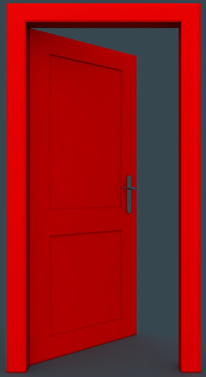
Final Project
Smart Door



Aisha Vazquez
Negina Atai

Intro

We created a smart door that enhances home security. Not only does it enhance security, but also enhances practicality and guest experience. One key problem that is resolved with our project is monitoring and controlling the door access remotely.

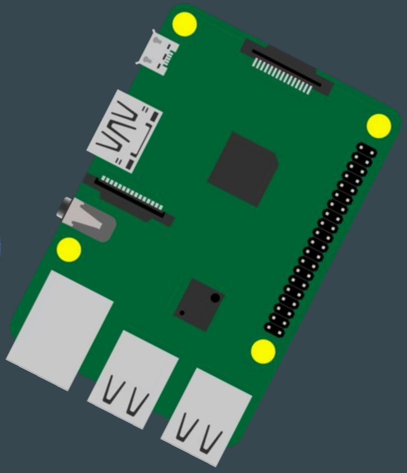
Project Objectives/Features

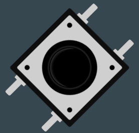
- Automated Door Control
- Proximity identifier
- Monitored Doorbell Activity
- Remote Security Enhancement

The primary objectives of this project are to create an automated door control system, enable security notifications, and incorporate LED feedback based on proximity. An LED connected to the system adjusts its brightness according to the distance of a detected object, providing visual feedback. Additionally, pressing a button acts as a doorbell, triggering a sound and sending a notification email, which informs the homeowner of a visitor. The door opens and closes based on the position of a switch, moving 90 degrees forward and back to allow or restrict access. Together, these features make the door responsive, secure, and interactive.

Hardware Components

- Raspberry Pi
- Stepper Motor
- Distance Sensor
- PWM LED
- Push Button
- Switch
- DC Connector (power supply)

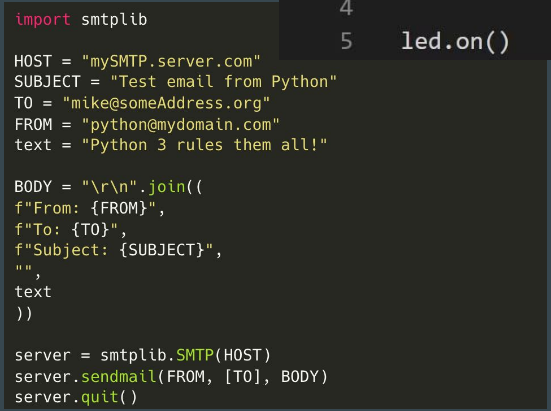




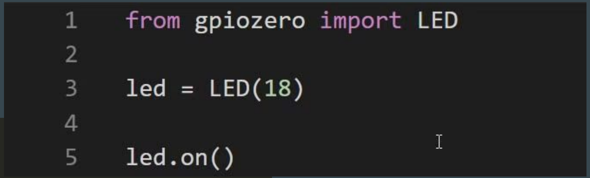
This project uses several key hardware components; the Raspberry Pi serves as the main controller, managing all input and output. The distance sensor detects proximity, controlling the brightness of a PWM LED. A push button acts as a doorbell, triggering a sound and email notifications. A stepper motor, allows precise 90-degree door movements. The switch is used to control the door's open and closed positions, while a breadboard, jumper wires, and power supply connect and power all components.

