IoT Project report

Energy conservation through Smart Management of Street Lights

Team members:

Avinash Prabhu - 2018102027	Sampath Kumar - 2018102026
Karan Mirakhor - 2018102034	Rahul Kashyap - 2018102037
Dipanwita G - 2018112004	Nihar P.V.G.H - 2018102039

The link to the codes is placed at the end of the report. The link to the videos displayed in the PPt is also placed at the end of the report.

Sensors and actuators:

In this project, choosing the right sensor was one of the most challenging tasks. It is important to turn on the lights when there is motion or when an object comes into the vicinity of the streetlight. The sensor should have the following features.

- It should have a range of 8-10 meters as the sensor should be able to detect any object along the width of the road.
- It should be able to detect objects moving at a variety of speeds. Typical sensors with a long-range mostly fall short in this aspect, i.e, they cannot detect objects moving faster than 5-10 kmph.

LiDAR sensors satisfy both the above mentioned criteria which is why we chose the **TF-MINI LiDAR Sensor** for our project.

TF-MINI LIDAR Sensor

It has the detection range of 0.3-12m with acceptance angle of 2.3 degrees. This is the one we found in the budget which satisfies the requirements and also interfaces with arduino.

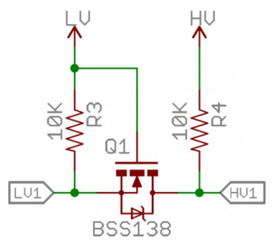
The sensor emits the laser which reflects off a surface and comes back. The sensor then measures the time of flight of the beam which gives us the distance between the sensor and the object . Even with 10% reflectivity it can detect up to 5m accurately. This was tested by us on the main road outside the campus and the sensor successfully detected any moving vehicle across the entire width of the road.

This module outputs the serial data using UART communication. It operates at 115200 baud rate. The sensor outputs 9 bytes of data to the controller.

- The first two bytes are headers which help to check if the data transmission has started.
- The third and fourth bytes contain information about distance. The third byte consists of the lower 8 bits and the fourth byte consists of the remaining 8 bits in the bit representation of the distance value.

  Distance = (fourth_byte)*256 + (third_byte)

- 5th and 6th bytes give information about signal strength. The procedure used for calculating signal strength is similar to the one used for calculating distance.
- 7th and 8th bytes are reserved and ensure signal quality degree.
- The 9th bit has a lot more importance. It is a parity bit which helps in finding error in the transmitted information. It does so by checking the sum of the bits. If the sum of all the 8 bit is not equal to the 9th bit then there is an error in the received data..



Bi-directional logic converter:

The communication between the microcontroller and the sensor is established through TX and RX pins of the sensors. The main problem regarding the sensor is that it operates at 3.3V logic and our microcontroller operates at 5V logic. As there is a difference in the logic it will be difficult to transmit the data and most of the information received would be false. This is because the threshold for 0 and 1 for different logics is different. So to overcome this, we used a bi-directional logic converter which helps in converting one logic to another. The circuit uses an n-channel MOSFET to convert the low voltage logic level to a high voltage logic level. A simple logic level converter can also be built using resistive voltage dividers but it will introduce voltage loss. MOSFET or transistor-based logic level converters are professional, reliable, and safer to integrate.

HC-SR04 ultrasonic sensor

Due to limitations in budget, we could not buy the required number of LiDAR modules, thus, to demonstrate the idea we used HC-SR04 ultrasonic sensor module to replace the LiDAR module at one of the streetlights.

The basic working principle of the ultrasonic sensor is based on the Doppler effect. But the detection is limited to 4 metres and it can only detect low speeds. In order to generate the ultrasonic sound, we need to set trig on for at least 10 microsec. It will then generate 8 ultrasonic pulses at 40KHZ frequency. Then the echo will be on until we receive the signal back. The distance is calculated by ((speed of sound)*(time of echo/2)).

