CONESTOGA
Connect Life and Learning

EECE8040 - Winter 2024 Engineering Capstone Project

Embedded Systems Development

Project: SMART Density-Based Traffic Light Control System

Group – 2

Ravija Arora (8833262)

Harleen Kaur (8868224)

Rishabh Bhatia (8810595)

ABSTRACT

In today's world, traffic congestion is a major issue, especially during peak hours. To tackle this problem, we've developed a traffic control system that monitors vehicle numbers and adjusts traffic light timings for smoother flow. This project utilizes a Raspberry Pi + STM32 as the core system. It features two intersections with traffic lights. One intersection uses a Raspberry Pi camera for vehicle counting via object detection. At the same time, the other employs ultrasonic sensors connected to an ESP32 to count vehicles and send data to the cloud. The Raspberry Pi communicates with an STM based on real-time traffic density to modify traffic light timings, reducing congestion and ensuring better traffic flow.

Acknowledgments

We want to extend our heartfelt gratitude to Professors Sandra, Amrinder, and Ralph for their invaluable guidance and support throughout the development of our capstone project. Their mentorship has been instrumental in shaping our ideas, refining our approach, and achieving measured goals.

Additionally, we offer a special thanks to God for His blessings, guidance, and unwavering support throughout our journey.

Table Of Contents

I. Introduction

Managing traffic flow efficiently in big cities has become a paramount challenge in this present era. Current traffic control systems often fail to adapt to real-time traffic conditions, resulting in congestion and delays. Keeping these challenges in mind, this project delivers a solution that monitors the traffic density and controls the traffic flow on the road using advanced technologies.
By leveraging the integration of Raspberry Pi, STM32 microcontroller, and ESP modules, the system aims to introduce a framework for real-time traffic and adaptive signal control.

II. Proposed Solution

The proposed system comprises an intelligent traffic control system incorporating advanced hardware and optimization algorithms to control traffic fluidity. Raspberry Pi is used as the main CPU of the system. Pi-camera is utilized for vehicle detection to count the number of vehicles on the road. The captured data is transmitted to Amazon Web Services (AWS) for comprehensive analysis and storage.

ESP module interfaced with ultrasonic sensors is also deployed at adjacent cross-section to gather additional real-time traffic information. The acquired data points are seamlessly sent to AWS for a holistic view of the traffic. This facilitates the system in understanding traffic patterns and changing signal timings.

A special case has been introduced, which includes switching the traffic signal to green whenever an emergency vehicle like an ambulance approaches the cross-section. Thus, it clears the emergency vehicle's path so that it can navigate through traffic quickly and safely.

III. Technology/Key Components Used

Key components of the system include the following:

1. **Raspberry Pi 4B:** It works as the main brain of the system. It is utilized for vehicle detection through camera-based surveillance.

Fig 1.1 Raspberry-Pi

2. **STM32F411RE board:** the traffic lights at the cross-sections are controlled by the STM32 board.

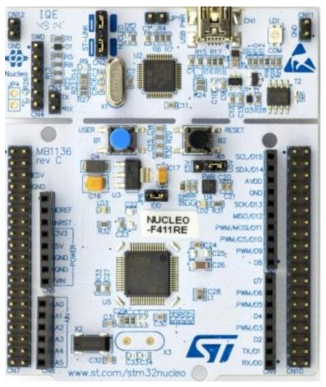


Fig 1.2 STM32F411RE Board

3. **ESP32 Board:** ESP is deployed at an alternative cross-section to sense the

number of vehicles from ultrasonic sensors and send the data to the cloud to supplement the vehicle count.

Fig.1.3 ESP32 Board

4. **HC – SR04 Ultrasonic Sensor:** 4 ultrasonic sensors are deployed at four sides of the cross-section to count the number of vehicles passed in the given interval.

Fig.1.4 HC - SR04 Ultrasonic Sensor

5. **Raspberry PiCamera-V2:** The PiCamera2 is connected via a ribbon cable with the Raspberry to facilitate taking photos of the current traffic for object detection.

