CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

With the rapid proliferation of vehicles availability and usage in recent years, finding a vacant parking space is becoming more and more difficult, resulting in a number of practical conflicts. The problem is more tough because of a continues growing number of vehicle and also size of vehicles. Car parking is not just a major problem in India but also in all over the world. In this system we propose an automatic and real-time system for automated car parking by taking car size when it enters parking area and displaying allocated parking slot which suits best for that car based on size this also makes effective utilization of parking space. This project reduces human effort at the parking area to great extent such as in case of searching of free slots and also in the areas where parking space is limited.

1.2 EXISTING SYSTEM

In the conventional parking system, a person who is appointed will manually identify the availability of parking spaces in the whole parking area and should allow people to park their cars. But there is no solution for discovery of availability of slots within less time and also there is no methods or procedures for effective management & utilization of the available parking spaces based upon the size of vehicle. This conventional system increases stress for searching a parking slot and wasting valuable time this has led to the need for efficient parking management systems.

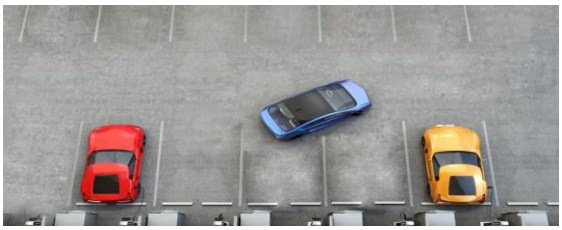


Figure 1.1: Existing system of Parking

1.3 PROPOSED SYSTEM

The proposed system can effectively manage and allocate parking slots in intelligent way. It can detects the presence of the vehicle while it enters the parking area and it allows the user to enter the car size as input to the system and system processes the information and displays the allocated parking slot in the display present at the entrance and then then entry gate opens automatically. The system has the information about size of the slot based on slot number predefined and each slot is equipped with ir sensor to detect the presence of vehicle. If vehicle is leaving the parking place the sensors placed at exit point detects the car and will open exit gate and the system will automatically update the parking slot information. Thus, the system solves the parking issue in the parking slots and the users can used an efficient IOT based parking management system.

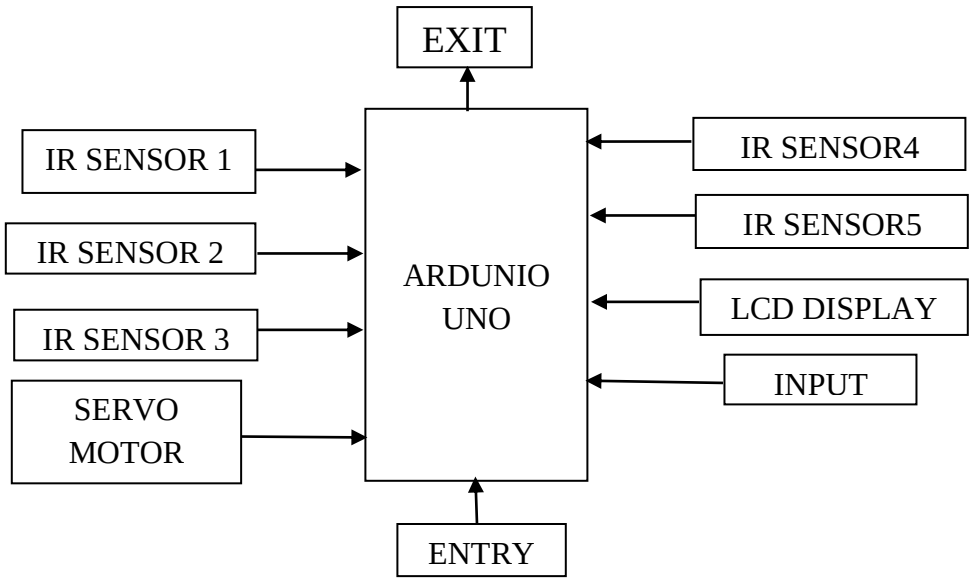


Figure 1.2: Block Diagram

1.4 SOFTWARE & HARDWARE REQUIREMENTS

SOFTWARE REQUIREMENTS:

- **Software: Arduino IDE:**

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them. Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor

- Verify
  Checks your code for errors compiling it.

- Upload
  Compiles your code and uploads it to the configured board. See uploading below for details.

- New
  Creates a new sketch.

- Open

  Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

- Save
  Saves your sketch.

- Serial Monitor
  Opens the serial monitor.

HARDWARE REQUIREMENTS

- **Arduino:**

  The UNO is the best board to get started with electronics and coding. If this is your first experience tinkering with the platform, the UNO is the most robust board you can start playing with. The UNO is the most used and documented board of the whole Arduino family.

- **IR Modules:**

  Active infrared sensors consist of two elements: infrared source and infrared detector. Infrared sources include an LED or infrared laser diode. Infrared detectors include photodiodes or phototransistors. The energy emitted by the infrared source is reflected by an object and falls on the infrared detector.

- **Servos:**

  A Servo is a small device that incorporates a two wire DC motor, a gear train, a potentiometer, an integrated circuit, and an output shaft. Of the three wires that stick out from the motor casing, one is for power, one is for ground, and one is a control input line.

The shaft of the servo can be positioned to specific angular positions by sending a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. If the coded signal changes, then the angular position of the shaft changes.

- **LCD Display (16*2):**

  The display units are very important in communication between the human world and the machine world. The display unit work on the same principle, it does not depend on the size of the display it may be big or the small. We are working with the simple displays like 16×1 and 16×2 units. The 16×1 display unit has the 16 characters which present in one line and 16×2 display units have 32 characters which are present in the 2 line. We should know that to display each character there are 5×10 pixels. Thus, to display one character all the 50 pixels should be together. In the display there is a controller which is HD44780 it is used to control the pixels of characters to display.

- **Push Buttons:**

  The pushbutton is a component that connects two points in a circuit when you press it. The example turns on an LED when you press the button.

- **Bread board:**

  A breadboard is a solderless device for temporary prototype with electronics and test circuit designs. Most electronic components in electronic circuits can be interconnected by inserting their leads or terminals into the holes and then making connections through wires where appropriate. The breadboard has strips of metal underneath the board and connects the holes on the top of the board. The metal strips are laid out as shown below. Note that the top and bottom rows of holes are connected horizontally and split in the middle while the remaining holes are connected vertically.

- **jumper wires:**

  Jumper wires are simply wires that have connector pins at each end, allowing them to be used to connect two points to each other without soldering. Jumper wires are typically used

with breadboards and other prototyping tools in order to make it easy to change a circuit as needed. Fairly simple. In fact, it doesn't get much more basic than jumper wires.

Though jumper wires come in a variety of colors, the colors don't actually mean anything. This means that a red jumper wire is technically the same as a black one. But the colors can be used to your advantage in order to differentiate between types of connections, such as ground or power.

Module description

- User
- Entry
- Exit
- Input

Modules Description

- **User:**

User get to parking area and he will follow the instructions given by the system to check whether there is parking available or not. They enter their vehicle size in the input area and for the system to process the request.

- **ENTRY:**

The sensor is placed at the entry point so that when ever vehicle arrives the parking area the system get activated and will allow the user to give the input details of his car to proceed for further processing.

- **EXIT:**

  A sensor is placed in exit point of the parking lot to detect the car when it is leaving the parking area and this will notify the system to updated the current details after the car is left to make the slot available for others.

- **INPUT:**

  The input section is the one through which the user gets a chance to enter the size of his vehicle by pressing a button. The sizes of car are categorized into three they are micro, sedan, suv.

1.5 SYSTEM REQUIREMENTS SPECIFICATIONS

1.5.1. Functional requirements:

In software engineering, a functional requirement defines a function of a software system or its component. A function is defined as a set of inputs, the behaviour and outputs. Functional requirements may be calculations, technicaldetails, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioural requirements describing all the cases where the system uses the functional requirements are captured in use case. Generally, functionalrequirements are expressed in the form “system shall do<requirements>”. The plan for implementation functional requirements are a detailed in the system design. In the requirements engineering, functional requirements specify particularresults of the system. Functionalrequirements drive the application architecture of a system. A requirements analyst generates use case after gathering and validating a set of functional requirements. The hierarchy of functional requirements is user/stakeholder request->feature->use case->business rule.

