Smart Car Parking

Domain – IOT

Abstract :

Smart Car Parking IoT: Revolutionizing Urban Parking. Our IoT-based system streamlines urban parking. It uses sensors and cameras to relay real-time parking availability, making it easy for users to find and reserve spots via mobile apps. Payment processing is automated, reducing wait times. This innovative system improves urban mobility and sustainability, reducing congestion and enhancing convenience.

Problem Statement :

The traditional urban parking management system is inefficient and often leads to congestion, wasted time, and environmental concerns. This project aims to address these issues by implementing an IoT-based Smart Car Parking system that offers real-time parking space availability information, reservation options, and automated payment processing, ultimately improving urban mobility and sustainability.

Design Thinking :

Design Thinking is a human-centered approach to problem-solving and innovation that places a strong emphasis on empathy, creativity, and iteration. It typically involves the following stages:

1.

\*\*Empathize\*\*: Understand the needs and perspectives of the people you are designing for. This stage often involves research, interviews, and observation to gain deep insights into users’ experiences and challenges.

2.

\*\*Define\*\*: Clearly articulate the problem or challenge based on the insights gained during the empathy stage. This involves synthesizing information to create a specific problem statement.

3.

\*\*Ideate\*\*: Generate a wide range of creative ideas to address the defined problem. Encourage brainstorming and free thinking to come up with innovative solutions.

4.

\*\*Prototype\*\*: Create tangible representations of your ideas. These can be rough sketches, physical models, or digital prototypes. The goal is to quickly test and iterate on different concepts.

5.

\*\*Test\*\*: Gather feedback by testing your prototypes with users. This helps refine and improve the solutions. This stage often involves multiple iterations and adjustments.

6.

\*\*Implement\*\*: Once a refined solution is identified through testing, it is ready for implementation. This may involve developing a final product, service, or process.

7.

\*\*Iterate\*\*: Design Thinking is an iterative process, and it’s important to revisit and refine solutions based on real-world feedback and changing needs.

Design Thinking is a versatile methodology that can be applied to various fields, from product design to service improvement and social innovation. It encourages collaboration, creativity, and a deep understanding of the end-users’ needs and perspectives to develop solutions that truly meet their requirements.

Approach :

Your request is a bit broad, so I’ll provide a general approach that you can tailor to a specific context or problem:

\*\*Problem Identification and Understanding:\*\*

1.

Begin by clearly defining the problem or challenge you want to address. Ensure you have a deep understanding of its scope and impact.

\*\*Research and Analysis:\*\*

2.

Conduct research to gather information, data, and insights related to the problem. This might involve market research, user interviews, data analysis, and competitor analysis.

\*\*Goal Setting:\*\*

3.

Define specific, measurable, achievable, relevant, and time-bound (SMART) goals. These goals should guide your approach and provide a clear sense of direction.

\*\*Brainstorming and Idea Generation:\*\*

4.

Encourage brainstorming sessions to generate a wide range of potential solutions or approaches to the problem. Foster a creative environment where all ideas are welcome.

\*\*Evaluation and Prioritization:\*\*

5.

Evaluate the generated ideas or approaches against your goals and criteria. Prioritize them based on feasibility, impact, and alignment with your objectives.

\*\*Prototyping and Testing:\*\*

6.

Develop prototypes or minimum viable products (MVPs) for the selected approaches. Test them with users or stakeholders to gather feedback and make improvements.

\*\*Implementation:\*\*

7.

Once you have a refined approach or solution, proceed with implementation. This may involve creating a detailed plan, allocating resources, and executing the chosen approach.

\*\*Monitoring and Iteration:\*\*

8.

Continuously monitor the progress of your approach and gather feedback. Be prepared to make adjustments and iterations based on real-world results and changing circumstances.

\*\*Communication and Collaboration:\*\*

9.

Throughout the entire process, maintain open communication with stakeholders, team members, and relevant parties. Collaboration and transparency are key to success.

\*\*Documentation:\*\*

10.

Document your approach, decisions, and outcomes. This documentation can serve as a reference for future projects and help with knowledge sharing.

Remember that the specific approach you take can vary greatly depending on the nature of the problem, your goals, and the resources available. Flexibility and adaptability are important, as you may need to adjust your approach based on feedback and new information.

