Shaurya Dhankar - 204420039
Rahul Malavalli - 204429252
EE 3
Final Report

Knock Unlock Final Report

Introduction and Background:

The goal of this project was to design and fabricate a device that could unlock a door after receiving a specific secret knock sequence. To establish portability, a mountable design was selected and an Arduino was used to control the entire system. In the best case scenario, the motor used to handle the door lever would be light as well, requiring minimal power and thus providing maximum portability; unfortunately, the motor chosen was too large and powerful, requiring an external power supply and damaging the design's portability.

An Arduino 101 was utilized with an on-board accelerometer to detect door knocks, especially with the help of a library (CurieIMU) provided by the board. To ensure accurate knock detections, the proper debouncing had to be calibrated by testing different debouncing times (minimum time interval between consecutive knocks to reduce effects of extra vibrations, etc.); to ensure accurate knock comparisons with the saved secret knock sequence, times between knocks were read and mapped to a set range for easy comparison (see code for more detail). An H-bridge chip was connected to the system to allow both clockwise and counterclockwise rotation of the motor so that the door handle could be both opened and closed. Other small components, such as LEDs for success and failure notification and a button to indicate saving a new knock sequence, were added to the design as well, all of which were implemented on a breadboard; in the future, it can be beneficial to solder all of the components onto a perfboard for compactness and ease of mounting on the physical frame used.

At the conclusion of the final project, multiple designs were cycled and a version of the above implementation, with a PVC pipe based door frame, was utilized successfully.

Testing Methodology:

Multiple designs were attempted and discarded in the design iteration process, and the final product utilized a relatively simpler and straightforward circuit that successfully met (most of) the initial project goals. Throughout the process, two main avenues of issues existed; software (Arduino code), and hardware (circuit, electrical components, etc.), of which the hardware issues nearly consistently posed more confounding problems.

The basic software design was completed earlier in the process, further developing gradually as general optimizations, code restructures, and debouncing times were accrued. Specifically:

- Minimum thresholds for accelerometer shock/knock detection were experimentally determined by placing the Arduino 101 on a flat hard surface (commonly a desk) and knocking on the surface from different distances and with varying intensities.
- Debouncing time (minimum times between knocks) was determined, also experimentally, by tweaking the time (higher to block more extraneous "noise", and lower to increase sensitivity to higher frequency knocks) for each surface used such that only all the knocks delivered are consistently reflected in the Arduino's knock input. In the future, it would be recommended to plot accelerometer values during knocks to determine, exactly, how much debouncing time is needed.

Hardware provided its own set of problems, generally following the trend of progress and debugging as detailed below:

- A single NPN transistor was first utilized to rotate the motor one way. After successful testing (switching the transistor on and off with the Arduino), a two transistor design was attempted to allow rotation of the motor in both ways.
- A PNP transistor along was added to the single NPN transistor setup to rotate the motor in both directions. This originally failed with a single voltage source since it was discovered (with a digital multimeter, or DMM) that a relatively negative voltage was not being applied to the base of the PNP, and thus opposite motion was not possible. Furthermore, the negative voltage source was required to connect to the PNP circuit that would turn the motor in the opposite direction. Thus, two voltage sources were connected in series and the positive of one was connected to the negative of another to simulate ground and the respective positive and negative voltages.
- To account for the lack of a negative output from the Arduino to trigger the PNP transistor, an operational amplifier (opamp) was utilized to invert the positive voltage signal output from the Arduino to direct to the PNP base. On some cases, this approach would work; however, it became evident that a short-circuit was occurring (the PNP was not turned completely off appropriately) when the transistors started smoking and the DMM was used to determine non-zero voltages being applied by the PNP at incorrect times. This failed circuit is shown in Figure 1 below.
- At this point, the H-bridge chip was ordered and successfully used to control motor direction. This circuit was simply tested with a DMM checking voltage across the H-bridge outputs in response to the input voltages applied by the Arduino denoting motor direction. The schematic for this circuit is included below in Figure 2.

