**CLOUD BASED SMART PARKING SYSTEM USING SURVEILLANCE CAMERA**

By

Ashish Kumar Sahu 16BLC1077

Sri Charan B.V.S 16BLC1093

Ajay Marampally 16BLC0176

A project report submitted to

## Dr. Vetrivelan P

**SCHOOL OF ELECTRONICS ENGINEERING**

in partial fulfilment of the requirements for the course of

## CSE4017 – SOFTWARE HARDWARE CO-DESIGN

in

## ech. ELECTRONICS AND COMPUTER ENGINEERING



**VIT UNIVERSITY, CHENNAI**

**Vandalur – Kelambakkam Road Chennai – 600127**

### March 2019

**BONAFIDE CERTIFICATE**

Certified that this project report entitled **“CLOUD BASED SMART PARKING SYSTEM USING SURVEILLANCE CAMERA”** is a bonafide work of **ASHISH SAHU (16BLC1077), B.V.S. SRI CHARAN (16BLC1093) and AJAY MARAMPALLY (16BLC1076)** who carried out the Project work under my supervision and guidance.

### Dr. Vetrivelan P

Associate Professor

School of Electronics Engineering (SENSE), VIT University, Chennai

Chennai – 600 127.

# ABSTRACT

This system is a prototype parking system that Utilizes surveillance cameras to check for vacancy.

Application: The parking area of any mall has a minimum of 1000 slots distributes over 3 floors. Every day, people spend about 15 minutes searching for a vacant slot. These fifteen minutes can be saved if the people know which slots are empty and simple proceed to empty slot

The main aim of our project is to implement a low cost, and reliable parking system.

The main objective of this project is

* To control the parking system from anywhere in the world using Thing Speak, Raspberry Pi and web-cam.
* To be able to control the parking system via Raspberry pi.
* To reduce the workload and human effort.
* To pave the way towards smart technologies.

# ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, Dr. **Vetrivelan P**, Associate Professor, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to the Dean of the School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Programme Chair Prof. **Susan Elias** and Co-Chair Prof. **Reena Monica P** for their support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

# CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| SERIAL NO. |  | NAME | PAGE NO. |
|  |  | ABSTRACT | 3 |
|  |  | ACKNOWLEDGEMENT | 4 |
| **1** |  | INTRODUCTION | 6 |
|  | 1.1 | OBJECTIVES AND GOALS | 6 |
| **2** |  | DESIGN | 7 |
|  | 2.1 | BLOCK DIAGRAM | 7 |
|  | 2.2 | HARDWARE AND SOFTWARE COMPONENTS | 7 |
|  | 2.3 | PROCESS FLOW | 8 |
| **3** |  | PROCEDURE | 9 |
|  | 3.1 | SETTING UP THE RASPBERRY PI | 9 |
|  | 3.2 | SETTING UP THE CAMERA MODULE | 10 |
|  | 3.3 | IMAGE PROCESSING | 11 |
|  | 3.4 | FINDING CO – ORDINATES | 13 |
| **4** |  | SOFTWARE IMPLEMENTATION | 16 |
| **5** |  | APPLICATION AND RESULTS | 27 |
| **6** |  | REFERENCESS | 28 |

* + 1. **INTRODUCTION**

This system is a prototype parking system that Utilizes surveillance cameras to check for vacancy.

Application: The parking area of any mall has a minimum of 1000 slots distributes over 3 floors. Every day, people spend about 15 minutes searching for a vacant slot. These fifteen minutes can be saved if the people know which slots are empty and simple proceed to empty slot.

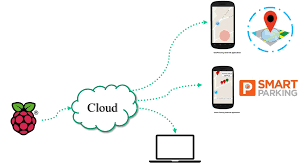
Although systems that produce the same results are available in the market they require infrared sensors in individual slots which adds an additional cost. However, this system utilizes surveillance cameras which are already present and can monitor multiple slots at once, making it more cost effective

### OBJECTIVES AND GOALS

* To control the parking system from anywhere in the world using Thing Speak, Raspberry Pi and web-cam.
* To be able to control the parking system via Raspberry pi.
* To reduce the workload and human effort.
* To pave the way towards smart technologies.

