Department of Computer Science
Technical University of Cluj-Napoca

DOCUMENTATION
ARDUINO PARKING SENSOR WITH BLUETOOTH FUNCTIONALITY

DMP LABORATORY

CURAC MIHAI – IONUȚ
30431

CONTENTS

1. INTRODUCTION

The purpose of this project is to design an Arduino parking system that is easy to use and has multiple functionalities. It can take commands via Bluetooth like checking whether a parking spot is taken by someone else or not, and it can send the result and the distance to the nearest obstacle in cm to the Bluetooth-connected device. If the parking spot is available, it serves as a car parking sensor, with a precise distance sensor and visual and sound signals that alert the user of close obstacles.

I chose this project because I think it has many real world applications and uses components that we have become familiar with at the laboratories. Using these components to design and build a more complex system was a compelling challenge for me. The idea behind the project is actually very well-known, but I wanted to try a more visually intuitive implementation that shows the parking progress through a strip of RGB LEDs.

2. BIBLIOGRAPHIC RESEARCH

As previously mentioned, there are many parking system projects available on the Internet, with various levels of complexities, implementation costs, power consumptions, and adaptations to user needs. I will compare some of the most popular ones to mine and explain their differences.

One of the easiest implementations uses an Arduino Uno, a Buzzer, and an Ultrasonic Sensor. It consists of a simple distance sensor that triggers a buzzer beep when the distance to the nearest obstacle is less than a specified safe value. This is only done via the tone function and some delays, while my implementation gradually increases the buzzer intensity in 8 steps, corresponding to 8 descending distances in cm to the nearest obstacle. Other similar projects use a single intermittent buzzer beep when the distance is, for example, less than 10 cm. However, it is still a rudimentary solution that only uses some delays to turn the buzzer on and off. My solution uses milis() for timing to turn the buzzer on and off at specific frequent intervals, which is a much more convenient function that I will explain in detail in the next chapter.

Another online implementation uses a similar intermittent buzzer sound with delays, but with three LEDs to add a visual signal: a green, orange, and red one. This is closer to my solution, but still not as accessible as an 8 LED strip that I can control individually however I want to, according to the 8 descending distances to the nearest obstacle that I presented above.

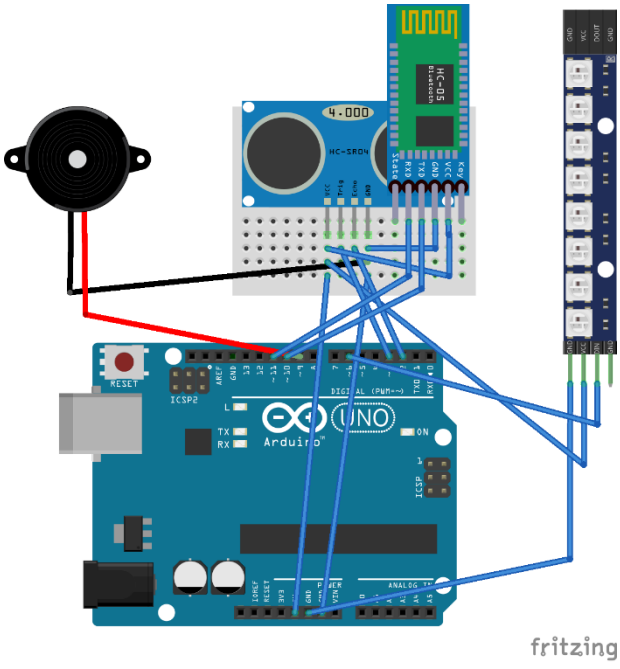
These solutions may have a lower power consumption and may be easier to implement with fewer code and costs, but my project is more adapted to usual users' needs. These needs include being able to clearly see and hear the progress of the parking once it has started. In addition to this 8-step process, my implementation also includes a Bluetooth module that the users can connect to via any BT-equipped device. Once connected to the system, the user can input commands like checking if the parking spot is taken, and seeing the distance to the nearest obstacle in cm at any time, all on their mobile phones or other devices.

