ECPRE 492 Team 10 Final Report

How design has evolved since ECPRE 491

The design has shifted focus to more on how a project similar to this could be implemented into the ISU ECPRE curriculum as a class or part of a class. We focused on creating a PCB schematic and a lesson plan along with a physical model. The lesson plan is an outline with several lab activities that could be adapted to different classes with different lab lengths. It focuses on teaching students the basics of machine learning and embedded programming using Arduinos for hardware and Edge Impulse for the machine learning library.

Requirements & Constraints

- Incorporate machine learning to an embedded system.
- When the user says their keyword, the door (un)locks.
- Users must agree to having voice recorded with purchase of lock.
- Appropriate technical complexity for an ISU course.
- Easy for the user to program a keyword/easy install.
- Machine learning accuracy of 90% on keyword recognition.
- Use an embedded system that is small enough to fit on a door lock (not Pi, Arduino Mega, etc.).
- Stay within the physical bounds of the microcontroller (memory, CPU speed, etc.).
- Be able to lock/unlock within 5 seconds of confirmation.
- Easy access to data sets/easy creation of data sets.
- Use tools and skills learned from the Coursera Embedded ML course.

Standards

Use IEEE standards when applicable

- IEEE P2089- Standards for Age Appropriate Digits Services
- IEEE P2660.1- Recommended Practice on Industrial Agents: Integration of Software Agents and Low Level Automation Functions.
- IEEE P2817 - Guide for Verification of Autonomous Systems.
- IEEE P2840 - Standard for Responsible AI Licensing Standard for Responsible AI Licensing
- IEEE P7001 - Transparency
- IEEE P7002 - Data Privacy

ANSI Grade 3- Standard/safety requirements for locks (Strength & Durability)

BHMA Residential Security Grade C- Standard/safety requirements for locks (Strength, Durability, Finish)

Legal standards for recording with/without someone's consent

Privacy standards for saving/distributing user audio data

Wifi/Bluetooth communication standards

Security concerns and countermeasures

Test for false positives in our machine learning algorithm:

-Try and use a recording of someone's spoken keyword

Implementation was going for a smaller design that could easily be installed on a door knob like Google Nest products or various other products that are slightly bigger than the lock itself.
Testing circuit designs

While creating the initial hardware design the goal was not only to create a working prototype but also to take into account the size and ease of use of the system. My goal was to keep the design fairly clean in terms of wiring in order to minimize the chances of errors along with increasing the portability of the system. Our team did not end up utilizing all of the components that were initially thought to be required. Over the course of the projects and different versions we were able to decrease the complexity of the system and decrease the size to eventually fit better on a PCB.

Physical System Testing
Testing the physical system was a bit of an issue with getting tolerances correct to make the parts mesh together. This was due to 3D printing tolerances not being as accurate as we would have liked to have on the first time printing the pieces. Cutting some of the metal pieces took time to get them within the tolerances as well.

Testing the Machine Learning Model
When data is uploaded to Edge Impulse, 80% is used to train the model, and 20% is saved for testing. After training the model, Edge Impulse runs the model with the test data to see how it performs. The testing accuracy should only be slightly lower than training accuracy, if it is much lower that is a sign of overfitting.

Testing the Code
Unit 1 - Testing the code was split into different sections:
Machine learning model testing:
Testing the keyword recognition was done through the serial monitor. Testing involved saying different words, including the keywords, and having its inference results displayed on the serial monitor

Unit 2 - Motor control:
A sketch was created and functions to control the motors were created. The roller switches were added to the board and the code was manually tested. The procedure was to start the arduino which started one of the motors running, then pressing down on the appropriate roller switch should stop the motor. This process was repeated for every motor.

Once the motors were running and passed individual testing, testing the whole motor setup proceeded. The arduino would be powered on and the stepper motor would start running. When the appropriate switch was pressed, the stepper motor would stop and the dc motor would start. When the appropriate switch was pressed, then the dc motor would stop and the stepper would move the opposite direction for 0.1 seconds. This simulates the stepper motor engaging the gear train, the dc motor running the gear train to lock/unlock the door, and then the stepper motor disengaging from the gear train. After testing and debugging the motors all consistently worked properly.