# DESIGN

### BLOCK DIAGRAM



* 1. **HARDWARE COMPONENTS**
     + Raspberry Pi with open-cv installed in it
     + Webcam

**SOFTWARE COMPONENTS**

* + - Thing speak cloud service
    - Telegram application

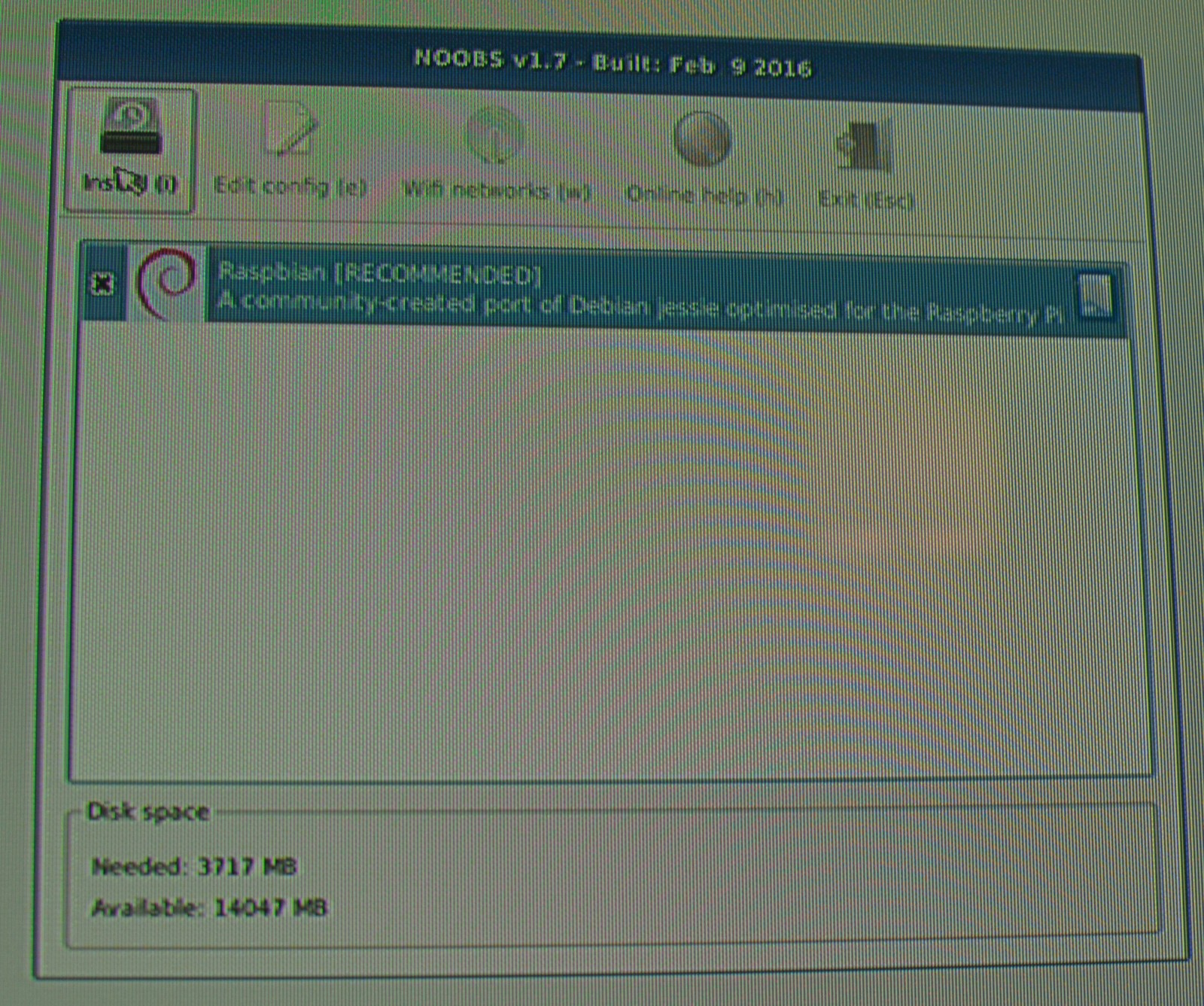
### 2.3 PROCESS FLOW

* The Raspberry is connected to the computer through the Putty a free SSH client software
* The terminal of the raspberry pi is used to execute the program main.py
* Input is provided through the telegram application to the parking bot which communicates with the raspberry pi.
* In this instance the command slot is sent to check the vacant slots in the system.
* The other two commands are the photo and the available slots which returns the snapshot of the system and the available slots in the parking lot .

