**Internet Of Things**

**Complete Project Submission**

IOT Based Smart Parking System (Phase-5)

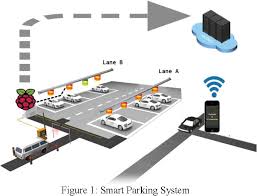
-Udhayanithi.N Parisutham institute of technology and science



**Problem Definition:**

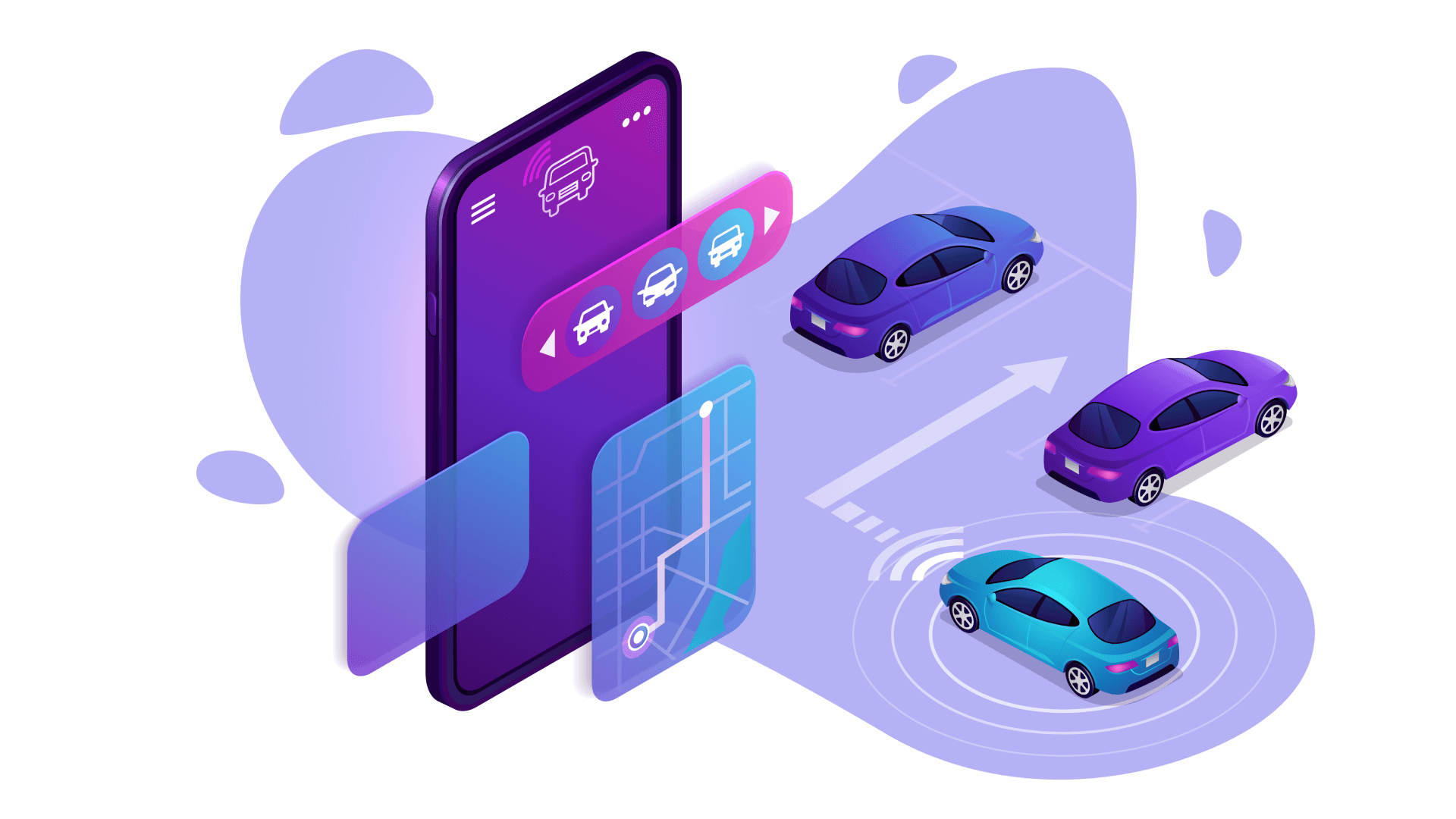
In the greater part of the advanced urban areas it is troublesome and costly to make additionally parking spots for vehicles since the quantities of vehicles that are running out and about are expanding step by step and the include of the free spaces in the urban communities are the equivalent. This issue prompts blockage for stopping searchers and drivers. To build up an IoT structure that objectives Parking Management which is greatest difficulties in current urban areas. Unavoidable presence of advanced cell urges clients to favor portable application-based arrangements. Development of IoT has cleared route for incorporation of cell phones, remote correspondence advances and portable Applications. This task is an IoT based Smart stopping framework for shrewd urban areas that coordinates with website page. It gives an extensive stopping arrangement both for the client and proprietor of the stopping space. The principle highlight of this venture is to distinguish the closest free parking spot and to exploring to the stopping opening. IR sensors are utilized to recognize if a parking space is free. Accessibility of a free space with its area data is sent utilizing GSM/GPRS module innovation, microcontroller and remote correspondence innovation to the worker and is recovered through a message application. A dynamic calculation is utilized to recognize whether the stopping space is unfilled or involved. The proprietor of the parking spot can likewise get the investigation of the quantity of filled and accessible openings energetic just by pinging our framework through short message. This framework helps in extemporizing the administration of stopping framework by adhering to rules of the legislature, for instance dealing with various parking spots in the city.

