

Micros Peripheral and Interfaces

Controlled Car Park

Mostafa Ghotbi

GHO03058783

Contents

1	Controlled Car Park	4
1.1	Inputs and Outputs	4
2	Flow Chart	6
3	Programming	7
3.1	Entering a car into the Car Park	9
3.2	Leaving a car from the car park when the Button is pressed	11
3.3	Leaving a car from the car park when the Button is Not pressed	12
3.4	The car is leaving when the car park is Full.....	13
4	Checking the Performance of the program in Tinkercard	14
4.1	Checking the performance of the press Button, Sensor1 and the lights for entering a car .	14
4.2	Leaving a car from the car park	17
4.3	Program Code	18
5	Conclusion.....	22
6	Future work and improvements	23

Table of Figures

Figure 1. Controlled car park.....	4
Figure 2. Connection between the default Arduino pins (D8, D9, D10 and D11) to the Motor shield.	5
Figure 3. The connection of the LED lights, Press Button, Sensor1 and Sensor2 and resistors to the Arduino in Tinkercard.....	5
Figure 4. The controlled car park flow chart.	6
Figure 5. Testing the program. Strat simulation.	14
Figure 6. Clicking on the press Button appears the “Rising” message and the green light turns on.	15
Figure 7. Click on the sensor1 button, shows the “Lowering” and the “the count is now 1” on the serial monitor. The Green lights turns off and the red light turn back on again.	15
Figure 8. Clicking on the press Button increases the number of cars in the car park.	16
Figure 9. Reaching the maximum number of cars in the car park. Pressing more on the Button, does not let any more car to enter the car park. Also, a blue light illuminates to show the Full sign.	16
Figure 10. Reducing the number of cars in the car park when it is already full, by pressing the sensor2 button. The blue light turns off and red turns on again.....	17
Figure 11. Pressing on the sensor2, shows that the cars can leave even if the car park is not full. This decrease in the capacity, appears on the Serial Monitor screen	17

1 Controlled Car Park

This report presents a design for a single entrance, exit flow free car park which is controlled by a traffic light system and a barrier. An Arduino microcontroller provides the control functionality of the system. A terminal connected via a USB link provides an operator interface. (See

Figure

1)

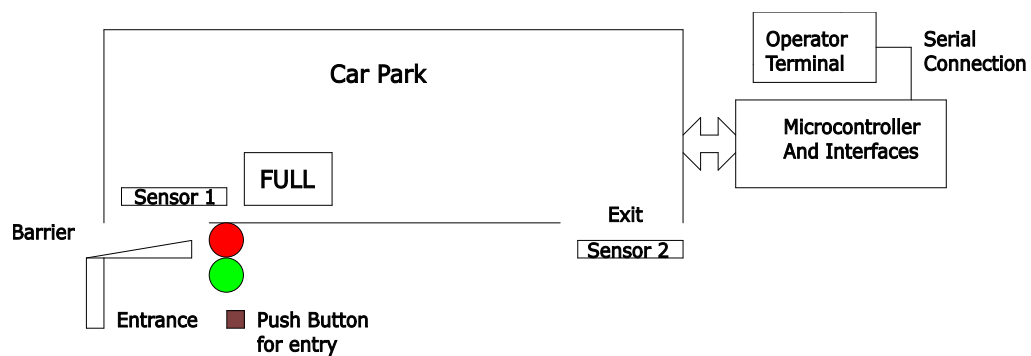


Figure 1. Controlled car park.

1.1 Inputs and Outputs

First, all the inputs and outputs and their connections to Arduino needs to be identified. The inputs are included of; the Press Button, Sensor1 (for the entrance) and Sensor2 (for the exit). The outputs are included of; the LED Green light (when the barrier is raised and shows that it is safe to pass the barrier to the car park), Red light (which is on before the barrier raises), Blue light (the Full Sign which shows that the car park is full) and the Motor shield. The Motor shield controls the barrier and contains of some high - power transistors that drive the motor. In order to find the number of connections between the Motor shield and Arduino, it needs to check the Stepper library. This library allows to control unipolar or bipolar stepper motors.

```
#include <Stepper.h
```

Using this command in the software, gives access to all functions in the library.

In Stepper One Revolution, it shows how to drive a Stepper motor and the interfaces and shows how it can be to logically connect. *Figure 2* shows four connections between the default Arduino pins, D8, D9, D10 and D11 and the shielded.

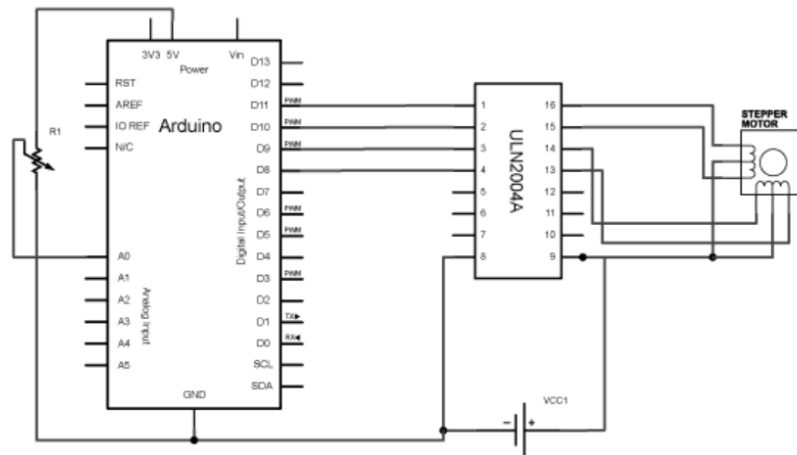


Figure 2. Connection between the default Arduino pins (D8, D9, D10 and D11) to the Motor shield.

