VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

Fog Computing in Intelligent Transport System
Fog enabled Smart Parking System

A project submitted
in partial fulfillment of the requirements for the degree
of
Bachelor of Technology
In
Computer Science and Engineering

By
Gajula Padmatej(20BCT0208)
Jayasri Jallipalli(20BCT0285)
Vineela Reddy(20BCT0257)
Konreddy Kishankumar Reddy(20BCT0253)

Course Instructor

Dr.Baiju B V

Assistant Professor

April -2023

UNDERTAKING

This is to declare that the project entitled "**Fog enabled Smart Parking System**" is an original work done by undersigned, in partial fulfillment of the requirements for the degree "Bachelor of Technology in Computer Science and Engineering" at School of Computer Science and Engineering, Vellore Institute of Technology (VIT), Vellore.

All the analysis, design and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or University.

Team Member Name

Gajula Padmatej
Jayasri Jallipalli
Vineela Reddy
Konreddy Kishankumar Reddy

ABSTRACT

The intelligent transportation system (ITS) concept was developed to improve road safety, traffic management efficiency, and environmental preservation. Nowadays, ITS applications are getting increasingly data-intensive, and the "5 Vs of Big Data" are used to define their data. As a result, big data analytics must be used to fully utilize such data. The Internet of Vehicles (IoV) links ITS equipment to cloud computing centers, which handle data processing. Transferring large amounts of data from geographically dispersed devices, on the other hand, causes network overhead and bottlenecks, as well as a drain on network resources. Furthermore, using a centralized strategy to analyze ITS big data leads to excessive latency, which is unacceptable for delay-sensitive ITS applications. For real-time big data analytics, fog computing is seen as a viable solution. Essentially, fog technology augments cloud computing by distributing data processing at the network's edge, allowing for faster replies to ITS application queries while conserving network resources. In the IoV dynamic environment, however, fog computing and the lambda architecture for real-time big data processing are difficult to implement. In this regard, this study proposes a unique architecture for real-time ITS big data analytics in the IoV environment. The suggested architecture combines three dimensions: intelligent computing (cloud and fog computing), real-time big data analytics, and the Internet of Things (IoV). In addition, this article covers the IoV environment, ITS big data characteristics, lambda architecture for real-time big data analytics, and numerous intelligent computing technologies. More importantly, this study analyses the advantages and disadvantages of using fog computing and real-time big data analytics in the IoV context. Finally, the section on crucial challenges and future research paths covers some of the issues that should be taken into account in order to implement the proposed design effectively.

ACKNOWLEDGEMENT

We would like to showcase our collective appreciation and gratitude towards everyone who has contributed to the successful and complete accomplishment of our project **"Fog enabled car parking system"** The completion of this project would not have been possible without this undeniable large support.

We would like to show a vote of thanks to our faculty, Prof. Baiju B V, for his continuous guidance, support, and mentorship which was central to the completion of this project within time. The lessons in this course have been instrumental in teaching us the significance of information management from both a societal and a job perspective, and inculcating us with core moral values related to the same.

We would also like to show our immense gratitude to VIT University, for providing us with a platform where we could learn practical concepts and develop practical and real-world projects.

Gajula Padmatej

Jayasri Jallipalli

Vineela Reddy

Konreddy Kishankumar Reddy

Place: Vellore

Date: 06th April, 2023

TABLE OF CONTENTS

Sl. No	Chapter	Page No
1	Introduction	6
2	Literature Survey	7
3	Proposed Methodology	12
4	Architecture/Design	15
5	Description on Various Modules	16
6	Implementation (Sample Code)	17
7	Testing (All Screen Snapshots)	32
8	Results and Discussions	40
9	Conclusion and Future Enhancements	41
10	References	42

1. INTRODUCTION

Rapid improvements in information communication and technology are driving people to migrate to cities, and as a result, cities have grown overcrowded. The number of automobiles utilised for everyday travelling has expanded rapidly as a result of the relocation of a large population to cities. As a result of the increased number of automobiles, parking spots in major cities have diminished. In this circumstance, it's become tough for drivers for parking spaces during peak hours in big cities. More vehicles competing for the same parking place causes traffic congestion. As a result, individuals waste time looking for a parking spot.

Because of the huge growth in the number of automobiles over the previous few years, parking issues have gotten greater attention in recent years. Given the challenges and concerns associated with locating parking spots in major cities, various academics have developed smart automobile parking systems based on cloud computing and the Internet of Things (IoT). Even though the mentioned smart automobile parking systems equipped with cutting-edge technology have added a new aspect to study in this area, their general deployment in real-time remains a difficulty.

The proposed methodology was:

- We will use cameras at each section of the parking area to capture images at required frequencies.
- We then use a fog node with image processing capabilities for each floor or a cluster which analyses the parking slots which are empty.
- The information from the fog nodes is then transmitted to the proxy server and then to the cloud server.
- We used MATLAB to show real time simulation of our project

2.LITERATURE SURVEY

Sl No.	Paper title(Year)	Authors	Proposed system	Gain
1	A hybrid fog architecture: Improving the efficiency in IoT-based smart parking systems. Recent Advances in Computer Science and Communications (2021)	Suri B., Pramanik P. K., & Taneja S.	This chapter presents an IoT-based smart parking system for shopping malls, in addition to explaining the fundamentals of smart parking, Internet of Things (IoT), Cloud computing, and Fog computing. In order to decrease latency, a hybrid Fog architecture is used to analyze IoT data , with Fog nodes connected throughout the hierarchy. By contrasting this auxiliary connection with other	We get to know how to improve the efficiency in IoT-based smart parking systems. We also get to know the recent advances in Computer Science and Communicatio ns
2	Optimizing smart parking system by using fog computing. In International Conference on Advances in Computing and Data Sciences (2019)	Tandon R., & Gupta P. K.	Their main goal is to find an empty location to park a car during peak hours is getting increasingly challenging. Parking at shopping malls, restaurants, and businesses, for example, is a time-consuming activity that also wastes gasoline. Through Vehicular Ad Hoc Networks (VANETs), smart automobile parking assists in finding a parking spot. Some devices, such as roadside units and on-board units , give parking spot information for vehicle communication	We get to know how to optimise smart parking system using fog computing
3	Fog computing for ubiquitous transportation applications—a smart parking case	Muzakkir Hussain M., Khan F., Alam M. S., & Sufyan	They look at the present status of cloud-based solutions for meeting the mission-critical storage and compute needs of IoT-assisted ITS infrastructure, as well as the reasons for using	We get to know how to use fog computing for ubiquitous transportation

	study. In Engineering Vibration, Communication and Information Processing Springer, Singapore. (2019)	Beg M. M.	edge-centered fog computing paradigms. A fog computing topology tailored to ITS designs was also presented. A smart parking case study is also used to show the feasibility of the suggested fog framework	applications
4	Smart parking utilizing iot embedding fog computing based on smart parking architecture. In 2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications, IEEE. (2020)	Maharjan A. M. S., & Elchouemi A.	This paper presents a fog computing architecture to minimize latency and effectively use all available technologies by constructing a fog computing architecture network, which is a multi tier structure in which applications operate concurrently, interact, and compute with one another. Smart parking has gotten a lot of attention because of how simple it is to use and the benefits it provides. The platform's utilization of the Internet of Things and fog computing allows it to shorten the time it takes to locate a parking spot, which saves time and reduces fuel consumption and CO2 emissions, both of which are caused by an excess of unmanaged automobiles in metropolitan areas and unmanaged parking lots.	We get to know how to make smart parking utilizing Iot embedding fog computing based on smart parking architecture
5	Big data processing in fog-smart parking case study. In 2018 IEEE Intl Conf on Parallel & Distributed Processing with	Nguyen, S., Salcic, Z., & Zhang, X.	they mainly talk about the introduction of the Internet of Things (IoT) and its positive effects in the context of city life will be seen first in Applications that directly affect people's lives, such as improving traffic efficiency, reducing time spent in	We get to know how to use big data processing in fog-smart parking

	Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, IEEE. (2018)		vehicles while traveling around the city, and generally reducing traffic congestion. Vehicle parking and management are key concerns that have a direct impact on people's time and can have large financial implications, making it of direct importance to both service providers and users.	
6	Building Decentralized Fog Computing-Based Smart Parking Systems From Deterministic Propagation Modeling to Practical Deployment(2020)	Mikel Celaya-Echarri, Leyre Azipilicueta, Paula Fraga-Lamas,Peio Lopez-Iturri, Fransisco Falcone,Tiago M Fernandez Carames	The proposed system was built by making use of parking sensor nodes based on ZigBee transceivers and ultrasound sensors, while fog computing nodes were created based on a Raspberry Pi Zero and ZigBee/BLE interfaces. Such fog nodes also provided decentralized storage, whose performance was evaluated. The results show that the proposed system is able to deliver information to the driver smartphone In less than 4 s by using lightweight and scalable protocols and with no need for relying on remote servers.	This paper detailed the complete development of a decentralized fog computing-based SP system. Such a development followed a methodology that includes its initial design, its theoretical simulation and its empirical validation.
7	Fog-based dynamic traffic light control system for improving public transport (2020)	Sakhawat Hossan, Naushin Nower	They have proposed a fog-based adaptive traffic light control scheme For multiple intersections with the purpose of increasing traffic throughput, reducing the average waiting time and average fuel consumption. They provided a distributed architecture by utilizing the advantages of fog computing. By using fog nodes,	Their simulation results demonstrate that the proposed scheme could produce a higher throughput, lower waiting time and

			the traffic lights at multiple intersections are controlled and coordinated in the proposed distributed architecture, where multiple intersections can cover the whole metropolitan area and can also be used for monitoring the traffic in the whole city.	consume less fuel compared to others.
8	Advances in Position Based Routing Towards ITS Enabled Fog-Oriented VANET–A Survey (2020)	Ata Ullah, Xuanxia Yao, Samiya Shaheen, Huansheng Ning	PBR protocols involve the position of a moving vehicle to assist in dynamic path discovery for successful and timely message transmissions. Many schemes have discussed about the global positioning, relative positioning and surrounding region based attributes to identify the roads and their intersections based on vehicle position	This paper includes a better taxonomy to present PBR schemes by considering the city environments. Existing survey papers have mainly focused on all type of routing but they have focused on PBR and considered the city environment with more congested topologies, densities and mobility scenarios
9	A Machine-Learning Based Time Constrained Resource Allocation Scheme for Vehicular Fog Computing (2019)	Xiaosha Chen, Supeng Leng, Ke Zhang, Kai Xiong	They introduced the metric called Perception-Reaction Time (PRT), which reflects the time consumption of safety-related applications and is closely related to road efficiency and security. With the integration of the incorporating information-centric networking technology and the fog virtualization approach, we propose a novel fog resource	We found that the deep reinforcement learning converges faster than the Q-Learning algorithm based on the simulation results.

