**Internet of Things (IoT)**

**IoT based Smart Parking System**

**Development Part-1**

- Udhayanithi.N

Parisutham Institute of Technology and Science

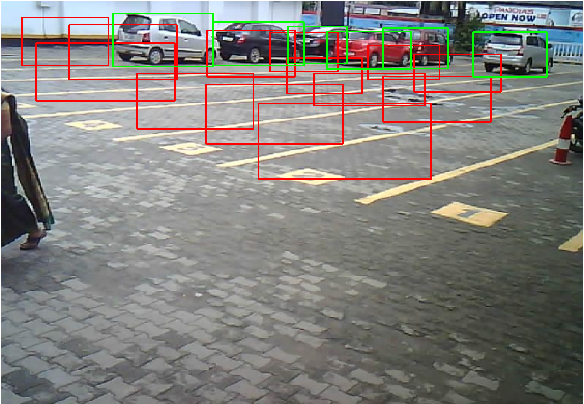
**IoT Based Smart Parking Using Computer Vision Techniques with Raspberry-Pi :**

The project describe an IoT based solution for monitoring of parking availability based on computer vision using Raspberry Pi. It allows us to detect and track cars in a parking lot. Parking Monitoring System tracks availability by analysing the image by detecting edges of cars using computer vision techniques by capturing images from raspberry pi controlled cameras in real-time. The system is developed to determine the number of slots and their location of parking places to inform the drivers. We provide a web application based interface for visualizing the status of each parking slot by recognizing a car in a parking lot and to define parking spaces as either reserved or available for parking

**Smart Parking with YOLO-V3:**

Maintaining empty parking spot count using YOLO real-time vehicle detection. Code readily runnable in google colab. Due to occlusions (coming due to the presence of mirror in the middle of camera and parking lot which slightly reflects nearby people passing through), low resolution of video and positioning of cars at different angles in the parking lot and limitations of yolo, it cannot detect every car in all the frames and hence the count fluctuates. Please don’t mind the wrong number of empty spots, I didn’t count them before running the program since my focus was to check whether the count fluctuates or not. Same is the case with Mask-RCNN.

But for a different video with high resolution and less occlusions, the case becomes different. Note that in the video below, the moving car comes in front of the parked on for few seconds and thus YOLO couldn’t detect the occluded car and the count changed. Other than that it worked fine for the resolution the video had.



**Smart Parking System:**

Vision based Smart Parking system using RASPBERRY PI CAMERA MODULE-CAMs, Raspberry Pi and cloud deployed NodeJS web-server.

**Contents:**

**#** Problem statement

**#** Solution proposed

**#** Segments involved

**#** Technical stack

**#** Technical explanation

**#** Flowchart of system

**#** Gallery

**Problem:**

Parking has always been an issue in metropolitan cities. Let it be a shop or a public place, parking wastes time, fuel and sometimes breaks sweat to get it done. Drivers tend to circle around the place searching for parking spots. This increases traffic congestions, wastes fuel and increases pollution. It escalates to a risky scale during festival times.Unable to find proper parking spots, some vehicles are parked in narrow spaces causing traffic jams.

**Solution proposed:**

* The project proposes to install low cost camera modules in multiple parking lots across the city, which streams live image to the corresponding remote server
* The remote server processes the data from the camera module and decides on the number of vacant parking spaces available in the parking lot
* The remote server updates the number of vacant parking slots and number of filled parking slots in a cloud database
* The number of vacant parking slots and their location is displayed in a web application accessible to general public and free to use.
* The database is updated continuously , ensuring a pristine user experience

**Segments involved:**

* Decentralized server (Raspberry Pi) for image processing, computation and network management.
* RASPBERRY PI CAMERA MODULE-CAM hardware setup for wireless image transmission and reception by server.
* Object detection and updation of database.
* Cloud Deployed and completely scalable Website and Cloud Database management.