This system describes the requirements of the system for those requirements which are expressed in natural language style. A client should be able to login to the system through first

page of the application and mention his required operations and thus activities performed accordingly. Smooth navigation should be provided to easily handle the data and also for easy navigation.

1.5.2. NON-FUNCTIONAL REQUIREMENTS

In system engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. This should be contrasted with functional requirements that define specific behaviour or functions. The plans for implementing non functional requirements are “system shall be <requirements>”. Non-functional requirements are often called Qualities of a system.The following are the Non-functional Requirements for our system

- **Availability:**

  A system’s “availability” or “uptime” is the amount of time that is operational and available for use. As our system will be used at any time our system must be available always. If there are any cases of updating they must be performed while it is rest position without interrupting the normal service.

- **Efficiency:**

  Specifies how well the system performs: sensor data, memory, power etc. All of the above mentioned resources can be effectively used by performing most of the validations.

- **Integrity:**

  Integrity requirements define the security attributes of the system, restricting access to system when something fails or performing in wrong manner so, for this a kill switch is provided in the system.

- **Usability:**

  Ease-of-use requirements address the factors that constitute the problems of the users so, it can be easier for the users to operate.

CHAPTER 2

SYSTEM ANALYSIS

2.1 FLOW ORIENTED MODELLING

2.1.1 DATA FLOW DIAGRAM (DFD)

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design).On a DFDs data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

A DFD provides no information about the timing of process, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come from and go to, nor where the data will be stored (all of which are shown on a DFD).

A Data Flow Diagram (DFD) is also known as a Process Model. Process Modelling is an analysis technique used to capture the flow of inputs through a system (or a group of processes) to their resulting output. The model is fairly simple in that there are only four types of symbols- process, dataflow, external entity, data store.

In a DFD, there are four symbols:

- A square defines a source (originator) or destination of system data.
- An arrow identifies data flow. It is the principle through which the information flows.

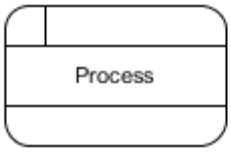
- A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
- An open rectangle is a data store.

The following are some of the symbols in Process Modelling represents

Process:

An activity or a function that is performed for some specific reason; can be manual or Computerized; ultimately each process should perform only one activity.

Symbol:



Data Flow:

Single piece of data or logical collection of information like a bill.

Symbol:

Data Store:

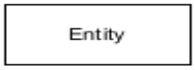
Collection of data that is permanently stored.

Symbol:

External Entity:

A person, organization, or system that is external to the system but intersects with it.

Symbol:



The following are the levels of the Data Flow Diagrams:

DFD Level0:

- Level 0 is called context DFD, shows the whole system in one process.
- It does not represent procedural logic.

DFD Level 1:

- The system in more detail.
- How the data enter the system.
- How these data items are transformed.
- How they leave the system.

DFD Level2:

- Show how the information moves from and to each of the processes.
- Level2 diagrams may not be needed for all Level1 processes.
- Correctly numbering each process helps the user understand where the process fits in to overall system.

Dataflow Diagram Principles

- The general principle in Data Flow diagramming is that a system can be decomposed into subsystem and subsystems can be decomposed into lower level subsystem and so on.
- Each subsystem represents a process or activity in which data is processed. At the lowest level, process can no longer be decomposed.
- Each process in a DFD has the characteristics of a system.
- Just as a system must have input and output, so a process must have an input and output.
- Data enters the system from the environment, data flows between processes within the system and data is produced as output from the system.

Salient features of Data flow diagram

- The DFD shows flow of data, not of control loops and decisions are controlled considerations do not appear on a DFD.
- The DFD does not indicate the time faster factor involved in any process whether the data flow take place daily, weekly, monthly or yearly.
- The sequence of events is not brought out of the DFD.

Constructing a Dataflow Diagrams

- Process should be named and must be numbered for an easy reference Each name should be representatives of the process.
- The direction of the flow is from top to bottom and left to right. Data traditionally flow form source to destination although they may flow back to the source. One way to indicate this is to draw long flow line back to destination.An alternative way is to repeat the source symbol as destination. Since it is used more than once in the DFD it is marked with a short diagonal.
- Thename of data stores and destinations are written in capital letters. Process and data flow names have the first letter of each capitalized.
- A DFD typically shows the minimum contents of the data store.Each data store should contain all the data elements that flow input and output.

Dataflow Diagrams

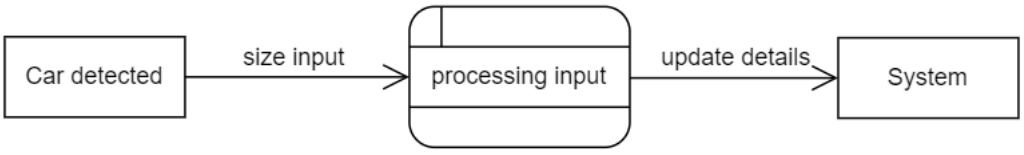


Figure: 2.1.1.1.showing initial process

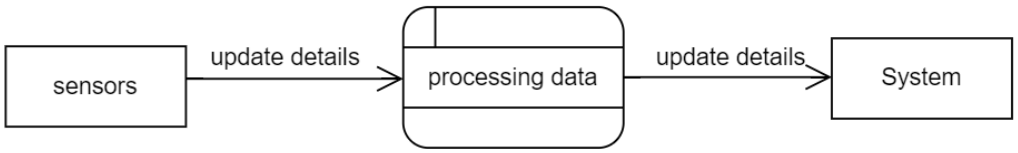


Figure: 2.1.1.2. Sensor level view

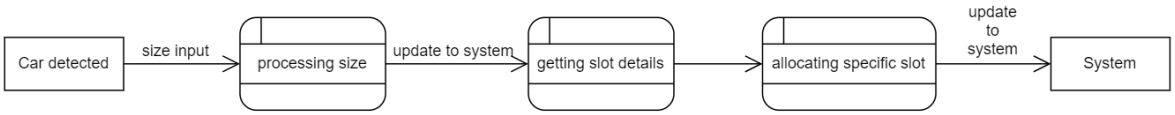


Figure2.1.1.3: Entire view

2.1.2. FLOWCHARTS

A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting them with arrow. This diagrammatic representation can give step-by-step solution to a givenproblem. Process operations are represented in these boxes, and arrows connecting them represent flow of control. Data flows are not typically represented in a flowchart, in contrast with data flow diagrams; rather, they are implied by the sequencing of operations. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

A flow chart diagram typically consists of shapes such as ovals, diamonds and boxes that contain words relating to a certain issue and lines with arrows connecting boxes to one another. Flowcharts are used in designing and documenting complex processes or programs. Like other types of diagrams, they help visualize what is going on and thereby help the viewer to understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it. There are many different types of flowcharts, and each type has its own repertoire of boxes and notational conventions. The two most common types of boxes in a flowchart are:

- a processing step, usually called activity, and denoted as a rectangular box.
- a decision, usually denoted as a diamond.