Figure 3 shows the components which are used in this project and their connections to the Arduino in Tinkercard software.

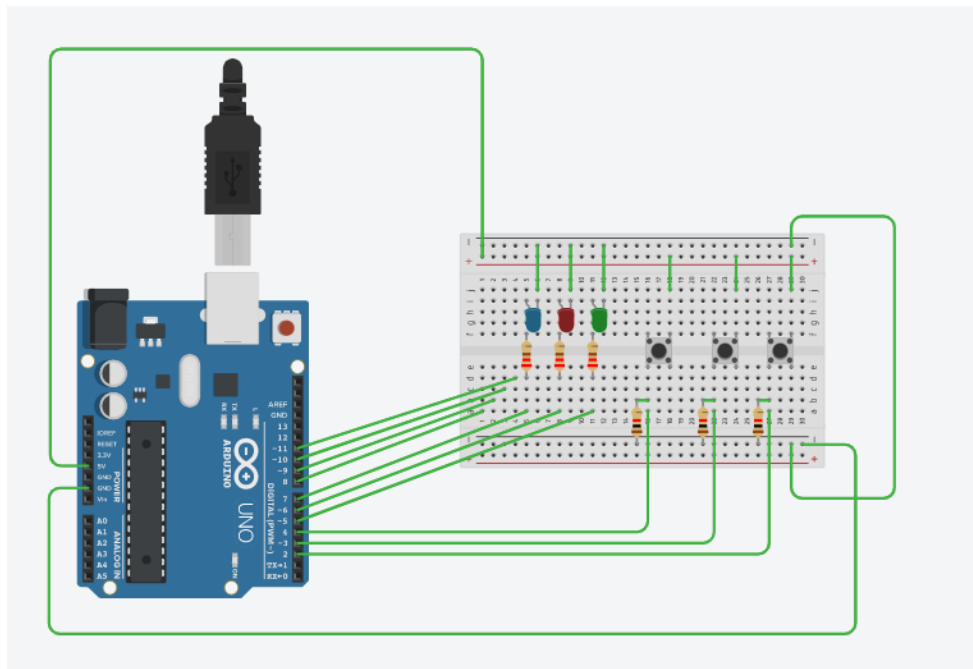


Figure 3. The connection of the LED lights, Press Button, Sensor1 and Sensor2 and resistors to the Arduino in Tinkercard.