***PHASE 2: INNOVATION***

* IOT is often used in smart parking systems to connect various devices and sensors, allowing for real-time data exchange and efficient management. This connectivity enables better monitoring, control, and automation of parking operations.
* An IOT based smart parking system, also known as a connected parking

system, is a centralized management system that allows drivers to use a smart phone app to search for and reserve a parking spot.

# PROJECT PLANNING:

|  |  |
| --- | --- |
|  |  |
|  |  |

The traditional urban parking management system is inefficient and often leads to congestion, wasted time, and environmental concerns. This project aims to address these issues by implementing an IOT-based Smart Car Parking system that offers real-time parking space availability information, reservation options, and automated payment processing, ultimately improving urban mobility and sustainability.

**HARDWARE COMPONENTS:**

* Choose the necessary IOT hardware components are Parking Sensors , Communication Devices , Servers and Databases , Network Infrastructure ,Display Devices and Power Supply.
* Select sensors based on the parameters you to plan the parking slot to park the vehicles in that place.

**SOFTWARE COMPONENTS:**

* Choose the necessary IOT Software components are Sensor Data Management Software, Mobile Application, Data Analytics and Reporting, Integration APIs, Security and Authentication.
* Using the sensors only we detect the parking slot vacant correctly in a python code crush .

# Sensors:

* Smart parking systems often utilize sensors installed in parking spaces that detect the presence of a vehicle. These sensors can be either surface-mounted or embedded in the ground and provide real-time data on parking space availability.
* These are essential components of a smart parking system. Sensors can be in-ground or above-ground devices equipped with various technologies such as ultrasonic, infrared, or magnetic fields to detect the presence of vehicles in parking spaces. They relay real-time information about parking occupancy, helping drivers locate available parking spots.

**Mobile Applications:**

* Many smart parking solutions offer mobile applications that allow users to easily find and reserve parking spaces in advance. These apps often provide detailed information about parking availability, pricing, and even navigation to the desired parking spot.
* A mobile application enables drivers to access real-time parking information on their smart phones. It provides features like finding available parking spaces, navigation to the nearest available spot, and making parking reservations or payments. The app can also send notifications or alerts to drivers regarding parking availability or time limits.

**Data Analytics:**

Smart parking systems collect and analyze data on parking space occupancy, duration, turnover, and more. This data can be used to optimize parking operations, predict parking demand, and make informed decisions to improve overall efficiency.

**Guidance Systems:**

Smart parking guidance systems use digital signage or mobile apps to guide drivers to available parking spaces within a parking facility. This helps reduce the time spent searching for a parking spot and improves traffic flow.

**Signage and displays:**

Visual displays, such as electronic signage or LED panels, can be installed at different points within the parking facility to guide drivers toward available parking spaces. This helps reduce traffic congestion and improve the overall parking experience.

**Controlling parking systems using sensors :**

**1. Sensor Installation:**

Sensors are installed in individual parking spaces to detect the presence of vehicles. These sensors can be embedded in the ground, mounted on walls, or placed on existing infrastructure like lampposts.  
   
2. Data Collection:

When a car enters or leaves a parking space, the sensor detects the change and sends a signal to a central system. This data collection happens in real-time and provides accurate information about the occupancy status of each parking space.  
   
3. Occupancy Monitoring:

The central system collects and analyzes the data from the sensors to determine the availability of parking spaces. It can create a live map or database that shows the status of each parking spot, indicating if it is occupied or vacant.  
   
  
**. Information Display:**

The availability information collected from the sensors can be displayed to drivers in various ways. This can be through digital signage at the entrance of the parking facility, a mobile app, or on websites. Drivers can quickly see which parking spaces are available and make informed decisions about where to park.  
   
5. Navigation and Reservation:

By integrating sensor data with navigation systems or parking apps, drivers can be guided to the nearest available parking spaces. Some systems even allow for reservation of parking spots in advance, ensuring a guaranteed parking space.

**6. Automated Enforcement:**

In some cases, parking systems can be integrated with enforcement mechanisms. For example, if a vehicle exceeds its allocated parking time, the sensors can alert parking enforcement personnel to take necessary action.

* Here some of the combination of these technologies helps optimize parking resource utilization, improve traffic flow, minimize congestion, and enhance the overall parking experience.

* By implementing parking sensors, smart parking systems help optimize parking resource utilization, reduce congestion, and enhance the overall parking experience for both drivers and parking operators.

|  |  |
| --- | --- |
|  |  |

* Parking sensors provide real-time data on parking space occupancy, which can be relayed to users through mobile applications or digital signage. This allows drivers to quickly find available parking spaces, reducing the time spent searching for a spot and improving overall parking efficiency.