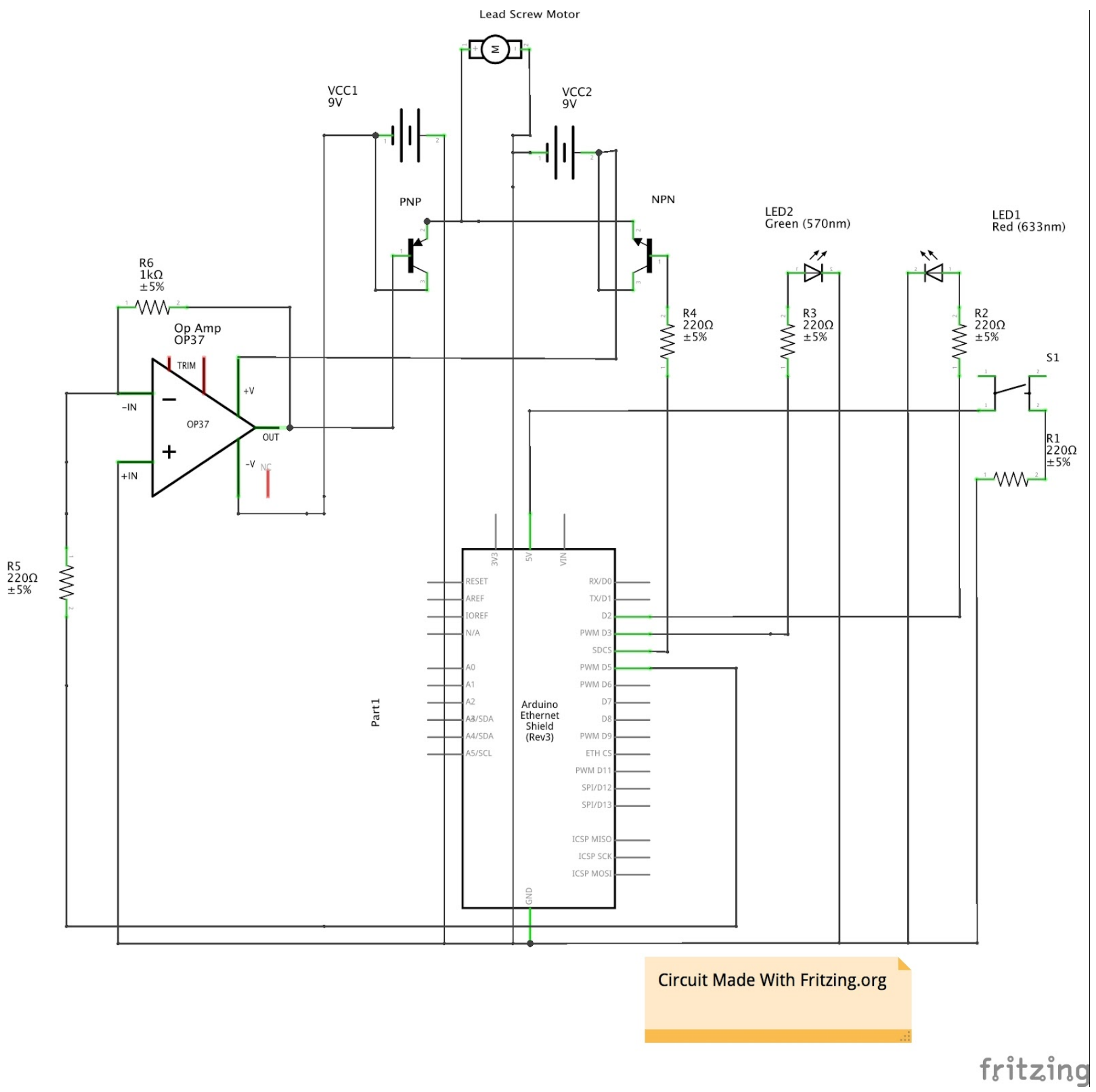

Figure 1: Failed Circuit

- Finally, an H-bridge was used in order to rotate the motor both ways. One signal input (1A) of the H-bridge was connected with the digital output of the Arduino and another signal input (2A) of the H-bridge with another digital output of the Arduino.
- The two signal outputs of the H-bridge (1Y and 2Y) were connected to the motor. 1Y rotated the motor in clockwise direction and 2Y rotated the motor in anticlockwise direction to achieve the goal of unlocking and locking the door. This way final circuit was set-up as shown in Figure 2.