2 Flow Chart

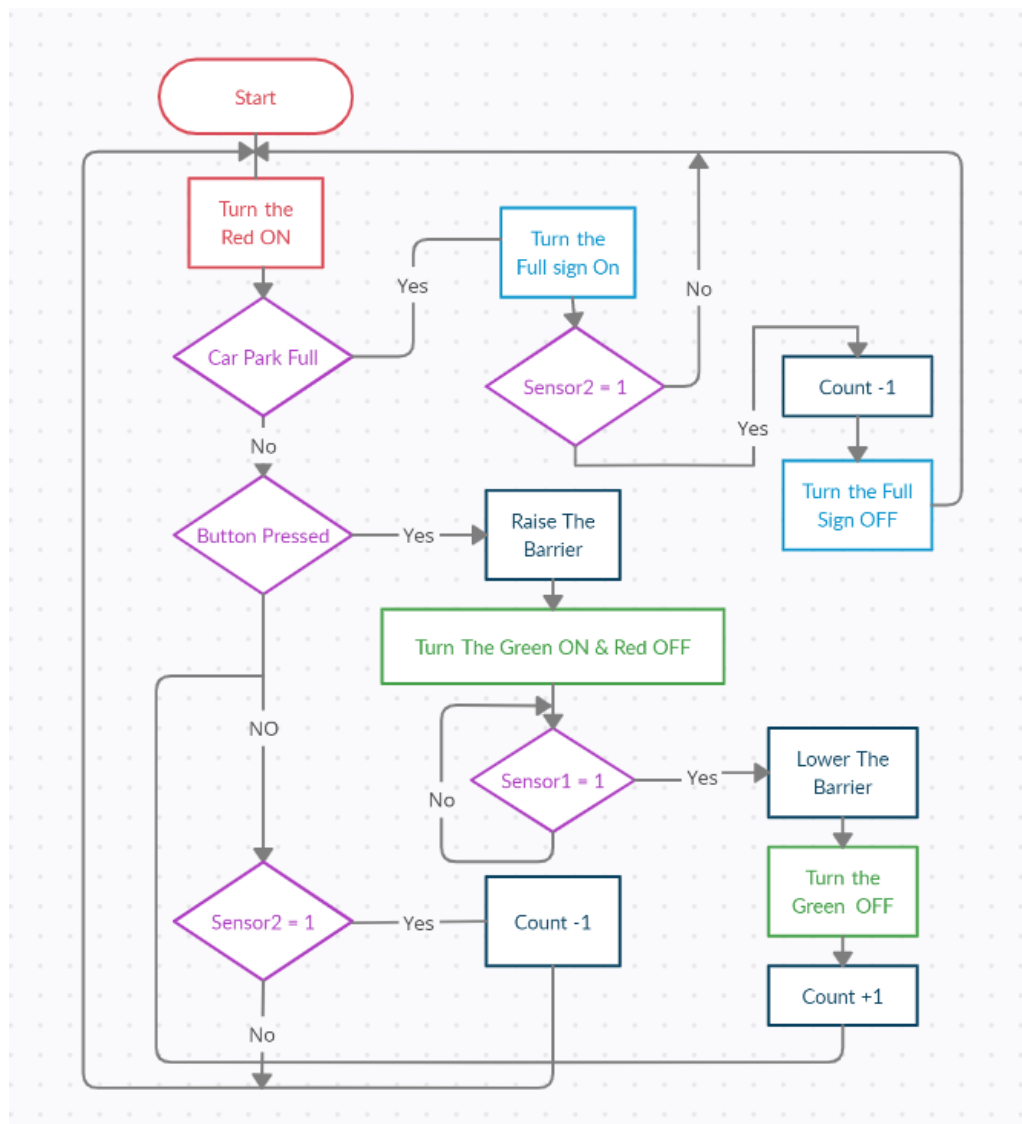


Figure 4. The controlled car park flow chart.

3 Programming

This command gives access to all functions in the library for stepping motor.

```
#include <Stepper.h>
```

This piece of program defines all inputs and outputs. It shows that the inputs Press Button, sensor1 and sensor 2 are connected to the ports 2, 3 and 4 of the Arduino, respectively. The outputs of Green, Red and Full (Blue) LED lights are connected to the pins 5, 6 and 7 respectively.

```
const int Button = 2;  
const int sensor1 = 3;  
const int sensor2 = 4;  
const int Green = 5;  
const int Red = 6;  
const int Full = 7;
```

This part defines the stepsPerRevolution in the program and shows that it has 200 steps.

```
const int stepsPerRevolution = 200; // change this to fit the number of steps per revolution  
                                     // for your motor
```

Integers are the primary data - type for number storage. We need to store the capacity of the carpark (fullpark), the number of cars in the car park (cars), the logical number (0 or 1) of the sensor1 (x), the cars are leaving the car park (outsense) and the storage for updating the number of leaving cars (storedsense).

Here also, the stepper library is initialized on pins 8 through 11.

```
int fullpark = 10;
int cars = 0; // this holds the number of cars in the carpark
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
int x=0;
int outsense = 0;
int storedsensor2 = 0;
```

This loop (void setup) sets the speed of the motor to 60 rpm. Serial.begin(9600) starts serial communication, so that the Arduino can send out commands through the USB connection. This is how fast the data is to be sent.

pinMode configures the specified pin to behave either as an input or an output.

```
void setup ( ) {
  // set the speed at 60 rpm:
  myStepper.setSpeed (60);
  // initialize the serial port:
  Serial.begin (9600);
  pinMode (Button, INPUT);
  pinMode (sensor1, INPUT);
  pinMode (sensor2, INPUT);
  pinMode (Green, OUTPUT);
  pinMode (Red, OUTPUT);
  pinMode (Full, OUTPUT);
}
```

3.1 Entering a car into the Car Park

This is where the bulk of Arduino sketch is executed. The program starts exactly after the opening curly bracket ({), runs until it sees the closing curly bracket (}), and jumps back up to the first line in loop () and starts all over.

The Red LED light is on because the barrier is still closed. If the car park is not full and the Button is pressed, then the barrier will rise up, the message of “Rising” will appear and the Red - light changes to the green light.

```
void loop ( ) {           // void loop started.
  digitalWrite (Red, HIGH);
  delay (500);
  if (cars < fullpark) {   //Car parc has space
    if (digitalRead (Button) == 1){ //Button loop started
      myStepper.step (stepsPerRevolution/4);
```

The (myStepper.step) is a command in the library and it says the number of steps we need to turn. For rising the barrier, we need to turn the motor quarter of the Revolution and clockwise. For that, we need to divide the stepsPerRevolution by 4. For lowering the barrier, it will be the same but in negative, which means it is anti - clockwise.

