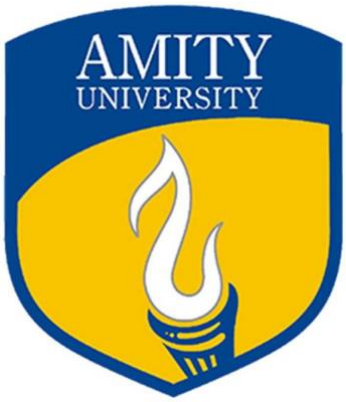
B.TECH. (2020-24)
Artificial Intelligence

INTERNET OF THINGS: SENSING AND ACTUATOR DEVICES [CSE449]

OPEN-ENDED PROJECT
SMART TRAFFIC MANAGEMENT SYSTEM



Submitted To
Mr. Jitendra Singh Jadon

Submitted By

JAITN	**A023119821028**	**6AI 1**
NIKHIL	**A0231198210**	**6AI 1**
DIVIIT	**A0231198210**	**6AI 1**

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH NOIDA (U.P)

AIM

To make a Smart Traffic Management System focused on waiting time optimization.

REQUIREMENTS

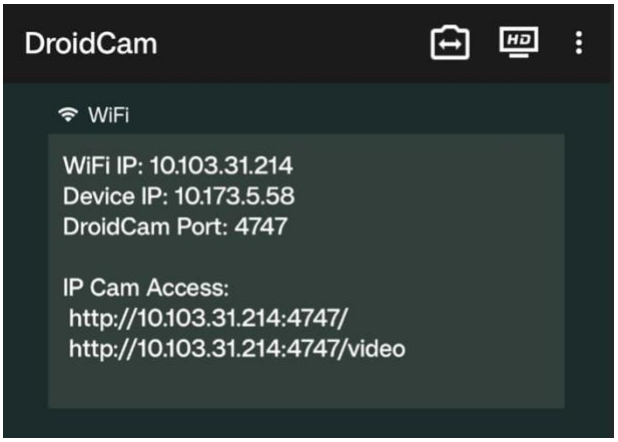
- Hardware
- Arduino Uno
- LEDs
- Jumper wires
- Resistors
- Bread Board
- Laptop (Processor)
- MicroUSB cable
- Phone Camera Software(Droid Cam)
- Arduino IDE
- Python
- Yolov5

THEORY

Phone Camera:

To cut the development cost of the project we used Phone Cameras instead of traditional ESP32 Cams, The phone cameras acted as the input to capture the amount of traffic on both lanes. The Phone was connected to the Laptop (Processor) using WIFI and was providing real-time video input feed.

Droid Cam:



Screenshot of droid cam interface

Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328P. At its core, the Arduino Uno operates on the principle of a simplified hardware and software environment that facilitates easy prototyping and development of interactive electronic projects. The board features a range of digital and analog input/output pins that can be easily programmed using the Arduino Integrated Development Environment (IDE), which is built on a simplified version of C++ language. This simplicity makes it accessible even to beginners in electronics and programming. With its open-source nature, the Arduino Uno has fostered a large and vibrant community of developers, makers, and enthusiasts who share their projects, knowledge, and code, contributing to its widespread adoption across various domains including education, hobbyist projects, and professional prototyping.

The versatility of the Arduino Uno lies in its ability to interface with a wide array of sensors, actuators, and other electronic components, enabling the creation of diverse projects ranging from simple blinking LED lights to complex robotic systems. Its modular design also allows for easy expansion through the use of shields, which are add-on boards that provide additional functionalities such as wireless communication, motor control, or sensor integration. This flexibility, coupled with its affordability and ease of use, has made the Arduino Uno a popular choice for both individuals and institutions seeking to explore the realms of electronics, programming, and physical computing. As a platform for innovation and experimentation, the Arduino Uno continues to inspire creativity and empower individuals to turn their ideas into reality.
The Arduino Uno is comprised of several key components, each playing a crucial role in its functionality:

Microcontroller: At the heart of the Arduino Uno lies a microcontroller unit (MCU). The Uno typically uses the Atmel ATmega328P microcontroller, which is responsible for executing the instructions and controlling the input/output operations of the board.