Integration Testing:
Once both units were tested, integration testing was performed. This involved testing so that when a keyword was spoken, the code section to run the motors was entered. Then the switches were pressed in the appropriate order to simulate the door locking. This was performed several times without cutting power to the arduino.

Appendix 1 - User Manual

Physical Setup

Physically the prototype consists of:

- An Arduino Nano 33 BLE Sense
- A 9V battery
- A 3.3v regulator
- 3 momentary roller switches wired from the regulator to the Nano
- A servo motor
- A DC motor
- A Motor driver, wired between the Nano and the DC motor
- A door lock attached to a ‘door’
- A gear train attached to the motors.

The battery powers the Nano, the motors, and the switches.
On hearing the correct keyword, the nano runs the motors in a specific pattern in order to engage the gear train, lock/unlock the door lock, and disengage the gear train.

Code Setup

The program “sketch” is created with Arduino IDE, which aids in writing and uploading C/C++ code to embedded systems such as the Arduino Nano used for the project. The code uses several built in libraries to control the stepper motor and the speed of the DC motor. The code also uses the machine learning algorithm that we trained on Edge Impulse.

The code initializes the serial monitor and the nano’s on-board microphone. It then enters a loop where it continually reads data from the microphone and adds it to an array. That array is then given to the machine learning library and the library processes the audio array and classifies the audio as either: The word “House”, the word “Marvin”, background audio noise, or some unknown word.
If keyword House or Marvin is not recognized, nothing happens and the program reiterates.
If either keywords House or Marvin is recognized. The code enters a subsection that:

1. Turns the stepper motor to engage it with the gear chain, the motor turns until it trips a switch which tells it that the motor is in the correct position
2. Decides whether to lock or unlock depending on the current state
3. Wake the DC motor (it sleeps to lessen power consumption)
4. Locks/Unlocks using the DC motor, which spins until trips a switch which tells it that the motor is in the correct position
5. Changes the locked/unlocked state
6. Puts the DC motor back to sleep
7. Disengages the stepper motor from the gear chain so that the lock can be used manually.

The full sketch code is given in Appendix 4.

Set-up and Testing

1. Connect the circuit together using the PCB (fig. 3) schematic or if not using a PCB use the Wiring Diagram (fig.1) to set up the circuit.
2. Connect a 9v battery to the circuit.
3. There should be a solid green light on the side of the arduino nano if the device is properly powered.
4. Say either keyword (HOUSE, MARVIN) within a foot or two of the arduino nano's microphone.
5. If word is recognized, motors will begin locking/unlocking.
6. If the word is not recognized, nothing will happen.
7. Say other words, some similar to the keywords, some not similar, to test for false positives. The system should not lock/unlock when any of these words are spoken.