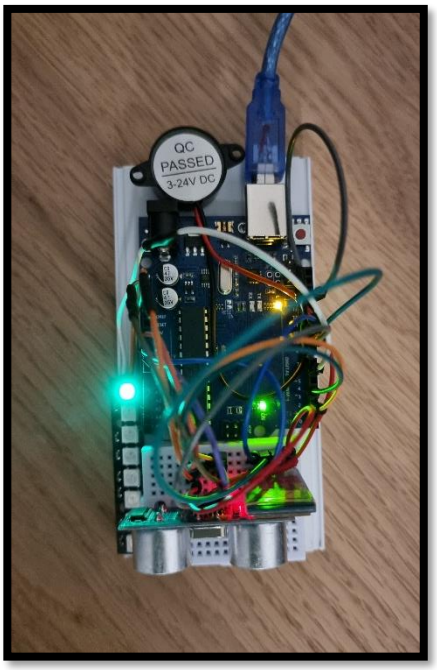
3. PROPOSED SOLUTION AND IMPLEMENTATION

My chosen parking system solution uses seven components: an Arduino Uno, a piezoelectric buzzer, a ZS-040 Bluetooth module, a HC-SR04 Ultrasonic sensor, an 8-bit 2812 RGB LED strip, a breadboard, and some wires with male connectors on their ends.

The system firstly sends a “Welcome! Enter your command:” message to the device that it is connected to via Bluetooth. It also turns on a green LED from the strip, signifying that the parking has not started yet and that the sensor is at a safe distance from any obstacles like cars. The whole structure can be mounted on top of a car to serve as a real-time parking sensor, but it was mostly meant to be placed in a fixed position, like at the back of a parking spot, always checking for cars that might park there and assisting them in the process. If a user wants to initiate parking in a place that uses this system, they can send the “Park” command via Bluetooth and the sensor will check if the distance to the nearest obstacle is more than 12 cm. If so, it will send back the “Parking spot available. Distance to nearest obstacle: x cm” message. If not, it will send back “Can't park here. Obstacle too close at: y cm”. When the systems detects a car at less than 12 cm, it turns the buzzer on and off at specific intervals, while also turning on subsequent LEDs from the strip (there are three green ones, three orange ones, and two red ones). The speed of the intermittent sound will be higher and higher as the car gets closer to the ultrasonic sensor, and when it is placed at less than 3 cm from it, all the LEDs will be turned on (including the two red ones) and a continuous loud sound will come from the buzzer, alerting the user that coming any closer to the parking spot border will result in a crash. At any time during parking, the user can send the “Distance” command via Bluetooth, and the system will send back the value to the nearest obstacle in cm.

Below is the circuit representation of the hardware parts I used and their connections, as well as a picture of the final circuit from above.





To get the distance values from the Ultrasonic sensor, I set the trigpin as an output and the echopin as an input, then, in the loop, I read the value from the echopin, which returned the sound wave travel time in microseconds. The distance was calculated with the formula *duration * 0.034 / 2*, which returned the speed of the sound wave divided by 2 (to the obstacle and back). I checked this value periodically by comparing it to 8 possible distances in cm, and used different visual and audio signals for each case. The LED RGB strip operations were done with the help of the official Adafruit NeoPixel documentation, which introduced me to functions like strip.setPixelColor, which takes as arguments the pixel number along the strip and the RGB value of the desired pixel colour, then sets that colour to the corresponding LED. I also used strip.fill, a very useful function that takes as arguments an RGB colour value, the index of the first pixel to fill, and the numbers of pixels to fill, and then sets multiple pixels to that same colour. I declared the three colours I worked with as single 32-bit types with the three RGB values (for green, orange, and red), in order to avoid passing as arguments the separate red, green and blue numbers every time. Finally, to "push" the colour data to the strip, I called strip.show().

I used a separate function to assist me with the buzzer sounds, which takes an integer value representing the speed of the intermittent beeps (amount of time between them) varying from 2000 (distance >= 9, two green LEDs on) to 100 (distance < 3, all LEDs on, alert). Here, I used milis() for the timing of the app. I had already recorded the start of the timing period in the setup function, so here I turned the buzzer off, got the current value of the milis() (actually the number of milliseconds since the program started), and then tested whether the period of time has elapsed (the period being the argument of the function). If so, I updated the start time of the timing period and turned the buzzer on with analogWrite, so that I could also control the loudness of the sound (using PWM).

Finally, the Bluetooth operations were done in the serialEvent() function, which is called in the loop function. SerialEvent checks whether the Bluetooth module is available, then takes the command from the user as a String. It checks if the command is "Park" or "Distance" and prints line by line the requested information, while continuously waiting for new commands.