Digital Input/Output (I/O) Pins: The Uno features a set of digital pins (14 in total), which can be configured as either inputs or outputs. These pins allow the Arduino to interface with digital sensors, control digital actuators such as LEDs or motors, and communicate with other digital devices.

Analog Input Pins: Alongside the digital pins, the Uno also includes a set of analog input pins (6 in total). These pins can read analog voltage values from sensors, potentiometers, or other analog devices, converting them into digital values that the microcontroller can process.

Power Supply: The Arduino Uno can be powered via various sources, including USB connection, a barrel jack, or an external power supply connected to the Vin pin. It supports a wide range of input voltages (typically 7-12V), which are regulated down to the 5V required by the microcontroller and other components.

Reset Button: A reset button allows users to restart the execution of the Arduino sketch (program) running on the board. Pressing the reset button momentarily resets the microcontroller, restarting the sketch from the beginning.

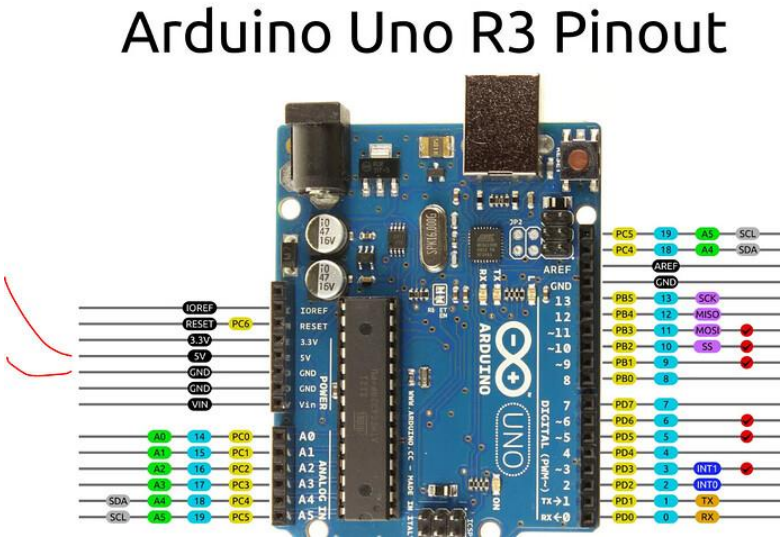
Crystal Oscillator: The Arduino Uno includes a crystal oscillator (typically 16 MHz) that provides the clock signal for the microcontroller, enabling precise timing of operations and facilitating communication with external devices.

Voltage Regulator: To ensure stable operation, the Uno incorporates a voltage regulator that regulates the input voltage to provide a steady 5V output for powering the microcontroller and other components on the board.

USB Interface: The Arduino Uno features a USB interface that allows it to connect to a computer for programming and serial communication. This interface also provides power to the board when connected to a computer via USB.

LEDs: The Uno includes several built-in LEDs for visual feedback. These include a power LED indicating when the board is powered on, a pin 13 LED that can be controlled programmatically, and LEDs associated with the USB communication and serial communication with the computer.

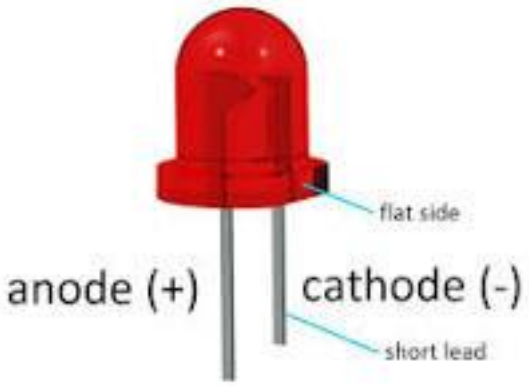
Arduino Uno in our project worked as a microcontroller that was used to control the current flow of LEDs to indicate the real-time output.
It was receiving its input in the form of integers 0 1 or 2 from the laptop through MicroUSB Cable.
The input indicates:
0: The system is turned off (Both sides will turn on the Red Light)
1: Side A can move (Side A will have a green light and Side B will have a red light)
2: Side B can move (Side B will have a green light and Side A will have a red light)