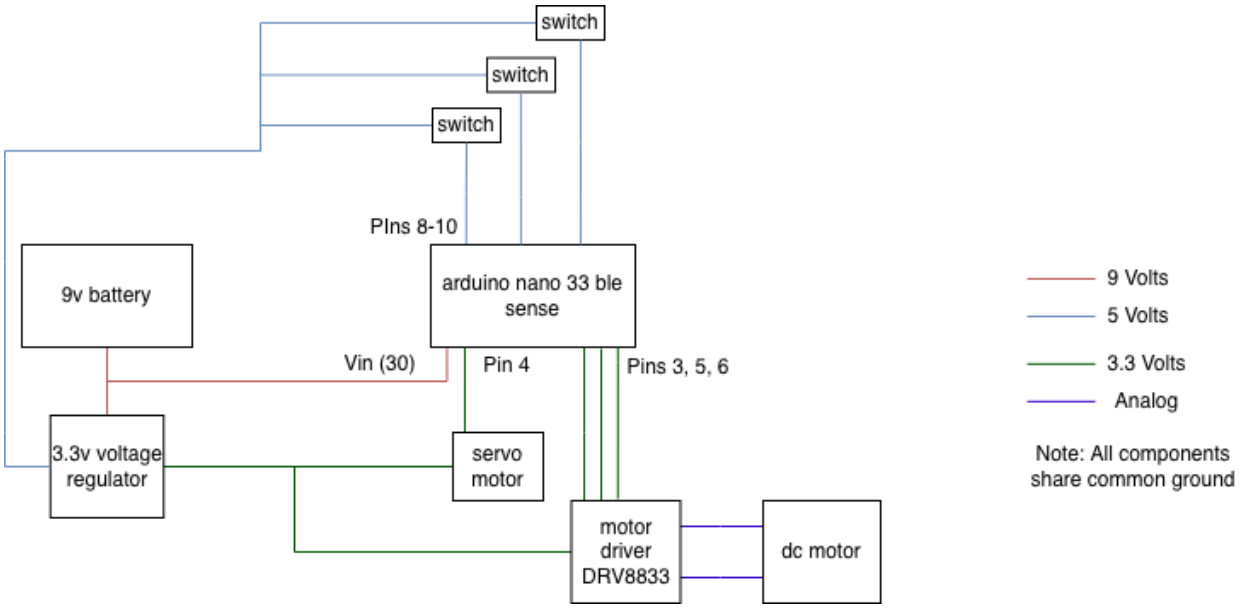

Fig.1: wiring diagram for the circuit

Appendix 2 - Alternate Versions

PCB Variations
Some variations of the circuits and PCB had different additions and features that may be good to implement in a final product before a class is made. These additions were a battery reader and a speaker. We did not implement these features to save on time of wiring the necessary resistors, speaker, and amp to the circuit. Not adding these items also helped us save space on our breadboard.

Battery Level Reader
Last semester we had plans to create functionality for reading the battery power level. This feature was put on the back burner while we worked on the essential functionality. It was later scrapped completely to save space and time while we finished the prototype.

Speaker
We originally had plans to add a speaker to give the user feedback about a correct or incorrect keyword. However, we found that the speaker was basically inaudible unless you were very close and there was no ambient noise. In order to make the speaker louder we would need to add an amplifier to the circuit. We decided to scrap the speaker idea as adding the amplifier would add more bulk to the circuit and end up taking up more power from the battery.

Appendix 3 - Other Deliverables

Data Collection Software: PyDataCollector

To help with the collection of voice data for this project we created a program that collects, formats, and packages audio recordings. This program was written in Python and was turned into an executable file that our team used when collecting data from ourselves and others.

The program asks for a name to package the data and what word will be recorded. This information, along with a unique timestamp creates a unique identifier so that no data is overwritten when it is uploaded to Edge Impulse. The program then prompts the user to select how many recordings will be made and which audio recording device they would like to use. After all that is selected the user goes through a series of recording events while the data is recorded, saved, and packaged. The original python code for this program is in Appendix 4.

Embedded Machine Learning Lesson Plan

Main objective:

This lesson plan intends to be an introductory project (or perhaps course) for students interested in machine learning. Students should go through the process of collecting data, creating a model, then using that model to perform a task on an embedded system.

Lab Activities:

Lab Activity 1: Record and understand voice data
Prerequisite: None
Students will be given a set of words to record a specific number of times. Students will use a given microphone and the Voice Recorder app on Windows to do the recordings. They will name their recordings and package them in a folder to be processed.

Lab Activity 2: Process and visualize voice data
Prerequisite: Lab Activity 1
Students will use Audacity to view their recordings and gain an understanding of .wav files. They will learn how to manipulate sampling rate and bit compression. They will get a visual understanding of the waveforms of different words and how they differ, as well as what differs between samples of the same word, and what patterns remain between samples of the same word. Students will cut the sound file to be the correct length (1 second for example) and package and submit these processed samples. The processed samples will go into a large repository for the whole class to be used later on with machine learning model training. This repository will grow as each section completes activities 1 and 2.

Lab Activity 3: Learn about Edge Impulse and upload data
Recommended after Lab Activity 2 (The coursera course introduces voice data)

Students will be asked to complete a Coursera Course on embedded machine learning with Edge Impulse: Introduction to Embedded Machine Learning | Coursera.
There are 3 sections:
Part 1 consists of 13 videos (107mins), 14 readings and 5 quizzes.
Part 2 consists of 10 videos (77mins), 10 readings, and 5 quizzes
Part 3 consists of 9 videos (75mins), 7 readings, and 4 quizzes.
Students will start the course during a lab session, and then be asked to complete the course by the next lab session. This course will show them the basics of using edge impulse, recording data, feature extraction, and model training/testing. It will also familiarize them with the Arduino Nano 33 BLE Sense which will be used for the entirety of the lab activities.