Fig.1.5 Pi-Camera

Technologies used:

- **MQTT protocol:** The MQTT test client on AWS handles the communication between the Raspberry Pi and ESP modules, allowing both to publish and subscribe to vehicle count. The generated data is then stored in the AWS database for decision-making.
- **Python 3.10:** Programming language to configure the camera for object detection and develop communication with AWS on Raspberry Pi.
- **STM32CubeMX IDE:** Employed to program STM32 in embedded C programming language to control traffic lights and communicate with Raspberry Pi through UART.
- **Arduino IDE:** Utilized to program ESP module to interface sensors and communicate with the Cloud.
- **Tensor Flow:** Open-source machine learning platform used to develop and train machine learning models.

IV. Background Theory

The background of the following is required to develop the system:

1. Proficiency in Python and Embedded C is required to program microcontrollers.
2. Hand-on experience in soldering the electronic components on a general-purpose PCB.
3. Knowledge about understanding the datasheets of electronics components and micro-controllers to know the basic specifications.
4. Brief overview of object detection and TensorFlow for generating models for machine learning.

5. Introduction to IoT core (MQTT protocol) and DynamoDB(database) on AWS

V. Methodology

The methodology used for this project includes the following steps:

- Camera module interfaced with Raspberry Pi to monitor traffic density by implementing an object detection algorithm.
- Processed the camera feed to detect the vehicle and estimate traffic density in real time.

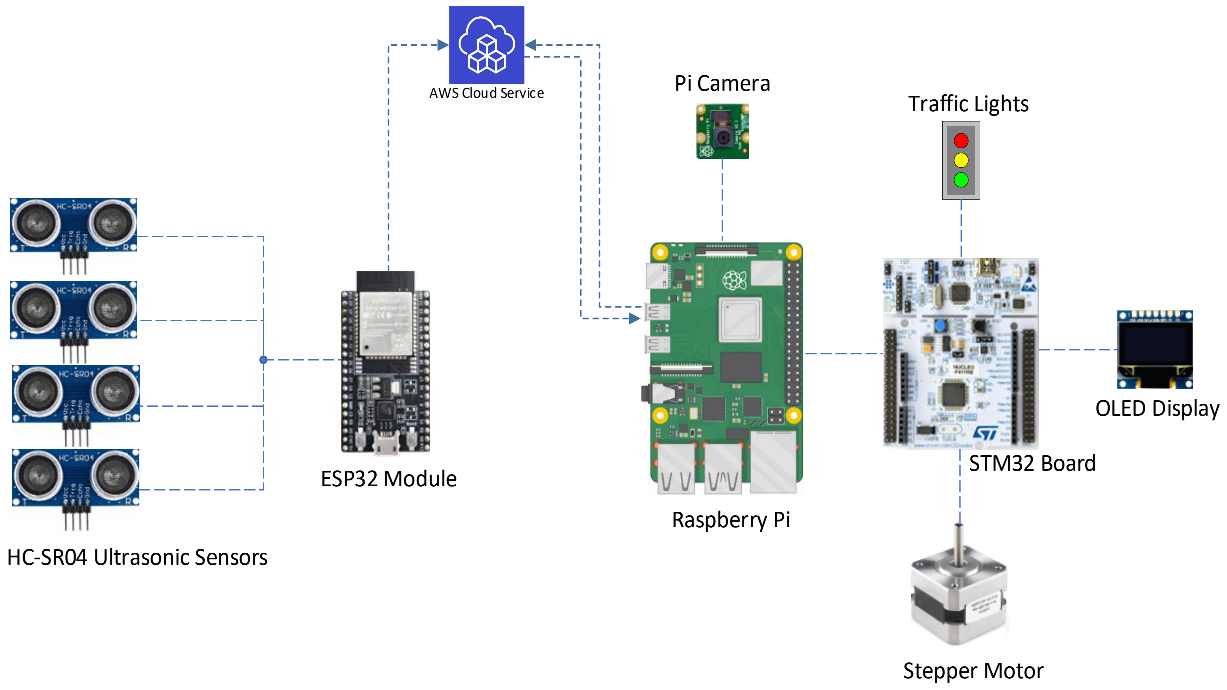


Fig 2. Block Diagram

1. **Traffic-Light control using the STM32:**
   - GPIO pins of STM32 are interfaced with RED, YELLOW, and GREEN LEDs that illustrate the traffic light signals.
   - UART communication with Raspberry Pi for data exchange.
   - Enabled the Raspberry Pi to send commands to STM32 to adjust traffic light timings based on real-time traffic density data.

