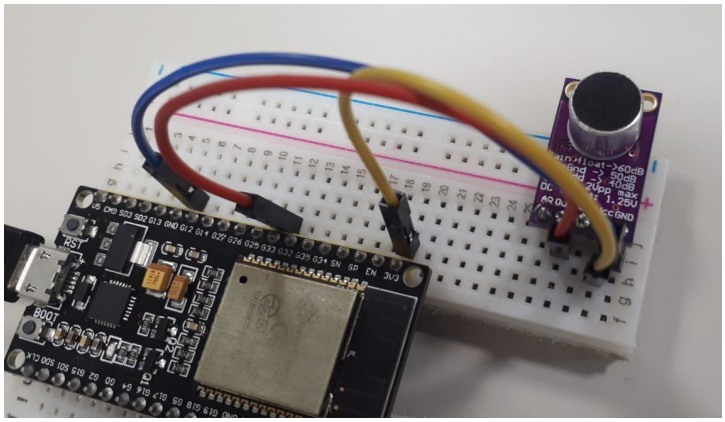
Drummer Project Documentation

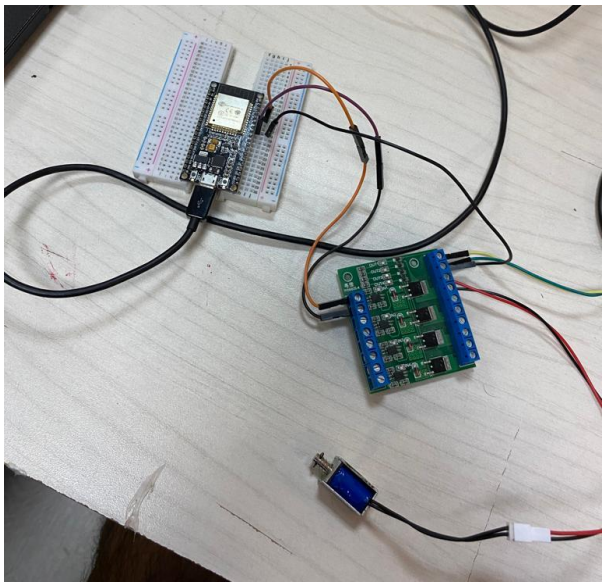


ESP32 Quick Start:

- We followed the instructions to get a feel for how the ESP32 works

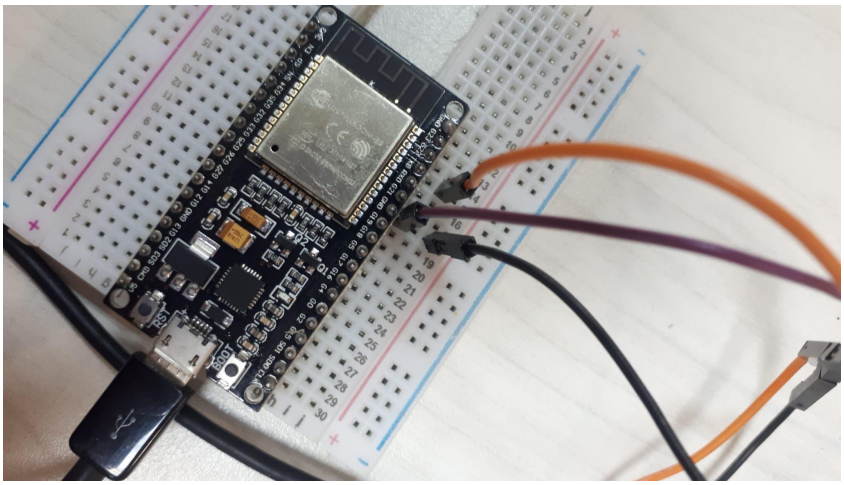
Testing the Mics:

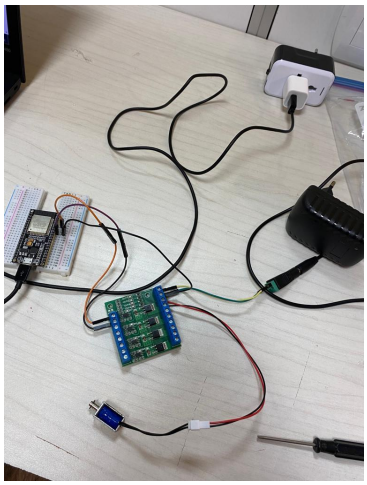
- We tested the mics using the set up of cables shown in the video and with the code from the second link
    - Image to the right of how we connected the mic
    - https://www.youtube.com/watch?v=2waBFdEBZDg
    - https://www.atomic14.com/2020/09/12/esp32-audio-input.html
- We found that the preamp mic max9814 is good at picking up sound and filtering out noise
- We found that the red 4 pin mics needed the sound to be loud in order to pick it up (not as good as the preamp max9814)
- We found that the red 3 pin mics is good at picking up sound and filtering out noise (the same as the preamp max9814)



