AUTONOMOUS SURVEILLANCE BOT

- Aaresh Bachana
- Abhilash Manjunath
- Harshil Sheth
- Shrivathsa Murthy

Executive Summary:

Security at workplaces has been one of the biggest obstacle for company progress. In 2010, U. S. retailers and businesses lost about $33 billion in revenue due to theft[1]. This project could prove to be a boon for such companies as it provides a potential solution to this so-called theft problem.

The fundamental idea is to use face recognition to capture facial images present in the surrounding and compare the captured facial features with the ones present in the database. This ensures that an unknown face is never allowed inside the office premises. An alarm is triggered each time the system detects an unknown face.

To achieve this, we used two Logitech C200 cameras that are used for two processes: face capturing and path finding. Additionally, the bot also consists of two ultrasonic sensors that are used to detect sudden obstacles to avoid damage to the Bot. The entire configuration is mounted on a simple robotic car. The project uses the NVIDIA JETSON TK1 as the base board.

This report provides a detailed description of the project and the test cases implemented to ensure robustness of the project. It also discusses the issues that we faced throughout the course of the project.

Introduction:

The purpose of an Autonomous surveillance bot is to navigate through a path by avoiding obstacles and to ensure it reaches its destination in the best route possible while constantly monitoring its surroundings using sensors and visual surveillance techniques.

In this project the visual navigation method guided by path boundaries is presented, boundaries are detected and located by visual cue, and the controls parameters are calculated according to boundaries for the autonomous navigation of the bot. It is based on a NVIDIA Jetson TK1 dev board with multiple cameras, ultrasonic sensors and a GPS interfaced to it. This entire system is portable and gives information to the bot about routes and informs what decisions to make.

The proposed bot detects the nearest obstacle via a sonar system and sends back echo feedback to inform the bot about its localization. [2] Surveillance techniques involves Face recognition, which begins with extracting the coordinates of features such as width of mouth, width of eyes, pupil, and compare the result with the measurements stored in the database and return the closest record (facial metrics). The best facial recognition algorithm (Fisherface) provided by the Open CV 2.4.8 was implemented in the autonomous system. [3]

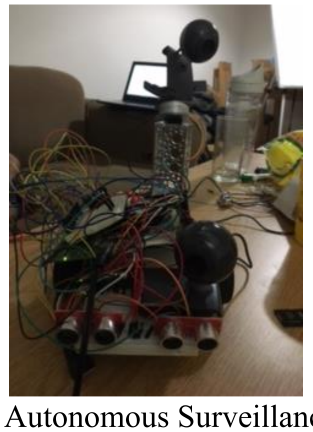


Fig 1. Autonomous Surveillance Bot

Functional Requirements:

The basic function performed by the autonomous bot was to detect the presence of an unknown person in the premises. Therefore, face recognition and facial learning were two fundamental requirements in terms of the functionality. These functionalities along with other secondary functionalities are explained in the table below. Along with these requirements, the most important requirement was to schedule the threads such that the real-time deadline for each thread is met.

Functional Requirements	Description	Status
Ability to learn faces	The system should possess the ability to learn from several input images, that is, extract facial data from the image and make the algorithm learn the images for an updated database.	Completed
Ability to recognize faces	The algorithm should be able to recognize the detected image and give a low confidence value on detecting an image which matches to its database and a high confidence value (typically >5000) for an unknown face	Completed
Ability to avoid collision with an obstacle	The system uses ultrasonic sensors to avoid obstacles that appear suddenly to ensure that the bot is protected from crashing at all times.	Completed
Ability to control the bot's motors depending on the obstacles detected	The motor must take prompt action on detecting an obstacle and change the path of the bot by controlling the motors	Completed
Ability to find the best possible path in the forward direction	The software must use the camera interfaced to the system to be able to extract the empty road part from an image in the forward direction	Completed
Ability to change the bot path depending on the best path found	The system must be able to control the motor per the x-y coordinates received from the path finding function to ensure that the best path is taken all the time	Completed
Ability to assist the bot to navigate from current position to the position given by user	Obtain navigation instructions by querying the google API from the current location until the destination is reached	Completed
Task Synchronization	Used semaphores between threads to synchronize their occurrences	Completed
Failure Mechanism	If the system misses a deadline several times, the system should reboot itself and begin running normally	Completed

Real-Time Requirements:

The whole purpose of doing the project was to understand the real-time requirements of any application and develop firmware considering the response time of a system. Thus, the real-time requirements were decided with careful analysis and understanding of the system performance.

Real - Time Requirements	Description	Status
Motor – Control should have the highest priority	The system design should be such that the motors control request should be services as soon as it arrives to ensure that the bot stops on time in case of a sudden obstacle appearing before it	Completed
Navigation using GPS coordinates should have the lowest priority	The GPS navigation thread is programmed to be of the lowest priority because it is acceptable to miss a few deadlines (not receive co-ordinates in the first pass) considering the application design	Completed
Both the Image processing threads may have medium priority	The image capture and processing threads are programmed to be of medium priority because missing some frames may not be as lethal to the system as missing an obstacle detection event and colliding into it	Completed
Ultrasonic sensor thread with a high priority	The ultrasonic sensors should also be given a very high priority because their readings have to be recorded and acted upon as soon as an obstacle is detected	Completed
Complete face detection and recognition in a fixed time	The face detection and the face recognition process should be completed within a maximum time of 8000ms	Completed
Update best-path periodically	The path finding algorithm must update the best path forward once every 900ms	Completed
Successfully implementing all five services without missing deadlines	All five services must run correctly in real-time without missing a deadline	Completed

Functional Design:

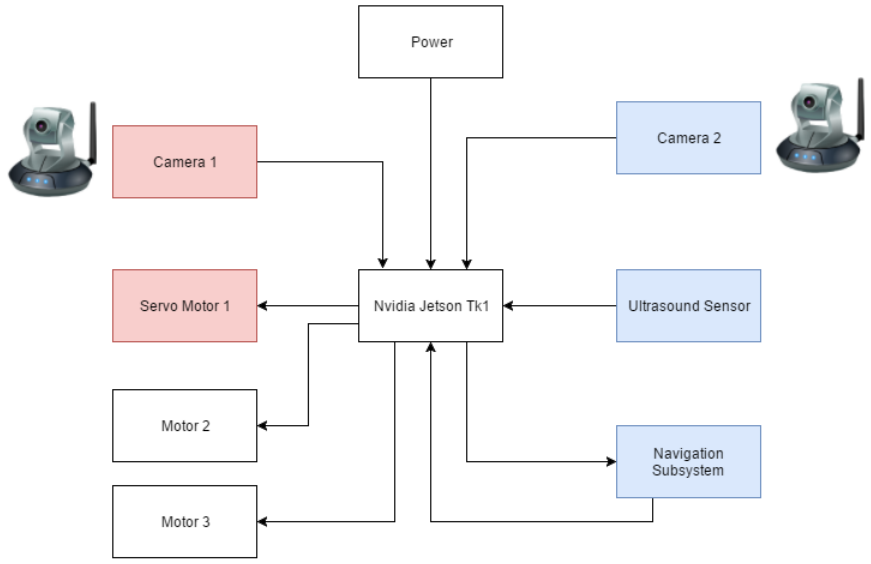


Fig 2. High level block diagram

The block diagram provides an overview of the components used in the project. The cameras and GPS have been interfaced via USB port and the Motors and Ultrasound sensors have been interfaced via the GPIO pins. The cameras used are Logitech C200 and two standard motors are used which are driven by a H-bridge SN754410IC. An extra hardware had to implemented in the project as the Jetson pins provide an output of only 1.8V which is not a valid signal on TTL level. The hardware used was a TTL level converter which converted signals to TTL logic levels.