* The specific software needs for a smart parking system using sensors may vary depending on the requirements and customization of the project. Partnering with a smart parking system provider or consulting with professionals in the field can help determine the appropriate software components for your specific implementation.

* Overall, sensor-based parking systems offer a more efficient, user-friendly, and sustainable approach to managing parking spaces. They optimize resource allocation, improve traffic flow, and enhance the overall experience for drivers.

**Smart Parking System Project Concept**

Smart car parking systems using IOT, which include sensors and microcontrollers, can be available in each parking slot. Smart parking system using IOT has smart phones and other sensors added into an interconnected system to determine parking space or level and provide real-time feedback. It is accomplished by using cameras, counters on the doors or gates of parking lots, sensors embedded in the paved area of individual parking lots, among other locations, depending on the deployment.

**Component Required**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Components** | **Quantity** |
| 1. | Arduino or Microcontroller | 1 |
| 2. | ICD 1602 | 1 |
| 3. | Servo Motors | 2 |
| 4. | Register | 8 |
| 5. | Buzzer | 1 |
| 6. | Push Button | 4 |
| 7. | LED Bulb | 8 |
| 8. | Slide Switch | 4 |
| 9. | Cables and Connectors | As Needed |

**Tools Required**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Tools** | **Examples** |
| 1. | Arduino IDE | Arduino IDE |
| 2. | CAD Software (Optional) | Autodesk Eagle. |
| 3. | Version Control System | GitHub, GitHub Desktop. |
| 4. | Project Management Software | ThingSpeak |
| 5. | Simulation Software (Optional) | Wokwi |
| 6. | Data Analysis Tools | Excel. |
| 7. | Communication Tools | Microsoft Teams, Discord, Zoom |

**Circuit Diagram of Smart Parking**

**Working principle**

The working principle of a typical car parking system involves several components and technologies:

Entry and Exit Gates: The system usually has entry and exit gates controlled by sensors or attendants. These gates regulate the flow of vehicles in and out of the parking facility.

Sensors: Sensors, such as ultrasonic or infrared sensors, are used to detect the presence of vehicles in parking spaces. These sensors send signals to a central control system.

Central Control System: This system processes the data from the sensors and manages the availability of parking spaces. It directs incoming vehicles to open spots and keeps track of occupied spaces.

Remote Management: parking systems are remotely monitored and managed through software, enabling operators to track occupancy, revenue, and perform maintenance tasks.

**Arduino Code**

#include <Servo.h>

#include <LiquidCrystal\_I2C.h>

uint8\_t indoor1 = 6;

uint8\_t indoor2 = 8;

uint8\_t outdoor1 = 7;

uint8\_t outdoor2 = 9;

uint8\_t buzzer = 12;

uint8\_t sncar[]={22,23,24,25};

uint8\_t outputgreen[]={26,27,28,29};

uint8\_t outputred []={30,31,32,33};

int count1 = 0;

boolean enablecount = false;

byte datax = 0b0000;

byte indexcar = 0x00;

LiquidCrystal\_I2C lcd(0x27,16,2);

Servo myservo1;

Servo myservo2;

void setup() {

Serial.begin(9600);

Serial.println("program start");

for(uint8\_t index = 0; index<4; index+=1){

pinMode(sncar[index],INPUT);

pinMode(outputgreen[index],OUTPUT);

pinMode(outputred[index],OUTPUT);

digitalWrite(sncar[index],LOW);

digitalWrite(outputgreen[index],LOW);

digitalWrite(outputred[index],LOW);

}

pinMode(indoor1,INPUT);

pinMode(indoor2,INPUT);

pinMode(outdoor1,INPUT);

pinMode(outdoor2,INPUT);

pinMode(buzzer,OUTPUT);

myservo1.attach(10);

myservo2.attach(11);

lcd.init(); // initialize the lcd

lcd.backlight();

lcd.clear();

delay(100);

}

void loop() {

/\* static unsigned long timer1 = millis();

if((millis()-timer1)>=1000){

timer1 = millis();

Serial.print("timeq1 = ");

Serial.print(timer1/1000);

Serial.println(" sec");

}\*/

if(enablecount==true){

// Serial.println("yeet");

count1+=1;

if(count1>=100){

myservo1.write(90);

enablecount = false;

