Report

On

Central Command of smart city

Submitted by

Central command team

Under the Guidance of

Prof KV Gangadharan

Dept. of Mechanical Engineering,

NITK, Surathkal

Date of Submission: 27 April, 2019

Department of Mechanical Engineering
National Institute of Technology Karnataka, Surathkal.

2019-2020

Abstract

The smart city project is a part of ME869 Theory and Practice of sensors and actuators course

Contents

1 Objective

The objective of central command team is to integrate and have a central control over the functions and working of other teams and display the respective task performed on the website running simultaneously on the display screen.

2 Hardware used

Raspberry Pi 3 Model B+ was used as the hardware which acts like a central server receiving instructions, processing them and sending the necessary control action.

3 Software

For the central command:

- Python

For the website:

- Flask (a web microframework for building backend services using python)
- HTML & CSS
- Javascript & JQuery
- Materialise CSS as a CSS framework
- Firestore as the database

4 Hardware specifications

5 Usage & application: Integration approach

- The Raspberry Pi is used as a central server where the image processing code is run in order to identify the current location of the bots.
- Once the location of the bots have been identified, the centroidal data is then sent to the path planning algorithm where the shortest path to the destination node from the current node is identified.
- Once the coordinates of the path is identified, the data is processed as a series of left and right turns which are then sent to the bot via the MQTT protocol which is then executed by the bot and the bot reaches the destination node.

6 Design approaches: Logic & Algorithms

- The main idea is to use a data structure to store the identity of the house and the bot called from the house and execute a certain algorithm based on the data stored and to remove the stored identity once the execution of the task is done.
- Hence, a queue is chosen as the primary data structure.
- A queue is a first in first out (FIFO) kind of data structure where the incoming data is stored and after the execution is removed. Multiple tasks can be stored in a queue but the latest task that has been added gets executed first in that order.

7 Work Completed

- The algorithm for storing the data has been completed. The sequence of call for the various tasks from the different buildings is stored and execution of each task through the medium of bots.
- Communication between the algorithm and the data via the MQTT protocol has been established.
- The path planning sequence is successfully generated and sent to the bots through communication.
- The website with database setup and live feed has been completed.