Arduino Uno Board

LED

In an LED (Light Emitting Diode), the positive and negative terminals play crucial roles in determining its functionality. The positive terminal, also known as the anode, is the electrode through which current flows into the LED. Conversely, the negative terminal, or cathode, is the electrode through which current exits the LED. When a voltage is applied across the LED with the positive terminal connected to a higher potential than the negative terminal, electrons flow from the negative terminal to the positive terminal through the LED's semiconductor material. This flow of electrons across the semiconductor junction results in the emission of photons, producing light.



LED

In our Project, LEDs are used to indicate the traffic lights. When the Green Light is turned on Side A then automatically the Red light will be turned on Side B and vice versa, the switching of the LED is controlled by the Arduino Uno based on the input given from the Laptop.

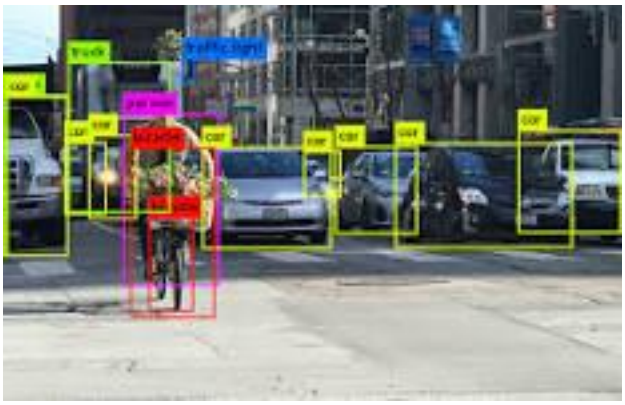
Breadboard

A breadboard is a fundamental tool in electronics prototyping, providing a platform for quickly building and testing circuits without the need for soldering. Its design consists of a grid of holes connected by metal strips beneath the surface. These holes allow components to be inserted and connected to create circuits easily. The breadboard typically features two sets of horizontal rows, often labelled as "rows" and "columns," with each row containing five connected holes. The rows on the breadboard are usually connected internally in a specific pattern, with the columns typically unconnected.

The breadboard's functionality relies on the concept of electrical connectivity. When a component, such as a resistor or an LED, is inserted into a hole on the breadboard, its leads make contact with the metal strips beneath the surface, establishing an electrical connection. By strategically placing components and connecting them using jumper wires, designers can create complex circuits for experimentation and testing. The breadboard's versatility lies in its ability to accommodate various types of electronic components, including integrated circuits, sensors, and passive elements, allowing for rapid prototyping and iterative design. Moreover, the temporary nature of connections on a breadboard enables quick modifications and troubleshooting, making it an indispensable tool for electronics enthusiasts, hobbyists, and professionals alike.

Laptop:

The laptop acted as the processing Unit, where we received input from Camera through Wi-Fi read using OpenCV in Python. And then we used YoloV5 to get the count of cars on both lanes.



object detection using Yolo v5

More about Yolo:

YOLOv5 is a powerful object detection model that belongs to the YOLO (You Only Look Once) family. Here's a breakdown of the theory behind it:

Single-Stage vs. Two-Stage Detection:

• Unlike some object detectors that use a two-stage approach (region proposal and classification), YOLOv5 is a single-stage detector. This means it predicts bounding boxes and object class probabilities in one go, making it faster.