Serial.println("ok");

count1=0;

}

}

for(byte i =0; i<4;i+=1){

bitWrite(datax,i,digitalRead(sncar[i]));

}

for(byte i =0; i<4;i+=1){

bitWrite(datax,i,digitalRead(sncar[i]));

//Serial.print(bitRead(datax,i),BIN);

indexcar = indexcar+!bitRead(datax,i);

if(i==3){

//Serial.println("");

lcd.setCursor(0,0);

lcd.print("Total : "+String(indexcar)+" car");

lcd.setCursor(0,1);

if(indexcar==4){

lcd.print(String(!bitRead(datax,0))+" "+String(!bitRead(datax,1))+" "+String(!bitRead(datax,2))+" "+String(!bitRead(datax,3))+" "+"FULL CAR");

}

else{

lcd.print(String(!bitRead(datax,0))+" "+String(!bitRead(datax,1))+" "+String(!bitRead(datax,2))+" "+String(!bitRead(datax,3))+" "+"Emptycar");

}

indexcar = 0;

}

}

if(digitalRead(indoor1)==0 && datax!= 0b0000){

myservo1.write(180);

count1=0;

}

if(digitalRead(outdoor1)==0 || datax == 0b0000 ){

enablecount = true;

}

if(digitalRead(indoor2)==0){

myservo2.write(180);

}

if(digitalRead(outdoor2)==0){

myservo2.write(90);

}

for(uint8\_t index = 0; index<4; index+=1){

digitalWrite(outputgreen[index],!digitalRead(sncar[index]));

digitalWrite(outputred[index],digitalRead(sncar[index]));

}

delay(10);

}

#include <Servo.h>

#include <LiquidCrystal\_I2C.h>

uint8\_t indoor1 = 6;

uint8\_t indoor2 = 8;

uint8\_t outdoor1 = 7;

uint8\_t outdoor2 = 9;

uint8\_t buzzer = 12;

uint8\_t sncar[]={22,23,24,25};

uint8\_t outputgreen[]={26,27,28,29};

uint8\_t outputred []={30,31,32,33};

int count1 = 0;

boolean enablecount = false;

byte datax = 0b0000;

byte indexcar = 0x00;

LiquidCrystal\_I2C lcd(0x27,16,2);

Servo myservo1;

Servo myservo2;

void setup() {

Serial.begin(9600);

Serial.println("program start");

for(uint8\_t index = 0; index<4; index+=1){

pinMode(sncar[index],INPUT);

pinMode(outputgreen[index],OUTPUT);

pinMode(outputred[index],OUTPUT);

digitalWrite(sncar[index],LOW);

digitalWrite(outputgreen[index],LOW);

digitalWrite(outputred[index],LOW);

}

pinMode(indoor1,INPUT);

pinMode(indoor2,INPUT);

pinMode(outdoor1,INPUT);

pinMode(outdoor2,INPUT);

pinMode(buzzer,OUTPUT);

myservo1.attach(10);

myservo2.attach(11);

lcd.init(); // initialize the lcd

lcd.backlight();

lcd.clear();

delay(100);

}

void loop() {

/\* static unsigned long timer1 = millis();

if((millis()-timer1)>=1000){

timer1 = millis();

Serial.print("timeq1 = ");

Serial.print(timer1/1000);

Serial.println(" sec");

}\*/

if(enablecount==true){

// Serial.println("yeet");

count1+=1;

if(count1>=100){

myservo1.write(90);

enablecount = false;

Serial.println("ok");

count1=0;

}

}

for(byte i =0; i<4;i+=1){

bitWrite(datax,i,digitalRead(sncar[i]));

}

for(byte i =0; i<4;i+=1){

bitWrite(datax,i,digitalRead(sncar[i]));

//Serial.print(bitRead(datax,i),BIN);

indexcar = indexcar+!bitRead(datax,i);

if(i==3){

//Serial.println("");

lcd.setCursor(0,0);

lcd.print("Total : "+String(indexcar)+" car");

lcd.setCursor(0,1);

if(indexcar==4){

lcd.print(String(!bitRead(datax,0))+" "+String(!bitRead(datax,1))+" "+String(!bitRead(datax,2))+" "+String(!bitRead(datax,3))+" "+"FULL CAR");

}

else{

lcd.print(String(!bitRead(datax,0))+" "+String(!bitRead(datax,1))+" "+String(!bitRead(datax,2))+" "+String(!bitRead(datax,3))+" "+"Emptycar");