8 Code with comments

```
import time
import threading
import paho.mqtt.client as paho
from path import path_planning, clear_path

import cv2
import cv2.aruco as aruco
import numpy as np

cap = cv2.VideoCapture(1)
flag = 0

global xref
global yref
global sumx
global sumy
global bx
bx = 0
```

```
global by
by = 0
v1_path = []
nodes_coordinates = np.load('calibration_array.npy')
for i in range(1, len(nodes_coordinates)):
nodes_coordinates[i][0] -= nodes_coordinates[0][0]
nodes_coordinates[i][1] -= nodes_coordinates[0][1]


def bot_detect(posf):
global stop_thread
global bx
global by
flag = -1
source = [1, 1]
previous = []
edge_no = []

def detection_function(image, id_num):
sumx = 0
sumy = 0
i = 0
j = 0

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_5X5_250)
parameters = aruco.DetectorParameters_create()
corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)

markers = aruco.drawDetectedMarkers(frame, corners, ids, (0, 0, 255))
output_Arr = np.array(corners)
centroid_array = np.zeros((len(corners), 2))
if len(corners) > 0:
for i in range(0, len(corners)):
sumx = 0
sumy = 0
for j in range(0, 4):
sumx += output_Arr[i][0][j][0]
sumy += output_Arr[i][0][j][1]
centroid_array[i][0] = sumx / 4
centroid_array[i][1] = sumy / 4

for i in range(0, len(corners)):
cv2.circle(markers, (int(centroid_array[i][0]), int(centroid_array[i][1])), 2, (0, 255,

for i in range(0, len(corners)):
```

```
if id_num == ids[i]:
return (centroid_array[i][0], centroid_array[i][1], markers)

def node_finder(x, y, id_num, node_coord):
temp = 0
for i in range(1, len(node_coord)):
d = 0
d = np.sqrt((x - node_coord[i][0]) ** 2 + (y - node_coord[i][1]) ** 2)
if d < 25:
temp = 1
# print(”haaaaaaaaaaaaa”)
return i

if temp == 0:
return None

while (True):
ret, frame = cap.read()
try:
bx, by, markers = detection_function(frame, 21)
bx = bx - nodes_coordinates[0][0]
by = by - nodes_coordinates[0][1]
flag = flag + 1
except:
print("bot1 not detected")
source_1 = node_finder(bx, by, 1, nodes_coordinates)
print("the bot is in this location:")
source[0] = source_1
need_val = source

print(source[0])
if posf == source[0]:
break
cv2.imshow('frame', frame)

# cv2.imshow(’res’,frame)
print("..............................")
if cv2.waitKey(1) & 0xFF == ord('q'):
break
if posf == need_val[0]:
break
cap.release()
cv2.destroyAllWindows()


global stop_thread
```

```
stop_thread = False
# t1.start()


apartment_bot = ''
hospital_bot = ''
mall_bot = ''
school_bot = ''
data = ''

broker = "iot.eclipse.org"
port = 1883

previous = [2, 2, 2]
source = [1, 1, 1]


# dummy function :)
# def pathplan(c1, c2):  file from image processing team
# co = []
# return co
#  let Building nodes named as A,b,c,d. and bots coordinates


class Queue:  # Each time a button is pressed the status gets stored in this queue

def __init__(self):
self.bot = []
self.building = []
self.prev_emergency = ''
self.prev_garbage = ''
self.prev_delivery = ''

def is_empty(self):
return self.bot == []

def prev(self, a, b):
if b == "emergency":
if a != self.prev_emergency:
self.prev_emergency = a
print("return 1 i.e. prev check tells  NOTcopy")
return 1
elif b == "garbage":
if a != self.prev_garbage:
self.prev_garbage = a
print("return 1 i.e. prev check tells  NOTcopy")
```

```
return 1
elif b == "delivery":
if a != self.prev_delivery:
self.prev_delivery = a
print("return 1 i.e. prev check tells  NOTcopy")
return 1

print("return 0 i.e. prev check tells copy")
return 0

def enqueue(self, x, y):

if self.is_empty():
if (self.prev(y, x)):
print("In empty enqueue done - " + x + y)
self.bot.append(x)
self.building.append(y)
time.sleep(1)
else:
print("Empty queue, redundant messages")
else:
if (x == self.bot[0] and y == self.building[0]) or not (self.prev(y, x)):
print("redundant entry failed")
time.sleep(1)
pass
else:
print("In NON- empty enqueue done - " + x + y)
self.bot.append(x)
self.building.append(y)
time.sleep(1)

def dequeue(self):
self.bot.pop(0)
self.building.pop(0)
print("pop out good")
time.sleep(1)


q = Queue()
a = int(0)


def subscribe(c_id, topic, server, port):
cl = mqtt.Client(c_id)
cl.connect(server, port)
cl.subscribe(topic)
cl.on_message = mess
cl.loop_forever()


def on_publish(client, userdata, result):  # create function for callback
print("data published \n")
pass


# define callback
def on_message(client, userdata, message):
global data
time.sleep(1)
print("received message =", str(message.payload.decode("utf-8")))
data = str(message.payload.decode("utf-8"))


# functions to access data from subscription

def apartment_val(client, userdata, message):
global apartment_bot
apartment_bot = str(message.payload.decode("utf-8"))
print("in callback func - apartement_val")
if apartment_bot == "delivery":
q.enqueue("delivery", "Apartment")
print("enqueued")
elif apartment_bot == "emergency":
q.enqueue("emergency", "Apartment")
print("enqueued")
elif apartment_bot == "garbage":
q.enqueue("garbage", "Apartment")
print("enqueued")
print(apartment_bot)


def hospital_val(client, userdata, message):
global hospital_bot
hospital_bot = str(message.payload.decode("utf-8"))
if hospital_bot == "delivery":
print("To enqueue func")
q.enqueue("delivery", "Hospital")
elif hospital_bot == "emergency":
q.enqueue("emergency", "Hospital")
print("To enqueue func")
elif hospital_bot == "garbage":
q.enqueue("garbage", "Hospital")
```