Technical Documentation

WOODONIC is a device that detects abnormal sounds in the cutting process of a table saw and warns users with the notification light to stop the process.

Hardware Components

Raspberry Pi Zero x1
Used for processing input audio signals and running trained Machine Learning Model in order to predict anomalies.

16 GB MicroSD card x1
Used to install Raspbian OS on the Raspberry Pi Zero and store the solution code

USB mini microphone x1
Used for capturing audio input

MPU 6050 Accelerometer and Gyroscope sensor x1
Used to predict whether machine is working or not. Takes role in Power Saving feature.

Light Emitting Diode x1
User is notified by LED blink.

L7805 9V to 5V Voltage Regulator x1
Used to downgrade voltage from power source to be compatible with Raspberry Pi

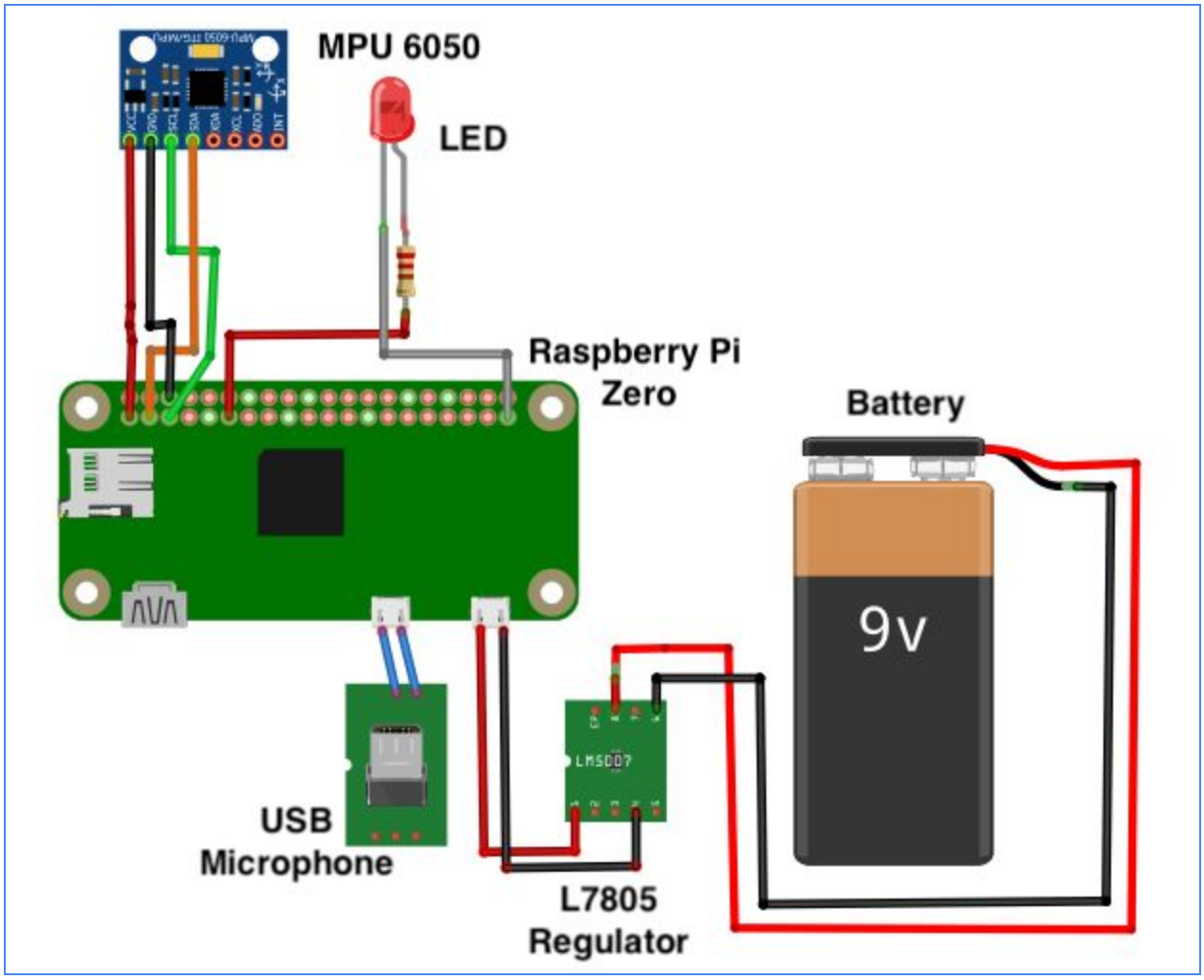
Software apps and Services

Jupyter Notebooks
Used while developing software solution in PC

Raspbian OS
Used to power up Raspberry PI Zero

