MYANMAR INSTITUTE OF INFORMATION TECHNOLOGY

I Semester (2023-2024)

Project Report

Microprocessors & Interfacing Lab

Arduino Powered Car Parking System

Project Title:	**Arduino Powered Car Parking System**
Presented By:	**Group II**
Instructor-in-Charge:	**Dr. Nu War**
Offered in:	**ECE 3011 (Microprocessors & Interfacing Lab)**
Date of Submission:	

Abstract

Parking in metropolitan areas is increasingly becoming a significant issue. Conventional parking systems are struggling to keep pace, leading to traffic congestion, and wasted time for drivers. This project introduces an Arduino-based Car Parking System, a creative approach to parking management. This system is designed to optimize the utilization of parking spaces and significantly enhance the convenience for users. Employing ultrasonic sensors, which serve as the cornerstone of our real-time vehicle detection mechanism. These sensors monitor the occupancy of parking spots and relay this information to an LCD screen. This dynamic update of parking spot availability provides drivers with immediate and accurate parking information, eliminating the need for manual search. Furthermore, our system incorporates servo motors to automate the operation of parking gates. This integration not only streamlines the entry and exit process but also elevates the overall user experience by reducing manual intervention. Our Arduino-based Car Parking System represents a significant stride towards smart and efficient parking management.

Table of Contents

List of Tables

List of Figures

CHAPTER 1
INTRODUCTION

As urban parking challenges escalate, conventional systems struggle, causing traffic congestion and wasted time. Arduino technology offers a transformative solution, revolutionizing parking space utilization. This system, driven by Arduino microcontrollers and sensors like ultrasonic or infrared, streamlines the parking experience. No more aimless searching; drivers are directed to available spots through real-time monitoring. The Arduino microcontroller efficiently processes sensor data using intelligent algorithms, optimizing space allocation based on proximity, availability, and convenience. This automated approach not only simplifies parking but also enhances overall space efficiency. Additionally, the Arduino-based system contributes to environmental conservation by reducing fuel consumption and greenhouse gas emissions. As cities expand and parking problems intensify, such intelligent, eco-friendly solutions are poised to play a crucial role in shaping urban landscapes.

1.1 Objectives

Our project aims to achieve following objectives:

Optimize Parking Space Utilization: Develop an Arduino-based car parking system that maximizes the efficient use of available parking spaces, addressing the growing challenges of congestion in metropolitan areas.

Real-time Parking Spot Monitoring: Implement sensors, such as ultrasonic or infrared, to enable real-time monitoring of parking spot occupancy. This feature ensures accurate and up-to-date information on the availability of parking spaces.

User-Friendly Parking Experience: Design an intuitive and user-friendly interface, incorporating an LCD screen to display real-time parking spot status. Direct drivers to available parking spaces, eliminating the need for aimless searching and enhancing the overall user experience.

Automated Gate Control: Integrate servo motors to automate parking gate operations, streamlining entry and exit processes for enhanced user convenience and efficient traffic flow within parking facilities.

Environmental Impact Reduction: Contribute to environmental conservation by reducing fuel consumption and greenhouse gas emissions through the implementation of a system that facilitates faster and more efficient parking.

Scalability and Cost-Effectiveness: Design the system with scalability in mind, allowing for easy adaptation to various parking scenarios. Utilize cost-effective components and Arduino microcontrollers to ensure accessibility and affordability for wider adoption in diverse urban settings.

Promote Sustainable Urban Development: Contribute to the creation of intelligent, eco-friendly cities by offering a technological solution to urban parking challenges. This project aims to be a step towards shaping more sustainable and efficient urban environments.

1.2 Requirements

Here are our essential prerequisites crucial for the seamless realization of the Arduino powered Car Parking System.

1.2.1 Hardware Requirements

Our hardware components are carefully selected to guarantee accuracy, reliability, and durability in the Arduino powered Car Parking System. This section described the information on the hardware and software requirements vital for the effective functioning of our parking system using Arduino.

Ultrasonic Sensors: Six units positioned to facilitate precise occupancy detection within parking spaces. Two additional units placed at the entrance and exit gates, serving as key components for seamless vehicle detection and management.
Servo Motors: Two units installed for the precise control of entrance and exit barriers, ensuring smooth and efficient operation.
Arduino Board: The central processing unit orchestrating the entire system, responsible for processing data, making decisions, and controlling various components.
LCD Display: A dynamic interface providing real-time information on parking space availability, enhancing the overall user experience.
Wiring and Connectors: Establishing the necessary connections for seamless operation.
PVC foam plates: These plates serve as a fundamental structural element, contributing to the construction of a two-tier parking lot.

1.2.2 Software Requirements

Arduino IDE: Our chosen Integrated Development Environment for programming the Arduino board.
Tinkercad Simulator: To evaluate and simulate the system components essential to the overall Car Parking System.
Proteus 8 Professional: To evaluate and simulate system components and the system as a whole to deliver a successful project.

