



iJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** XI **Month of publication:** November 2024

DOI: <https://doi.org/10.22214/ijraset.2024.65474>

www.ijraset.com

Call: ☎ 08813907089

E-mail ID: ijraset@gmail.com

IOT Based Parking Lot for Smart City

Vaishali Rajput¹, Adwait Kavishwar², Vivek Agrawal³, Adnaan Momin⁴, Yash Agiwal⁵, Advay Rath⁶, Aditya Nagdive⁷

Computer Science Engineering (Artificial Intelligence), Vishwakarma University

Abstract: This project presents an intelligent parking lot system using infrared sensors to detect available parking spaces in real-time. The system improves user convenience and reduces traffic congestion by providing accurate parking availability data. Infrared sensors, chosen for their reliability and low power consumption, are installed in each parking slot. The sensor data is processed centrally and shared via thingspeak website and digital displays.

The system will be piloted in a section of the parking lot, with calibration ensuring accuracy. Results show reduced time searching for parking and decreased congestion. The system's scalability, energy efficiency, and real-time capabilities enhance user experience and optimize parking operations. Future developments include predictive analytics and expansion to other high-traffic areas.

Keywords: infrared sensors, Arduino esp8266, ThingSpeak website, Parking availability, Parking management, Realtime detection, User convenience.

I. INTRODUCTION

Finding a place to park in cities is becoming a bigger issue, particularly in busy spots like shopping centers, business hubs, and public areas [1][8]. As city populations grow and the number of cars on the road increases, the demand for managing traffic has reached an all-time high. The current transportation systems often fall short in handling this demand, leading to delays and inefficiencies. Drivers frequently have to walk through parking lots in search of vacant spots, which not only annoys users but also leads to more traffic and higher carbon emissions. Research indicates that a large portion of city traffic is due to drivers looking for parking spots, creating a cycle of congestion and environmental harm [2]. Moreover, the absence of up-to-date information on parking availability makes it difficult for drivers to choose the best parking spots. Poor parking management can also hurt businesses in busy areas, as some customers may choose to avoid places known for their limited parking space [3]. As cities continue to grow, the need for creative solutions to make parking more efficient becomes more critical [5]. To tackle these significant issues, this project introduces a smart parking lot system that uses infrared sensors for instant detection of available parking spaces. By linking these sensors to ThingSpeak to show parking status in real-time, the system offers users precise, up-to-the-minute information, making parking more convenient and improving the flow of traffic inside the mall. This method enhances the dependability and low energy use of infrared sensors and ensures the system runs smoothly [6]. In the end, the system marks a major advancement in contemporary parking solutions, offering a flexible and economical way to address the urban parking challenge while supporting a greener city environment.

II. METHODOLOGY/EXPERIMENTAL

A. Theory

The intelligent parking system was designed to optimize parking management in cities [4], especially in high traffic areas such as shopping malls. This system uses infrared sensors with an Arduino microcontroller and ESP8266 Wi-Fi module to detect the availability of parking spaces in real time and provide accurate and timely information to users to improve their car experience.

This section describes the methods used to design, develop and implement a smart parking system incorporating infrared sensors, Arduino, ESP8266 and real-time data visualization [7]. The method is divided into several main parts, each of which contributes to the success of the project.

B. Components Used

- 1) **Infrared Sensor Nodes:** Each parking lot is equipped with infrared sensors that detect the presence or absence of vehicles. These sensors are reliable and efficient [6] and can operate continuously with reduced power consumption.
- 2) **Arduino Microcontroller:** The data collected from the infrared sensors is processed using the Arduino microcontroller. The Arduino is responsible for reading the sensor inputs, processing the data and determining the status of the parking spaces in real time.

- 3) **ESP8266 Wi-Fi Module:** The ESP8266 Wi-Fi module allows for seamless communication between the Arduino and the cloud-based platform, ThingSpeak. This module transmits the parking availability data to ThingSpeak, enabling users to access real-time information remotely [7].
- 4) **Boom Barrier:** The system includes a boom barrier that automatically opens only when a parking space is available in the lot. This feature enhances security and ensures that only authorized vehicles can access the parking area.
- 5) **LCD Screen:** An LCD screen is integrated into the system to display the status of each parking slot, indicating whether it is empty or full. This provides drivers with immediate visual feedback, helping them locate available spaces more efficiently.
- 6) **Data Visualization:** Integration with ThingSpeak displays available information in real time. Users can access this information through digital display boards placed at strategic points in the parking lot and the thingspeak console, which allows them to make parking decisions before they arrive.
- 7) **User Interfaces:** User-friendly interfaces, including the thingspeak and digital displays, provide intuitive access to parking information. These interfaces ensure that users can easily find available slots, reducing search time and improving overall user satisfaction.

C. Circuit Diagram

The connections of Esp8266 to Arduino Uno is shown (Fig 1), These connections are important to establish a valid connection through esp8266 to Thingspeak website for data transfer.