Testing the Small Solenoid:

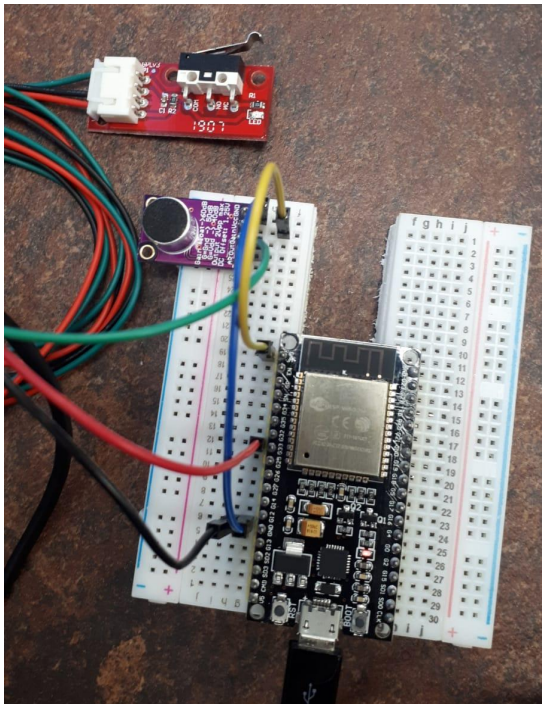
- We use the cable set up as follows-
    - Orange - on mosfet connected to PWM1, on esp connected to TX0 (which is pin 1)
    - Purple - on mosfet connected to GND1, on esp connected to GND
    - Black - this is the ground sharing - connected to GND on the esp, and on mosfet connected to DC-
- We upload the code onto the esp32 (using the basic Blink example to visually see what is going on) and the code from the link below. This is when the esp is connected to the laptop through the USB cable
    - https://core-electronics.com.au/guides/solenoid-control-with-arduino/
- Next we unplug the USB from the computer and plug it into the wall. At this point we should see the LED's blinking in the esp and the mosfet.
- Then we plug in the 12V power source (for the mosfet) into the wall and the solenoid moves as we set it too (switches on and off).
- We notice that the solenoid gets stuck and gets very hot after only a few cycles.

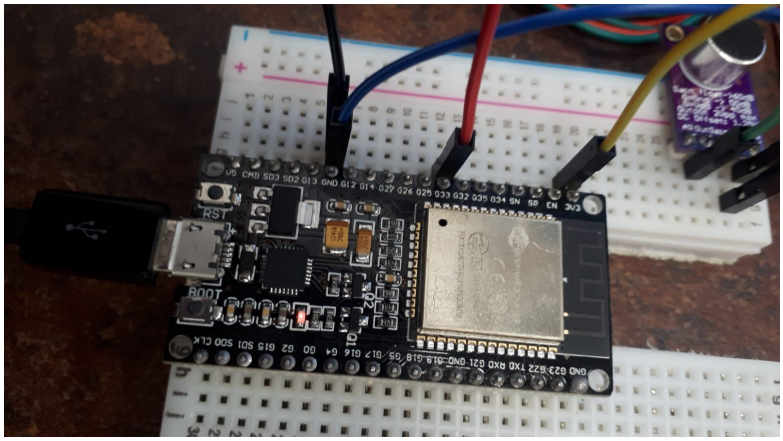




Pedal with the mic:

- https://reprap.org/wiki/Mechanical_Endstop
- We connected the switch to the mic signal as shown in the picture.
- We connected the switch to the signal input/output so that when we press on the switch it the mic “stops recording” and when we let pressure off of the switch the mic “continues recording”
- When the switch is held down, the mic does not record any input.
- We were unable to have the switch work with our hit detection code - the mic continuously got high sound values. This is possibly due to close power cables.