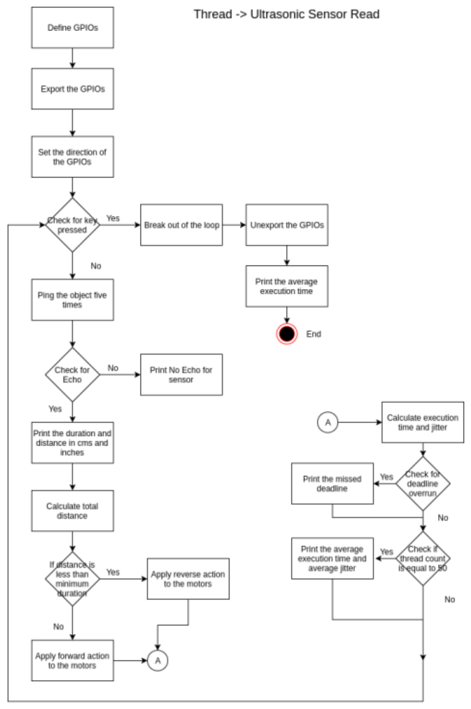

Thread -> Ultrasonic Sensor Read
Define GPIOs
Export the GPIOs
Set the direction of the GPIOs
Check for key pressed
Yes
Break out of the loop
Unexport the GPIOs
Print the average execution time
End
No
Ping the object five times
Check for Echo
No
Print No Echo for sensor
Yes
Print the duration and distance in cms and inches
Calculate total distance
If distance is less than minimum duration
Yes
Apply reverse action to the motors
No
Apply forward action to the motors
A
A
Calculate execution time and jitter
Check for deadline overrun
Yes
Print the missed deadline
No
Check if thread count is equal to 50
Yes
Print the average execution time and average jitter
No

Fig 3. Ultrasonic Sensor read

On boot up of Linux on the Nvidia Jetson, the “capture” executable file is run. On startup of the program, the threads are spawned and start running. A constant monitoring of the ultrasound sensor data is maintained to avoid obstacles as this is the highest priority task. As it can be seen from the above block diagram, a sophisticated algorithm was developed to get a good accuracy of the data from the sensors. The senor would ping the objects in front of it at least 5 times and the duration of the echo would be recorded in an array. The median of this array is taken to get the accurate positioning/location of the object detected by the sensor. Using this value the distance was calculated and commands were given to the motors.

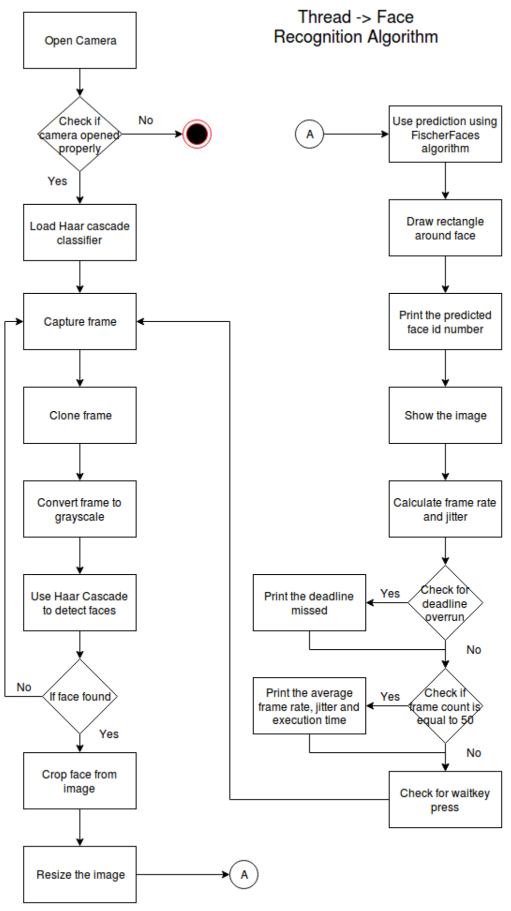

Thread -> Face Recognition Algorithm
Open Camera
Check if camera opened properly
No
Yes
Load Haar cascade classifier
Capture frame
Clone frame
Convert frame to grayscale
Use Haar Cascade to detect faces
If face found
No
Yes
Crop face from image
Resize the image
A
A
Use prediction using FischerFaces algorithm
Draw rectangle around face
Print the predicted face id number
Show the image
Calculate frame rate and jitter
Check for deadline overrun
Yes
Print the deadline missed
No
Check if frame count is equal to 50
Yes
Print the average frame rate, jitter and execution time
No
Check for waitkey press

Fig 4. Face recognition algorithm

The face recognition algorithm and Path finder algorithm run in parallel to the Ultrasonic sensor read algorithm. These two are very crucial to the functioning of the robot as the direction of the bot and movement of the motors are dependent on these threads. The face recognition function captures a frame and uses Haar cascade filter to detect faces from the grayscale image. The face detection and learning process is achieved using the fisherface algorithm. The frame rate and jitter for each frame is calculated and displayed using a print function.

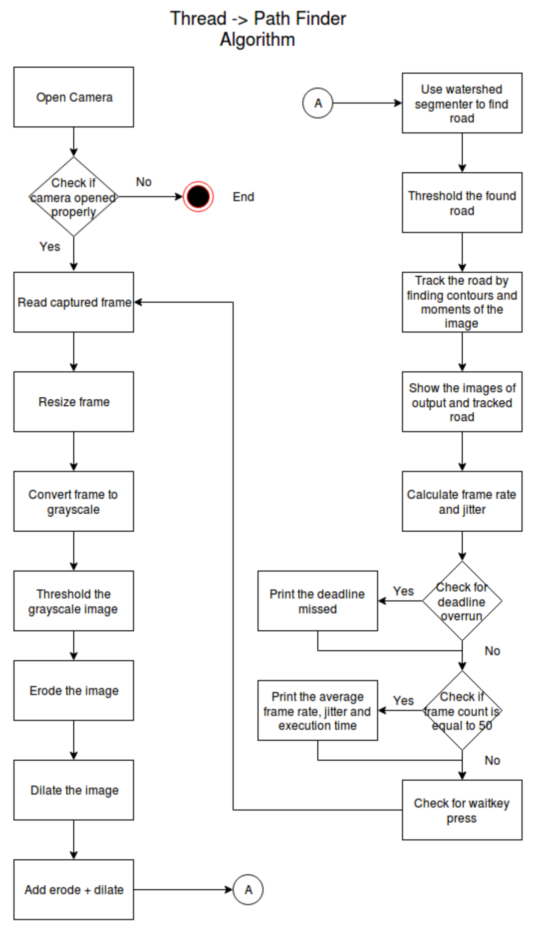

Thread -> Path Finder Algorithm
Open Camera
Check if camera opened properly
No
End
Yes
Read captured frame
Resize frame
Convert frame to grayscale
Threshold the grayscale image
Erode the image
Dilate the image
Add erode + dilate
A
Use watershed segmenter to find road
Threshold the found road
Track the road by finding contours and moments of the image
Show the images of output and tracked road
Calculate frame rate and jitter
Check for deadline overrun
Yes
Print the deadline missed
No
Check if frame count is equal to 50
Yes
Print the average frame rate, jitter and execution time
No
Check for waitkey press

Fig 5. Path Finder algorithm

Similar to the face recognition algorithm, the path finding algorithm captures an image and converts it to its grayscale form. The image is then dilated and eroded to improve accuracy. Using a greyscale threshold slider, the output image is filtered such that an empty road forms the largest white portion on the image. The area and the center of this portion is calculated, displayed and checked to control the position of the motor.

