# AUTOMATED TRAFFIC SIGNAL CONTROLLED IN EMERGENCY SITUATION

**A MINI PROJECT REPORT**

**EC23401-DIGITAL SIGNAL PROCESSING**

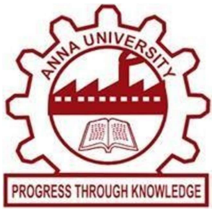
***Submitted by***


| | |
|---|---|
| **SUPRIYA R** | **2023504052** |
| **NIVETHITHA V** | **2023504011** |
| **RAJESHWARI K** | **2023504025** |


**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**





**DEPARTMENT OF ELECTRONICS ENGINEERING**

**MADRAS INSTITUTE OF TECHNOLOGY**

**ANNA UNIVERSITY: CHENNAI 600 044**


**APRIL 2025**

## BONAFIDE CERTIFICATE

Certified that this project report **"AUTOMATED TRAFFIC SIGNAL CONTROLLED IN EMERGENCY SITUATION"** is the bonafide work of

| | |
|---|---|
| **SUPRIYA R** | **2023504052** |
| **NIVETHITHA V** | **2023504011** |
| **RAJESHWARI K** | **2023504025** |

**SIGNATURE**

**Dr. M. VASIM BABU**

**SUPERVISOR**

Assistant Professor

Department of Electronics Engineering.

Madras Institute of Technology,

Anna University, Chennai- 600 044.

## ABSTRACT

The world demands for fast signal transmitting and receiving and the ability to connect devices simulataneously has pushed wireless communication technologies**.** In today's connected world, wireless signal transmission plays a vital role in emergency response systems, enabling real-time communication between vehicles and infrastructure. This project leverages this technology to help ambulances cut through traffic more efficiently. Wireless communication between vehicles and infrastructure is revolutionizing emergency response systems. Our project harnesses this potential by combining Arduino-based hardware control with mobile technology to create a smart ambulance priority system. This project uses an arduino board for receiving signal. It's realtime processing capability and input and output control, cost effectiveness makes it ideal for this Application. Our project uses an innovative mobile based solution for emergency vehicle priority signalling, replacing RF transmitter and receiver with smartphone apps for wireless communication.Our project provides uninterrupted passage for ambulance which is in emergency condition.

# INDEX

**AUTOMATED TRAFFIC SIGNAL CONTROLLED IN EMERGENCY SITUATION**

---

## Introduction:

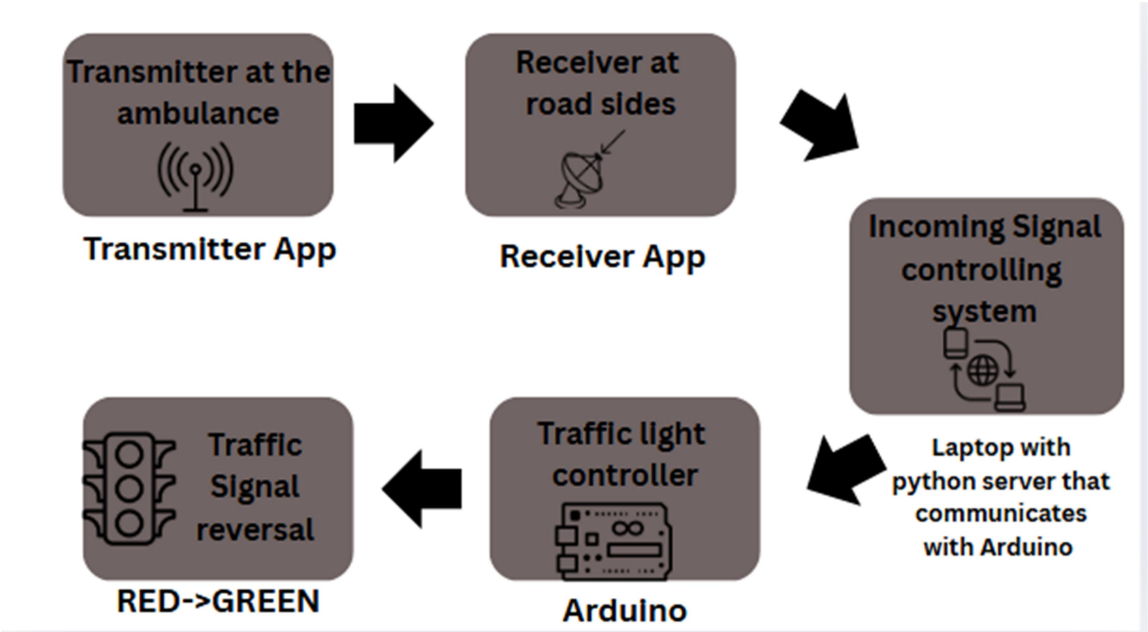
Traffic congestion and emergency response delays have become increasingly critical in today's urban transportation systems. One of the major challenges is ensuring that emergency vehicles such as ambulances and fire trucks can navigate through traffic smoothly and reach their destinations on time. In many instances, even a few seconds of delay can cost lives. Addressing this issue, the project titled "AUTOMATED TRAFFIC SIGNAL CONTROLLED IN EMERGENCY SITUATION" presents a solution to prioritize emergency vehicles at traffic junctions by automating the traffic signal switching based on emergency conditions. The system integrates wireless communication via Android apps with embedded control using an Arduino board. The transmitter app from an emergency vehicle sends a signal which is detected by the receiver app at the traffic junction, prompting the Arduino to temporarily override the current light sequence to give a green signal for the emergency path. This project combines simplicity, affordability, and real-time automation, making it a practical and scalable solution for modern traffic systems.