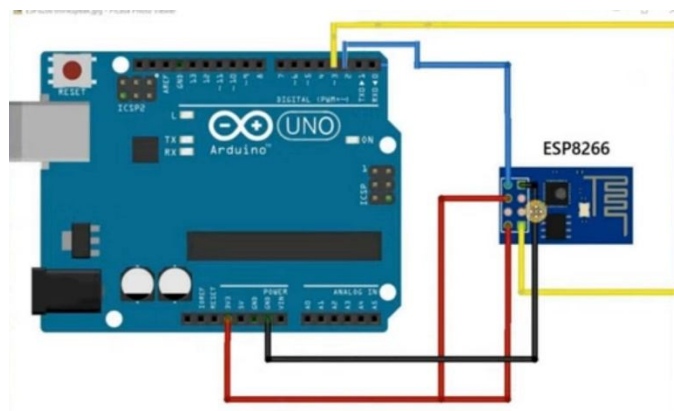


Fig2.1:Wifi Module circuit

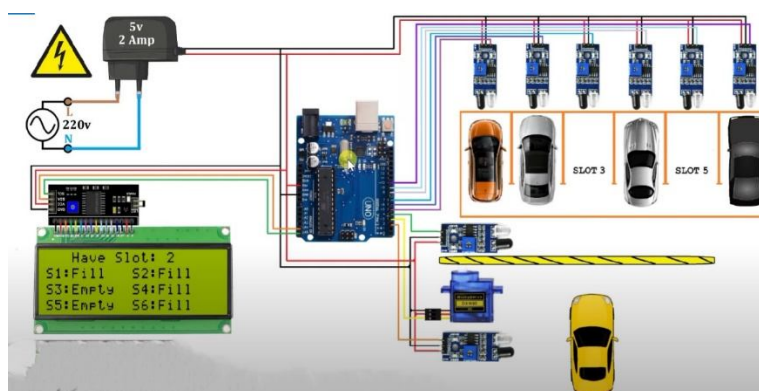


Fig2.2:Parking lot circuit

D. Implementation

The intelligent parking lot system is designed to enhance the management and monitoring of parking spaces by integrating several key components, including the ESP8266 microcontroller, infrared sensors, an LCD display, and a boom barrier. Each parking slot is equipped with infrared sensors that detect the presence of a vehicle, sending corresponding signals to the ESP8266 microcontroller. These signals are processed in real-time to determine whether a parking slot is occupied or available. The status of each slot is then displayed on the LCD screen, with "Full" (0) indicating that the slot is occupied and "Empty" (1) signalling availability.

To further improve the efficiency of the system, the ESP8266 is configured to communicate with ThingSpeak, a cloud-based IoT platform, where all parking data is stored and visualized in real time (Fig 3(a) & Fig 3(b)). This cloud integration allows for remote monitoring of the parking lot, enabling operators or users to access live updates on parking availability from anywhere via the internet. The system's ability to store historical data on ThingSpeak also offers opportunities for advanced data analysis, such as identifying patterns in parking usage and optimizing space management based on demand trends.

The boom barrier plays a crucial role in controlling entry to the parking facility. It is connected to the ESP8266 and operates based on real-time parking availability. If all slots are full, the boom barrier will remain closed, preventing further entry, while an available slot triggers the barrier to open, allowing the vehicle to park. This automated access control not only reduces congestion but also ensures efficient use of available parking spaces.

By providing immediate and accurate parking status updates, automating the entry process, and enabling remote monitoring, the system significantly enhances parking management operations. Its ability to streamline parking operations in real time contributes to reduced driver frustration, minimized congestion, and a more organized parking facility, making it particularly useful in high-traffic environments such as shopping malls, airports, and urban centers.



Figure 3(a): Status Of Parking Space "S1"

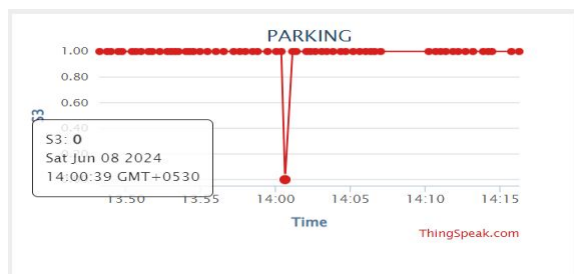


Figure 3(b). Status Of Parking Space "S3"

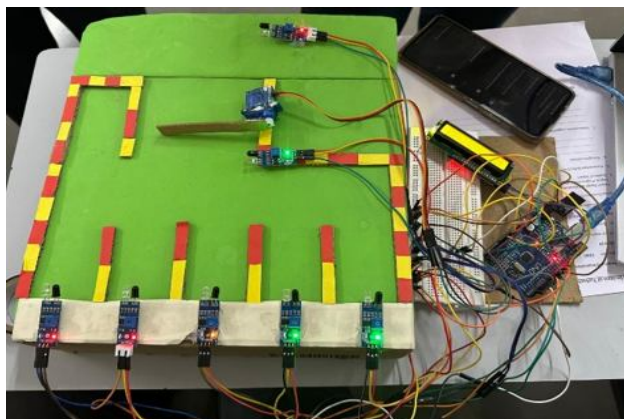


Fig 4: Real Life Implementation

The LCD screen has 6 specified abbreviations S1, S2, S3, S4, S5 (Parking Slot Numbers) and SA (Slot Availability). (Fig 5) The LCD screen displays the total amount of slots available as "SA:5", meaning there are currently 5 available spaces in the parking lot. This is followed by the status of every space with the letters "E" or "F" for showing empty or full spaces.

This inclusion of the LCD screen in the parking lot system offers several benefits that enhance user experience and operational efficiency. It provides real-time visual feedback on parking status, clearly indicating whether slots are empty or full. This immediate display of information helps drivers make quick decisions, reducing the time spent searching for parking and minimizing congestion in the lot.