**Technical stack:**

**Processing server:**

* + - Raspberry Pi 4B
    - Raspbian 32-bit OS
    - SSH access – PuTTy, FileZilla
    - Auto run on boot-up
    - TCP Sockets
    - PILS library

**Raspberry pi camera module setup:**

* Arduino IDE
* RASPBERRY PI CAMERA MODULE board
* Socket library
* TCP protocol

**Website:**

* Node.js(express)
* HTML5/CSS3
* MongoDB
* Google Maps API services

**Object detection and updation:**

* OpenCV library
* pyMongo library
* Numpy library
* OS module
* python

**Technical explanation:**

**The Website:**

* The website is a scalable, cloud deployed, responsive web application accessible to general public and free to use.
* Depending on the functionality desired, the user can either search for parking lots near their current location(Mode 1) or a desired destination(Mode 2).
* The application uses geolocation technology to find the device location and prints out an interactive map with the 10 closest parking lots and the availability.
* The user can click on the parking slot in the map which links to google maps for directions to that parking lot from the user location(in Mode 1) or from the destination(in Mode 2)
* It uses Node Js as the back end environment and Express as the server technology.

**Raspberry pi camera module setup:**

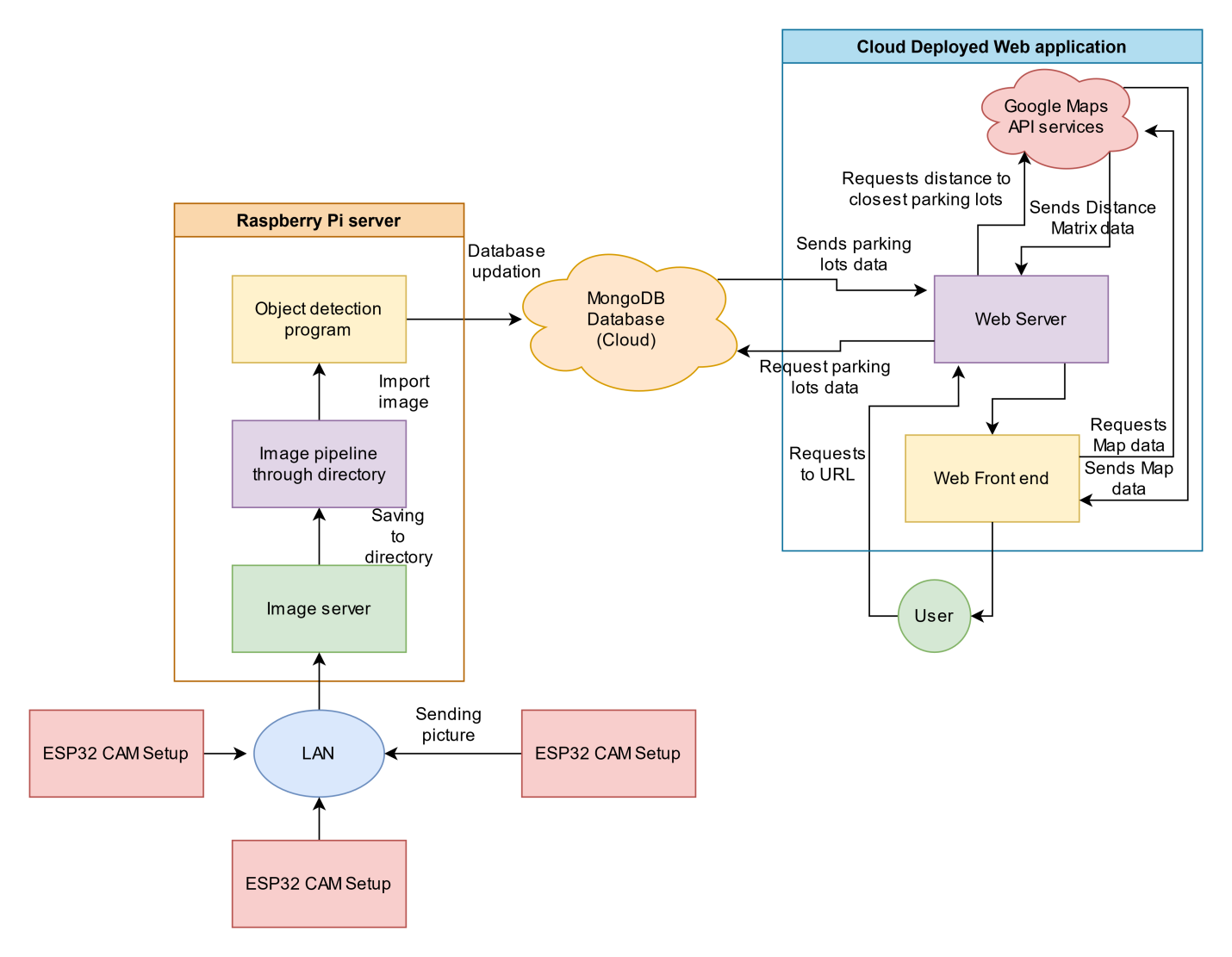
* Raspberry pi camera module is used to take pictures in regular intervals and transmit it to a remote server which is in the same WiFi network.
* Camera pins are defined , camera settings are configured and is initialised.
* Raspberry pi camera module connects to LAN.
* Raspberry pi camera module establishes connection with remote server.
* A frame is captured from the camera and stored in a camera\_fb\_t pointer , which holds the pixel data , height , width of the image.
* Pixel data is transmitted to the server through TCP protocol.
* Since the size of pixel data exceeds the size limit of single TCP socket, pixel data is broken up into chunks and sent individually.
* The process repeats for every two seconds approximately.