Scikit Sklearn
Machine Learning Part is implemented using this Python Library

Connecting Hardware Parts

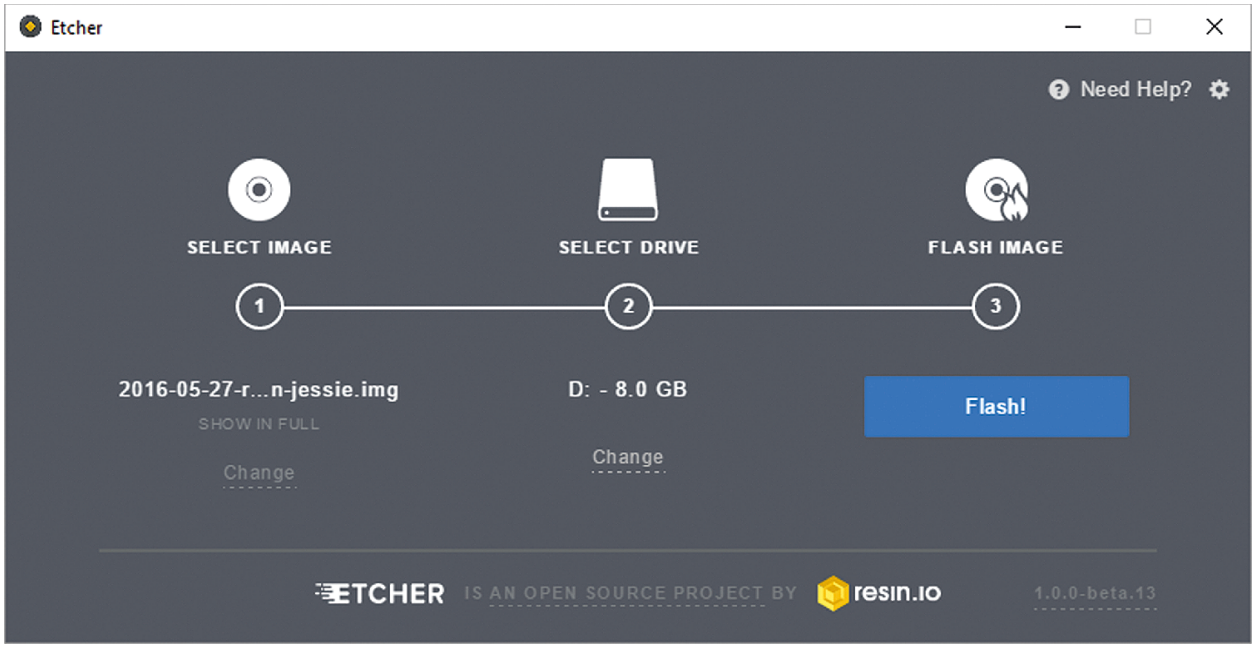


- Connect LED to Raspberry PI by connecting Anode leg into GPIO 17 and Cathode into GND pin of Raspberry PI
- Connect MPU6050 by connecting SDA, SCL, VCC and GND pins into corresponding pins of Raspberry PI. VCC of MPU6050 should be connected 5V output of Raspberry PI. Note that other pins on MPU6050 will be free and not connected
- Connect USB Microphone, via “USB to MicroUSB”, converter into Raspberry PI. Note that, despite there are 2 Micro USB ports available, only one of them can be used for data transfer. Other one is usable for power purposes.
- Insert SD Card into Raspberry PI

Preparing Embedded Device

In order to use Raspberry PI we need to install Raspbian OS. Of course, there exist other distributions as well but for our purposes Raspbian OS is pretty much suitable. Follow below instructions:

- Go to Raspberry PI website and navigate to download section
- Download full version of Raspbian OS
- Download USB Flash Imaging tool - Etcher
- Connect Micro SD card to PC
- Open Etcher, select previously downloaded image and Micro SD card and flash

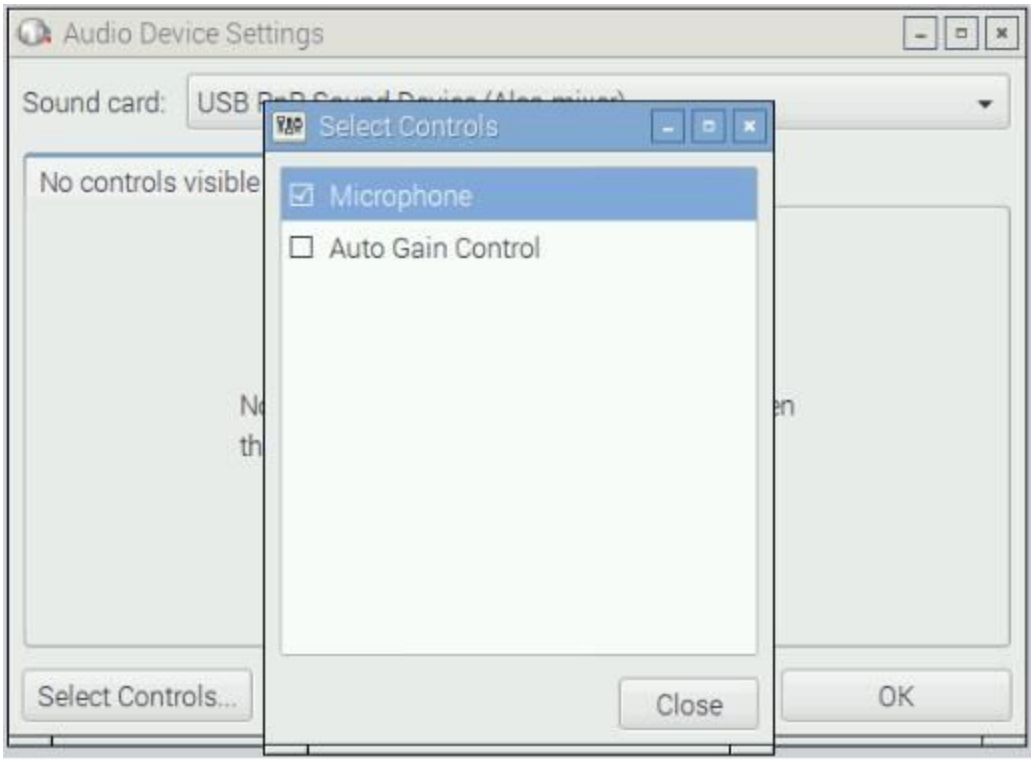


- Once flashing finishes, navigate to Micro SD card using File Explorer
- Specify your WiFi credentials in the file ‘etc/network/interfaces’

- Create an empty file and rename it to ‘ssh’. File should not have any extensions. Its required due to security reasons
- Power up Raspberry PI
- Connect to it using ‘ssh’. In Linux and Mac OS you can open terminal and type ‘ssh pi@ip_addr_of_rpi’
- When you’re asked about password, type ‘raspberry’. Its default password

At this step you should’ve installed Raspbian OS and successfully logged in. Next we need to configure Microphone.

- Connect to Raspberry PI using VNC
- Navigate to Preferences => Audio Device Settings
- Select ‘USB PnP Sound Device’ and click on ‘Select Controls’
- Make sure ‘Microphone’ checkbox is ticked
- Adjust the microphone level to MAX
- Save settings and reboot Raspberry PI