}

indexcar = 0;

}

}

if(digitalRead(indoor1)==0 && datax!= 0b0000){

myservo1.write(180);

count1=0;

}

if(digitalRead(outdoor1)==0 || datax == 0b0000 ){

enablecount = true;

}

if(digitalRead(indoor2)==0){

myservo2.write(180);

}

if(digitalRead(outdoor2)==0){

myservo2.write(90);

}

for(uint8\_t index = 0; index<4; index+=1){

digitalWrite(outputgreen[index],!digitalRead(sncar[index]));

digitalWrite(outputred[index],digitalRead(sncar[index]));

}

delay(10);

}

{

"version": 1,

"author": "Joice britto",

"editor": "wokwi",

"parts": [

{ "type": "wokwi-arduino-mega", "id": "mega", "top": 197.28, "left": -14.43, "attrs": {} },

{

"type": "wokwi-buzzer",

"id": "bz1",

"top": 86.14,

"left": 126.88,

"attrs": { "volume": "0.2" }

},

{

"type": "wokwi-lcd1602",

"id": "lcd1",

"top": 27.26,

"left": 372.32,

"attrs": { "pins": "i2c", "background": "blue", "color": "white" }

},

{

"type": "wokwi-pushbutton",

"id": "btn4",

"top": 1.31,

"left": 267.73,

"attrs": { "color": "red" }

},

{

"type": "wokwi-pushbutton",

"id": "btn5",

"top": 0.84,

"left": 173.2,

"attrs": { "color": "green" }

},

{

"type": "wokwi-pushbutton",

"id": "btn6",

"top": 3.5,

"left": -37.33,

"attrs": { "color": "red" }

},

{

"type": "wokwi-pushbutton",

"id": "btn7",

"top": 3.42,

"left": -124.81,

"attrs": { "color": "green" }

},

{ "type": "wokwi-slide-switch", "id": "sw1", "top": 192.7, "left": 398.9, "attrs": {} },

{ "type": "wokwi-slide-switch", "id": "sw2", "top": 192.15, "left": 557.93, "attrs": {} },

{ "type": "wokwi-slide-switch", "id": "sw3", "top": 191.08, "left": 508.93, "attrs": {} },

{ "type": "wokwi-slide-switch", "id": "sw4", "top": 193.63, "left": 458.67, "attrs": {} },

{

"type": "wokwi-resistor",

"id": "r1",

"top": -58.5,

"left": 122.24,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-resistor",

"id": "r2",

"top": -59.02,

"left": 260.94,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-resistor",

"id": "r3",

"top": -58.36,

"left": -32.96,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-resistor",

"id": "r4",

"top": -57.7,

"left": -127.19,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-resistor",

"id": "r5",

"top": 412.62,

"left": 570.89,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-resistor",

"id": "r6",

"top": 362.48,

"left": 510.79,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-resistor",

"id": "r7",

"top": 336.04,

"left": 435.75,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-resistor",

"id": "r8",

"top": 303.1,

"left": 396.12,

"attrs": { "value": "1000" }

},

{

"type": "wokwi-led",

"id": "led1",

"top": 78.27987252863988,

"left": -196.04819410029904,

"attrs": { "color": "red" }

},

{

"type": "wokwi-led",

"id": "led2",

"top": 93.27987252863988,

"left": -181.04819410029904,

"attrs": { "color": "red" }

},

{

"type": "wokwi-led",

"id": "led3",

"top": 108.27987252863988,

"left": -166.04819410029904,

"attrs": { "color": "red" }

},

{

"type": "wokwi-led",

"id": "led4",

"top": 123.28,

"left": -151.05,

"attrs": { "color": "green" }

},

{

"type": "wokwi-led",

"id": "led5",

"top": 138.28,

"left": -136.05,

"attrs": { "color": "green" }

},

{

"type": "wokwi-led",

"id": "led6",

"top": 153.27987252863988,

"left": -121.04819410029904,

"attrs": { "color": "green" }

},

{

"type": "wokwi-led",

"id": "led7",

"top": 168.27987252863988,

"left": -106.04819410029904,

"attrs": { "color": "green" }

},

{

"type": "wokwi-led",

"id": "led8",

"top": 65.16,

"left": -212.17,

"attrs": { "color": "red" }

},

{ "type": "wokwi-servo", "id": "servo1", "top": -245.3, "left": -155.87, "attrs": {} },