2. **Traffic Density Monitoring with Object Detection:**

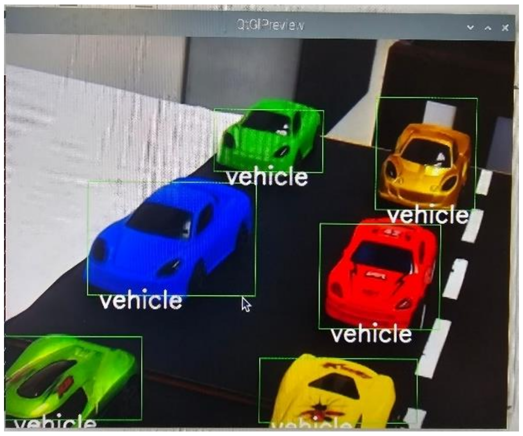


Fig 2.1 Vehicle Detection using object detection

3. **Camera Detection Adjustment with Stepper Motor:**
   - Interface a stepper motor with STM32 to rotate the camera in multiple directions to capture images from all four sides of the intersection.

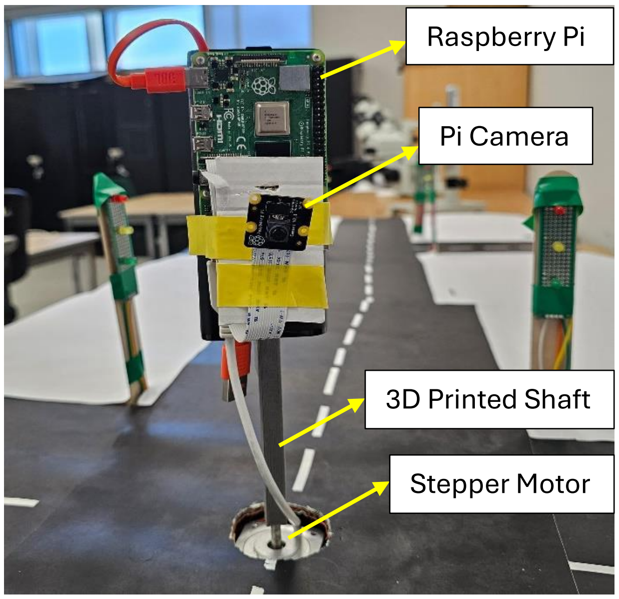
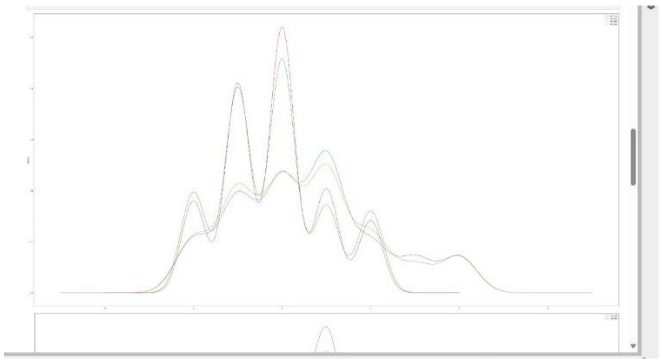


Fig 2.2 Camera Rotation through Stepper Motor

4. **Vehicle Counting with ESP module and Ultrasonic Sensors:**
   - ESP module connected with ultrasonic sensors deployed at another cross-section counts the vehicles approaching the intersection.
   - Transmits vehicle count data to AWS via MQTT protocol over Wi-Fi.