1.3 Project Members, Budget, and Plan

Our project requires a collaborative effort from a dedicated team, careful financial planning, and a well-structured project timeline. In this section, we introduce the key individuals driving the project forward, outline the budgetary considerations for sourcing the components and a well-planned project timeline. The simplicity of our organizational structure and financial planning aims to facilitate a clear understanding of our team's roles and the strategic roadmap for project development.

1.3.1 Project Members

No.	Name	Roll No	Task
1.	Min Htet Mon	2016-MIIT-ECE-017	System Flowchart
2.	Saung Wut Yee	2018-MIIT-CSE-043	Coding, Hardware Software Integration
3.	Swan Yee Htet	2018-MIIT-CSE-048	Hardware Implementation, PowerPoint
4.	William Aye Tun Oo	2018-MIIT-CSE-055	Hardware Implementation
5.	Kaung Khant Hein	2019-MIIT-CSE-009	Project Report
6.	Khant Zaw Phyo	2019-MIIT-CSE-018	Project Report, PowerPoint
7.	Zwe Shein Lin	2019-MIIT-CSE-060	Hardware Implementation
8.	Khaing Thinzar Tun	2019-MIIT-ECE-014	Hardware Implementation, Report
9.	Kyaw Bon Thar	2019-MIIT-ECE-017	Coding, Hardware Software Integration
10.	Min Ko Khant	2019-MIIT-ECE-028	Project Report
11.	Zay Yar Paing Min	2019-MIIT-ECE-051	Hardware Implementation
12.	Zwe Wai Yan Aung	2019-MIIT-ECE-055	Hardware Implementation

Table I: Project Members

1.3.2 Budget Table

No.	Description	Price	Quantity	Amount
1.	InfraRed Sensor (Not utilized)	6200	10	62,000
2.	Servo Motor	8500	2	17,000
3.	Ultrasonic Sensor	6000	8	48,000
4.	Jumper Wires	9000	1	9000
5.	PVC 5mm 4’ x 2’	8504	2	17,008
6.	20x4 LCD	20,000	1	20,000
7.	Male Header	500	1	500
8.	Extra Components	-	-	10,000
			Total:	**183,508**

Table II: Budget Table

1.3.3 Project Timeline

Task	Start Date	End Date	Week 1	Week 2	Week 3	Week 4
Research/Idea Pitching	1-03-24	2-03-24				
Project Outline/ Proposal	3-03-24	9-03-24				
Project Planning	3-03-24	9-03-24				
Sourcing Components	9-03-24	12-03-24				
Software Implementation	3-03-24	12-03-24				
Hardware Implementation	11-03-24	15-03-24				
Project Report	10-03-24	15-03-24				
Presentation	11-03-24	16-03-24				
Testing	12-03-24	16-03-24				
D Day	18-03-24	18-03-24				

Table III: Project Timeline

CHAPTER 2
THEORY BACKGROUND

2.1 Arduino IDE Software

The Arduino Integrated Development Environment (IDE) is an open-source software platform used for programming Arduino boards. It provides a user-friendly interface for writing, compiling, and uploading code to Arduino microcontrollers. The Arduino IDE is available for Windows, macOS, and Linux operating systems, and it can be downloaded for free from the Arduino website. Additionally, the Arduino IDE is open source, allowing users to contribute to its development and customize it to suit their needs.

2.1.1 Features of Arduino-Programming Language

The Arduino programming language has several key features :

Simple Syntax: The Arduino language has a straightforward syntax that is easy to understand and learn, making it accessible to beginners. It uses familiar programming constructs such as variables, loops, and functions.

Built-in Functions: Arduino provides a set of built-in functions for common tasks such as controlling digital and analog pins, reading from sensors, and communicating with other devices. These functions abstract away many low-level details, making it easier to work with hardware.

Library Support: Arduino libraries contain pre-written code modules that extend the functionality of Arduino projects. These libraries provide functions for interfacing with various sensors, actuators, communication protocols, and other peripherals. Users can easily include and use these libraries in their sketches.

Support for Microcontroller Features: The Arduino language provides access to the features and capabilities of the underlying microcontroller, such as digital and analog input/output (I/O), interrupts, timers, and serial communication.

Event-Driven Programming: Arduino supports event-driven programming through interrupts and timers. This allows users to respond to external events (e.g., sensor readings, button presses) asynchronously without blocking the main program loop.

Serial Communication: Arduino supports serial communication, allowing it to communicate with other devices such as computers, sensors, and displays via UART, SPI, or I2C interfaces. This enables data logging, debugging, and interaction with external systems.

Low-level Access: While Arduino abstracts away many hardware details, it still provides low-level access to the microcontroller's registers and features. Advanced users can directly manipulate registers and use inline assembly code for optimized performance or access to specialized features.

Platform Independence: Arduino sketches are written in C/C++ and can be compiled and run on different platforms (e.g., Windows, macOS, Linux) without modification, if the Arduino IDE or compatible tools are available.

2.1.2 Advantages of Arduino Programming Language

Advantages of Arduino Programming Language are :

Ease of Learning: The Arduino language has a simple syntax and straightforward structure, making it easier for beginners to grasp programming concepts without being overwhelmed by complex syntax or language features.

