Abdelati Zelbane, 2016

Acknowledgements Note

This document represents the culmination of a passionate side project driven by curiosity and the joy of exploration in the field of mechatronics. The Interactive Puzzle-Activated Unlocking Box is a personal endeavor by **Abdelati Zelbane**, developed entirely as a hobby project in 2016, while living in London.

The journey of creating this interactive puzzle box was enriched by the support and camaraderie of friends and colleagues, who provided invaluable assistance and shared in the fun of discovery.

Every step—from the initial sketches to the final prototype—was fueled by a desire to blend art, technology, and innovation for pure enjoyment.

It is with great pride and gratitude that I present this work as a testament to the spirit of creative exploration and the endless possibilities that arise when passion meets technology.

Interactive Puzzle-Activated Unlocking Box

Abdelati Zelbane,

London UK, 2016

Abstract

This paper presents the conceptualization, design, and prototyping of an interactive puzzle box that leverages multiple sensor modalities and mechatronic components. With a focus on blending artistic expression with engineering rigor, the project uses MDF for a custom enclosure, a Raspberry Pi as the central controller, and various sensors—including force resistors, gyroscopes, accelerometers, touch sensors, and a vision system—to create a series of challenges that must be solved to unlock the box. Detailed technical analysis, prototyping iterations, embedded Python code, and hardware integration strategies are provided.

1. Introduction

Interactive devices have become a hallmark of modern mechatronics, merging physical interfaces with digital intelligence. The Puzzle-Activated Unlocking Box is designed to be both an art object and a demonstration of engineering principles. The project challenges users with a series of sensor-based puzzles that require physical interaction and computational interpretation. This paper documents the entire process from conceptual design through multiple iterations of prototyping, including detailed technical analysis and sample code.

2. Project Concept and Design Objectives

2.1 Concept Overview

The core idea is to create a secure, puzzle-driven box that only opens after a predefined sequence of sensor inputs is detected. The puzzles are multi-modal:

- **RFID Authentication:** Recognizing specific tags.
- **Force/Pressure Pattern Recognition:** Interpreting pressure signals using force sensitive resistors (FSRs).
- **Motion and Orientation Detection:** Using a gyroscope and accelerometer to detect orientation or specific gestures.
- **Touch Interaction:** Incorporating capacitive or resistive touch sensors.
- **Visual Recognition:** Leveraging a camera and computer vision algorithms for gesture or object detection.

2.2 Design Objectives

- **Interactivity:** The box should engage users with varied sensor-based puzzles.
- **Robustness:** The design must tolerate variability in sensor data through calibration and filtering.
- **Scalability:** Additional sensors and puzzle mechanisms can be integrated in future iterations.
- **Aesthetic and Structural Integrity:** Constructed with MDF, the enclosure is both durable and visually appealing.
- **Modular Software:** The control code is designed in Python for ease of iteration and community sharing.

3. Materials and Components

3.1 Enclosure and Mechanical Parts

- **MDF Sheets:** For constructing the box enclosure with precision-cut panels.
- **Fasteners and Adhesives:** Wood screws, brackets, and non-permanent adhesives for assembly.
- **Servo Motors / Solenoids:** To actuate the locking mechanism.
-

3.2 Electronics

- **Raspberry Pi:** Chosen for its versatility and support for various I/O interfaces.
- **Force Sensitive Resistors (FSRs):** For pressure-sensing puzzles.
- **Gyroscope and Accelerometer:** Integrated sensors (e.g., MPU6050) for motion and orientation detection.
- **Touch Sensors:** Capacitive touch modules for interactive triggers.
- **Camera Module:** Raspberry Pi Camera for real-time video/image processing.
- **RFID Reader:** RC522 or similar module for tag detection.
- **Additional Sensors:** Options include temperature sensors, light sensors, or proximity sensors for future expansion.
- **Power Supply:** A regulated 5V/3.3V source with battery backup options.
-

3.3 Software and Programming Tools

- **Python 3.x:** Main programming language for sensor interfacing and logic.
- **OpenCV:** For image processing and gesture recognition.
- **GPIO Libraries:** Such as RPi.GPIO or gpiozero for sensor interfacing.
- **Calibration and Data Processing Algorithms:** Custom scripts to map sensor data to meaningful states.

4. Mechanical and Electrical Integration

4.1 Enclosure Fabrication Using MDF

4.1.1 Design and Iteration Process

- **Initial Design:** Create 2D drawings of the enclosure panels using CAD software.
- **Cutting:** Use a laser cutter or CNC router to precisely cut MDF panels.
- **Assembly:** Assemble the prototype using wood screws and brackets. Early prototypes may use a simple cardboard version before transitioning to MDF.
- **Iteration:** Test sensor placements by marking sensor locations on MDF, then adjust based on physical interactions and feedback.