# PROCEDURE

### SETTING UP RASPBERRY PI

The Noobs OS is installed on the raspberry .In the later stages the Raspbian is installed on the Raspberry pi . The configuration of the Wi-fi module and the other modules is done in the later stages



**Step 2:-**Setting up the camera module for the raspberry pi

The camera module is used to capture the image and read the image to store the image in the png format

Source code:

# -\*- coding: utf-8 -\*-

import cv2

def capture\_img():

camera\_port = 0

ramp\_frames = 30

camera = cv2.VideoCapture(camera\_port)

def get\_image():

retval, im = camera.read()

return im

print("Taking image...")

camera\_capture = get\_image()

file = "process\_img.png"

cv2.imwrite(file, camera\_capture)

#del(camera)

if \_\_name\_\_ == "\_\_main\_\_":

capture\_img()



Step 3: Image Processing

1.Canny Algorithm:

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply [Gaussian filter](https://en.wikipedia.org/wiki/Gaussian_filter) to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by [hysteresis](https://en.wikipedia.org/wiki/Hysteresis) Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Source Code:

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('image\_default.jpeg',0)

edges = cv2.Canny(img,170,255)

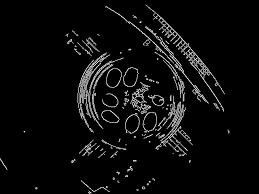
plt.subplot(121),plt.imshow(img,cmap = 'gray')

plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(edges,cmap = 'gray')

plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()



**Step 4:-** Finding the co ordinates

The co ordinates of the parking slot are found using the open-cv library

Source Code:

# -\*- coding: utf-8 -\*-

import cv2

import numpy as np

from matplotlib import pyplot as plt

img\_org = cv2.imread('image\_default.jpeg',0)

edges\_org = cv2.Canny(img\_org,100,200)

#cv2.rectangle(img, (x1, y1), (x2, y2), (255,0,0), 2)

#cv2.rectangle(edges\_org, (20, 35), (230, 210), (255,0,0), 2)

#cv2.rectangle(edges\_org, (8, 240), (230, 430), (255,0,0), 2)

#cv2.rectangle(edges\_org, (410, 195), (605, 20), (255,0,0), 2)

#cv2.rectangle(edges\_org, (415, 230), (630, 410), (255,0,0), 2)

plt.imshow(edges\_org,cmap = 'gray')

plt.title('Check by tracing Cursor'), plt.xticks([]), plt.yticks([])

plt.show()

**Step 5:-** Design of the rectangle

The parking slots are divided into rectangle shapes of same size

Source Code:

#------------------------Importing Libraries-----------------------------------

import cv2

import numpy as np

from matplotlib import pyplot as plt

from camera import \*

#------------------------------------------------------------------------------

#--------------------------Canny Edge Algorithm--------------------------------

def auto\_canny(image, sigma=0.33):

# compute the median of the single channel pixel intensities

v = np.median(image)

# apply automatic Canny edge detection using the computed median

lower = int(max(0, (1.0 - sigma) \* v))

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

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

# return the edged image

return edged

#------------------------------------------------------------------------------

#-------------------------Count fuction for pixel calculation------------------

def cal\_count(edges,x1,y1,x2,y2):

#global edges

count=0

for y in range(y1,y2):

for x in range(x1,x2):

if edges[y][x] != 0:

#print "[",x,"]","[",y,"] =",edges[x][y]

count+=1

return count

#------------------------------------------------------------------------------

def main():

#pass

#capture\_img() # capturing img

img = cv2.imread('2017-07-02-001321.jpg',0) #import img

edges=auto\_canny(img) #send for canny edges calculation

height = np.size(img, 0)

width = np.size(img, 1)

#print height

#print width

#cv2.rectangle(edges, (20, 35), (230, 210), (255,0,0), 2)

slot\_1=cal\_count(edges, 20, 35, 230, 210)

#cv2.rectangle(edges, (8, 240), (230, 430), (255,0,0), 2)

slot\_2=cal\_count(edges, 8, 240, 230, 430)