			scheduling mechanism to minimize the PRT	
10	An Efficient Parking Solution for Shopping Malls Using Hybrid Fog Architecture (2020)	Bhawna Suri, Pijush Kanti Dutta Pramanik and Shweta Taneja	They proposed that in the malls, the parking is available at more than one floor which could be ground, basement 1, basement 2 and so on. This is a form of tree topology. Also, on a floor, there are many parking slots, and so the connections are spread across a floor. In HFA, both the topologies are merged to get an efficient parking solution. On each floor, there are sensor nodes or the fog nodes which are connected across and along with the level. Whenever a vehicle is about to reach the entrance gate of the parking area of the building or mall, the request for its allocation is then assigned to a Fog node at the entrance called the front node. The allocation calculated results are displayed on the screen connected to the front node.	The increasing number of personal vehicles has caused major trouble in finding suitable parking space in shopping malls. IoT-based smart parking system has emerged as a viable solution

3. PROPOSED METHODOLOGY

3.1 Proposed Architecture:

The suggested design consists of three levels, as shown in Figure 1. The system's first layer includes the cameras above the parking slots or lanes and is in charge of capturing pictures of parking spaces and identifying whether they are filled or not. The system's second layer is a fog node, which is linked to the cameras via a microcontroller device. The system's third layer has a cloud that is linked to fog and is in charge of storing and handling picture data for longer periods of time. When the vehicle approaches the parking space, the LED panel indicates the status of available parking spaces to the operator or driver. The suggested methodology does not need the phone to validate details regarding available parking spaces in the parking lot. As a result, in order to reduce traffic congestion in parking spots and gas waste, drivers are instructed to the parking slot at the main gate and park their cars.

3.2 Overview of the proposed system:

The suggested fog-based framework is made up of several components, including cameras, a microcontroller chip, a fog node, a LED, and a server side. The proposed framework cameras are in charge of taking pictures of the parking spaces. Following that, using the photo processing approach described in, the number of parking spaces is calculated based on the collected photos. Every five seconds, the fog node gathers the photos and refreshes the parking status display. Many cameras are strategically positioned across the parking lot to cover all parking spaces. The microcontroller serves as a link between both the fog node as well as the cameras in the proposed design. The fog node connects with the cloud, and data from the fog node is sent to the cloud after a certain amount of time. Cloud technology offers the ability to process and handle data for a longer period of time; however, regular connection to the cloud to send and access information not only improves latency but also uses a large amount of bandwidth capacity. The latency is reduced by introducing the middle part of fog node because common accesses to the cloud are avoided because frequent transmission of data to the cloud, processing the pictures to establish available parking space, and access to the data from the LED are time-consuming tasks when compared to fog.

3.3 Workflow employed in the project

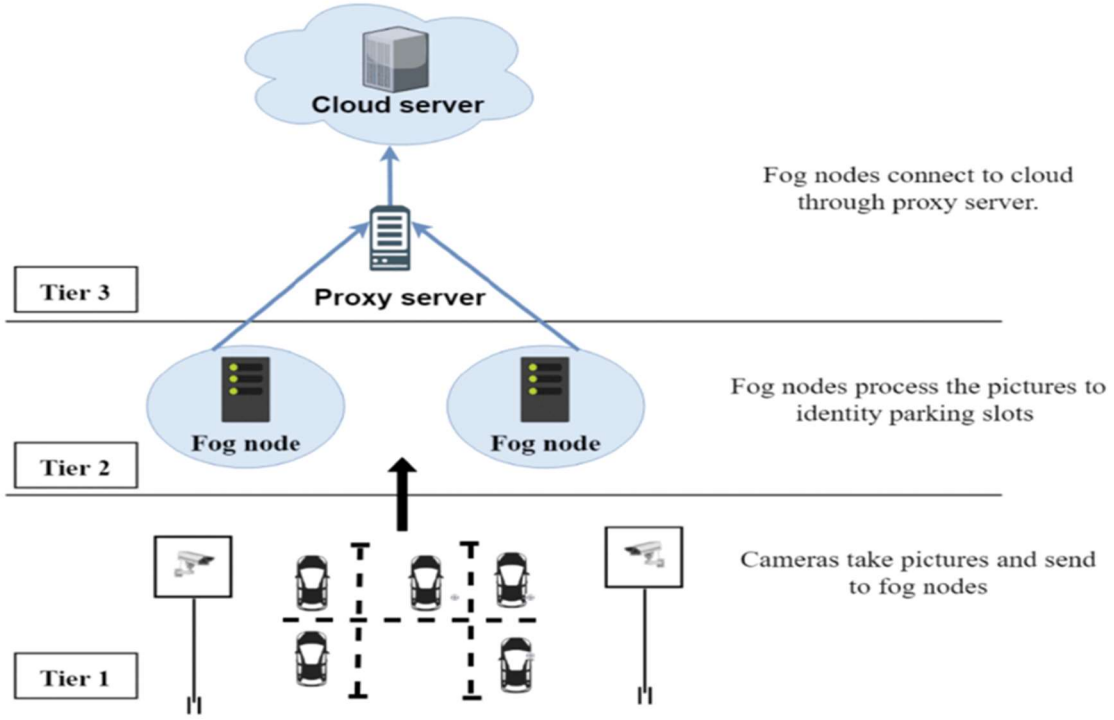

Figure 1 Flowchart of project

3.4 The components of the proposed architecture:

3.4.1 Camera's and Microcontrollers:

The following are the processes for taking the image and determining the parking slot:

1) Set up the system.
    a) Place the cameras in a steady position
2) Image Capture.
    a) Take a picture of the parking lot.
3) Segmentation of images.
    a) Input the RGB picture
    b) Convert the RGB image to grayscale
    c) Use the threshold approaches.

d) Obtain the final picture for segmentation.

4) Make use of picture enhancing techniques.
    a) Remove noise from picture
5) Image detection
    a) Extract image features

3.4.2 The Fog node

The processing of the data produced by the smart parking system is the responsibility of the fog computing module. It gathers data from numerous sensors, cameras, and other connected devices before processing, analyzing, and storing it in real-time. It aids in lowering latency, minimizing data transmission, and enhancing system performance as a whole.

3.4.3 The cloud layer

The cloud's purpose in our proposed system is to store the picture data when it is no longer required by the fog node. Communication between both the fog and the cloud is made possible by a proxy server. After a certain time period, the fog node sends the image data to the cloud, and even if the fog requires picture data, the cloud provides it. Among the many notable aspects of fog computing is interoperability, which is critical when considering the underlying variety of edge nodes. Fog nodes withdraw part of their resources in order to collaborate with one another and meet the processing and storage demands of their neighbours. In this situation, we suppose that fog nodes can interact via proxy server and convey crucial data among nearest neighbours. When there are no available parking spaces in a certain parking location, the accessibility of the closest parking space is shown to the requesting node.

4. ARCHITECTURE AND DESIGN

The architecture of the smart car parking system for one parking area

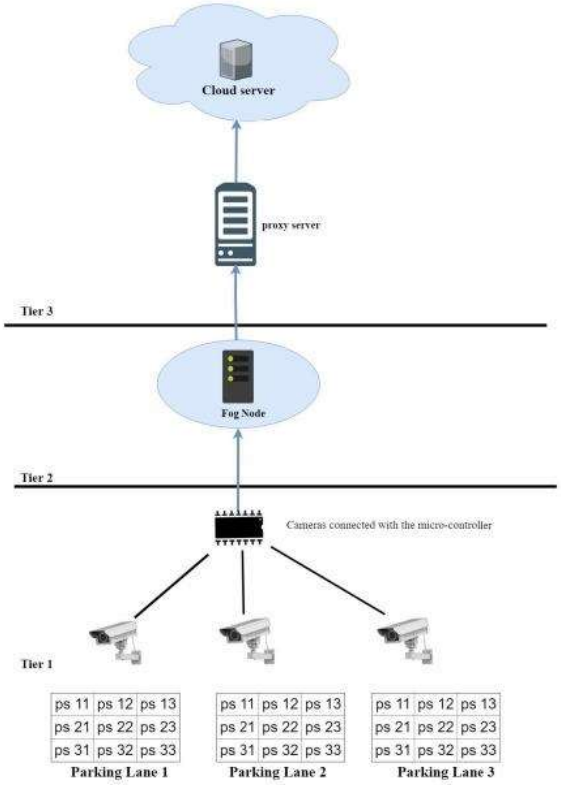

Figure 3:Architecture of the smart car parking system for one parking area

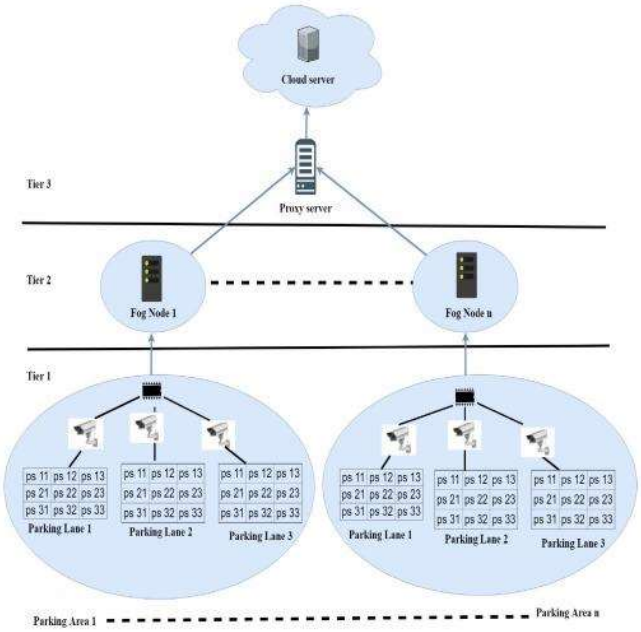

Figure 4:Architecture of smart car parking system for multiple parking areas.