4. TESTING AND VALIDATION

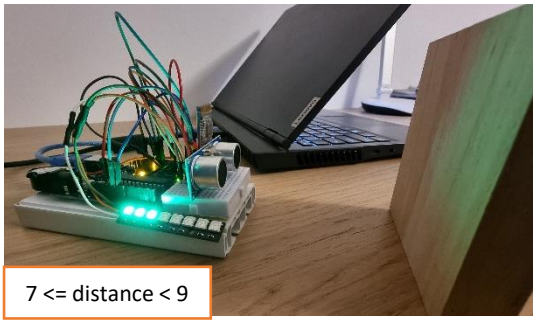
I encountered many difficulties while implementing this solution, so the project went though many phases to reach its goal. I worked in a modular fashion in order to make the debugging process easier. The first functional state of the system (excluding the Bluetooth module operations) was different to the final state of the project, since I firstly turned on the LEDs from the strip using many *for* instructions instead of the official functions presented before. That made the code quite hard to read and introduced other timing problems. The buzzer was also turned on using digitalWrite(buzzer, HIGH), then using tone(buzzer,261), both of which made it output a very loud and piercing sound that I didn't like. When I first started using analogWrite, I thought of only changing the volume of the alert sound as the car was getting closer and closer to the ultrasonic sensor. This did not sound great either, so I was finally satisfied by an intermittent beep that got faster and faster as the car was approaching the sensor.

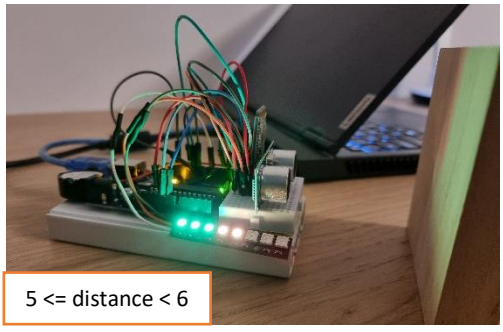
Once I had the distance sensor, buzzer and LED strip working, I added the Bluetooth module to the board and instantly encountered more problems. This time, the module did not get any

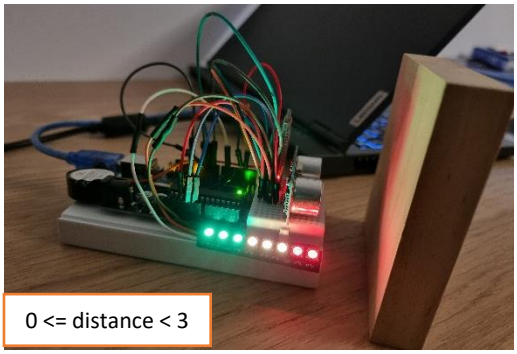
commands from my mobile phone, and it could not send information back either. I explained the solutions to these problems in the table below:

Component	Problem	Solution
Bluetooth Module	No data transfer between mobile device and module	Establishing serial communication on two digital pins of Arduino Uno, other than TX and RX, using the SoftwareSerial Library
Ultrasonic Sensor	Ultrasonic sensor randomly sending “0 cm” values to the mobile phone even with obstacles far away from it, after subsequent commands from phone	Increasing the delay values in the loop function
Buzzer	Buzzer turned on continuously (alert mode) after sending too many commands from mobile phone	Same as above

Below are some images from the testing phase. In the first photo, the obstacle is at a distance between 7 and 9 cm from the sensor, thus at the third level (three green LEDs are on, buzzer time is 1000). In the second photo, the obstacle is at a distance between 5 and 6 cm from it, thus at the fifth level (three green LEDs and two orange LEDs are on, buzzer time is 500). In the third photo, the obstacle is at a distance between 0 and 3 cm from the system, thus at the eighth and final level (three green LEDs, three orange LEDs, and two red LEDs are on, buzzer speed is 100 and in alert mode).







5. CONCLUSION

This project was a great opportunity to acquire more hands-on experience with Arduino devices and components. I learnt many things about coding for these types of boards and about useful libraries that exist for specific components, like the Adafruit RGB LED strip. Reaching a valid solution meant going through numerous debugging sessions and changing pieces of code to make the parking system more intuitive and helpful for users.

I think this solution has many real-world applications and can be used reliably to measure distances to obstacles like cars, while also providing the user with sound, visual, and Bluetooth text alerts once they decide to initiate the parking process.

It can definitely still be improved by adding more commands that the user can send, like one to receive continuous distance values on the phone until the parking is finished, without needing to repeatedly type the corresponding command. The distance sensor's accuracy could also be improved with additional proximity levels and more sound and LED alerts.