The microcontroller keeps track of the number of objects passed through it. The sensor is placed with the module, when the sensor detects an object the microcontroller increases the count.

How the sensor data used to count the objects

The sensor by default gives the high value i.e. infinity if the object is not detected. If something passes through the sensor it gives the distance between the sensor and the object. So if the distance value changes then we can say that an object has crossed the sensor. But in real life objects have length like car, bus, etc. So our sensor keeps detecting the object until the complete length of the object is passed through it. In that case instead of detecting it as one object we may end up in counting it as many objects have crossed the sensor. To overcome this, whenever the distance is less than infinity it goes into a loop, it exits the loop only when the count goes back to infinity. The count increases only when it exits the loop. In this way even if the object has length we can count it.

Relay

This is used to control the street lights based on the logic levels received from the microcontroller. We have used a 3.3V logic relay module which remains unconnected from the mains in NC (normally closed) configuration when the input to the module is HIGH (5V) from the microcontroller and it connects the street light to the mains in NO (normally open) configuration when the input to the module is LOW(0V) from the microcontroller.

LoRa Communication:

In our project we need communication between the masters of each set of street lights to ensure that our logic is implemented properly. Since we need a range of around 2-3 km to ensure communication within street lights and the server and we're sending only about 50 kbps, LoRa communication is perfect for this use case.
We mainly needed to interface our LoRa module (HPD13A SX1276) with an Arduino Board in the case of street lights and an ESP8266 board in the case of the gateway.

Interfacing LoRa and Arduino

The LoRa Module uses SPI communication and has the following pins to be connected (We have included a picture of the pins in the slides) : Ground, 3.3V VCC, Digital Input-Output Pin 0, MISO, MOSI, RESET, SCK and an EN(enable)/NSS pin. We now need to connect these pins to the appropriate pins of Arduino. In Arduino, pin 13 = SCK, pin 12 = MISO, pin 11 = MOSI, pin 10 = NSS. The VCC and ground pins are connected to the appropriate pics of the arduino along with the above mentioned pins. The remaining pins are RESET and DIO pins. We connected RST to pin 9 and DIO0 to pin 2. We then have to include this piece of code
" LoRa.setPins(ss, rst, dio0)" where ss = 10, rst = 0, dio0 = 2 as explained above. On completing all the above steps we are done with interfacing LoRa with Arduino.

Interfacing LoRa and ESP8266

We use the same LoRa module mentioned above so the pins required remain the same. In ESP8266, pin D5 = SCK, pin D6 = MISO, pin D7 = MOSI, pin 10. The VCC and ground pins are connected to the appropriate pics of the arduino along with the above mentioned pins.The remaining pins are RESET, DIO and NSS pins. We connected RST to GPIO16 (D0), DIO0 to GPIO10 (SD3) and NSS to GPIO15 (D8). We then have to include this piece of code " LoRa.setPins(ss, rst, dio0)" where ss = 15, rst = 16, dio0 = 10 as explained above. On completing all the above steps we are done with interfacing LoRa with ESP8266.

Arduino IDE coding for LoRa

Once we are done setting up all the connections, we can start coding. For arduino, we need the LoRa.h library and for ESP8266, we need LoRa.h and SPI.h to enable the LoRa modules. We then set up the LoRa module to operate at the required frequency in the void setup(). After completing these steps, we can start implementing our logic. There are various commands made available to us through the LoRa.h library to enable communication. A few examples which we used are - **LoRa.begin()** [This will start up the LoRa module at the required frequency], **LoRa.available()** [This will put our LoRa module into listening mode a], **LoRa.write**() [This is used to transmit the required data.], **LoRa.read**() [This will read the incoming data], **LoRa.parsePacket()** [This divides the received data into the required fields.] and more.