5. DESCRIPTION ON VARIOUS MODULES

5.1 Cloud Computing Module

This module is in charge of processing, analysing, and storing data. It offers the ability to store system-generated data in a safe, scalable manner. Also, it enables data analysis, which enhances system performance overall and offers insightful information about parking

5.2 Proxy server

The system's various components must communicate with one another through the communication module. It enables data communication between the fog computer module, camera module, and sensor module.

5.3 Fog computing module

The processing of the data produced by the smart parking system is the responsibility of the fog computing module. It gathers data from numerous sensors, cameras, and other connected devices before processing, analysing, and storing it in real-time. It aids in lowering latency, minimizing data transmission, and enhancing system performance as a whole.

5.4 Camera Module

The camera module takes pictures of the parking lot, which are then used for object tracking, object classification, and object identification. The camera module also provides parking lot monitoring and aids in the detection of vehicle license plate numbers.

5.5 User Interface Module

With the help of this module, users can interact with the smart parking system through a simple interface. It enables users to reserve a parking space, see the status of parking availability in real-time, and get updates about parking availability.

6. IMPLEMENTATION

6.1 Real time simulation using MATLAB

```
%PROGRAM FOR SMART PARKING SYSYTEM
image = imread('C:\Users\Jayasri\OneDrive\Desktop\car.jpg');
%To read Cars parked in parking area.
background =
imread('C:\Users\Jayasri\OneDrive\Desktop\bg.jpg'); %To read
background image or initial image of parking area
img = double(rgb2gray(image));%convert to gray
bg = double(rgb2gray(background));%convert 2nd image to gray
[height width] = size(img); %image size?
%ITS WORK FOR WHOLE PARKING SYSTEM
totalslot=36;    %Given Total number of slot in the parking
area.
 %Foreground Detection
 thresh=11;
 fr_diff = abs(img-bg);
 for j = 1:width
 for k = 1:height
 if (fr_diff(k,j)>thresh)
 fg(k,j) = img(k,j);
 else
 fg(k,j) = 0;
 end
 end
 end
 park=sprintf('(Orignal Frame) Parking Area with %d
slot',totalslot);
 subplot(2,2,1) , imshow(image), title (park);
 subplot(2,2,2) , imshow(mat2gray(img)), title ('converted
Frame');
 subplot(2,2,3) , imshow(mat2gray(bg)), title ('BACKGND Frame
');

 sd=imadjust(fg);% adjust the image intensity values to the
color map
 level=graythresh(sd);
 m=imnoise(sd,'gaussian',0,0.025);% apply Gaussian noise
 k=wiener2(m,[5,5]);%filtering using Weiner filter
 bw=im2bw(k,level);
 bw2=imfill(bw,'holes');
 bw3 = bwareaopen(bw2,5000);
 labeled = bwlabel(bw3,8);
 %Blob measurements
 blobMeasurements = regionprops(labeled,'all');
numberofcars = size(blobMeasurements, 1);
cars=sprintf('[FOREGROUND] , Total space available is
%d',totalslot-numberofcars);
```

```
 subplot(2,2,4) , imagesc(labeled), title (cars);
 hold off;

 %CONDITION TO CHECK THE VACANT SPACE
 %IF YES then it divide it itno 6 parts as 6 LANEs are there
then for each
 %lane image processing is applied as before and lane with
vacant space
 %comes first with their space.

 if((totalslot-numberofcars)>0);
     fprintf('You can enter into the parking area');
     fprintf('\n Number of car present');
    disp(numberofcars);% display number of cars
     fprintf('Number of vacant space present present');
    disp(totalslot-numberofcars);
    fprintf('PARKING AREA STRUCTURE with LANE:- \n LANE 1\t\t
LANE 2 ');
    fprintf('\n LANE 3\t\t LANE4 \n LANE 5\t\t LANE 6');
%These code just divide the image of full parking area into 6
parts as 3
%rows and 2 coloums.
  r3=int32(height/3);
    c2=int32(width/2);
    img1=img(1:r3,1:c2);
    img2=img(1:r3,c2+1:end);
    img3=img(1+r3:2*r3,1:c2);
    img4=img(1+r3:2*r3,c2+1:end);
    img5=img(2*r3+1:end,1:c2);
    img6=img(2*r3+1:end,c2+1:end);
    imga={img1 img2 img3 img4 img5 img6};   %An Array
iscreated which store 6 image.
    bg1=bg(1:r3,1:c2);
    bg2=bg(1:r3,c2+1:end);
    bg3=bg(1+r3:2*r3,1:c2);
    bg4=bg(1+r3:2*r3,c2+1:end);
    bg5=bg(2*r3+1:end,1:c2);
    bg6=bg(2*r3+1:end,c2+1:end);
    bga={bg1 bg2 bg3 bg4 bg5 bg6};

    % LOOP is taken from LANE 6 to LANE 1 for better
understandibilty.
    % And again previos process is taken for each lane for
vacant space
    % detection.
    for a=6:-1:1
     totalslota=6;     %Also all variables are changed by
adding a as suffix.
        imgb=imga{a};

        bgb=bga{a};
```

```
        [heighta widtha] = size(imgb);
        thresha=11;
 fr_diffa = abs(imgb-bgb);
 for j = 1:widtha
 for k = 1:heighta
 if (fr_diffa(k,j)>thresha)
 fga(k,j) = imgb(k,j);
 else
 fga(k,j) = 0;
 end
 end
 end

 sda=imadjust(fga);% adjust the image intensity values to the
color map
 levela=graythresh(sda);
 ma=imnoise(sda,'gaussian',0,0.025);% apply Gaussian noise
 ka=wiener2(ma,[5,5]);%filtering using Weiner filter
 bwa=im2bw(ka,level);
 bw2a=imfill(bwa,'holes');
 bw3a = bwareaopen(bw2a,5000);
 labeleda = bwlabel(bw3a,8);

 blobMeasurementsa = regionprops(labeleda,'all');
numberofcarsa = size(blobMeasurementsa, 1);
 %If lane is available with vacant slot then this statement is
true and car will get its direction.
 if((totalslota-numberofcarsa)>0)
     fprintf('\n \nGo to Lane %d',a);
 fprintf('\n Number of car present');
    disp(numberofcarsa);% display number of cars
     fprintf('Number of vacant space present present');
    disp(totalslota-numberofcarsa);
    figure, subplot(2,2,1), imshow(image), title('Whole
parking area');
    lane=sprintf('Lane %d is availabe with vacant slot',a);
    subplot(2,2,2) , imshow(mat2gray(imgb)), title (lane);
 subplot(2,2,3) , imshow(mat2gray(bgb)), title ('BACKGND Frame
');
 cars=sprintf('[FOREGROUND] , Total space available in lane %d
is %d',a,totalslota-numberofcarsa);
 subplot(2,2,4) , imagesc(labeleda), title (cars);
 hold off;
 break;
 end
    end
    else
     fprintf('\n No space available in parking area.\n Exit');
     % If whole parking area is full then this statement will
execute.
 end
```