One last step in our embedded device configuration is the installation of required libraries. Open terminal in Raspberry PI and issue following commands one by one:

```
sudo apt-get update
sudo apt-get upgrade
dpkg -l > ~/Desktop/packages.list
pip freeze > ~/Desktop/pip-freeze-initial.list
```

```
sudo apt-get install xsel xclip libxml2-dev libxslt-dev python-lxml
sudo apt-get install python-h5py python-numexpr python-dateutil python-six
sudo apt-get install python-tz python-bs4 python-html5lib python-openpyxl
sudo apt-get install python-tables python-xlrd python-xlwt cython
sudo apt-get install python-sqlalchemy python-xlsxwriter
sudo apt-get install python-jinja2 python-boto python-gflags
sudo apt-get install python-googleapi python-httplib2 python-zmq
sudo apt-get install libspatialindex-dev
sudo pip install bottleneck rtree

sudo apt-get install python-numpy python-matplotlib
sudo apt-get install python-mpltoolkits.basemap python-pandas
sudo apt-get install python-scipy python-sklearn python-statsmodels

pip install librosa
pip install scikit
```

Preparing Dataset for Machine Learning

During the project, team has visited MakerSpace and collected Audio Data for Machine Learning model. In total there was 4 hours of recording in a Wave format. Wave format was chosen to capture all insights of the data and to prevent any loss due to compression algorithms. Additionally, noise cancellation features of recording devices were disabled.

These data was divided into 2 parts. Normal sound consists of 3 and half hours and was captured while target machine was working in idle, cutting, starting and stopping phases. Additionally background sound was also captured by generating different combinations of working machines. Other part of data consists of abnormal sound and was just used for testing of the Machine Learning model.

Later this data was re-played into Raspberry PI and recorded again by Raspberry PI in order to get audio record through microphone attached to it. Last captured data was used to train another Machine Learning model which was copied to Raspberry PI for predictions.

Preparation of Machine Learning Model

Device makes predictions based on Gaussian Mixture Model and Isotonic Regression.
Firstly, import necessary libraries and define constants:

```
import scipy
import scipy.fftpack
import numpy as np
from matplotlib import pyplot as plt
import scipy.io.wavfile as wavfile
from sklearn import svm
from sklearn import preprocessing
import multiprocessing
from scipy.signal import welch
from multiprocessing import Pool
from tqdm import tqdm_notebook
import time
from sklearn.preprocessing import StandardScaler
import pickle
import librosa
from sklearn.externals import joblib


SAMPLE_RATE = 32000 #sampling rate of audio
SEGMENT_TIME = 2 # in seconds, duration of each split
```

Next we define function to extract features from audio signal. This function heavily relies on Librosa Audio Processing library.

```
# Function to extract features from input signal(in our case Audio).
# Various features are extracted using Librosa and concatenated as a 1D Array
def extract_feature(X):
    sample_rate = SAMPLE_RATE

    stft = np.abs(librosa.stft(X))
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
    chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
    mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
    contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,axis=0)
    tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X), sr=sample_rate).T,axis=0)
    return np.hstack([mfccs,chroma,mel,contrast,tonnetz])
```

We also need some function to read files and convert them into Matrices for Machine Learning part.

```
# Function to read file, split into segments and retrieve features for each segment
def read_process_file(filename):
    print("Processing file: ", filename)

    # Read file into memory
    signal, fs_rate = librosa.core.load(filename, SAMPLE_RATE)
    N = len(signal)
    duration = int(SEGMENT_TIME * fs_rate)

    # Split input signal into segments
    signal = signal[: N - (N % duration)]
    samples = np.reshape(signal, (-1, duration))

    # Process each segment to get features
```

```
    with Pool(4) as p: # Use 4 threads to accelerate processing
         features = list(tqdm_notebook(p.imap(extract_feature, samples), total=len(samples)))

    return np.array(features).astype('float')

# Build single dataset with size of [N, F],
# where N is number of segments and F is number of features for each segment
def get_files(dirname, files):
    X = None
    for file in files:
        samples_features = read_process_file(dirname + file)
        if X is None:
            X = samples_features
        else:
            X = np.vstack((X, samples_features))
    return X
```