5. **Data Transmission and storage on AWS:**
   - Established MQTT communication between Raspberry Pi and Esp module with AWS IoT core.
   - Publish vehicle count to generate a database on DynamoDB to process traffic information.



6. **Adaptive Signal Control Decision Making:**
   - Machine Learning algorithm used to make decisions regarding changing traffic light timings.
   - Using 15-minute intervals to set patterns for each day of the week as per analyzed data from the database.
   - At every 15-minute interval, the Raspberrypi sends updated green light timings for both the cross-sections to the STM, which might reject them in case the real-time running timers are very different from what is being requested.

7. **Special Case (Emergency Vehicles):**
   - It includes switching the traffic light signal to green for both cross-sections when an emergency vehicle like an ambulance approaches the cross-section.

- Basically, the North-South and South-North direction has an emergency lane enabled. Thus, when an ambulance arrives at the cross-section, the pi-camera detects the vehicle and turns the traffic lights to yellow for a few seconds in the East-West/West-East direction and then red. This is done to give enough time for vehicles to stop safely.
- The pi-camera monitors the ambulance, and when the ambulance leaves the cross-section, a timer of 10 seconds starts. The lights turn yellow in North-South/South-North directions and then to red. Simultaneously, the lights will turn green in East-West/West-East directions.

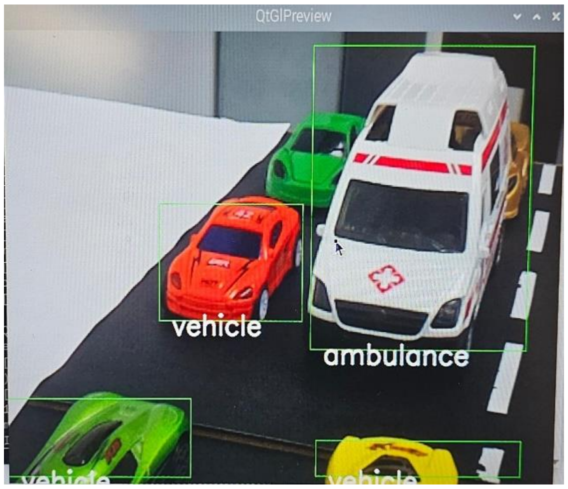
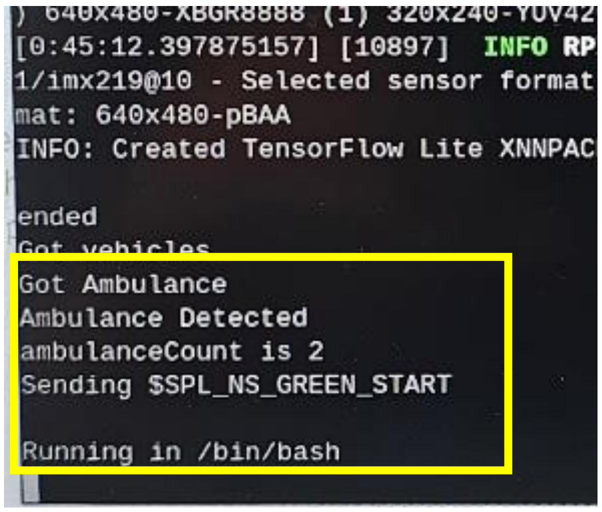


Fig 2.3.1 Emergency Vehicle Detection

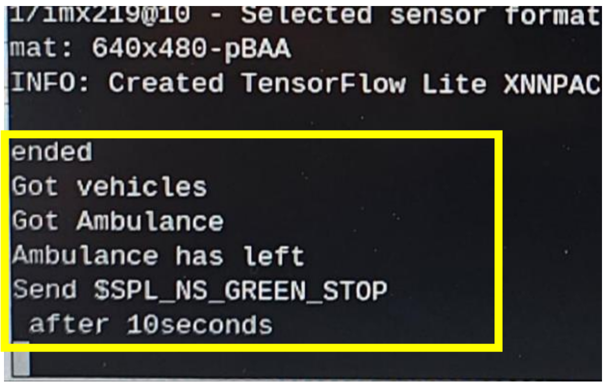


```
) 640x480-XBGR8888 (1) 320x240-YUV42
[0:45:12.397875157] [10897]  INFO RP
1/imx219@10 - Selected sensor format
mat: 640x480-pBAA
INFO: Created TensorFlow Lite XNNPAC

ended
Got vehicles
Got Ambulance
Ambulance Detected
ambulanceCount is 2
Sending $SPL_NS_GREEN_START

Running in /bin/bash
```