Differentiating between LoRa modules

In our project, we have 4 different LoRa modules. When we transmit from a particular LoRa module, all the remaining LoRa will receive the data sent. However, this is undesirable. To overcome this, we can assign addresses to each module as described in the 'Basic Working Principle' section. In the transmitted data, we add the address of the node we have to transmit to. While receiving, we used a simple IF condition to check if that particular node should process the data or just ignore.

To keep track of the modules they need to connect to a server, this is done through a **LoRa Gateway**.

Gateway:

The LoRa gateway consists of an ESP8266 module and a LoRa module and is installed separately indoors for maintenance and control. The ESP8266 module communicates to a cloud server:- ThingSpeak via WiFi and LoRa module is to receive data from LoRa modules installed at the streetlights. Since the gateway is placed indoors, WiFi connection should not be a problem. The output graphs are obtained on the dashboard of the ThingSpeak channel on ThingSpeak.com which can be analyzed for monitoring the performance of our system. We have included the link for these graphs in the powerpoint.

Connecting ESP8266 to WiFi

We use the Arduino library **ESP8266WiFi.h** to connect to a WiFi connection (for testing purposes we have used a mobile hotspot) via the WiFi SSID and password, which is the same as that used to set up the hotspot. The code checks for WiFi connection using **WiFi.status() != WL_CONNECTED** and is put on delay till the connection is established.

Pushing data to ThingSpeak server:

Setting up the server

We have used a free account on ThingSpeak.com that allows us to set up our channel. We can make use of the Channel Number and the WriteAPIKey associated with the channel to access it.

Connecting to the server

Once WiFi is connected, we use the Arduino library **ThingSpeak.h** to connect to our ThingSpeak channel using Channel Number and the WriteAPIKey. We set three fields in our channel, corresponding to the three sender modules, which are the master nodes in each set. We have assigned address '0xAA','0xBB' and '0xCC' to the three master modules which are linked to Fields 1,2 and 3 respectively and address 'DD' to the gateway. Since ThingSpeak can support up to 8 data fields, we can send multiple values to ThingSpeak using ***ThingSpeak.setField(#field, value)*** for each value. We test the connection to the server by sending a default value of 0 to Field1.

Sending data to the fields

While receiving data from the LoRa module we check if the recipient address matches that of the Gateway for efficient communication. We also verify if the incoming message length matches the expected LoRa message length (this is also sent along with the message packet found using length() inbuilt function). Once the acknowledgment is done on the receiver side, we check for the sender address associated with the LoRa incoming data and write to the corresponding Field on the ThingSpeak channel using ***ThingSpeak.writeField(myChannelNumber, #field,value, myWriteAPIKey)***

Delay associated with pushing to the server

ThingSpeak free accounts permit a minimum of 15 seconds delay between two successive data writes to the channel. So we have placed a delay of 15 seconds after every Write operation to the ThingSpeak server to account for this limitation. Since we expect our system to be in operation over a large period of time and the gateway is primarily used to maintain the modules, this delay is not expected to cause problems in the operation of our system.

Logic to control the street lights :

After having divided the street lights into different groups with each group's master installed with the sensor, LoRa module and a microprocessor we now move on to the logic used behind in controlling them.

Notations and assumptions

Herewith, any reference to a road should be taken as for a one-way road only, the master node of the first group of street lights i.e. the group at the beginning of a road is referred to as **node A,** the master node of a group lying in between the beginning and the end of the road is referred to as **node B** and master node of the group at the end of the road is referred to as **node C.** Without loss of generality, we assume that there are only three groups on the road with master **node A, node B and node C.** The LoRa addresses of the master nodes in hexadecimal are **0xAA, 0xBB, and 0xCC** respectively. There is also a **gateway** connected to these master nodes which monitors the vehicle count data coming from them and the LoRa address of the **gateway** in hexadecimal is **0xDD.**

Node A working: This node and its corresponding group always remain on as it is at the beginning of the road and there are no means to control it. When a vehicle passes in front of node A, the LIDAR sensor on it, registers the detected vehicle by incrementing the count A by 1.This count is then sent to **node B** with LoRa address **0xBB** and the gateway with LoRa address as **0xDD**. So, this node is used only for registering a vehicle entering the road and notifying the following node and the gateway about it.