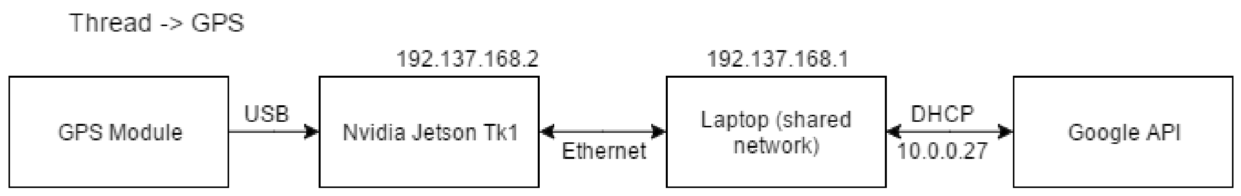


Fig 6. GPS algorithm

The GPS module is connected to the NVIDIA Jetson via a USB and uses the laptop to generate queries to the Google API for coordinates.

Real Time Analysis and Design:

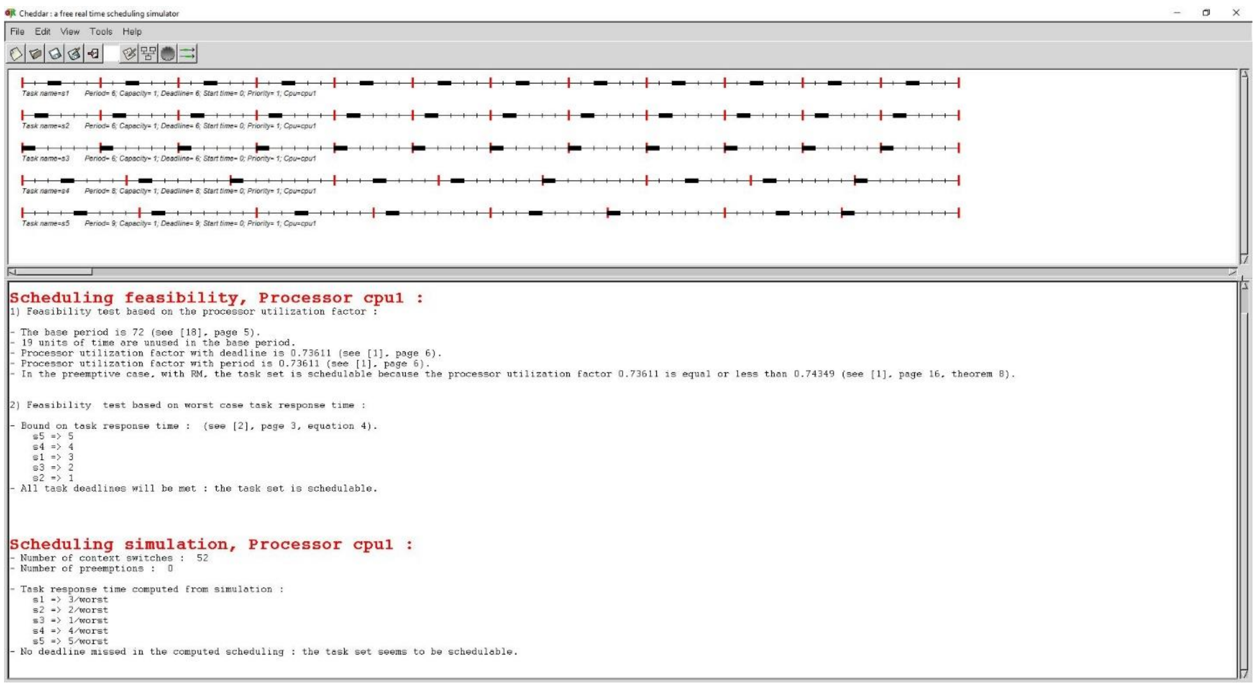
Thread	Average Execution Time (ms)	Deadline (ms)	Request Time (In ms)	Request Frequency
Path Finder	200	1200	1/6	6
Face Recognition	1500	9000	1/6	6
Ultrasonic Sensor Read	75	450	1/6	6
Motor Control	4000	32000	1/8	8
GPS control	700	6300	1/9	9

Total Utilization (U) = 200/1200 + 1500/9000 + 75/450 + 4000/32000 + 700/6300
= 1/6 + 1/6 + 1/6 + 1/8 + 1/9
= 0.73611

Rate Monotonic Analysis:

RM LUB for 5 services = $m*(2^{1\backslash m} - 1)$
= 0.74349

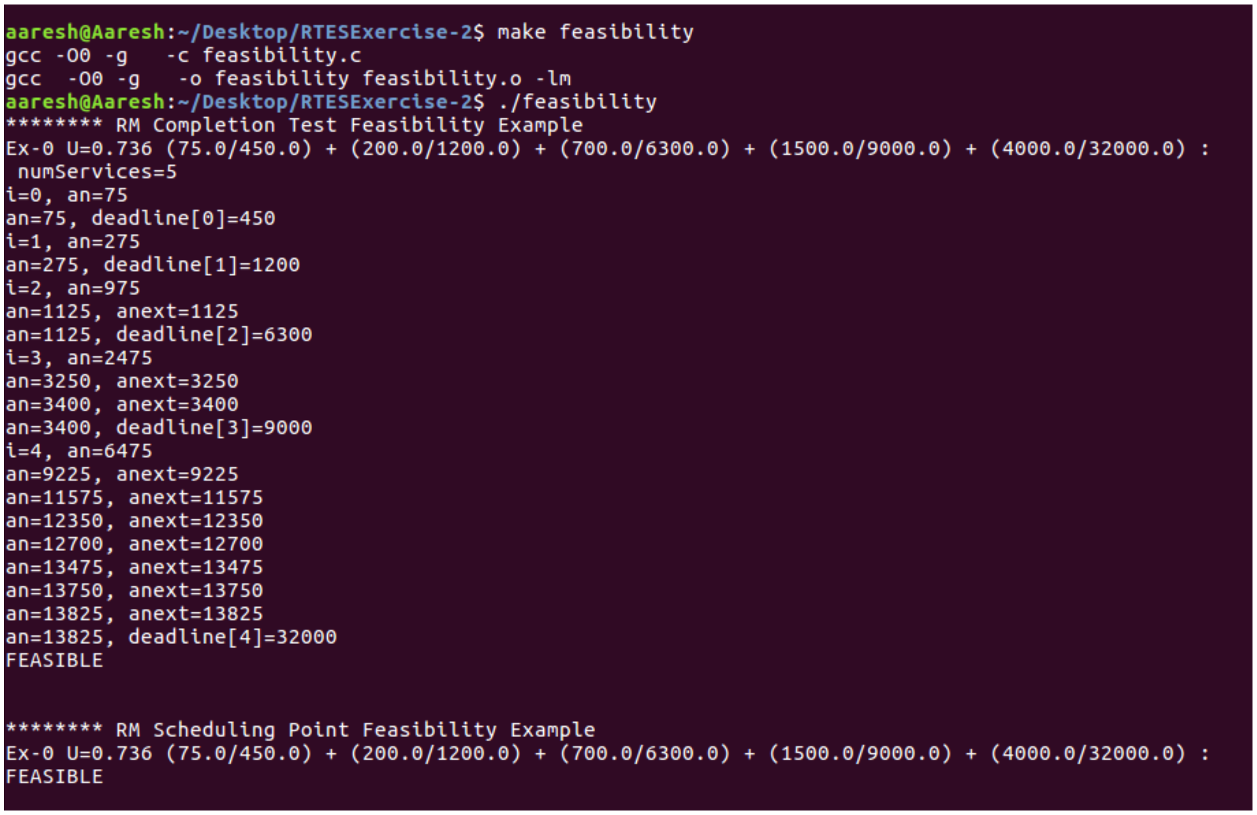
Therefore, U < RM LUB.


