

AIM:

To implement smart parking system which can provide :

- a.) automatic billing generation
- b.) automatic report generation.

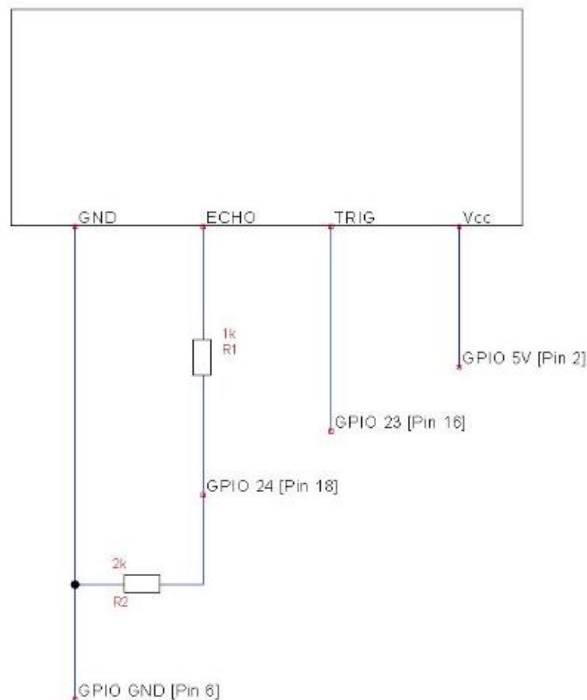
DESCRIPTION:

Car parking is a major issue in modern congested cities of today. There are too many vehicles on the road and not enough parking spaces. Getting stuck in the vehicle because of parking is quite complicated due to less space and time. Moreover, a manually handled car parking lot can be stressful like anything. This is not only stressful for the visitors but also for every parking owner. Such a situation has led to the need for efficient parking management systems.

The system uses ultrasonic sensors to detect either car park occupancy . Different detection technologies are reviewed and compared to determine the best technology for developing SPS. Features of SPS include vacant parking space detection, a billing technology.

HARDWARE AND SOFTWARE REQUIREMENTS:

Raspberry pi
Ultrasonic sensors
male and female jumper wires
A/c adapter.

CIRCUIT DIAGRAM:

CODE:

```
#Libraries

import RPi.GPIO as GPIO

import time

#GPIO Mode (BOARD / BCM)

GPIO.setmode(GPIO.BCM)

#set GPIO Pins

GPIO_TRIGGER = 23

GPIO_ECHO = 24

StartTime = time.time()

StopTime = time.time()

#set GPIO direction (IN / OUT)

GPIO.setup(GPIO_TRIGGER, GPIO.OUT)

GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():

    # set Trigger to HIGH

    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW

    time.sleep(0.00001)

    GPIO.output(GPIO_TRIGGER, False)

    # save StartTime

    while GPIO.input(GPIO_ECHO) == 0:

        StartTime = time.time()

    # save time of arrival
```

```

while GPIO.input(GPIO_ECHO) == 1:

    StopTime = time.time()

    # time difference between start and arrival

    TimeElapsed = StopTime - StartTime

    # multiply with the sonic speed (34300 cm/s)

    # and divide by 2, because there and back

    distance = (TimeElapsed * 34300) / 2

    return distance

if __name__ == '__main__':

    try:

        while True:

            dist = distance()

            if (dist<=111):

                print ("Slot is empty")

                time.sleep(3)

            else:

                print("Slot occupied")

                print("slot occupied for:%d msec" %(StopTime/1000000))

                print ("Amount:%d" %((StopTime/1000000000)*3))

                time.sleep(5)

        # Reset by pressing CTRL + C

    except KeyboardInterrupt:

        print("Measurement stopped by User")

        GPIO.cleanup()

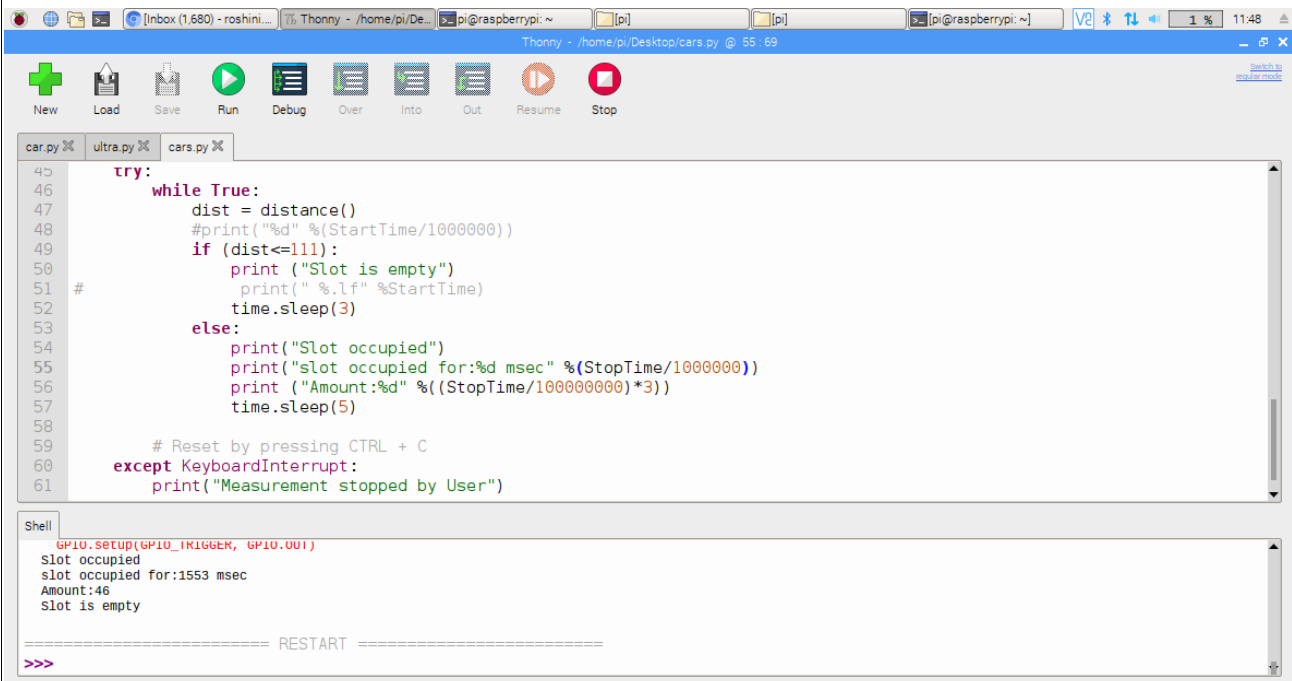
```