6.2 Implementation using ECLIPSE

6.2.1 Fog only architecture

```
package org.fog.test.perfeval;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.Application;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.Actuator;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDevice;
import org.fog.entities.FogDeviceCharacteristics;
import org.fog.entities.Sensor;
import org.fog.entities.Tuple;
import org.fog.placement.Controller;
import org.fog.placement.ModuleMapping;
import org.fog.placement.ModulePlacementEdgewards;
import org.fog.placement.ModulePlacementMapping;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
import org.fog.utils.distribution.DeterministicDistribution;
public class smartCarParkingFog {
	static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
	static List<Sensor> sensors = new ArrayList<Sensor>();
	static List<Actuator> actuators = new ArrayList<Actuator>();
	static int numOfAreas = 8;
	static int numOfCamerasPerArea1=4;
	static double CAM_TRANSMISSION_TIME = 5;
	private static boolean CLOUD = false;
	public static void main(String[] args) {
		Log.printLine("Starting simulation with fog...");
		try {
```

```
                        Log.disable();
                        int num_user = 1; // number of cloud users
                        Calendar calendar = Calendar.getInstance();
                        boolean trace_flag = false; // mean trace events
                        CloudSim.init(num_user, calendar, trace_flag);
                        String appId = "dcns"; // identifier of the application
                        FogBroker broker = new FogBroker("broker");
                        Application application = createApplication(appId, broker.getId());
                        application.setUserId(broker.getId());
                        createFogDevices(broker.getId(), appId);
                        Controller controller = null;
                        ModuleMapping moduleMapping =
ModuleMapping.createModuleMapping(); // initializing a module mapping
                        for(FogDevice device : fogDevices){
                                if(device.getName().startsWith("c")){ // names of all Smart
Cameras start with 'm'
                                        moduleMapping.addModuleToDevice("picture-
capture", device.getName());  // fixing 1 instance of the Motion Detector module to each
Smart Camera
                                }
                        }
                        for(FogDevice device : fogDevices){
                                if(device.getName().startsWith("a")){ // names of all fog
devices start with 'a'
                                        moduleMapping.addModuleToDevice("slot-detector",
device.getName());  // fixing 1 instance of the Motion Detector module to each Smart Camera
                                }
                        }
                        //moduleMapping.addModuleToDevice("user_interface", "cloud"); //
fixing instances of User Interface module in the Cloud
                        if(CLOUD){
                                // if the mode of deployment is cloud-based
                                moduleMapping.addModuleToDevice("picture-capture",
"cloud"); // placing all instances of Object Detector module in the Cloud
                                moduleMapping.addModuleToDevice("slot-detector",
"cloud"); // placing all instances of Object Tracker module in the Cloud
                        }

                        controller = new Controller("master-controller", fogDevices, sensors,
                                        actuators);

                        controller.submitApplication(application,
                                        (CLOUD)?(new
ModulePlacementMapping(fogDevices, application, moduleMapping))
                                                        :(new
ModulePlacementEdgewards(fogDevices, sensors, actuators, application, moduleMapping)));


                TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().getTimeIn
Millis());
```

```
                        CloudSim.startSimulation();

                        CloudSim.stopSimulation();

                        Log.printLine("VRGame finished!");
                } catch (Exception e) {
                        e.printStackTrace();
                        Log.printLine("Unwanted errors happen");
                }
        }

        /**
         * Creates the fog devices in the physical topology of the simulation.
         * @param userId
         * @param appId
         */
        private static void createFogDevices(int userId, String appId) {
                FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000, 0,
0.01, 16*103, 16*83.25);
                cloud.setParentId(-1);
                fogDevices.add(cloud);
                FogDevice proxy = createFogDevice("proxy-server", 2800, 4000, 10000,
10000, 1, 0.0, 107.339, 83.4333);
                proxy.setParentId(cloud.getId());
                proxy.setUplinkLatency(100); // latency of connection between proxy server
and cloud is 100 ms
                fogDevices.add(proxy);
                for(int i=0;i<numOfAreas;i++){
                        addArea(i+"", userId, appId, proxy.getId());
                }
        }

        private static FogDevice addArea(String id, int userId, String appId, int parentId){
                FogDevice router = createFogDevice("a-"+id, 2800, 4000, 1000, 10000, 2,
0.0, 107.339, 83.4333);
                fogDevices.add(router);
                router.setUplinkLatency(2); // latency of connection between router and proxy
server is 2 ms
                for(int i=0;i<numOfCamerasPerArea1;i++){
                        String mobileId = id+"-"+i;
                        FogDevice camera = addCamera(mobileId, userId, appId,
router.getId()); // adding a smart camera to the physical topology. Smart cameras have been
modeled as fog devices as well.
                        camera.setUplinkLatency(2); // latency of connection between camera
and router is 2 ms
                        fogDevices.add(camera);
                }
                router.setParentId(parentId);
                return router;
```

```
	}

	private static FogDevice addCamera(String id, int userId, String appId, int parentId){
		FogDevice camera = createFogDevice("c-"+id, 500, 1000, 10000, 10000, 3, 0,
87.53, 82.44);
		camera.setParentId(parentId);
		Sensor sensor = new Sensor("s-"+id, "CAMERA", userId, appId, new
DeterministicDistribution(CAM_TRANSMISSION_TIME)); // inter-transmission time of
camera (sensor) follows a deterministic distribution
		sensors.add(sensor);
		Actuator ptz = new Actuator("ptz-"+id, userId, appId, "PTZ_CONTROL");
		actuators.add(ptz);
		sensor.setGatewayDeviceId(camera.getId());
		sensor.setLatency(40.0);  // latency of connection between camera (sensor)
and the parent Smart Camera is 1 ms
		ptz.setGatewayDeviceId(parentId);
		ptz.setLatency(1.0);  // latency of connection between PTZ Control and the
parent Smart Camera is 1 ms
		return camera;
	}

	/**
	 * Creates a vanilla fog device
	 * @param nodeName name of the device to be used in simulation
	 * @param mips MIPS
	 * @param ram RAM
	 * @param upBw uplink bandwidth
	 * @param downBw downlink bandwidth
	 * @param level hierarchy level of the device
	 * @param ratePerMips cost rate per MIPS used
	 * @param busyPower
	 * @param idlePower
	 * @return
	 */
	private static FogDevice createFogDevice(String nodeName, long mips,
			int ram, long upBw, long downBw, int level, double ratePerMips,
double busyPower, double idlePower) {

		List<Pe> peList = new ArrayList<Pe>();

		// 3. Create PEs and add these into a list.
		peList.add(new Pe(0, new PeProvisionerOverbooking(mips))); // need to store
Pe id and MIPS Rating

		int hostId = FogUtils.generateEntityId();
		long storage = 1000000; // host storage
		int bw = 10000;

		PowerHost host = new PowerHost(
				hostId,
```

```
                                new RamProvisionerSimple(ram),
                                new BwProvisionerOverbooking(bw),
                                storage,
                                peList,
                                new StreamOperatorScheduler(peList),
                                new FogLinearPowerModel(busyPower, idlePower)
                        );
                List<Host> hostList = new ArrayList<Host>();
                hostList.add(host);
                String arch = "x86"; // system architecture
                String os = "Linux"; // operating system
                String vmm = "Xen";
                double time_zone = 10.0; // time zone this resource located
                double cost = 3.0; // the cost of using processing in this resource
                double costPerMem = 0.05; // the cost of using memory in this resource
                double costPerStorage = 0.001; // the cost of using storage in this
                                                                        // resource
                double costPerBw = 0.0; // the cost of using bw in this resource
                LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not
adding SAN

        // devices by now
                FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
                                arch, os, vmm, host, time_zone, cost, costPerMem,
                                costPerStorage, costPerBw);
                FogDevice fogdevice = null;
                try {
                        fogdevice = new FogDevice(nodeName, characteristics,
                                        new AppModuleAllocationPolicy(hostList), storageList,
10, upBw, downBw, 0, ratePerMips);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                fogdevice.setLevel(level);
                return fogdevice;
        }

        /**
         * Function to create the Intelligent Surveillance application in the DDF model.
         * @param appId unique identifier of the application
         * @param userId identifier of the user of the application
         * @return
         */
        @SuppressWarnings({"serial" })
        private static Application createApplication(String appId, int userId){

                Application application = Application.createApplication(appId, userId);
                /*
                 * Adding modules (vertices) to the application model (directed graph)
                 */
```

```
                application.addAppModule("picture-capture", 10);
                application.addAppModule("slot-detector", 10);

                /*
                 * Connecting the application modules (vertices) in the application model
(directed graph) with edges
                 */
                application.addAppEdge("CAMERA", "picture-capture", 1000, 500,
"CAMERA", Tuple.UP, AppEdge.SENSOR); // adding edge from CAMERA (sensor) to
Motion Detector module carrying tuples of type CAMERA
                application.addAppEdge("picture-capture", "slot-detector",
                                1000, 500, "slots",Tuple.UP, AppEdge.MODULE);
                                // adding edge from Slot Detector to PTZ CONTROL (actuator)
                                application.addAppEdge("slot-detector", "PTZ_CONTROL",
100,
                                28, 100, "PTZ_PARAMS",
                                Tuple.UP, AppEdge.ACTUATOR);
                                application.addTupleMapping("picture-capture", "CAMERA",
"slots",
                                new FractionalSelectivity(1.0));
                                application.addTupleMapping("slot-detector", "slots",
                                "PTZ_PARAMS", new FractionalSelectivity(1.0));
                                final AppLoop loop1 = new AppLoop(new
ArrayList<String>()
                                {{add("CAMERA");
                                add("picture-capture");add("slot-detector");
                                add("PTZ_CONTROL");}});
                                List<AppLoop> loops = new
ArrayList<AppLoop>(){{add(loop1);}};
                                application.setLoops(loops);
                                return application;
                                }
        }
```