Cheddar : a free real time scheduling simulator
File Edit View Tools Help

```
Task name=s1    Period= 6; Capacity= 1; Deadline= 6; Start time= 0; Priority= 1; Cpu=cpu1
Task name=s2    Period= 6; Capacity= 1; Deadline= 6; Start time= 0; Priority= 1; Cpu=cpu1
Task name=s3    Period= 6; Capacity= 1; Deadline= 6; Start time= 0; Priority= 1; Cpu=cpu1
Task name=s4    Period= 8; Capacity= 1; Deadline= 8; Start time= 0; Priority= 1; Cpu=cpu1
Task name=s5    Period= 9; Capacity= 1; Deadline= 9; Start time= 0; Priority= 1; Cpu=cpu1
```

```
Scheduling feasibility, Processor cpu1 :
1) Feasibility test based on the processor utilization factor :

- The base period is 72 (see [18], page 5).
- 19 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.73611 (see [1], page 6).
- Processor utilization factor with period is 0.73611 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.73611 is equal or less than 0.74349 (see [1], page 16, theorem 8).


2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    s5 => 5
    s4 => 4
    s1 => 3
    s3 => 2
    s2 => 1
- All task deadlines will be met : the task set is schedulable.



Scheduling simulation, Processor cpu1 :
- Number of context switches :  52
- Number of preemptions :  0

- Task response time computed from simulation :
    s1 => 3/worst
    s2 => 2/worst
    s3 => 1/worst
    s4 => 4/worst
    s5 => 5/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
```

Fig 7. Cheddar output for timing diagram with scaled deadlines. The deadlines in our code are too big in units for Cheddar to plot and thus we scaled down our deadlines and provided them to Cheddar which gave us the following output.



```
aaresh@Aaresh:~/Desktop/RTESExercise-2$ make feasibility
gcc -O0 -g   -c feasibility.c
gcc  -O0 -g   -o feasibility feasibility.o -lm
aaresh@Aaresh:~/Desktop/RTESExercise-2$ ./feasibility
******** RM Completion Test Feasibility Example
Ex-0 U=0.736 (75.0/450.0) + (200.0/1200.0) + (700.0/6300.0) + (1500.0/9000.0) + (4000.0/32000.0) :
 numServices=5
i=0, an=75
an=75, deadline[0]=450
i=1, an=275
an=275, deadline[1]=1200
i=2, an=975
an=1125, anext=1125
an=1125, deadline[2]=6300
i=3, an=2475
an=3250, anext=3250
an=3400, anext=3400
an=3400, deadline[3]=9000
i=4, an=6475
an=9225, anext=9225
an=11575, anext=11575
an=12350, anext=12350
an=12700, anext=12700
an=13475, anext=13475
an=13750, anext=13750
an=13825, anext=13825
an=13825, deadline[4]=32000
FEASIBLE


******** RM Scheduling Point Feasibility Example
Ex-0 U=0.736 (75.0/450.0) + (200.0/1200.0) + (700.0/6300.0) + (1500.0/9000.0) + (4000.0/32000.0) :
FEASIBLE
```

Fig 8. Completion Test and Scheduling Point Feasibility Test outputs for the above values

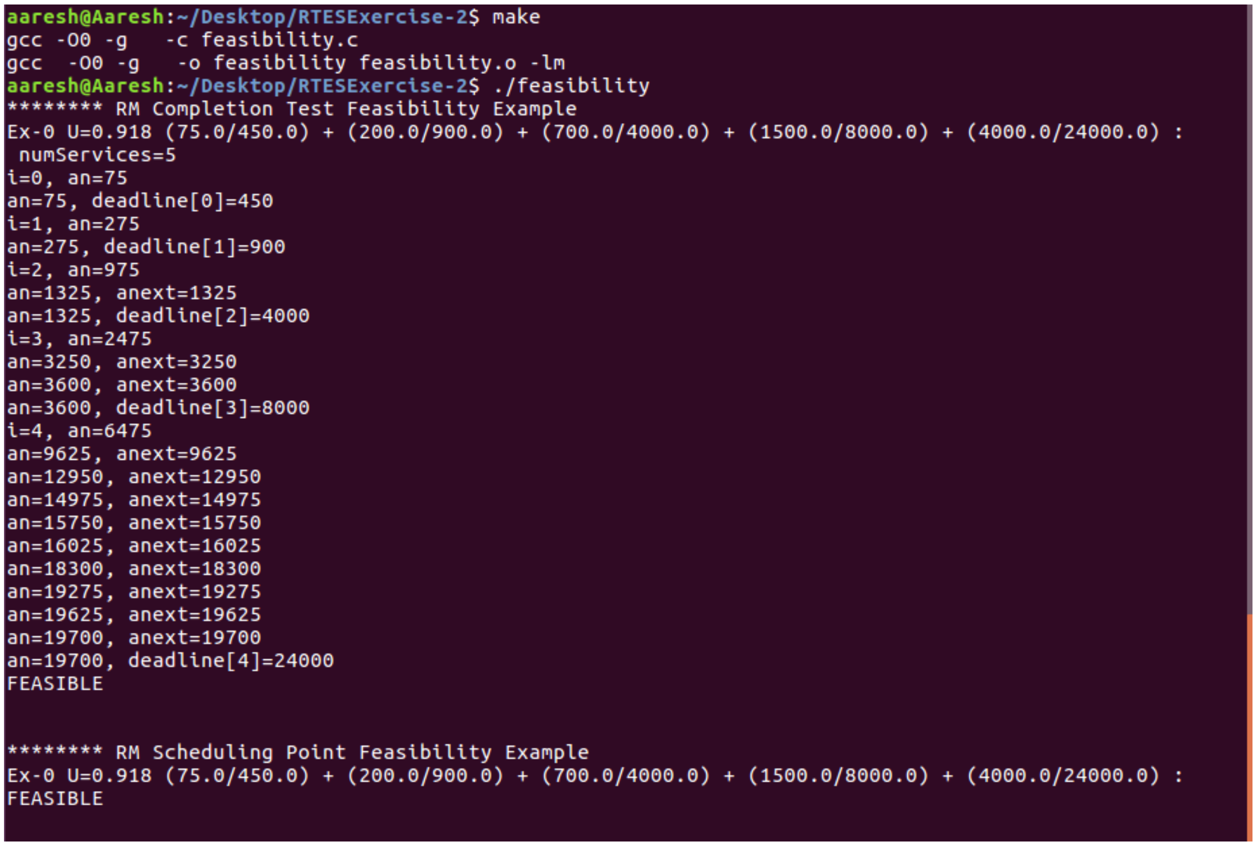
The above values for each service ensures that the system design not only successfully passes the Feasibility and the Completion Tests but also ensures that the total CPU utilization does not exceed the RM LUB value for five services. The RM LUB test however is pessimistic. Thus, to ensure maximum CPU utilization, we adjust the deadlines such that the design successfully passes the Feasibility and the Completion Tests but exceeds the RM LUB. The updated values are given below:

Thread	**Average Execution Time (ms)**	**Deadline (ms)**	**Request Time (In ms)**	**Request Frequency**
Path Finder	200	900	2/9	4.5
Face Recognition	1500	8000	3/16	5.33
Ultrasonic Sensor Read	75	450	1/6	6
Motor Control	4000	24000	1/6	6
GPS control	700	4000	7/40	5.71

Total Utilization (U) = 200/900 + 1500/8000 + 75/450 + 4000/24000 + 700/4000
= 2/9 + 3/16 + 1/6 + 1/6 + 7/40
= 0.9175