4.1.2 Prototyping Phases

1. **Prototype 1 – Concept Validation:**
    - Build a simple mock-up with cutouts for sensors.
    - Evaluate sensor accessibility and aesthetic layout.
2. **Prototype 2 – Functional Prototype:**
    - Incorporate all sensors with wiring channels.
    - Test the integration between hardware and the Raspberry Pi.
3. **Prototype 3 – Final Iteration:**
    - Refine the enclosure for durability and aesthetics.
    - Integrate feedback mechanisms such as LEDs and sound buzzers.

4.2 Electrical Circuit Integration

4.2.1 Wiring Diagrams and Sensor Connectivity

- **Raspberry Pi GPIO:** Connect sensors via GPIO, I2C (for MPU6050), and SPI (for RFID).
- **Circuit Protection:** Use resistors and capacitors to stabilize sensor signals.
- **Power Distribution:** Ensure that the power regulator supplies appropriate voltage levels to each module.

4.2.2 Technical Analysis: Sensor Calibration

For instance, calibrating the FSR:

$$V_{\text{out}} = V_{\text{in}} \times \frac{R_{\text{FSR}}}{R_{\text{FSR}} + R_{\text{fixed}}}$$

Where:

- V_{in} is the input voltage.
- R_{FSR} is the variable resistance of the force sensor.
- R_{fixed} is a known resistor value used in the voltage divider.

Similarly, for the MPU6050 inertial measurement unit (IMU), the tilt angle θ in the XY plane can be estimated from the accelerometer readings as:

$$\theta = \arctan\left(\frac{a_y}{a_x}\right)$$

Where:

- a_x and a_y are the accelerometer readings in the X and Y directions respectively.

These formulas help convert raw sensor readings into interpretable physical quantities, enabling threshold-based decision-making and triggering specific actions in the system.

These formulas help convert raw sensor readings into interpretable values used for triggering specific actions.

5. Software Architecture and Programming

5.1 Overall Software Structure

The control software is structured as a state machine where each sensor puzzle represents a state transition. Only when all states are confirmed is the unlocking mechanism activated.

5.1.1 Module Overview

- **Sensor Interface Module:** Handles data acquisition from each sensor.
- **Data Processing Module:** Implements calibration, filtering, and interpretation algorithms.
- **State Machine Controller:** Manages the sequence of puzzles.
- **Feedback Module:** Provides visual (LEDs/LCD) or auditory (buzzer) feedback.
- **Unlocking Control:** Activates the servo motor or solenoid when conditions are met.

5.2 Python Code Examples

5.2.1 Sensor Reading and Calibration (Example: FSR)

test.py 1 ●

C: > Users > azelb > Downloads > test.py > ...

```python
import RPi.GPIO as GPIO
import time

# Example function to read FSR value
Qodo Gen: Options | Test this function
def read_fsr(channel, v_in=3.3, r_fixed=10000):
    # Simulated analog read using an ADC (since Raspberry Pi has no built-in ADC)
    analog_value = read_adc(channel)  # Placeholder for ADC read function
    # Map the analog value (0-1023) to voltage
    voltage = (analog_value / 1023.0) * v_in
    # Calculate FSR resistance using voltage divider formula
    r_fsr = (voltage * r_fixed) / (v_in - voltage) if voltage != v_in else float('inf')
    return voltage, r_fsr

# Example ADC read simulation function
Qodo Gen: Options | Test this function
def read_adc(channel):
    # For demonstration purposes; replace with actual ADC interfacing code.
    return 512  # Mid-range value

# Testing FSR reading
voltage, resistance = read_fsr(0)
print("FSR Voltage: {:.2f} V, Resistance: {:.2f} Ohms".format(voltage, resistance))
```

Snippet of basic code sample

5.2.2 State Machine Logic for Puzzle Sequencing

test.py

C: > Users > azelb > Downloads > test.py > ...

```python
Qodo Gen: Options | Test this class
class PuzzleBoxStateMachine:
    def __init__(self):
        self.states = {"rfid": False, "fsr": False, "motion": False, "touch": False, "gesture": False}

    Qodo Gen: Options | Test this method
    def update_state(self, sensor_name, status):
        if sensor_name in self.states:
            self.states[sensor_name] = status
        self.check_unlock_condition()

    Qodo Gen: Options | Test this method
    def check_unlock_condition(self):
        if all(self.states.values()):
            self.unlock_box()

    Qodo Gen: Options | Test this method
    def unlock_box(self):
        print("All puzzles solved! Unlocking the box...")
        # Activate unlocking mechanism here
        activate_servo()

Qodo Gen: Options | Test this function
def activate_servo():
    # Placeholder for servo activation code
    print("Servo activated. Box is unlocked.")

# Example usage:
puzzle_box = PuzzleBoxStateMachine()
puzzle_box.update_state("rfid", True)
puzzle_box.update_state("fsr", True)
puzzle_box.update_state("motion", True)
puzzle_box.update_state("touch", True)
puzzle_box.update_state("gesture", True)
```