---

**Proposed Work & Block Diagram:**

The project comprises two major components: an Android-based transmitter and receiver system and an Arduino-controlled traffic light mechanism. The **transmitter app**, installed in an emergency vehicle, activates Bluetooth and transmits a signal when an emergency button is pressed. The **receiver app**, situated at the traffic signal post, continuously scans for nearby devices. When it detects the emergency signal from the transmitter (within a defined proximity), it communicates with the Arduino via python server built across the mobile and the laptop, that feeds input to arduino. Upon receiving this signal, the Arduino checks the current signal state. If the red light or yellow light is on, it first blinks a white LED for 5 seconds to indicate an emergency. Then it switches the signal to green, allowing the emergency vehicle to pass.

**Block Diagram**:




:

## 1. Transmitter at the Ambulance (Transmitter App)

- The **Transmitter App** is installed on the **ambulance driver's Android phone**.
- **Function:**
  1. **Scans for available traffic signal receivers** using Bluetooth.
  2. Displays a **list of available signals** (i.e., receiver devices placed at road junctions).
  3. The driver **selects the closest signal** on the screen.
  4. The app **connects via Bluetooth** to the selected receiver.
  5. Once connected, it **transmits an emergency signal** (e.g., text "1" or "EMG") to notify the receiver app that an ambulance is approaching.

---

## 2. Receiver at Road Sides (Receiver App)

- This app runs on Android devices placed at **traffic signal junctions**.
- **Function:**
  1. Waits for incoming **Bluetooth connections** using the **Bluetooth Server component**.
  2. On receiving a connection and **emergency signal from the transmitter**, it:
     - **Measures RSSI (signal strength)** to ensure the ambulance is within range (e.g., ≤ 20 meters).
     - If valid, it **sends data** to the central controlling system.

---

### 3. Incoming Signal Controlling System (Laptop with Python Server)

- Acts as a **central hub** to receive signals from **multiple receiver apps** placed at different roads leading to a junction.
- **Function:**
    1. Connected to receiver apps over **Wi-Fi/Bluetooth/USB OTG**.
    2. On receiving an emergency signal, it:
        - **Validates the message** and the source.
        - Sends a command (like `"1"`) over **USB Serial** to the **Arduino controller**.
    3. It helps **prioritize which side's signal to turn green** depending on the closest ambulance.

---

### 4. Arduino (Traffic Light Controller)

- This Arduino board is connected to **LEDs representing traffic lights**.
- **Function:**
    1. Waits for incoming data from the **laptop's Python server** via serial USB.
    2. If it receives `"1"`, indicating an emergency:
        - It checks the **current light status**.
        - If **red**, it switches to **green** and optionally turns on a **white emergency LED**.
        - Holds green for a **predefined duration** (e.g., 10 seconds) to allow the ambulance to pass.

---

### 5. Traffic Signal Reversal (RED → GREEN)

- Once the signal from Arduino is processed:
    - The traffic light on the ambulance side is **turned green**.
    - All other directions remain red.
- After the emergency is handled:
    - The system **returns to normal operation** or resets based on a timer.

Now that the signal has turned green, the **ambulance can pass through without delay**, reducing emergency response time.

---

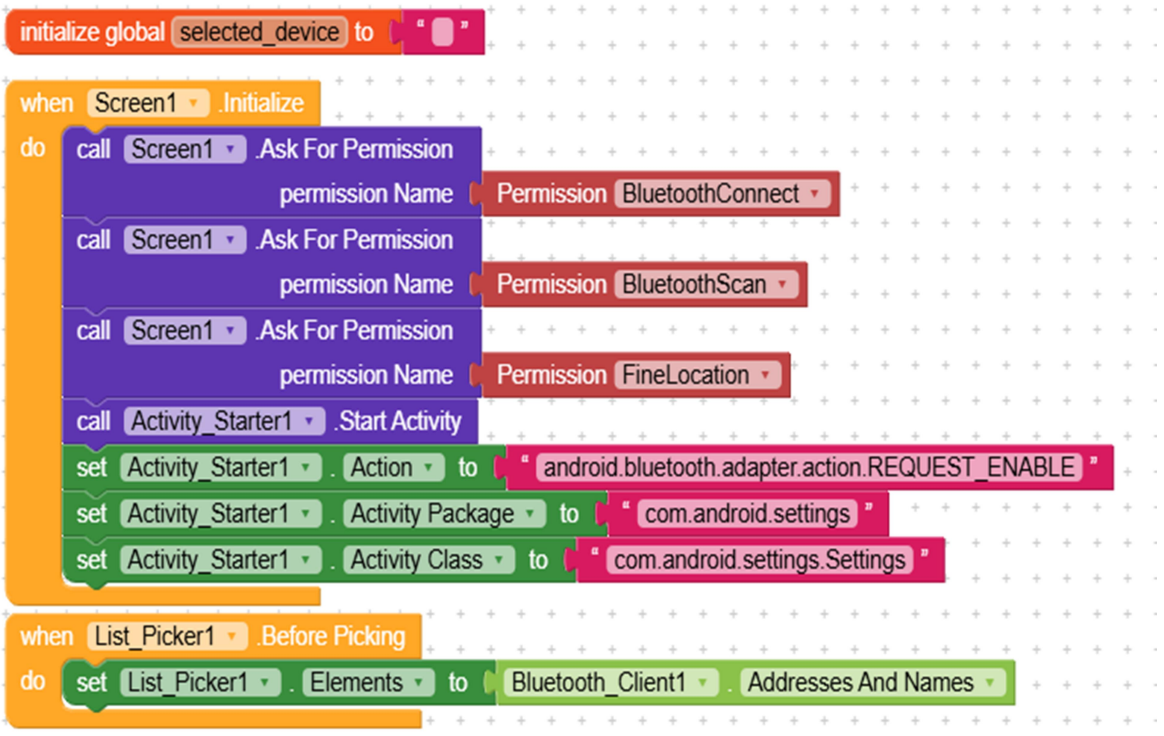
### Application and Python Server Details:

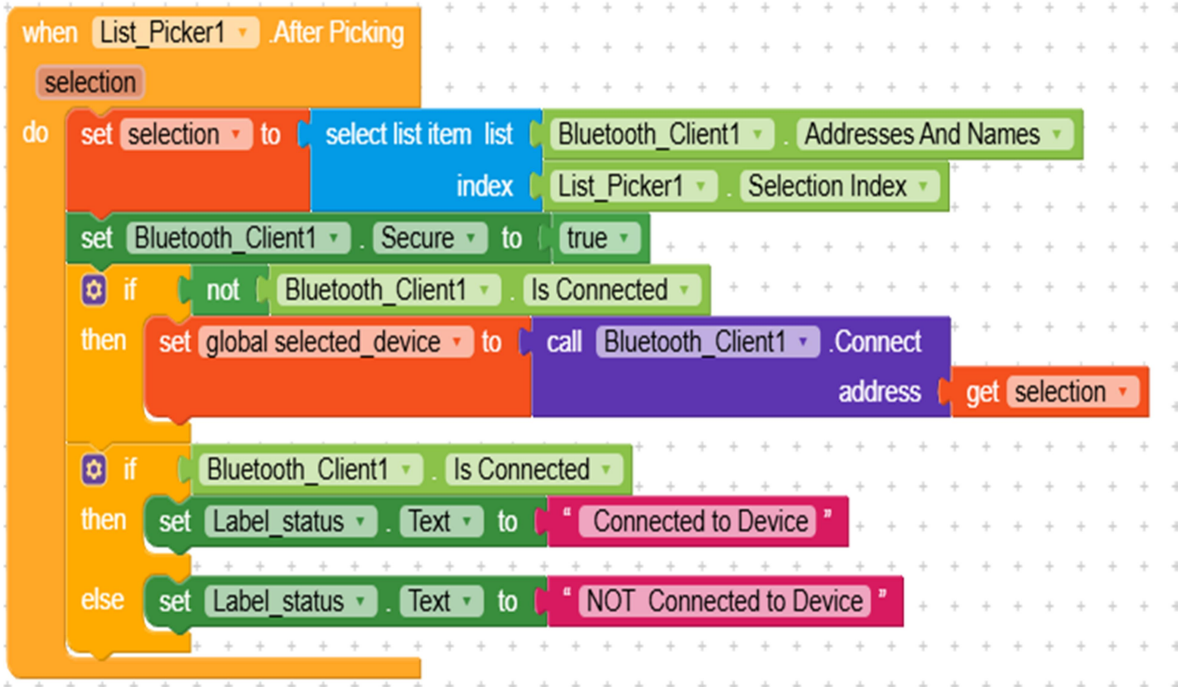
The system includes two Android applications developed on **Kodular**, a visual programming platform that allows the creation of apps using drag-and-drop blocks.