Core Architecture:

YOLOv5 is built on a three-part backbone:

1. Backbone (Feature Extraction): This is a Convolutional Neural Network (CNN) like CSPDarknet, which extracts important features from the input image.
2. Neck (Feature Fusion): This stage, often using PANet, combines the detailed features from the early layers of the backbone with the semantic features from later layers. This fusion allows for detection of objects at various scales within the image.
3. Head (Detection): The YOLO layer takes the fused features and predicts bounding boxes and class probabilities for each object it detects.

Key Points:

• YOLOv5 comes in different versions (s, m, l, and x) offering a trade-off between speed and accuracy.
• It leverages techniques like non-max suppression to remove duplicate detections.

In our project,

• Here, we are using yolo to identify the vehicles mainly cars
• The yolov5 will work on real time detection using camera systems in the traffic signals
• The detection result produced by yolo is used to find the number of cars for further computations in the algorithm development process.
• We have used both smaller yolov5s and bigger yolov5x models and noticed there is no significant change in the test results

Install Requirements:

The First thing we need to install is an Arduino IDE and CH340 Driver to connect the Arduino Uno to the laptop. Then We also required to install YoloV5 for car detection.

1) Setup Camera:

Now we need the laptop and Phone to be on the same network so that OpenCV can read the camera feed of the phone using the IP Host address.

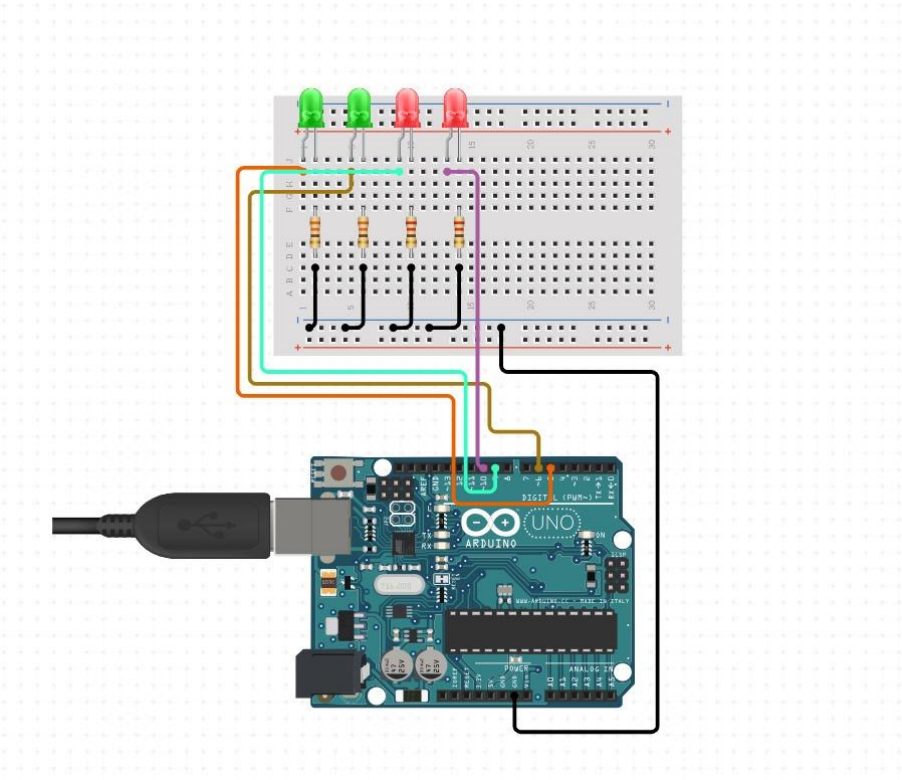
2) Model Car Detection:

Now we use YoloV5 to detect the number of cars and feed it to the algorithm to determine which side should get the green light then the corresponding output is fed to Arduino as input using MicroUSB Cable.

3) Arduino Setup:

Now we write the code on Arduino Uno to set up the GPIO Pins for the LEDS and MicroUSB Port to read the incoming Data from the laptop and then provide the necessary signal to the LEDS.