#cv2.rectangle(edges\_org, (410, 195), (605, 20), (255,0,0), 2)

slot\_3=cal\_count(edges, 410, 195, 605, 20)

#cv2.rectangle(edges\_org, (415, 230), (630, 410), (255,0,0), 2)

slot\_4=cal\_count(edges, 415, 230, 630, 410)

print slot\_1, slot\_2, slot\_3,slot\_4

#----------------------------main----------------------------------------------

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Step 6:-** Checking the pixel density of the rectangular slots

1.The Pixel density of all the rectangular parking slots is calculated

2.Based on the Pixel density the parking slot is termed as full or empty

# 4.SOFTWARE IMPLEMENTATION

# 1.Telegram:

This library provides a pure Python interface for the [Telegram Bot API](https://core.telegram.org/bots/api). It's compatible with Python versions 2.7, 3.3+ and [PyPy](http://pypy.org/).

In addition to the pure API implementation, this library features a number of high-level classes to make the development of bots easy and straightforward. These classes are contained in the telegram.ext submodule.

Installation:

$ pip install python-telegram-bot==12.0.0b1 --upgrade

See CHANGES.rst for the changelog and make sure to report any bugs you find!

You can install or upgrade the stable python-telegram-bot with:

$ pip install python-telegram-bot --upgrade

Or you can install from source with:

$ git clone https://github.com/python-telegram-bot/python-telegram-bot --recursive

$ cd python-telegram-bot

$ python setup.py install

In case you have a previously cloned local repository already, you should initialize the added urllib3 submodule before installing with:

$ git submodule update --init --recursive

Source-Code:

import telepot

bot = telepot.Bot('768091658:AAFnWx\_BPy2sLmc6W8YhLSkhtBmt\_zNpgHk')

bot.getMe()

{'first\_name': 'Your Bot', 'username': 'YourBot', 'id': 123456789}

###############

import telepot

from pprint import pprint

response = bot.getUpdates()

pprint(response)

[{'message': {'chat': {'first\_name': 'Nick',

'id': 999999999,

'type': 'private'},

'date': 1465283242,

'from': {'first\_name': 'Nick', 'id': 999999999},

'message\_id': 10772,

'text': 'Hello'},

'update\_id': 100000000}]"""

from pprint import pprint

import telepot

from telepot.loop import MessageLoop

def handle(msg):

pprint(msg)

MessageLoop(bot, handle).run\_as\_thread()

# 2.Thing speak:

# Thing Speak is an IoT analytics platform service that allows you to aggregate, visualize and analyze live data streams in the cloud.

# Thing Speak provides instant visualizations of data posted by your devices to Thing Speak. With the ability to execute MATLAB® code in Thing Speak you can perform online analysis and processing of the data as it comes in.

# Thing Speak is often used for prototyping and proof of concept IoT systems that require analytics.

Thing Speak allows you to aggregate, visualize and analyse live data streams in the cloud. Some of the key capabilities of Thing Speak include the ability to:

* Easily configure devices to send data to Thing Speak using popular IoT protocols.
* Visualize your sensor data in real-time.
* Aggregate data on-demand from third-party sources.
* Use the power of MATLAB to make sense of your IoT data.
* Run your IoT analytics automatically based on schedules or events.
* Prototype and build IoT systems without setting up servers or developing web software.

# Source-code:

# import urllib.request as urllib

# import re

# from time import sleep

# #define send1():

# 

# #data=urllib.urlopen("https://api.thingspeak.com/update?api\_key=EDLC8KHZZ4EV4KKZ&field1="+str(900));

# #print data;

# f1=0

# f2=2

# """

# datafromwebsite1=urllib.urlopen("https://api.thingspeak.com/update?api\_key=70HJS9ECYJCTT7X2&field1=1");

# datafromwebsite2=urllib.urlopen("https://api.thingspeak.com/update?api\_key=70HJS9ECYJCTT7X2&field2=1");

# datafromwebsite3=urllib.urlopen("https://api.thingspeak.com/update?api\_key=70HJS9ECYJCTT7X2&field3=1");

# #sleep(1)

# """

# datafromwebsite4=urllib.urlopen("https://api.thingspeak.com/update?api\_key=70HJS9ECYJCTT7X2&f"+'&field1=%s&field2=%s' % (f1, f2));

