EEC172 Final Project
BOLT: Bike Overwatch & Location Tracker

Anirudh Venkatachalam, Manasvini Narayanan
https://bolt-eec172.web.app/

I. Introduction

Our project emerged from a fear that all bike owners in Davis face: will my bike get stolen? Bike theft remains a persistent and stressful issue in Davis - a city known for its cycling culture. To directly address this problem, we developed **BOLT: Bike Overwatch & Location Tracker** which is a robust and proactive solution designed to continously safeguard bicycles.

When the bike is locked, the system actively monitors for unauthorized movements. If suspicious activity is detected, the system immediately alerts the bike owner through email along with the exact coordinates of latitude and longitude via the GT-U7 GPS module.

BOLT allows state transitions between `Locked`, `Unlocked` and `Alert` modes using IR remote and accelerometer. The integration of local visual alerts and cloud-based email notifications ensures immediate awareness and rapid response, greatly enhancing bike security and giving bike owners peace of mind.

II. Design

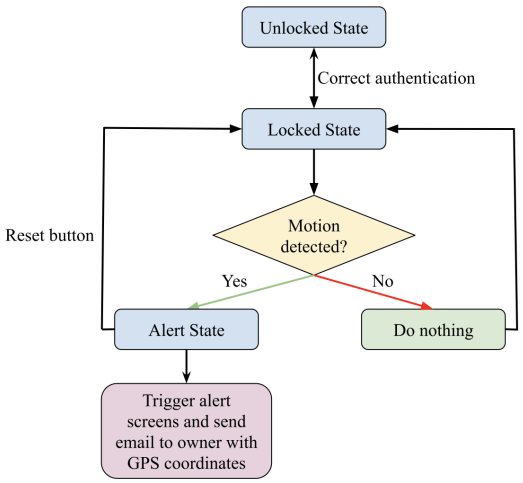

Fig. 1. State Machine for BOLT

Functional Specification

The BOLT system operates based on a structured state-machine model, ensuring seamless transitions between security modes. The system initially starts in the `Unlocked` state, allowing the bike to move freely without triggering any alerts. When the user sends the correct input via the IR remote, the system transitions into the `Locked` state, where it begins monitoring motion using the accelerometer. While in the `Locked` state, the system continuously checks for unauthorized movement. If no motion is detected, the system remains in this state without taking further action. However, if movement exceeds a predefined threshold, the system enters the ALERT state, where it retrieves the real-time GPS coordinates of the bike from the GT-U7 GPS module. The system then sends an alert notification to the owner's email via AWS IoT, providing the exact location of the bike. Once in the `Alert` state, the system remains active until the user presses the RESET button, which clears the alert and transitions the system back to the `Locked` state, ready to detect further unauthorized activity. At any time, the user can unlock the bike by sending the correct IR remote input, returning the system to the `Unlocked` state where movement is permitted without triggering alarms. This implementation ensures that the bike is continuously protected while providing the user with an easy and efficient way to control the security system. The seamless integration of motion detection, GPS tracking, and real-time notifications offers a proactive and reliable solution for theft prevention. The states are as follows

`LOCKED`: Security mode activated; the system continuously monitors movement using the accelerometer. Unauthorized motion activates the `ALERT` state.
`ALERT`: Unauthorized motion activates visual alerts and activates the AWS IoT SNS system. The state then reads data from the GPS module, parses the GPS coordinates and sends them to the bike owner. A reset (`SW3`) button resets the system back to the unlocked state.
`UNLOCKED`: Security disabled by the owner using an IR remote, allowing the bike to move freely without activating alerts.

System Architecture

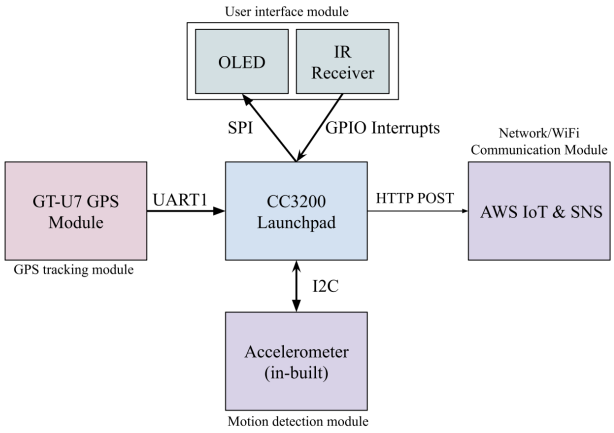

Fig. 2. Block diagram

Bolt has a very simple architecture that is easy to understand and follow. The following components were used

A. Microcontroller

- **CC3200 LaunchPad**: A microcontroller that is the central processor for movement detection, GPS tracking, WiFi communication, IR input decoding, and email notifications.