Accessibility: Arduino is designed to be accessible to a wide range of users, including artists, designers, students, and hobbyists who may not have a background in programming or electronics. The language's simplicity lowers the barrier to entry for those interested in creating interactive projects.

Rapid Prototyping: Arduino facilitates rapid prototyping by providing a quick and straightforward way to develop and test ideas. The combination of a simple language, integrated development environment (IDE), and extensive libraries enables users to iterate on their designs quickly and efficiently.

Abstraction of Hardware Complexity: Arduino abstracts away much of the complexity of microcontroller programming and hardware interfacing. Built-in functions and libraries provide high-level abstractions for common tasks, such as reading from sensors, controlling actuators, and communicating with other devices, allowing users to focus on their projects rather than low- level details.

Community Support: Arduino has a large and active community of users and developers who contribute tutorials, documentation, libraries, and support forums. This community-driven ecosystem provides valuable resources and assistance to beginners and experienced users alike.

Cross-Platform Compatibility: Arduino sketches written in the Arduino language can be compiled and run on different platforms, including Windows, macOS, and Linux, using the Arduino IDE or compatible tools. This platform independence makes it easier for users to develop and share their projects across different operating systems.

Integration with Hardware: Arduino is tightly integrated with Arduino boards, which are widely available and inexpensive. The Arduino IDE provides built-in support for uploading sketches to Arduino boards via USB or other communication interfaces, simplifying the development and deployment process.

Flexibility and Customization: Although Arduino abstracts away many hardware details, it still provides access to low-level features of the microcontroller, allowing users to customize and optimize their code as needed. Advanced users can directly manipulate registers, use inline assembly code, or integrate external libraries to extend the capabilities of Arduino projects.

2.1.3 Language Used in Arduino IDE

The Arduino IDE primarily uses a simplified version of C and C++ programming languages. This simplified version is tailored to make it easier for beginners and hobbyists to start programming microcontrollers, such as those used in Arduino boards.

C programming Language

The C programming language is a general-purpose programming language that was developed in the early 1970s by Dennis Ritchie at Bell Labs. It was designed to provide low-level access to memory and efficient execution, making it well-suited for system programming, embedded systems, and performance-critical applications. [1]

C++ programming Language

C++ is a general-purpose programming language that was developed as an extension of the C programming language. It was created by Bjarne Stroustrup at Bell Labs in the early 1980s. C++ retains many features of C while also introducing several new features, particularly support for object-oriented programming (OOP).[2]

[1] Prinz, Peter; Crawford, Tony (December 16, 2005). C in a Nutshell. O'Reilly Media, Inc.
[2] Stroustrup, Bjarne (2013). The C++ Programming Language (Fourth ed.). Addison-Wesley.

2.2 System Model and Components Analysis

1. **Ultrasonic sensor**: An ultrasonic sensor is a type of sensor that emits ultrasonic sound waves and then measures the time it takes for the waves to bounce back after hitting an object. These sensors are commonly used for distance measurement, object detection, and navigation in robotics, automation, and various other applications.

2. **Arduino Uno Boards**: The Arduino Uno is a popular microcontroller board based on the ATmega328P microcontroller chip. It is widely used in hobbyist projects, educational activities, and prototyping due to its simplicity, ease of use, and extensive community support. The Arduino Uno is powered by the ATmega328P microcontroller, which has 32KB of flash memory for storing programs, 2KB of SRAM for data storage, and 1KB of EEPROM for non-volatile storage.[3]

3. **LCD(Liquid Crystal Display)**: LCD stands for Liquid Crystal Display. It is a type of flat panel display commonly used in electronic devices such as computer monitors, television screens, instrument panels, and handheld devices like smartphones and tablets. LCDs consist of a layer of liquid crystals sandwiched between two transparent electrodes and two polarizing filters. Liquid crystals are organic molecules that can change orientation when an electric current is applied, altering the passage of light through the display.

4. **Servo motor**: A servo motor, often simply referred to as a servo, is a type of motor commonly used in robotics, remote-controlled vehicles, automation systems, and various other applications where precise control of angular position, speed, and acceleration is required.

[3] Smith, A. G. (2011). Introduction to Arduino. Publisher.