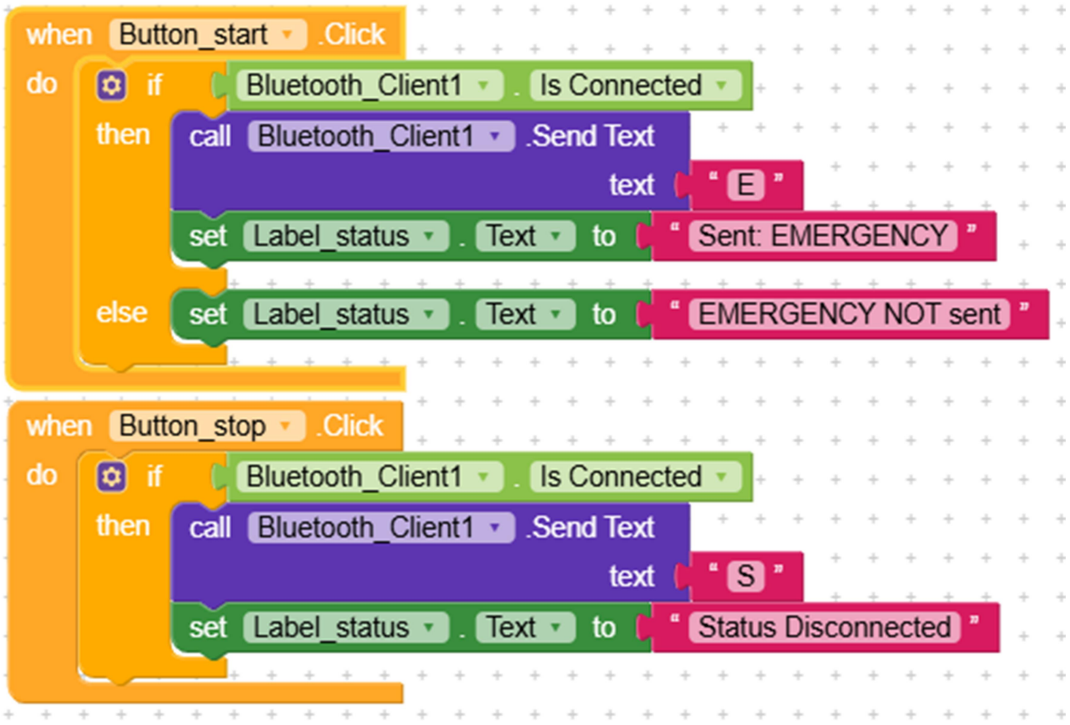
### Transmitter App:

- Purpose: Installed in emergency vehicles to send Bluetooth signals.
- Function: On button press, Bluetooth is enabled and a message is transmitted continuously.
- Built With: Kodular blocks using Bluetooth Client component.
- Key Feature: Automatically connects to the receiver app using device address.