**IOT Sensor Design:**



IoT-based smart parking system deployment requires integrating various devices, sensors, and microcontrollers. The IoT data picked up from the sensors gets transmitted over a wireless connection to a cloud server. The information is collated and analyzed in real-time to create a map of available parking slots, which is reflected on the smartphone application.  
 These sensors can be based on a variety of sensing technologies, the most commonly used are magnetometer, infrared, ultrasonic, and radar.

**Real-Time Transit Information Platform:**



Cameras are installed on every level of the parking structure to track where vehicles are located . In this case, specialised software is needed to identify the number of vacant and occupied spots. Cameras can also be mounted on light poles or buildings structures fo on-street parking systems.

This proposed smart parking system consists of the onsite deployed Interet of Things(IoT) module which delivers real-time output and moniters the flow of the parking of vehicles in and out of that particular parking lot.

Integration Approach

**Raspberry Pi:**

Raspberry Pi board with an operating system installed, such as Raspbian.  
  
**camera:**

Cameras are installed on every level of the parking structure to track where vehicles are located .

**Cloud Platform:**

Choose a cloud platform for data storage and analysis. Options like Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure are popular choices.

**Algorithm For Vehicle Detection:**

The vehicles are detected using the YOLOv3 object detection algorithm . this algorithm can improve the small object detection effect and solve the problem that the object is difficult to detect due to sharp change of the object scale .

**Parking Space Detection Camera:**

Parking Detection Camera is a smart parking solution that uses cutting-edge guesswork the system will notify you when someone leaves or enters a certain parking place, as well as how long they remain there. This technology is simple to operate and very dependable. The camera captures the parking image/video, and the parking floor plan is formed based on it. As a result, the user enjoys continuous video streaming of the monitored parking space.

**Hardware Specification:**

* Size – around 25\*24\*9 mm
* Weight- 3 g
* Still resolution – 5 Megapixels
* Video modes – 1080p30,720p60 and 640 X 480P60/90

**Vehicle Detection By Camera Using Python Code:**

Detecting vehicles in a smart parking system using a camera can be achieved with more advanced techniques like object detection using YOLO (You Only Look Once). Here's a Python code example using the opencv-python and darknet library for YOLO-based vehicle detection. We will need to install the required libraries and download the YOLO weights and configuration files.

**Python Source Code:**

#Use the following Python code to perform vehicle detection

import cv2

import numpy as np

# Load YOLOv3 or YOLOv4 model

net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')

# Replace with your YOLO model files

# Load the COCO dataset class names

classes = []

with open('coco.names', 'r') as f:

    classes = f.read().strip().split('\n')

# Initialize the camera or video source (0 for webcam)

cap = cv2.VideoCapture(0)

while True:

    ret, frame = cap.read()

    if not ret:

        break

    # Get the frame dimensions

    height, width, \_ = frame.shape

    # Prepare the frame for YOLO detection

    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

    net.setInput(blob)

    # Get output layer names

    layer\_names = net.getUnconnectedOutLayersNames()

    # Run YOLO detection

    outs = net.forward(layer\_names)

    # Initialize lists for detected objects' information

    class\_ids = []

    confidences = []

    boxes = []

    # Analyze the output

    for out in outs:

        for detection in out:

            scores = detection[5:]

            class\_id = np.argmax(scores)

            confidence = scores[class\_id]

            if confidence > 0.5: # You can adjust the confidence threshold

                center\_x = int(detection[0] \* width)

                center\_y = int(detection[1] \* height)

                w = int(detection[2] \* width)

                h = int(detection[3] \* height)

                # Rectangle coordinates

                x = int(center\_x - w / 2)

                y = int(center\_y - h / 2)

                boxes.append([x, y, w, h])

                confidences.append(float(confidence))

                class\_ids.append(class\_id)