Software Components

- Raspi
- RPI.GPIO and gpiozero for GPIO control
- Smtplib for email notification
- Subprocess
- Time
- DateTime



```python
import smtplib

HOST = "mySMTP.server.com"
SUBJECT = "Test email from Python"
TO = "mike@someAddress.org"
FROM = "python@mydomain.com"
text = "Python 3 rules them all!"

BODY = "\r\n".join((
f"From: {FROM}",
f"To: {TO}",
f"Subject: {SUBJECT}",
"",
text
))

server = smtplib.SMTP(HOST)
server.sendmail(FROM, [TO], BODY)
server.quit()
```



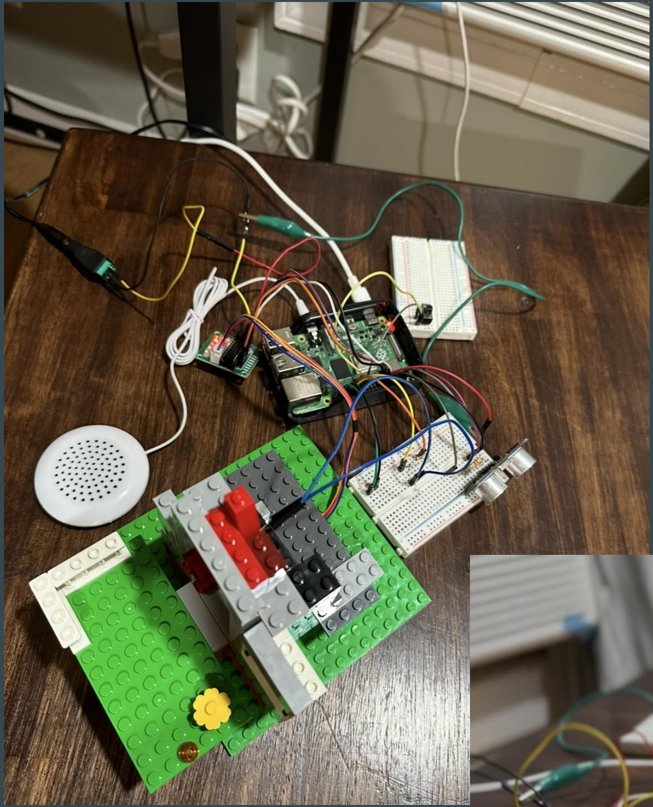
```python
from gpiozero import LED

led = LED(18)

led.on()
```

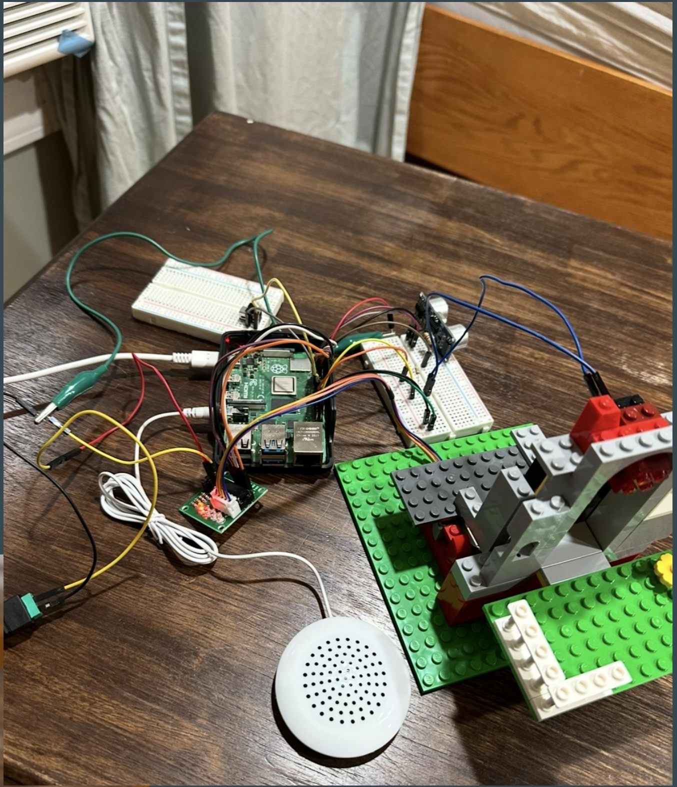
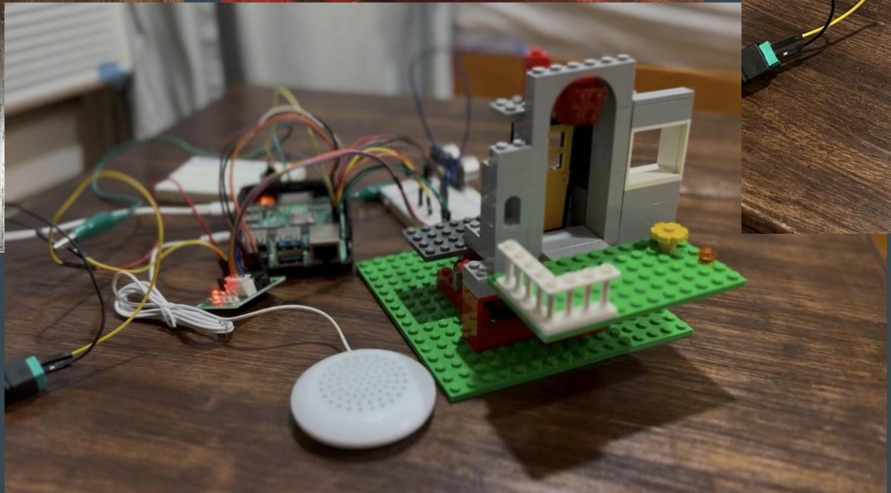
The project is programmed in Python, utilizing several libraries to control the hardware and enable additional functionalities. RPi.GPIO and gpiozero are used to control GPIO pins on the Raspberry Pi, allowing it to read inputs from sensors and switches and to control outputs like the motor and LED. smtplib enables the email notification function, sending alerts via Gmail's SMTP server when the doorbell button is pressed.