# """select=repr(datafromwebsite.read());

# select=select[300:];

# pick=re.search('field1":"(.+?)",',select);

# if pick:

# print (pick.group(1));"""

# OUTPUT:

# A main driver code is written for the simulation of the function of all the modules connected to the raspberry pi

# Source Code:

#-----------------------------import lib---------------------------------------

import telepot

import time

import cv2

import numpy as np

from time import sleep

import urllib.request as urllib

import re

#------------------------------------------------------------------------------

#---------------------------Camera Image Take---------------------------------

def capture\_img():

camera\_port = 0

ramp\_frames = 30

camera = cv2.VideoCapture(camera\_port)

def get\_image():

retval, im = camera.read()

return im

print("Taking image...")

camera\_capture = get\_image()

file = "image\_default.jpeg"

cv2.imwrite(file, camera\_capture)

#------------------------------------------------------------------------------

#--------------------------Canny Edge Algorithm--------------------------------

def auto\_canny(image, sigma=0.33):

# compute the median of the single channel pixel intensities

v = np.median(image)

# apply automatic Canny edge detection using the computed median

lower = int(max(0, (1.0 - sigma) \* v))

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

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

# return the edged image

return edged

#------------------------------------------------------------------------------

#-------------------------Count fuction for pixel calculation------------------

def cal\_count(edges,x1,y1,x2,y2):

#global edges

count=0

for y in range(y1,y2):

for x in range(x1,x2):

if edges[y][x] != 0:

#print "[",x,"]","[",y,"] =",edges[x][y]

count+=1

return count

#------------------------------------------------------------------------------

def solt\_info():

img = cv2.imread('image\_default.jpeg',0) #import img

edges=auto\_canny(img) #send for canny edges calculation

#height = np.size(img, 0)

#width = np.size(img, 1)

#cv2.rectangle(edges, (98, 117),( 214, 191), (255,0,0), 2)

slot\_1=cal\_count(edges, 98, 117, 214, 191)

#cv2.rectangle(edges, (71, 257), (204, 360), (255,0,0), 2)

slot\_2=cal\_count(edges, 71, 257, 204, 360)

#cv2.rectangle(edges\_org, (381, 105),( 538, 190), (255,0,0), 2)

slot\_3=cal\_count(edges, 381, 105, 538, 190)

#cv2.rectangle(edges\_org, (383, 247),( 553, 346), (255,0,0), 2)

slot\_4=cal\_count(edges, 383, 247, 553, 346)

#rec4=cv2.rectangle(edges\_org, (383, 247),( 553, 346), (255,0,0), 2)

#cv2.imshow('rectangle4',rec4)

slots = [slot\_1,slot\_2,slot\_3,slot\_4]

for s in slots:

if s > 50:

slots[slots.index(s)] = "Not Available"

else:

slots[slots.index(s)] = "Available"

return slots

#-----------------------send\_slot info------------------------------------------

def send\_txt(chat\_id, msg):

bot.sendMessage(chat\_id, msg)

#-----------------------send\_slot\_photo----------------------------------------

def send\_photo(chat\_id, photo, caption=None):

msg1 = bot.sendPhoto(chat\_id=chat\_id, photo=open(photo, "rb"))

#file\_id = msg.photo[0].file\_id

#bot.sendPhoto(photo=file\_id)

"""

with open(photo, mode='r') as f:

bot.sendPhoto(chat\_id,f, caption)

cv2.imshow('image',img)

k=cv2.waitKey(0)

if k==27:

cv2.destroyAllWindows()

elif k==ord('s'):

cv2.imwrite('image.jpeg',img)

cv2.destroyAllWindows()

with open(photo, mode='r') as f: #mode='r'

cv2.imshow('image',f)

#bot.sendPhoto(chat\_id,f, caption)

"""

#-------------------------input masg handle loop -------------------------------

def handle(msg):

print( msg)

try:

print("Enter")

#date = msg['date']

#usr\_name = msg['from']['username']

chat\_id = msg['chat']['id']

#msg\_id = msg['message\_id']

command = msg['text']

print(chat\_id)

print(command)

except:

print("Unexpected msg")

pass

else:

print('Got command: ', command)

if command == "photo":