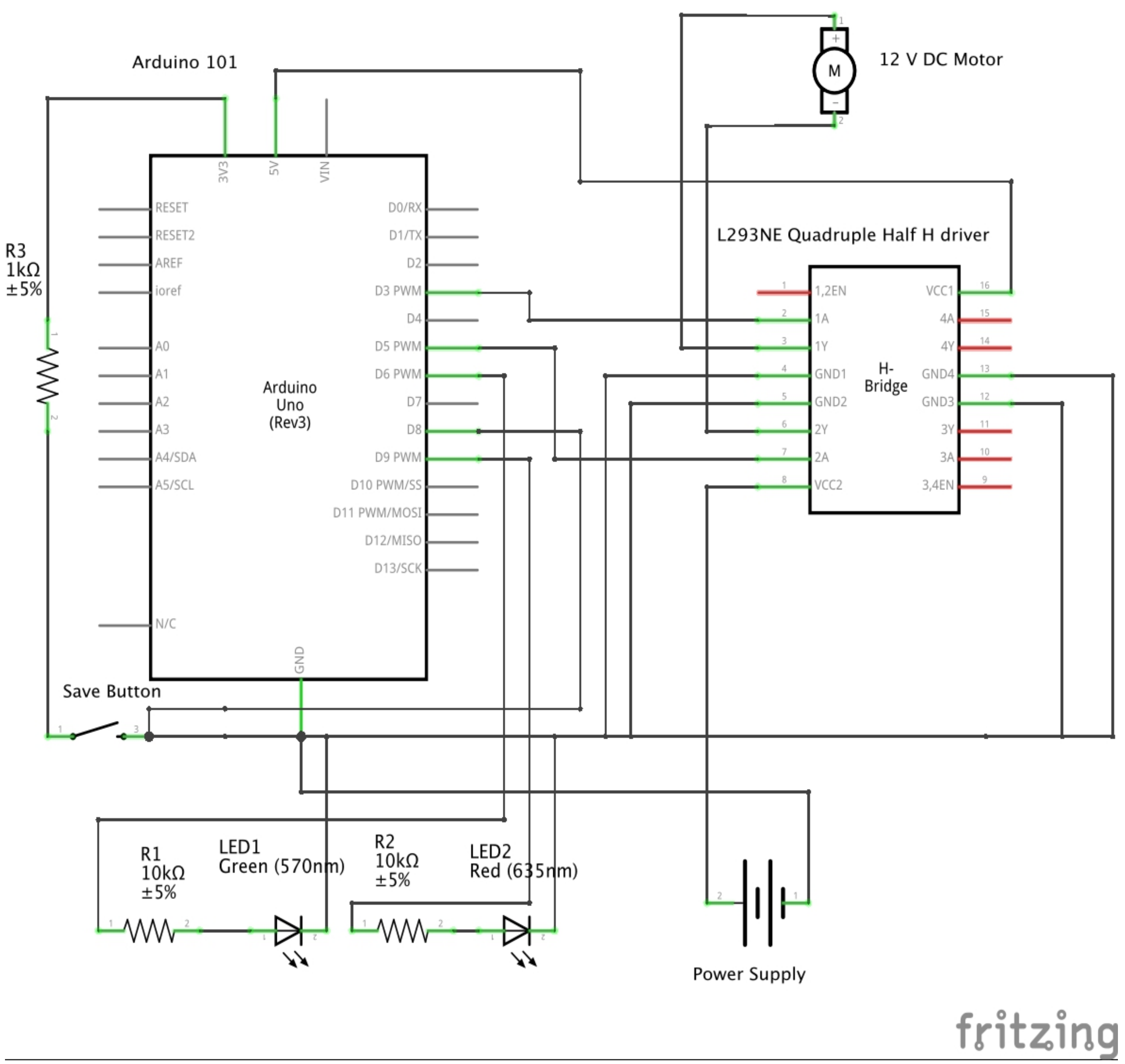

Figure 2: Final Circuit (made with fritzing.org)

- Next, comparison was made between the saved knock and actual knock to convert the shock detected by the arduino to the knock required by the user to open the door
- Once the knock was compared, the arduino was directed to send a signal to the motor to rotate and unlock the door when the knock code was correctly recognised.
- This way it was able to determine the real owner of the house and was able to open the door successfully.

Results and Discussion:

The required current necessary to rotate our high powered motor was generated successfully. The external power supply came to good use to provide +V to power the motor. The motor started to rotate at 6V and drew about 0.7A and the H-bridge was successfully able to rotate the motor in both directions for the set time of 15 seconds.

Then the arduino was mounted, circuit board and the motor on the PVC pipes with the save button on top and our final mount looked the one shown in Figure 3 and Figure 4.

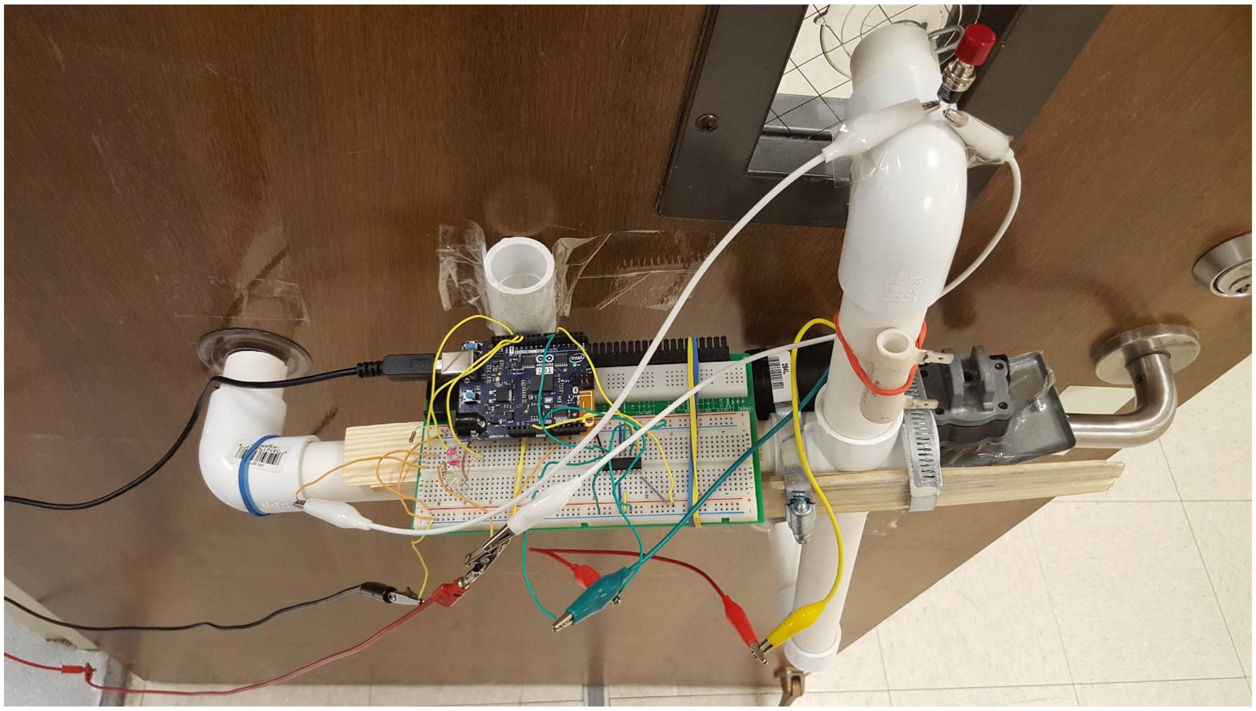

Figure 3: Final mount of the knock unlocker.