B. Motion Detection Module

- **Accelerometer (Bosch in built)**: Detects movement in the locked state

C. GPS Tracking Module

- **GT-U7 NEO-6M GPS Module**: Captures real-time geographic location data

D. User Interface Module

- **AT&T S10-S3 Universal Remote**: Transmits button pressed data via IR to the receiver
- **Vishay TSOP31336 IR Receiver Module**: Receives data from the remote and the data is read via GPIO Interrupts.
- **Resistor**: 100Ω
- **Capacitor**: 100μF
- **Adafruit 128px x 128px OLED Display**: Displays the OLED screens and provides interfaces for each state

E. Network Communication Module

- **Wireless Communication using IoT** (Wi-Fi & HTTP Requests): AWS SNS to send emails to the bike owner

F. Debugging

- **Saleae USB Logic Analyzer**: Used to debugging and decoding remote and GPS inputs

III. Implementation

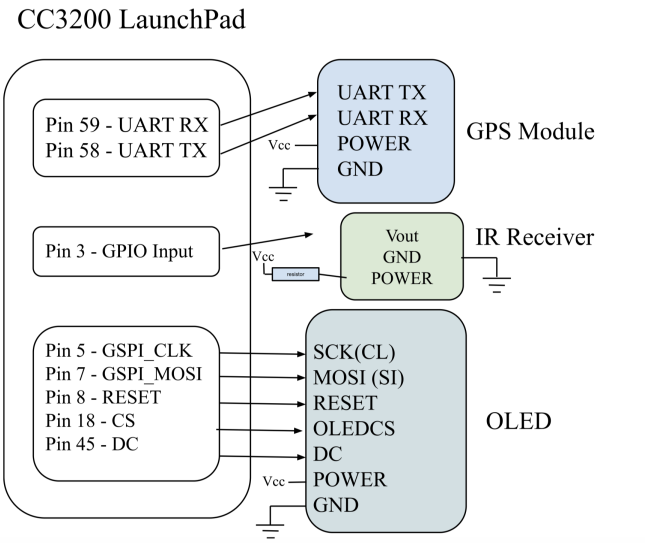

Fig. 3. Circuit connections

The implementation of BOLT required careful integration of hardware and software components to ensure a reliable and effective bike security system.

A. Microcontroller

1) Hardware Implementation:

- The CC3200 microcontroller is used as the central processing unit.
- It manages peripherals, processes inputs, and handles communication.
- Power is supplied from the computer it is connected to.

2) Software Implementation:

- Configures GPIO, UART, I2C, and SPI for different peripherals.
- Manages system states (Locked, Unlocked, Alert) and transitions.
- Handles secure HTTP requests for alert notifications.

B. Motion Detection (Accelerometer)

1) Hardware Implementation:

- Connects via I2C
  - SCL: Pin 1
  - SDA: Pin 2

2) Software Implementation:

- Reads acceleration values and processes data to determine unauthorized movement when system is `Locked`
- Compares speed thresholds to detect motion.
- Triggers an alert if unauthorized movement is detected. The alert state then shows an alert screen on the OLED and collects the location of the bike from the GPS module and initiates a POST to the bike owner's email with the coordinates

C. GPS Module)

1) Hardware Implementation:

- Uses UART1 for communication
- TX: Pin 58, RX: Pin

2) Software Implementation:

- Reads and parses `$GPRMC` NMEA sentences.
- Extracts latitude and longitude.
- Sends location updates via HTTP POST.

D. User Interface Module

1) Infrared (IR) Receiver:

- Hardware Implementation
  - Connected to GPIO Pin 3 for input.
  - Uses AT&T remote control for sending security commands.
- Software Implementation
  - Detects and decodes 48-bit IR signals via GPIO Interrupts
  - Compares the last 16 bits with predefined commands.
  - Triggers actions such as unlocking and locking

2) OLED Display:

- Hardware Implementation
  - Uses an Adafruit 128x128 SSD1351 OLED.
  - Communicates via SPI using:
    - CS: Pin 18
    - DC: Pin 45
    - MOSI: Pin 7
    - CLK: Pin 5
    - RESET: Pin 8
    - Vcc: 3V on CC3200
    - GND: GND on CC3200

3) Reset Button: The SW3(`GPIO13`) switch is used to reset the system when the system is in ALERT state. It is meant to mimic the real life sitation that when a bike is stolen, it should remain in ALERT state till the owner finds the bike and resets it themselves.

- Software Implementation
  - Displays system status messages (Locked, Unlocked, Alert and Resetting).