Fig 2.3.2 Setting up of green lights in North-South direction



```
1/imx219@10 - Selected sensor format
mat: 640x480-pBAA
INFO: Created TensorFlow Lite XNNPAC

ended
Got vehicles
Got Ambulance
Ambulance has left
Send $SPL_NS_GREEN_STOP
 after 10seconds
```

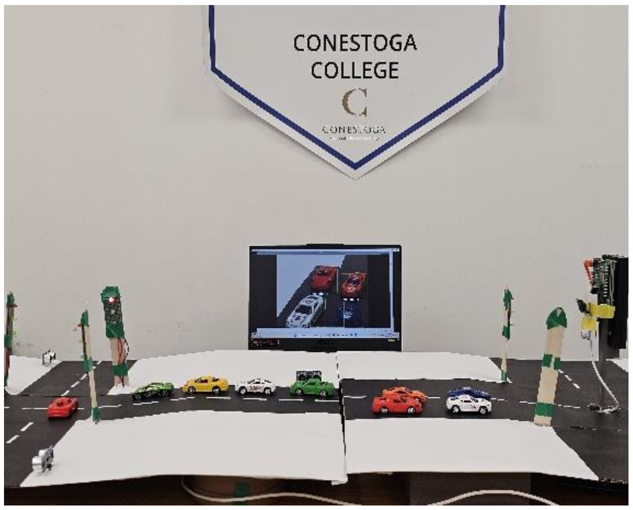
Fig 2.3.3 Green Lights stopped after emergency vehicle left the intersection

VI. Results/Analysis

Greenlight durations for specific lanes have been extended through real-time traffic monitoring via camera and ultrasonic sensors.

As per learned data patterns, green light timings for both cross-sections are updated for each day of the week.

Live green lanes for emergency vehicles have been implemented, optimizing their response time.



VII. Cost analysis

Estimating the expenses of this project includes the cost of purchasing microcontrollers like STM32 board, Raspberry Pi, Pi-Camera, and ESP module. The additional hardware cost includes a stepper motor and its driver, cables, connectors, general-purpose PCBs, and LEDs. Open-source IDEs are utilized to program ESP, STM, and Raspberry Pi.

VIII. Recommendations

One potential future direction for this project could involve synchronizing traffic light signals across multiple intersections based on traffic density levels.

This also involves implementing a singular control board for each cross-section, which can talk to the cloud, capture real-time traffic, and control that particular traffic light.