RM LUB for 5 services = $m*(2^{1\backslash m} - 1)$
= 0.74349

Thus, U > RM LUB but services are yet schedulable because it passes the completion point and feasibility test



```
aaresh@Aaresh:~/Desktop/RTESExercise-2$ make
gcc -O0 -g   -c feasibility.c
gcc  -O0 -g   -o feasibility feasibility.o -lm
aaresh@Aaresh:~/Desktop/RTESExercise-2$ ./feasibility
******** RM Completion Test Feasibility Example
Ex-0 U=0.918 (75.0/450.0) + (200.0/900.0) + (700.0/4000.0) + (1500.0/8000.0) + (4000.0/24000.0) :
 numServices=5
i=0, an=75
an=75, deadline[0]=450
i=1, an=275
an=275, deadline[1]=900
i=2, an=975
an=1325, anext=1325
an=1325, deadline[2]=4000
i=3, an=2475
an=3250, anext=3250
an=3600, anext=3600
an=3600, deadline[3]=8000
i=4, an=6475
an=9625, anext=9625
an=12950, anext=12950
an=14975, anext=14975
an=15750, anext=15750
an=16025, anext=16025
an=18300, anext=18300
an=19275, anext=19275
an=19625, anext=19625
an=19700, anext=19700
an=19700, deadline[4]=24000
FEASIBLE


******** RM Scheduling Point Feasibility Example
Ex-0 U=0.918 (75.0/450.0) + (200.0/900.0) + (700.0/4000.0) + (1500.0/8000.0) + (4000.0/24000.0) :
FEASIBLE
```

Fig 9. Completion Test and Scheduling Point Feasibility Test outputs for the deadlines such that they exceed RM LUB but pass the necessary and sufficient tests

- The analysis of various threads in our code is mentioned below:

Thread	Average execution time (ms)	Deadline (ms)	No. frames missed deadlines
Path Finder	195	200	32
	193	400	19
	202	600	9
	209	900	1
	203	1200	0

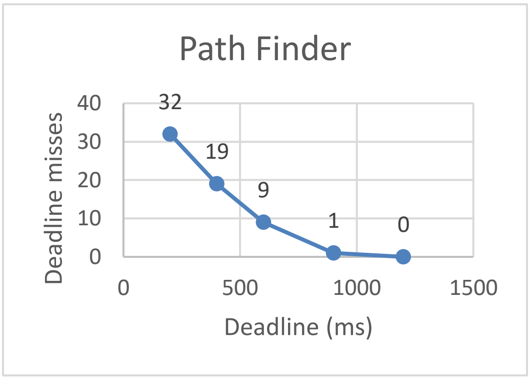


Fig 10. Path Finder Analysis for deadline (ms) vs deadline misses

Thread	Average execution time (ms)	Deadline (ms)	No. frames missed deadlines
Face Recognition	1421	2000	21
	1123	4000	11
	1574	5000	9
	1832	8000	2
	1647	9000	0

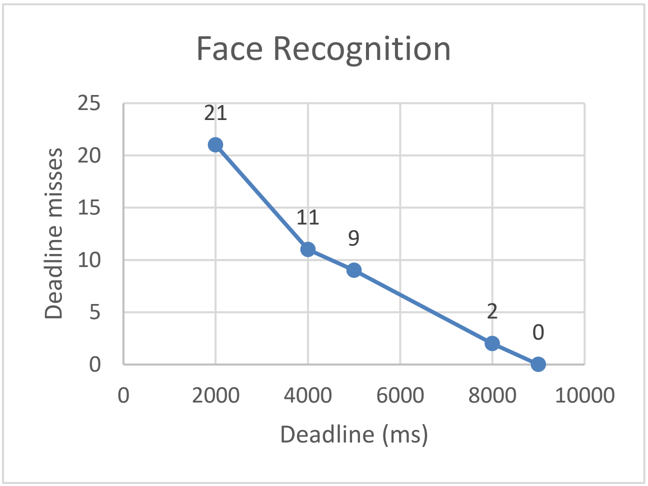


Fig 11. Face Recognition Analysis for deadline (ms) vs deadline misses

Thread	Average execution time (ms)	Deadline (ms)	No. frames missed deadlines
Ultrasonic Sensor	67	100	16
	74	200	12
	71	250	8
	79	300	1
	81	450	0

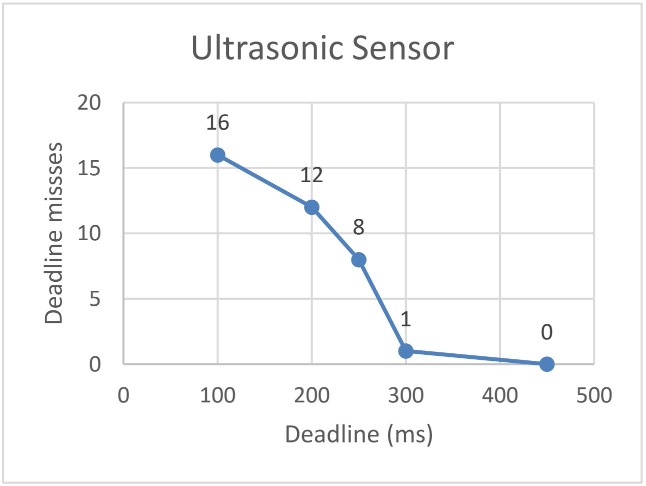


Fig 12. Ultrasonic Sensor Analysis for deadline (ms) vs deadline misses

Thread	Average execution time (ms)	Deadline (ms)	No. frames missed deadlines
Motor Control	3931	5000	23
	3864	7000	14
	4058	9000	4
	4197	15000	2
	3927	24000	0
	4169	32000	0

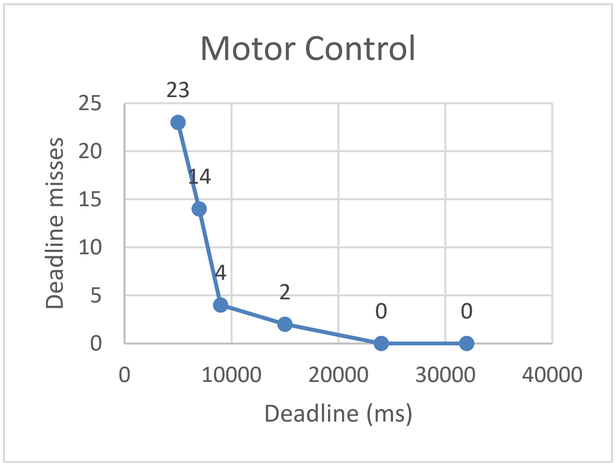


Fig 13. Motor Control Analysis for deadline (ms) vs deadline misses

Thread	Average execution time (ms)	Deadline (ms)	No. frames missed deadlines
GPS	583	1000	7
	785	1500	3
	735	2000	1
	702	4000	1
	683	6300	0

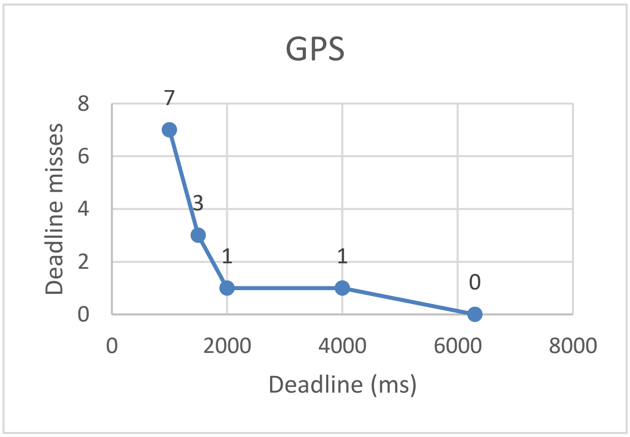


Fig 14. GPS Navigation Analysis for deadline (ms) vs deadline misses

Proof of Concept and Tests Completed:

The autonomous surveillance bot was divided into separate modules like Image processing, Obstacle avoidance, Navigation etc. and all these were coded and executed separately using the respective cameras and sensors and the Nvidia Jetson TK1.