Lab Activity 4: Embedded programming basics
Prerequisite: None
Students will use the Arduino Nano 33 BLE Sense along with 2 LEDs. They will learn how to wire up the LEDs properly to the Arduino pins and to a common ground. They will then create a program that uses the serial monitor to send commands to the board to turn a certain light on or off. This activity will give them the basics of hardware and circuitry, and the Arduino IDE and serial monitor usage.

Lab Activity 5: Understand and implement libraries supplied by Edge Impulse
Prerequisite: Lab Activities 3 & 4
In this activity students will take their knowledge from activities 3 and 4 and create a program where their LEDs turn on and off when given the proper voice command. This will give the students a chance to use the supplied Edge Impulse libraries and examples to create a program of their own that performs the given task.

Lab Activity 6: Observe and mitigate skewed data
Prerequisite: Lab Activity 5
Students will be given data that is decidedly skewed. For example, a model that was trained on all the same voice, or a model that has many duplicates of the same exact data. Students can also be given data and parameters to over or under-fit a specific set of known data. Students can then experience what skewed data results in, when actually put to an embedded system.

Lab Activity 7: Demonstrate training a model on Edge Impulse
Prerequisite: Lab Activity 6
Students will upload data to Edge impulse, as was taught in lab activity 3. This time, with the data supplied by the current class, and classes past. With the data, train a model adjusting parameters to get the model confidence in an acceptable range.

Lab Activity 8: Demonstrate mastery by optimizing a model for best performance
Prerequisite: Lab Activity 7

Students will demonstrate their understanding in a final project using the data from previous classes. With the same dataset, students will train a model and tweak parameters or add synthetic data to demonstrate the highest confidence value. Students can then use their own model on the equipment in the lab. Then, students will go around to each different team and try out every other team's model. Students with the highest success rate could be rewarded to incentivise further effort into the project.

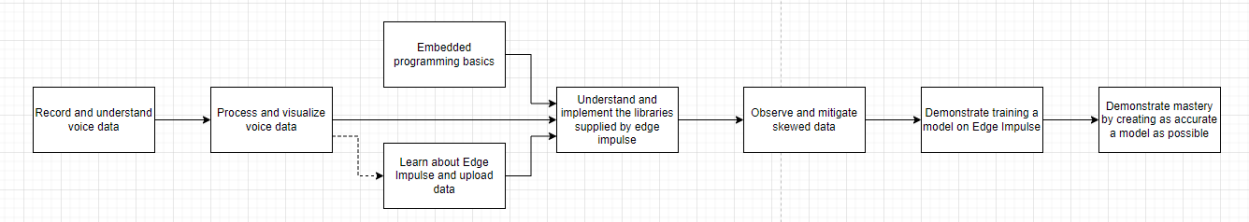

Fig.2: prerequisite flow-chart for the lesson plan

Printed Circuit Board

As the circuit was being created and tested, a Printed Circuit Board (PCB) was created alongside it. As the physical circuit was being tested and modified, the PCB was keeping up to date with the changes that were being made. The hardest part of the PCB design was getting to know a fairly new tool that is based in a web browser which made doing anything offline almost impossible. Design Rule Checking was also a major factor in the design of the PCB as these are standard rules that have to be followed either for the constraints of the machine making the PCB or the loads that the PCB will be under. Since this PCB is under fairly small loads, we can make it with the default specifications of the tool which also are the defaults of the machine that was making the PCB due to the partnership between JLCPCB, a PCB manufacturer, and EasyEDA, the creator of the tool. A physical version of the PCB was unable to be made due to the time and cost involved with shipping along with the demands and shortages caused by the recent pandemic.

Below is an image of the 2 layer PCB schematic that was created. The yellow lines are visible on the top of the board, the red lines are the connections in the first layer of the circuit and the blue are the connections within the second layer. The border of the board is represented by the purple line.

