ME 102B Final Project Report

Team 23: Smart Door Lock

Victor Ceja , Yuqi Tang , Ryan Christopher Montis

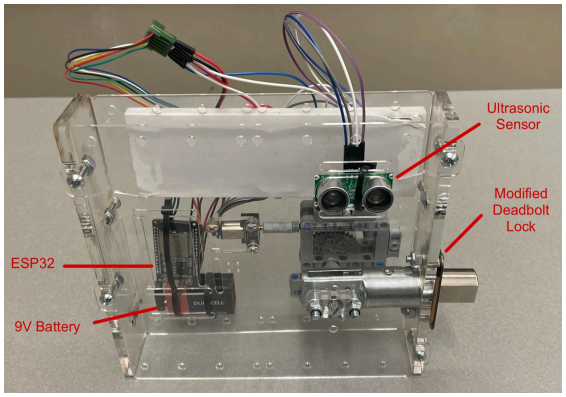
Opportunity: How might we make residential security more convenient in the modern world, without sacrificing safety?

High Level Strategy: Our initial design aimed to provide hands-free locking and unlocking through proximity detection, paired with features to ensure safety and reliability. The final product met and, in some areas, exceeded our expectations.

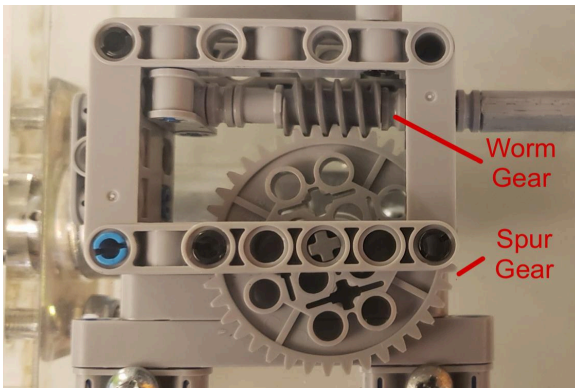
- **Proximity Unlocking:** This proximity feature worked well in our final design by only unlocking the door through motion after entering “unlock” in the app instead of just the proximity detection of a smart key without any inputs.
- **Locking Time:** Operates within 1-2 seconds, outperforming the original goal of 3 seconds.
- **Additional App Control:** Initially, the smart lock was only supposed to open through proximity and physical key entry, but with the addition of the smart app we can unlock and lock through “unlock” and “lock” commands given from the app.
- **Motor Strength:** From our initial calculations of the motor strength we found that the motor would be plenty powerful enough for our device and in the final product proved to lock and unlock with ease and proved to be strong enough to strip the custom motor shaft coupling when not given the proper controls.