**The Server:**

* Raspberry Pi 4B is used as local server for image reception, processing, slot computation and updation to cloud.
* PuTTy is used to connect wirelessly to the Pi over SSH and *FileZilla was used to transfer the file over SFTP.*
* TheRaspberry pi camera moduleimage reception and object detection python scripts run on boot-up. This was implemented by modifying the .bashrc script to execute the python scripts on boot-up or when a terminal is launched.
* TheRaspberry pi camera modulescript receives pixel data of the image as a chunk of bytes. It is stored in a byte array and is converted into a JPEG image using Pillow library.
* The script then stores each image under proper naming convention (ESP\_XX\_CHN\_X\_time) and stores in the respective directory.
* The Pi is connected to the same local network as the RASPBERRY PI CAMERA MODULEs through WiFi. Internet is also enabled to perform updation to the cloud.

**Object detection and computation:**

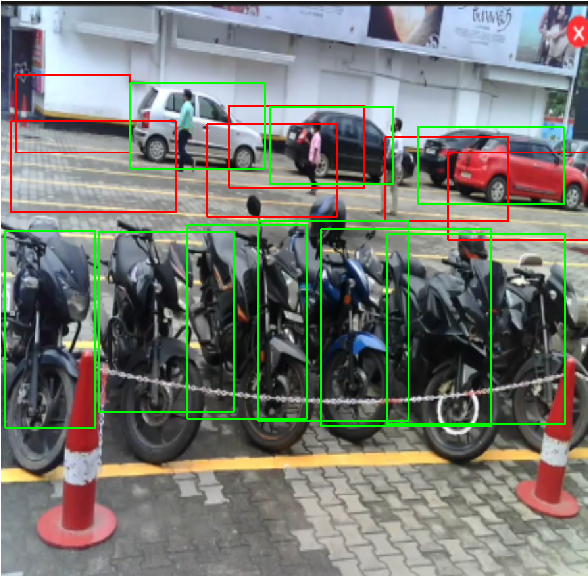
* It processes the Images obtained from parking lots one after the other and identifies the bounding boxes of the objects (cars and bikes) utilizing a pretrained model(MobileNet SSD model) in OpenCV DNN module.
* Based on Intersection Over Union calculation between the predicted bounding boxes and manually drawn bounding boxes representing parking areas Occupancy of respective parking lots gets updated in the server.
* After processing, the image gets deleted automatically to avoid reprocessing.