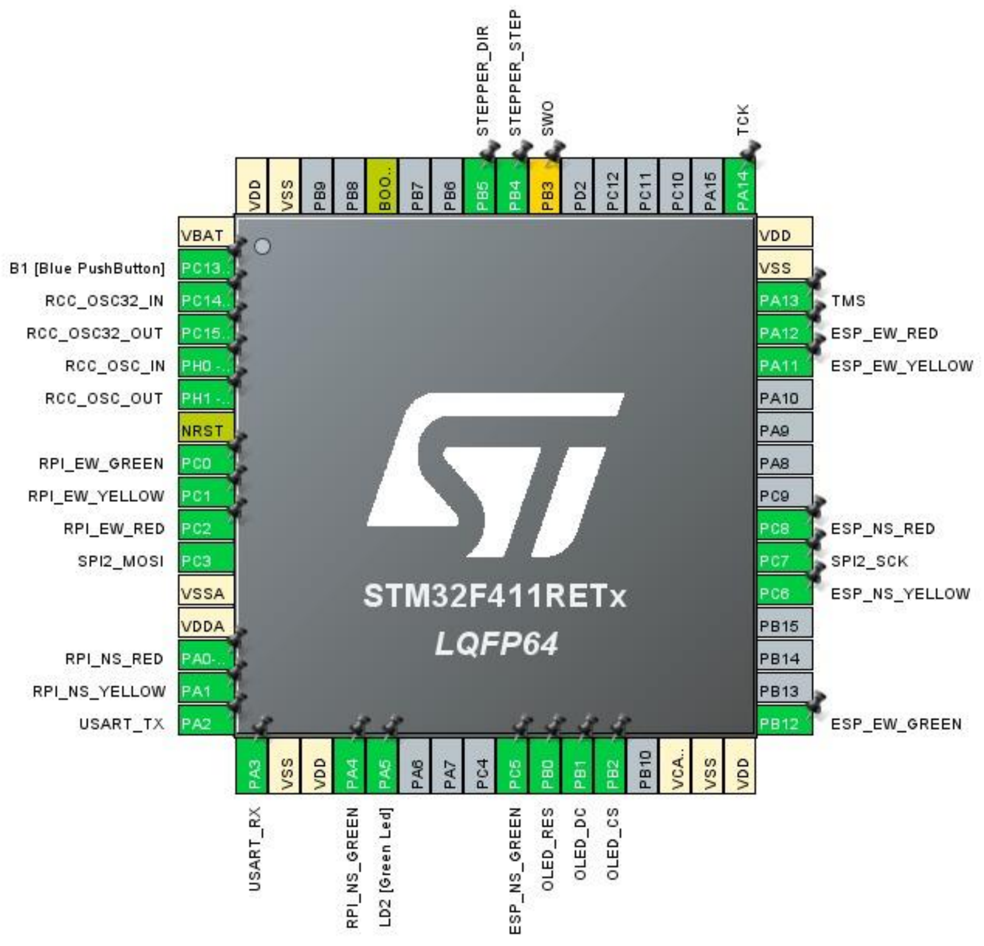
There should also be a minimalistic backup controller that can control the traffic lights in case the cloud connection fails or the main controller becomes unresponsive.

IX. References

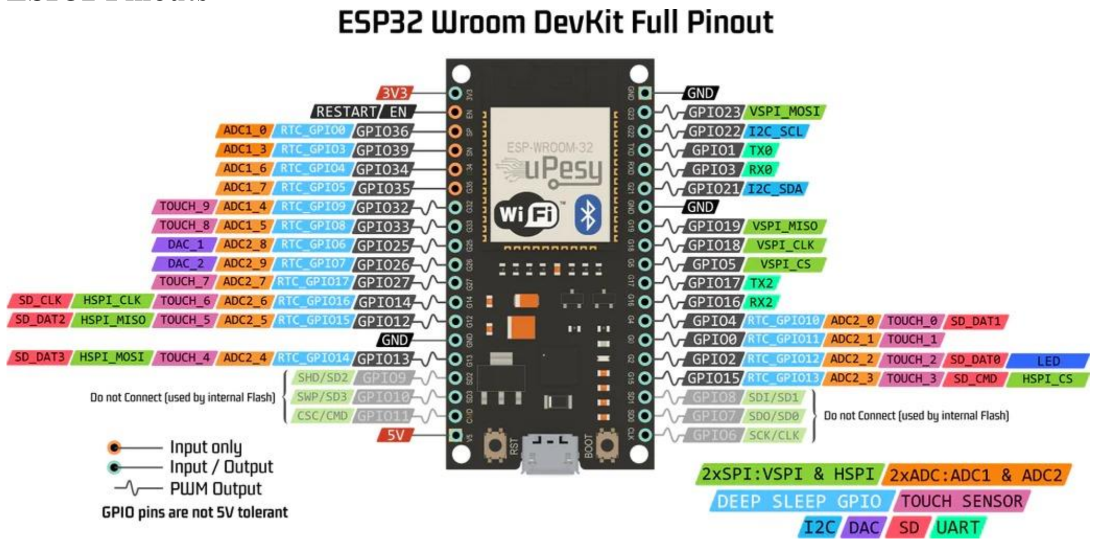
- https://help.realvnc.com/hc/en-us/articles/360002249917-RealVNC-Connect-and-Raspberry-Pi.
- https://github.com/raspberrypi/picamera2?tab=readme-ov-file
- https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf
- https://docs.aws.amazon.com/iot/latest/developerguide/connecting-to-existing-device.html
- https://docs.aws.amazon.com/iot/latest/developerguide/iot-ddb-rule.html
- https://medium.com/@maheshwar.ramkrushna/step-by-step-guide-inserting-mqtt-data-into-a-database-using-aws-iot-core-and-lambda-ba0ec0d25b3e
- https://how2electronics.com/connecting-esp32-to-amazon-aws-iot-core-using-mqtt/
- https://github.com/nithin-k-shine/Controlling-remote-LED-connected-to-ESP32-from-AWS-cloud-using-MQTT/blob/main/LED_AWS.ino
- https://youtu.be/aH3sLEQI4_w?si=Wlvf_1Yh54TJjYH8
- https://youtu.be/z-I-r3PX2lU?si=vtYSGGohMvBrds1H
- https://github.com/harshkzz/ESP32-HCSRO4-SERIAL-MONITOR/tree/main/ESP32_HCSRO4_SM
- https://youtu.be/sXNcZdf4NS0?si=0Fym1LcTPAH0D8X7
- https://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf
- https://neo4j.com/blog/build-predictive-model-python/