✓ **Block Style Code:**



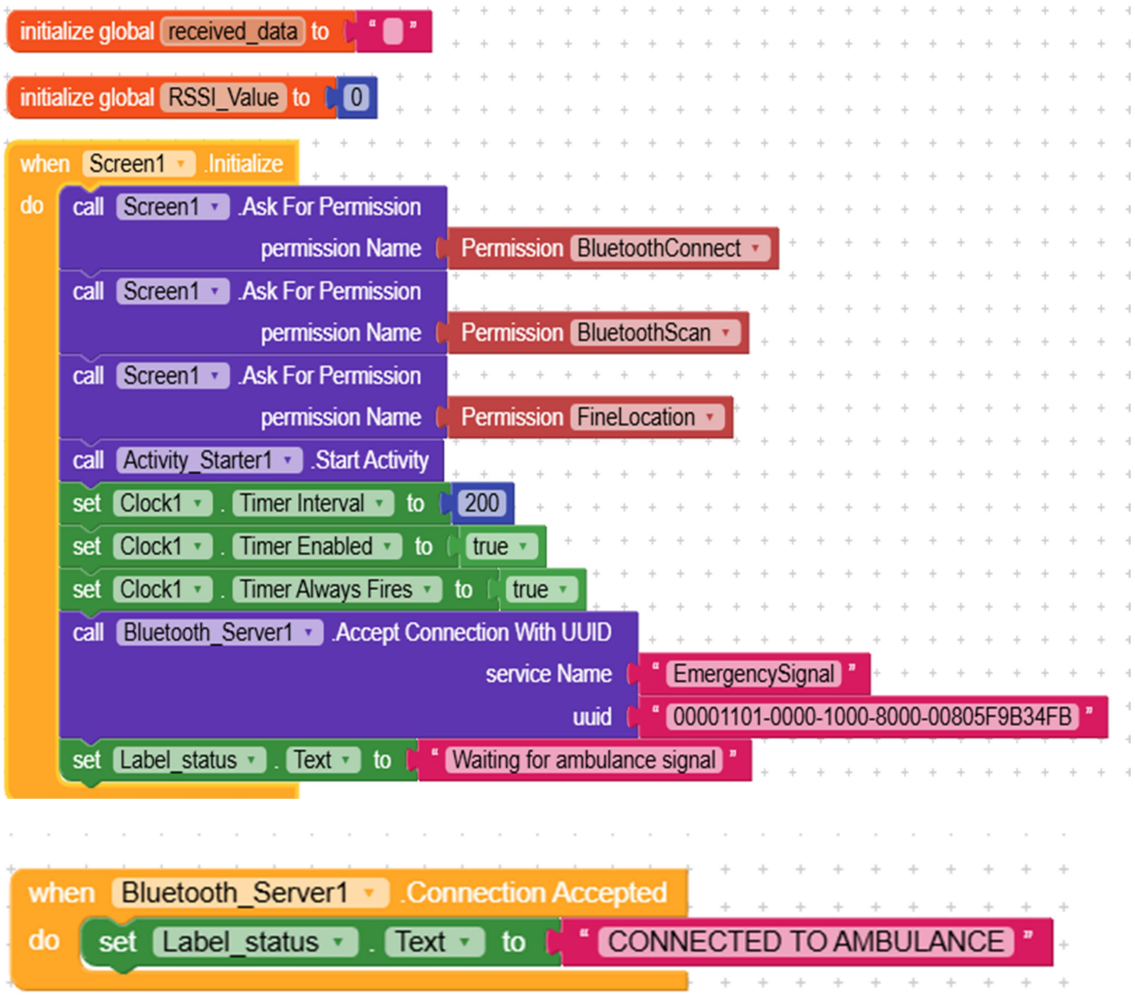


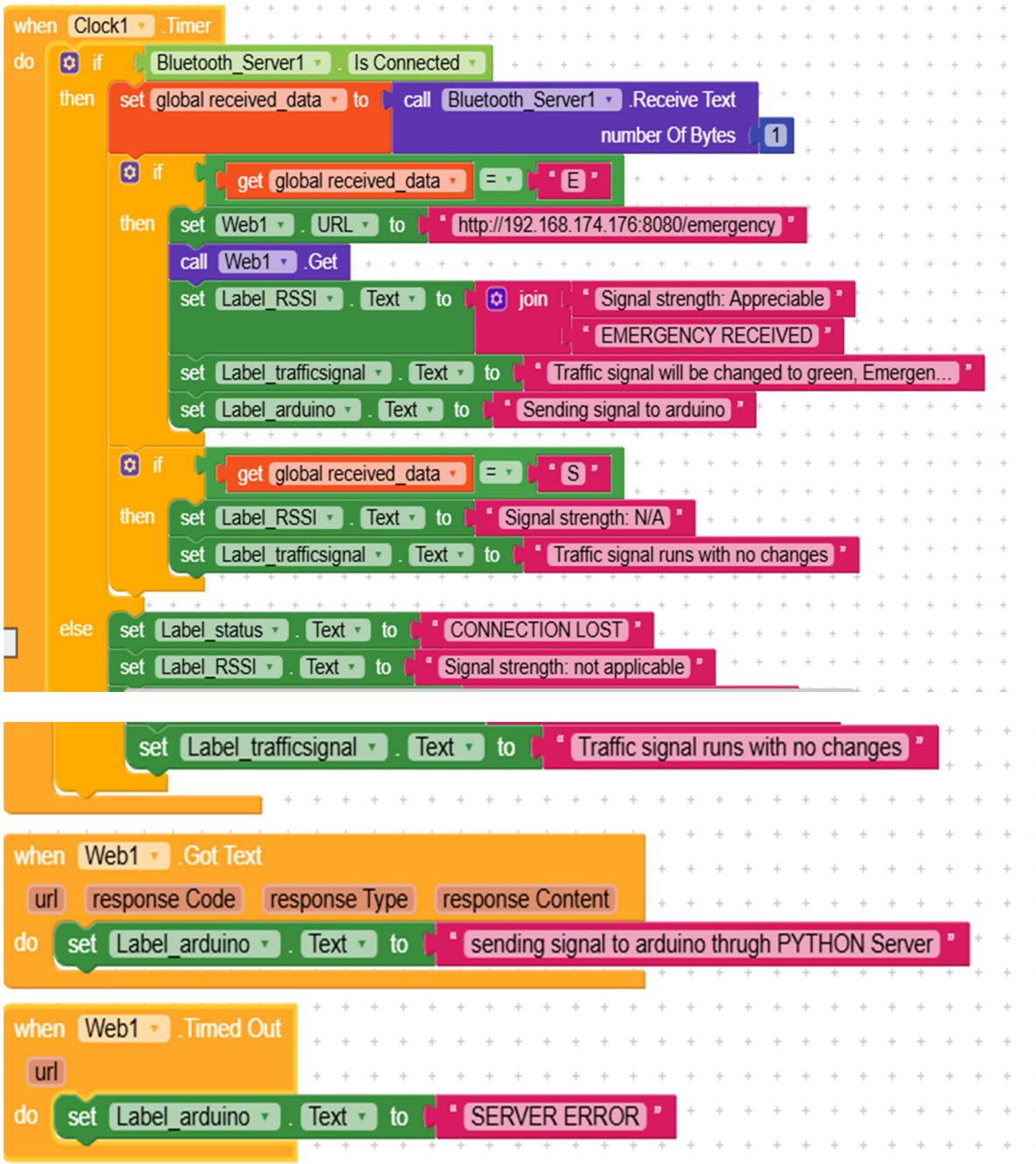

✓ **Block code explanation**:

- **Bluetooth Initialization**:
  When the app starts, it checks if Bluetooth is enabled. If not, it asks the user to turn it on.
- **Connect to Receiver**:
  When the emergency button is pressed, the app connects to the receiver device (either automatically or using a list of paired devices).
- **Send Emergency Signal**:
  Once connected, a timer starts sending the text **"1"** repeatedly via Bluetooth every second.
- **Continuous Transmission**:
  The signal **"1"** is sent continuously until the emergency is stopped or the app is closed.
- **Purpose**:
  The receiver (traffic signal controller) reads this signal and gives priority to the emergency vehicle by changing the traffic light.