Node B working: There are two cases in which this node and its corresponding group of street lights are switched on, they are:

Case 1: Vehicle detected at node A

In this case, on receiving the data for change in vehicle count of node A, the lights are turned on.
Also, a flag named as **car_present** is set to HIGH and a flag named as **final_count** is set to LOW on receiving change in count of **node A** to indicate the presence of a vehicle between the **node A** and **node B** and a timer named as **timer_1** counts down the maximum duration for which a vehicle can be between **node A** and **node B (**the use of which will be explained later in this part).

Case 2: Vehicle not detected at node A but detected at node B

In this case this node is not intimidated by the previous node about the incoming vehicle(implies that the flag **car_present** is still LOW) but the sensor at this node detects a vehicle passing by, and hence it increments the count B by 1 and sends this data to the **node A, node C** and **the gateway.** When there is an increment in this node's count, the node and its corresponding group of street lights are turned on along with starting off a **timer_3** which counts down the maximum duration for which a vehicle can be between **node B** and **node C** and a flag named as **final_count** is set HIGH to indicate present of vehicle between **node B** and **node C.**

Now coming to turning off the street lights of **node B** and its corresponding group, we have three main cases, they are:

Case 1: Vehicle detected properly at all three nodes i.e. at node A, node B and node C

This is the most trivial case in which the **node A** detects the incoming vehicle, increments the count A by 1 and sends this data to **node B** and the gateway. The flag **car_present** is set to HIGH, **final_count** is set to LOW and the **timer_1** countdown starts; to indicate the presence of vehicle between the **node A** and **node B.** The **timer_1** has sufficiently large time value to account for a moving vehicle in between **node A** and **node B.** As we assume a moving vehicle between the two nodes therefore, it reaches the **node B** before the countdown for **timer_1** runs out. Now, when the same vehicle is detected at **node B** we increment count B by 1 and send this data to **node A, node C** and **gateway** and we set the flag **car_present** to LOW, to indicate the vehicle has passed between **node A** and **node B.** Also, the flag **final_count** is set to HIGH in order to indicate the presence of vehicle between **node B** and **node C** only and the **timer_3** countdown starts, which accounts for the maximum time a moving vehicle can be between **node B** and **node C.** So when the moving vehicle detected at **node B** reaches **node C** it is detected there without the **timer_3** running out. On detection at **node C,** the count C is incremented by 1 and then this data is sent to **node B** and **gateway**. When this increment in count C is received by **node B,** the values of count C and count B are compared and if they are equal, a flag **equal_count** set to HIGH which is always LOW in all other cases, indicating the counter in **node B** and **node C** has become equal; hence the street lights of **node B** and its corresponding group can go off after **timer_2** reaches a small value. The buffer **timer_2** is kept in order to prevent frequent switching on and off of street lights and it basically waits to detect some other vehicle just after a previous vehicle has gone through.

Case 2: Vehicle detected properly only at two of the three nodes

There are three subcases possible:

(i) Vehicle detected at node A and node B only

In this subcase,**node A** detects the incoming vehicle, increments the count A by 1 and sends this data to **node B** and the gateway. The flag **car_present** is set to HIGH, **final_count** is set to LOW and the **timer_1** countdown starts; to indicate the presence of vehicle between the **node A** and **node B.** Now, when this moving vehicle reaches **node B,** it is detected and we increment the count B by 1 and send this data to **node A, node C** and **gateway** and we set the flag **car_present** to LOW, the flag **final_count** is set to HIGH and the **timer_3** countdown starts. Since the vehicle goes undetected at **node C,** there is no chance for the count C and count B to become equal. Hence based on our assumption that **timer_3** accounts for the maximum time a moving vehicle can be between **node B** and **node C,** the street lights of **node B** and and its corresponding group are turned off after **timer_3** reaches its maximum value and also the flag **final_count** is set to LOW to indicate that the vehicle has passed through the path between **node B** and **node C**.

