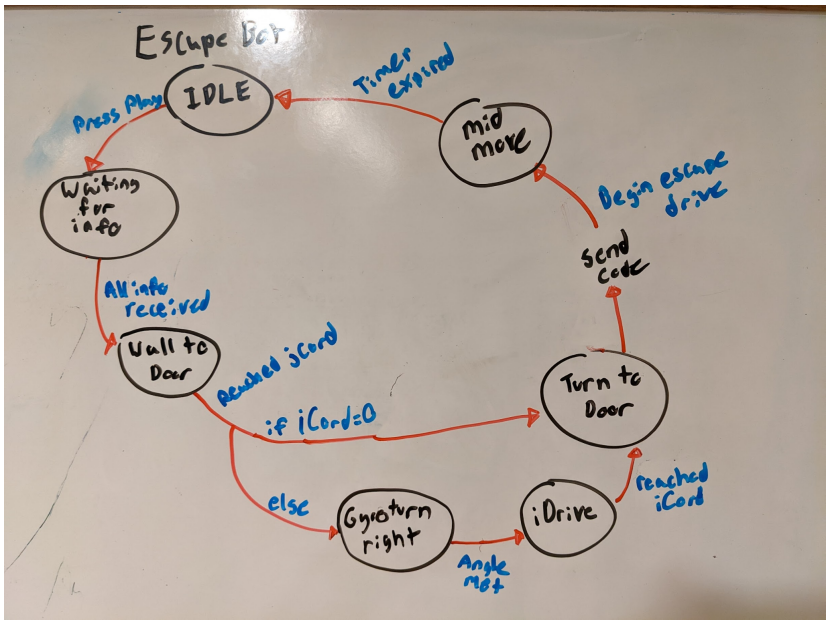
Button Bot:



Camera Bot:

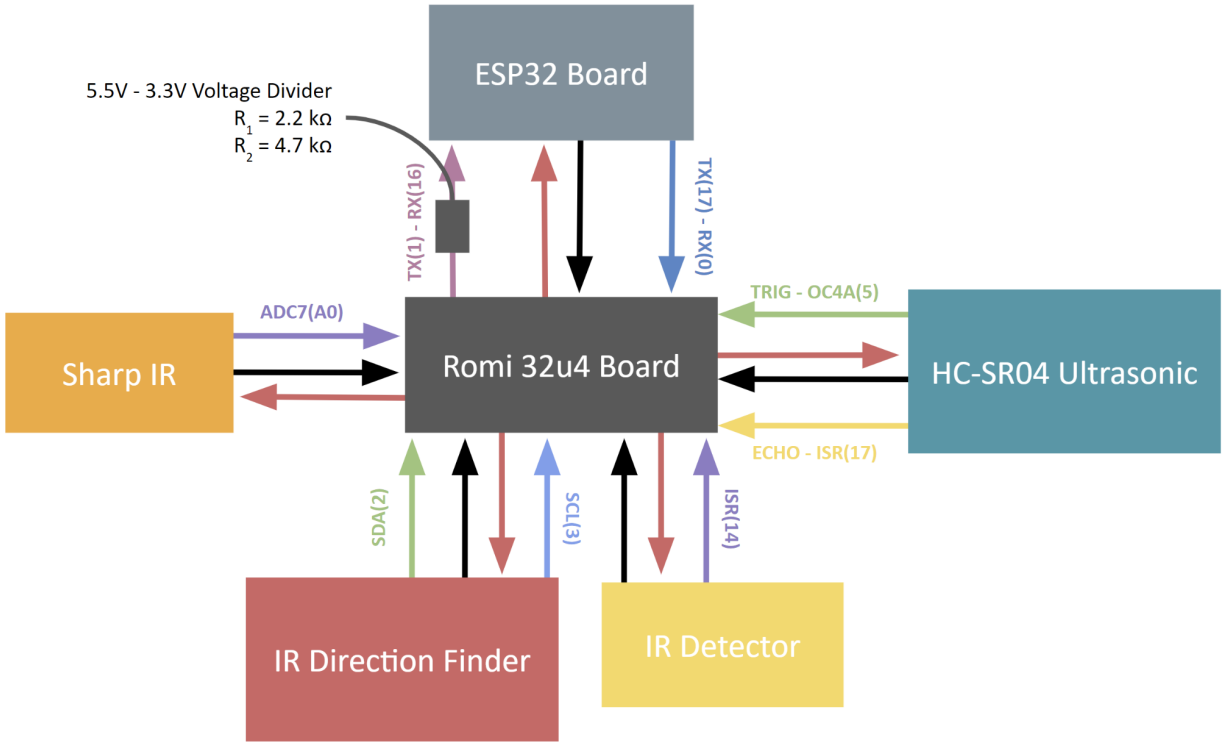


Escape Bot:



Interface Diagrams:

Button Bot:



ESP32:

The ESP32 is primarily used for connection to the MQTT, which is valuable for sending and accessing data for the other robots on the team. This uses a UART interface, which is composed of a TX and RX pin for each member respectively. The TX and RX pins for the 32u4 are 1 and 0 respectively, while the ESP32 TX and RX pins are 17 and 16 respectively. Note that the TX pin on the ESP32 sends out 5V, which is much greater than the ESP32's 3.3V interface, so the Romi TX input needed to be passed through a voltage divider as shown.

Sharp IR:

There were multiple sensor options for the task of wall-following: the HC-SR04 Ultrasonic sensor, the Sharp IR Proximity Sensor, and the Maxbotix Sensor. For the task of wall following, the highest priorities were speed of readings and consistency. Accuracy is important, but there was an assumption that all of the sensors would have a base level of accuracy when implemented correctly. Based on our experience with the Sharp IR from previous labs, the readings from the sensor were both incredibly fast and remarkably consistent when placed in the right orientation on the robot. Another bonus was the ease of use, as the Sharp IR only required power, ground, and an ADC input for the readings themselves.

HC-SR04 Ultrasonic:

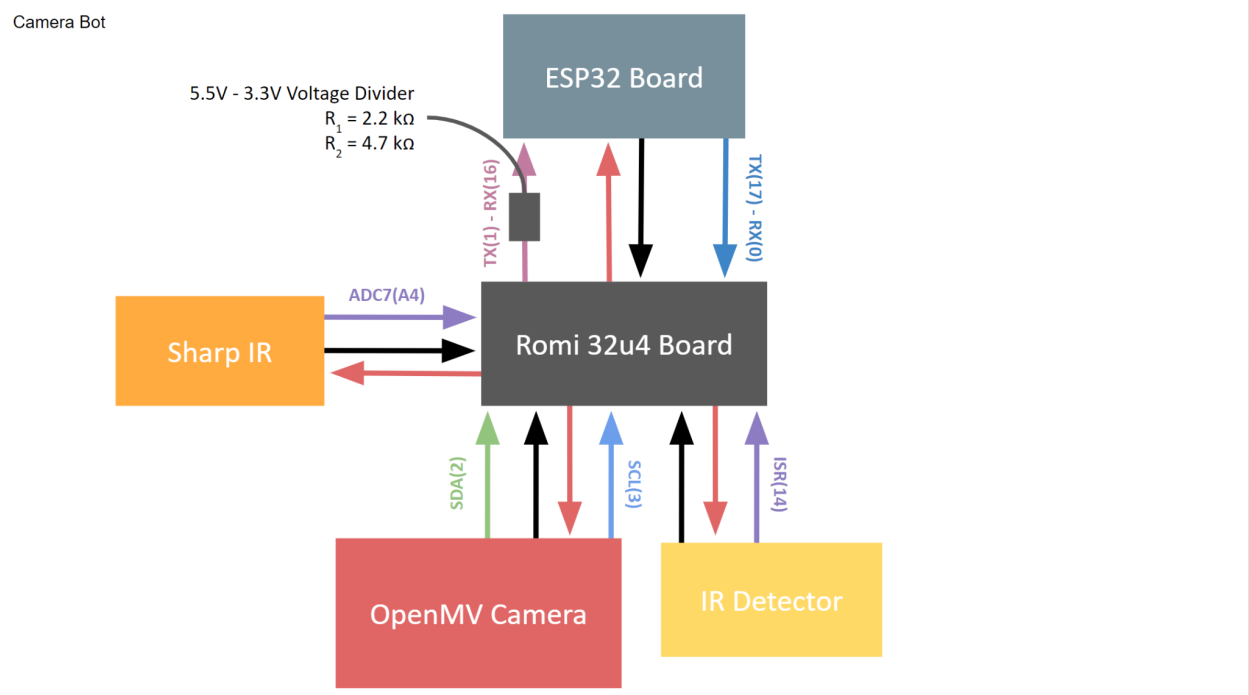
This sensor was used for detecting front walls, but we did not always plan for this sensor for this task. After some initial difficulty setting up the Ultrasonic, we decided early on to switch to the Maxbotix. Unfortunately, we found that the Maxbotix struggled in both speed and accuracy. Due to a low sampling rate and high amount of noise, the sensor struggled to reliably find a distance threshold and tell the robot to turn when the threshold was reached. We eventually switched back

to the Ultrasonic, which worked flawlessly once we implemented a timer to time the read requests to when new readings were available.