Device Overview

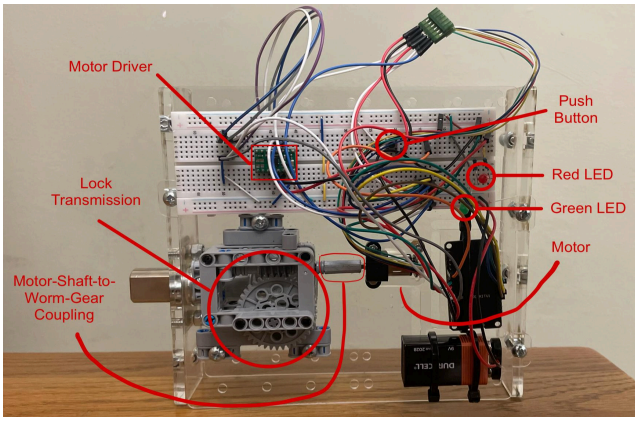
“Outside of Door” View



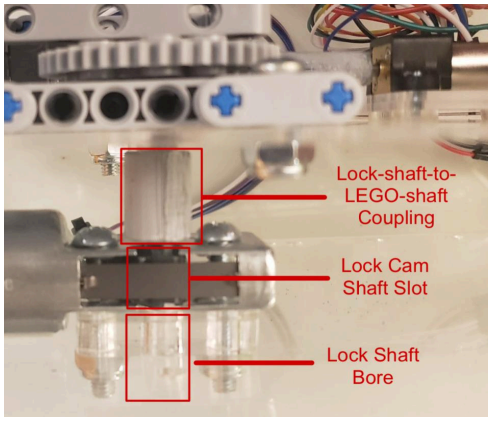
Lock Transmission - Front View



“Inside of Door” View



Lock Transmission - Bottom View



Function-Critical Design Decisions and Calculations

- **Mechanical Design:** The mechanical design focuses on reliability and security. A worm gear transmission was selected for its inherent self-locking capabilities, which prevent the lock from disengaging unless powered. This ensures robust security against forced entry.
- **Actuation and Load-Bearing Components:** The DC motor, paired with an encoder, provides precise control over the lock's movement. The motor is closely connected to the worm gear through a coupling, with minimal radial loads being applied to any of the shafts, minimizing the need for bearings and simplifying the assembly (bearings would still be an optimal choice for long-term durability). The worm gear transmits torque to the deadbolt lock mechanism at a great mechanical advantage, ensuring smooth operation with reduced load to transmission components.

Calculations

- **Motor Torque:** The torque required to actuate the deadbolt was assumed based on other similar models as 0.35 Nm but calculated to be 0.2Nm for our model. A motor capable of delivering 0.5 Nm was selected to include a safety margin, ensuring reliable operation.
- **Load Analysis:** The worm gear and associated components were analyzed to withstand loads of up to 20N, ensuring the system's integrity under typical use conditions. Also being able to output 386.59 Nm of torque with the given motor selection and the 40:1 gearing ratio
- **Speed and Efficiency:** The gear ratio was optimized to balance speed and torque, achieving a locking time of 1-2 seconds while maintaining power efficiency. The system consumes approximately 2W during active operation and 0.1W in standby mode, meeting design goals for energy efficiency.

Component Specifications

- **Motor:** 75:1 Micro Metal Gearmotor HP (6V), providing 0.5 Nm torque.
- **Transmission:** Worm gear and spur gear made from high-strength plastic, providing effective proof-of-concept functionality at a low cost.
- **Control Components:** ESP32 microcontroller for communication and processing, with an H-Bridge motor driver for actuation control.

MATLAB Code for Component Feasibility Verification:

```
% Motor and transmission specs Given
gear_ratio_motor = 75.81;            % Motor gear ratio (75:1)
worm_gear_ratio = 40;                % Worm gear transmission (40:1)
total_gear_ratio = gear_ratio_motor * worm_gear_ratio;  % Total gear ratio

stall_torque_kg_cm = 1.3;            % Stall torque of the motor (in kg-cm)
stall_torque_Nm = stall_torque_kg_cm * 0.0980665;  % Convert kg-cm to Nm

% Lock bolt specifications
deadbolt_force_N = 10;               % ASSUMED: Force required to move deadbolt (in Newtons)
lever_arm_length_m = 0.02;           % ASSUMED: Lever arm length in meters
%                                       (assumed to be 2 cm for cam lobes)

% Calculate output torque at the worm gear's output shaft
output_torque_Nm = stall_torque_Nm * total_gear_ratio;  % Stall torque at the output shaft

% Calculate the required torque to move the deadbolt
required_torque_Nm = deadbolt_force_N * lever_arm_length_m;  % Torque needed to move deadbolt
```

```
% Check if the motor's output torque is sufficient
if output_torque_Nm >= required_torque_Nm
    fprintf('Success: The motor provides enough torque to move the deadbolt.\n');
else
    fprintf('Warning: The motor does not provide enough torque to move the deadbolt.\n');
end
```

```
Success: The motor provides enough torque to move the deadbolt.
```