**Receiver App:**

- Purpose: Installed at the traffic control unit to receive emergency signals.
- Function: Uses Bluetooth Server to detect incoming signals and sends a command to Arduino via Serial OTG.
- Built With: Kodular blocks using Bluetooth Server and web.
- Key Feature: Monitors signal strength to ensure the vehicle is within range and sends a serial command if emergency is detected.






✓ **Block Code Explanation**:

- The receiver continuously listens using the Bluetooth Server component.
- When data is received from the transmitter app, it processes the message.
- If the data is "1," it writes this to Arduino using the Serial OTG extension.
- Arduino receives the character via its Serial monitor and triggers the logic accordingly.

**Python Server Code:**

**Code Overview:**

This Python script sets up a local HTTP server on a laptop connected to an Arduino via USB. The server listens for emergency requests from the receiver app, which runs on an Android device. Both devices are connected to the same Wi-Fi network, enabling local communication. When the receiver app detects an emergency (e.g., from a transmitter app in an ambulance), it sends a request to the server's /emergency endpoint. The server then sends a '1' over serial to the Arduino. Upon receiving this, the Arduino switches the traffic light to green, allowing emergency vehicle passage. This system ensures quick response without relying on internet connectivity. It provides an efficient and low-latency solution for smart traffic management in emergency situations.

**Code**:

```
import serial

from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer


# Setup serial connection to Arduino (adjust the port as needed)

arduino = serial.Serial('COM3', 9600)  # Windows users: COMx, Linux: /dev/ttyUSB0


# Define the HTTP request handler

class EmergencyHandler(BaseHTTPRequestHandler):

  def do_GET(self):

   print("Received request: " + self.path)  # <-- This will print every request path

   if self.path == '/emergency':

      print("Emergency request received. Sending '1' to Arduino...")

      arduino.write(b'1')

      self.send_response(200)

      self.send_header('Content-type', 'text/plain')
```

```
        self.end_headers()
        self.wfile.write(b'Emergency Signal Sent')
    else:
        self.send_response(404)
        self.end_headers()



# Define and start the HTTP server
def run():
    # IP is 0.0.0.0 (accepts all connections on the local network), port is 8080
    httpd = HTTPServer(('0.0.0.0', 8080), EmergencyHandler)
    print("Server started on port 8080...")
    httpd.serve_forever()


# Start the server
if __name__ == '__main__':
     run()
```

---

**Arduino Code:**

**Code Overview:**

The Arduino is programmed to:

- Cycle through traffic lights in a loop: Green (20s) → Yellow (20s) → Red (20s)
- Monitor serial input for the value '1'
- If '1' is received and the red signal is active:
    - Blink a white LED for 5 seconds

    - Then turn off the red and turn on green
- Return to normal cycle after emergency

**MATLAB code:**

```
#define GREEN_PIN 2
#define YELLOW_PIN 3
#define RED_PIN 4
#define WHITE_PIN 5

unsigned long previousMillis = 0;
const unsigned long interval = 20000; // 20 seconds per signal

int currentState = 0; // 0 = Green, 1 = Yellow, 2 = Red
bool emergencyTriggered = false;
bool emergencyHandled = false;

void setup() {
  Serial.begin(9600);

  // Setup pin modes
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(YELLOW_PIN, OUTPUT);
  pinMode(RED_PIN, OUTPUT);
  pinMode(WHITE_PIN, OUTPUT);

  // Ensure all are LOW initially
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(YELLOW_PIN, LOW);
  digitalWrite(RED_PIN, LOW);
  digitalWrite(WHITE_PIN, LOW);

  // Force start from GREEN
  currentState = 0;
  updateLights(currentState);
  previousMillis = millis();
}

void loop() {
  // Check for emergency signal
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    input.trim();

    if (input == "1" && !emergencyTriggered && (currentState == 1 || currentState == 2))
{
      emergencyTriggered = true;
      emergencyHandled = false;
```

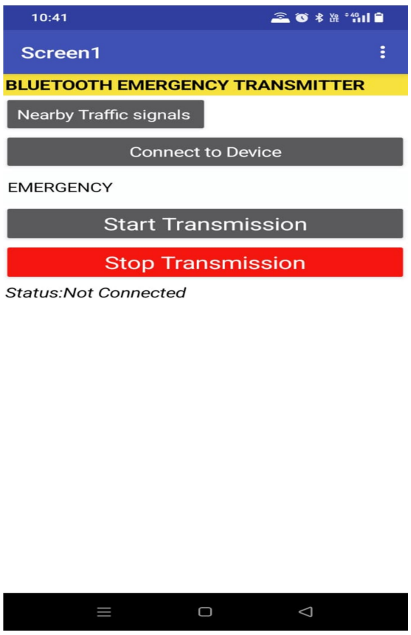
```
    }
  }

  unsigned long currentMillis = millis();

  // Emergency routine
  if (emergencyTriggered && !emergencyHandled) {
    blinkWhiteLED(5000); // Blink white for 5s
    currentState = 0; // Switch to Green
    updateLights(currentState);
    previousMillis = millis();
    emergencyHandled = true;
    emergencyTriggered = false;
  }
  // Normal cycle
  else if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    currentState = (currentState + 1) % 3;
    updateLights(currentState);
  }
}

void updateLights(int state) {
  digitalWrite(GREEN_PIN, LOW);
  digitalWrite(YELLOW_PIN, LOW);
  digitalWrite(RED_PIN, LOW);
  digitalWrite(WHITE_PIN, LOW);

  if (state == 0) digitalWrite(GREEN_PIN, HIGH);
  else if (state == 1) digitalWrite(YELLOW_PIN, HIGH);
  else if (state == 2) digitalWrite(RED_PIN, HIGH);
}

void blinkWhiteLED(unsigned long duration) {
  unsigned long startTime = millis();
  while (millis() - startTime < duration) {
    digitalWrite(WHITE_PIN, HIGH);
    delay(250);
    digitalWrite(WHITE_PIN, LOW);
    delay(250);
  }
}
```