IR Direction Finder:
This sensor's purpose was for following the IR beacon from the button on the field. This sensor had a basic camera setup, where the sensor would report the signal from a range of 0 to 1023 in both x and y coordinates. There are two specific notes about the IR direction finder: it is designed to track up to 4 unique signals at a time, and if there are no detected signals the sensor reads 1023 for all four of its inputs. This requires that a very small for loop is set up to cycle between the sensor's 4 tracking inputs so that the camera always has a visual on the beacon. Additionally, we can filter out inputs that do not read anything simply by only looking at inputs that are less than 1023.

IR Detector:
This is a very simple sensor just for reading inputs from the IR remote. This allows us to give the robot specific instructions, such as starting or pausing procedures or sending predetermined MQTT values, based on specific button presses. The setup for this sensor is also very simple: the sensor only requires power, ground, and an interrupt pin. Most of the calculations for how to handle frequencies and signals is accounted for in software. The only downside to this system is that it is not associated with a unique remote ID, so anyone can press a button with their remote to mess up the robot.

Camera Bot:



ESP32:
The ESP32 is primarily used for connection to the MQTT, which is valuable for sending and accessing data for the other robots on the team. This uses a UART interface, which is composed of a TX and RX pin for each member respectively. The TX and RX pins for the 32u4 are 1 and 0

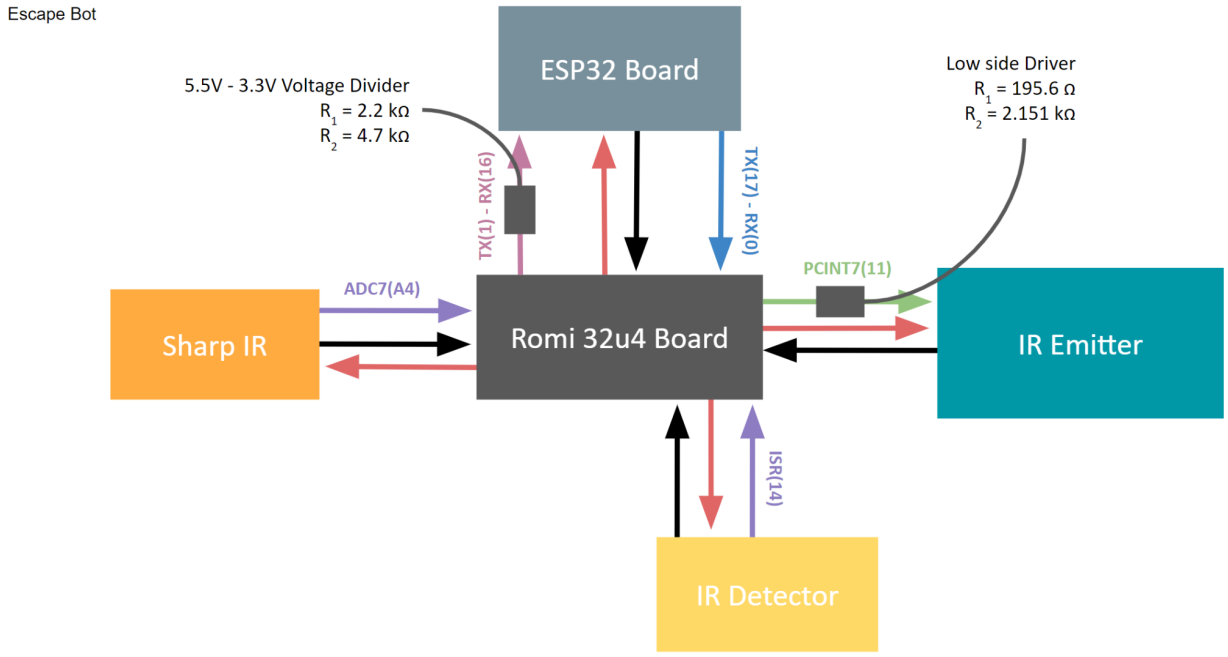
respectively, while the ESP32 TX and RX pins are 17 and 16 respectively. Note that the TX pin on the ESP32 sends out 5V, which is much greater than the ESP32's 3.3V interface, so the Romi TX input needed to be passed through a voltage divider as shown.