The various components used in our project include Nvidia Jetson TK1, Logitech C200 camera, H-bridge SN754410IC, DC motors, Ultrasonic HCSR04 sensor and the Global Sat GPS BU-353. These components helped us in completing the hardware part of our project. The hardware and software were then executed using the Nvidia Jetson as the brain of the project and all test cases were validated and verified.

The various tests that were performed for code validation and verification are:

Test	Description	Status
Path Finder test for open road calculation	A road was built with chart paper and images were taken for various road colors to set an appropriate grayscale threshold	Passed Successfully
Face Recognition with confidence levels	Tested for face recognition accuracy and predictability in various illuminated scenarios as well as faces with or without spectacles and faces with headphones	Passed Successfully
Ultrasonic sensor response for various distances	Measured distance taken by the ultrasonic sensor and compared that to values which were measured manually using a foot ruler	Passed Successfully
Motor control tests	Motors were tested for various delays to get the best movement in terms of direction	Passed Successfully
GPS co-ordinate verification	Setting the destination and source address and verified the navigation instructions received by our program with that of Google Maps.	Passed Successfully

Lessons Learnt:

- **Issues Faced:**

1. USB Bandwidth issue: For our application, we required two webcams. However, on connecting the two devices to our Jetson Board, we got an 'Insufficient USB bandwidth error'. This error was because the NVIDIA JETSON USB port is configured as a USB 2.0 port by default.

Solution: To fix the error, we made changes to the config file and the bootloader file of the NVIDIA JETSON board and modified the ODM data from 0 to 2.

2. GPIO 'device or resource busy' issue: To use the GPIO pin of the NVIDIA JETSON board, the user must export the GPIO pins in the super-user mode. If these GPIO pins aren't unexported before exiting and reusing them the system will assume the GPIO to be busy and give an error.

Solution: To get rid of this issue, in the code we wrote a gpiounexport() function that was executed on program exit using a signal(SIGINT, interrupthandler) in the program.

3. Face Recognition Accuracy: The accuracy of the face recognition algorithm used in the code greatly varied depending on the amount of light in the image captured for detection. In low-lighting conditions, the accuracy of the algorithm reduced.

Solution: To improve accuracy, we used images of different background lightings for training the algorithm initially. We also ensured that all images of the same size and the eyes and the lip features are uniform in all the images.

4. Processor Core Affinity: The multicore functionality of the Nvidia Jetson was exposed in our code and we achieved better results by running threads on multiple cores. The face recognition thread does not share any memory or variables with the other threads and thus, this thread was made to run on a separate core. This led to better CPU utilization (as high as 91%) and a safety margin of 9%.

- **Learnings:**

This project helped us improve our real-time programming skills and understanding of the firmware development process from a real-time point of view. We understood the use of Mutexes and Semaphores for resource sharing and task synchronization. We were also introduced to image processing using OpenCV and realized the amazing things that we can do using a camera and a processor. Lastly, the project helped us get hands-on experience with the Jetson board and Linux.

Conclusion:

In conclusion, we were glad to successfully implement the services we proposed in the project proposal with relative accuracy. We would like to thank Prof. Timothy Scherr for giving us this opportunity. It was indeed a wonderful learning experience. We would also like to thank all the T.A.s for their valuable inputs. Our aim was to design a Soft real-time system that successfully implemented multithreading and ensured maximum CPU utilization ensuring that no deadline is missed as much as possible and we were glad we could implement it.

Project Staffing:

- Aaresh Bachana – aaresh.bachana@colorado.edu
- Abhilash Manjunath – abhilash.manjunath@colorado.edu
- Harshil Sheth – harshil.sheth@colorado.edu
- Shrivathsa Murthy – vathsa98@gmail.com

References:

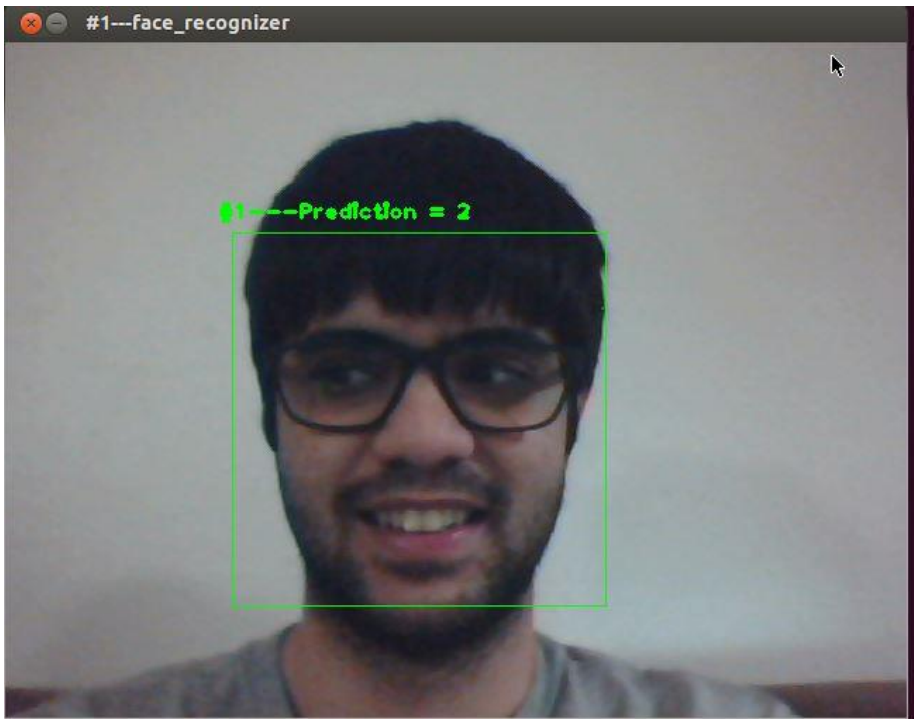
[1] https://www.brivo.com/brivo_blog/5-surprising-office-security-fact
[2] http://ieeexplore.ieee.org/document/4728491/
[3] http://ieeexplore.ieee.org/document/7013165/
[4] http://docs.opencv.org/2.4/modules/contrib/doc/facerec/tutorial/facerec_video_recognition.html
[5] https://github.com/jetsonhacks/JHHC-SR04/tree/master/src
[6] https://github.com/tomazas/opencv-lane-vehicle-track
[7] https://www.intorobotics.com/how-to-detect-and-track-object-with-opencv/

Appendix:

Note: Project files and code have been uploaded to github

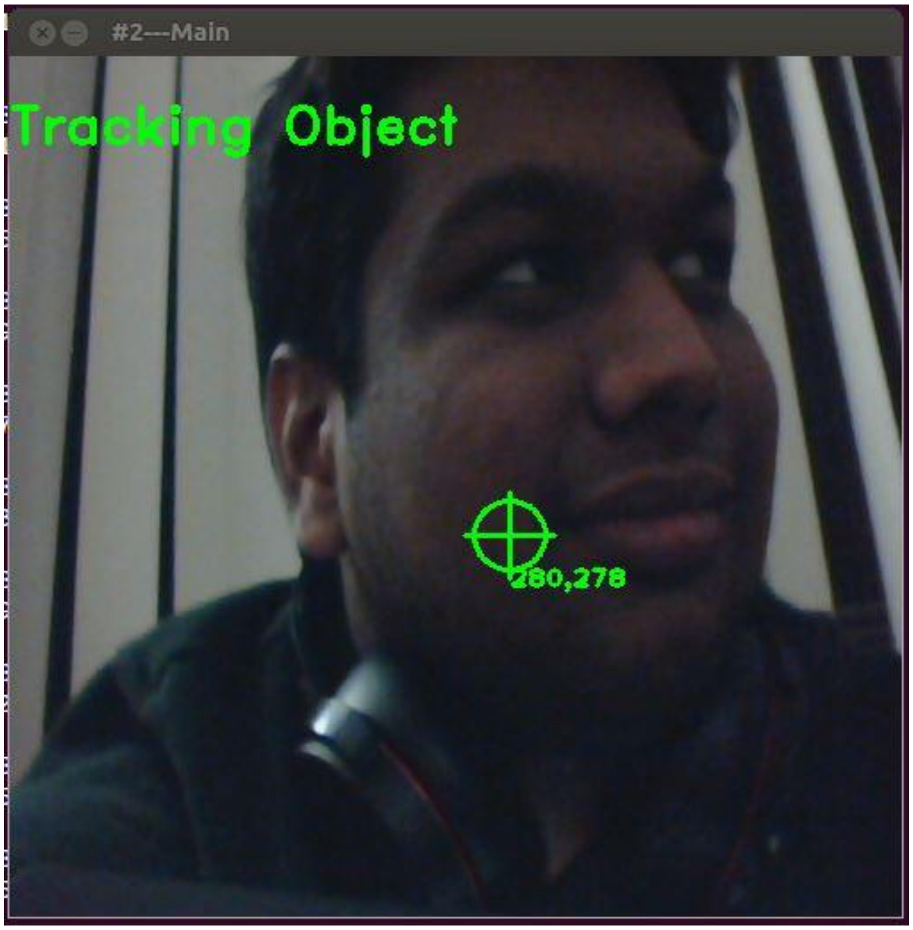
Github Link: https://github.com/shmu7023/rtes_project/tree/master/root

1. Face Recognition:



The face recognition algorithm was trained for multiple faces and we realized that the lighting of the image affected the accuracy of the detection. Thus, the algorithm was trained with images of variable brightness and the accuracy was tested with Aaresh's image as shown in the figure,

2. Object tracking (indicating Path Finding)



The path finding algorithm is based on adjusting the algorithm for a fixed shade of gray in the image (indicating a road). Thus, to test the algorithm, we adjusted the threshold such that it tracks Abhilash's face and always calculates the X and the Y co-ordinates of the detected image.

3. The GPS module – Setup and Algorithm

We have used a USB based GPS module in which the GPS data can be queried through a python script. The python script calls gpsd GPS daemon to interact with the hardware. Gmaps.py is the main python file which grabs the current GPS data using GpsPoller class. The result is a python dictionary. We are only interested in latitude and longitude which can be parsed from 'lat' and 'lon' dictionary keys.

GPS setup

Change from Binary to NMEA mode

```
gpsctl -f -n /dev/ttyUSB0
```

Configure serial port baud rate

```
stty -F /dev/ttyUSB0 ispeed 4800
```

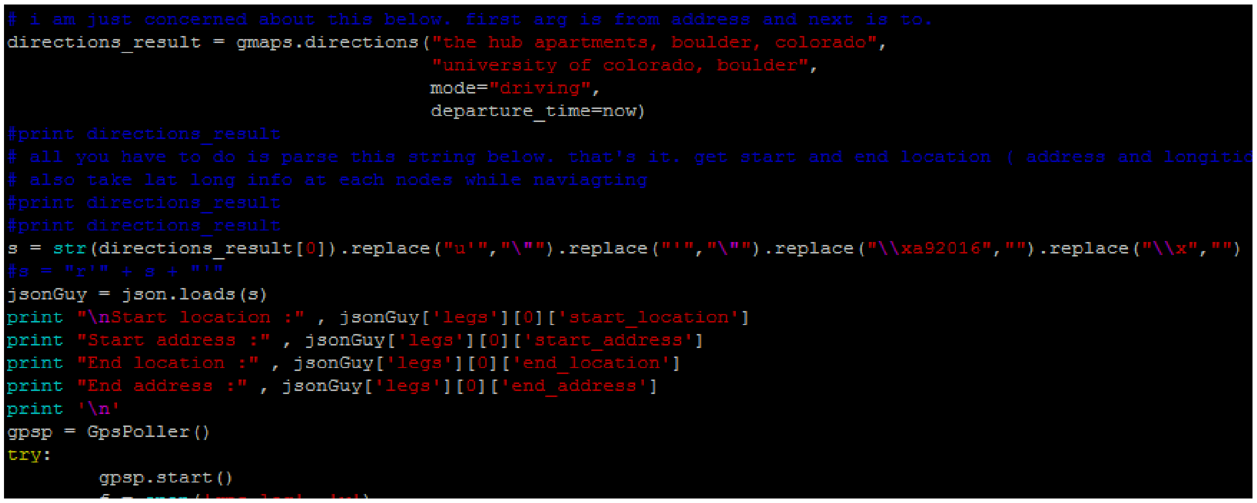
GPS data is written to /dev/ttyUSB. We are not parsing this file as gpsd daemon does the job for us. GPS stdout is redirected to gps_log file which is parsed by the GPS c thread to calculate deadline misses, if any.

Navigation

The destination address is given as an input though command line argument. Now for testing purposes, it is hardcoded. Ubuntu, the operating system running on the Jetson, talks to Google map APIs to receive navigation instructions. The algorithm that I have developed waits until it reaches the first hop by taking current location into consideration. Until then, it displays ‘keep going straight’ message. Once first hop is reached, it will display the next maneuver instruction and the same process repeats until destination is reached. Navigation stdout is redirected to nav_log and gmaps_log files which are parsed by the GPS c thread.

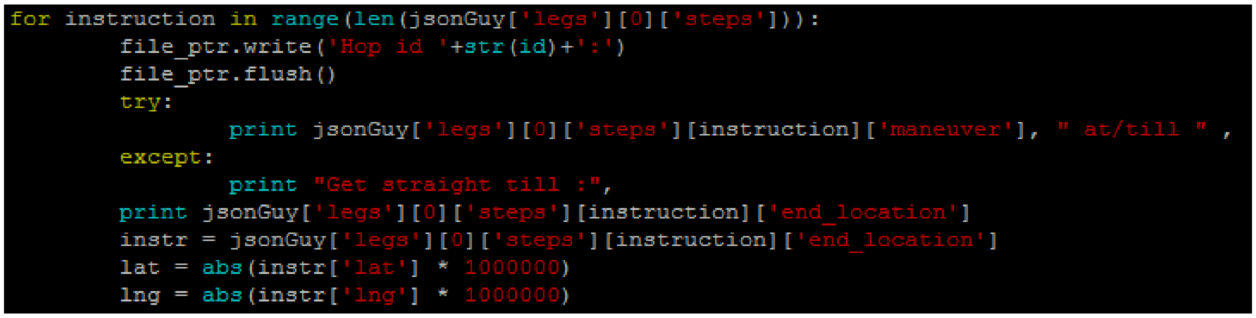
The Algorithm

The current latitude and longitude is compared with that of next hop location. The code is in python which talks to Google to receive navigation instructions. Json parser is used to parse the Google response as shown in the snippet below. Next hop latitude and longitude are subtracted with that of current value up to 6 decimal points. Only if the difference is less than 300, we can conclude that the user has approached current hop destination. Once the user moves to the next hop, difference will get smaller and respective instruction is displayed. This process is repeated until the user travels through all the hops to the final destination.