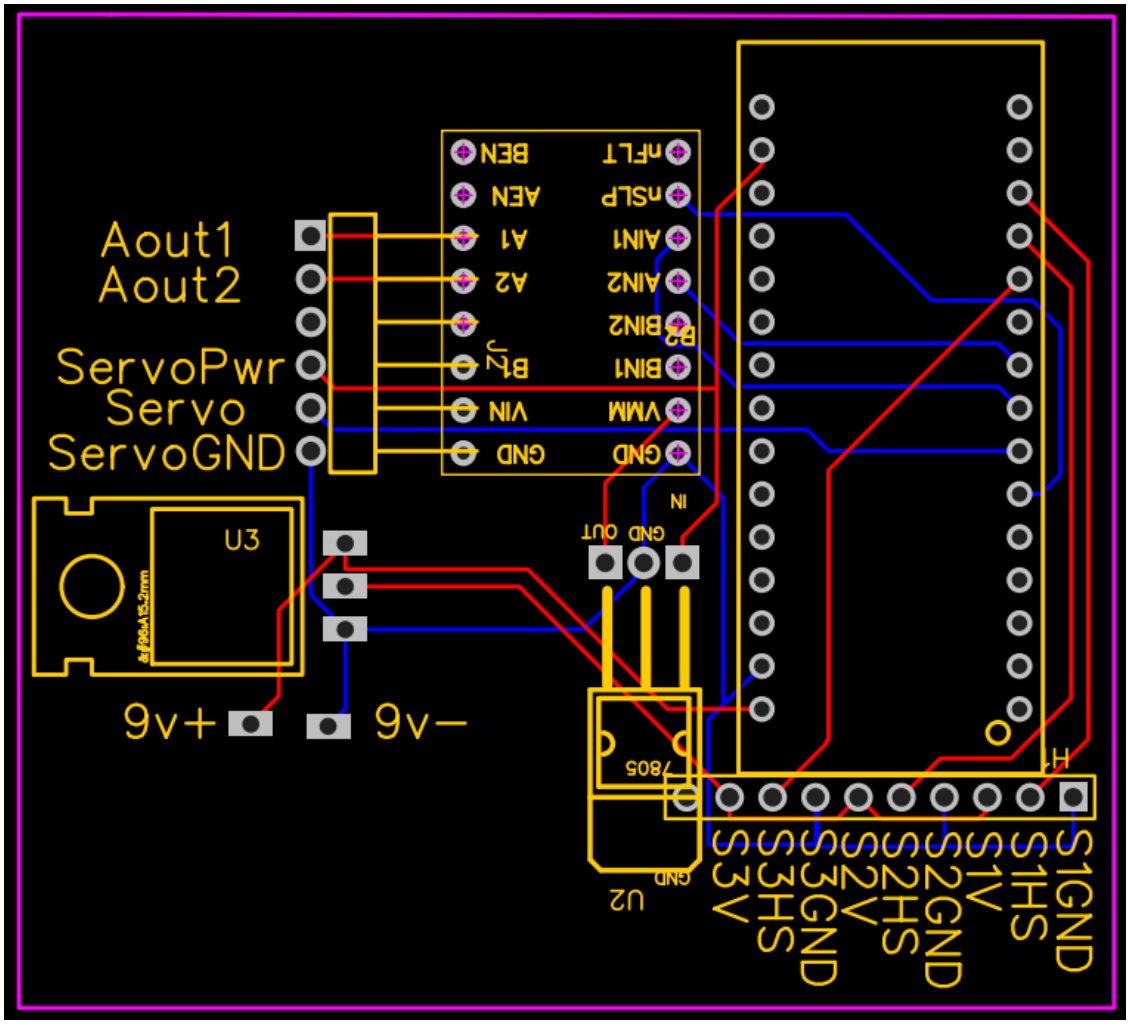

Fig.3 PCB diagram

Appendix 4 - Code

Smart Door Lock Code:

init.h

```
#ifndef _INIT_H
#define _INIT_H
/* Includes ------------------------------------------------------------------ */
#include <PDM.h>
#include <Servo.h>
#include <Lock_v1_inferencing.h>
#include "frame_controller.h"
#include "servo_motor.h"

// SERIAL
//
#define BAUD_RATE 115200

// FPS CONTROL
//
#define TIME_PER_FRAME 33
float elapsed_time = 0;
frame_controller fps_con(TIME_PER_FRAME);

// dc motor
//
#define UNLOCK_PIN 9   //D9
#define LOCK_PIN 8     //D8
#define DRIVER_A1_PIN 5 //D5
#define DRIVER_A2_PIN 6 //D6
#define SLP_PIN 3       //D3
bool locked = false;
int last_pos;
mbed::PwmOut driver1(digitalPinToPinName(DRIVER_A1_PIN));
mbed::PwmOut driver2(digitalPinToPinName(DRIVER_A2_PIN));

// servo motor
//
#define SERVO_PIN 4     //D4
#define ENGAGE_PIN 10   //D10
#define ENGAGE_ANGLE 20
#define DISENGAGE_ANGLE 150
#define STOP_ANGLE 90
Servo sm;
bool engaged = false;


// INFERENCE
```

```
//
#define CONFIDENCE_LEVEL 0.8

/* Edge Impulse Arduino examples
 * Copyright (c) 2021 EdgeImpulse Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */
// If your target is limited in memory remove this macro to save 10K RAM
#define EIDSP_QUANTIZE_FILTERBANK   0

/**
 * Define the number of slices per model window. E.g. a model window of 1000 ms
 * with slices per model window set to 4. Results in a slice size of 250 ms.
 * For more info: https://docs.edgeimpulse.com/docs/continuous-audio-sampling
 */
#define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 3



/** Audio buffers, pointers and selectors */
typedef struct {
    signed short *buffers[2];
    unsigned char buf_select;
    unsigned char buf_ready;
    unsigned int buf_count;
    unsigned int n_samples;
} inference_t;

static inference_t inference;
static bool record_ready = false;
```

```
static signed short *sampleBuffer;
static bool debug_nn = false; // Set this to true to see e.g. features generated
from the raw signal
static int print_results = -(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW);

#endif
```