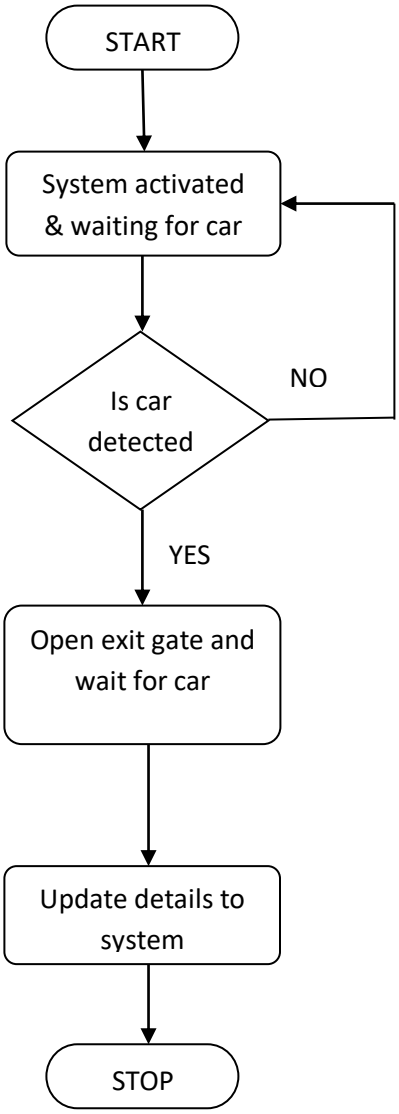


Figure2.1.2.1: Level-0 Diagram user view

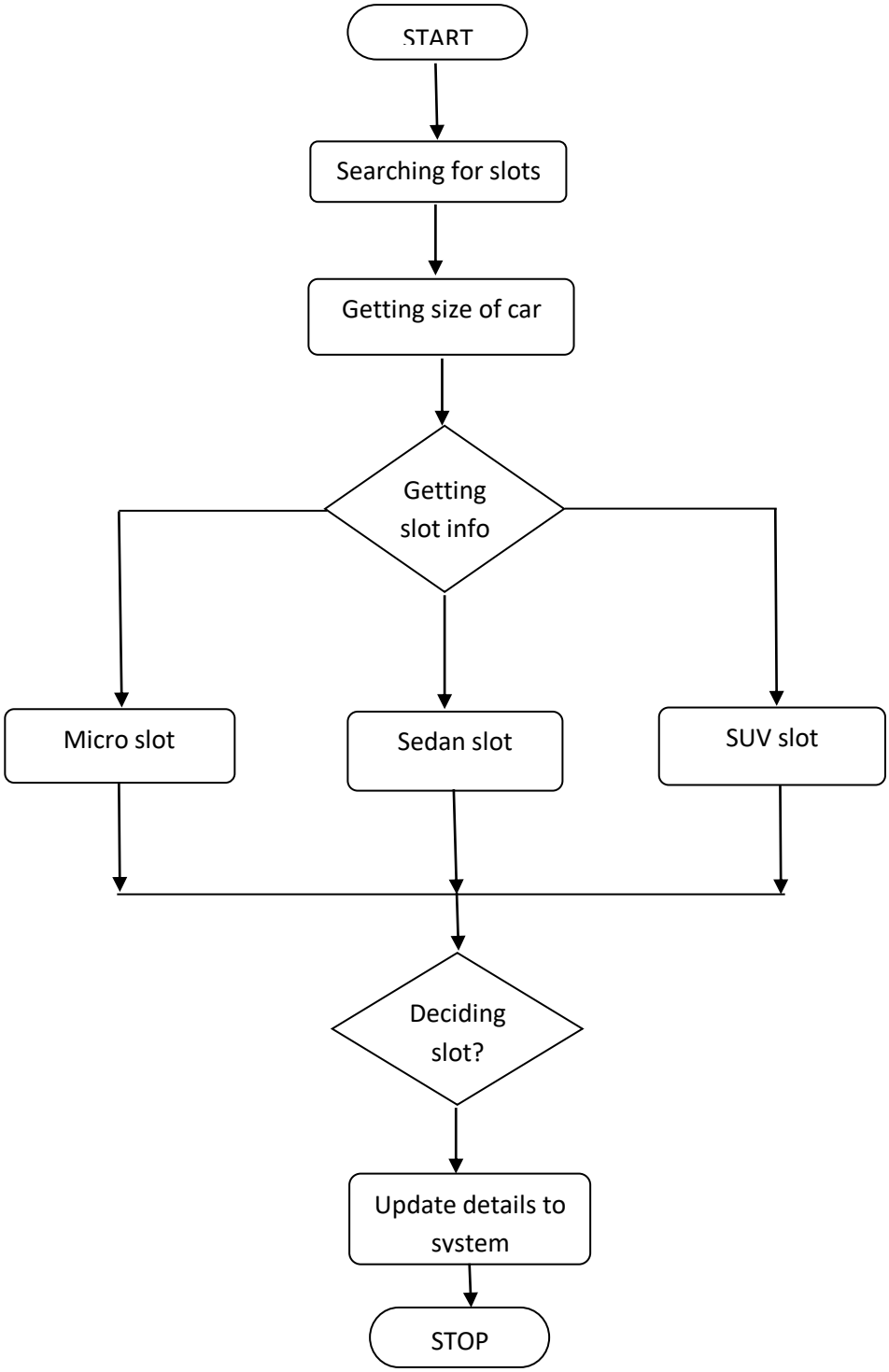


Figure2.1.2.2: Level-1 Diagram System structure.

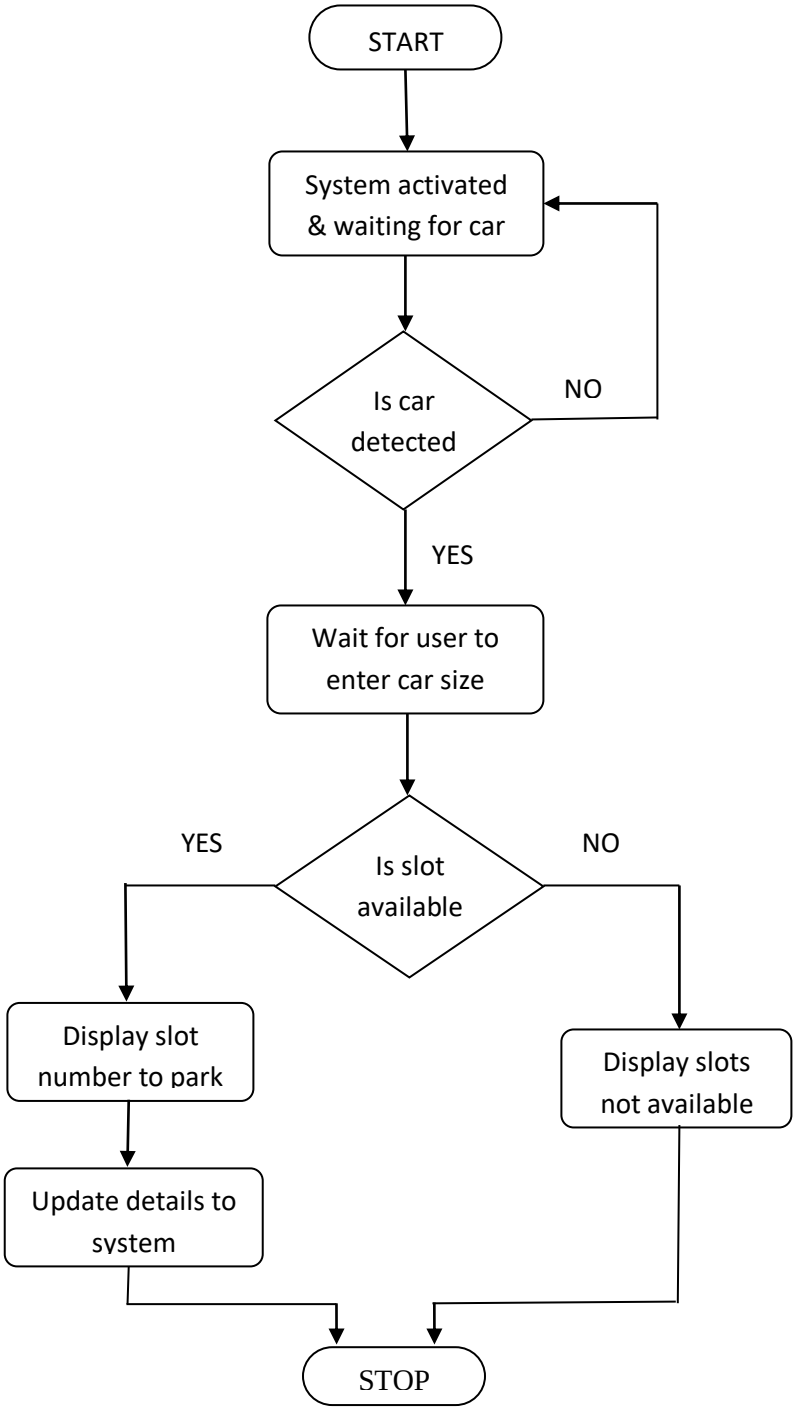


Figure2.1.2.3: Level-2 Diagram System view

CHAPTER 3

SYSTEM IMPLEMENTATION

Introduction to Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

Features

- Inexpensive
- Cross-platform
- Simple, clear environment
- Open source
- Extensible hardware support