CHAPTER 3
IMPLEMENTATION

3.1 System Flowchart

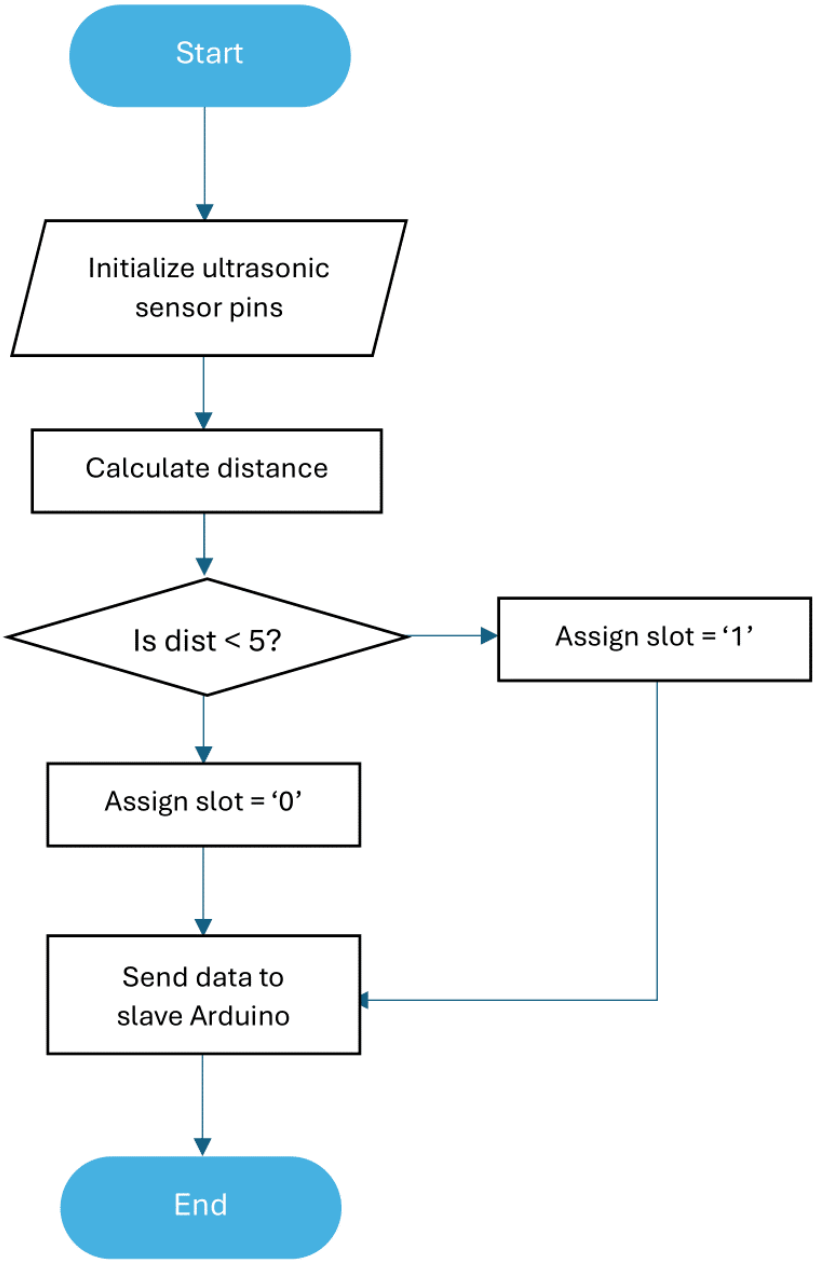
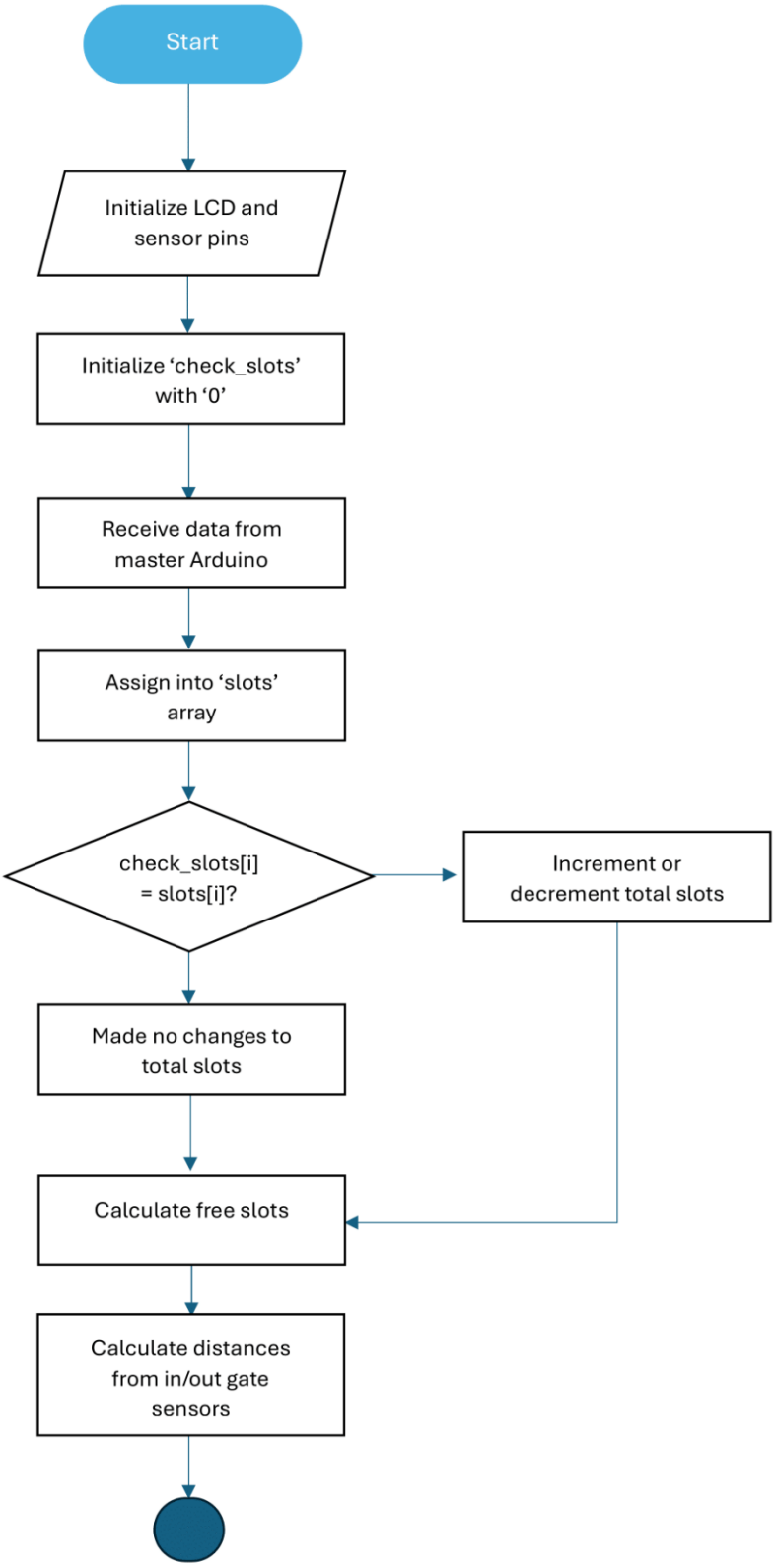