6.2.2 Cloud only architecture

```
package org.fog.test.perfeval;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
```

```
import org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.Application;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.Actuator;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDevice;
import org.fog.entities.FogDeviceCharacteristics;
import org.fog.entities.Sensor;
import org.fog.entities.Tuple;
import org.fog.placement.Controller;
import org.fog.placement.ModuleMapping;
import org.fog.placement.ModulePlacementEdgewards;
import org.fog.placement.ModulePlacementMapping;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
import org.fog.utils.distribution.DeterministicDistribution;
public class smartCarParkingCloud {
	static List<FogDevice> fogDevices = new ArrayList<FogDevice>();
	static List<Sensor> sensors = new ArrayList<Sensor>();
	static List<Actuator> actuators = new ArrayList<Actuator>();
	static int numOfAreas = 8;
	static int numOfCamerasPerArea1=4;
	static double CAM_TRANSMISSION_TIME = 5;
	private static boolean CLOUD = true;
	public static void main(String[] args) {
		Log.printLine("Starting simulation with cloud...");
		try {
			Log.disable();
			int num_user = 1; // number of cloud users
			Calendar calendar = Calendar.getInstance();
			boolean trace_flag =false; // mean trace events
			CloudSim.init(num_user, calendar, trace_flag);
			String appId = "dcns"; // identifier of the application
			FogBroker broker = new FogBroker("broker");
			Application application = createApplication(appId, broker.getId());
			application.setUserId(broker.getId());
			createFogDevices(broker.getId(), appId);
			Controller controller = null;
			ModuleMapping moduleMapping =
ModuleMapping.createModuleMapping(); // initializing a module mapping
//			for(FogDevice device : fogDevices){
//				if(device.getName().startsWith("c")){ // names of all Smart
Cameras start with 'm'
```

```
//                                      moduleMapping.addModuleToDevice("picture-
capture", device.getName());  // fixing 1 instance of the Motion Detector module to each
Smart Camera
//                                  }
//                          }
//                          for(FogDevice device : fogDevices){
//                                  if(device.getName().startsWith("a")){ // names of all fog
devices start with 'a'
//                                      moduleMapping.addModuleToDevice("slot-detector",
device.getName());  // fixing 1 instance of the Motion Detector module to each Smart Camera
//                                  }
//                          }
                            //moduleMapping.addModuleToDevice("user_interface", "cloud"); //
fixing instances of User Interface module in the Cloud
                            if(CLOUD){
                                    // if the mode of deployment is cloud-based
                                    moduleMapping.addModuleToDevice("picture-capture",
"cloud"); // placing all instances of Object Detector module in the Cloud
                                    moduleMapping.addModuleToDevice("slot-detector",
"cloud"); // placing all instances of Object Tracker module in the Cloud
                            }

                            controller = new Controller("master-controller", fogDevices, sensors,
                                    actuators);

                            controller.submitApplication(application,
                                    (CLOUD)?(new
ModulePlacementMapping(fogDevices, application, moduleMapping))
                                                                    :(new
ModulePlacementEdgewards(fogDevices, sensors, actuators, application, moduleMapping)));


        TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().getTimeIn
Millis());

                            CloudSim.startSimulation();

                            CloudSim.stopSimulation();

                            Log.printLine("VRGame finished!");
                    } catch (Exception e) {
                            e.printStackTrace();
                            Log.printLine("Unwanted errors happen");
                    }
            }

            /**
             * Creates the fog devices in the physical topology of the simulation.
             * @param userId
             * @param appId
```

```
	 */
	private static void createFogDevices(int userId, String appId) {
		FogDevice cloud = createFogDevice("cloud", 44800, 40000, 100, 10000, 0,
0.01, 16*103, 16*83.25);
		cloud.setParentId(-1);
		fogDevices.add(cloud);
		FogDevice proxy = createFogDevice("proxy-server", 2800, 4000, 10000,
10000, 1, 0.0, 107.339, 83.4333);
		proxy.setParentId(cloud.getId());
		proxy.setUplinkLatency(100); // latency of connection between proxy server
and cloud is 100 ms
		fogDevices.add(proxy);
		for(int i=0;i<numOfAreas;i++){
			addArea(i+"", userId, appId, proxy.getId());
		}
	}

	private static FogDevice addArea(String id, int userId, String appId, int parentId){
		FogDevice router = createFogDevice("a-"+id, 2800, 4000, 1000, 10000, 2,
0.0, 107.339, 83.4333);
		fogDevices.add(router);
		router.setUplinkLatency(2); // latency of connection between router and proxy
server is 2 ms
		for(int i=0;i<numOfCamerasPerArea1;i++){
			String mobileId = id+"-"+i;
			FogDevice camera = addCamera(mobileId, userId, appId,
router.getId()); // adding a smart camera to the physical topology. Smart cameras have been
modeled as fog devices as well.
			camera.setUplinkLatency(2); // latency of connection between camera
and router is 2 ms
			fogDevices.add(camera);
		}
		router.setParentId(parentId);
		return router;
	}

	private static FogDevice addCamera(String id, int userId, String appId, int parentId){
		FogDevice camera = createFogDevice("c-"+id, 500, 1000, 10000, 10000, 3, 0,
87.53, 82.44);
		camera.setParentId(parentId);
		Sensor sensor = new Sensor("s-"+id, "CAMERA", userId, appId, new
DeterministicDistribution(CAM_TRANSMISSION_TIME)); // inter-transmission time of
camera (sensor) follows a deterministic distribution
		sensors.add(sensor);
		Actuator ptz = new Actuator("ptz-"+id, userId, appId, "PTZ_CONTROL");
		actuators.add(ptz);
		sensor.setGatewayDeviceId(camera.getId());
		sensor.setLatency(40.0);  // latency of connection between camera (sensor)
and the parent Smart Camera is 1 ms
		ptz.setGatewayDeviceId(parentId);
```

```
		ptz.setLatency(1.0);  // latency of connection between PTZ Control and the parent Smart Camera is 1 ms
		return camera;
	}

	/**
	 * Creates a vanilla fog device
	 * @param nodeName name of the device to be used in simulation
	 * @param mips MIPS
	 * @param ram RAM
	 * @param upBw uplink bandwidth
	 * @param downBw downlink bandwidth
	 * @param level hierarchy level of the device
	 * @param ratePerMips cost rate per MIPS used
	 * @param busyPower
	 * @param idlePower
	 * @return
	 */
	private static FogDevice createFogDevice(String nodeName, long mips,
			int ram, long upBw, long downBw, int level, double ratePerMips, double busyPower, double idlePower) {

		List<Pe> peList = new ArrayList<Pe>();

		// 3. Create PEs and add these into a list.
		peList.add(new Pe(0, new PeProvisionerOverbooking(mips))); // need to store Pe id and MIPS Rating

		int hostId = FogUtils.generateEntityId();
		long storage = 1000000; // host storage
		int bw = 10000;

		PowerHost host = new PowerHost(
				hostId,
				new RamProvisionerSimple(ram),
				new BwProvisionerOverbooking(bw),
				storage,
				peList,
				new StreamOperatorScheduler(peList),
				new FogLinearPowerModel(busyPower, idlePower)
			);
		List<Host> hostList = new ArrayList<Host>();
		hostList.add(host);
		String arch = "x86"; // system architecture
		String os = "Linux"; // operating system
		String vmm = "Xen";
		double time_zone = 10.0; // time zone this resource located
		double cost = 3.0; // the cost of using processing in this resource
		double costPerMem = 0.05; // the cost of using memory in this resource
		double costPerStorage = 0.001; // the cost of using storage in this
```

```
                                                                        // resource
                double costPerBw = 0.0; // the cost of using bw in this resource
                LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not
adding SAN

        // devices by now
                FogDeviceCharacteristics characteristics = new FogDeviceCharacteristics(
                                arch, os, vmm, host, time_zone, cost, costPerMem,
                                costPerStorage, costPerBw);
                FogDevice fogdevice = null;
                try {
                        fogdevice = new FogDevice(nodeName, characteristics,
                                        new AppModuleAllocationPolicy(hostList), storageList,
10, upBw, downBw, 0, ratePerMips);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                fogdevice.setLevel(level);
                return fogdevice;
        }

        /**
         * Function to create the Intelligent Surveillance application in the DDF model.
         * @param appId unique identifier of the application
         * @param userId identifier of the user of the application
         * @return
         */
        @SuppressWarnings({"serial" })
        private static Application createApplication(String appId, int userId){

                Application application = Application.createApplication(appId, userId);
                /*
                 * Adding modules (vertices) to the application model (directed graph)
                 */
                application.addAppModule("picture-capture", 10);
                application.addAppModule("slot-detector", 10);

                /*
                 * Connecting the application modules (vertices) in the application model
(directed graph) with edges
                 */
                application.addAppEdge("CAMERA", "picture-capture", 1000, 500,
"CAMERA", Tuple.UP, AppEdge.SENSOR); // adding edge from CAMERA (sensor) to
Motion Detector module carrying tuples of type CAMERA
                application.addAppEdge("picture-capture", "slot-detector",
                                1000, 500, "slots",Tuple.UP, AppEdge.MODULE);
                                // adding edge from Slot Detector to PTZ CONTROL (actuator)
                                application.addAppEdge("slot-detector", "PTZ_CONTROL",
100,
                                28, 100, "PTZ_PARAMS",
```

```
                                Tuple.UP, AppEdge.ACTUATOR);
                                application.addTupleMapping("picture-capture", "CAMERA",
"slots",
                                new FractionalSelectivity(1.0));
                                application.addTupleMapping("slot-detector", "slots",
                                "PTZ_PARAMS", new FractionalSelectivity(1.0));
                                final AppLoop loop1 = new AppLoop(new
ArrayList<String>()
                                {{add("CAMERA");
                                add("picture-capture");add("slot-detector");
                                add("PTZ_CONTROL");}});
                                List<AppLoop> loops = new
ArrayList<AppLoop>(){{add(loop1);}};
                                application.setLoops(loops);
                                return application;
                                }
        }
```

7.TESTING(SCREENSHOTS)