Project Approach

We approached the project step-by-step, building each part gradually. We began by researching and setting up the necessary circuits on the board using the recipes as a guide. Once each component was working as expected, we moved on to coding. After confirming the functionality of each component, we added additional parts one at a time and updated the code to integrate them into the overall system.

- Setup: Hardware assembly on the breadboard; wiring each component.
  - First set aside what GPIO’s will be used and color coding them to ensure there won't be a mix up when it comes to coding
- Programming: Functions for motor control, LED, and doorbell with email.
  - Had to know the order of steps in order for the program to execute correctly.
- Testing and Debugging:
  - Adjustments made for each code since we were piggybacking off of the last code
- Challenges: Debugging code, ensuring correct 90-degree motor rotation.



Code Overview:

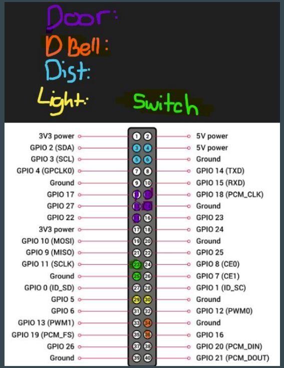
The code for the smart door project is organized into functions that handle each feature. The motor control function rotates the motor forward or backward by 90 degrees based on the switch state. The LED brightness control function uses the distance sensor to adjust brightness according to proximity, adding a visual indicator. The doorbell and email function triggers a doorbell sound and sends a notification email when the button is pressed. This code structure allows each function to work independently yet cohesively for the overall project.

1.

```
import RPi.GPIO as GPIO
import subprocess
import smtplib
from time import sleep
from datetime import datetime
from gpiozero import DistanceSensor, PWMLED

# Email configuration
GMAIL_USER = 'doorbellhome13@gmail.com'
GMAIL_PASS = 'lkpe bcly nmdm tkwj'
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587

# Set up email details
def send_email(recipient, subject, text):
    smtpserver = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
    smtpserver.ehlo()
    smtpserver.starttls()
    smtpserver.ehlo()
    smtpserver.login(GMAIL_USER, GMAIL_PASS)
    header = f'To: {recipient}\nFrom: {GMAIL_USER}\nSubject: {subject}\n'
    msg = header + '\n' + text + '\n\n'
    smtpserver.sendmail(GMAIL_USER, recipient, msg)
    smtpserver.close()


# Function to play the doorbell sound
def play_doorbell_sound():
    subprocess.run(["aplay", "doorbell.wav"])

# GPIO setup for doorbell button, distance sensor, LED, and stepper motor
button_pin = 16  # GPIO pin for doorbell button
switch_pin = 11  # GPIO pin for the switch controlling the motor

IN1 = 17  # GPIO pins for the stepper motor
IN2 = 18
IN3 = 27
IN4 = 22

GPIO.setmode(GPIO.BCM)
GPIO.setup(button_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)
GPIO.setup(IN3, GPIO.OUT)
GPIO.setup(IN4, GPIO.OUT)

# Define distance sensor and LED
sensor = DistanceSensor(echo=15, trigger=14)
led = PWMLED(5)

# Define motor step sequences
forward_sequence = [
    [1, 0, 0, 1],
    [1, 0, 0, 0],
    [1, 1, 0, 0],
    [0, 1, 0, 0],
    [0, 1, 1, 0],
    [0, 0, 1, 0],
    [0, 0, 1, 1],
    [0, 0, 0, 1],
]

backward_sequence = list(reversed(forward_sequence))
STEPS_90_DEGREES = 128  # Number of steps for 90-degree rotation, adjust if necessary
```