#send\_photo(chat\_id, "t.png")

try:

capture\_img()

except:

print ("fail to capture Image")

else:

send\_photo(chat\_id, "image\_default.jpeg")

if command == "slots":

try:

f1=0

f2=0

f3=0

f4=0

capture\_img()

time.sleep(1)

slots = solt\_info()

#print(slots[0])

if(slots[0]=="Not Available"):

f1=1

if(slots[1]=="Not Available"):

f2=1

if(slots[2]=="Not Available"):

f3=1

if(slots[3]=="Not Available"):

f4=1

datafromwebsite4=urllib.urlopen("https://api.thingspeak.com/update?api\_key=70HJS9ECYJCTT7X2&f"+'&field1=%s&field2=%s&field3=%s&field4=%s' % (f1, f2,f3,f4));

except:

print("Fail to send slots info")

else:

msg = "Parking Slots : \n" +"Slot-1: "+slots[0]+" Rs 50/hr"+"\n" + "Slot-2: "+slots[1]+" Rs 70/hr"+"\n"+ "Slot-3: "+slots[2]+" Rs 30/hr"+"\n"+ "Slot-4: "+slots[3]+" Rs 50/hr"

send\_txt(chat\_id, msg)

if command == "available slots":

try:

capture\_img()

time.sleep(1)

slots = solt\_info()

except:

print("Fail to send slots info")

else:

msg = "Available Slots \n"

count=0

for s\_no, s in enumerate(slots, start=1):

if s == "Available":

msg +="slot: "+ str(s\_no) +"\n"

count+=1

if count == 0:

msg = "Parking Full"

send\_txt(chat\_id, msg)

#commands(chat\_id, msg\_id, date, command, usr\_name, first\_name, last\_name)

#-------------------------Digi-server-bot config -------------------------------

bot = telepot.Bot('768091658:AAFnWx\_BPy2sLmc6W8YhLSkhtBmt\_zNpgHk')

bot.message\_loop(handle)

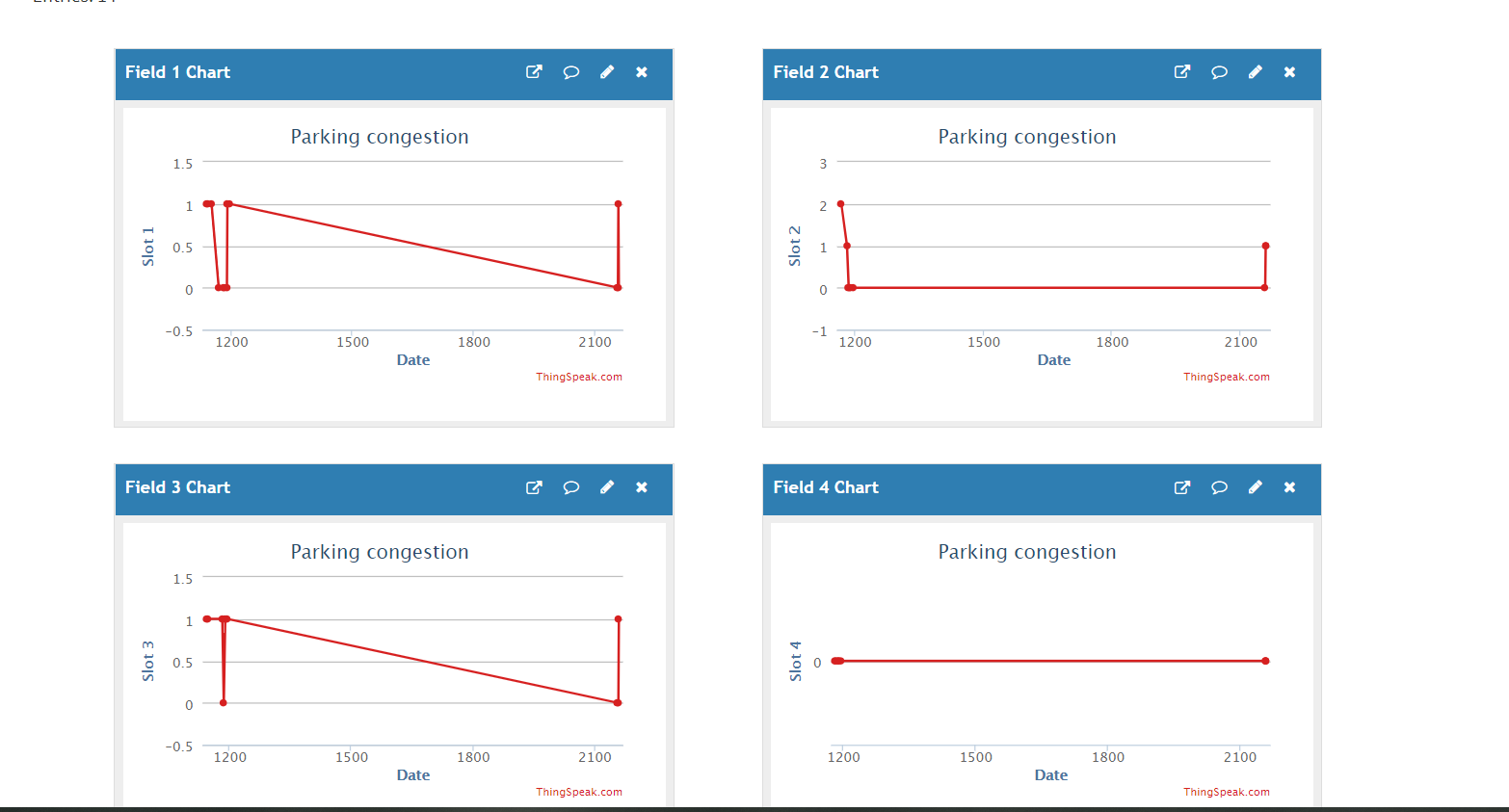
print ('I am listening...')