{ "type": "wokwi-servo", "id": "servo2", "top": -242.77, "left": 130.29, "attrs": {} }

],

"connections": [

[ "lcd1:SCL", "mega:21", "green", [ "h0" ] ],

[ "lcd1:SDA", "mega:20", "green", [ "h0" ] ],

[ "lcd1:GND", "mega:GND.1", "black", [ "h0" ] ],

[ "lcd1:VCC", "mega:5V", "red", [ "h-384.61", "v332.71", "h174.72" ] ],

[ "mega:12", "bz1:2", "green", [ "v-21.94", "h37.5" ] ],

[ "mega:GND.1", "bz1:1", "black", [ "v-32.01", "h24.01" ] ],

[

"mega:22",

"sw1:2",

"green",

[ "v1.52", "h-11.78", "v-26.02", "h43.83", "v46.57", "h33.56" ]

],

[ "sw4:2", "mega:23", "green", [ "v16.09", "h-106.25", "v-28.52" ] ],

[ "mega:24", "sw3:2", "green", [ "v27.2", "h54.49" ] ],

[ "mega:25", "sw2:2", "green", [ "v2.18", "h10.18", "v32.17", "h182.28" ] ],

[ "sw1:3", "sw4:3", "green", [ "v-0.56", "h16.85", "v-40.94", "h54.37", "v42.29" ] ],

[ "sw3:3", "sw4:3", "green", [ "v1.74", "h10.85", "v-42.29", "h-46.31", "v42.29" ] ],

[ "sw2:3", "sw3:3", "green", [ "v1.34", "h18.23", "v-42.29", "h-52.35", "v38.93" ] ],

[ "sw2:3", "mega:GND.3", "green", [ "v178.98", "h-415.94" ] ],

[ "btn7:2.r", "btn6:2.r", "green", [ "h-1.84", "v26.77", "h79.71" ] ],

[ "btn5:2.r", "btn4:2.r", "green", [ "h5.21", "v20.49", "h77.74" ] ],

[ "lcd1:GND", "btn4:2.r", "black", [ "h0" ] ],

[ "btn6:2.r", "mega:GND.1", "green", [ "h11.06", "v32.59", "h63.96" ] ],

[

"btn7:1.l",

"mega:6",

"green",

[ "h-15.56", "v-21.05", "h231.04", "v96.34", "h120.41", "v104.94", "h-25.39" ]

],

[

"btn6:1.l",

"mega:7",

"green",

[ "h-12.25", "v-35.66", "h157.41", "v115.94", "h111.71", "v95.63", "h-41.47" ]

],

[ "btn5:1.l", "mega:8", "green", [ "h-51.83", "v54.17", "h106.63", "v115.1", "h-58.39" ] ],

[

"btn4:1.l",

"mega:9",

"green",

[ "h-22.8", "v-35.16", "h-126.94", "v101.56", "h123.56", "v111.71", "h-87.17" ]

],

[ "r2:1", "btn4:1.l", "green", [ "v0" ] ],

[ "r1:1", "btn5:1.l", "green", [ "v43.02", "h51.22" ] ],

[ "r3:1", "btn6:1.l", "green", [ "v8.68", "h-15.34" ] ],

[ "r4:1", "btn7:1.l", "green", [ "v0" ] ],

[ "r4:2", "r3:2", "green", [ "v-21.09", "h100.23" ] ],

[ "r1:2", "r3:2", "green", [ "v-27.43", "h-87.32" ] ],

[ "r2:2", "r1:2", "green", [ "v-22.15", "h-124.88" ] ],

[ "r4:2", "lcd1:VCC", "green", [ "v-19.9", "h-87.79", "v110.67" ] ],

[ "r8:2", "r7:2", "green", [ "v10.62", "h-40.75", "v48.96", "h78.1" ] ],

[ "r6:2", "r7:2", "green", [ "v22.34", "h-67.99" ] ],

[ "r5:2", "r6:2", "green", [ "v33.98", "h-93.12", "v-69.94", "h31.47", "v2.33" ] ],