Snippet of basic code sample

5.2.3 Gesture Recognition Using OpenCV

6.

test.py 1

C: > Users > azelb > Downloads > test.py > ...

```python
import cv2

Qodo Gen: Options | Test this function
def detect_gesture(frame):
    # Convert to grayscale and apply Gaussian blur
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    # Simple thresholding for demonstration
    _, thresh = cv2.threshold(blur, 60, 255, cv2.THRESH_BINARY)
    # Find contours to detect hand shapes (placeholder algorithm)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        # Assume the largest contour is the hand
        gesture_detected = True  # In a real implementation, analyze contour properties
    else:
        gesture_detected = False
    return gesture_detected

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    if detect_gesture(frame):
        print("Gesture detected!")
        # Update state machine accordingly
    cv2.imshow("Gesture Recognition", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Snippet of basic code sample

6.1 Sensor Fusion and Data Filtering

Combining data from multiple sensors requires noise reduction and synchronization. A simple complementary filter can fuse accelerometer and gyroscope data for reliable motion detection:

$$\theta_{\text{fused}} = \alpha \times (\theta_{\text{previous}} + \omega \Delta t) + (1 - \alpha) \times \theta_{\text{acc}}$$

Where:

- θ_{previous} is the previous angle estimate.
- ω is the angular velocity from the gyroscope.
- Δt is the time difference between readings.
- θ_{acc} is the angle computed from the accelerometer.
- α is the blending factor $(0 < \alpha < 1)$.

6.2 Thresholding and Calibration Algorithms

Each sensor's output is calibrated using empirical testing. For example, the FSR is calibrated by determining the resistance threshold for activation:

$$\text{Threshold}_{\text{FSR}} = \text{Mean}_{\text{no load}} + k \times \sigma_{\text{noise}}$$

Where:

- $\text{Mean}_{\text{no load}}$ is the average resistance reading without pressure.
- σ_{noise} is the standard deviation of the sensor noise.
- k is a sensitivity factor determined experimentally.

7.1 Initial Concept and Material Selection

- **Sketching and CAD Drawings:** Early sketches were converted into CAD designs for MDF cutting.
- **Material Tests:** MDF samples were tested for durability and sensor mounting feasibility.

7.2 Iteration 1: Basic Assembly

- **Prototype Construction:** A simple MDF box was assembled with cutouts for sensors.
- **Sensor Integration:** Early versions of sensor circuits were mounted and wired to the Raspberry Pi.
- **Feedback Loop:** Early tests highlighted issues in sensor alignment and wiring interference.

7.3 Iteration 2: Functional Testing

- **Refinement of Sensor Placement:** Adjustments were made to the MDF layout to optimize sensor responsiveness.
- **Software Debugging:** Python scripts were refined to improve sensor calibration and state management.
- **User Testing:** Informal tests with peers led to insights about the user experience and puzzle difficulty.

7.4 Final Prototype: Integration and Aesthetics

- **Final Enclosure Design:** A polished MDF enclosure with integrated wiring channels, LED indicators, and hidden sensor ports.
- **Robust Electronics Integration:** All sensors, feedback modules, and the locking mechanism were fully integrated.
- **Complete System Validation:** Extensive testing confirmed that all sensor puzzles trigger the correct state transitions and the box reliably unlocks when all conditions are met.

8. Conclusion and Future Work

This project serves as a proof-of-concept for interactive, sensor-based puzzle systems. The integration of multiple sensor types with a Raspberry Pi demonstrates how hardware and software can be coalesced into an engaging, multifaceted user experience. Future work may include:

- **Dynamic Puzzle Generation:** Introducing randomized puzzle patterns for increased replayability.
- **Remote Connectivity:** Incorporating IoT features to allow remote monitoring and control.
- **Enhanced Computer Vision:** Utilizing advanced machine learning algorithms for more sophisticated gesture recognition.
- **Extended Sensor Array:** Experimenting with additional sensors such as environmental monitors to expand the interactive potential.

9. References

For further reading and inspiration, please consult:

- Hackaday.io and Hackster.io for community projects and detailed build logs.
- Instructables for step-by-step tutorials on MDF construction and sensor integration.
- Technical datasheets for the MPU6050, RFID modules, and FSRs.
- OpenCV documentation for implementing robust computer vision algorithms.