myStepper.step

Using do – while loop, the sensor1 is checked and read until the car reaches this sensor position in the entrance (x=1), then it is safe to lower the barrier and the green light changes to Red again. Then, the number of cars in the car park will be increment using; cars ++ .

```
do
{
} while
```

```

void loop ( ) {                                // void loop started.
  digitalWrite (Red, HIGH);
  delay (500);
  if (cars < fullpark) {                        //Car parc has space
    if (digitalRead (Button) == 1){           //Button loop started
      myStepper.step (stepsPerRevolution/4);
      Serial.println ("Rising ");
      delay (500);
      digitalWrite (Red, LOW);
      digitalWrite (Green, HIGH);
    }
    do
    {
      delay(50);                               // wait for sensors to stabilize
      x = digitalRead (sensor1);               // check the sensors
    } while (x < 1);
    myStepper.step (-stepsPerRevolution/4);
    Serial.println ("lowering ");
    digitalWrite (Green, LOW);
    cars ++;                                   // increments the number of cars in the car park
    Serial.print (" the count is now ");
    Serial.println (cars);
  }
}

```

3.2 Leaving a car from the car park when the Button is pressed

In this part, we reached the point in the flow chart when a car is leaving, and the Button is pressed. Sensor2 is responsible to check the cars are leaving, therefore, it needs to be checked and read and the logical number 0 means no car is leaving and 1 means that a car is leaving, will be saved in outsense and the value of the outsense will be stored and updated in the storedsensor2. If outsense = 1, (a car is leaving), then the number of cars inside the car park will be reduce (cars --) and every time will be updated in the storedsensor2. We need to put a condition of (cars > 0), otherwise if we click the sensor2 in the Tinkercard more than the number of available cars in the car park, the value of available cars, will go to negative number which doesn't really make any sense (such as; -1, -2, ...).

```
if (cars > 0) { ;
  outsense = digitalRead (sensor2);  // see if the car is leaving the car park
  if (outsense == 1) {
    Serial.println ("sensor2 at 1");
    if (storedsensor2 == 0) {
      cars -- ;
      Serial.print ("reduce number to ");
      Serial.println (cars) ;
    }
    // sensor2 read and reduced the number of cars
  }
  //sensor2 read and reduced the number of cars
  storedsensor2 = outsense;
}
//Button loop finished
```

3.3 Leaving a car from the car park when the Button is Not pressed

Looking at the flow chart shows that the program needs to cover the situation when the Button is not pressed but a car which is already in the car park wants to leave. This part of the program is out of the pressed Button loop and contains the similar commands and lines in the previous section.

```
if (cars > 0){;
  outsense = digitalRead (sensor2);      // see if the car is leaving the car park
  if (outsense == 1) {
    Serial.println ("sensor2 at 2");
    if (storedsensor2 == 0) {
      cars -- ;
      Serial.print ("reduce number to ");
      Serial.println (cars);
    }
  } // sensor2 read and reduced the number of cars
} //sensor2 read and reduced the number of cars

storedsensor2 = outsense;
  }
} // Car park space loop finished
```

3.4 The car is leaving when the car park is Full

All above program lines were true if the car park was not full but in case of the car park is reached its maximum space and is full, and a car wants to leave the car park, (check the flow chart when the car park is full and we reach the sensor2), we need to add another part to this program. This part is very similar to both previous parts, just the position of this part is out of the loop when the car park is not full. In this section, if the entrance Button is pressed when the car park is full, the blue LED light which is the Full sign, will be eliminate. The whole void loop will finish here.

```
if (cars == fullpark) { ;
    outsense = digitalRead (sensor2); // see if the car is leaving the car park
    if (outsense == 1) {
        Serial.println ("sensor2 at 3");
        if (storedsensor2 == 0) {
            cars -- ;
            digitalWrite (Full, LOW);
            Serial.print ("reduce number to ");
            Serial.println (cars);
        } // sensor2 read and reduced the number of cars
    } //sensor2 read and reduced the number of cars
    if (digitalRead (Button) == 1){;
        digitalWrite (Full, HIGH);
        }
    storedsensor2 = outsense;
    }
} //Void loop finished
```

4 Checking the Performance of the program in Tinkercard

The written program is checked in the Tinkercard software. This is the results;

4.1 Checking the performance of the press Button, Sensor1 and the lights for entering a car

Clicking on the start simulation, is the stage that the push button is not pressed yet and the barrier is closed therefore, the red light is on. See Figure 5.