2.

```
# Function to rotate the motor
def rotate_motor(steps, delay=0.01):
    sequence = forward_sequence if steps > 0 else backward_sequence
    steps = abs(steps)
    for _ in range(steps):
        for step in sequence:
            GPIO.output(IN1, step[0])
            GPIO.output(IN2, step[1])
            GPIO.output(IN3, step[2])
            GPIO.output(IN4, step[3])
            sleep(delay)
#WHEN ACTIVE
try:
    current_position = "closed"
    previous_switch_state = GPIO.input(switch_pin)

    print("System is active. Press the button to ring the doorbell.")

    while True:
        # Check doorbell button state
        button_state = GPIO.input(button_pin)
        if button_state == GPIO.LOW:  # Button pressed
            play_doorbell_sound()
            current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            email_text = f'The doorbell rang at {current_time}'
            send_email('aisha.vaa@gmail.com', 'DoorBell Notification', email_text)
            sleep(1)  # Debounce delay

        # Check switch state for motor control
        switch_state = GPIO.input(switch_pin)
        if switch_state != previous_switch_state:
            previous_switch_state = switch_state
            if switch_state == GPIO.LOW and current_position == "closed":  # Open the door
                print("Opening door by rotating +90 degrees...")
                rotate_motor(STEPS_90_DEGREES)
                current_position = "open"
            elif switch_state == GPIO.HIGH and current_position == "open":  # Close the door
                print("Closing door by rotating -90 degrees...")
                rotate_motor(-STEPS_90_DEGREES)
                current_position = "closed"
            sleep(0.1)

        # Adjust LED brightness based on distance
        cm = sensor.distance * 100
        if cm <= 6:
            brightness = 1.0
        elif cm >= 10:
            brightness = 0.0
        else:
            brightness = 1 - ((cm - 6) / 4)
        led.value = brightness

        sleep(0.1)

except KeyboardInterrupt:
    print("Exiting program.")

finally:
    GPIO.cleanup()
```



Video Presentation

Conclusion

Successes: Achieved automated control, email alerts, and responsive LED.

Challenges: Initial wiring issues and debugging code logic.

Future Improvements:

- Add a camera for visitor photos.
- Remote access through a mobile app.
- Improved security features, such as authentication.

The project was successful in creating a functional smart door that can open and close, provide LED feedback, and notify the homeowner when someone is at the door. The main challenges included precise motor control and ensuring each component responded correctly to inputs, but these were overcome through testing and debugging. Looking ahead, this project could be improved by adding a camera to capture images of visitors, integrating remote control via a mobile app, and implementing additional security features like multi-factor authentication.
Overall, this project demonstrates the potential of smart home technology in enhancing security and convenience.

Q & A

Q: What was the most challenging part?

A: Since each piece of code was built on the previous one, coding itself wasn't the main challenge. However, one of the most difficult aspects was the final step: controlling the rotation of the door. This was especially challenging because we were not familiar with some of the more advanced code involved in the recipe for motor control.

Q: Did the project change from the original proposal?

A: Yes, the project evolved quite a bit. Initially, we didn't plan on including an actual door; the focus was on creating a "smart door" concept. It wasn't until our professor asked if we were building a physical door that we decided to add the door mechanism (otherwise, it might have just been a smart doorbell!). We also considered adding an OLED display to inform visitors if no one was home, but we decided against it, as it might not be ideal to publicly display that information.