Sharp IR:
There were multiple sensor options for the task of wall-following: the HC-SR04 Ultrasonic sensor, the Sharp IR Proximity Sensor, and the Maxbotix Sensor. For the task of wall following, the highest priorities were speed of readings and consistency. Accuracy is important, but there was an assumption that all of the sensors would have a base level of accuracy when implemented correctly. Based on our experience with the Sharp IR from previous labs, the readings from the sensor were both incredibly fast and remarkably consistent when placed in the right orientation on the robot. Another bonus was the ease of use, as the Sharp IR only required power, ground, and an ADC input for the readings themselves.

IR Detector:
This is a very simple sensor just for reading inputs from the IR remote. This allows us to give the robot specific instructions, such as starting or pausing procedures or sending predetermined MQTT values, based on specific button presses. The setup for this sensor is also very simple: the sensor only requires power, ground, and an interrupt pin. Most of the calculations for how to handle frequencies and signals is accounted for in software. The only downside to this system is that it is not associated with a unique remote ID, so anyone can press a button with their remote to mess up the robot.

OpenMV Camera:
The OpenMV Camera is a remarkably powerful camera capable of a variety of image-based processes due to its onboard processing and dedicated IDE. There was a choice of interfacing with the camera using either I2C or UART connections. For this project we decided to use I2C, due to the fact that it leaves the serial pins of the Romi open, allowing us to more easily connect to the ESP32. The OpenMV camera was used to detect April Tags and report a number of their characteristics, namely ID and rotation. Though the sensor had some minor issues over the term, such as needing to have a full hardware reboot, it was overall incredibly reliable and intuitive to use.

Escape Bot:



ESP32:
The ESP32 is primarily used for connection to the MQTT, which is valuable for sending and accessing data for the other robots on the team. This uses a UART interface, which is composed of a TX and RX pin for each member respectively. The TX and RX pins for the 32u4 are 1 and 0 respectively, while the ESP32 TX and RX pins are 17 and 16 respectively. Note that the TX pin on the ESP32 sends out 5V, which is much greater than the ESP32's 3.3V interface, so the Romi TX input needed to be passed through a voltage divider as shown.

Sharp IR:
There were multiple sensor options for the task of wall-following: the HC-SR04 Ultrasonic sensor, the Sharp IR Proximity Sensor, and the Maxbotix Sensor. For the task of wall following, the highest priorities were speed of readings and consistency. Accuracy is important, but there was an assumption that all of the sensors would have a base level of accuracy when implemented correctly. Based on our experience with the Sharp IR from previous labs, the readings from the sensor were both incredibly fast and remarkably consistent when placed in the right orientation on the robot. Another bonus was the ease of use, as the Sharp IR only required power, ground, and an ADC input for the readings themselves.

IR Detector:
This is a very simple sensor just for reading inputs from the IR remote. This allows us to give the robot specific instructions, such as starting or pausing procedures or sending predetermined MQTT values, based on specific button presses. The setup for this sensor is also very simple: the sensor only requires power, ground, and an interrupt pin. Most of the calculations for how to handle frequencies and signals is accounted for in software. The only downside to this system is that it is not associated with a unique remote ID, so anyone can press a button with their remote to mess up the robot.

IR Emitter:
The IR emitter is a very simplistic electrical device which acts as an IR emitting LED. For this project, we had to use it to send values to an IR receiver which allows for a door to open. In order to do this we needed to implement a low side divider off of the Romi's interrupt pin 11, which controlled the timing of the pulses from the emitter. Sending a digit required us to send a preamble of 9ms HIGH, then 4.5ms LOW, followed by a binary address and its inverse, then the digit in binary and its inverse, ended by one last zero. Zero was denoted by a 535 microsecond HIGH then a 535 microsecond HIGH, while a one was a 535 microsecond HIGH then a 1688 microsecond LOW. The signal is also sent using a 38kHz signal because that is what is required by the IR detector. The method to send these signals unfortunately had a blocking effect in our code, but it resulted in a successful code transmission to open the door as needed.

Code Releases:

Button Bot:
https://github.com/RBE-2002/lab-2-wall-following-team-15/releases/tag/FinalProject

Camera Bot:
https://github.com/RBE-2002/lab-2-wall-following-team-15/releases/tag/CameraBotFinal

EscapeBot:
https://github.com/RBE-2002/lab-2-wall-following-team-15/releases/tag/EscapeBotFinal

Annotated Video:

https://youtu.be/K_7oJrs-G8U