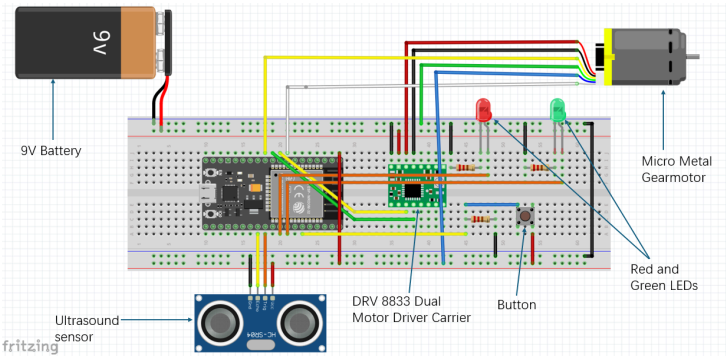
```
% Display relevant values
fprintf('Output Torque at Worm Gear: %.2f Nm\n', output_torque_Nm);
```

```
Output Torque at Worm Gear: 386.59 Nm
```

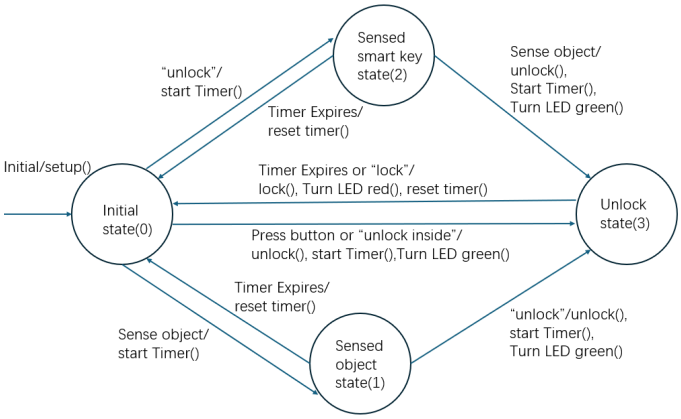
```
fprintf('Required Torque to Move Deadbolt: %.2f Nm\n', required_torque_Nm);
```

```
Required Torque to Move Deadbolt: 0.20 Nm
```

Circuit Diagram



State Transition Diagram



Reflection: Through this project, we gained valuable insights into the importance of effective communication and the strategic allocation of time and resources. Utilizing Discord proved highly advantageous, as it enabled seamless cross-platform communication and file sharing, greatly enhancing our collaboration. Additionally, we learned that reviewing deliverables thoroughly and well in advance gives a clearer understanding of priorities, particularly for ordering and manufacturing parts, and is crucial for allowing sufficient time for full-system integration and debugging.