Smart_Door_Lock.ino

```
//V1.0.0
#include "init.h"

void setup()
{
  // motor setup

  // put your setup code here, to run once:
  Serial.begin(BAUD_RATE);
  //servo motor setup
  pinMode(ENGAGE_PIN, INPUT_PULLUP);
  sm.attach(SERVO_PIN);
  engaged = false;

  //dc motor setup
  pinMode(UNLOCK_PIN, INPUT_PULLUP);
  pinMode(LOCK_PIN, INPUT_PULLUP);
  pinMode(DRIVER_A1_PIN, OUTPUT);
  pinMode(DRIVER_A2_PIN, OUTPUT);
  pinMode(SLP_PIN, OUTPUT);
  driver1.resume();
  driver2.resume();
  driver1.period(0.5f);
  driver2.period(0.5f);
  locked = false;

  Serial.println("Edge Impulse Inferencing Demo");

  // summary of inferencing settings (from model_metadata.h)
  ei_printf("Inferencing settings:\n");
  ei_printf("\tInterval: %.2f ms.\n", (float)EI_CLASSIFIER_INTERVAL_MS);
  ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
  ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
  ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories)
/

sizeof(ei_classifier_inferencing_categories[0]));

  run_classifier_init();
```

```
  if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false)
  {
      ei_printf("ERR: Failed to setup audio sampling\r\n");
      return;
  }
}

void loop()
{
    bool m = microphone_inference_record();
    if (!m)
    {
        ei_printf("ERR: Failed to record audio...\n");
        return;
    }

    signal_t signal;
    signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
    signal.get_data = &microphone_audio_signal_get_data;
    ei_impulse_result_t result = {0};

    EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
    if (r != EI_IMPULSE_OK)
    {
        ei_printf("ERR: Failed to run classifier (%d)\n", r);
        return;
    }

    if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW))
    {
        // print the predictions
        ei_printf("Predictions ");
        ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
            result.timing.dsp, result.timing.classification,
result.timing.anomaly);
        ei_printf(": \n");
        for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++)
        {
            ei_printf("    %s: %.5f\n",
result.classification[ix].label,result.classification[ix].value);
        }

        // INSERT SPEAKER/MOTOR CODE HERE
        // HOUSE = 0 MARVIN = 1
        if (result.classification[0].value >= CONFIDENCE_LEVEL ||
result.classification[1].value >= CONFIDENCE_LEVEL)
        {
```

```
          digitalWrite(LED_BUILTIN, HIGH);
          engage();
          Serial.println("ENGAGED");
          if (locked)
          {
            unlock();
          }
          else
          {
            lock();
          }
          Serial.println("LOCKED/UNLOCKED");
          disengage();
          Serial.println("DISENGAGED");
        }
        else
        {
          digitalWrite(LED_BUILTIN, LOW);
        }

        #if EI_CLASSIFIER_HAS_ANOMALY == 1
                ei_printf("    anomaly score: %.3f\n", result.anomaly);
        #endif

        print_results = 0;
    }
}

/**
 * @brief      Printf function uses vsnprintf and output using Arduino Serial
 *
 * @param[in]  format     Variable argument list
 */
void ei_printf(const char *format, ...)
{
    static char print_buf[1024] = { 0 };

    va_list args;
    va_start(args, format);
    int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
    va_end(args);

    if (r > 0) {
        Serial.write(print_buf);
    }
}
```

```
/**
 * @brief      PDM buffer full callback
 *             Get data and call audio thread callback
 */
static void pdm_data_ready_inference_callback(void)
{
    int bytesAvailable = PDM.available();

    // read into the sample buffer
    int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);

    if (record_ready == true)
    {
        for (int i = 0; i<bytesRead>> 1; i++) {
            inference.buffers[inference.buf_select][inference.buf_count++] =
sampleBuffer[i];

            if (inference.buf_count >= inference.n_samples)
            {
                inference.buf_select ^= 1;
                inference.buf_count = 0;
                inference.buf_ready = 1;
            }
        }
    }
}

/**
 * @brief      Init inferencing struct and setup/start PDM
 *
 * @param[in]  n_samples  The n samples
 *
 * @return     { description_of_the_return_value }
 */
static bool microphone_inference_start(uint32_t n_samples)
{
    inference.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed
short));

    if (inference.buffers[0] == NULL)
    {
        return false;
    }

    inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed
short));
```

```
    if (inference.buffers[1] == NULL)
    {
        free(inference.buffers[0]);
        return false;
    }

    sampleBuffer = (signed short *)malloc((n_samples >> 1) * sizeof(signed short));

    if (sampleBuffer == NULL)
    {
        free(inference.buffers[0]);
        free(inference.buffers[1]);
        return false;
    }

    inference.buf_select = 0;
    inference.buf_count = 0;
    inference.n_samples = n_samples;
    inference.buf_ready = 0;

    // configure the data receive callback
    PDM.onReceive(&pdm_data_ready_inference_callback);

    PDM.setBufferSize((n_samples >> 1) * sizeof(int16_t));

    // initialize PDM with:
    // - one channel (mono mode)
    // - a 16 kHz sample rate
    if (!PDM.begin(1, EI_CLASSIFIER_FREQUENCY)) {
        ei_printf("Failed to start PDM!");
    }

    // set the gain, defaults to 20
    PDM.setGain(127);

    record_ready = true;

    return true;
}

/**
 * @brief      Wait on new data
 *
 * @return     True when finished
 */
static bool microphone_inference_record(void)
{
```

```
    bool ret = true;

    if (inference.buf_ready == 1)
    {
        ei_printf(
            "Error sample buffer overrun. Decrease the number of slices per model
window "
            "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
        ret = false;
    }

    while (inference.buf_ready == 0)
    {
        delay(1);
    }

    inference.buf_ready = 0;

    return ret;
}

/**
 * Get raw audio signal data
 */
static int microphone_audio_signal_get_data(size_t offset, size_t length, float
*out_ptr)
{
    numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset],
out_ptr, length);

    return 0;
}

/**
 * @brief      Stop PDM and release buffers
 */
static void microphone_inference_end(void)
{
    PDM.end();
    free(inference.buffers[0]);
    free(inference.buffers[1]);
    free(sampleBuffer);
}

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR !=
EI_CLASSIFIER_SENSOR_MICROPHONE
#error "Invalid model for current sensor."
```

```
#endif

void dc_wake(){
  digitalWrite(SLP_PIN, HIGH);
}
void dc_sleep(){
  digitalWrite(SLP_PIN, LOW);
}
void lock(){
  dc_wake();
  driver1.write(0.25f);
  driver2.write(0);
  while (digitalRead(LOCK_PIN) == 0){}
  driver1.write(0);
  driver2.write(0);
  locked = true;
  dc_sleep();
}
void unlock(){
  dc_wake();
  driver1.write(0);
  driver2.write(0.25f);
  while (digitalRead(UNLOCK_PIN) == 0){}
  driver1.write(0);
  driver2.write(0);
  locked = false;
  dc_sleep();
}
bool is_locked(){
  return locked;
}
//---------------------------------
//SERVO FUNCTIONS
void engage(){
  sm.write(0);
  int i = 1;
  while(digitalRead(ENGAGE_PIN)==0){
    Serial.println(digitalRead(ENGAGE_PIN));
    delay(100);
    sm.write(0+i);
    i++;
  }
  delay(500);
  engaged = true;
}
void disengage(){
  delay(500);
```

```
  sm.write(0);
  delay(500);
  engaged = false;
}
```