Fig 5: LCD Screen With Status Of Each Slot

E. Code

```
1 #include <Servo.h>
2 #include <Wire.h>
3 #include <LiquidCrystal_I2C.h>
4 #include <SoftwareSerial.h>
5
6 LiquidCrystal_I2C lcd(0x27, 16, 2); // Adjust the I2C address and LCD size as needed
7 Servo myservo;
8 #define ir_enter 2
9 #define ir_back 4
10 #define ir_car1 5
11 #define ir_car2 6
12 #define ir_car3 7
13 #define ir_car4 8
14 #define ir_car5 9
15
16 int S1 = 0, S2 = 0, S3 = 0, S4 = 0, S5 = 0, enter = 0, leave = 0;
17 int slot = 5;
18
19 // WiFi
20 #define RX 10
21 #define TX 11
22 String AP = "VIVEK";
23 String PASS = "vivek123";
24 String API = "852APW0ULXK0H0AC";
25 String HOST = "api.thingspeak.com";
26 String PORT = "80";
27 String field1 = "field1";
28 String field2 = "field2";
29 String field3 = "field3";
30 String field4 = "field4";
31 String field5 = "field5";
32
33 int countTrueCommand;
34 int countTimeCommand;
35 boolean found = false;
36 SoftwareSerial esp8266(RX, TX);
37
38 void setup() {
39   lcd.init();
40   lcd.backlight();
41   Serial.begin(9600); // Initialize serial communication
42   pinMode(ir_car1, INPUT);
43   pinMode(ir_car2, INPUT);
44   pinMode(ir_car3, INPUT);
45   pinMode(ir_car4, INPUT);
46   pinMode(ir_car5, INPUT);
47   pinMode(ir_enter, INPUT);
48   pinMode(ir_back, INPUT);
49
50   myservo.attach(3);
51   myservo.write(180);
52
53   lcd.begin(16, 2); // Initialize LCD with 16 columns and 2 rows
54
55   Read_Sensor();
56   int total = S1 + S2 + S3 + S4 + S5;
57   slot = 5 - total; // Calculate initial slot availability
58
59   // Initialize Serial and Wi-Fi
60   Serial.begin(9600);
61   esp8266.begin(115200);
62   sendCommand("AT", 5, "OK");
63   sendCommand("AT+CWJAP=1", 5, "OK");
64   sendCommand("AT+CWJAP=\"\" + AP + \"\", \"\" + PASS + \"\", 20, \"OK\"");
65 }
66
67 void loop() {
68   myservo.write(180);
69   lcd.backlight();
70   Read_Sensor();
71
72   // For servo
73   if ((slot > 0 && enter == 0) || leave == 0) {
74     myservo.write(90);
75     delay(3000);
76   } else {
77     myservo.write(180);
78   }
79
80   // Update slot availability
81   slot = 0 + (S1 + S2 + S3 + S4 + S5);
82
83   // Display information on LCD
84   lcd.setCursor(0, 0);
85   lcd.print("SA:");
86   lcd.print(slot);
87   lcd.print(" S1:"); lcd.print(S1 ? "E" : "F");
88   lcd.print(" S2:"); lcd.print(S2 ? "E" : "F");
89
90   lcd.setCursor(0, 1);
91   lcd.print("S3:"); lcd.print(S3 ? "E" : "F");
92   lcd.print(" S4:"); lcd.print(S4 ? "E" : "F");
93   lcd.print(" S5:"); lcd.print(S5 ? "E" : "F");
94
95   // Display information on serial monitor
96   Serial.println("Slot Empty:");
97   Serial.println(slot);
98   Serial.println("Sensor status:");
99   Serial.print("S1: ");
100  Serial.println(S1);
101  Serial.print("S2: ");
102  Serial.println(S2);
103  Serial.print("S3: ");
104  Serial.println(S3);
105  Serial.print("S4: ");
106  Serial.println(S4);
107  Serial.print("S5: ");
108  Serial.println(S5);
109
110  // Read sensor data
111  Read_Sensor();
112
113  // Send data to Thingspeak
114  sendDataToThingspeak();
115
116  // Delay for 10 seconds
117  delay(10000);
118
119  // Check if slot is full
120  if (slot == 0) {
121    // Slot is full, stop the servo
122    myservo.write(180);
123    delay(3000);
124  }
125
126  // Check if slot is empty
127  if (slot > 0) {
128    // Slot is empty, start the servo
129    myservo.write(90);
130    delay(3000);
131  }
132
133  // Check if slot is full
134  if (slot == 0) {
135    // Slot is full, stop the servo
136    myservo.write(180);
137    delay(3000);
138  }
139
140  // Check if slot is empty
141  if (slot > 0) {
142    // Slot is empty, start the servo
143    myservo.write(90);
144    delay(3000);
145  }
146
147  // Check if slot is full
148  if (slot == 0) {
149    // Slot is full, stop the servo
150    myservo.write(180);
151    delay(3000);
152  }
153
154  // Check if slot is empty
155  if (slot > 0) {
156    // Slot is empty, start the servo
157    myservo.write(90);
158    delay(3000);
159  }
160
161  // Check if slot is full
162  if (slot == 0) {
163    // Slot is full, stop the servo
164    myservo.write(180);
165    delay(3000);
166  }
167
168  // Check if slot is empty
169  if (slot > 0) {
170    // Slot is empty, start the servo
171    myservo.write(90);
172    delay(3000);
173  }
174
175  // Check if slot is full
176  if (slot == 0) {
177    // Slot is full, stop the servo
178    myservo.write(180);
179    delay(3000);
180  }
181
182  // Check if slot is empty
183  if (slot > 0) {
184    // Slot is empty, start the servo
185    myservo.write(90);
186    delay(3000);
187  }
188
189  // Check if slot is full
190  if (slot == 0) {
191    // Slot is full, stop the servo
192    myservo.write(180);
193    delay(3000);
194  }
195
196  // Check if slot is empty
197  if (slot > 0) {
198    // Slot is empty, start the servo
199    myservo.write(90);
200    delay(3000);
201  }
202
203  // Check if slot is full
204  if (slot == 0) {
205    // Slot is full, stop the servo
206    myservo.write(180);
207    delay(3000);
208  }
209
210  // Check if slot is empty
211  if (slot > 0) {
212    // Slot is empty, start the servo
213    myservo.write(90);
214    delay(3000);
215  }
216
217  // Check if slot is full
218  if (slot == 0) {
219    // Slot is full, stop the servo
220    myservo.write(180);
221    delay(3000);
222  }
223
224  // Check if slot is empty
225  if (slot > 0) {
226    // Slot is empty, start the servo
227    myservo.write(90);
228    delay(3000);
229  }
230
231  // Check if slot is full
232  if (slot == 0) {
233    // Slot is full, stop the servo
234    myservo.write(180);
235    delay(3000);
236  }
237
238  // Check if slot is empty
239  if (slot > 0) {
240    // Slot is empty, start the servo
241    myservo.write(90);
242    delay(3000);
243  }
244
245  // Check if slot is full
246  if (slot == 0) {
247    // Slot is full, stop the servo
248    myservo.write(180);
249    delay(3000);
250  }
251
252  // Check if slot is empty
253  if (slot > 0) {
254    // Slot is empty, start the servo
255    myservo.write(90);
256    delay(3000);
257  }
258
259  // Check if slot is full
260  if (slot == 0) {
261    // Slot is full, stop the servo
262    myservo.write(180);
263    delay(3000);
264  }
265
266  // Check if slot is empty
267  if (slot > 0) {
268    // Slot is empty, start the servo
269    myservo.write(90);
270    delay(3000);
271  }
272
273  // Check if slot is full
274  if (slot == 0) {