Arduino Uno board structure

Technology specifications:

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 Ma
DC Current for 3.3V Pin	50 Ma
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by boot loader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz

Table 1: Tech specs

Block diagram

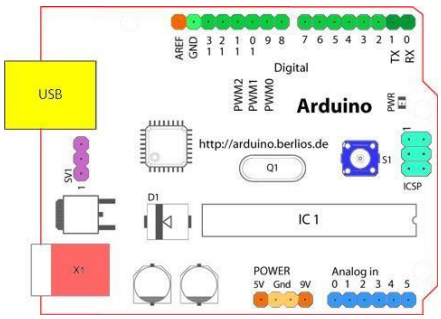


Fig 3.1: Arduino Uno R3

Pin configuration

Pin Category	Pin Name	Details
Power	Vin, 3.3V, 5V, GND	Vin: Input voltage to Arduino when using an external power source. 5V: Regulated power supply used to power microcontroller and other components on the board. 3.3V: 3.3V supply generated by on-board voltage regulator. Maximum current draw is 50mA. GND: ground pins.
Reset	Reset	Resets the microcontroller.
Analog Pins	A0 – A5	Used to provide analog input in the range of 0-5V
Input/Output Pins	Digital Pins 0 - 13	Can be used as input or output pins.
Serial	0(Rx), 1(Tx)	Used to receive and transmit TTL serial data.
External Interrupts	2, 3	To trigger an interrupt.
PWM	3, 5, 6, 9, 11	Provides 8-bit PWM output.
SPI	10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK)	Used for SPI communication.
Inbuilt LED	13	To turn on the inbuilt LED.
TWI	A4 (SDA), A5 (SCA)	Used for TWI communication.
AREF	AREF	To provide reference voltage for input voltage.

Table 2: Pin configurations

Introduction to Arduino IDE

The Arduino IDE is incredibly minimalistic, yet it provides a near-complete environment for most Arduino-based projects. The top menu bar has the standard options, including “File” (new, load save, etc.), “Edit” (font, copy, paste, etc.), “Sketch” (for compiling and programming), “Tools” (useful options for testing projects), and “Help”.

The middle section of the IDE is a simple text editor that where you can enter the program code. The bottom section of the IDE is dedicated to an output window that is used to see the status of the compilation, how much memory has been used, any errors that were found in the program, and various other useful messages.

Projects made using the Arduino are called sketches, and such sketches are usually written in a cut-down version of C++ (a number of C++ features are not included). Because programming a microcontroller is somewhat different from programming a computer, there are a number of device-specific libraries (e.g., changing pin modes, output data on pins, reading analog values, and timers). This sometimes confuses users who think Arduino is programmed in an "Arduino language." However, the Arduino is, in fact, programmed in C++. It just uses unique libraries for the device

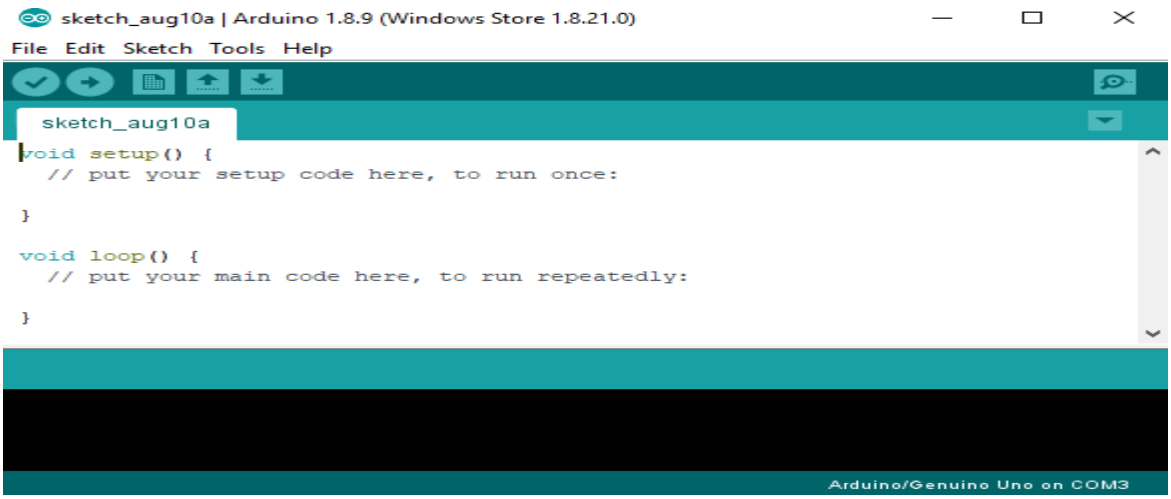


Fig 3.2: Arduino Ide in default mode

The 6 Buttons

While more advanced projects will take advantage of the built-in tools in the IDE, most projects will rely on the six buttons found below the menu bar.

Fig 3.3: The button bar

- The **check mark** is used to verify your code. Click this once you have written your code.
- The **arrow** uploads your code to the Arduino to run.
- The **dotted paper** will create a new file.
- The **upward arrow** is used to open an existing Arduino project.
- The **downward arrow** is used to save the current file.
- The far right button is a **serial monitor**, which is useful for sending data from the Arduino to the PC for debugging purposes

SOURCE CODE

Park.ino

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>

// servo object1
Servo servo1;
// servo object2
Servo servo2;

//entry gate sensor
const int servo_sensor1 = A4;
//exit gate sensor
const int servo_sensor2 = A5;

//servo1 connection
const int servo_1 =2;
//servo2 connection
const int servo_2 =3;

//object for servo1 positon
int value_1 =0;
//object for servo2 positon
int value_2 =0;


//buttons
const int micro=10;
const int sedan=11;
const int suv=12;

// Set the LCD address to 0x27 for a 16 chars and 2 line display
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2);

//micro sensors
const int s1=4;
const int s2=7;

//sedan sensors
const int s3=5;
const int s4=8;

//suv sensors
const int s5=6;
const int s6=9;

// objects
int a,b,c;
int p,q,r,s,t,u;

void setup()
{
  //initializing serial communication
  Serial.begin(9600);

  // initialize the LCD
  lcd.begin();
  // Turn on the blacklight and print a message.
  lcd.backlight();

  // setting sensors as input
  pinMode(servo_sensor1,INPUT);
  pinMode(servo_sensor2,INPUT);

  //attaching servos
```

```
  servo1.attach(servo_1);
  servo2.attach(servo_2);

  //setting buttons as input
  pinMode(micro,INPUT);
  pinMode(sedan,INPUT);
  pinMode(suv,INPUT);

  //setting sensors as input
  //micro
  pinMode(s1,INPUT);
  pinMode(s2,INPUT);
  //sedan
  pinMode(s3,INPUT);
  pinMode(s4,INPUT);
  //suv
  pinMode(s5,INPUT);
  pinMode(s6,INPUT);
}

void loop()
{
  //reading value from entry sensor
  value_1 = analogRead(servo_sensor1);
  //reading value from exit sensor
  value_2 = analogRead(servo_sensor2);

  // entry gate(servo) code

  // moving up
  {
    if (analogRead(A4)<500)
      {
```

```
      servo1.write(90);
      delay(5000);
    }
}
//moving down
if (analogRead(A4)>500)
    {
      servo1.write(0);
      delay(60);
    }

// exit gate(servo) code

// moving up
{
  if (analogRead(A5)<500)
    {
      servo2.write(90);
      delay(5000);
    }
}
//moving down
if (analogRead(A5)>500)
    {
      servo2.write(0);
      delay(60);
    }