CIRCUIT DIAGRAM:



Circuit Diagram showing connections

CODE:

a) Arduino Uno IDE(C++):

```
#include <SoftwareSerial.h>



const int redPinA = 10;
const int greenPinA = 8;
const int redPinB = 12;
const int greenPinB = 6;
void setup() {
  // put your setup code here, to run once:
  pinMode(redPinA, OUTPUT);
  pinMode(greenPinA, OUTPUT);
  pinMode(redPinB, OUTPUT);
```

```
  pinMode(greenPinB, OUTPUT);
  Serial.begin(9600);

}

void loop() {
  int response;
  Read response from Raspberry Pi
  if(Serial.available()>0){
     int response = Serial.parseInt();
     if (response == 1){
    digitalWrite(redPinA, LOW);
    digitalWrite(greenPinA, HIGH);
    digitalWrite(redPinB, HIGH);
    digitalWrite(greenPinB, LOW);
    delay(6000); // Wait for 3 seconds

  }
  if(response==2){
    // Yellow light
    digitalWrite(redPinA, HIGH);
    digitalWrite(greenPinA, LOW);
    digitalWrite(redPinB, LOW);
    digitalWrite(greenPinB, HIGH);
    delay(6000); // Wait for 3 seconds
  }
if(response==-1){
    // Green light
    digitalWrite(redPinA, HIGH);
    digitalWrite(greenPinA, LOW);
    digitalWrite(redPinB, HIGH);
    digitalWrite(greenPinB, LOW);
    delay(1000);

  }
  // Process response as needed
  // put your main code here, to run repeatedly:
  // Red light

  //  Wait for 3 seconds
```

```
  // Pedestrian signal
  digitalWrite(redPin, LOW);
  digitalWrite(yellowPin, LOW);
  digitalWrite(greenPin, LOW);
  digitalWrite(pedestrianPin, HIGH);
  delay(1000); // Wait for 3 seconds


 // digitalWrite(redPinA, HIGH);
 // digitalWrite(redPinB, HIGH);
 // digitalWrite(greenPinA, HIGH);
 // digitalWrite(greenPinB, HIGH);


}
```

b) YOLO v5 Algorithm(Python):