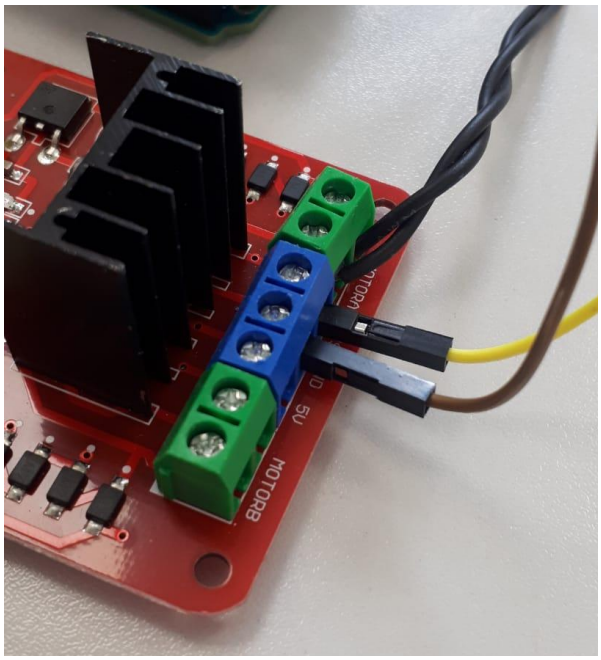


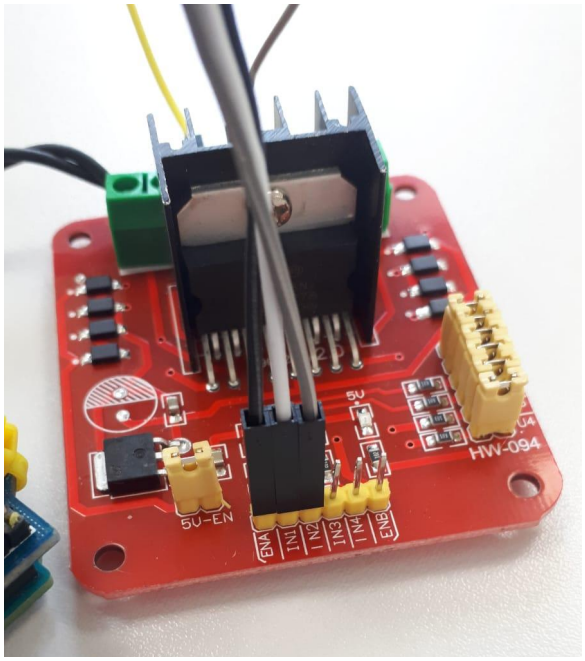


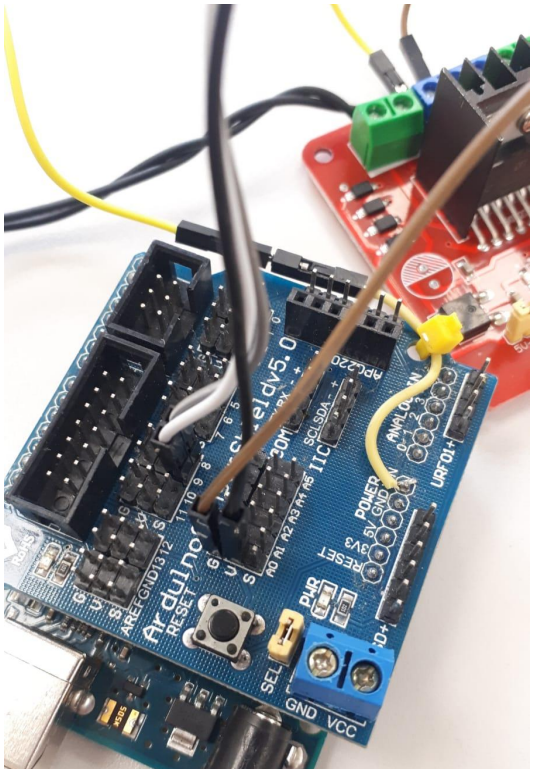
Testing the bigger solenoid:

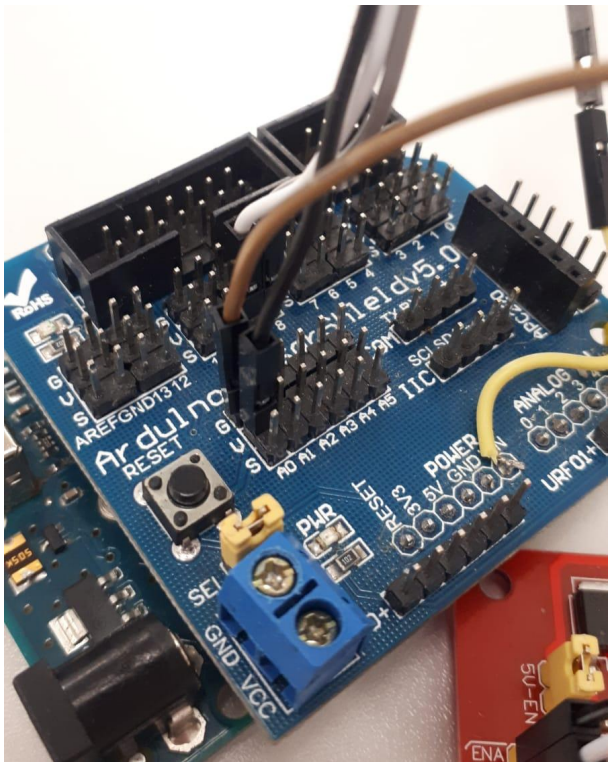
- We got the big solenoid and tested it using the arduino attached.
- We send a code with high/low digital write which causes the solenoid to make a hit. We then disconnect the arduino from the computer and plus the external power source into the wall