7.1 Output of Real time simulation using MATLAB

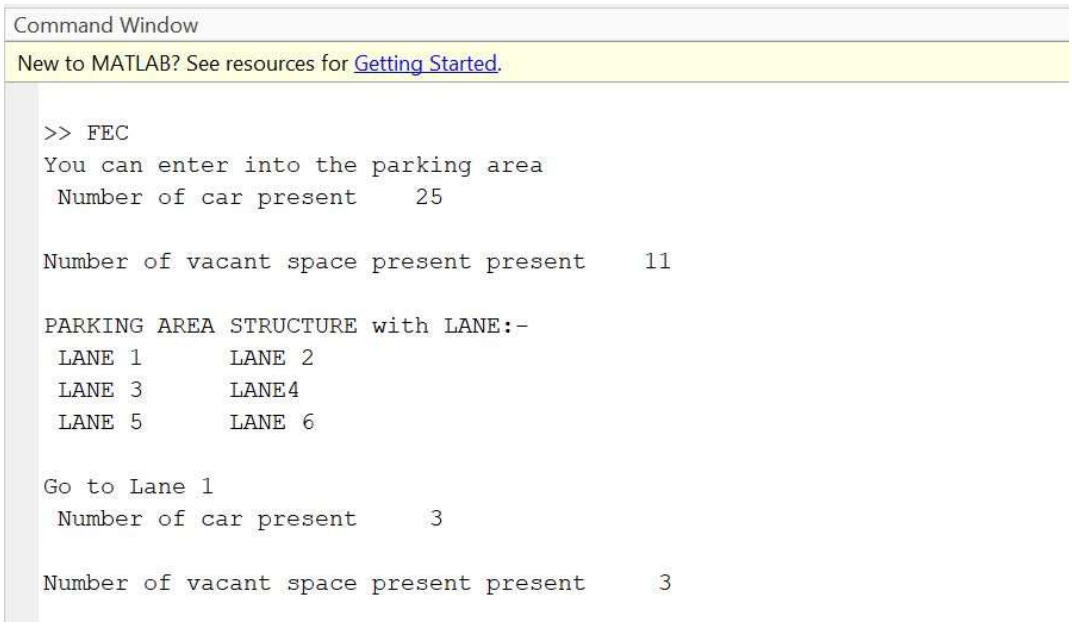

Figure 5a:Description about the given parking lot

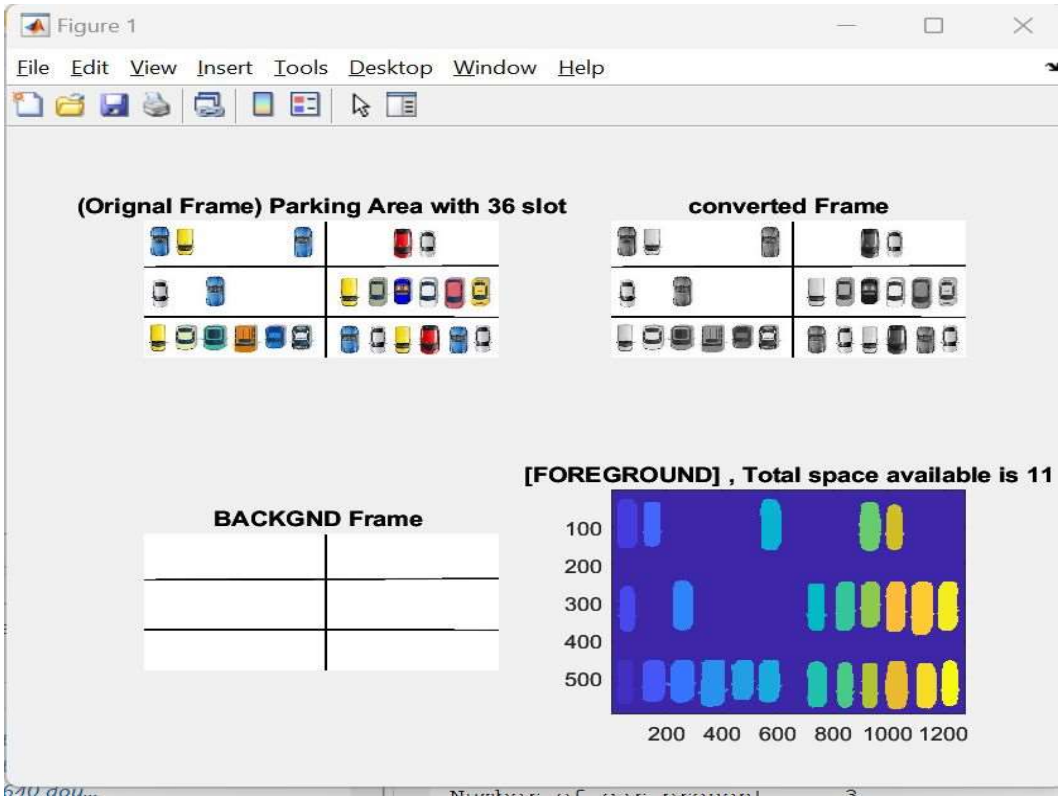

Figure 5b:Conversion of original frame to Black and white

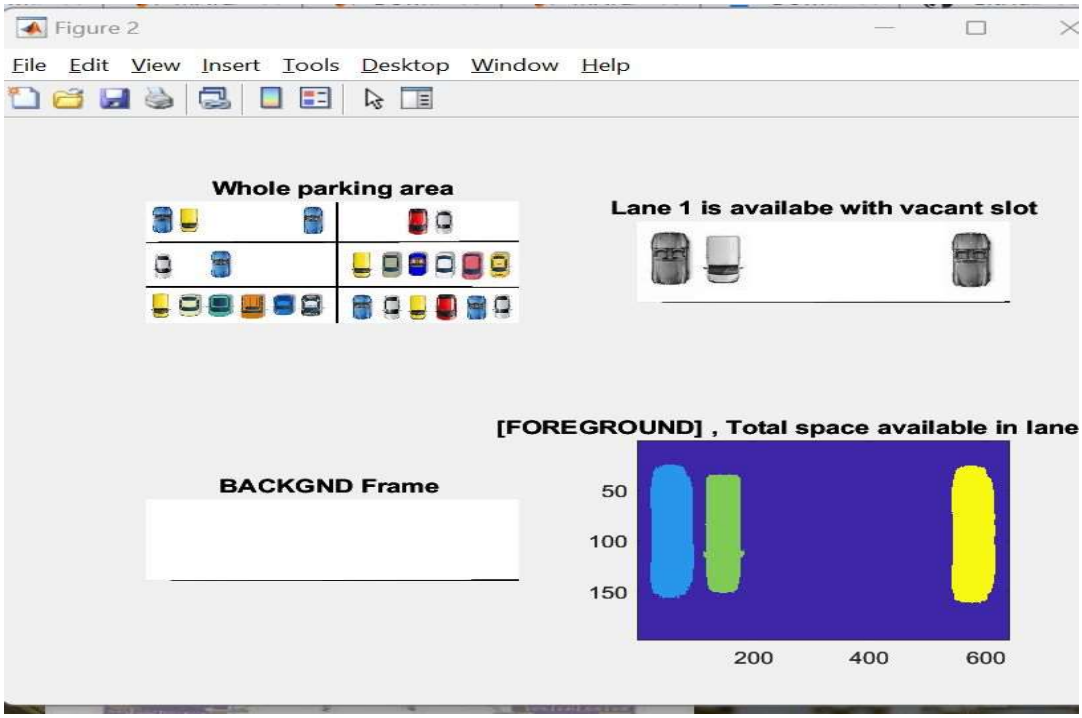

Figure 5c:Output for real time simulation using MATLAB

7.2 Output of implementation using ECLIPSE

Case 1:

In the first case let's examine the architecture with 16 cameras in the edge layer. There are 8 parking areas with 2 cameras each.

7.2.1 Fog only architecture:

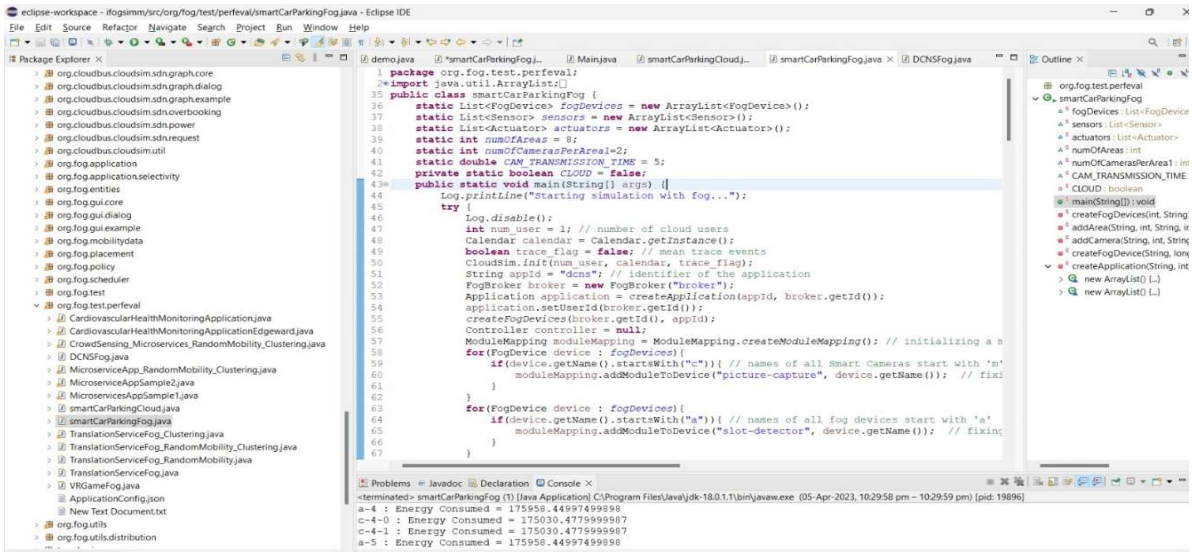

Figure 6:Energy consumed for fog(8 parking areas and 2 cameras each)

Figure 7:Simulation using fog(8 parking areas and 2 cameras each)

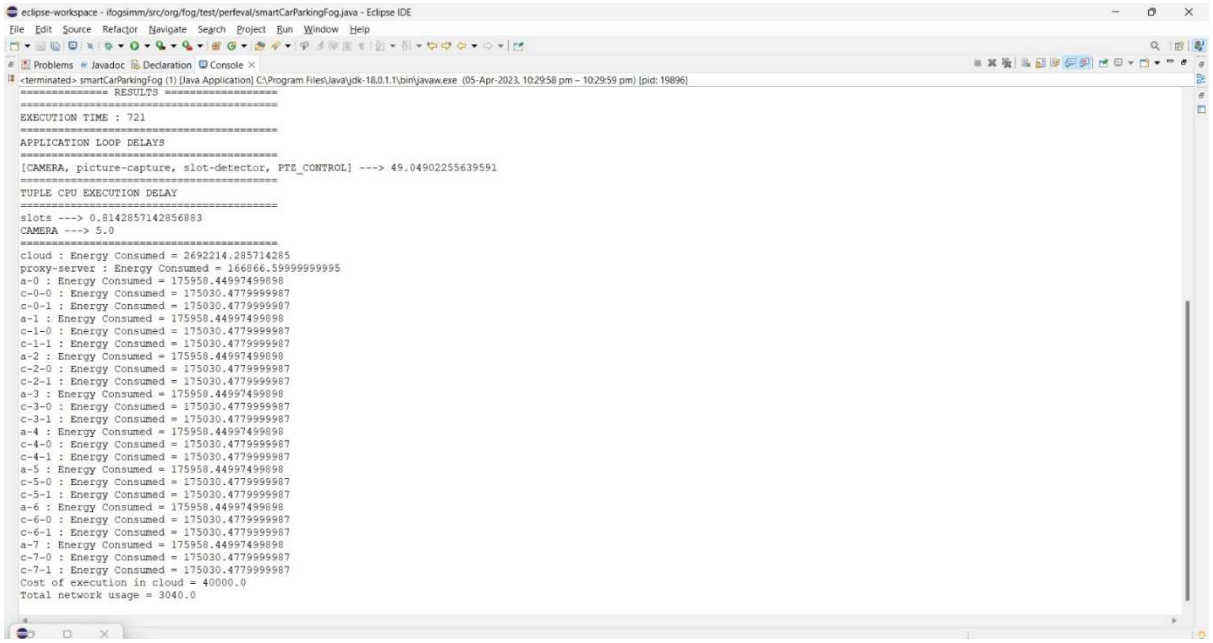

Figure 8:Results of simulation after using fog(8 parking areas and 2 cameras each)

7.2.2 Cloud only architecture

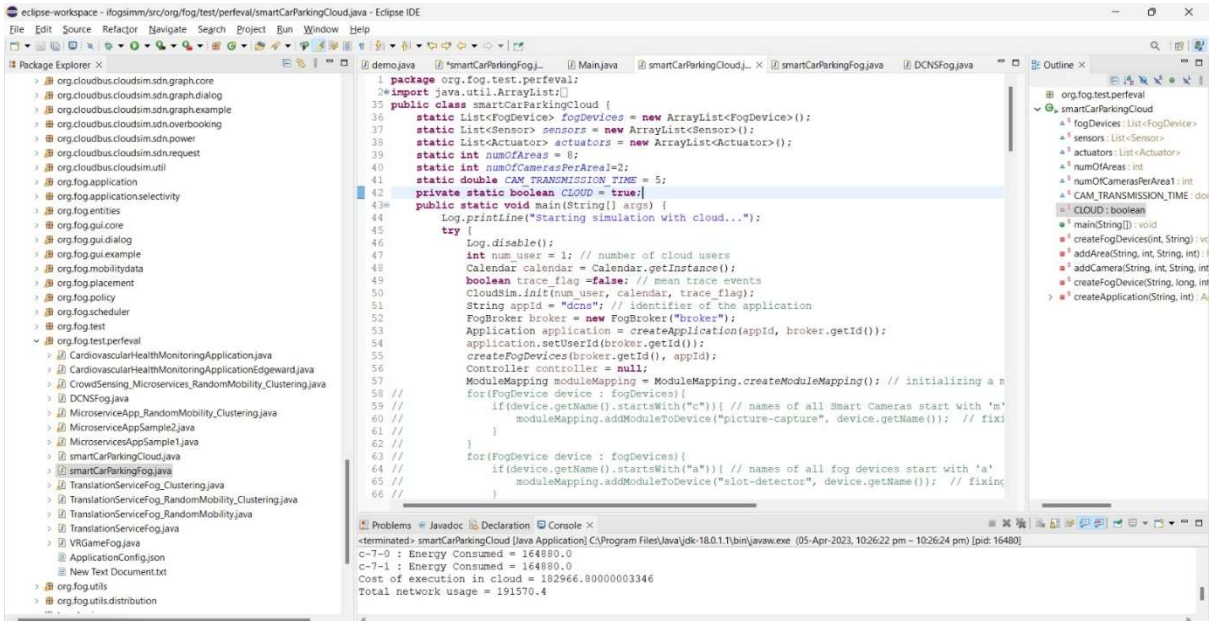

Figure 9:Energy consumed for cloud(8 parking areas and 2 cameras each)