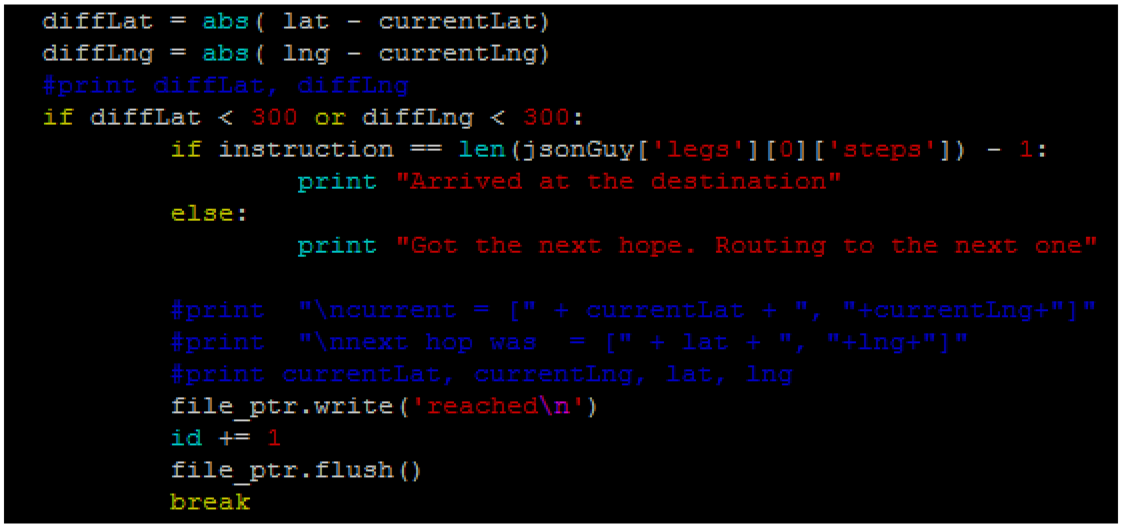
```
# i am just concerned about this below. first arg is from address and next is to.
directions_result = gmaps.directions("the hub apartments, boulder, colorado",
                                     "university of colorado, boulder",
                                     mode="driving",
                                     departure_time=now)
#print directions_result
# all you have to do is parse this string below. that's it. get start and end location ( address and longitid
# also take lat long info at each nodes while naviagting
#print directions_result
#print directions_result
s = str(directions_result[0]).replace("u'","\"").replace("'","\"").replace("\\xa92016","").replace("\\x","")
#s = "r'" + s + "'"
jsonGuy = json.loads(s)
print "\nStart location :" , jsonGuy['legs'][0]['start_location']
print "Start address :" , jsonGuy['legs'][0]['start_address']
print "End location :" , jsonGuy['legs'][0]['end_location']
print "End address :" , jsonGuy['legs'][0]['end_address']
print '\n'
gpsp = GpsPoller()
try:
        gpsp.start()
```

Google maps API to receive navigation instruction from a source to destination



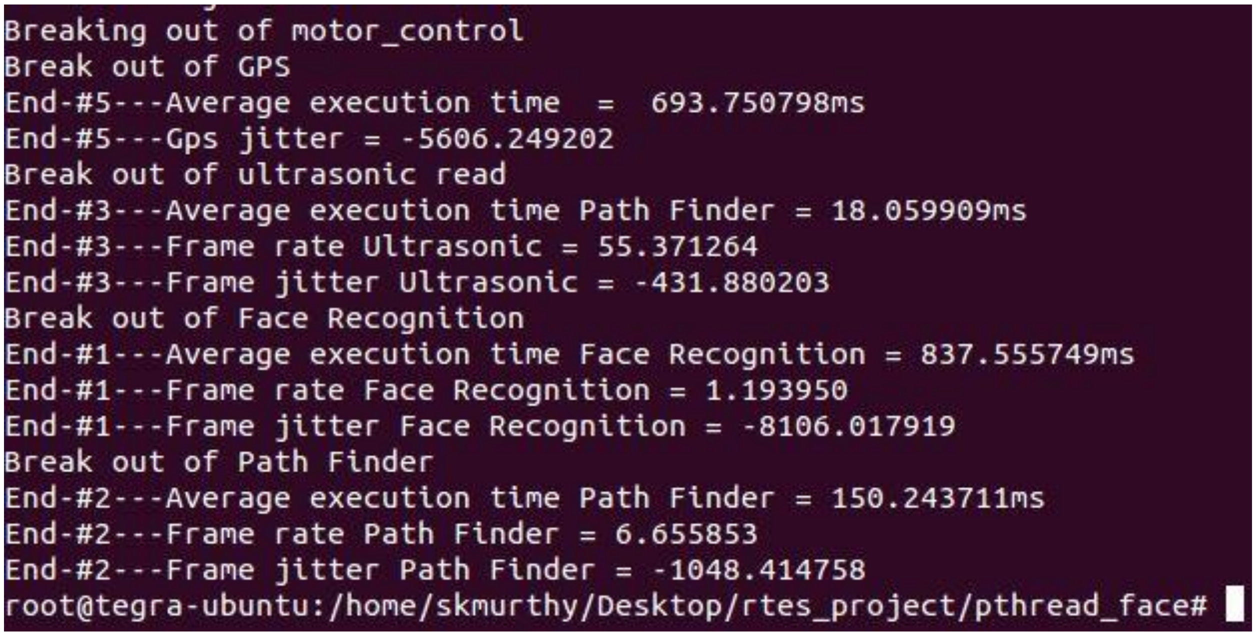
```
for instruction in range(len(jsonGuy['legs'][0]['steps'])):
        file_ptr.write('Hop id '+str(id)+':')
        file_ptr.flush()
        try:
                print jsonGuy['legs'][0]['steps'][instruction]['maneuver'], " at/till " ,
        except:
                print "Get straight till :",
        print jsonGuy['legs'][0]['steps'][instruction]['end_location']
        instr = jsonGuy['legs'][0]['steps'][instruction]['end_location']
        lat = abs(instr['lat'] * 1000000)
        lng = abs(instr['lng'] * 1000000)
```

Json parser to parse each hops in the Google response string



```
diffLat = abs( lat - currentLat)
diffLng = abs( lng - currentLng)
#print diffLat, diffLng
if diffLat < 300 or diffLng < 300:
        if instruction == len(jsonGuy['legs'][0]['steps']) - 1:
                print "Arrived at the destination"
        else:
                print "Got the next hope. Routing to the next one"

        #print  "\ncurrent = [" + currentLat + ", "+currentLng+"]"
        #print  "\nnext hop was  = [" + lat + ", "+lng+"]"
        #print currentLat, currentLng, lat, lng
        file_ptr.write('reached\n')
        id += 1
        file_ptr.flush()
        break
```

Decision making based on current location



```
Breaking out of motor_control
Break out of GPS
End-#5---Average execution time  =  693.750798ms
End-#5---Gps jitter = -5606.249202
Break out of ultrasonic read
End-#3---Average execution time Path Finder = 18.059909ms
End-#3---Frame rate Ultrasonic = 55.371264
End-#3---Frame jitter Ultrasonic = -431.880203
Break out of Face Recognition
End-#1---Average execution time Face Recognition = 837.555749ms
End-#1---Frame rate Face Recognition = 1.193950
End-#1---Frame jitter Face Recognition = -8106.017919
Break out of Path Finder
End-#2---Average execution time Path Finder = 150.243711ms
End-#2---Frame rate Path Finder = 6.655853
End-#2---Frame jitter Path Finder = -1048.414758
root@tegra-ubuntu:/home/skmurthy/Desktop/rtes_project/pthread_face#
```

Average Execution Time for all the services