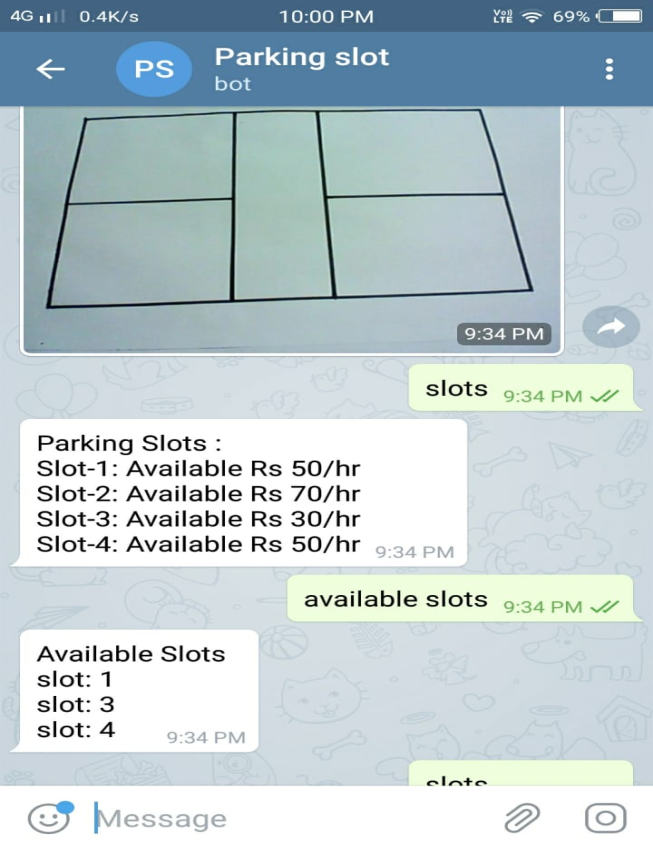
while 1:

time.sleep(1)

**OUTPUTS:**

**1.Cloud:**

****



2.

# 

# 3.

# 

**5.APPLICATIONS**

* + - The range of this project is a lot. So, the parking system can be used for billing the users based on the fare and other applications by improvising it a bit.
    - The project can be used for remote actuation and hardware can be modified according to need.
    - It decreases human effort to do a work by just making it do work by human commands through a computer.
    - It has a wide variety of applications and leads to the formation of smart de which can thereby lead to smart homes and thereby smart cities.

# 6.REFERENCES

1. <https://thingspeak.com/pages/learn_more>
2. <https://github.com/python-telegram-bot/python-telegram-bot>
3. <https://www.raspberrypi.org/forums/viewtopic.php?t=185134>
4. <https://en.wikipedia.org/wiki/Canny_edge_detector>