- We figured out the optimal heights to put the solenoid at for loud, medium, and soft hits.
    - Make sure that the stand is super stable to get the most accurate sounds
- We are working on detecting the strengths of the sounds correctly.
- Trying to understand the arduino situation and how to transfer the cables to the esp32:
    - How the arduino/mosfet/solenoid was originally set up when we got it

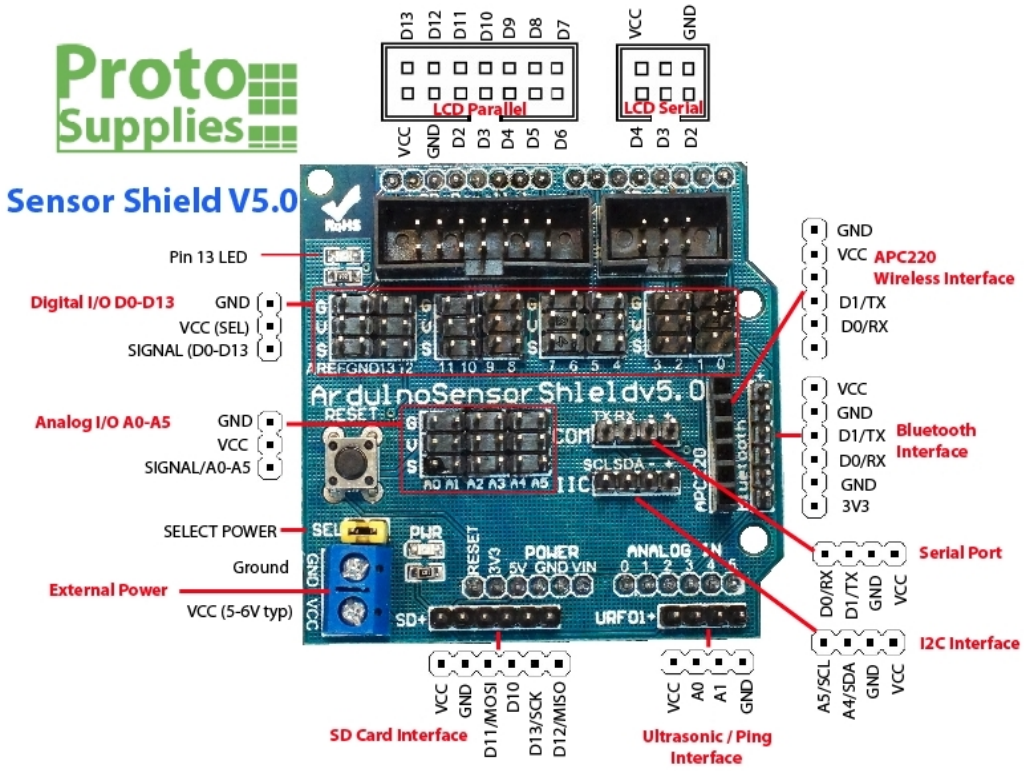






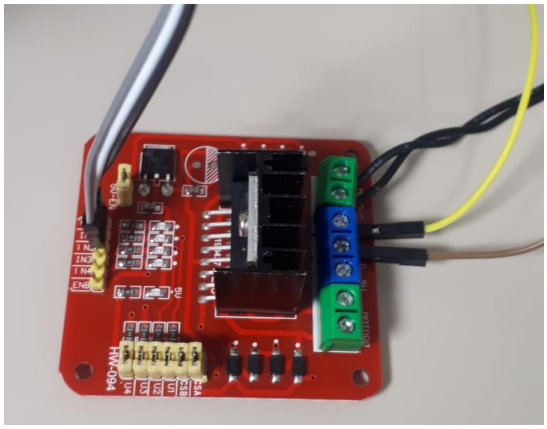


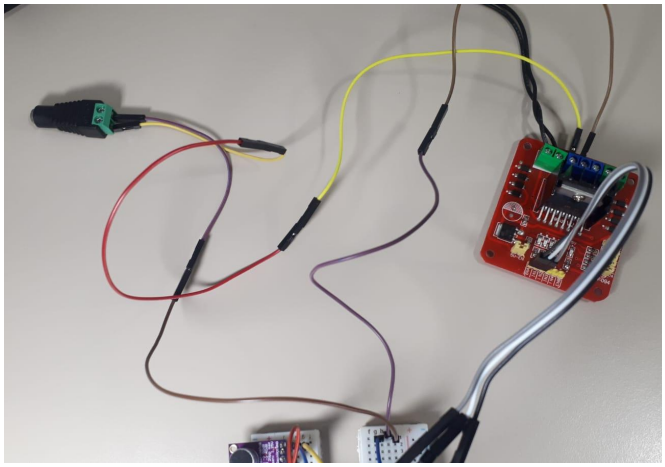
    - We used this diagram to try to understand the arduino