---

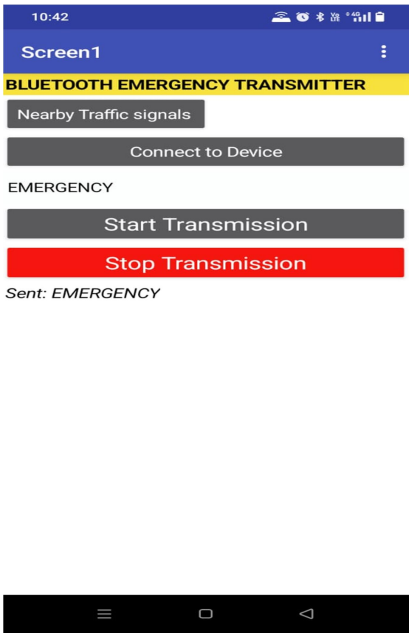
## Project Output:

## Transmitter app

### Before transmitting



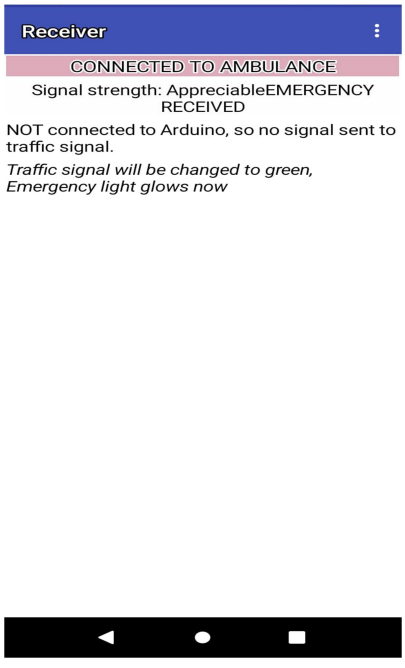

### After transmitting




## Receiver app

### Before Receiving


### After Receiving



**Python Server**

**Before Receiving**

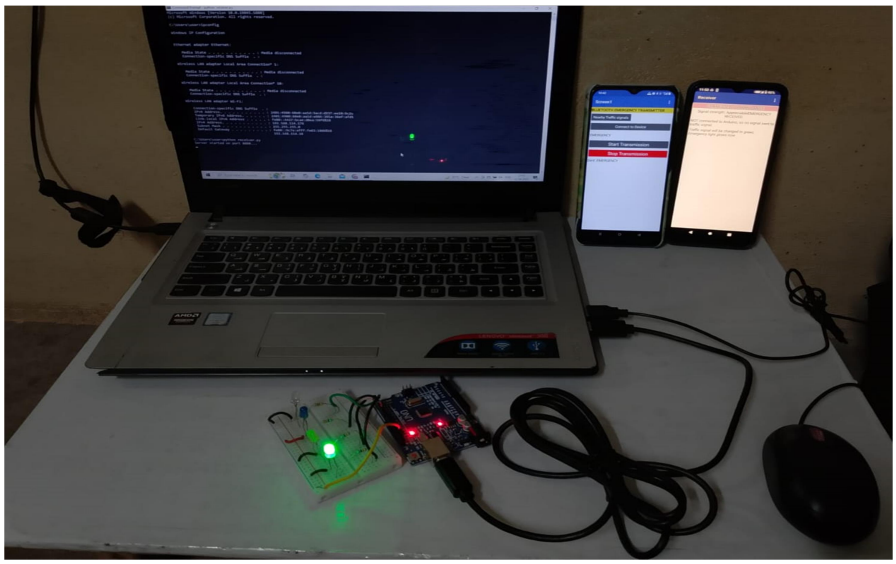
```
C:\Users\user>python receiver.py
Server started on port 8080...
```