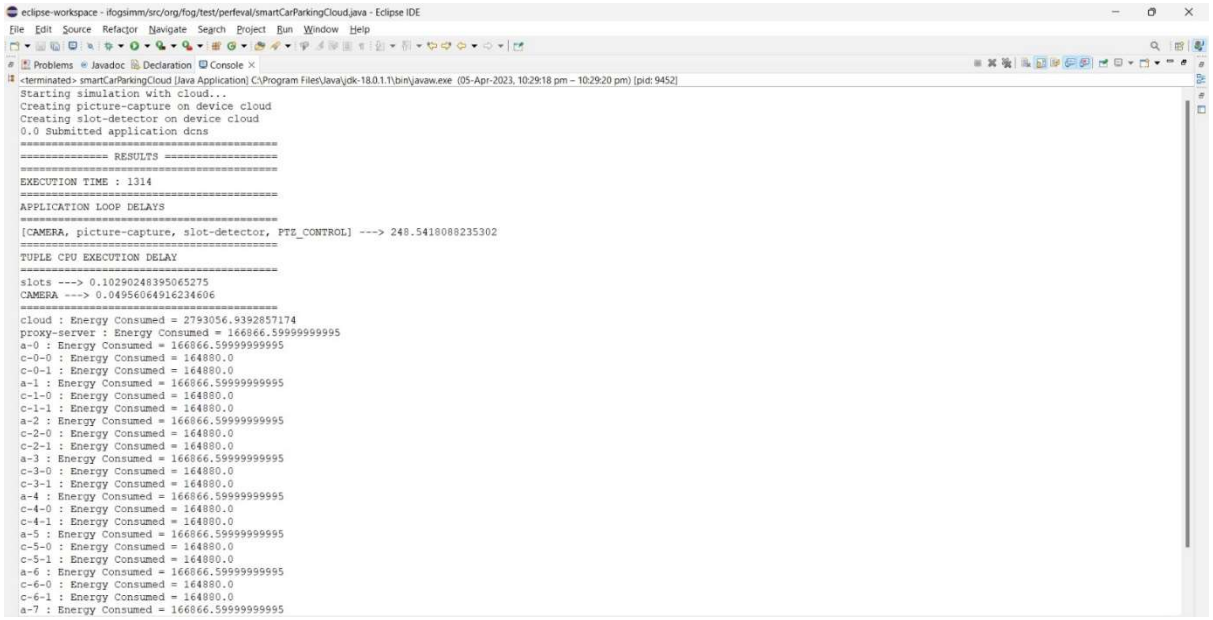

Figure 10:Simulation using cloud(8 parking areas and 2 cameras each)

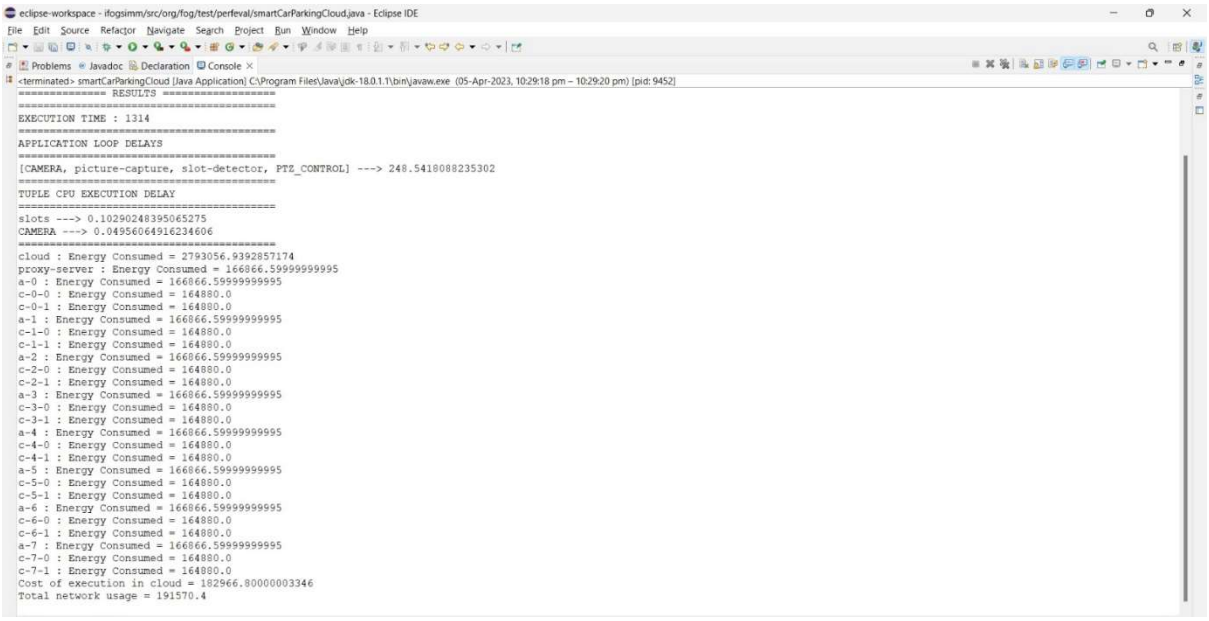

Figure 11:Results of simulation after using cloud(8 parking areas and 2 cameras each)

Case 2:

In the second case let's examine the architecture with 64 cameras in edge layer. There are 8 parking areas with 8 cameras each.

7.2.3 Fog only architecture:

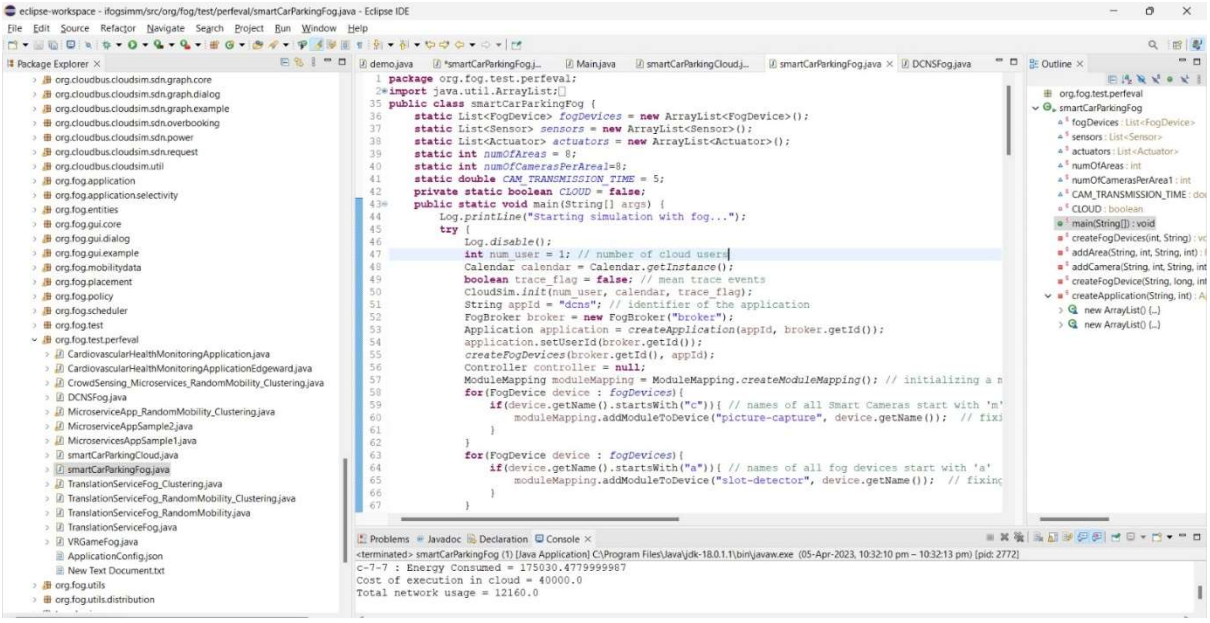

Figure 12:Energy consumed for fog(8 parking areas and 8 cameras each)

Figure 13:Simulation using fog(8 parking areas and 8 cameras each)

Figure 14:Results of simulation after using fog(8 parking areas and 8 cameras each)

7.2.4 Cloud only architecture

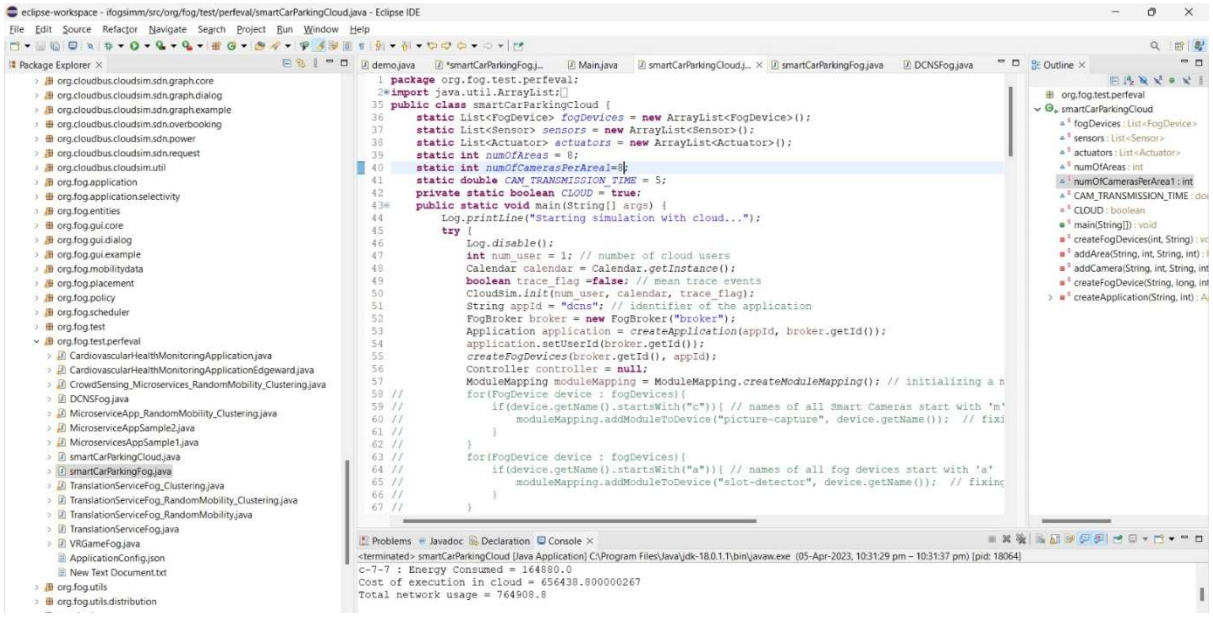

Figure 15:Energy consumed for cloud(8 parking areas and 8 cameras each)