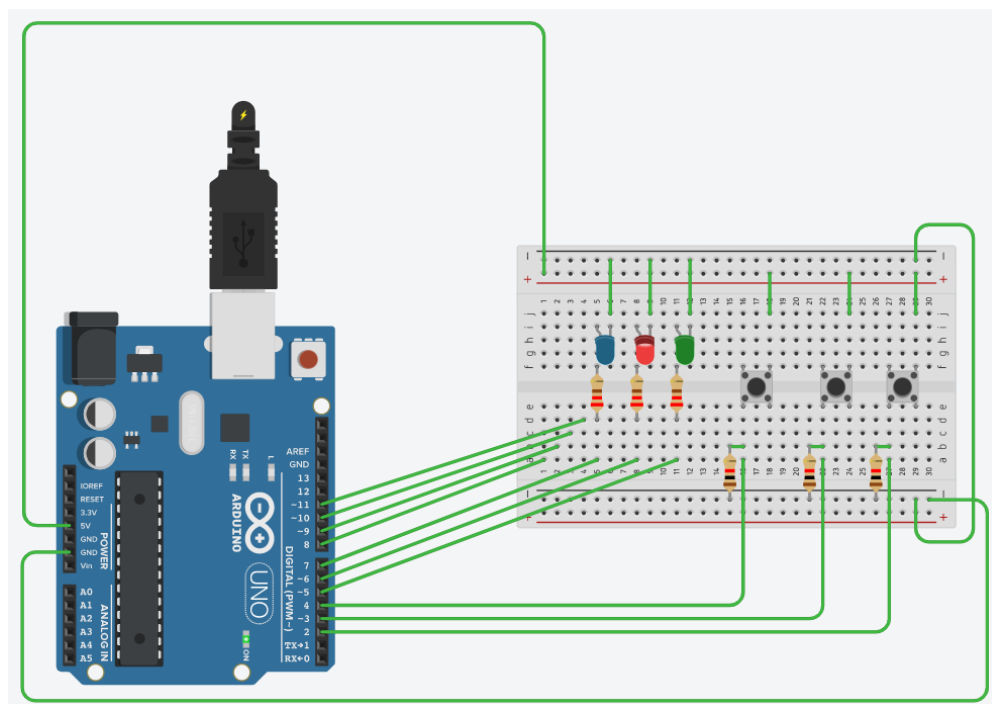


Figure 5. Testing the program. Strat simulation.

Clicking on the press Button (carpark is not full yet) shows the “Rising” message. The Red light turns off and the Green turns on. See Figure 6.

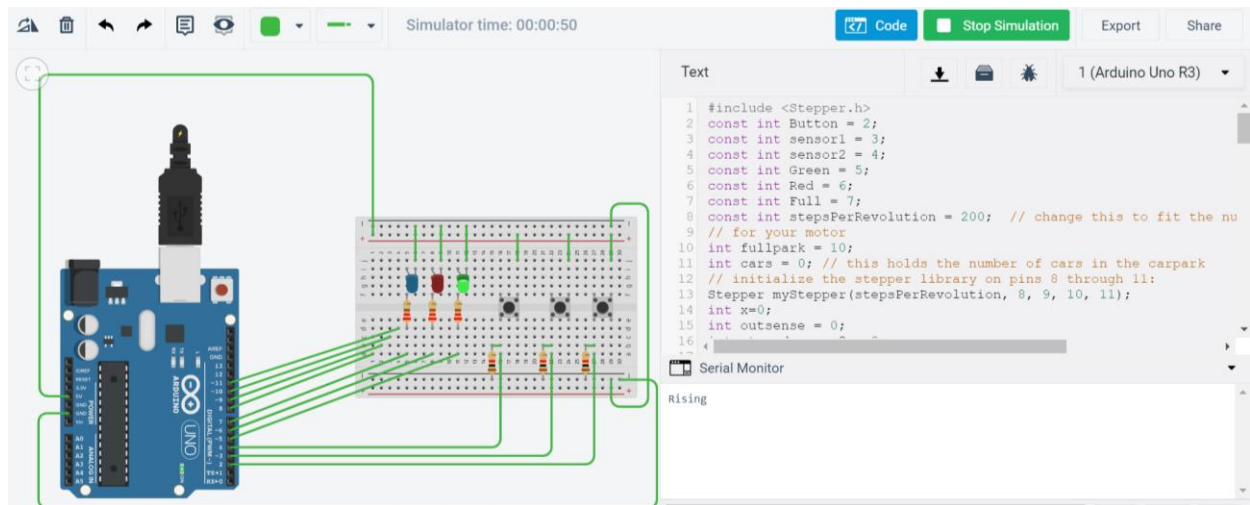


Figure 6. Clicking on the press Button appears the “Rising” message and the Green light turns on.

Clicking on the sensor1, turns the green light off and the red light on. It shows the “Lowering” and “the count is now 1” message on the serial monitor. See Figure 7.