    # Non-maximum suppression to remove duplicate detections

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    # Draw bounding boxes around vehicles

    for i in range(len(boxes)):

        if i in indexes:

            x, y, w, h = boxes[i]

            label = str(classes[class\_ids[i]])

            confidence = confidences[i]

            color = (0, 255, 0) # Green

            cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

            cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

    # Display the frame with vehicle detection

    cv2.imshow('Vehicle Detection', frame)

    if cv2.waitKey(1) & 0xFF == 27:

# Press 'Esc' to exit

        break

cap.release()

cv2.destroyAllWindows()

Replace 'yolov3.weights', 'yolov3.cfg', and 'coco.names' with the paths to your downloaded YOLO model files and class names. This code captures video from a camera (you can change the source as needed) and detects vehicles using YOLOv3. You can adjust confidence thresholds and customize the appearance of detected vehicles.

**Output:**

****

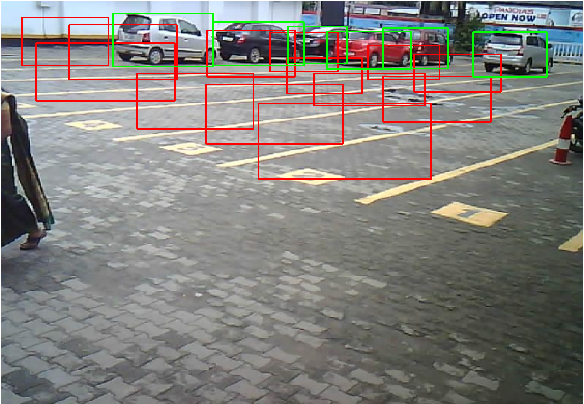
**IoT Based Smart Parking Using Computer Vision Techniques with Raspberry-Pi :**

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

**SmartParking withYOLO-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.



**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:**

**Technical explanation:**

**Website:**

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

**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.

. **Processing server:**

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

**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.

**Raspberry pi camera module:**



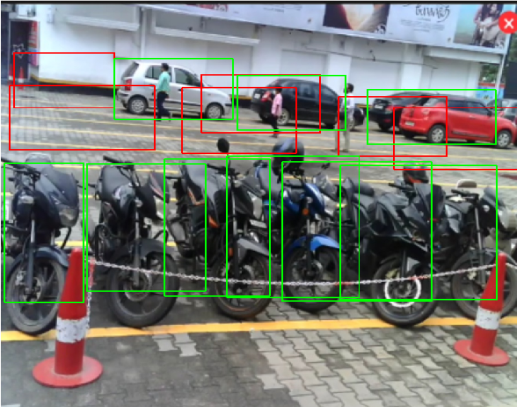
**Raspberry pi camera module setup:**

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

**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\_tpointer , 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.

**Object detection:**



**Object detection and updation:**

* OpenCV library
* pyMongo library
* Numpy library
* OS module
* Python

**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.

**Raspberry pi integration :**

**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 parkingarea",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")