//main logic

//reading button values
a=digitalRead(micro);
b=digitalRead(sedan);
```

```
  c=digitalRead(suv);

  //reading sensor values
  //micro
  p=digitalRead(s1);
  q=digitalRead(s2);
  //sedan
  r=digitalRead(s3);
  s=digitalRead(s4);
  //sedan
  t=digitalRead(s5);
  u=digitalRead(s6);

//micro
  if(a == 0 && p ==  1)
  {
    lcd.print("goto slot 1");
    delay(5000);
    lcd.clear();
  }
  else if(a == 0 && p == 0 && q==1)
  {
    lcd.print("goto slot 2");
    delay(5000);
    lcd.clear();
  }
  else if(a == 0 && p == 0 && q==0 && r==1)
  {
    lcd.print("goto slot 3");
    delay(5000);
    lcd.clear();
  }
  else if(a == 0 && p == 0 && q==0 && r==0)
```

```
  {
    lcd.print("sorry parking full");
    delay(5000);
    lcd.clear();
  }

// sedan
  if(b == 0 && r ==  1)
  {
    lcd.print("goto slot 3");
    delay(5000);
    lcd.clear();
  }
  else if(b == 0 && r == 0 && s==1)
  {
    lcd.print("goto slot 4");
    delay(5000);
    lcd.clear();
  }
  else if(b == 0 && r == 0 && s==0 && t==1)
  {
    lcd.print("goto slot 5");
    delay(5000);
    lcd.clear();
  }
  else if(b == 0 && r == 0 && s==0 && t==0)
  {
    lcd.print("sorry parking full");
    delay(5000);
    lcd.clear();
  }

// suv
```

```
 if(c == 0 && t ==  1)
 {
   lcd.print("goto slot 5");
   delay(5000);
   lcd.clear();
 }
 else if(c == 0 && t == 0 && u==1)
 {
   lcd.print("goto slot 6");
   delay(5000);
   lcd.clear();
 }
 else if(c == 0 && t == 0 && u==0)
 {
   lcd.print("sorry parking full");
   delay(5000);
   lcd.clear();
 }
}
```

CHAPTER 4

SYSTEM TESTING

Testing is the process of detection errors. Testing performs a very quality role for assurance and for ensuring the ability of software.

4.1 Sensors Testing

The main objective of testing is check whether every part used in project is completely working well.

4.1.1 Ir Sensor

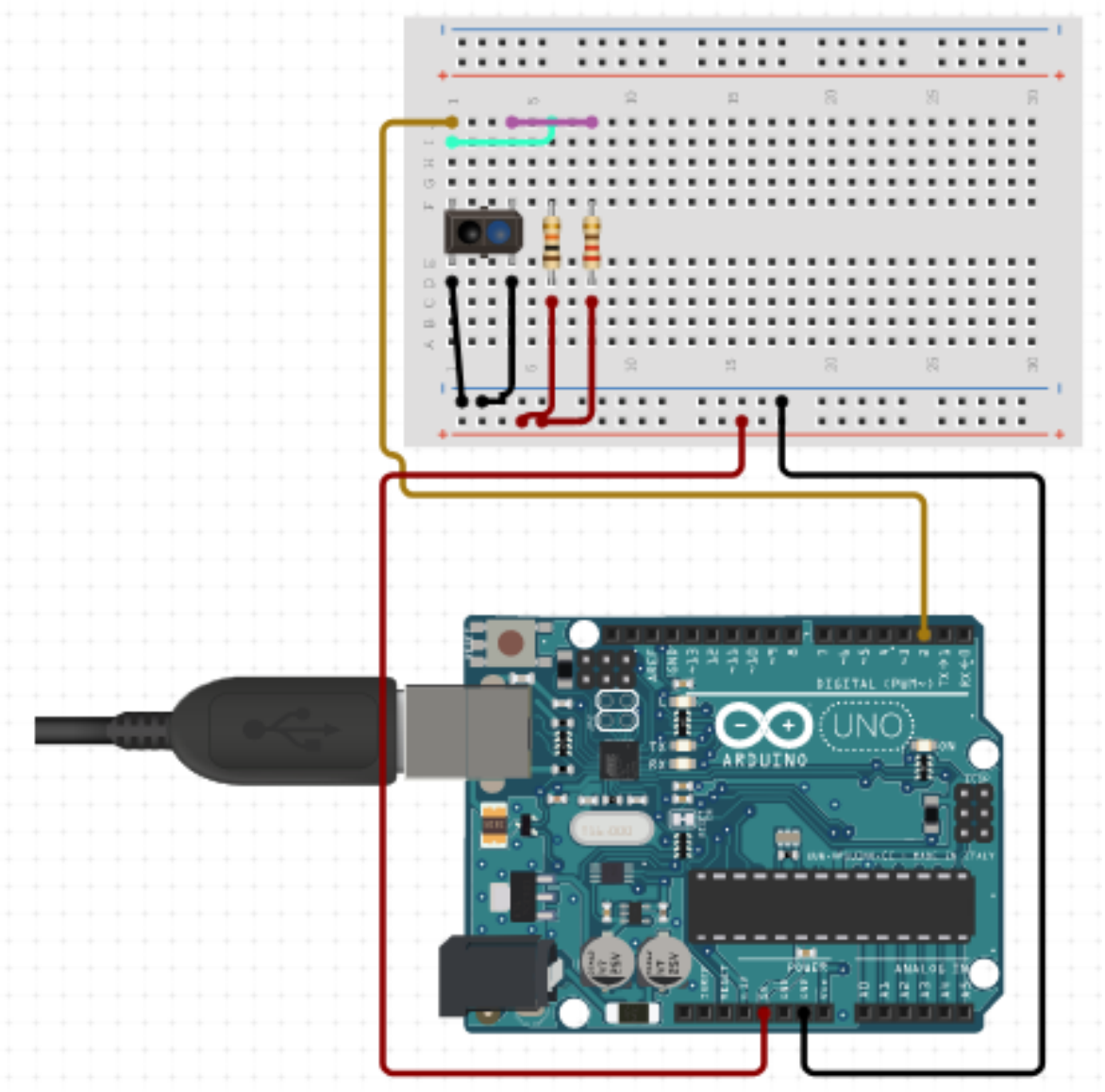


Fig 4.1.1.1 Ir Sensor

Connections

- The Ir sensor which consists of three pinouts vcc, gnd, out.
- The vcc pin is connected to 5v of arduino.
- The gnd pin is connected to gnd of arduino.
- The out pin is connected to digtal pin 3 of arduino.

Procedure

Now connect the sensors to arduino with the help of breadboard and jumper wires According to connections specified above connecting a single sensor to arduino and upload code and check working by seeing data in serial monitor.

Source Code

```
int sen=3;//sensor
int sen1=4;//sensor2
int a,b;
void setup()
{
 pinMode(sen,INPUT);
 pinMode(sen1,INPUT);
 Serial.begin(9600);
}
void loop()
{
  a=digitalRead(sen);
  b=digitalRead(sen1);
  if(a==LOW || b==LOW)
  {
   Serial.print("presence detected\n");
  }
  else if(a==HIGH ||b==LOW)
  {
```