275    // Slot is full, stop the servo
276    myservo.write(180);
277    delay(3000);
278  }
279
280  // Check if slot is empty
281  if (slot > 0) {
282    // Slot is empty, start the servo
283    myservo.write(90);
284    delay(3000);
285  }
286
287  // Check if slot is full
288  if (slot == 0) {
289    // Slot is full, stop the servo
290    myservo.write(180);
291    delay(3000);
292  }
293
294  // Check if slot is empty
295  if (slot > 0) {
296    // Slot is empty, start the servo
297    myservo.write(90);
298    delay(3000);
299  }
300
301  // Check if slot is full
302  if (slot == 0) {
303    // Slot is full, stop the servo
304    myservo.write(180);
305    delay(3000);
306  }
307
308  // Check if slot is empty
309  if (slot > 0) {
310    // Slot is empty, start the servo
311    myservo.write(90);
312    delay(3000);
313  }
314
315  // Check if slot is full
316  if (slot == 0) {
317    // Slot is full, stop the servo
318    myservo.write(180);
319    delay(3000);
320  }
321
322  // Check if slot is empty
323  if (slot > 0) {
324    // Slot is empty, start the servo
325    myservo.write(90);
326    delay(3000);
327  }
328
329  // Check if slot is full
330  if (slot == 0) {
331    // Slot is full, stop the servo
332    myservo.write(180);
333    delay(3000);
334  }
335
336  // Check if slot is empty
337  if (slot > 0) {
338    // Slot is empty, start the servo
339    myservo.write(90);
340    delay(3000);
341  }
342
343  // Check if slot is full
344  if (slot == 0) {
345    // Slot is full, stop the servo
346    myservo.write(180);
347    delay(3000);
348  }
349
350  // Check if slot is empty
351  if (slot > 0) {
352    // Slot is empty, start the servo
353    myservo.write(90);
354    delay(3000);
355  }
356
357  // Check if slot is full
358  if (slot == 0) {
359    // Slot is full, stop the servo
360    myservo.write(180);
361    delay(3000);
362  }
363
364  // Check if slot is empty
365  if (slot > 0) {
366    // Slot is empty, start the servo
367    myservo.write(90);
368    delay(3000);
369  }
370
371  // Check if slot is full
372  if (slot == 0) {
373    // Slot is full, stop the servo
374    myservo.write(180);
375    delay(3000);
376  }
377
378  // Check if slot is empty
379  if (slot > 0) {
380    // Slot is empty, start the servo
381    myservo.write(90);
382    delay(3000);
383  }
384
385  // Check if slot is full
386  if (slot == 0) {
387    // Slot is full, stop the servo
388    myservo.write(180);
389    delay(3000);
390  }
391
392  // Check if slot is empty
393  if (slot > 0) {
394    // Slot is empty, start the servo
395    myservo.write(90);
396    delay(3000);
397  }
398
399  // Check if slot is full
400  if (slot == 0) {
401    // Slot is full, stop the servo
402    myservo.write(180);
403    delay(3000);
404  }
405
406  // Check if slot is empty
407  if (slot > 0) {
408    // Slot is empty, start the servo
409    myservo.write(90);
410    delay(3000);
411  }
412
413  // Check if slot is full
414  if (slot == 0) {
415    // Slot is full, stop the servo
416    myservo.write(180);
417    delay(3000);
418  }
419
420  // Check if slot is empty
421  if (slot > 0) {
422    // Slot is empty, start the servo
423    myservo.write(90);
424    delay(3000);
425  }
426
427  // Check if slot is full
428  if (slot == 0) {
429    // Slot is full, stop the servo
430    myservo.write(180);
431    delay(3000);
432  }
433
434  // Check if slot is empty
435  if (slot > 0) {
436    // Slot is empty, start the servo
437    myservo.write(90);
438    delay(3000);
439  }
440
441  // Check if slot is full
442  if (slot == 0) {
443    // Slot is full, stop the servo
444    myservo.write(180);
445    delay(3000);
446  }
447
448  // Check if slot is empty
449  if (slot > 0) {
450    // Slot is empty, start the servo
451    myservo.write(90);
452    delay(3000);
453  }
454
455  // Check if slot is full
456  if (slot == 0) {
457    // Slot is full, stop the servo
458    myservo.write(180);
459    delay(3000);
460  }
461
462  // Check if slot is empty
463  if (slot > 0) {
464    // Slot is empty, start the servo
465    myservo.write(90);
466    delay(3000);
467  }
468
469  // Check if slot is full
470  if (slot == 0) {
471    // Slot is full, stop the servo
472    myservo.write(180);
473    delay(3000);
474  }
475
476  // Check if slot is empty
477  if (slot > 0) {
478    // Slot is empty, start the servo
479    myservo.write(90);
480    delay(3000);
481  }
482
483  // Check if slot is full
484  if (slot == 0) {
485    // Slot is full, stop the servo
486    myservo.write(180);
487    delay(3000);
488  }
489
490  // Check if slot is empty
491  if (slot > 0) {
492    // Slot is empty, start the servo
493    myservo.write(90);
494    delay(3000);
495  }
496
497  // Check if slot is full
498  if (slot == 0) {
499    // Slot is full, stop the servo
500    myservo.write(180);
501    delay(3000);
502  }
503
504  // Check if slot is empty
505  if (slot > 0) {
506    // Slot is empty, start the servo
507    myservo.write(90);
508    delay(3000);
509  }
510
511  // Check if slot is full
512  if (slot == 0) {
513    // Slot is full, stop the servo
514    myservo.write(180);
515    delay(3000);
516  }
517
518  // Check if slot is empty
519  if (slot > 0) {
520    // Slot is empty, start the servo
521    myservo.write(90);
522    delay(3000);
523  }
524
525  // Check if slot is full
526  if (slot == 0) {
527    // Slot is full, stop the servo
528    myservo.write(180);
529    delay(3000);
530  }
531
532  // Check if slot is empty
533  if (slot > 0) {
534    // Slot is empty, start the servo
535    myservo.write(90);
536    delay(3000);
537  }
538
539  // Check if slot is full
540  if (slot == 0) {
541    // Slot is full, stop the servo
542    myservo.write(180);
543    delay(3000);
544  }
545
546  // Check if slot is empty
547  if (slot > 0) {
548    // Slot is empty, start the servo
549    myservo.write(90);
550    delay(3000);
551  }
552
553  // Check if slot is full
554  if (slot == 0) {
555    // Slot is full, stop the servo
556    myservo.write(180);
557    delay(3000);
558  }
559
560  // Check if slot is empty
561  if (slot > 0) {
562    // Slot is empty, start the servo
563    myservo.write(90);
564    delay(3000);
565  }
566
567  // Check if slot is full
568  if (slot == 0) {
569    // Slot is full, stop the servo
570    myservo.write(180);
571    delay(3000);
572  }
573
574  // Check if slot is empty
575  if (slot > 0) {