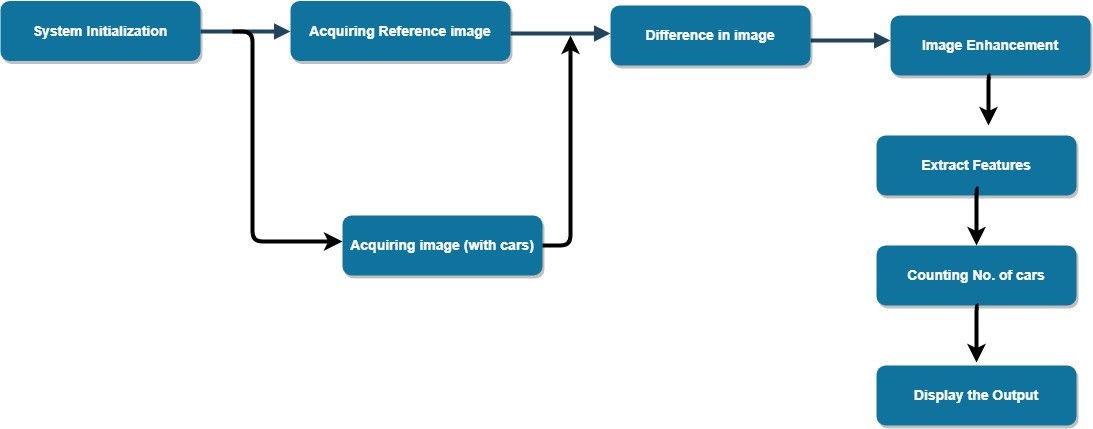
Coord list=coordinates.box

**Output:**

****

**Proposed Model And Methodlogy:**

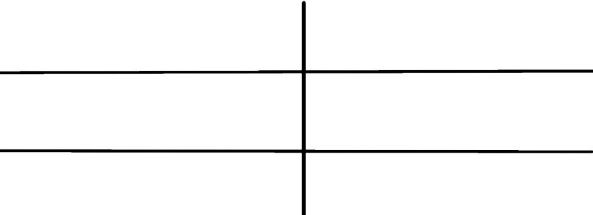
The main flow of the framework is shown in the Fig-1. Videos are acquired from the top view of the parking arena with the help of a fixed camera. Video is segmented into frames. Then from each segment a key frame is extracted and further processing is applied on this key frame, to reduce the computational complexity.



**Feature engineering:**

1. **System Initialization:**

In the initial stage, an image is captured by steady CCTV camera at time of installation which is the background reference image. This reference image does not contain any cars. The main purpose is to identify the parking slots in the image. The camera which is used to take the images is fixed at certain position and it faces a fixed direction all the time.



1. **Image Acquisition:**

In this step, the picture of parking space containing cars is taken with the help of a high-definition camera.  
The image frame containing six lane image is divided lane-wise.  
The image data is then supplied to the MATLAB software for further processing.



1. **Thresholding of image:**

The RGB image acquired is then converted to gray-scale image and then binary image is created in the Image segmentation module. The equation used for the conversion to gray-scale image is  
Gray= 0.229R+0.587G+0.11B  
The gray scale image of the parking space with cars. From the resulting gray-scale image, binary image is obtained using thresholding technique. The binary image contains all the information about the position and shape of interest. The threshold level is set in such a way that the objects of interest are made into white and the rest of the image black.

1. **Image enhancement:**

The binary image contains a lot of noise which is removed using morphological operations and filters such as the Weiner filter. The holes are removed with the help of imfill and bwareaopen function.



1. **Image dectition:**

In order to detect the cars, blob analysis is done using predefined functions in MATLAB and the number of cars is counted.

**Working model:**

* This operates under the straightforward tenet of obstacle detection and visual feedback. An infrared transmitter and a receiver make up the proximity sensor, which is fixed to the parking lot ceiling. Infra-red rays are emitted by the IR emitter, and these rays typically reflect off of objects. These rays are picked up by the IR receiver, which transforms them into an electrical signal and a potential difference. The ensuing potential discrepancy aids in closing the circuit. The Light Emitting Diode (LED) is positioned along the driveway and turned on in response to the input the IR sensor receives. In order to fix a certain distance based on the typical height of the cars used to send and then receive the radiations, a threshold distance is calibrated using a potentiometer.
* After putting together every part in accordance with the circuit layout and programming the Arduino board. Now, precisely position the sensors and servo motor. We have four parking slots in this to park our vehicle and we place two IR sensors. IR sensor-1 is placed at the entrance and IR sensor-2 is placed at the exit gates respectively and a servo motor is used to operate the common single entry and exit gate. We can place our LCD display based on our convenient place.
* We used IR sensor 1 to identify whether or not vehicles were coming at the gate and IR sensor 2 to determine whether or not the parking space was unoccupied. The LCD display initially indicates that all parking spaces are unoccupied when they are all empty. The IR sensor-1 detects a vehicle when it approaches the parking area gate, and the system then permits entry into the car by opening the servo barrier. Once a vehicle has entered the parking lot and is parked there, a Light Emitting Diode (LED) display indicates that the specified slot is full. This way, the system permits 4 automobiles automatically.
* The mechanism barred the entrance gate by closing the servo barrier in the event that there is no more room for parking. Additionally, the LED display reveals that slots 1, 2, 3, and 4 are all taken. The IR sensor-2 identifies the car when it exits a slot and approaches the parking area gate, at which point the system opens the servo barrier. The slot is then empty as indicated by the LED display. Once more, the system will let the entry of a new vehicle.

**Evaluation:**

**Challenges in-smart-parking-system**

* For making smart parking we need to improve infrastructure an existing structures will not work.
* There should be combination of technology which needs to be used as single technology will not fil the nod The maintenance co will increuse as loT devices will consume power
* Managing the data base of free parking spot is tgh as any other object is present in the spot and sensus will think the parking spot is not free
* If the system is down due to any error, then there will be lot of trouble to drivers in finding the parking
* The installed equipment will be costly and they can be stolen Sometimes if the driver who has previously booked the slut was little late to move the car from parking alot and someone else might have booked for the same slot then that person will not be able to park the car and it will create a conflict.n:

**Output:**