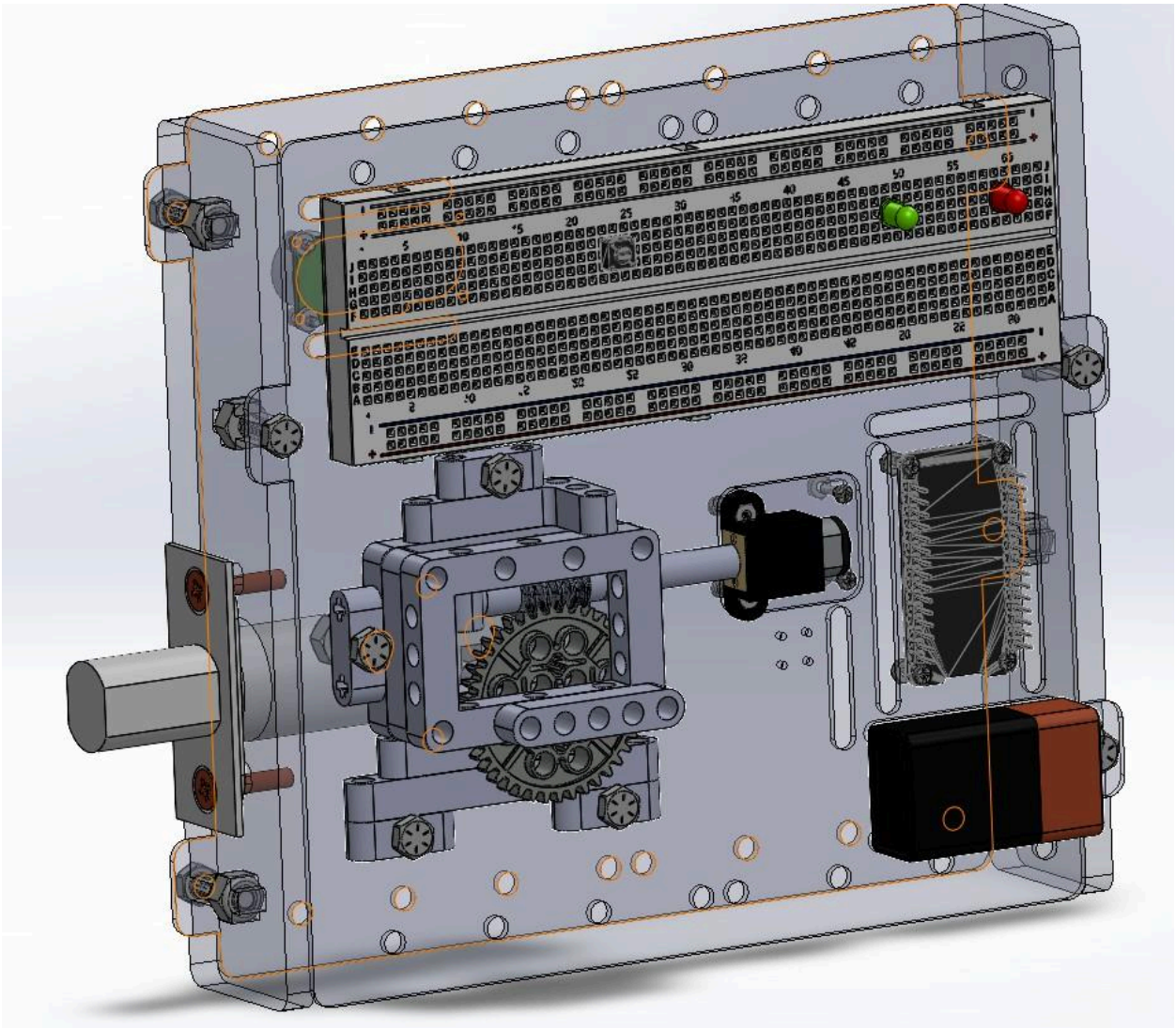
Appendices

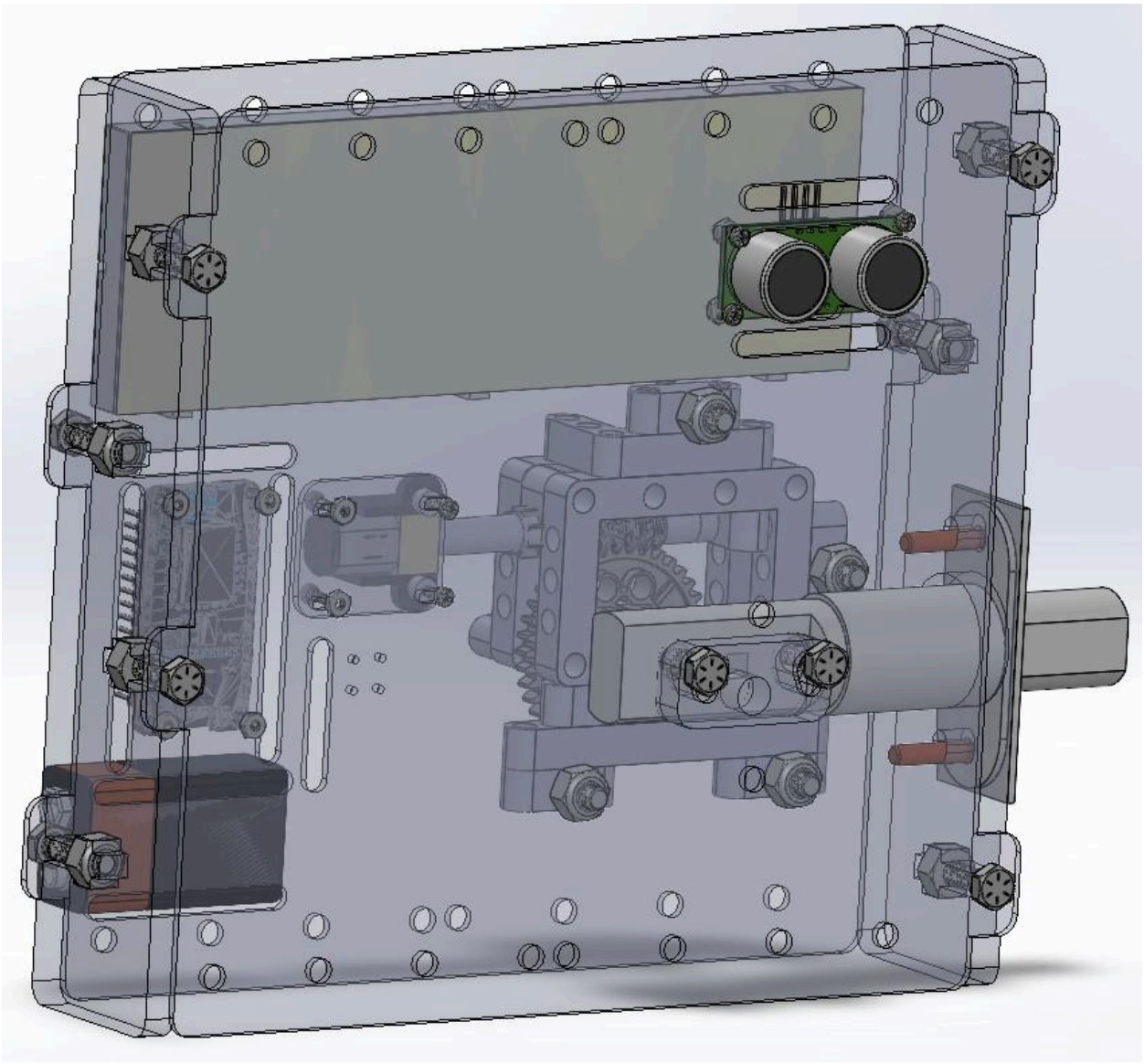
- **Bill of Materials**

Name	QTY	Price per	Source
Deadbolt Door Lock	1	$32.97	Schlage B60 Series Antique Brass Single Cylinder Deadbolt Certified Highest for Security and Durability B60N 609 - The Home Depot
582 PCS Lego Technic Compatible Parts Set	1	$27.99	582pcs Technical Parts and Pieces
Duracell 9V Battery	1	$4.25	Duracell Coppertop 9V Battery, 4 Count Pack (1 used)
100+PCS Lego Technic Compatible Set	1	$9.99	100+PCS Technic Gears and Axles
9V Battery Connector	1	$0.50	9V Battery Connector,10 PCS T-Type (1 used)
Custom 3D Printed Lock-Shaft-to-Lego Shaft Coupling	1	$0.03	Overture PLA 3D Printer Filament 1.75mm (1.5g used) Overture PLA
Custom 3D Printed Motor-Shaft-to-Lego-Shaft Coupling	1	$0.02	Overture PLA 3D Printer Filament 1.75mm (1g used) Overture PLA
DRV8833 Dual Motor Driver Carrier	1	$0	Microkit https://www.pololu.com/product/2130
75:1 Micro Metal Gearmotor HP 6V with Extended Motor Shaft	1	$0	Microkit https://www.pololu.com/product/2215
M1.6 6 mm Screws	4	$0	Microkit
M1.6 Nuts	4	$0	Microkit
ESP32 Microcontroller	1	$0	Microkit