Appendix

1. STM 32F411RE Pinouts



2. ESP32 Pinouts


ESP32 Wroom DevKit Full Pinout
3V3
RESTART/ EN
ADC1_0 RTC_GPIO0 GPIO36
ADC1_3 RTC_GPIO3 GPIO39
ADC1_6 RTC_GPIO4 GPIO34
ADC1_7 RTC_GPIO5 GPIO35
TOUCH_9 ADC1_4 RTC_GPIO9 GPIO32
TOUCH_8 ADC1_5 RTC_GPIO8 GPIO33
DAC_1 ADC2_8 RTC_GPIO6 GPIO25
DAC_2 ADC2_9 RTC_GPIO7 GPIO26
TOUCH_7 ADC2_7 RTC_GPIO17 GPIO27
SD_CLK HSPI_CLK TOUCH_6 ADC2_6 RTC_GPIO16 GPIO14
SD_DAT2 HSPI_MISO TOUCH_5 ADC2_5 RTC_GPIO15 GPIO12
GND
SD_DAT3 HSPI_MOSI TOUCH_4 ADC2_4 RTC_GPIO14 GPIO13
SHD/SD2 GPIO9
SWP/SD3 GPIO10
CSC/CMD GPIO11
Do not Connect (used by internal Flash)
5V
GND
GPIO23 VSPI_MOSI
GPIO22 I2C_SCL
GPIO1 TX0
GPIO3 RX0
GPIO21 I2C_SDA
GND
GPIO19 VSPI_MISO
GPIO18 VSPI_CLK
GPIO5 VSPI_CS
GPIO17 TX2
GPIO16 RX2
GPIO4 RTC_GPIO10 ADC2_0 TOUCH_0 SD_DAT1
GPIO0 RTC_GPIO11 ADC2_1 TOUCH_1
GPIO2 RTC_GPIO12 ADC2_2 TOUCH_2 SD_DAT0 LED
GPIO15 RTC_GPIO13 ADC2_3 TOUCH_3 SD_CMD HSPI_CS
GPIO8 SDI/SD1
GPIO7 SDO/SD0
GPIO6 SCK/CLK
Do not Connect (used by internal Flash)
ESP-WROOM-32
uPesy
RST
BOOT
Input only
Input / Output
PWM Output
GPIO pins are not 5V tolerant
2xSPI:VSPI & HSPI
2xADC:ADC1 & ADC2
DEEP SLEEP GPIO
TOUCH SENSOR
I2C
DAC
SD
UART