Figure 3.1: Master Flowchart


Start
Initialize LCD and sensor pins
Initialize 'check_slots' with '0'
Receive data from master Arduino
Assign into 'slots' array
check_slots[i] = slots[i]?
Increment or decrement total slots
Made no changes to total slots
Calculate free slots
Calculate distances from in/out gate sensors

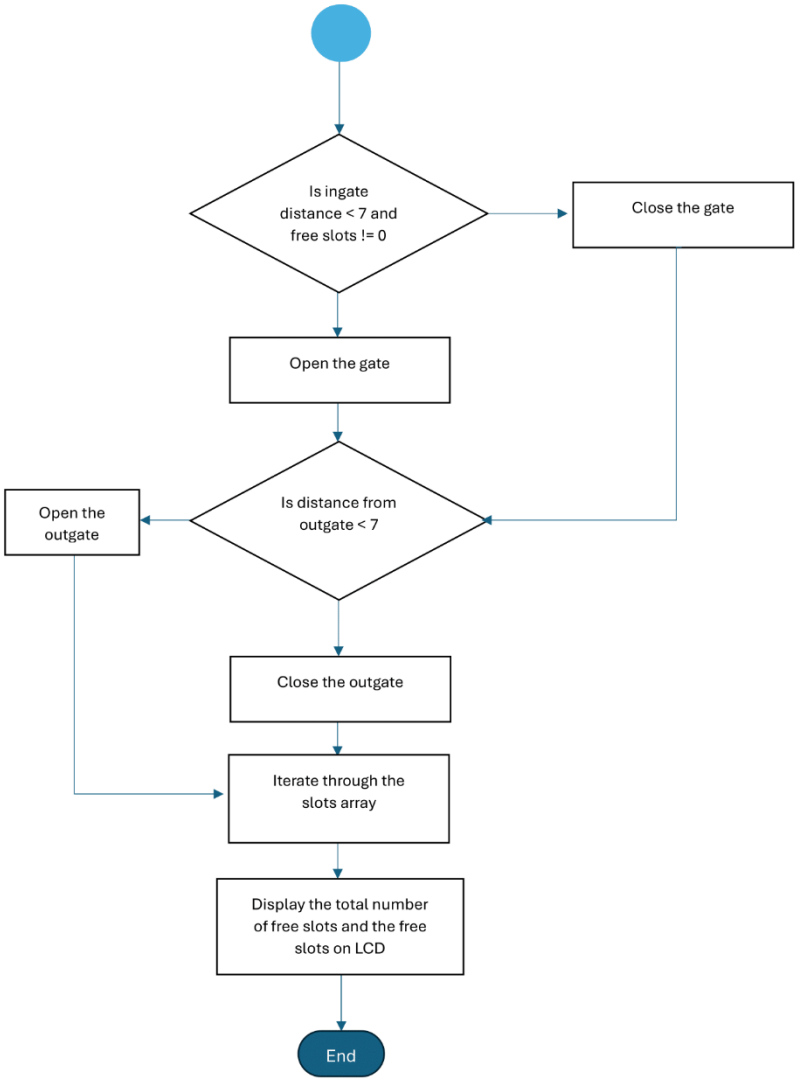


Figure 3.2: Slave Flowchart

3.2 Implementation

3.2.1 Software Implementation

Main program:

```
//mastercode

int trigpins[6] = { 2, 4, 6, 8, 10, 12 };

int echopins[6] = { 3, 5, 7, 9, 11, 13 };

void setup() {
  Serial.begin(9600);
  //trigpins
  pinMode(2, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(12, OUTPUT);

  //echopins
  pinMode(3, INPUT);
  pinMode(5, INPUT);
  pinMode(7, INPUT);
  pinMode(9, INPUT);
  pinMode(11, INPUT);
  pinMode(13, INPUT);
}

void loop() {
  char slot;
  int trig, echo;
  for (int i = 0; i < 6; i++) {
    trig = trigpins[i];
```

```
    echo = echopins[i];
    slot = ultrasensor_slot_assigning(trig, echo);
    Serial.write(slot);
  }
  delay(100);
}

char ultrasensor_slot_assigning(int trig, int echo) {
  char slot;
  long duration, distance;

  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  duration = pulseIn(echo, HIGH);
  distance = duration / 29 / 2;

  if (distance < 5) {
    slot = '1';
  } else {
    slot = '0';
  }
  return slot;
}
```

Program for slave Arduino:…

```
//slave
#include <LiquidCrystal.h>
#include <Servo.h>

#define trigIn 2
#define echoIn 4
#define trigOut 5
#define echoOut 7
LiquidCrystal lcd(13, 12, 11, 10, 9, 8);

Servo inGate;
Servo outGate;

char check_slots[6] = { '0', '0', '0', '0', '0', '0' };
char slots[6];
int free_slot = 6;  //free slot for parking
void setup() {

  Serial.begin(9600);
  inGate.attach(3);
  outGate.attach(6);

  pinMode(echoIn, INPUT);
  pinMode(echoOut, INPUT);
  pinMode(trigIn, OUTPUT);
  pinMode(trigOut, OUTPUT);

  lcd.begin(20, 4);
```

```
  inGate.write(180);
  outGate.write(0);
}

void loop() {
  long distanceIn, distanceOut = 0;
  int inflag = 1;
  if (Serial.available() >= 6) {
    for (int i = 0; i < 6; i++) {
      slots[i] = Serial.read();
    }

    for (int i = 0; i < 6; i++) {
      Serial.print("slot");
      Serial.print(i + 1);
      Serial.print(":");
      Serial.println(slots[i]);
    }
    int total = 0;
    for (int j = 0; j < 6; j++) {
      if (slots[j] == check_slots[j]) {
        total += 0;
      } else {
        if (slots[j] == '1') {
          check_slots[j] = slots[j];
          total += 1;
        }
        if (slots[j] == '0') {
          check_slots[j] = slots[j];
          total -= 1;