576    // Slot is empty, start the servo
577    myservo.write(90);
578    delay(3000);
579  }
580
581  // Check if slot is full
582  if (slot == 0) {
583    // Slot is full, stop the servo
584    myservo.write(180);
585    delay(3000);
586  }
587
588  // Check if slot is empty
589  if (slot > 0) {
590    // Slot is empty, start the servo
591    myservo.write(90);
592    delay(3000);
593  }
594
595  // Check if slot is full
596  if (slot == 0) {
597    // Slot is full, stop the servo
598    myservo.write(180);
599    delay(3000);
600  }
601
602  // Check if slot is empty
603  if (slot > 0) {
604    // Slot is empty, start the servo
605    myservo.write(90);
606    delay(3000);
607  }
608
609  // Check if slot is full
610  if (slot == 0) {
611    // Slot is full, stop the servo
612    myservo.write(180);
613    delay(3000);
614  }
615
616  // Check if slot is empty
617  if (slot > 0) {
618    // Slot is empty, start the servo
619    myservo.write(90);
620    delay(3000);
621  }
622
623  // Check if slot is full
624  if (slot == 0) {
625    // Slot is full, stop the servo
626    myservo.write(180);
627    delay(3000);
628  }
629
630  // Check if slot is empty
631  if (slot > 0) {
632    // Slot is empty, start the servo
633    myservo.write(90);
634    delay(3000);
635  }
636
637  // Check if slot is full
638  if (slot == 0) {
639    // Slot is full, stop the servo
640    myservo.write(180);
641    delay(3000);
642  }
643
644  // Check if slot is empty
645  if (slot > 0) {
646    // Slot is empty, start the servo
647    myservo.write(90);
648    delay(3000);
649  }
650
651  // Check if slot is full
652  if (slot == 0) {
653    // Slot is full, stop the servo
654    myservo.write(180);
655    delay(3000);
656  }
657
658  // Check if slot is empty
659  if (slot > 0) {
660    // Slot is empty, start the servo
661    myservo.write(90);
662    delay(3000);
663  }
664
665  // Check if slot is full
666  if (slot == 0) {
667    // Slot is full, stop the servo
668    myservo.write(180);
669    delay(3000);
670  }
671
672  // Check if slot is empty
673  if (slot > 0) {
674    // Slot is empty, start the servo
675    myservo.write(90);
676    delay(3000);
677  }
678
679  // Check if slot is full
680  if (slot == 0) {
681    // Slot is full, stop the servo
682    myservo.write(180);
683    delay(3000);
684  }
685
686  // Check if slot is empty
687  if (slot > 0) {
688    // Slot is empty, start the servo
689    myservo.write(90);
690    delay(3000);
691  }
692
693  // Check if slot is full
694  if (slot == 0) {
695    // Slot is full, stop the servo
696    myservo.write(180);
697    delay(3000);
698  }
699
700  // Check if slot is empty
701  if (slot > 0) {
702    // Slot is empty, start the servo
703    myservo.write(90);
704    delay(3000);
705  }
706
707  // Check if slot is full
708  if (slot == 0) {
709    // Slot is full, stop the servo
710    myservo.write(180);
711    delay(3000);
712  }
713
714  // Check if slot is empty
715  if (slot > 0) {
716    // Slot is empty, start the servo
717    myservo.write(90);
718    delay(3000);
719  }
720
721  // Check if slot is full
722  if (slot == 0) {
723    // Slot is full, stop the servo
724    myservo.write(180);
725    delay(3000);
726  }
727
728  // Check if slot is empty
729  if (slot > 0) {
730    // Slot is empty, start the servo
731    myservo.write(90);
732    delay(3000);
733  }
734
735  // Check if slot is full
736  if (slot == 0) {
737    // Slot is full, stop the servo
738    myservo.write(180);
739    delay(3000);
740  }
741
742  // Check if slot is empty
743  if (slot > 0) {
744    // Slot is empty, start the servo
745    myservo.write(90);
746    delay(3000);
747  }
748
749  // Check if slot is full
750  if (slot == 0) {
751    // Slot is full, stop the servo
752    myservo.write(180);
753    delay(3000);
754  }
755
756  // Check if slot is empty
757  if (slot > 0) {
758    // Slot is empty, start the servo
759    myservo.write(90);
760    delay(3000);
761  }
762
763  // Check if slot is full
764  if (slot == 0) {
765    // Slot is full, stop the servo
766    myservo.write(180);
767    delay(3000);
768  }
769
770  // Check if slot is empty
771  if (slot > 0) {
772    // Slot is empty, start the servo
773    myservo.write(90);
774    delay(3000);
775  }
776
777  // Check if slot is full
778  if (slot == 0) {
779    // Slot is full, stop the servo
780    myservo.write(180);
781    delay(3000);
782  }
783
784  // Check if slot is empty
785  if (slot > 0) {
786    // Slot is empty, start the servo
787    myservo.write(90);
788    delay(3000);
789  }
790
791  // Check if slot is full
792  if (slot == 0) {
793    // Slot is full, stop the servo
794    myservo.write(180);
795    delay(3000);
796  }
797
798  // Check if slot is empty
799  if (slot > 0) {
800    // Slot is empty, start the servo
801    myservo.write(90);
802    delay(3000);
803  }
804
805  // Check if slot is full
806  if (slot == 0) {
807    // Slot is full, stop the servo
808    myservo.write(180);
809    delay(3000);
810  }
811
812  // Check if slot is empty
813  if (slot > 0) {
814    // Slot is empty, start the servo
815    myservo.write(90);
816    delay(3000);
817  }
818
819  // Check if slot is full
820  if (slot == 0) {
821    // Slot is full, stop the servo
822    myservo.write(180);
823    delay(3000);
824  }
825
826  // Check if slot is empty
827  if (slot > 0) {
828    // Slot is empty, start the servo
829    myservo.write(90);
830    delay(3000);
831  }
832
833  // Check if slot is full
834  if (slot == 0) {
835    // Slot is full, stop the servo
836    myservo.write(180);
837    delay(3000);
838  }
839
840  // Check if slot is empty
841  if (slot > 0) {
842    // Slot is empty, start the servo
843    myservo.write(90);
844    delay(3000);
845  }
846
847  // Check if slot is full
848  if (slot == 0) {
849    // Slot is full, stop the servo
850    myservo.write(180);
851    delay(3000);
852  }
853
854  // Check if slot is empty
855  if (slot > 0) {
856    // Slot is empty, start the servo
857    myservo.write(90);
858    delay(3000);
859  }
860
861  // Check if slot is full
862  if (slot == 0) {
863    // Slot is full, stop the servo
864    myservo.write(180);
865    delay(3000);
866  }
867
868  // Check if slot is empty
869  if (slot > 0) {
870    // Slot is empty, start the servo
871    myservo.write(90);
872    delay(3000);
873  }
874
875  // Check if slot is full
876  if (slot == 0) {