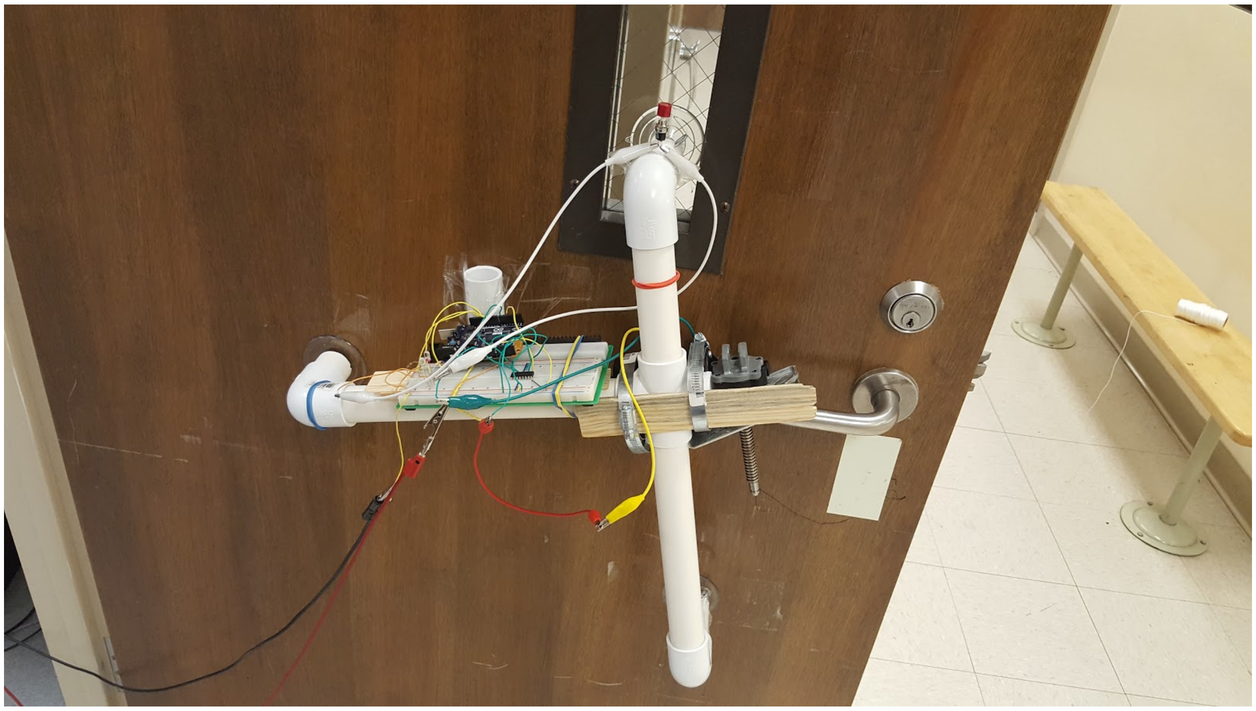

Figure 4: Final mount of the knock unlocker.

The clamp attached to the lead screw of the motor was able to push the door handle down and unlock the door with ease.

Conclusions and Future Work:

The design worked successfully as it was able to both unlock and lock the door with ease. It however did need an external power supply since our motor required high current. It could have used a lower powered motor and attached it to an external battery that could have been mounted with the motor on our setup. That way, the design would have been more easily portable.

