SVKM's NMIMS Mukesh Patel School of Technology Management and Engineering, Mumbai, India

Team Technovators

Telemetry Report

Prof. Sawankumar Naik
Sourabh Saptarshi
Tarang Patel
Mit Vaidya
Ishan Sonavane
Medha Jain

Telemetry System

The Telemetry System is an integral part of the Vehicle. It helps us focus on the short coming and also supports and guides us, to prevent damage to the vehicle and to ensure higher safety of drivers.

The information gathered from another planet, the pictures and the ability of the riders' performance is the key goal of any mission.
Thus, it is necessary to gather all the data possible from there so that it can be analysed and studied. Real time data transmission to the base station is an icing on the cake.

The telemetry system is divided into 2 parts:

1. The real time sensor monitoring system
2. The real time video and 2 way audio system

1. Sensor System

The main aim behind designing this part is to ensure monitoring of various parameters of the rover, the drivers and the environment of the rover as well. This circuitry collects various important and necessary information and saves it for research.

There are a total of 16 sensors which we have implemented:

a) **GPS:** Provides location information in all whether conditions, in and around the Earth.
b) **Accelerometer**: Measures proper acceleration in space. It is the key component of the inertial navigation for aircrafts.
c) **Gyroscope**: uses Earth's gravity to help determine orientation. It can measure the rate of rotation around a particular axis.
d) **Magnetometer**: It measures the magnetization, strength of a magnetic material. It also measures the direction of the magnetic field at a point in space. It has key role in space missions as it can be used to measure both the magnitude and direction of the magnetic field of a planet or moon.
e) **Orientation**: It detects if the rover is tilted from the reference position. It detects tilts and degrees from magnetic North. It is very useful in space applications as it provides us with the information of the exact orientation from the reference point.

f) **Gravity sensor**: Measures the force of gravity.
g) **Rotation Vector**: Measures the orientation.
h) **Acceleration Linear**: Measures acceleration excluding force of gravity.
i) **Audio**: Provides information of how much noise is being produced.
j) **Light**: Measure the light level.
k) **Proximity**: It detects the presence of nearby objects without physical contact
l) **Pressure**: Measures the air pressure level.
m) **Temperature**: Measures the current Temperature
n) **Humidity**: Measure the humidity level.
o) **Time**: Measures the current time. Useful if the planet's time changes over the period of years.
p) **Date**: Also useful for the same purpose as above.

The android phone sensors were used to determine the various 16 sensor parameters. After the values were determined, they were sent to the ARDUINO DUE with Ethernet Shield attached on it via Bluetooth. Bluetooth HC-05 was used for this purpose.

We sent the data to a network adapter, which acted as a client here, via a switch. The network adapter transfers the data to the destination website through internet via cloud.

The data sent to the website is calibrated, managed and analysed to give meaningful information and helps us to clearly know the situation on the rover. All the sensor data is converted to REAL TIME graphs giving us the exact situation every second with a minimal lag.

The graphs can be monitored manually and can be analysed in various ways by changing the timescale, the x-axis gap, the y-axis gap.

Also, the average, median, sum etc parameters and results can also be found out and the graph can be updated in real time in just one click.

This interface of the website helps the people in the base station to study the outcomes much easier and faster.

Also, we have real time values of all the sensors log saved on a web server.

Description of the sensors:

Sr. no.	Sensor	Description	Uses
1	**GPS**	Provides location information in all whether conditions, in and around the Earth.	Acquiring location information
2	**Accelerometer**	Measures the acceleration force in m/s^2that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.). Inertial Navigation
3	**Gyroscope**	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
4	**Magnetometer**	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass. Finding magnitude and direction of the magnetic field of a planet or moon.
5	**Orientation**	Measures degrees of rotation that a device makes around all three physical axes (x, y, z).	Determining Rover position.
6	**Gravity**	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Gravity information of another planet.
7	**Rotation Vector**	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Orientation and rotation detection.
8	**Linear Acceleration**	Measures the acceleration force in m/s^2that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
9	**Light**	Measures the ambient light level (illumination) in lx.	To check the luminous level at different times.
10	**Audio**	Measure the Audio level	To ensure safety and to check unusual noise, if any.
11	**Proximity**	Measures the proximity of an object in cm relative to the view screen of a device.	To measure the distance of objects nearby.
12	**Pressure**	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
13	**Temperature**	Measures the temperature in degrees Celsius (°C).	Monitoring temperatures.
14	**Humidity**	Measures the relative ambient humidity in percent (%).	Monitoring dew point, absolute, and relative humidity.
15	**Time**	Shows the time with respect to a reference.	Monitoring the time and date changes
16	**Date**	Shows the date with respect to a reference.	Monitoring the time and date changes