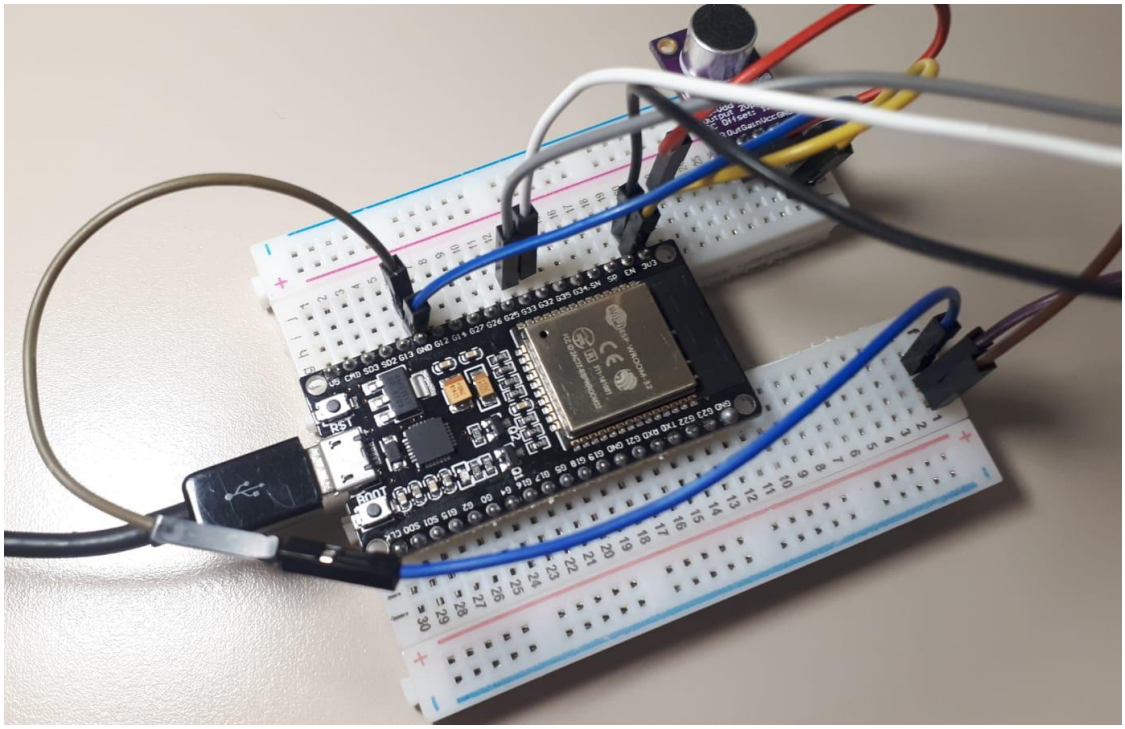


- Next we try to understand using the following links:
  - https://forum.arduino.cc/t/using-a-mosfet-to-control-a-governor-pressure/950177
  - https://create.arduino.cc/projecthub/ryanchan/how-to-use-the-l298n-motor-driver-b124c5 *****
  - https://acoptex.com/project/167/basics-project-033b-l298n-dual-h-bridge-motor-driver-module-one-or-two-dc-motors-9v12v-at-lex-c/ *****
  - https://www.electroduino.com/introduction-to-l298n-motor-driver-how-its-work/
  - https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/
  - Google - arduino solenoid L298
  - Google - arduino solenoid mosfet
  - Important to remember: After loading the code to the esp32 through the usb, disconnect the usb, plug the solenoid power source (big black box) into the wall, and do not forget to also plug the esp32 into the wall using the iphone charger
- Pictures of how we connected the solenoid to the esp32:
  - Brown/purple/blue cables are ground - we shared the ground between the power source and the esp through the breadboard
  - Yellow/red cables are connected to the power source for power
  - Black cable - connected to ENA on mosfet and 3v3 on ESP32
    - Must be connected for solenoid to work
    - Can connect to v5 instead
  - White cable - connected to IN1 on mosfet and pin 32 on ESP32