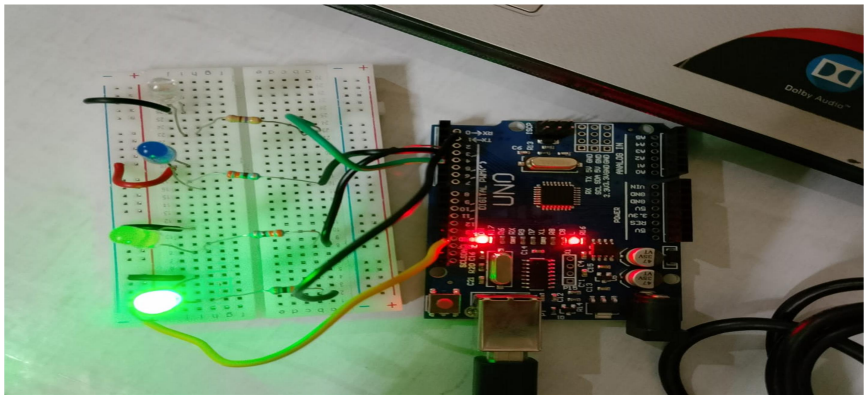
**After Receiving**

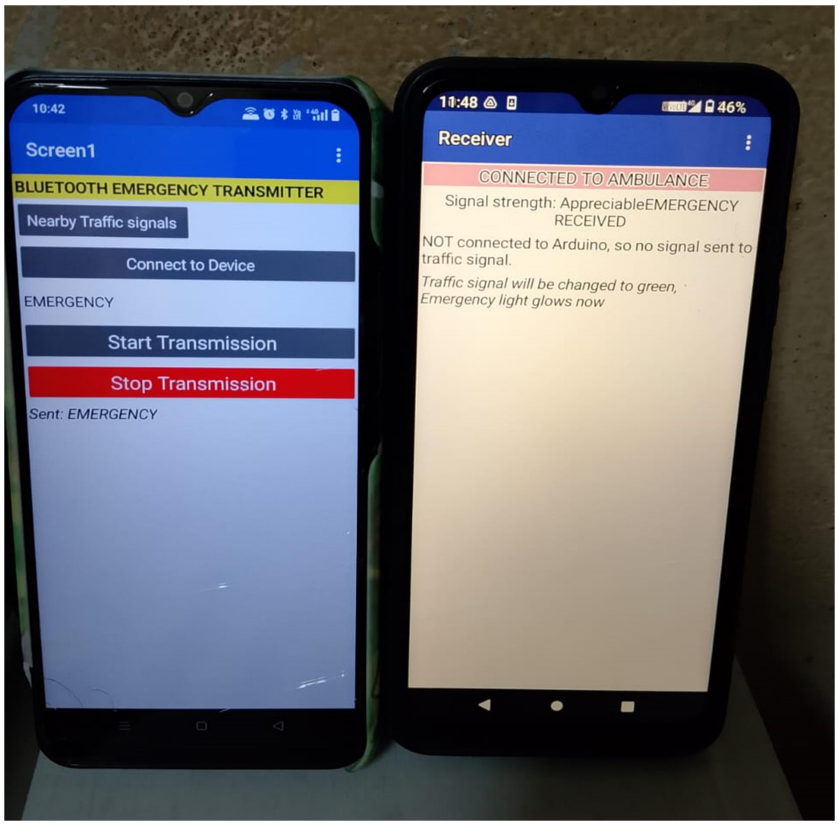
```
C:\Users\user>python receiver.py
Server started on port 8080...
Received request: /emergency
Emergency request received. Sending '1' to Arduino...
192.168.174.176 - - [22/Apr/2025 19:11:34] "GET /emergency HTTP/1.1" 200 -
Received request: /favicon.ico
192.168.174.176 - - [22/Apr/2025 19:11:34] "GET /favicon.ico HTTP/1.1" 404 -
```

---

**Snapshot of project:**








---

**Conclusion:**

The project successfully demonstrates how smart automation can optimize traffic signal operations, especially during emergency scenarios. By bridging mobile communication with embedded systems, it ensures emergency vehicles get uninterrupted passage through traffic lights without manual intervention. This not only reduces response time but also enhances safety for all road users. The system is designed to be low-cost, scalable, and implementable in real-world traffic environments with minimal infrastructure change. It represents a significant step towards smarter cities and intelligent transportation systems.

With further optimization, such systems could be integrated with real-time GPS tracking, camera-based detection, or even centralized traffic control networks for broader city-wide applications.

---

**Application:**

This system can be adopted in urban areas where emergency delays are frequent due to dense traffic. Ambulances, fire engines, or police vehicles equipped with the transmitter app can automatically influence traffic signals in their favor, reducing the need for human intervention or escort. Additionally, the solution can be extended to VIP convoys, disaster response vehicles, or even public transport prioritization. The use of simple smartphones and low-cost Arduino boards makes this an affordable yet effective option for developing cities with limited smart infrastructure. Moreover, the wireless communication model allows for further integration with IoT ecosystems, thereby enhancing the scope of traffic automation and public safety.

---

**Future Improvements:**

- Integrate **GPS and Google Maps** API to track and pre-clear routes for emergency vehicles.
- Use **Camera + Image Processing (OpenCV)** for automatic ambulance detection without transmitter app.
- Add **cloud data logging** for emergency analytics.
- Employ **voice command or siren-based detection** via audio DSP in noisy environments.
- Build a **web dashboard** to monitor traffic signal overrides in real-time for admin use.

---

**References:**


- Kodular documentation and community forums
- Arduino.cc official website
- Serial OTG Extension by Juan Antonio
- MIT App Inventor Bluetooth communication tutorials
- Research articles on emergency traffic signal management