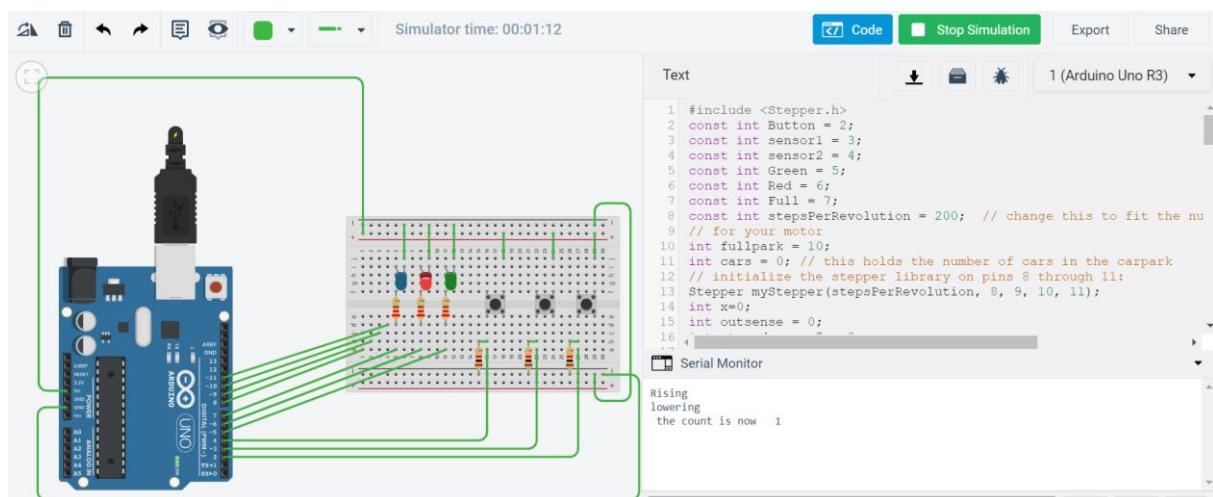


Figure 7. Click on the sensor1 button, shows the “Lowering” and the “the count is now 1” on the serial monitor. The Green lights turns off and the red light turn back on again.

More clicking on the press Button, increases the number of cars in the car park. See Figure 8.

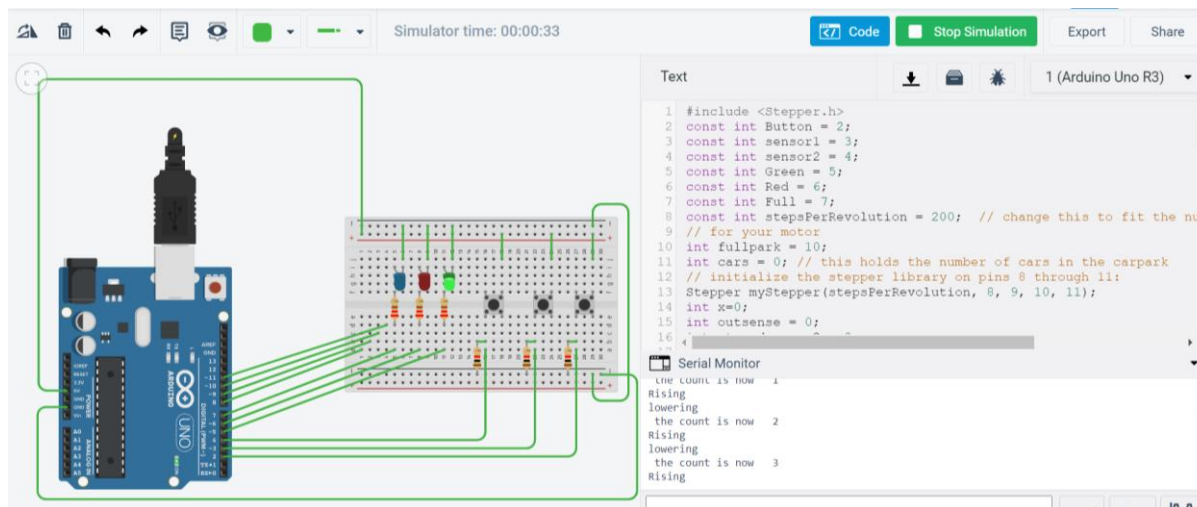


Figure 8. Clicking on the press Button increases the number of cars in the car park.

After reaching the maximum capacity of the car park (which is 10 here), more clicking on the press Button does not let any car to come into the car park. Also, clicking on the press Button at this stage, turns the blue light on which is designed to alert that the car park is Full. This light will not change until a car leaves the car park. See Figure 9.

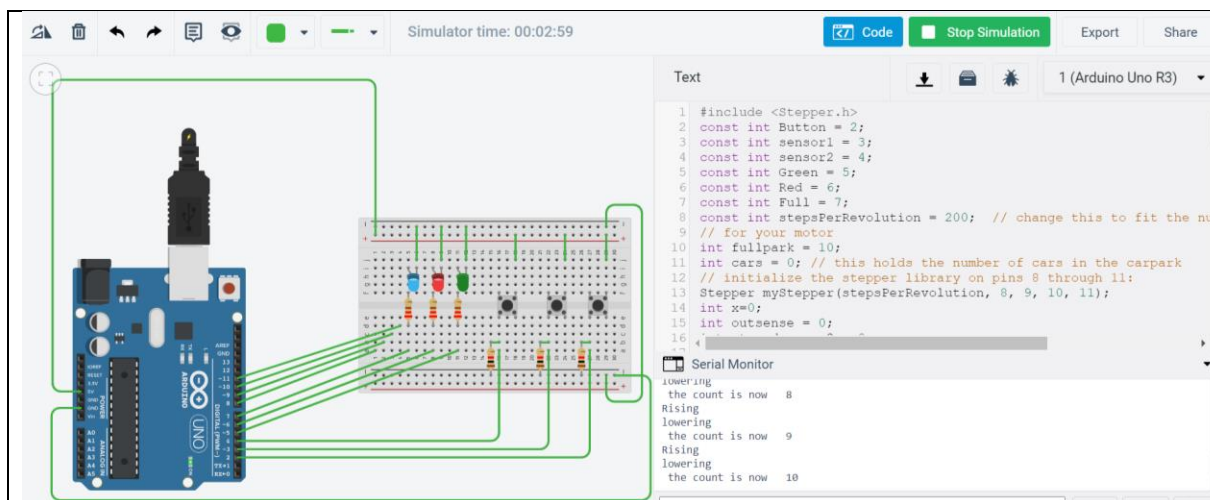


Figure 9. Reaching the maximum number of cars in the car park. Pressing more on the Button, does not let any more car to enter the car park. Also, a blue light illuminates to show the Full sign.