- Grey cable - connected to IN2 on mosfet and pin 33 on ESP32 – check what happens if disconnect and only use IN1 in code (solenoid_test)
  - https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/#:~:text=The%20IN1%20and%20IN2%20pins,various%20combinations%20and%20their%20outcomes.
  - https://esp32io.com/tutorials/esp32-dc-motor
- In the pictures the mic is not connected to anything (has a yellow, blue and red cable)

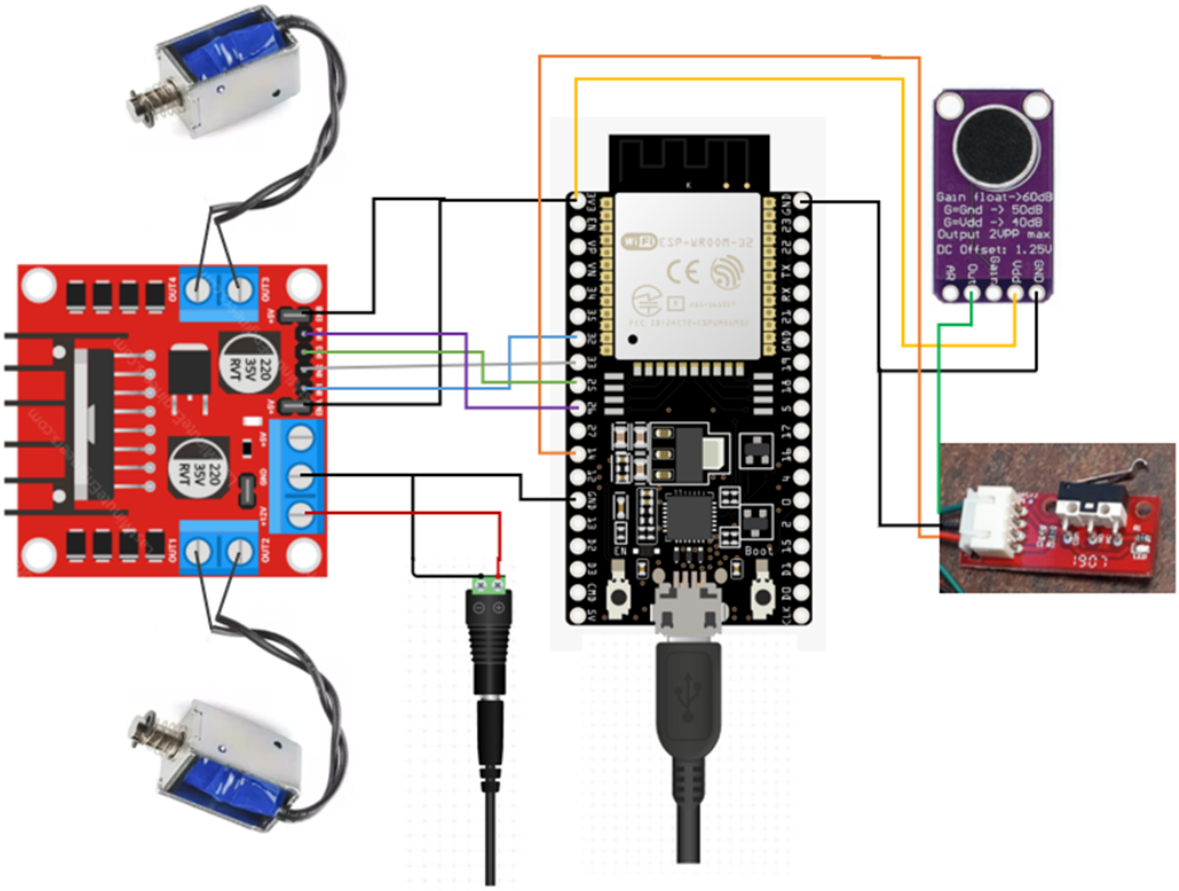






- We used the solenoid_test file to test that the solenoid does indeed work.
  - The only thing we are confused about is why in the arduino had IN1, and IN2 all connected since there is only one solenoid
  - We are able to make the solenoid hit just by using IN1

- We connected the second solenoid using IN3 and IN4 - IN3 connected to pin 25 and IN4 to pin 26. Below is a diagram of what we currently have:



(Note that since this point we have updated the wiring, a final diagram is in the README.)

<u>Switching to VS Code with PlatformIO</u>:

- We did this because it will be easier to have all the code together in one place (for the machine learning and controlling the solenoids/mic)
- We used the following links (given to us from Yoav):
  - https://www.youtube.com/watch?v=JmvMvIphMnY
  - https://docs.platformio.org/en/stable/integration/ide/vscode.html
  - https://randomnerdtutorials.com/vs-code-platformio-ide-esp32-esp8266-arduino/
- Remember to press the BOOT button on the ESP32 if uploading the code doesn't work or you get this error:
  - A fatal error occurred: Failed to connect to ESP32: Wrong boot mode detected (0x13)! The chip needs to be in download mode.