[ "mega:5V", "r5:2", "red", [ "v93.51", "h6.06" ] ],

[ "led7:A", "mega:33", "green", [ "v222.87", "h463.49" ] ],

[ "led6:A", "mega:32", "green", [ "v243.05", "h479.34" ] ],

[ "led5:A", "mega:31", "green", [ "v268.51", "h499.89" ] ],

[ "led4:A", "mega:30", "green", [ "v285.63", "h510.94" ] ],

[ "led3:A", "mega:29", "green", [ "v289.31", "h533.27" ] ],

[ "led2:A", "mega:28", "green", [ "v311.87", "h487.53", "v-200.2" ] ],

[ "led1:A", "mega:27", "green", [ "v336.35", "h543.2" ] ],

[ "led8:A", "mega:26", "green", [ "v329.6", "h572.61" ] ],

[ "led8:C", "led1:C", "green", [ "v27.73", "h-1.38" ] ],

[ "led2:C", "led8:C", "green", [ "v11.46", "h-28.85" ] ],

[ "led3:C", "led8:C", "green", [ "v8.3", "h-46.59" ] ],

[ "led4:C", "led8:C", "green", [ "v8.8", "h-61.59" ] ],

[ "led5:C", "led8:C", "green", [ "v2.91", "h-75.68" ] ],

[ "led6:C", "led8:C", "green", [ "v2.27", "h-89.28" ] ],

[ "led7:C", "led8:C", "green", [ "v0.97", "h-105.99" ] ],

[ "mega:GND.2", "led7:C", "black", [ "v33.4", "h-158.84" ] ],

[ "servo1:V+", "servo2:V+", "green", [ "h-16.79", "v-60.36", "h262.46", "v63.66" ] ],

[ "servo1:GND", "servo2:GND", "black", [ "h-5.62", "v-68.73", "h274.74", "v55.84" ] ],

[ "mega:10", "servo1:PWM", "green", [ "v-29.03", "h-79.19", "v-308.97", "h-218.93" ] ],

[ "servo2:PWM", "mega:11", "green", [ "h-76.96", "v365.72", "h80.51" ] ],

[ "lcd1:VCC", "servo2:V+", "red", [ "h-16.04", "v-335.03", "h-257.46", "v77.67" ] ],

[ "sw1:2", "r8:1", "green", [ "v0" ] ],

[ "r7:1", "sw4:1", "green", [ "v0.58", "h25.07" ] ],

[ "r6:1", "sw3:2", "green", [ "v-0.73", "h22.64" ] ],

[ "r5:1", "sw2:2", "green", [ "v-21.07", "h41.68", "v-133.13", "h-43.76" ] ]

]

}

Certainly, I'll provide an explanation of the code you've provided. This code is written in the Arduino programming language and seems to control servos and digital outputs based on the state of various digital input pins. Let's break down the code step by step:

cppCopy code

if (datax != 0b0000) {  
 myservo1.write(180);  
 count1 = 0;  
}

* This block of code checks whether the variable **datax** is not equal to zero (binary **0000**). If it's not zero, it moves **myservo1** to the 180-degree position and sets **count1** to 0.

cppCopy code

if (digitalRead(outdoor1) == 0 || datax == 0b0000) {  
 enablecount = true;  
}

* Here, it checks if the digital input pin **outdoor1** is reading LOW (0) or if **datax** is equal to zero. If either condition is met, it sets **enablecount** to true.

cppCopy code

if (digitalRead(indoor2) == 0) {  
 myservo2.write(180);  
}

* This part checks if the digital input pin **indoor2** is reading LOW (0). If it is, it moves **myservo2** to the 180-degree position.

cppCopy code

if (digitalRead(outdoor2) == 0) {  
 myservo2.write(90);  
}

* Similar to the previous section, this checks if the digital input pin **outdoor2** is reading LOW (0). If it is, it moves **myservo2** to the 90-degree position.

cppCopy code

for (uint8\_t index = 0; index < 4; index += 1) {  
 digitalWrite(outputgreen[index], !digitalRead(sncar[index]));  
 digitalWrite(outputred[index], digitalRead(sncar[index]));  
}

* This loop iterates through an array of digital input pins called **sncar** (likely sensors). For each pin in the array, it reads its state, negates it, and then sets two corresponding output pins: **outputgreen[index]** to the negated value and **outputred[index]** to the original value.

cppCopy code

delay(10);

* Finally, there's a 10-millisecond delay. This is a short delay, often used to prevent rapid or unintended repeated execution of the code.

Please note that the effectiveness and correctness of this code depend on the complete context and the hardware connections to the Arduino pins (e.g., servos, sensors, and output pins). If you have specific questions or need further assistance with this code, please provide more details about the hardware and the specific functionality you are trying to achieve.