If we had more time, we would have liked to add an additional layer of security to our model by verifying the knocker identity to the arduino with the knocker's phone by using the bluetooth sensor of the arduino and smartphone. We would have connected our phones bluetooth to the arduino so that the Arduino would have been able to detect the presence of the real owner's phone since a knock code security can be compromised.

Illustration Credits:

1. Fritzing.org

References:

1. Circuit made with fritzing.org
2. http://www.instructables.com/id/Secret-Knock-Detecting-Door-Lock/

Code:

```
#include "CurieIMU.h"

// Knock reading properties
const unsigned long KNOCK_LISTEN_TIMEOUT = 3000;   // 50000 ms = 5 seconds
const unsigned long minKnockInterval = 180;                // 100 ms
const unsigned int READ_KNOCK_THRESHOLD = 1000;
//const unsigned int READ_KNOCK_THRESHOLD = 900;
const unsigned int READ_KNOCK_DURATION = 1100;

// Saved Knocks
const unsigned long MAX_KNOCKS = 30;
volatile unsigned long savedKnocksIntervals[MAX_KNOCKS];
volatile unsigned int savedKnocksLen = 0;

// Reading Knocks
volatile unsigned long currKnockIntervals[MAX_KNOCKS];
volatile unsigned int currKnockIndex = 0;                // Used to denote
current position in currKnockIntervals and the size of the array after
population

// Mapping constants
const unsigned long RANGE_LOW = 0;
const unsigned long RANGE_HIGH = 100;
//const unsigned long MAX_PERCENT_TOLERANCE = 35;        // Percent difference
allowed for valid knock checking. TODO: may need tweaking
const unsigned long MAX_PERCENT_TOLERANCE = 45;        // Percent difference
allowed for valid knock checking. TODO: may need tweaking
const unsigned long AVE_PERCENT_TOlERANCE = 25;        // Average percent
difference allowed for valid knock checking. TODO: may need tweaking

// Pins
const unsigned int PIN_ONBOARD_LED = 13;
const unsigned int PIN_BUTTON = 13;
const unsigned int PIN_POS = 6;
const unsigned int PIN_NEG = 5;
const unsigned int PIN_LED_GREEN = 11;
const unsigned int PIN_LED_RED = 12;

// Motor
//const unsigned long MOTOR_RUN_TIME = 3000;
const unsigned long MOTOR_RUN_TIME = 14000;
const unsigned long MOTOR_OFF_TIME = 2000;

// Debugging
const boolean DEBUG = true;
boolean DEBUG_SAVE_KNOCK = true;
```

```
void printKnocks (volatile unsigned long knocks[], volatile unsigned int len)
{
  if (!DEBUG) return;
  // Testing: printing array of times
  Serial.print("[");
  unsigned int i;
  for (i = 0; i < len; i++) {
    Serial.print(knocks[i]);
    if (i < len - 1) Serial.print(", ");
  }
  Serial.println("]");
}

// End Debugging

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while (!Serial);

  CurieIMU.begin();
  CurieIMU.attachInterrupt(readKnock);
  //  CurieIMU.setDetectionThreshold(CURIE_IMU_SHOCK, 990);  // 10 mg
  //  CurieIMU.setDetectionDuration(CURIE_IMU_SHOCK, 900);  // 200 ms
  CurieIMU.setDetectionThreshold(CURIE_IMU_SHOCK, READ_KNOCK_THRESHOLD);  //
10 mg
  CurieIMU.setDetectionDuration(CURIE_IMU_SHOCK, READ_KNOCK_DURATION);  // 200
ms
  CurieIMU.interrupts(CURIE_IMU_SHOCK);

  // Set up pins
  pinMode(PIN_ONBOARD_LED, OUTPUT);
  pinMode(PIN_POS, OUTPUT);