4.2 Leaving a car from the car park

Clicking on the sensor2, reduces the number of cars in the car park when the car park is full. It also, turns the blue light off and red on again. See Figure 10.

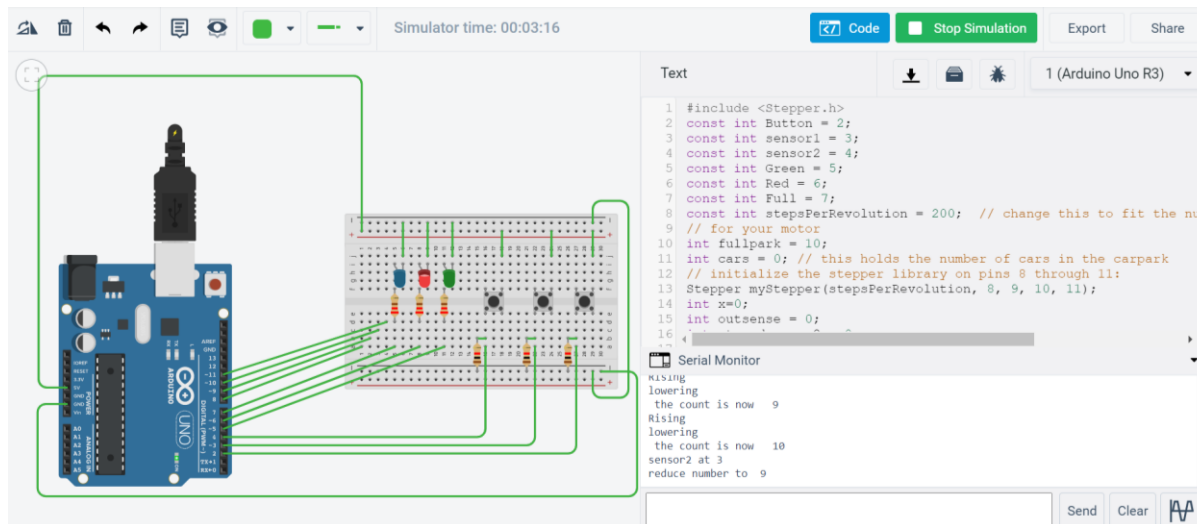


Figure 10. Reducing the number of cars in the car park when it is already full, by pressing the sensor2 button. The blue light turns off and red turns on again.

The cars can leave the car park even when the car park is not full, and this decrease the number of cars inside the car park which appears on the Serial Monitor. See Figure 11.

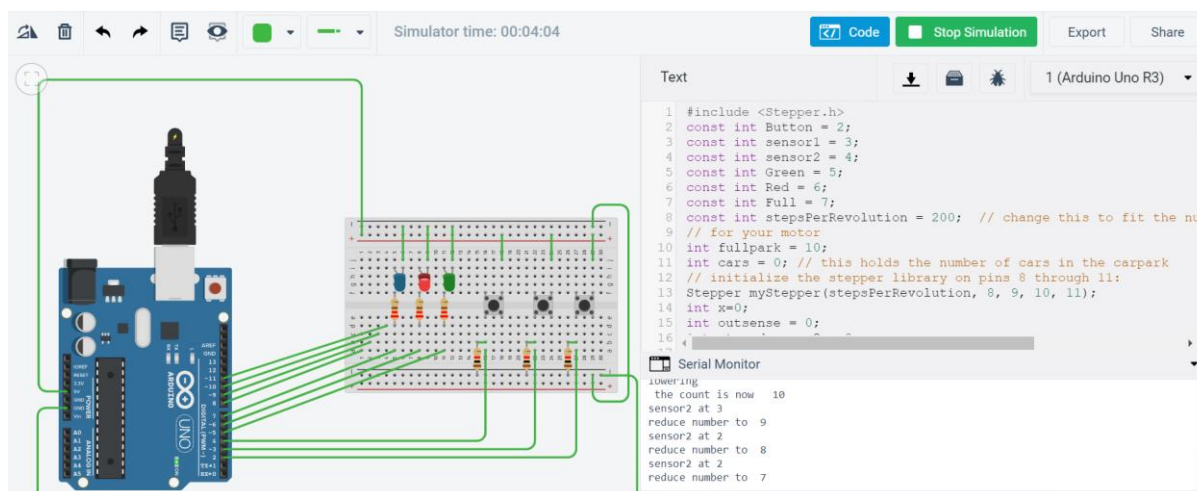


Figure 11. Pressing on the sensor2, shows that the cars can leave even if the car park is not full. This decrease in the capacity, appears on the Serial Monitor screen