2. Video and Audio Transmission

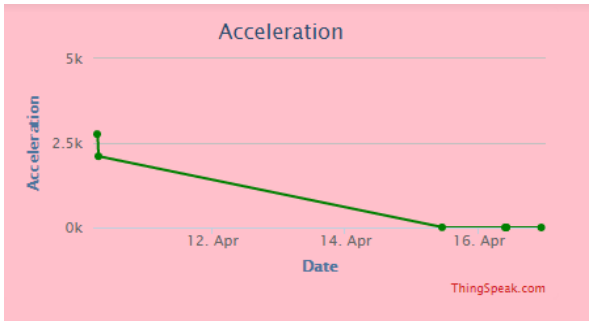
To ensure safety and efficiency during the race, we implemented a video transmission system along with 2 way audio system so that the drivers in the buggy can remain in touch with the base station at all times without interference.

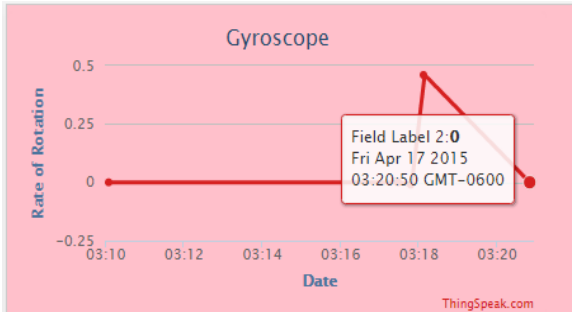
We put two cameras, one to capture images and video of the surface there and the other one near to the driver so that the base station are always able to see the faces of the riders and are able to guide and support them in case of any emergency ensuring total safety.

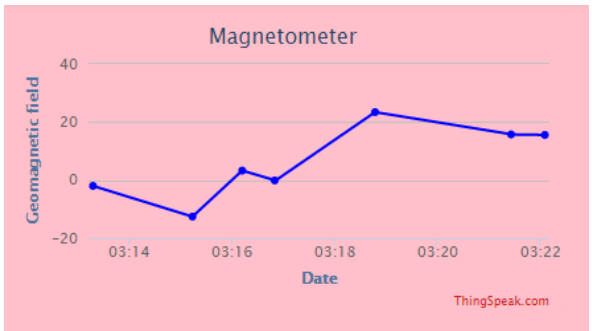
The cameras used are Wi-Fi wireless IP cameras also with night vision and 360^{o} angle view which can be monitored manually from the base station at all times. These cameras are also inclusive of high quality microphone and both way audio transmission. Even the slightest disturbance and noise in the rover can be detected at the base station.

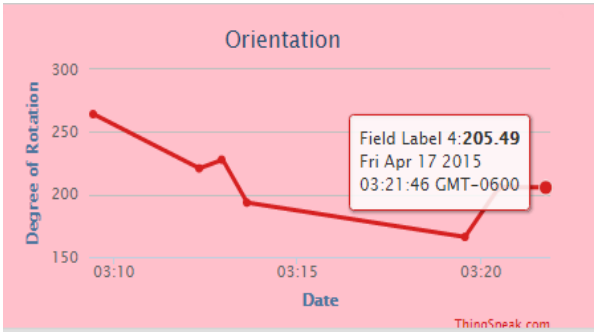
The cameras are wirelessly connected to the base station but internally wired in the rover via network adapter so as to ensure the robustness. This was deliberately done so that the transmission it as accurate as possible with minimal interference and connection is not lost at any times.

Example of Real Time Graphs:

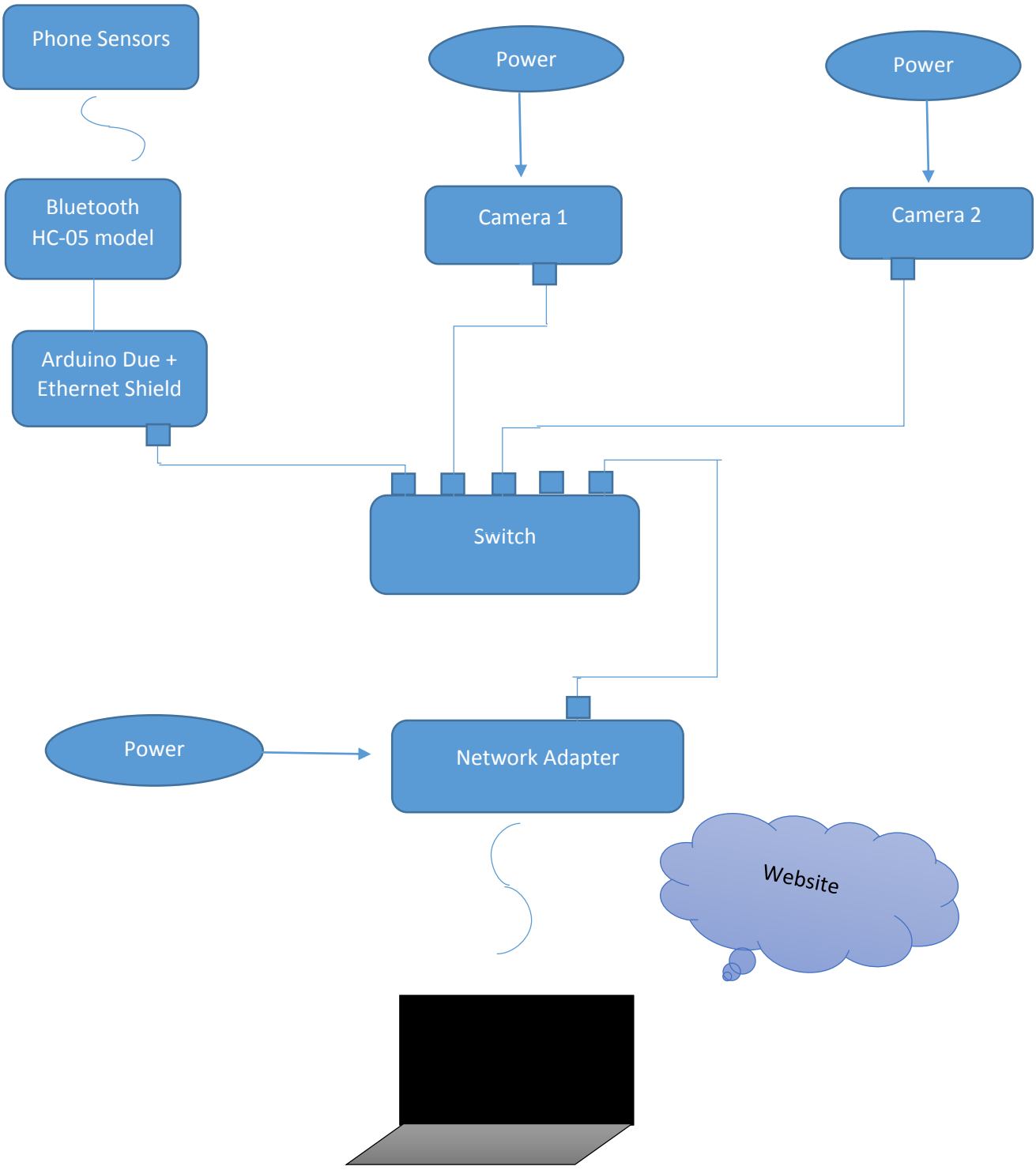






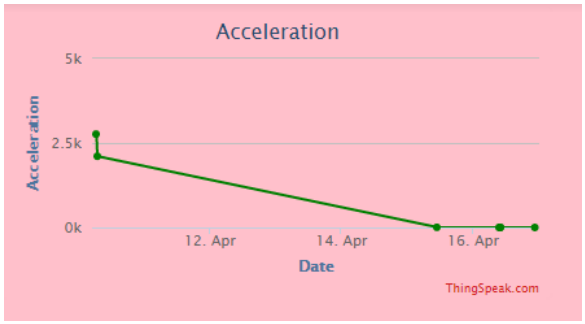


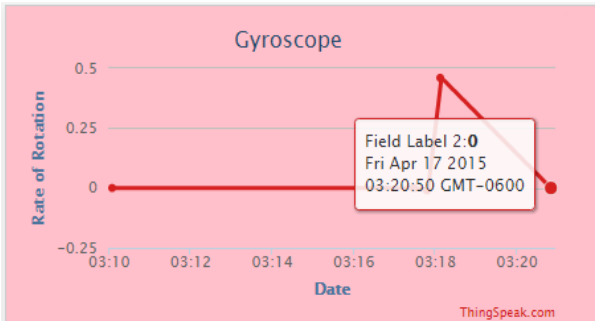
Block diagram of the circuit Assembly

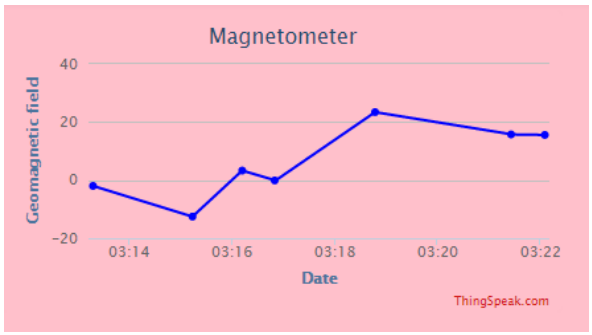


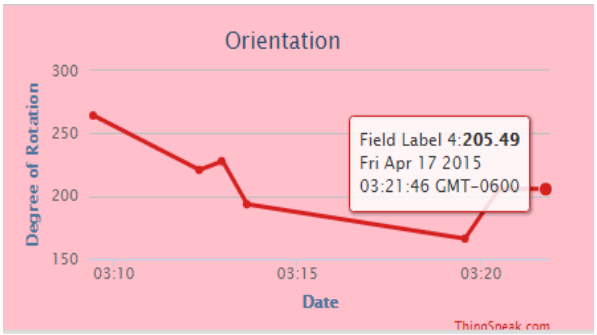
Base station

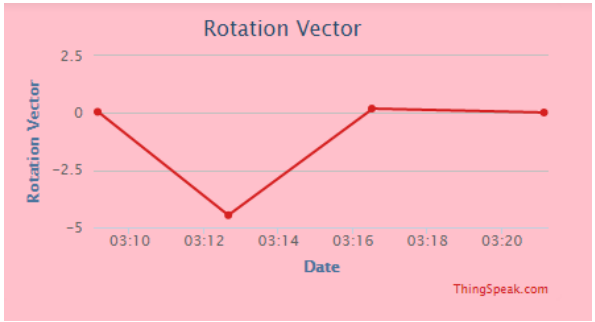
Real Time Graphs:

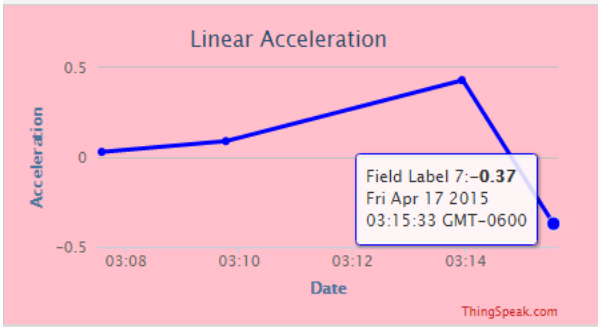




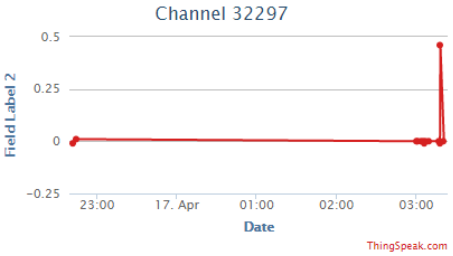


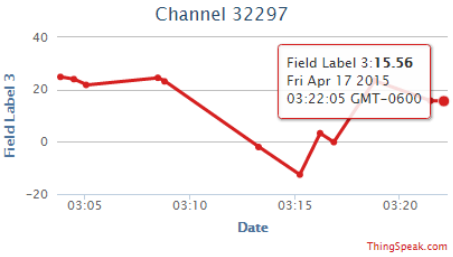




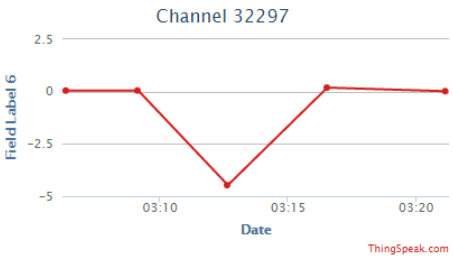


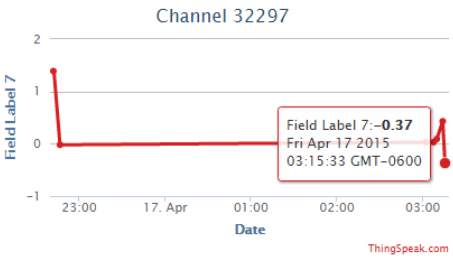
NASA TELEMETRY





NASA TELEMETRY





```
//#include <stdlib.h>
#define START_CMD_CHAR '>'
#define END_CMD_CHAR '\n'
#define DIV_CMD_CHAR ','

#include <SPI.h>
#include <Ethernet.h>

#define DEBUG 1 // Set to 0 if you don't want serial output of sensor data
//bluetooth values
String inText;
float value0=0.0;
float value1=0.0;
float value2=0.0;
int sensorType;
String analogValue0;
// Local Network Settings
byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDD, 0x02 }; // Must be unique on local network

// ThingSpeak Settings
char thingSpeakAddress[] = "api.thingspeak.com";
String writeAPIKey = "9BRUNK3O4L1Q8EHV";
const int updateThingSpeakInterval = 1000;      // Time interval in milliseconds to update ThingSpeak (number of seconds * 1000 = interval)

// Variable Setup
long lastConnectionTime = 0;
boolean lastConnected = false;
int failedCounter = 0;
```

```
// Initialize Arduino Ethernet Client
EthernetClient client;

void setup()
{
  // Start Serial for debugging on the Serial Monitor
  Serial.begin(9600);

  // Start Ethernet on Arduino
startEthernet();

}

void loop()
{
  int inCommand = 0;
 // int sensorType = 0;
  unsigned long logCount = 0L;

  char getChar = ' ';  //read serial

  // wait for incoming data
 if (Serial.available() < 1) return; // if serial empty, return to loop().

  // parse incoming command start flag
 getChar = Serial.read();
 if (getChar != START_CMD_CHAR) return; // if no command start flag, return to loop().

  // parse incoming pin# and value
   sensorType = Serial.parseInt(); // read sensor typr
```

```
 // logCount = Serial.parseInt();  // read total logged sensor readings
   value0 = Serial.parseFloat(); // 1st sensor value
   value1 = Serial.parseFloat();
   value2 = Serial.parseFloat();
/*   if(DEBUG)
   {
  Serial.print("value0 =");
  Serial.println(value0);
  Serial.print("value1 =");
  Serial.println(value1);
  Serial.print("value2 =");
  Serial.println(value2);
  Serial.print("--------------------");

  delay(10);
   }*/
  // analogValue0+=String(int(value0))+ "."+String(getDecimal(value0)); //combining both whole and