```

```
    }
  }
}
Serial.print("Total = ");
Serial.println(total);
free_slot -= total;

//lcd display
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Free slots :");
lcd.print(free_slot);

// Display Parking Full message if no free slots available
if (free_slot == 0) {
  lcd.setCursor(0, 1);
  lcd.print("Parking FULL");
} else {
  lcd.setCursor(0, 1);
  lcd.print("              ");  // Clear the line if not full.
}

// Display available slots numbers for upstairs
lcd.setCursor(0, 2);
lcd.print("1st: ");
for (int i = 0; i < 3; i++) {
  if (slots[i] == '0') {
    lcd.print(i + 1);
    lcd.print(" ");
  }
```

```
}
// Display available slots numbers for downstairs
lcd.setCursor(0, 3);
lcd.print("2nd: ");
for (int i = 3; i < 6; i++) {
  if (slots[i] == '0') {
    lcd.print(i + 1);
    lcd.print(" ");
  }
}


Serial.print("Free_slot = ");
Serial.println(free_slot);

if (free_slot == 0) {
  inflag = 0;
}

//for ingate and outgate with ultrasonic sensors
long durationIn, durationOut;

digitalWrite(trigIn, LOW);
delayMicroseconds(2);
digitalWrite(trigIn, HIGH);
delayMicroseconds(10);
digitalWrite(trigIn, LOW);

durationIn = pulseIn(echoIn, HIGH);
distanceIn = durationIn / 29 / 2;
```

```
    if (distanceIn < 7 && inflag != 0) {
      inGate.write(90);
      Serial.println("InGate opened.");
    } else {
      inGate.write(180);
      Serial.println("InGate not opened.");
    }

    digitalWrite(trigOut, LOW);
    delayMicroseconds(2);
    digitalWrite(trigOut, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigOut, LOW);

    durationOut = pulseIn(echoOut, HIGH);
    distanceOut = durationOut / 29 / 2;

    if (distanceOut < 7) {
      outGate.write(90);
      Serial.println("OutGate Opened.");
    } else {
      outGate.write(0);
      Serial.println("OutGate not opened.");
    }
  }
  delay(1000);
}
```

3.2.2 Hardware Implementation

The project aims to construct a two-tiered car parking system comprising six parking slots, evenly distributed across the ground floor and the first floor. The system is designed with one entrance and one exit, each equipped with an ultrasonic sensor to detect the presence of a vehicle. Upon detection of a vehicle, a lever connected to a servo motor is activated, causing it to rotate 90 degrees and allow the vehicle to enter or exit the parking system. To keep the users informed about the parking situation, an LCD display is installed at the entrance. This display provides real-time updates on the availability of parking slots. When all slots are occupied, the display notifies incoming drivers, thus enhancing the user experience.