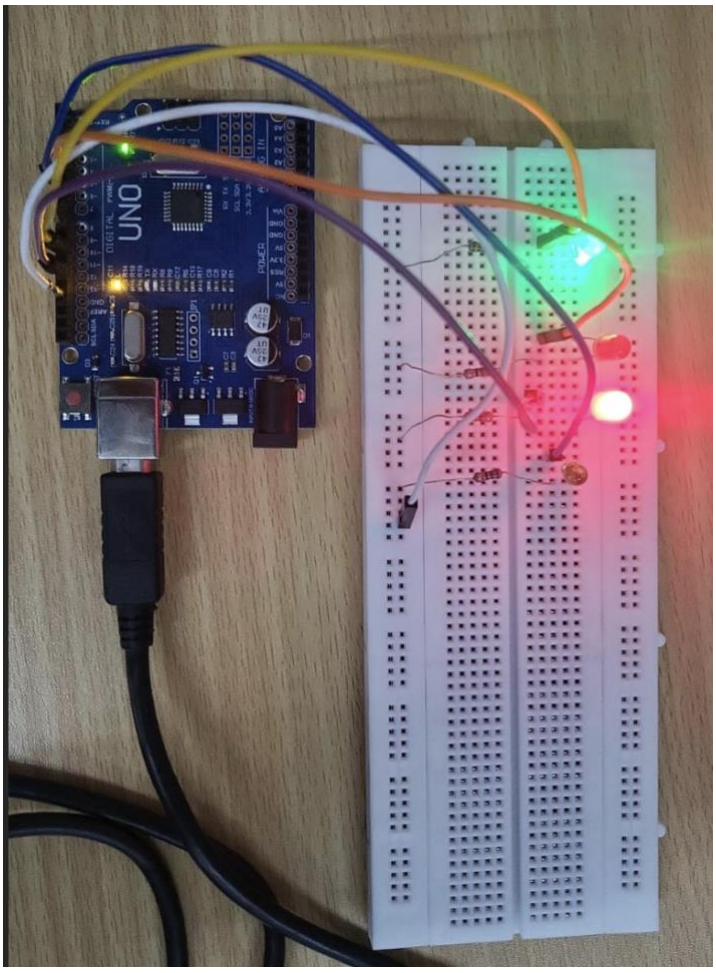
```
import torch
from IPython.display import display, clear_output
import cv2
from PIL import Image
import time
import serial


model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
model.conf = 0.2

# ser = serial.Serial('COM4', 9600)  # Adjust port and baud rate as needed
ser = serial.Serial('/dev/cu.usbserial-10', 9600)

cl = "bottles"
# define a video capture object
# phn1 = "http://192.168.24.127:4747/video"
# phn2 = "http://192.168.24.107:4747/video"
phn1 = "http://10.103.121.121:4747/video"
phn2 = "http://10.103.223.49:4747/video"
# Create a VideoCapture objectqqqqq
```

```
cap1 = cv2.VideoCapture(phn1)
cap2 = cv2.VideoCapture(phn2)

# Check if the camera is opened successfully
if not cap1.isOpened():
    print("Error: Could not open camera1.")
    exit()
if not cap2.isOpened():
    print("Error: Could not open camera2.")
    exit()
counterA=0
counterB=0
# Read and display frames from the camera
st = time.time()
while True:

    ret1, frame1 = cap1.read()
    ret2, frame2 = cap2.read()
    if not ret1:
        print("Error: Failed to receive frame1.")
        break
    if not ret2:
        print("Error: Failed to receive frame2.")
        break

    if time.time()-st >= 5:

        result1 = model(frame1)
        result2 = model(frame2)

        # Check if "person" is in detection results
        if cl in str(result1):
            traffic1 = str(result1)[(str(result1).index(cl)-2)]
        else:
            traffic1 = '0'  # No person detected

        if cl in str(result2):
            traffic2 = str(result2)[(str(result2).index(cl)-2)]
        else:
            traffic2 = '0'  # No person detected
```

```
traffic = [traffic1, traffic2]
print(traffic)
if(max(traffic)=="0"):
    command = f"{-1}\n"
    print(command)
    ser.write(command.encode())
    counterA=0
    counterB=0

elif traffic.index(max(traffic)) == 0 :
    if(counterA>3) and traffic2!="0":
        command = f"{2}\n"
        print(command)
        ser.write(command.encode())
        counterB+=1
        counterA=0
    else:
        command = f"{1}\n"
        print(command)
        ser.write(command.encode())
        counterB=0
        counterA+=1

elif traffic.index(max(traffic)) == 1:
    if(counterB>3) and traffic1!="0":
        command = f"{1}\n"
        print(command)
        ser.write(command.encode())
        counterA+=1
        counterB=0
    else:
        command = f"{2}\n"
        print(command)
        ser.write(command.encode())
        counterA=0
        counterB+=1

st = time.time()
```

```
    # Display the frame
    cv2.imshow('DroidCam Feed1', frame1)
    cv2.imshow('DroidCam Feed2', frame2)

    # Break the loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the VideoCapture object and close all OpenCV windows
cap1.release()
cap2.release()
cv2.destroyAllWindows()
```

OUTPUT



Project circuit

RESULT

We have successfully made a smart traffic management system using Yolo v5 and Arduino Uno with a waiting time optimization algorithm while capturing images via mobile phones.

PRECAUTIONS:

- Properly check the connections.
- Make sure that the reset button has been pressed before uploading the code so that the download code mode can be activated on the Arduino.
- Loop() and Setup() should be error-free and properly defined.
- Make sure that you are connected to stable Wi-fi and all the devices should be connected to the same Wi-fi.
- Properly check the IP addresses.