  pinMode(PIN_NEG, OUTPUT);
  pinMode(PIN_LED_GREEN, OUTPUT);
  pinMode(PIN_LED_RED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);

  Serial.println("Started");
}

void preProcessKnocks() {
  /** Preprocess knocks by mapping interval values to an arbitrary range
(RANGE_LOW to RANGE_HIGH) **/

  unsigned int i;
  unsigned long currInter;
  //  Find the lowest and highest interval values
  unsigned long low = KNOCK_LISTEN_TIMEOUT;       // This is the maximum
possible interval value
  unsigned long high = 0;                         // This is the lowest
possible interval value

  for (i = 0; i < currKnockIndex; ++i) {
    currInter = currKnockIntervals[i];
    if (currInter < low) low = currInter;
    if (currInter > high) high = currInter;
  }

  // Map values from RANGE_LOW to RANGE_HIGH
  for (i = 0; i < currKnockIndex; ++i) {
    currKnockIntervals[i] = map(currKnockIntervals[i], low, high, RANGE_LOW,
RANGE_HIGH);
  }
}

void saveKnocks() {
  Serial.println("Saving knocks");
  savedKnocksLen = currKnockIndex;
  unsigned int i = 0;
  for (i = 0; i < savedKnocksLen; ++i) {
    savedKnocksIntervals[i] = currKnockIntervals[i];
  }
  //  memcpy(savedKnocksIntervals, currKnockIntervals, currKnockIndex);
}

boolean checkKnockPattern() {

  // First check if the number of knocks are the same
  if (currKnockIndex != savedKnocksLen) {
    Serial.println("Failing because lengths don't match");
    return false;
  }

  Serial.println("SAVED: Knocks saved");
  printKnocks(savedKnocksIntervals, savedKnocksLen);
  Serial.println("Knocks Read");
  printKnocks(currKnockIntervals, currKnockIndex);
  Serial.println("");

  unsigned int i;
  unsigned long percDiff;
  unsigned long totalDiff = 0;
  for (i = 0; i < currKnockIndex; ++i) {
    percDiff = (unsigned long) abs((long)savedKnocksIntervals[i] -
(long)currKnockIntervals[i]);
    if (percDiff > MAX_PERCENT_TOLERANCE) {
      Serial.print("FAILED: ");
      Serial.print(percDiff);
      Serial.print(" out of tolerance at index ");
      Serial.print(i);
      Serial.print(" with saved ");
      Serial.print(savedKnocksIntervals[i]);
      Serial.print(" and read ");
      Serial.println(currKnockIntervals[i]);
      Serial.println();
      return false;
    }
    totalDiff += percDiff;
  }

  // Check if the total average time differences are acceptable
  if ((totalDiff / currKnockIndex) > AVE_PERCENT_TOlERANCE) {
    Serial.print("FAILED: ");
    Serial.print(totalDiff);
    Serial.println(" out of AVERAGE tolerance.");
    Serial.println();
    return false;
  }

  // Otherwise, we're good!
  return true;
}

void handleSuccess() {
  Serial.println("Success");
  digitalWrite(PIN_ONBOARD_LED, HIGH);
  digitalWrite(PIN_LED_GREEN, HIGH);
  turnFront();
  delay(MOTOR_RUN_TIME);
  turnOff();
  delay(MOTOR_OFF_TIME);
  turnBack();
  delay(MOTOR_RUN_TIME);
  turnOff();
  digitalWrite(PIN_LED_GREEN, LOW);
  digitalWrite(PIN_ONBOARD_LED, LOW);
}

void handleFailure() {
  Serial.println("Failure");
  digitalWrite(PIN_ONBOARD_LED, HIGH);
  digitalWrite(PIN_LED_RED, HIGH);
  delay(MOTOR_OFF_TIME);
  digitalWrite(PIN_LED_RED, LOW);
  digitalWrite(PIN_ONBOARD_LED, LOW);
}

void handleSave() {