LED - Green	1	$0	Microkit
LED - Red	1	$0	Microkit
Ultrasonic Sensor	1	$0	Microkit
Bread Board	1	$0	Microkit
10-32 Lock Nuts	15	$0	Jacobs Hall Makerspace
10-32 ¾ “ Screws	13	$0	Jacobs Hall Makerspace
10-32 1” Screws	2	$0	Jacobs Hall Makerspace
Clear Acrylic 1/16” x 8” x 15”	1	$0	Jacobs Hall Makerspace Scrap Bin
Clear Acrylic 1/4” x 5” x 8”	1	$0	Jacobs Hall Makerspace Scrap Bin
Zip Ties	4	$0	Jacobs Hall Makerspace
Total Price:		$75.75	

- **CAD Images**





- **Code Screenshots**

```
#include <ESP32Encoder.h>
#include "BLEDevice.h"
#include "BLEServer.h"
#include "BLEUtils.h"
#include "BLE2902.h"
ESP32Encoder encoder;
volatile int omg = 0;
volatile int D = 0;
int Kp=2;
int Ki=100;
int IMax = 15;
int sum_e=0;
const char *bleName = "Smart_lock";
const int threshold= 30;
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;
String receivedText = "";
BLECharacteristic *Characteristicptr;
volatile int state;
volatile int count = 0;
volatile int ultrasoundread;
volatile int duration_us;
volatile bool timerflag= false;
volatile int desomg=0;
volatile bool deltaT = false;
volatile bool buttonIsPressed=false;
hw_timer_t* timer1 = NULL;
#define SERVICE_UUID           "165df3cf-697d-4faf-a09f-e45aa9e1a647"
#define CHARACTERISTIC_UUID "fcd907e9-e55a-445f-b560-e9f07b74f15f"
```

```
#define LED_PIN_RED 18
#define LED_PIN_GREEN 19
#define US_PIN_IN 16
#define US_PIN_OUT 17
#define UNLOCK 0
#define LOCK 3700
#define BIN_1 26
#define BIN_2 25
#define buttonPIN 21
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
hw_timer_t* timer = NULL;

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount();
  encoder.clearCount();
  deltaT = true;  // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR onTime() {
  timerStop(timer);
  portENTER_CRITICAL_ISR(&timerMux);
  timerflag=true;  // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux);
}

void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  buttonIsPressed = true;
}

void setup() {
```

```
  Serial.begin(115200);
  state=0;
  pinMode(LED_PIN_RED,OUTPUT);
  pinMode(LED_PIN_GREEN,OUTPUT);
  pinMode(US_PIN_IN,INPUT);
  pinMode(US_PIN_OUT,OUTPUT);
  pinMode(buttonPIN,INPUT);
  attachInterrupt(buttonPIN, isr, RISING);
  timer = timerBegin(1000000);            // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer, &onTime);  // Attach onTimer0 function to our timer.
  timerAlarm(timer, 15000000, false, 0);
  timerStop(timer);
  setupBLE();
  turnledred();
  ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the weak pull up resistors
  encoder.attachHalfQuad(27, 33);                           // Attache pins for use as encoder pins
  encoder.setCount(0);
  timer1 = timerBegin(1000000);            // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to our timer.
  timerAlarm(timer1, 10000, true, 0);      // 10000 * 1 us = 10 ms, autoreload true
  ledcAttach(BIN_1, freq, resolution);
  ledcAttach(BIN_2, freq, resolution);
  desomg=LOCK;
  omg=LOCK;
}

void loop() {
  calculatedis();
  feedbackcontrol();
  switch(state){//state machine
    case 0:// initial state
      if (ultrasoundread<=threshold){//event checker: ultrasound sensor detect object within range
        Serial.println("sense object within range, enter state 1");
```

```
100           starttimer();//service function start timer()
101           state=1;
102         }
103         else if (receivedText=="unlock inside"){// event checker: smart key input instruction "unlock inside"
104           Serial.print("Received message: ");
105           Serial.println(receivedText);
106           Serial.println("smart key use unlock inside instruction, enter state 3, unlock");
107           Characteristicptr->setValue(receivedText.c_str());