(ii) Vehicle detected at node A and node C only

In this subcase, **node A** detects the incoming vehicle, increments the count A by 1 and sends this data to **node B** and the gateway. The flag **car_present** is set to HIGH, **final_count** is set to LOW and the **timer_1** countdown starts; to indicate the presence of vehicle between the **node A** and **node B.** Now, when this moving vehicle reaches **node B,** it is not detected there for some reason, maybe some other car moving parallel to it or the incremented count B data sent by it could not reach **node C** so there is no chance for the count C and count B to become equal**.** In such a scenario we have **timer_1** which accounts for the maximum time a vehicle can be between **node A** and **node B,** it would reach its maximum value and the flag **car_present** is set back to LOW. Now, as the flag **car_present** is LOW and the count B and count C are unequal, if the flag **final_count** is also LOW, we set the flag **final_count** as HIGH and start the **timer_3** counter which will eventually reach its maximum value and if **final_count** has remained HIGH throughout the process indicating that there was vehicle probably in between **node B** and **node C** only. Hence, the street lights of **node B** and its corresponding group will be turned off, once the **timer_3** reaches its maximum value in the above mentioned case.

(iii) Vehicle detected at node B and node C only

In this subcase, **node B** is not intimidated by the previous **node A** about the incoming vehicle(implies that the flag **car_present** is still LOW). **Node B** detects the vehicle passing by, and hence it increments the count B by 1 and sends this data to the **node A, node C** and **the gateway** and the flag **final_count** is set to HIGH in order to indicate the presence of vehicle between **node B** and **node C** only and the **timer_3** countdown starts. As we are talking about a moving vehicle, it would reach **node C** without the **timer_3** running out. On detection at **node C,** the count C is incremented by 1 and then this data is sent to **node B** and **gateway**. This increment in **node C** is received by **node B** and if the count B and count C are equal, a flag **equal_count** is set to HIGH and **timer_2** is started and the street lights of **node B** and its corresponding group can go off after **timer_2** reaches a small value if **equal_count** remains HIGH.

Case 3: Vehicle detected properly only at node B

In this subcase, **node B** is not intimidated by the previous **node A** about the incoming vehicle(implies that the flag **car_present** is still LOW). **Node B** detects the vehicle passing by, and hence it increments the count B by 1 and sends this data to the **node A, node C** and **the gateway** and the flag **final_count** is set to HIGH in order to indicate the presence of vehicle between **node B** and **node C** only and the **timer_3** countdown starts. Since the vehicle goes undetected at **node C,** there is no chance for the count C and count B to become equal. Hence based on our assumption that **timer_3** accounts for the maximum time a moving vehicle can be between **node B** and **node C,** the street lights of **node B** and and its corresponding group are turned off after **timer_3** reaches its maximum value and also the flag **final_count** is set to LOW.

Node C working: This node and its corresponding group of streetlights are turned on only when there is an increment in count registered in **node B.** To turn off this set of lights either the count B and count C should be equal or the maximum time for the car to be in this set of lights should elapse. The lights of this set are not turned off immediately after the count becomes equal but a small relaxation time is given to avoid frequent toggling of lights. This is accomplished using **timer_1** which counts up to a small value of time and **equal_count** flag, which goes HIGH only when the count becomes equal, and remains HIGH till the **timer_1** counts till the specified value. If the count is not equal; even in such a case to turn off the lights, we use the **timer_2** which is set to count to a preset value of time that accounts for the maximum time a car could be in the last set of the street lights on the road. So, if the count remains unequal and there is no increment in the value of count B, **timer_2** will count to its maximum value and this set of lights would be turned off after that.

Codes: Please click here for the final codes
Videos : Please click here for the Videos

****Thank you****