```
  Serial.print("no presence\n");
 }
}
```

Output:

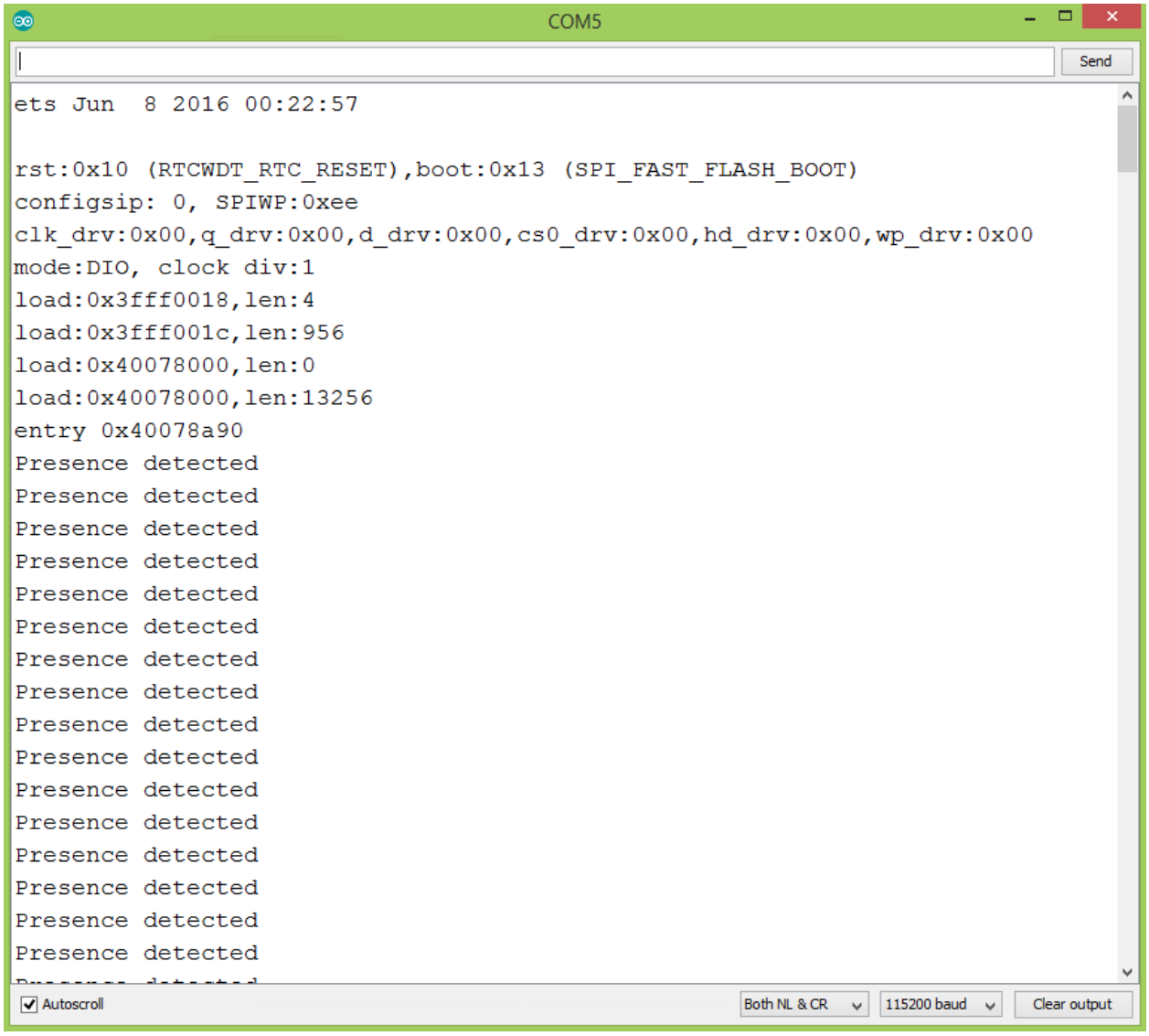


Fig 4.1.1.2 output of sensor testing

4.1.2 Servo

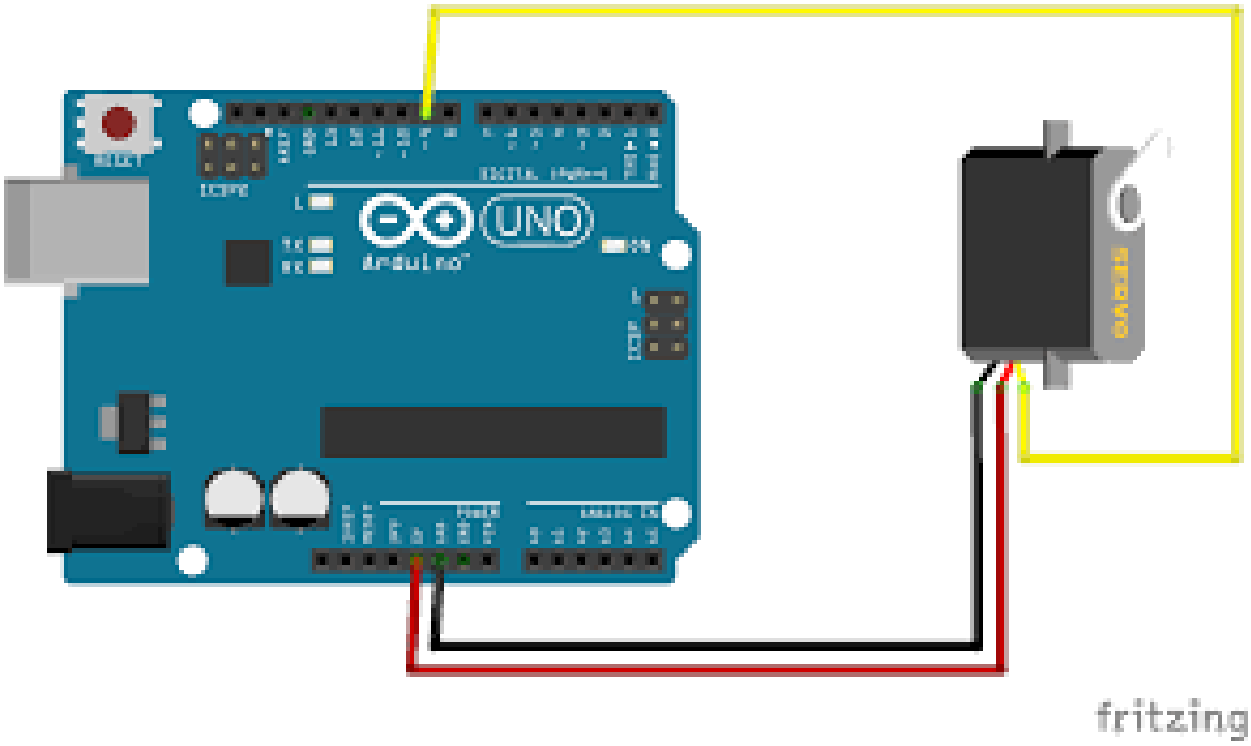


Fig 4.1.2.1 servo

Procedure

You can connect small servo motors directly to an Arduino to control the shaft position very precisely. Because servo motors use feedback to determine the position of the shaft, you can control that position very precisely. As a result, servo motors are used to control the position of objects, rotate objects, move legs, arms or hands of robots, move sensors etc. with high precision. Servo motors are small in size, and because they have built-in circuitry to control their movement, they can be connected directly to an Arduino.

Connections

The best thing about a servo motor is that it can be connected directly to an Arduino. Connect to the motor to the Arduino as shown in the table below:

- Servo red wire – 5V pin Arduino
- Servo brown wire – ground pin Arduino
- Servo yellow wire – PWM pin Arduino

Source Code

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

Output:

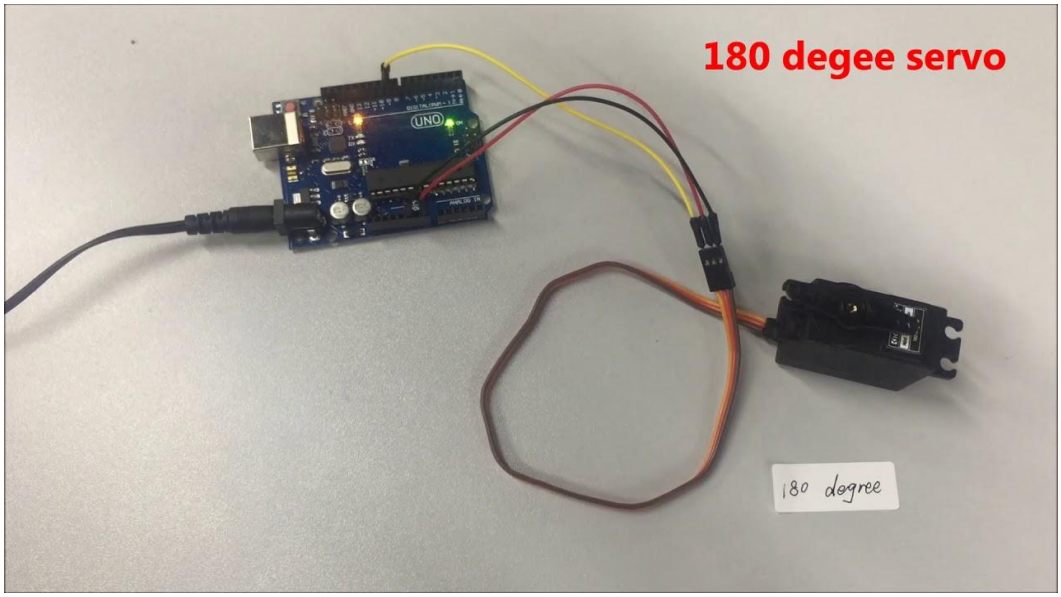


Fig 4.1.2.2 Servo output

4.2 Project Testing

Connections

- Connect Ir sensors to arduino according to it connections.
- Connect servo motor ground pin to ground of arduino.
- Connect servo motor input pins to digital input pins of arduino.
- Connect Dc battery pack to the servo motor for powering motors.
- Connect switches to System.

Procedure

- Connect 5v and ground pin of arduino to '+' and '-' of breadboard.
- Connect vcc and ground pins of Ir sensor to '+' and '-' of breadboard.
- Connect out pins of Ir sensors to digital pins of arduino.
- Connect the ground pin of servos to ground pin of arduino.
- Connect dc battery pack to servo motor.
- Connect servo motors control to input of Arduino.
- Connect arduino to system and upload the source code of the project to arduino.
- Powerup the whole system by switching battery pack and powering arduino.
- Set the components to system and test working of it.