**Flow chart of the system:**

**Raspberry pi camera module:**



**Object detection:**



**Coding Part:**

Import argparse

Import cv2

Import numpy as np

Import click\_coordinate,coordinates

Import urllib.request

#import car\_classifier

r = argparse.ArgumentParser()

Parser.add\_argument("-setup\_park", help="To plot parking slots in new parking area",type=bool,default=False)

Parser.add\_argument("-slots", help="Number of parking slots",type=int,default=6) Args = parser.parse\_args()

Def saveData(coord\_list):

Fl = open('coordinates.py', 'w+')

Fl.write("boxes=")

Fl.write(str(coord\_list))

Fl.close()

capture\_img():

Camera = cv2.VideoCapture('video input final cmprsd.mp4')

Print("Taking image..." )

Retval, camera\_capture = camera.read()

File = "process\_img.png"

Cv2.imwrite(file, camera\_capture)

V = np.median(image)

Lower = int(max(0, (1.0— sigma) \* v))

Upper = int(min(255, (1.0 + sigma) \* v))

Edged = cv2.Canny(image, lower, upper)

Return edged

Count=O

For y in range(y1,y2):

For x in range(x1,x2):

If edges[y] [x] != O:

Count+=l

Return count

Def solt\_info(args,coord\_list):

Camera = cv2.VideoCapture('video input final cmprsd.mp4') While (camera.isOpened()):

Retval, img = camera.read() If retval==True:

Scale\_percent = 30

Width = int(img.shape[l] \* scale\_percent / 100) Height = int(img.shape[0] \* scale\_percent / 100)

Dim = (width, height)

Img = cv2.resize(img, dim, interpolation = cv2.lNTER AREA) cv2.COLOR BGR2GRAY) Edges=auto\_canny(imge) l=edges.copy()

For j in range(len(coord\_list)):

Cv2.rectangle(img, coord\_list[j][l][l]), (O,255, 0), 2)

For j in range(len(coord\_list)):

Cv2.rectangle(l,  coord\_list[j][0][1]), (coord\_list[j][1][0],(255, O, 0), 2)

cv2.imshow("parking slots",i)

If cv2.waitKey(1) & OxFFord("q"):

Break

rea.index(s)coord\_list[white\_area.index(s)][l][l]), (0, 0, 255), 2)

Cv2.imshow("Real Timee",img)

Continue;

Camera.release()

Cv2.destroyAllWindows()

Def display(white\_area):

For I in range(len(white\_area)):

print("slot-"+str(i+l)+"="+white\_area[i]+"\n")

Info=inf0+white\_area[i]

Print("\nString:",info)

If args.setup\_park:

Print("true")

Capture\_img()

Image = cv2.imread('process\_img.png')

Scale\_percent = 30

Width = int(image.shape[l] \* scale\_percent / 100) Height = int(image.shape[0] \* scale\_percent / 100)

Dim = (width, height)

Image = cv2.resize(image, dim, interpolation = cv2.lNTER AREA) Coord\_list=click\_coordinate.find\_coord(args,image) saveData(coord\_list)

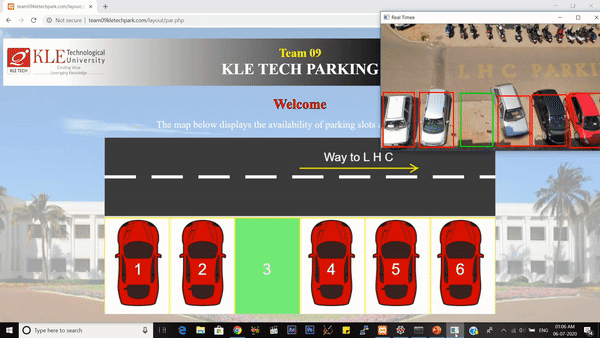
Solt\_info(args,coord\_list)

Else:

Print("False")

Coord list=coordinates.box

**Output:**



**Conclusion**:

Thus,we did a python code for loading and preprocessing the dataset contains the parking slot information.