877    // Slot is full, stop the servo
878    myservo.write(180);
879    delay(3000);
880  }
881
882  // Check if slot is empty
883  if (slot > 0) {
884    // Slot is empty, start the servo
885    myservo.write(90);
886    delay(3000);
887  }
888
889  // Check if slot is full
890  if (slot == 0) {
891    // Slot is full, stop the servo
892    myservo.write(180);
893    delay(3000);
894  }
895
896  // Check if slot is empty
897  if (slot > 0) {
898    // Slot is empty, start the servo
899    myservo.write(90);
900    delay(3000);
901  }
902
903  // Check if slot is full
904  if (slot == 0) {
905    // Slot is full, stop the servo
906    myservo.write(180);
907    delay(3000);
908  }
909
910  // Check if slot is empty
911  if (slot > 0) {
912    // Slot is empty, start the servo
913    myservo.write(90);
914    delay(3000);
915  }
916
917  // Check if slot is full
918  if (slot == 0) {
919    // Slot is full, stop the servo
920    myservo.write(180);
921    delay(3000);
922  }
923
924  // Check if slot is empty
925  if (slot > 0) {
926    // Slot is empty, start the servo
927    myservo.write(90);
928    delay(3000);
929  }
930
931  // Check if slot is full
932  if (slot == 0) {
933    // Slot is full, stop the servo
934    myservo.write(180);
935    delay(3000);
936  }
937
938  // Check if slot is empty
939  if (slot > 0) {
940    // Slot is empty, start the servo
941    myservo.write(90);
942    delay(3000);
943  }
944
945  // Check if slot is full
946  if (slot == 0) {
947    // Slot is full, stop the servo
948    myservo.write(180);
949    delay(3000);
950  }
951
952  // Check if slot is empty
953  if (slot > 0) {
954    // Slot is empty, start the servo
955    myservo.write(90);
956    delay(3000);
957  }
958
959  // Check if slot is full
960  if (slot == 0) {
961    // Slot is full, stop the servo
962    myservo.write(180);
963    delay(3000);
964  }
965
966  // Check if slot is empty
967  if (slot > 0) {
968    // Slot is empty, start the servo
969    myservo.write(90);
970    delay(3000);
971  }
972
973  // Check if slot is full
974  if (slot == 0) {
975    // Slot is full, stop the servo
976    myservo.write(180);
977    delay(3000);
978  }
979
980  // Check if slot is empty
981  if (slot > 0) {
982    // Slot is empty, start the servo
983    myservo.write(90);
984    delay(3000);
985  }
986
987  // Check if slot is full
988  if (slot == 0) {
989    // Slot is full, stop the servo
990    myservo.write(180);
991    delay(3000);
992  }
993
994  // Check if slot is empty
995  if (slot > 0) {
996    // Slot is empty, start the servo
997    myservo.write(90);
998    delay(3000);
999  }
1000
1001  // Check if slot is full
1002  if (slot == 0) {
1003    // Slot is full, stop the servo
1004    myservo.write(180);
1005    delay(3000);
1006  }
1007
1008  // Check if slot is empty
1009  if (slot > 0) {
1010    // Slot is empty, start the servo
1011    myservo.write(90);
1012    delay(3000);
1013  }
1014
1015  // Check if slot is full
1016  if (slot == 0) {
1017    // Slot is full, stop the servo
1018    myservo.write(180);
1019    delay(3000);
1020  }
1021
1022  // Check if slot is empty
1023  if (slot > 0) {
1024    // Slot is empty, start the servo
1025    myservo.write(90);
1026    delay(3000);
1027  }
1028
1029  // Check if slot is full
1030  if (slot == 0) {
1031    // Slot is full, stop the servo
1032    myservo.write(180);
1033    delay(3000);
1034  }
1035
1036  // Check if slot is empty
1037  if (slot > 0) {
1038    // Slot is empty, start the servo
1039    myservo.write(90);
1040    delay(3000);
1041  }
1042
1043  // Check if slot is full
1044  if (slot == 0) {
1045    // Slot is full, stop the servo
1046    myservo.write(180);
1047    delay(3000);
1048  }
1049
1050  // Check if slot is empty
1051  if (slot > 0) {
1052    // Slot is empty, start the servo
1053    myservo.write(90);
1054    delay(3000);
1055  }
1056
1057  // Check if slot is full
1058  if (slot == 0) {
1059    // Slot is full, stop the servo
1060    myservo.write(180);
1061    delay(3000);
1062  }
1063
1064  // Check if slot is empty
1065  if (slot > 0) {
1066    // Slot is empty, start the servo
1067    myservo.write(90);
1068    delay(3000);
1069  }
1070
1071  // Check if slot is full
1072  if (slot == 0) {
1073    // Slot is full, stop the servo
1074    myservo.write(180);
1075    delay(3000);
1076  }
1077
1078  // Check if slot is empty
1079  if (slot > 0) {
1080    // Slot is empty, start the servo
1081    myservo.write(90);
1082    delay(3000);
1083  }
1084
1085  // Check if slot is full
1086  if (slot == 0) {
1087    // Slot is full, stop the servo
1088    myservo.write(180);
1089    delay(3000);
1090  }
1091
1092  // Check if slot is empty
1093  if (slot > 0) {
1094    // Slot is empty, start the servo
1095    myservo.write(90);
1096    delay(3000);
1097  }
1098
1099  // Check if slot is full
1100  if (slot == 0) {
1101    // Slot is full, stop the servo
1102    myservo.write(180);
1103    delay(3000);
1104  }
1105
1106  // Check if slot is empty
1107  if (slot > 0) {
1108    // Slot is empty, start the servo
1109    myservo.write(90);
1110    delay(3000);
1111  }
1112
1113  // Check if slot is full
1114  if (slot == 0) {
1115    // Slot is full, stop the servo
1116    myservo.write(180);
1117    delay(3000);
1118  }
1119
1
```

```

146 unsigned long startTime = millis();
147
148 while ((millis() - startTime) < (maxTime * 1000)) { // Timeout loop
149     while (esp8266.available()) {
150         char c = esp8266.read();
151         if (c == readReply.charAt(0)) {
152             unsigned long timeout = millis() + 1000; // Timeout for remaining characters in the reply
153             size_t i = 1;
154             while (i < readReply.length()) {
155                 if (timeout < millis()) break; // Timeout
156                 if (esp8266.available()) {
157                     if (esp8266.read() == readReply.charAt(i)) {
158                         i++;
159                     } else {
160                         break; // Mismatch
161                     }
162                 }
163             }
164             if (i == readReply.length()) {
165                 found = true;
166                 break;
167             }
168         }
169     }
170     if (found) break;
171 }
172
173 if (found) {
174     Serial.println("OVI");
175     countTrueCommand++;
176 } else {
177     Serial.println("Fail");
178     countTrueCommand = 0;
179 }
180 }
181