STEPS:

1. Connect keyboard and mouse to Raspberry Pi using USB cables.
2. Connect Monitor to Raspberry Pi using HDMI cable
3. Connect Led and PIR sensor to Raspberry Pi as shown in Circuit Diagram
4. Run the Python code above.

OUTPUT:

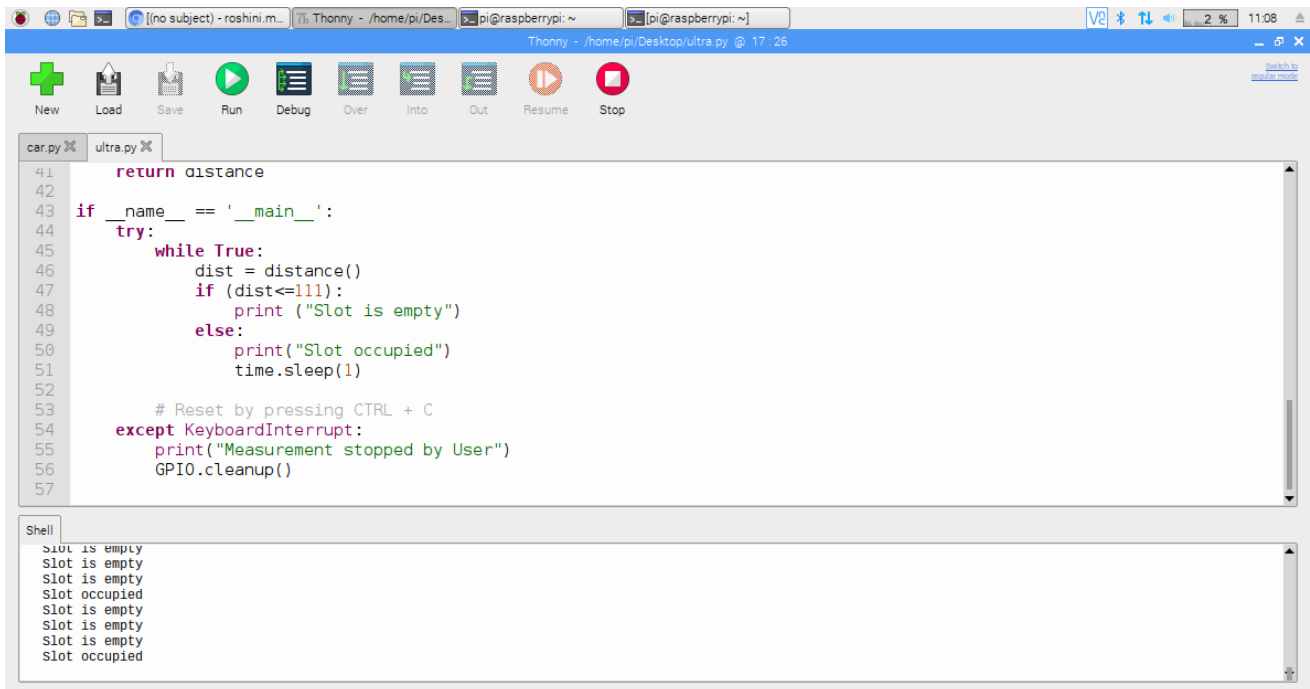
a.) automatic billing generation



```
45
46     try:
47         while True:
48             dist = distance()
49             #print("%d" %(StartTime/1000000))
50             if (dist<=111):
51                 print ("Slot is empty")
52                 print(" %.1f" %StartTime)
53                 time.sleep(3)
54             else:
55                 print("Slot occupied")
56                 print("slot occupied for:%d msec" %(StopTime/1000000))
57                 print ("Amount:%d" %((StopTime/100000000)*3))
58                 time.sleep(5)
59
60         # Reset by pressing CTRL + C
61     except KeyboardInterrupt:
62         print("Measurement stopped by User")
```

```
Shell
GPIO.setmode(GPIO_TRIGGER, GPIO.OUT)
slot occupied
slot occupied for:1553 msec
Amount:46
Slot is empty

===== RESTART =====
>>>
```



The screenshot shows the Thonny IDE interface on a Raspberry Pi. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Resume, and Stop. The main editor window displays the code for 'ultra.py'.

```
41     return distance
42
43 if __name__ == '__main__':
44     try:
45         while True:
46             dist = distance()
47             if (dist <= 111):
48                 print ("Slot is empty")
49             else:
50                 print("Slot occupied")
51                 time.sleep(1)
52
53         # Reset by pressing CTRL + C
54     except KeyboardInterrupt:
55         print("Measurement stopped by User")
56         GPIO.cleanup()
57
```

The Shell window at the bottom shows the output of the program:

```
Slot is empty
Slot is empty
Slot is empty
Slot occupied
Slot is empty
Slot is empty
Slot is empty
Slot is empty
Slot occupied
```