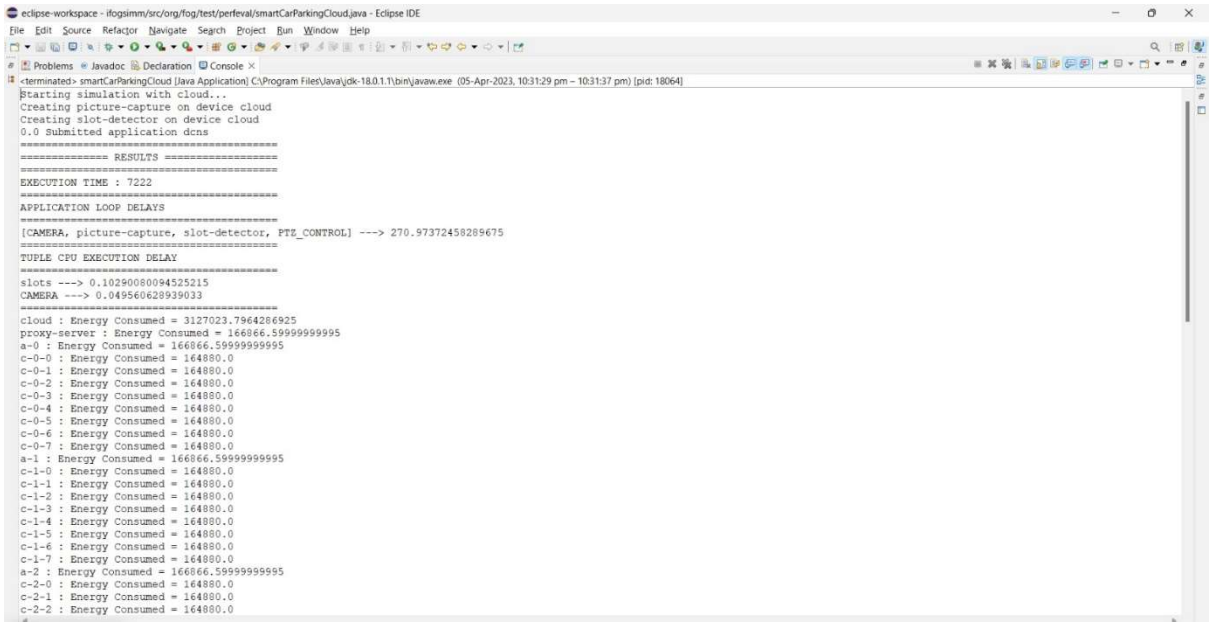

Figure 16:Simulation using cloud(8 parking areas and 8 cameras each)

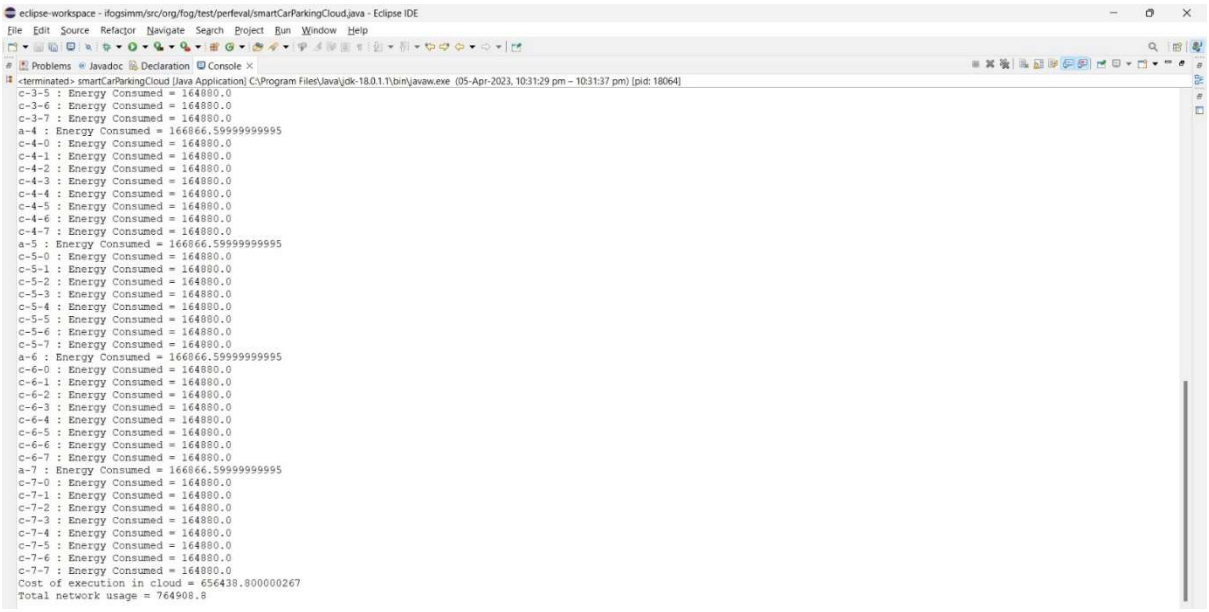

Figure 17:Results of simulation after using cloud(8 parking areas and 8 cameras each)

8. RESULTS AND DISCUSSIONS

Table 1: Comparison of cloud and fog using different parameters

Cameras	**Fog Latency(ms)**	**Cloud latency(ms)**	**Fog network Utilization(KB)**	**Cloud network Utilization(KB)**
16	49.049	253.49	3040	191570
24	49.406	253.84	4560	287198
32	49.76	254.19	6080.0	384829.5
48	50.477	255.105	9120.0	577133.5
64	51.19	312.08	12160	764908

We compared the performance of the two architectures by testing them against different network configurations by changing the number of cameras in the edge layer and arrived with the above results.

9. CONCLUSION AND FUTURE ENHANCEMENTS

Fog computing has emerged as a key technology in recent years, especially for time-sensitive applications. The proliferation of data-generating tools has increased the demand for.

The emphasis on a faster reaction time has also grown. To this purpose, we presented a fog-based smart parking architecture that makes use of computer vision to detect an available parking spot, hence reducing the amount of time and energy cars must spend looking for a spot. The experimental findings show that in comparison to the cloud, the proposed fog-based architecture significantly reduces both latency and network consumption.

The proposed study's reliance on cameras to identify parking spots is a major drawback. Since the photographs would be stored in the cloud, privacy concerns may arise for vehicle owners.

When you think about how much of the processing and storing is done locally on the fog nodes. However, future research could benefit from a focus on ensuring the security of cloud-stored information by using appropriate encryption methods. Furthermore, it is also essential to note that balancing the load on fog nodes will be necessary to ensure efficiency in the large-scale deployment of the suggested framework with the growing number of parking spots. As a result, we want to look into the problems with load balancing in fog nodes and come up with a good solution to them in the future.

10.REFERENCES

[1] Suri, B., Pramanik, P. K. D., & Taneja, S. (2019). 'A hybrid fog architecture: Improving the efficiency in IoT-based smart parking systems. *Recent Adv. Comput. Sci. Commun*

[2] Tandon, R., & Gupta, P. K. (2019). Optimizing smart parking system by using fog computing. In *Advances in Computing and Data Sciences: Third International Conference, ICACDS 2019, Ghaziabad, India, April 12–13, 2019, Revised Selected Papers, Part II 3* (pp. 724-737). Springer Singapore.

[3] Muzakkir Hussain, M., Khan, F., Alam, M. S., & Sufyan Beg, M. M. (2019). Fog computing for ubiquitous transportation applications—A smart parking case study. In *Engineering Vibration, Communication and Information Processing: ICoEVCI 2018, India* (pp. 241-252). Springer Singapore.

[4] Maharjan, A. M. S., & Elchouemi, A. (2020, November). Smart parking utilizing IoT embedding fog computing based on smart parking architecture. In *2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)* (pp. 1-9). IEEE.

[5] Nguyen, S., Salcic, Z., & Zhang, X. (2018, December). Big data processing in fog-smart parking case study. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)* (pp. 127-134). IEEE.

[6] Celaya-Echarri, M., Froiz-Míguez, I., Azpilicueta, L., Fraga-Lamas, P., Lopez-Iturri, P., Falcone, F., & Fernández-Caramés, T. M. (2020). Building decentralized fog computing-based smart parking systems: From deterministic propagation modeling to practical deployment. *IEEE Access*, *8*, 117666-117688.

[7] Hossan, S., & Nower, N. (2020). Fog-based dynamic traffic light control system for improving public transport. *Public Transport*, *12*, 431-454.

[8] Ullah, A., Yao, X., Shaheen, S., & Ning, H. (2019). Advances in position based routing towards ITS enabled FoG-oriented VANET–A survey. *IEEE Transactions on Intelligent Transportation Systems*, *21*(2), 828-840.

[9] Chen, X., Leng, S., Zhang, K., & Xiong, K. (2019). A machine-learning based time constrained resource allocation scheme for vehicular fog computing. *China Communications*, *16*(11), 29-41.

[10] Suri, B., Dutta Pramanik, P. K., & Taneja, S. (2020). An efficient parking solution for shopping malls using hybrid fog architecture. In *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2019, Volume 2* (pp. 313-320). Springer Singapore.