Now we can read recorded normal and abnormal audio files and convert them into Training, Test and Validation datasets.

```
# Specify audio files with normal sound recording
# Used for training, testing and validation
normal_file_dir = 'records_from_pi/'
normal_files = [
    'normal_1.wav',
    'normal_2.wav',
    'normal_3.wav',
    'normal_4.wav',
    'normal_5.wav',
    'silence_1.wav',
    'silence_2.wav',
    'silence_long.wav',
    'silence_short.wav'
]

# Append label to each segment
X_normal = get_files(normal_file_dir, normal_files)
normal_labels = np.ones((X_normal.shape[0], 1))
X_normal = np.hstack((X_normal, normal_labels))

# Specify audio files with abnormal sound recording
# Used for testing and validation
abnormal_file_dir = 'records_from_pi/'
abnormal_files = [
    'abnormal_1.wav',
    'abnormal_2.wav',
]

# Append label to each segment
X_abnormal = get_files(abnormal_file_dir, abnormal_files)
abnormal_labels = np.zeros((X_abnormal.shape[0], 1))
X_abnormal = np.hstack((X_abnormal, abnormal_labels))

# Show some output about data
```

```
print("Normal data shape: ", X_normal.shape)
print("Abnormal data shape: ", X_abnormal.shape)

# Split normal data, get training dataset
train_mask = np.random.choice([False, True], len(X_normal), p=[0.40, 0.60])
X_train = X_normal[train_mask]
np.random.shuffle(X_train)
X_other = X_normal[~train_mask]

# Concatenate remaining normal data (on that's not used for training) and abnormal data
X_other_abnormal = np.vstack((X_abnormal, X_other))

# Split concatenated data for validation and test, also shuffle data
other_abnormal_mask = np.random.choice([False, True], len(X_other_abnormal), p=[0.50, 0.50])
X_val = X_other_abnormal[other_abnormal_mask]
np.random.shuffle(X_val)
X_test = X_other_abnormal[~other_abnormal_mask]
np.random.shuffle(X_test)

# Separate labels from segment features
label_index = 193
Y_train = X_train[:, label_index]
X_train = X_train[:, 0:label_index]

Y_val = X_val[:, label_index]
X_val = X_val[:, 0:label_index]

Y_test = X_test[:, label_index]
X_test = X_test[:, 0:label_index]

# Show some output about data
print("Train data shape: ", X_train.shape)
print("Train labels shape: ", Y_train.shape)
print("Validation data shape: ", X_val.shape)
print("Validation labels shape: ", Y_val.shape)
print("Test data shape: ", X_test.shape)
print("Test labels shape: ", Y_test.shape)
```

At this point data is ready for Machine Learning model. We can build the model and fit to training dataset. Don’t forget to apply Standard Scaler to the input on the runtime also. It ensures normalization and contributes to numerical stability.

```
# Apply standard scaler to data
ss = StandardScaler()
ss.fit(X_train)
X_train = ss.transform(X_train)
X_val = ss.transform(X_val)
X_test = ss.transform(X_test)

# Also assuming that resnet feature generation is done
from sklearn.mixture import GaussianMixture
from sklearn.isotonic import IsotonicRegression

# Build Gaussian Mixture Model and fit to training set
gmm_clf = GaussianMixture(covariance_type='spherical', n_components=4, max_iter=int(1e7))  # Obtained
```

```
via grid search
gmm_clf.fit(X_train)
log_probs_val = gmm_clf.score_samples(X_val)
# Also build Isotonic Regression for predictions
isotonic_regressor = IsotonicRegression(out_of_bounds='clip')
isotonic_regressor.fit(log_probs_val, Y_val)  # y_val is for labels 0 - not food 1 - food (validation
set)

# Obtaining results on the validation set
log_probs_val = gmm_clf.score_samples(X_val)
val_probabilities = isotonic_regressor.predict(log_probs_val)
val_predictions = [1 if prob >= 0.5 else 0 for prob in val_probabilities]

# Calculate accuracy metrics
val_correct_pred = np.equal(Y_val, val_predictions)
val_acc = np.sum(val_correct_pred) / val_correct_pred.shape[0]
print("Validation accuracy: ", val_acc)

# Obtaining results on the test set
log_probs_test = gmm_clf.score_samples(X_test)
test_probabilities = isotonic_regressor.predict(log_probs_test)
test_predictions = [1 if prob >= 0.5 else 0 for prob in test_probabilities]

# Calculate accuracy metrics
test_correct_pred = np.equal(Y_test, test_predictions)
test_acc = np.sum(test_correct_pred) / test_correct_pred.shape[0]
print("Test accuracy: ", test_acc)
```