PyDataCollector Code:
data_collector_main.py:

```
import io_handler

NAME = io_handler.getName()
WORD = io_handler.wordToRecord()
SESSION_ID = io_handler.getDateTime()
FILEPATH = "" + NAME + "_" + WORD + "_" + SESSION_ID
io_handler.makePathForData(FILEPATH)
NUMSAMPLES = io_handler.getNumSamples()
DEVICE_INDEX = io_handler.getDeviceIndex()

for i in range(1, NUMSAMPLES + 1):
    io_handler.prepSample(i, NUMSAMPLES, FILEPATH, WORD, DEVICE_INDEX)
io_handler.terminateAudio()
```

io_handler.py:

```
import os
import audio
from datetime import datetime

def getDateTime():
    currentTime = datetime.now()
    currentDateTime = str(currentTime) + str(currentTime.date())
    currentDateTime = currentDateTime.replace(".", "_").replace(" ",
"_").replace(":", "_")
    return currentDateTime

def wordToRecord():
    WORD = ""
    wordnum = input("1. House\n2. Marvin\nWhat keyword would you like to create
samples for? (Type 1 or 2): ")
    if wordnum == "1":
        WORD = "House"
    elif wordnum == "2":
        WORD = "Marvin"
    elif (int(wordnum) % 2) == 1:
        print("Input not 1 or 2, but input is odd... so defaulting to 1")
        WORD = "House"
```

```
    elif (int(wordnum) % 2) == 0:
        print("Input not 1 or 2, but input is even... so defaulting to 2")
        WORD = "Marvin"

    print("You will be making data for the word: '%s'" % WORD)
    return WORD

def makePathForData(PATH):
    if not os.path.isdir(PATH):
        os.mkdir(PATH)

def getNumSamples():
    NUMSAMPLES = input("How many samples would you like to record?\nSamples to
record: ")
    return int(NUMSAMPLES)

def getDeviceIndex():
    DEVICE_INDEX = audio.get_device_index()
    return DEVICE_INDEX

def prepSample(i, NUMSAMPLES, FILEPATH, WORD, DEVICE_INDEX):
    input("Ready to record sample %s of %s, press enter to begin recording the
1 second sample..." % (i, NUMSAMPLES))
    audio.record_audio(FILEPATH, FILEPATH+WORD+str(i), DEVICE_INDEX)

def getName():
    NAME = input("Enter name of data collector: ")
    return NAME

def terminateAudio():
    audio.terminateAudio()
```

audio.py:

```
import pyaudio
import wave

FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 16000
FRAMES_PER_BUFFER = 512
RECORD_SECONDS = 1
audio = pyaudio.PyAudio()

def get_device_index():
```

```
    print("----------------------record device list---------------------")
    info = audio.get_host_api_info_by_index(0)
    numdevices = info.get('deviceCount')
    for i in range(0, numdevices):
        if (audio.get_device_info_by_host_api_device_index(0,
i).get('maxInputChannels')) > 0:
            print("Input Device id ", i, " - ",
audio.get_device_info_by_host_api_device_index(0, i).get('name'))

    print("-------------------------------------------------------------")
    index = int(input())
    print("recording via index "+str(index))
    return index

def record_audio(filepath, filename, device_index):
    stream = audio.open(format=FORMAT, channels=CHANNELS,
                    rate=RATE, input=True,input_device_index = device_index,
                    frames_per_buffer=FRAMES_PER_BUFFER)
    Recordframes = []

    for i in range(0, int(RATE / FRAMES_PER_BUFFER * RECORD_SECONDS)):
        data = stream.read(FRAMES_PER_BUFFER)
        Recordframes.append(data)

    stream.stop_stream()
    stream.close()

    waveFile = wave.open(filepath + "/" + filename + ".wav", 'wb')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(FORMAT))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(Recordframes))
    waveFile.close()

def terminateAudio():
    audio.terminate()
```