4.3 Program Code

```
#include <Stepper.h>

const int Button = 2;

const int sensor1 = 3;

const int sensor2 = 4;

const int Green = 5;

const int Red = 6;

const int Full = 7;

const int stepsPerRevolution = 200; // change this to fit the number of steps per revolution
// for your motor

int fullpark = 10;

int cars = 0; // this holds the number of cars in the carpark

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

int x=0;

int outsense = 0;

int storedsensor2 = 0;

void setup() {

    // set the speed at 60 rpm:
    myStepper.setSpeed(60);

    // initialize the serial port:
    Serial.begin(9600);

    pinMode(Button, INPUT);

    pinMode(sensor1, INPUT);

    pinMode(sensor2, INPUT);
```

```

pinMode(Green, OUTPUT);

pinMode(Red, OUTPUT);

pinMode(Full, OUTPUT);
}

void loop() {

    digitalWrite(Red, HIGH);

    delay (500);

    if (cars < fullpark){ //Car parc has space

        if(digitalRead(Button)==1){ //Button loop started

            myStepper.step(stepsPerRevolution/4);

            Serial.println("Rising ");

            delay (500);

            digitalWrite(Red, LOW);

            digitalWrite(Green, HIGH);

        do

        {

            delay(50);    // wait for sensors to stabilize

            x = digitalRead(sensor1); // check the sensors

        } while(x < 1);

            myStepper.step(-stepsPerRevolution/4);

            Serial.println("lowering ");

            digitalWrite(Green, LOW);

            cars ++; // increments the number of cars in the car park

            Serial.print(" the count is now ");

            Serial.println(cars);

```

```

if (cars > 0){
  outsense = digitalRead(sensor2); // see if the car is leaving the car park
  if (outsense == 1){
    Serial.println("sensor2 at 1");
    if (storedsensor2 == 0){
      cars --;
      Serial.print("reduce number to ");
      Serial.println(cars);

    } // sensor2 read and reduced the number of cars
  } //sensor2 read and reduced the number of cars

  storedsensor2 = outsense;
}

} //Button loop finished

if (cars > 0){
  outsense = digitalRead(sensor2); // see if the car is leaving the car park
  if (outsense == 1){
    Serial.println("sensor2 at 2");
    if (storedsensor2 == 0){
      cars --;
      Serial.print("reduce number to ");
      Serial.println(cars);

```

```

    } // sensor2 read and reduced the number of cars

} //sensor2 read and reduced the number of cars

storedsensor2 = outsense;

    }

} // Car park space loop finished

    if (cars == fullpark){
outsense = digitalRead(sensor2); // see if the car is leaving the car park
if (outsense == 1){
    Serial.println("sensor2 at 3");
    if (storedsensor2 == 0){
        cars --;
        digitalWrite(Full, LOW);
        Serial.print("reduce number to ");
        Serial.println(cars);

    } // sensor2 read and reduced the number of cars

} //sensor2 read and reduced the number of cars

    if(digitalRead(Button)==1){
        digitalWrite(Full, HIGH);

    }

storedsensor2 = outsense;

    }

} //Void loop finished

```

5 Conclusion

The program is tested in the Tinkercard software, and it shows that it is working very well and able to achieve all the expectations.

The program is able to :

1. An operator sets the maximum available number of parking spaces (in the range 10 to 255.
2. Cars arriving at the entrance are initially prevented from entry by a barrier.

A red light will also be illuminated.
3. The driver of the car may request entry by pressing a push button.
4. If the car park is full then a full sign is illuminated, and entry will be denied.
5. If the car park is not full then the barrier will raise. When it has risen the red light will extinguish and a green lamp will become illuminated.
6. When the car has passed the barrier as detected by the car's presence at sensor 1 then the green lamp will extinguish, the red lamp will be illuminated, and the barrier will lower. (note there is an assumption that vehicles are clear of the barrier at this time).
7. Cars may leave the car park at any time. In doing so they activate sensor 2.

(Billing is handled by a separate number plate reading system)
8. The operator has additional controls and indications namely :-
 - a. The operator can view how many cars are parked at any time.
 - b. The operator can see the state of the barrier, red lamp, green lamp, Full lamp.
 - c. The operator can cause the barrier to raise or lower.
 - d. The operator can cause the indicators (Red lamp, Green lamp, Full Lamp) to be ON or OFF.
 - e. The operator can prevent more cars from entering the car park regardless of how many cars are actually in the car park.

(details of exactly how the operator will perform these actions is left to your design. You *may* wish to consider a Manual / Automatic mode of operation)

Also, the cars can leave the car park any time if the car park is full or not full. The lights and the barrier operate as it is expected. When the car park reaches the maximum capacity, the barrier will not let any other car to enter to the car park. The Serial Monitor shows all the messages about Rising and Lowering the barrier, the number of cars is entering into the car park and the number of available cars after leaving the car park.

6 Future work and improvements

Employing a digital screen would be a next step to improve this program.