<u>Sound detection</u>:

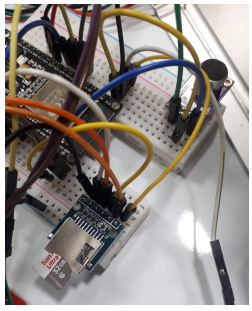
- We can detect when there is a hit/loud sound when the number/variable MicVolume, is higher than a certain threshold.

- Since we see in the serial monitor that for one hit, there are multiple high values recorded, we have to be careful that due to this we do not detect multiple hits. To make sure that we detect the hit as ONE hit and not multiple, we use a counter and a flag in the code.
- We can detect 2 strengths/intensities of hits.
- When the solenoid hits the drums, sometimes the mic picks up a softer sound first and only then the louder sound. To prevent a soft hit being recorded instead of a loud hit we wait for a few cycles to see if a loud sound is recorded and if it is, instead of being classified as soft, it is classified as loud.
- Keep in mind that there should not be a lot of background noise.

Playback on a small scale - storing the data buffer:

- Challenging working with the timestamps in the buffer - we are currently trying to use millis() but it does not seem to be keeping the correct timestamps (millis() - returns the current time in milliseconds from when we powered up the board)
    - Some intermediate research we did before finding the solution which we will actually use:
    - Look here - https://www.best-microcontroller-projects.com/arduino-millis.html
    - Also can try to google using millis to find offsets or something like difference in time
    - Look here too - https://learn.adafruit.com/multi-tasking-the-arduino-part-1/using-millis-for-timing
    - Maybe here - https://duino4projects.com/arduino-timer-millis/
    - https://github.com/RobTillaart/timing
- We found the following library driver/timer.h
    - https://embeddedexplorer.com/esp32-timer-tutorial/
- Trying to get the timer to work with a blinking LED code
    - Use help from the following
    - https://github.com/espressif/esp-idf/blob/v4.3/examples/peripherals/timer_group/main/timer_group_example_main.c
- Recording and storing the data buffer on the sd card
    - https://www.instructables.com/DIY-ESP32-Recorder/
    - https://randomnerdtutorials.com/esp32-microsd-card-arduino/
    - Connecting the SD card:

MicroSD card module	ESP32
3V3	3.3V
CS	GPIO 5
MOSI	GPIO 23
CLK	GPIO 18
MISO	GPIO 19
GND	GND



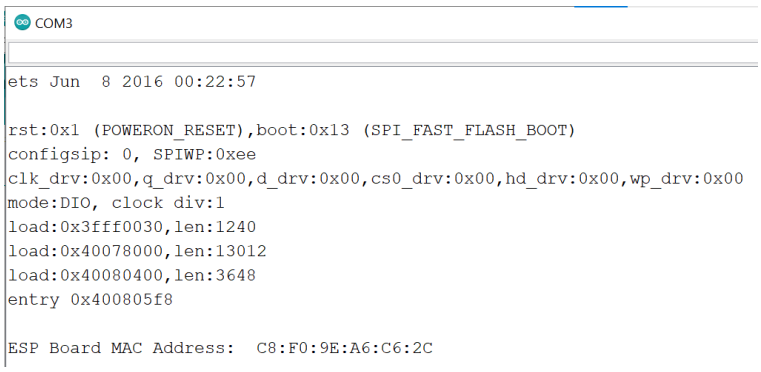
- How to get the data from the mic properly in order to store it on the SD
- I2S protocol setup

    - https://docs.espressif.com/projects/esp-idf/en/v4.2.3/esp32/api-reference/peripherals/i2s.html#_CPPv416i2s_pin_config_t
    - https://github.com/espressif/esp-idf/tree/v4.2.3/examples/peripherals/i2s
    - https://software-dl.ti.com/simplelink/esd/simplelink_cc13x2_26x2_sdk/4.20.00.35/exports/docs/tidrivers/doxygen/html/_i2_s_8h.html#ti_drivers_I2S_Example_RepeatMode
    - https://github.com/atomic14/esp32_sdcard_audio
    - https://medium.com/codex/the-complete-guide-to-recording-an-analog-microphone-with-esp32-to-an-sd-card-60440ec2d1a2
    - https://www.youtube.com/watch?v=pPh3_ciEmzs
        - https://github.com/atomic14/esp32_audio
    - https://www.youtube.com/watch?v=m-MPBjScNRk
    - https://github.com/atomic14/diy-alexa/blob/master/firmware/src/main.cpp
    - https://www.hackster.io/esikora/audio-visualization-with-esp32-i2s-mic-and-rgb-led-strip-4a251c
    - https://github.com/esikora/M5StickC_AudioVisLed/blob/main/src/M5StickC_AudiVisLedApp.cpp
- We decided to not use that protocol and to just record the micVolume in an array. We will give all modes a time constraint because otherwise we could have problems with memory.
- We decided not to use the SD card and the protocol so that we could focus our time on creating the application for the user interface.