108           receivedText = "";//reset receivedText
109           unlock();// service function unlock()
110           starttimer();//service function start timer()
111           turnledgreen();// service function turn led green()
112           state=3;
113         } else if(receivedText=="unlock"){//event checker: smart key input instruction "unlock"
114           Serial.print("Received message: ");
115           Serial.println(receivedText);
116           Serial.println("smart key use unlock instruction, enter state 2");
117           Characteristicptr->setValue(receivedText.c_str());
118           receivedText = "";//reset receivedText
119           starttimer();//service function start timer()
120           state=2;
121         } else if(buttonpressed()){//event checker: press button
122           Serial.println("press button, enter state 3");
123           unlock();// service function unlock()
124           starttimer();//service function start timer()
125           turnledgreen();// service function turn led green()
126           state=3;
127         }
128         break;
129       case 1:
130         if (receivedText=="unlock"){//event checker:smart key input instruction "unlock"
131           Serial.print("Received message: ");
132           Serial.println(receivedText);
```

```
133           Serial.println("smart key use unlock instruction, enter state 3, unlock");
134           Characteristicptr->setValue(receivedText.c_str());
135           receivedText = "";
136           unlock();//service function unlock()
137           starttimer();//service function start timer()
138           turnledgreen();//service function turn led green()
139           state=3;
140         } else if(timerflag){
141           Serial.println("timer expire, enter state 0, lock");
142           state=0;
143           resettimer();//service function reset timer()
144         }
145         break;
146       case 2:
147         if (ultrasoundread<=threshold){//event checker: ultrasound sensor detect object within range
148           Serial.println("sense object within range, enter state 3, unlock");
149           unlock();//service function unlock()
150           starttimer();//service function start timer()
151           turnledgreen();//service function turn led green()
152           state=3;
153         } else if(timerflag){
154           Serial.println("timer expire, enter state 0, lock");
155           state=0;
156           resettimer();//service function reset timer()
157         }
158         break;
159       case 3:
160         if(receivedText=="lock"){//event checker: smart key input instruction "lock"
161           Serial.print("Received message: ");
162           Serial.println(receivedText);
163           Serial.println("smart key use lock instruction, enter state 0, lock");
164           Characteristicptr->setValue(receivedText.c_str());
165           receivedText = "";
```

```
166             timerStop(timer);
167             lock();//service function lock()
168             turnledred();//service function turn led red()
169             state=0;
170             resettimer();//service function reset timer()
171           }else if(timerflag){//event checker: timer expire
172             Serial.println("timer expire, enter state 0, lock");
173             lock();//service function lock()
174             turnledred();//service function turn led red()
175             state=0;
176             resettimer();//service function reset timer()
177           }
178           break;
179       }
180       if (receivedText.length() > 0) {
181         Serial.print("Received message: ");
182         Serial.println(receivedText);