```

5.

```

void Read_Sensor() {
    S1 = digitalRead(ir_car1);
    S2 = digitalRead(ir_car2);
    S3 = digitalRead(ir_car3);
    S4 = digitalRead(ir_car4);
    S5 = digitalRead(ir_car5);
    enter = digitalRead(ir_enter);
    leave = digitalRead(ir_back);
}

```

6.

III. RESULTS AND DISCUSSIONS

A. Results

The implementation of the intelligent parking lot system delivered notable results in both operational efficiency and user experience, demonstrating the potential of IoT technology in modern parking management. The system's use of infrared sensors proved highly effective in detecting the presence of vehicles, achieving a consistently high detection rate with minimal false positives or negatives. This accuracy ensured that the LCD display could provide users with real-time, reliable status updates, indicating whether each parking slot was occupied (0) or available (1), significantly improving the decision-making process for drivers (Fig 6). This immediate and clear feedback helped reduce confusion and minimize the time spent searching for available slots, contributing to a smoother flow of vehicles within the parking area.

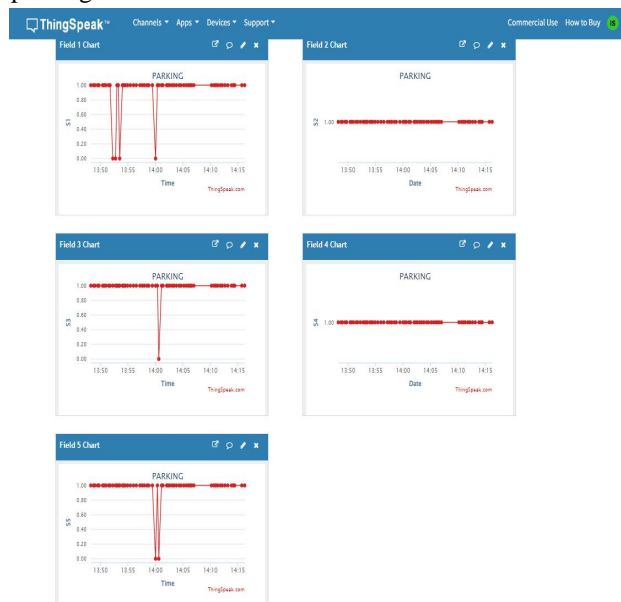


Fig 6: Dashboard of Thingspeak Showing Status Of Slots

The incorporation of the ESP8266 microcontroller was crucial for the system's operation, as it facilitated smooth and uninterrupted communication among the different parts of the parking lot system and the ThingSpeak platform. This connection allowed for the real-time transmission of data on the status of each parking space, whether it was in use or vacant, to ThingSpeak, where it was safely stored and visually represented. This information was not only available through ThingSpeak's online portal but also made compatible with potential future mobile applications, offering a highly engaging and user-friendly interface.

For visitors, the capability to check parking availability from a distance was immensely beneficial. Before arriving at the site, they could assess the status of parking spots, avoiding the hassle of searching for a space in a busy lot. This feature helped to reduce delays related to parking and lessened congestion, particularly during busy periods. Furthermore, the system's precision in detecting the presence of vehicles using infrared sensors ensured the reliability of the displayed parking status, boosting user confidence and satisfaction.

For those managing the parking lot, the ESP8266's function in linking to ThingSpeak offered an effective method for overseeing parking resources. Managers could keep an eye on the lot in real-time, pinpointed the busiest times, and collect data for later analysis. This data could be utilized to refine the allocation of parking spaces, spot patterns, and enhance the efficiency of operations. Moreover, the automation of the boom barrier based on the availability of parking spaces led to a more orderly and efficient parking process, ensuring that only vehicles could enter when spaces were vacant.

In conclusion, the combination of the ESP8266 with ThingSpeak significantly enhanced both the experience for users and the management of operations, establishing the system as a scalable solution for larger parking areas.