Bluetooth - BLE:

- Setting up the bluetooth server client thing for the esp
    - https://www.instructables.com/ESP32-BLE-Android-App-Arduino-IDE-AWESOME/
    - https://www.malabdali.com/bluetooth-low-energy-ble/
- Diagram of how the esp and app are related with the bluetooth
    - https://www.youtube.com/watch?v=AFGC9iYBxFQ
- https://github.com/sbis04/flutter_bluetooth/blob/master/lib/main.dart
- https://www.youtube.com/watch?v=fQz7aajslJc
- https://blog.codemagic.io/creating-iot-based-flutter-app/

- Found mac address of the esp32 using the device_address.ino code
    - C8:F0:9E:A6:C6:2C



- Having the app recognize that the phone is connected to the esp over bluetooth:

    - https://www.youtube.com/watch?v=AFGC9iYBxFQ
    - ***https://github.com/husamhamu/mm_app
- Classic bluetooth vs BLE
    - https://circuitdigest.com/microcontroller-projects/using-classic-bluetooth-in-esp32-and-toogle-an-led
- General explanation
    - https://developer.android.com/guide/topics/connectivity/bluetooth

Application - User Interface:

- We decided to use an app for the user interface so that there are less problems with user error. (With physical buttons or switches there could be a lot of errors or unexpected actions that can occur.)
- Running the app on a phone
    - https://appmaking.com/run-flutter-apps-on-android-device/
    - https://kobiton.com/blog/develop-deploy-and-test-flutter-apps/

Calibration mode:

- In order to account for the user's changing environment, we enable him to calibrate what he defines to be a loud hit and a soft hit. We record the user's loud hit twice, and take the average in order to define a threshold. We then do the same for the soft hit. Without calibrating the system, the user cannot go on to the other modes.
- The time given to record a hit is 3 milliseconds.

Metronome mode:

- The function has a loop that runs forever, and produces a metronome at a rate specified by the app. The weaker solenoid hits beats_per_measure times, and then the stronger solenoid hits, and the process repeats until "cmd" is changed.

Playback mode:

- The mode starts when the user presses the pedal to record. As long as the pedal is pressed, hits are recorded, and then finally when the user releases the pedal, the beat is played back to the user. The user can then keep going with the same process and record new beats to be played once each time. The linked list containing all hits is deleted after each beat played.

Looper mode:

- The mode starts when the user presses the pedal to record. As long as the pedal is pressed, hits are recorded, and then finally when the user releases the pedal, the beat is played back to the user. Unlike playback mode, the user can now keep inputting hits into the beat - at any given time during the beat, the user can jump in by pressing the pedal to record, and then hitting the drum. The next time the beat is played, it will contain the new hit, and will keep playing it in a loop. The user can even decide to hold the pedal down the entire time, and be able to record hits at any given time as he goes. We added the instantaneous pedal feature in order to enable the user to play the drum with the

looper, without the looper adding the new hits into the beat. In order to avoid a situation where the pedal is pressed, and the new hit recorded is by the solenoid and not the user's, we check the whole linked list to see if the hit exists already. We do that by checking if a hit within 300 milliseconds of the new hit_time is already in the linkedlist. If it isn't, we take that it is a user hitting the drum and inputting a new hit and so we add it to the linked list in the correct place - the linked list is in an ascending hit_time order.

Interactive mode:

- The mode starts when the user presses the pedal, and a beat is recorded. Once the pedal is released, we iterate over the linked list containing hits' information and input the data into the neural network, which in turn returns a class number the beat is associated with. Then, depending on the class, we randomly select one of three possible responses for that class to respond to the user's beat with, and then produce it using the solenoids. The user is then able to repeat the process by pressing the pedal again, and then the mode will respond with the appropriate beat.
- In the end, although the neural network did work on the desktop, once we integrated it into the project workspace we kept getting the same class as the output, this may be related to the model binary management in cpp. We tried looking for examples and different possible solutions including changing the buffer management and allocating more memory space for the model since this could be an issue and we don't fully understand what may be causing this. For giving an interactive experience anyway in this mode we generated a random value between 1 and 4 in order to decide the response of the drummer.
- Tensorflow lite - https://github.com/atomic14/tensorflow-lite-esp32