Test Cases

Test case 1: Micro car entering parking area

Test case functionality:

A Micro car enters parking area and enters its size as input to system already one Micro slot is filled so second micro slot number is displayed in the screen.

Excepted output:

Second micro slot number should be displayed in the screen.

Obtained output:

Second micro Slot number is displayed.

Test case 2: Sedan car entering parking area

Test case functionality:

A Sedan car enters parking area and enters its size as input to system already both sedan slots are filled so next available size slot is displayed in the screen.

Excepted output:

Mega size slot number should be displayed in the system.

Obtained output:

Mega Slot number is displayed.

Test case 3: Suv car entering parking area

Test case functionality:

A Suv car enters parking area and enters its size as input to system already both suv slots are filled so next available size slot is displayed in the screen.

Excepted output:

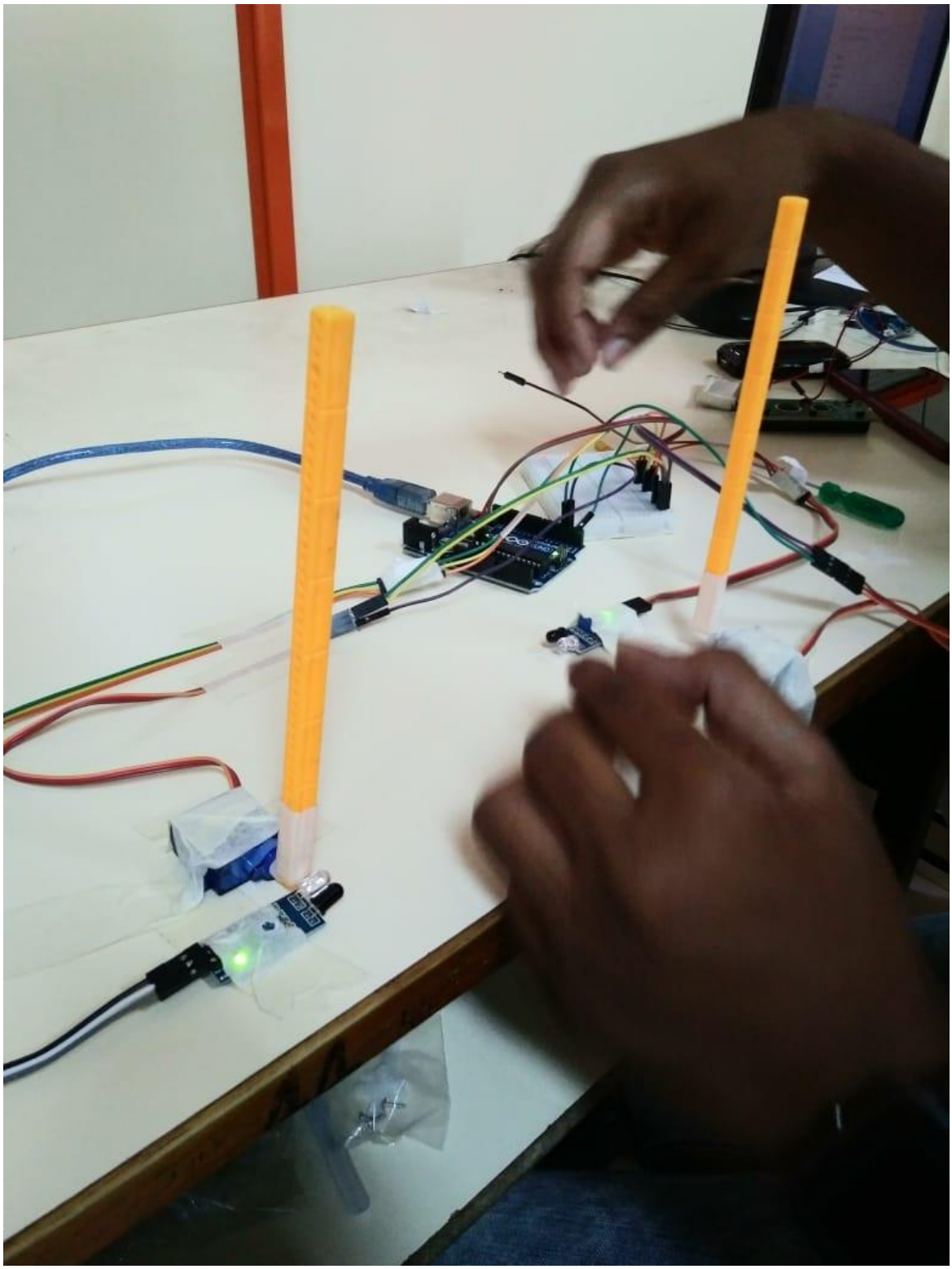
No slot number should be displayed in the system as there is no availability.

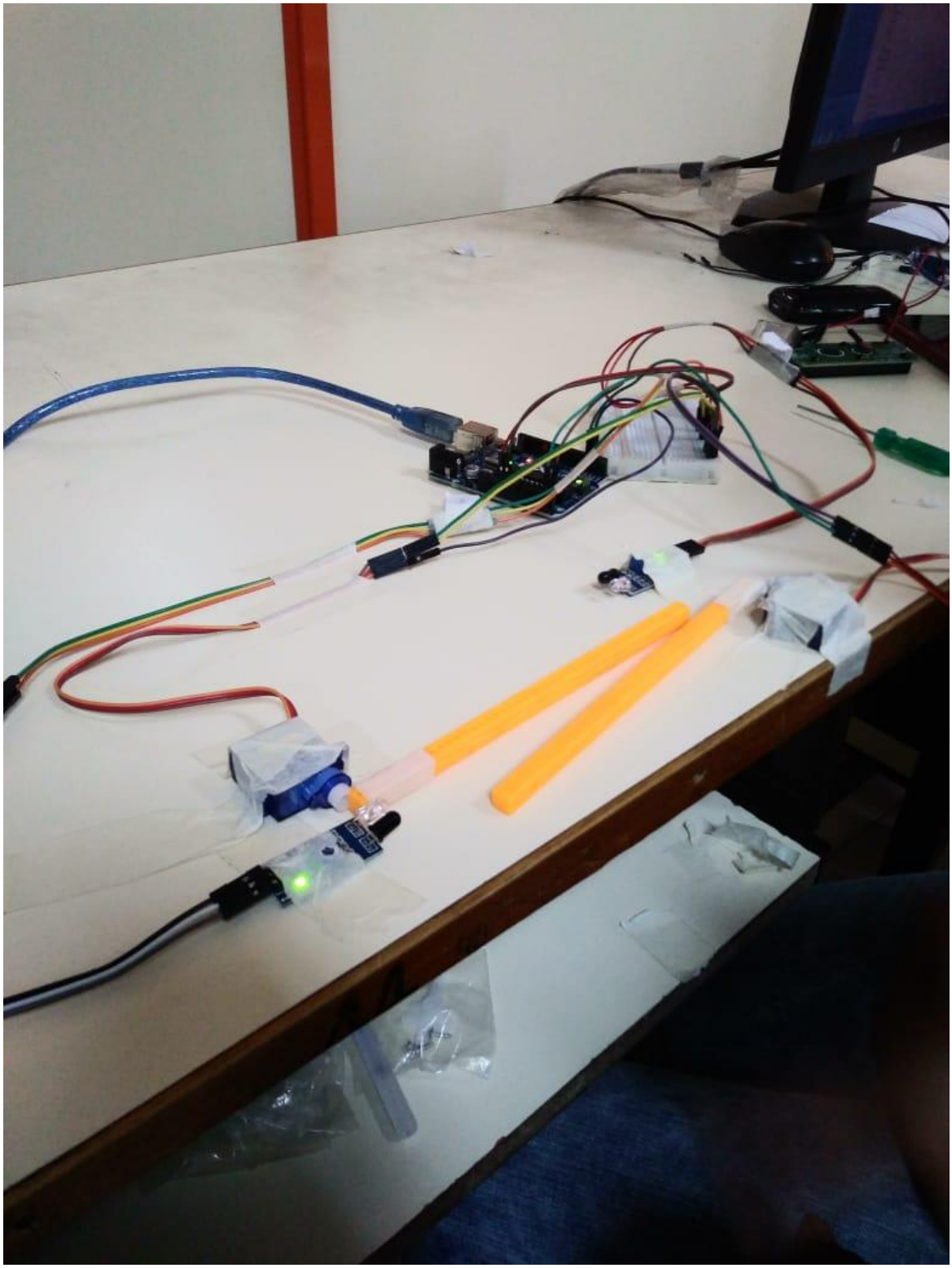
Obtained output:

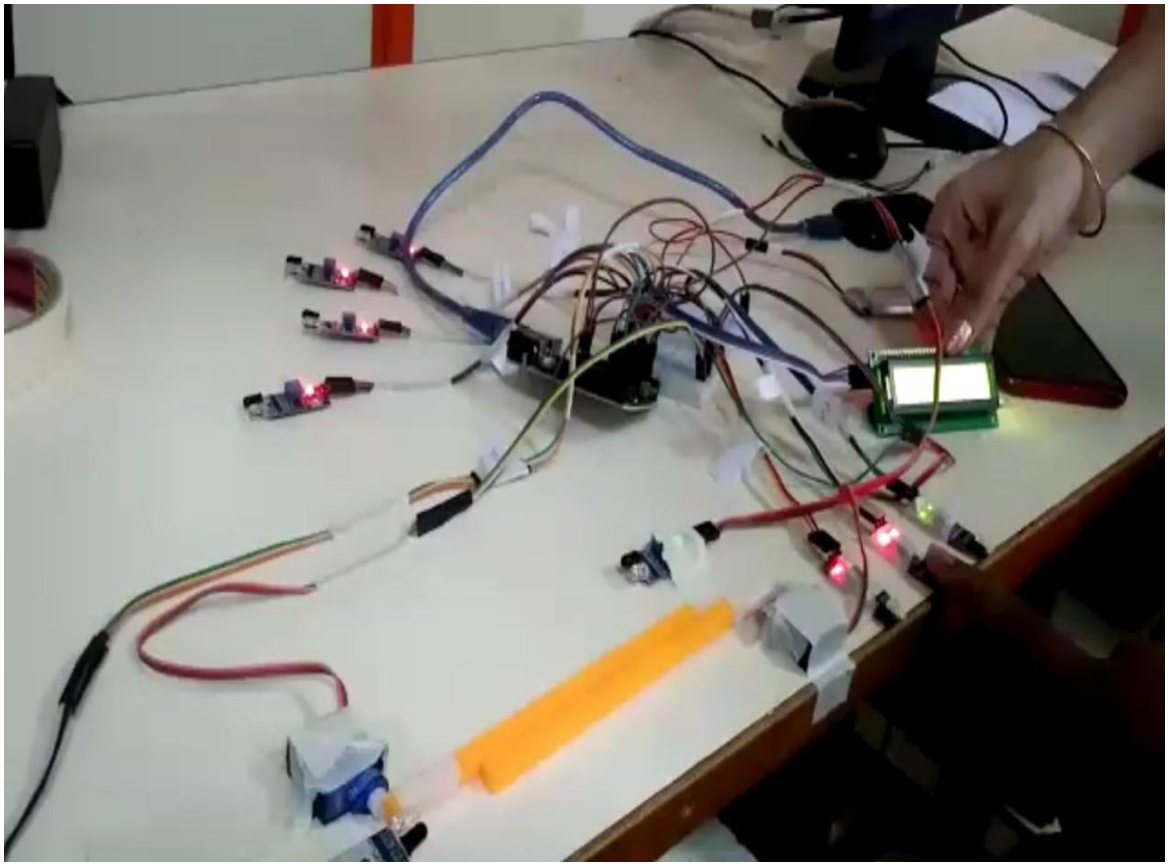
Parking full is displayed.

CHAPTER 5

SCREENS







CHAPTER 6

WORKING PROCEDURE

6.1 HOW TO OPERATE SYSTEM

- First, we need to power on the system to get started.
- Wait for system to get activated.
- Make sure the display is working.
- System starts and waits for vehicle to enter area.
- Give size as input through the buttons placed.
- Data will be processed in the system.
- Output will be displayed in LCD screen.
- If any sensor stops working power off system and replace it.
- If system is not working accurately just power off and on again.

CHAPTER 7

SCOPE FOR FUTURE DEVELOPMENT