Once the users have completed their activities, they can conveniently exit the parking lot through a designated exit lane. Similar to the entrance, the exit gate is equipped with a servo-connected lever and an ultrasonic sensor. The sensor detects the departing vehicle and activates the lever to rotate 90 degrees, allowing the vehicle to leave. The parking system is designed to accommodate vehicles on both floors, giving users the flexibility to park wherever they prefer, provided slots are available. The key component of the system is the use of infrared sensors installed at each parking slot. These sensors detect whether a slot is occupied or not, enabling efficient management of the parking space.

Due to the extensive use of peripherals like the LCD display, the project employs two Arduino boards to overcome the limitation of insufficient pins. A “Master-Slave” configuration is used to control the system, ensuring smooth and efficient operation.

The system is powered by an AC power supply, supplemented by a 9V battery, ensuring reliable and continuous operation. This combination of methods and resources allows for the successful implementation of the Arduino-based car parking system.

3.3 System Architecture

Our Arduino based Car Parking System is a practical solution designed for efficient management and monitoring of parking slots. It uses Arduino microcontrollers, ultrasonic sensors for detecting occupied slots and detecting incoming/outgoing cars at the entrance and exit gates, servo motors for gate control, and an LCD for displaying parking availability. Here is an overview of our Arduino-based car parking system.

3.3.1 Car Parking System Overview

The system consists of two main components:

1. **Hardware**: Includes six ultrasonic sensors for car parking slot detection, two servo motors for gate barriers, two ultrasonic sensors at the gates, and two Arduino boards acting as master and slave. The six ultrasonic sensors are used to detect whether a parking spot is occupied or not. The two ultrasonic sensors are used to detect the presence of a car at the entrance and exit gates. The servo motors control the opening and closing of the gates. The Arduino boards process the sensor data and control the servo motors and the LCD.

2. **Software**: The software running on the Arduino boards takes the data from the sensors, processes it, and controls the servo motors and the LCD accordingly. When a car approaches the entrance gate, the ultrasonic sensor detects it and sends a signal to the Arduino board. The board then commands the servo motor to open the gate. Once the car is parked in a slot, the corresponding ultrasonic sensor detects it and sends a signal to the Arduino board. The board then updates the parking slot availability information on the LCD.

3.3.2 Components

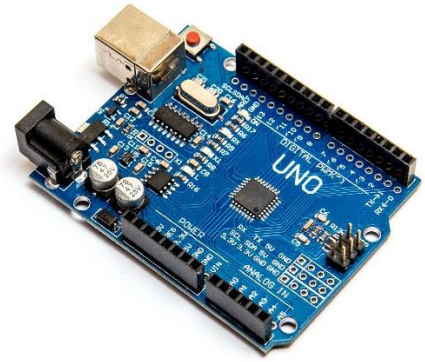


Figure 3.3.1: Arduino Uno

Figure 3.3.2: Servo Motor

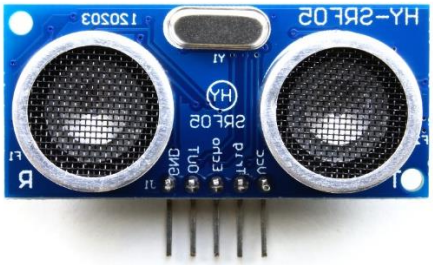


Figure 3.3.3: Ultrasonic Sensor

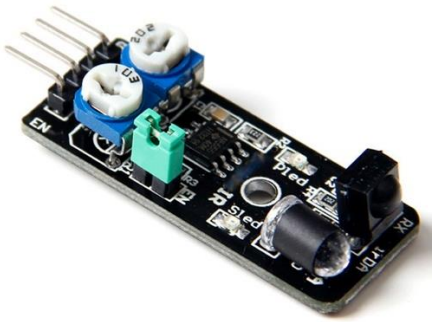


Figure 3.3.4: InfraRed Sensor

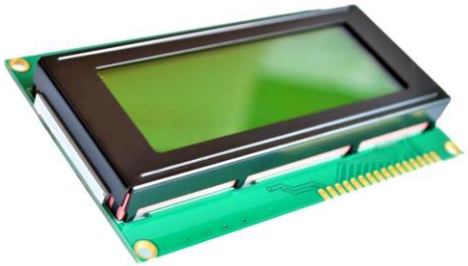


Figure 3.3.5: 20x4 LCD

CONCLUSION

4.1 Summary

Our Arduino-based car parking system project has successfully demonstrated the potential of microcontroller technology in automating and enhancing the efficiency of car parking systems. This project has effectively harnessed the capabilities of Arduino microcontrollers to develop a user-friendly and cost-effective solution that can be seamlessly integrated into any parking lot. The system's primary function is to manage parking spaces by providing real-time updates on slot availability, thereby significantly improving the user experience and the overall parking management process. The implementation of this system underscores the practical application of microcontroller technology in addressing everyday challenges. Furthermore, the project has achieved our objectives by designing a system that employs a combination of sensors and indicators to guide drivers to available parking spaces. This approach not only optimizes the use of parking space but also reduces the time taken by drivers to find a vacant spot, thereby enhancing the overall efficiency of the parking process. Despite its simplicity, the system has proven to be an effective tool in managing car parking, demonstrating the immense potential of Arduino microcontrollers in creating practical, affordable, and efficient solutions for real-world problems.