  Serial.println("Saved");
  unsigned int i;
  digitalWrite(PIN_LED_GREEN, LOW);
  for (i = 0; i < 4; ++i) {
    digitalWrite(PIN_LED_GREEN, HIGH);
    digitalWrite(PIN_ONBOARD_LED, HIGH);
    delay(800);
    digitalWrite(PIN_ONBOARD_LED, LOW);
    digitalWrite(PIN_LED_GREEN, LOW);
  }
  digitalWrite(PIN_LED_GREEN, LOW);
}

void analyzeKnock() {
  noInterrupts();
  Serial.println("Analyzing knock");

  Serial.println("RAW: Knocks Read");
  printKnocks(currKnockIntervals, currKnockIndex);
  Serial.println("");

  // Pre process knocks to apply appropriate interval
  preProcessKnocks();

  Serial.println("PROCESSED: Knocks Read");
  printKnocks(currKnockIntervals, currKnockIndex);
  Serial.println("");

  // Check if we were saving a new knock
  if (digitalRead(PIN_BUTTON) == HIGH) {
    saveKnocks();
    handleSave();
  } else {
    if (checkKnockPattern() == true) {
      handleSuccess();
    }
    else {
      handleFailure();
    }
  }
  interrupts();
}

boolean knockDetected() {
  return (CurieIMU.shockDetected(X_AXIS, POSITIVE) ||
          CurieIMU.shockDetected(X_AXIS, NEGATIVE) ||
          CurieIMU.shockDetected(Y_AXIS, POSITIVE) ||
          CurieIMU.shockDetected(Y_AXIS, NEGATIVE) ||
          CurieIMU.shockDetected(Z_AXIS, POSITIVE) ||
          CurieIMU.shockDetected(Z_AXIS, NEGATIVE));
}

void loop() {
  // Only continue if a shock was received
  if (!knockDetected())
    return;

  // Set up timing characteristics
  unsigned long timenow = millis();
  unsigned long prevknock = timenow;
  unsigned long timediff = 0;
  Serial.print("Shock loop received at time: ");
  Serial.println(timenow);

  // Reset current knocks saved
  currKnockIndex = 0;

  // Debounce and wait
  delay(minKnockInterval);

  while ((timediff < KNOCK_LISTEN_TIMEOUT) && (currKnockIndex < MAX_KNOCKS)) {
    timenow = millis();
    timediff = timenow - prevknock;

    //    Serial.print("timenow (");
    //    Serial.print(timenow);
    //    Serial.print(") - prevknock (");
    //    Serial.print(prevknock);
    //    Serial.print(") = timediff (");
    //    Serial.print(timediff);
    //    Serial.println(")");

    if (knockDetected()) {
      Serial.print("Knock detected at time: ");
      Serial.println(timenow);

      currKnockIntervals[currKnockIndex] = timediff;
      currKnockIndex++;
      prevknock = timenow;

      // Debounce and wait
      delay(minKnockInterval);
    }
  }

  //  if (!(timediff < KNOCK_LISTEN_TIMEOUT)) {
  //    Serial.print("ENDED: final timediff = ");
  //    Serial.println(timediff);
  //  } else if (!(currKnockIndex < MAX_KNOCKS)) {
  //  }

  // Analyze knock
  analyzeKnock();
}

static void readKnock(void) {
  return;
}


/** Motor Control **/
void turnOff() {
  Serial.println("Turning off");
  Serial.println("");
  digitalWrite(PIN_POS, LOW);
  digitalWrite(PIN_NEG, LOW);
}

void turnFront() {
  turnOff();
  Serial.println("Turning front");
  Serial.println("");
  digitalWrite(PIN_POS, HIGH);
  digitalWrite(PIN_NEG, LOW);
}

void turnBack() {
  turnOff();
  Serial.println("Turning back");
  Serial.println("");
  digitalWrite(PIN_POS, LOW);
  digitalWrite(PIN_NEG, HIGH);
}
/** End Motor Control **/
```