183         Characteristicptr->setValue(receivedText.c_str());
184         receivedText = "";
185       }
186     }
187
188     class InputCharacteristicCallbacks : public BLECharacteristicCallbacks {
189       void onWrite(BLECharacteristic *Characteristicptr) {
190         std::string value = std::string(Characteristicptr->getValue().c_str());
191         receivedText = String(value.c_str());
192       }
193     };
194
195     void setupBLE() {
196       BLEDevice::init(bleName);
197       BLEServer *Serverptr = BLEDevice::createServer();
198       BLEService *Serviceptr = Serverptr->createService(SERVICE_UUID);
```

```
  Characteristicptr = Serviceptr->createCharacteristic(CHARACTERISTIC_UUID, BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_READ);
  Characteristicptr->setAccessPermissions(ESP_GATT_PERM_READ_ENCRYPTED | ESP_GATT_PERM_WRITE_ENCRYPTED);
  Characteristicptr->setCallbacks(new InputCharacteristicCallbacks());
  Serviceptr->start();
  Serverptr->getAdvertising()->start();
  BLESecurity *Securityptr = new BLESecurity();
  Securityptr->setStaticPIN(999999);
  Serial.println("Waiting for a client connection...");              // Wait for a client connection
}

void turnledred(){
  digitalWrite(LED_PIN_RED,HIGH);
  digitalWrite(LED_PIN_GREEN,LOW);
}

void turnledgreen(){
  digitalWrite(LED_PIN_RED,LOW);
  digitalWrite(LED_PIN_GREEN,HIGH);
}

void unlock(){
  omg=LOCK;
  desomg=UNLOCK;
}

void lock(){
  omg=UNLOCK;
  desomg=LOCK;
}

void starttimer(){
  timerflag=false;
  timerWrite(timer,0);
```

```
232     if(state==0){
233       timerStart(timer);
234     }
235   }
236
237   void resettimer(){
238     timerflag=false;
239     timerWrite(timer,0);
240     buttonIsPressed=false;
241   }
242
243   void calculatedis(){
244     digitalWrite(US_PIN_OUT,HIGH);
245     delayMicroseconds(10);
246     digitalWrite(US_PIN_OUT,LOW);
247     duration_us=pulseIn(US_PIN_IN,HIGH);
248     if(duration_us!=0){
249       ultrasoundread=duration_us*0.017;
250     }
251     //Serial.println(ultrasoundread);
252   }
253
254   void feedbackcontrol(){
255     if (deltaT){
256       portENTER_CRITICAL(&timerMux1);
257       deltaT = false;
258       portEXIT_CRITICAL(&timerMux1);
259       omg-=count;
260       int e;
261       e=desomg-omg;
262       sum_e+=e;
263       int Ki_sum_e=0;
264       if (sum_e>0 && sum_e/Ki>IMax){
```

```
      Ki_sum_e=IMax;
      sum_e-=e;
    } else if(sum_e<0 && sum_e/Ki<-IMax){
      Ki_sum_e=-IMax;
      sum_e-=e;
    } else{
      Ki_sum_e=sum_e/Ki;
    }

    D=e/Kp+Ki_sum_e;
    Serial.println(D);
    Serial.println(omg);
    Serial.println(desomg);
    if (D > MAX_PWM_VOLTAGE) {
      D = MAX_PWM_VOLTAGE;
    } else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
    }
    if (D > 0) {
      ledcWrite(BIN_1, LOW);
      ledcWrite(BIN_2, D);
    } else if (D < 0) {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, -D);
    } else {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, LOW);
    }
  }
}

bool buttonpressed(){
  if(state==0){
```

```
    Serial.println(omg);
    Serial.println(desomg);
    if (D > MAX_PWM_VOLTAGE) {
      D = MAX_PWM_VOLTAGE;
    } else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
    }
    if (D > 0) {
      ledcWrite(BIN_1, LOW);
      ledcWrite(BIN_2, D);
    } else if (D < 0) {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, -D);
    } else {
      ledcWrite(BIN_2, LOW);
      ledcWrite(BIN_1, LOW);
    }
  }
}

bool buttonpressed(){
  if(state==0){
    if(buttonIsPressed){
      buttonIsPressed=false;
      return true;
    } else{
      return false;
    }
  } else{
    buttonIsPressed=false;
    return false;
  }
}
```