Based on the results from this project and its benefits to the users in saving their time, we have created a user-friendly system. It also reduces overall fuel energy of the vehicle which is consumed in the search of the car.This project provides a real time process and information of the parking slots. In future works, this framework can be enhanced by including different applications, For Example, internet booking by utilizing GSM. The driver or client can book their parking area at home or while in transit to the shopping centre. This can diminish the season of the client to seeking the empty parking area. As a further review, distinctive sensor frameworks can be added to enhance this framework to distinguish the question and guide the driver or clients speediest. We will attempt to decrease the mechanical structure and attempt to make it eco-friendly.

CONCLUSION

Our project detects the empty slots and helps the drivers to find parking space in unfamiliar city. The average waiting time of users for parking their vehicles is effectively reduced in this system. The optimal solution is provided by the proposed system, where most of the vehicles find a free parking space successfully. Our preliminary test results show that the performance of the Arduino UNO based system can effectively satisfy the needs and requirements of existing car parking hassles thereby minimizing the time consumed to find vacant parking lot and real time information rendering. This smart parking system provides better performance, low cost and efficient large-scale parking system. It also eliminates unnecessary travelling of vehicles across the filled parking slots in a city.

REFERENCES

[1] https://www.softwebsolutions.com/resources/smart-parking-iot-solution.html

[2] https://www.instructables.com/id/Arduino-Servo-Motors/

[3] https://create.arduino.cc/projecthub/techmirtz/using-16x2-lcd-with-arduino-d89028

[4] https://www.makerguides.com/character-i2c-lcd-arduino-tutorial/

[5] https://www.hackerearth.com/blog/developers/a-tour-of-the-arduino-uno-board/

[6] https://www.programmingelectronics.com/tutorial-17-using-a-button-old-version/

[7] http://www.infiniteinformationtechnology.com/iot-smart-city-what-is-smart-parking

[8] https://create.arduino.cc/projecthub/Manikantsavadatti/how-to-make-an-ir-sensor-66786b