# TRAFFIC MANAGEMENT

**Phase3-DevelopmentPart1**

**HARDWARESETUP**

1. Place the Arduino board on the breadboard.
2. Connect the cathode (shorter lead) of each LED to a current-limiting resistor.
3. Connect the anode (longer lead) of each LED to separate digital pins on the Arduino (e.g., pin 2, 3, and 4).
4. Connect the other end of the current-limiting resistors to the ground (GND) pin on the Arduino.
5. Ensure that you have the common ground between the LEDs, resistors, and Arduino

**INSTALL REQUIRED LIBRARIES**

1. Open the Arduino IDE and go to "Sketch" > "Include Library" > "Manage Libraries..."
2. Search for the library you need, click "Install," and wait for it to be installed.
3. You can install Arduino IDE with the following command

### Code Explanation

first of all, we included the timerone library. This library is used to repetitively measure a period of time in microseconds and at the end of each period, an interrupt function will be called.

We have used this library because we want to read from the sensors and control LED’s at the same time. We will have to use the delay in between the traffic signal so we can’t read from the sensors continuously. Therefore we have used this library which will allow us to call a function in which we will read from the sensors continuously and in the loop function, we will control the traffic signals.

#include<TimerOne.h>

In the setup function, we have used the Timer1.initialize(microseconds) function. This must be called before you use any of the other methods of timerone library. “Microseconds” is actually the period of time the timer takes. It is optionally to specify the timer’s period here. The default period is 1 second. Keep in mind that it breaks analogWrite() on digital pins 9 and 10.

Timer1.initialize(100000);

Timer1.attachInterrupt(softInterr) calls a function each time the timer period finishes. We have set the timer period at 100000 so our function will be called after 100 milli seconds.

Timer1.attachInterrupt(softInterr);

In the loop function it is looking if there is any vehicles under the 5 cm distance or not. If there will be vehicle, then the function to that signal will be called.

void loop()  
{  
 // If there are vehicles at signal 1  
 if(S1<t)  
 {  
 signal1Function();  
 }  
 // If there are vehicles at signal 2  
 if(S2<t)  
 {  
 signal2Function();  
 }  
 // If there are vehicles at signal 3  
 if(S3<t)  
 {  
 signal3Function();  
 }  
 // If there are vehicles at signal 4  
 if(S4<t)  
 {  
 signal4Function();  
 }  
}

‘Softinterr()’ is the interrupt function and it will called after every 100 milliseconds. In this function, we have read from the ultrasonic sensors and have calculated the distance.

void softInterr()  
{  
 // Reading from first ultrasonic sensor  
 digitalWrite(triggerpin1, LOW);   
 delayMicroseconds(2);  
 digitalWrite(triggerpin1, HIGH);   
 delayMicroseconds(10);  
 digitalWrite(triggerpin1, LOW);  
 time = pulseIn(echopin1, HIGH);   
 S1= time\*0.034/2;

### Code

The Arduino code for density based traffic light controller using Arduino is as follows

#include<TimerOne.h>  
int signal1[] = {23, 25, 27};  
int signal2[] = {46, 48, 50};  
int signal3[] = {13, 12, 11};  
int signal4[] = {10, 9, 8};  
int redDelay = 5000;  
int yellowDelay = 2000;  
volatile int triggerpin1 = 31;   
volatile int echopin1 = 29;   
volatile int triggerpin2 = 44;   
volatile int echopin2 = 42;   
volatile int triggerpin3 = 7;   
volatile int echopin3 = 6;   
volatile int triggerpin4 = 5;   
volatile int echopin4 = 4;   
volatile long time; // Variable for storing the time traveled  
volatile int S1, S2, S3, S4; // Variables for storing the distance covered  
int t = 5; // distance under which it will look for vehicles.  
void setup(){

**SEND DATA TO THE SERVER**: Adapt the script to send data to your cloud server or mobile app server. You can use the same HTTP request method as in the previous example, or you can use a different protocol, such as MQTT, for real-time updates.

**SERVER-SIDE IMPLEMENTATION**: Implement an endpoint on your server to receive the data. The server can analyze the distance data to determine parking space occupancy.

**SECURITY, ERROR HANDLING, LOGGING, AND AUTOMATION**: Follow the same security, error handling, logging, and automation guidelines as mentioned in the previous example.

**DATA ANALYSIS**: On the server-side, implement logic to analyze the distance data to determine if a parking space is occupied or vacant. You can set a threshold distance value to make this determination.

**SECURITY**: Ensure your data transfer is secure. For example, use HTTPS for web requests and consider implementing authentication and authorization mechanisms.

**ERROR HANDLING**: Implement proper error handling to account for network issues and other potential problems.

**LOGGING AND MONITORING**: Implement logging and monitoring to track the status of data collection and transmission.