Now we have our model and performance metrics. As a last step Scaler, Model and Regression should be saved and copied into Raspberry PI.

```
# Save model
save_dirname = "saved_models/"
joblib.dump(ss, save_dirname + "ss.pkl", compress=9)
joblib.dump(gmm_clf, save_dirname + "gmm_clf.pkl", compress=9)
joblib.dump(isotonic_regressor, save_dirname + "isotonic_regressor.pkl", compress=9)

print("Your models are saved, please copy them into Raspberry PI")
```

Embedded Device side code

The code for Raspberry PI is somehow similar to the code above with some additions like hardware output control. This is due to cross platform compatibility of Python programming language and support for Python and other libraries in the Raspberry PI side. As usual the code starts by importing necessary libraries, defining constants and setting up IO pin for output mode.

```
import sklearn
import pickle
import subprocess
import os
import librosa
import numpy as np
```

```
from sklearn.externals import joblib
import RPi.GPIO as GPIO

# Define Constants
SAMPLE_RATE = 32000
SEGMENT_TIME = 2
RECORD_TIME = 2

# Setup LED Output pin
ledpin = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(ledpin, GPIO.OUT, initial=GPIO.LOW)
```

Next some helper functions are defined. This functions are very similar the ones above as we follow the same feature extraction process for compatibility between inputs.

```
# Helper functions
def extract_feature(X):
    sample_rate = SAMPLE_RATE
    stft = np.abs(librosa.stft(X))
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T,axis=0)
    chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T,axis=0)
    mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T,axis=0)
    contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T,axis=0)
    tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X), sr=sample_rate).T,axis=0)
    return np.hstack([mfccs,chroma,mel,contrast,tonnetz])

# Read and process recorded file
def read_process_file():
    signal, fs_rate = librosa.core.load("temp.wav", SAMPLE_RATE)
    N = len(signal)

    duration = int(SEGMENT_TIME * fs_rate)

    signal = signal[: N - (N % duration)]
    samples = np.reshape(signal, (-1, duration))
    features = extract_feature(samples[0, :])

    return np.array(features).astype('float')
```

One last thing before the main processing starts, we need to load Input Scaler, Gaussian Mixture Model and Isotonic Regression. These files were prepared using more powerful machine, like PC, and copied to Raspberry PI.

```
# Load Standard Scaler, Gaussian Mixture Model and Isotonic Regression
ss = joblib.load('saved_models/ss.pkl')
gmm_clf = joblib.load('saved_models/gmm_clf.pkl')
isotonic_regressor = joblib.load('saved_models/isotonic_regressor.pkl')
print("Models loaded successfully")
```

Now we are ready for the main logic. Everytime we call another process in the Linux OS to record audio for us. This audio is processed in order to get features which are feed into Machine Learning model. Finally, the model will output a prediction whether the audio signal is normal or

abnormal. Based on this prediction, user will be notified by LED light.

```
while True: # loop forever
    # Record audio to 'temp.wav'
    subprocess.run(['arecord', '-D', 'hw:1', '-f', 'S16_LE', '-c1', '-r44100', '-d', str(RECORD_TIME),
'temp.wav'])

    # Extract features from recorded audio
    features = read_process_file()

    # Scale features in order to normalize and prevent numerical stability issues
    scaled_features = ss.transform(features.reshape(1, -1))


    log_probs = gmm_clf.score_samples(scaled_features)
    probabilities = isotonic_regressor.predict(log_probs)
    predictions = [1 if prob >= 0.5 else 0 for prob in probabilities]

    if predictions[0] == 0: # anomaly detected, turn on LED
        GPIO.output(ledpin, GPIO.HIGH)
    else: # normal workflow, turn off LED
        GPIO.output(ledpin, GPIO.LOW)

GPIO.output(ledpin, GPIO.LOW)
```

Images of final prototype

Source code, Audio Recording Dataset and 3D Design files

Source code, audio recordings of machines that were used for training above model and 3D design of Woodonic can be accessed via following GitHub link:

https://github.com/faridyagubbayli/tech_challenge_ws1819