B. Discussion

The intelligent parking lot system presented in this study effectively addresses the issue of parking space management through the integration of IoT technologies. The use of infrared sensors, the ESP8266 microcontroller, and real-time communication with ThingSpeak has demonstrated significant improvements in parking efficiency by automating the process of detecting vehicle presence and controlling access through the boom barrier.

One key advantage of this system is its real-time monitoring capability, which ensures that users are always aware of the availability of parking spaces. This reduces congestion and improves the overall user experience, particularly in high-traffic environments like airports and malls. The system's integration with a mobile app, as proposed for future work, would further enhance this by allowing users to check slot availability remotely, reserve parking spaces in advance, and even automate payments.

However, the system is not without limitations. The current implementation relies on infrared sensors, which may be affected by environmental factors such as light and weather conditions. Exploring alternative sensing technologies, such as ultrasonic or camera-based solutions, could improve the accuracy and robustness of the system in future iterations.

Moreover, while the communication with ThingSpeak provides basic cloud functionality, integrating more advanced data analytics platforms could allow for predictive analysis, identifying patterns in parking usage over time. This would enable parking lot operators to anticipate demand and optimize space management, potentially reducing operational costs.

In terms of scalability, while the system works well for small to medium-sized parking lots, expanding it to larger facilities could pose challenges related to network bandwidth and data processing. Addressing these issues would be critical for future deployments in larger urban centers.

Overall, the system presents a solid foundation for smart parking solutions, with numerous opportunities for enhancement and broader implementation.

IV. CONCLUSIONS

In conclusion, the intelligent parking lot system demonstrates a successful application of modern technology to address common parking challenges, particularly in high-traffic environments such as airports and shopping malls. By integrating components such as the ESP8266, infrared sensors, an LCD display, and a boom barrier, the system effectively monitors parking availability and provides real-time updates to users.

This capability is especially beneficial in busy settings, where drivers often face long wait times searching for parking spaces. With the LCD displaying occupancy status, travelers can quickly ascertain available slots, significantly reducing congestion and enhancing the overall experience. Additionally, the automated control of the boom barrier ensures that only vehicles with confirmed parking can enter, further streamlining access and improving security.

The ability to communicate with ThingSpeak facilitates remote monitoring and data analysis, allowing fleet managers to track usage patterns and optimize space management. This compatibility not only increases operational efficiency, but also supports strategic planning for future developments or changes. Overall, this project demonstrates the potential of IoT solutions to transform traditional fleet management into a better and more user-friendly experience and sets the stage for future developments in smart car technologies can improve user satisfaction in airport and shopping areas.

V. ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to everyone who contributed to the development of the intelligent parking lot system. Special thanks to our academic advisors for their invaluable guidance and support throughout the project. We also appreciate our peers for their constructive feedback and collaboration, which greatly enhanced our work. Additionally, we are grateful to our families and friends for their unwavering encouragement and belief in our capabilities. This project would not have been possible without the collective efforts of all involved, and we sincerely thank you for your contributions.

VI. FUTURE SCOPE

The future scope of the parking lot system includes the implementation of a camera system that can enhance vehicle monitoring and fee collection, along with a mobile app to provide users with real-time status updates and data analytics.

- 1) **Camera System Integration:** By incorporating a camera system equipped with license plate recognition technology, the system can automatically record vehicle entries and exits. This would facilitate automated fee collection based on parking duration, reducing the need for physical payment stations and streamlining the entire process. Additionally, this system can enhance security by monitoring vehicle movements and preventing unauthorized access.
- 2) **Mobile Application Development:** A dedicated mobile app would allow users to view real-time parking availability, reserve slots, and receive notifications on their smartphones. The app could also provide a user-friendly interface for managing payments and accessing transaction histories, further enhancing convenience.
- 3) **Data Science Analytics:** Leveraging data science techniques to analyze parking patterns and user behavior can yield valuable insights. Predictive analytics could forecast peak parking times, enabling better space allocation and management. Additionally, data visualization tools could present historical usage trends, aiding parking lot operators in decision-making and operational improvements.
- 4) **User Feedback and Engagement:** The mobile app can incorporate a feedback mechanism to gather user input on their parking experience, helping to identify areas for improvement and enhance customer satisfaction. Engaging users through loyalty programs or rewards for frequent parking could also encourage repeated use and foster a sense of community.
- 5) **Integration with Smart City Initiatives:** Aligning the parking system with broader smart city initiatives could enhance urban mobility. Collaborating with local transportation agencies to provide information on public transport options near the parking facility would offer users alternative travel solutions, reducing reliance on personal vehicles.
- 6) **Environmental Impact Monitoring:** Implementing sensors to monitor emissions and air quality in and around the parking facility could support sustainability goals. Providing users with information on the environmental impact of their parking habits could encourage eco-friendly choices.

REFERENCES

- [1] Hossam El-Din I. S. Ahmed (2017) Car Parking Problem In Urban Areas, Causes And Solutions The 1st International Conference: Towards A Better Quality of Life 24 - 26 November 2017 Technische Universität Berlin Campus El Gouna, Egypt.
- [2] Janak Parmar 2020 Study on demand and characteristics of parking system in urban areas: A review Journal of Traffic and Transportation Engineering (English Edition) Volume 7, Issue 1, February 2020, Pages 111-124.
- [3] Poor parking management takes its toll on business reputation [\[online\]](#)
- [4] Gurkan Celik, Aknur Sarsenbay, Abdelmalik taleb-ahmed. Innovative Parking Solutions in Smart Cities: Conference 2023 8th International Conference on Computer Science and Engineering (UBMK)
- [5] Why traditional parking management needs an upgrade, [\[online\]](#) wayleadr 2022
- [6] Advantages of Infrared Sensors [\[online\]](#) GSTiR
- [7] Debapriya Parida, 2019, Real-time Environment Monitoring System using ESP8266 and ThingSpeak on Internet of Things Platform: Conference 2019 International Conference on Intelligent Computing and Control Systems ICCS
- [8] Tejash Kumar Uttambhai Patel, A Review on "Parking Issues and Challenges in CBD Area", International Journal for Modern Trends in Science and Technology, 8(07): 74-80, 2022



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)