4.2 Advantages

The Arduino-based car parking system offers several advantages:

- **Efficiency**: The system optimizes the use of parking space and reduces the time taken by drivers to find a vacant spot.
- **Real-time updates**: The system provides real-time information about the availability of parking slots, which helps with better parking management.
- **User-friendly**: The system is easy to use, making it convenient for all drivers, irrespective of their technical expertise.
- **Cost-effective**: The use of Arduino makes the system affordable and easy to maintain compared to other proprietary solutions.
- **Reduced Traffic Congestion**: By efficiently guiding drivers to available parking spaces, the system can significantly reduce the time spent by drivers circling the parking lot looking for a space. This can lead to a reduction in traffic congestion within the parking lot.
- **Data Collection and Analysis**: The system can collect valuable data about parking usage patterns, which can be analyzed to make informed decisions about parking pricing, future expansions, and other operational aspects.

4.3 Limitations

While the Arduino-based car parking system has proven to be effective in managing parking spaces and enhancing the user experience, there are several limitations that should be acknowledged:

- **Sensor Limitations**: The system's performance is heavily dependent on the reliability and accuracy of the ultrasonic and infrared sensors. Any malfunction or inaccuracy in these sensors could impact the system's ability to correctly detect the presence of vehicles and the occupancy of parking slots.
- **Limited Scalability**: The current system is designed to manage a two-storied parking lot with six parking slots. While this is sufficient for small-scale parking lots, the system may face challenges when scaling up to larger parking lots with more parking slots.
- **Weather Conditions**: The performance of the ultrasonic and infrared sensors could be affected by extreme weather conditions, such as heavy rain, snow, or fog, which could lead to inaccurate readings.
- **Power Outages**: In the event of a power outage, the system may cease to function unless there is a backup power supply in place.
- **Sensor Placement**: The placement of sensors is crucial for the accurate detection of vehicles. Incorrect placement could lead to false readings.
- **User Error**: Users unfamiliar with the system may not follow the correct procedures, which could lead to system errors. For example, a user might not park correctly within a slot, causing the sensor to not detect the vehicle.

These limitations provide valuable insights for future improvements and enhancements to the system.

4.4 Further Enhancements

While the current implementation of the Arduino-based Car Parking System has proven to be effective, there are several potential enhancements that could further improve its functionality and user experience:

- **Integration with a Mobile Application:** Developing a mobile application that interfaces with the parking system could provide users with the ability to check the availability of parking spaces remotely, reserve a parking lot, and even pay parking fees. This would add a significant level of convenience for the users.
- **Advanced Sensor Technology:** While the current system uses ultrasonic sensors for vehicle detection at the entrance and exit, and infrared sensors for parking slot occupancy detection, the incorporation of more advanced sensor technologies could improve the system's accuracy and reliability. For instance, LiDAR sensors could provide more precise distance measurements, and image recognition technology could be used for vehicle identification.
- **Enhanced Scalability**: The current system supports a two-storied parking lot with six parking slots. Future enhancements could focus on increasing the system's scalability to support larger parking lots with more parking slots. This could involve optimizing the sensor network and improving the system architecture.
- **Energy Efficiency:** The system could explore options for making it more energy efficient. This could include the use of solar-powered sensors or low-power consuming communication technologies.

These enhancements could significantly improve the system's performance, usability, and scalability, making it an even more effective solution for automated car parking management.

REFERENCES

1. *Margolis, M. (2011). Arduino Cookbook (2nd ed.). Publisher.*
2. *Smith, A. G. (2011). Introduction to Arduino. Publisher.*
3. *Arduino. Arduino Documentation. Available at: https://docs.arduino.cc/*
4. *Arduino. Servo Motor Documentation. Available at: https://www.arduino.cc/reference/en/libraries/servo/*
5. *Arduino. Liquid Crystal Displays (LCD) with Arduino. Available at: https://docs.arduino.cc/learn/electronics/lcd-displays/*
6. *Dejan. Ultrasonic Sensor HC-SR04 and Arduino – Complete Guide. Available at https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/*
7. *Ultrasonic Sensor with Arduino uno with code / 3 pin ultrasonic sensor for object distance measuring. Available at: https://www.electronicscuriosities.com/2020/10/ultrasonic-sensor-with-arduino-uno-with.html*