E. Wireless Communication

1) Hardware Implementation:

- Uses CC3200's built-in Wi-Fi module.
- Connects to an access point.

2) Software Implementation:

- Establishes a secure connection to AWS IoT.
- Sends HTTP POST requests with GPS data upon motion detection.
- Uses TLS encryption for secure data transmission.

IV. Challenges

During the implementation of BOLT, several technical challenges arose that required extensive research, troubleshooting, and iterative problem-solving. The most significant challenges included handling GPS data parsing, ensuring stable communication with AWS, and calibrating motion detection for accurate security alerts.

Component	Function	Pin	Direction
Accelerometer	I2C SCL	PIN_01	Output
Accelerometer	I2C SDA	PIN_02	Bidirectional
IR Receiver	GPIO Input	PIN_03	Input
Reset Button	GPIO Input	PIN_04	Input
SPI Clock	GSPI_CLK	PIN_05	Output
SPI MOSI	GSPI_MOSI	PIN_07	Output
Reset	GPIO Output	PIN_08	Output
Chip Select	GPIO Output	PIN_18	Output
Data/Command	GPIO Output	PIN_45	Output
Debug UART	UART0 TX	PIN_55	Output
Debug UART	UART0 RX	PIN_57	Input
GPS Module	UART1 TX	PIN_58	Output
GPS Module	UART1 RX	PIN_59	Input
Indicator LED	GPIO Output	PIN_64	Output

TABLE I
PIN CONFIGURATION TABLE

A. GPS Module Data Parsing and Integration

The GPS module utilized UART1 for communication and continuously outputted NMEA (National Marine Electronics Association) sentences containing various types of navigational data. Initially, we struggled to interpret the raw NMEA data, as it consisted of multiple message types, most of which were not directly relevant to our application.

To address this, we conducted extensive research into NMEA 0183 standards and identified that the $GPRMC (Recommended Minimum Specific GPS/Transit Data) sentence contained the most critical information, including latitude, longitude, speed, and timestamp. We developed a custom parsing algorithm to extract only the essential data, reducing unnecessary processing overhead.

Furthermore, our initial implementation attempted to read GPS data using UART interrupts. However, since the GPS module transmits data every second, the system was flooded with interrupts, significantly impacting performance. The solution was to shift from an interrupt-driven approach to a polling mechanism, where data was read at a controlled interval. This reduced system overhead and prevented instability.

B. Unstable AWS IoT Connectivity and JSON Formatting Issues

Another major challenge was maintaining consistent connectivity with AWS IoT for sending real-time location updates. During testing, we observed that the connection to AWS would randomly fail or become intermittently unavailable, requiring a complete system restart to re-establish communication.

To diagnose this issue, we analyzed the MQTT/TLS handshake process and verified TLS certificate validity, network stability, and cloud service response times. Despite this, occasional failures persisted. However, we discovered that by running the demo before our program often fixed the problem.

Additionally, sending properly formatted JSON payloads was critical for AWS to process our location updates correctly. Initially, formatting errors in the HTTP POST request resulted in AWS rejecting the data. We refined our request structure,

ensuring correct JSON syntax and embedding latitude and longitude within the ”default” object in the desired format. Through debugging and testing, we successfully achieved consistent data transmission.

C. Motion Detection Calibration to Minimize False Positives

The accelerometer-based motion detection required precise calibration to prevent false alarms while ensuring reliable theft detection. Initially, the accelerometer was too sensitive, triggering alerts even with minor vibrations such as wind movement or accidental bumps.

To fine-tune the detection mechanism, we conducted multiple tests by adjusting the acceleration threshold and implementing a time-based filter. Instead of reacting to a single instance of motion, the system was configured to trigger an alert only if motion was detected continuously over a specific time window. This significantly improved reliability by distinguishing between genuine movement (e.g., theft) and environmental noise.

V. Future Work

- We believe that adding biometric verification or an app-based unlock mechanism would increase the security
- We believe that developing a companion app for tracking and controlling the lock remotely would be highly useful.
- Geofencing along with Google Maps API is an interesting feature as we would know where the bike is visually and also be alerted if it has left a certain perimeter.

VI. Bill of Materials

Component	Price (USD)	Source / Link
CC3200 LaunchPad	$55.00	Texas Instruments
Adafruit OLED Display	$39.95	Adafruit
Pack of 2 GPS Module (NEO-6M)	$18.99	Amazon
Jumper Wires	$6.98	Amazon
AT&T S10-S3 Universal Remote	$33.00	Amazon
Vishay TSOP31336 IR Receiver Module	$0.77	Amazon
Total	**$154.69**	

TABLE II
Bill of Materials (BOM)