decimal part in string with a full                                          //stop between them
 // analogValue0 = String(value0, DEC);

   char buffer[10];
   String tem=dtostrf(value1,5,2,buffer);
   analogValue0=(tem);
   //sprintf(fmt, "%%%d.%df", width, prec);
  //  analogValue0=dtostrf(value0,5,5,t_buffer);

  }

void serialEvent()
{
//  while(Serial.available())
//  {
```

```
 // Serial.setTimeout(0.1);
 // value1 = Serial.parseFloat();  // 2rd sensor value if exists
 //value2 = Serial.parseFloat();  // 3rd sensor value if exists
 // Read value from Analog Input Pin 0
 // String analogValue0 = String(value0, DEC);

  // Print Update Response to Serial Monitor
if (client.available())
 {
   char c = client.read();
 Serial.print(c);
 }

 // Disconnect from ThingSpeak
 if (!client.connected() && lastConnected)
 {
   Serial.println("...disconnected");
   Serial.println();

   client.stop();
 }

 // Update ThingSpeak
 if(!client.connected() && (millis() - lastConnectionTime > updateThingSpeakInterval))
 {
   if(sensorType==2)//magnetometer
   updateThingSpeak("field1="+analogValue0);

   if(sensorType==4)//gyroscope
   updateThingSpeak("field2="+analogValue0);
```

```
   if(sensorType==10)//Linear Acceleration
   updateThingSpeak("field3="+analogValue0);

   if(sensorType==3)//Orientation
   updateThingSpeak("field4="+analogValue0);

   if(sensorType==9)//Gravity
   updateThingSpeak("field5="+analogValue0);

   if(sensorType==11)//Rotation Vector
   updateThingSpeak("field6="+analogValue0);

   if(sensorType==1)//Acccelerometer
   updateThingSpeak("field7="+analogValue0);

   if(sensorType==97)//Audio
   updateThingSpeak("field8="+analogValue0);



  }

  // Check if Arduino Ethernet needs to be restarted
  if (failedCounter > 3 ) {startEthernet();}

  lastConnected = client.connected();
//}
}
  char *dtostrf (float val, signed char width, unsigned char prec, char *sout) {
  char fmt[10];
  sprintf(fmt, "%%%d.%df", width, prec);
```

```
 sprintf(sout, fmt, val);
 return sout;
}
//function to extract decimal part of float
void updateThingSpeak(String tsData)
{
 if (client.connect(thingSpeakAddress, 80))
 {
  client.print("POST /update HTTP/1.1\n");
  client.print("Host: api.thingspeak.com\n");
  client.print("Connection: close\n");
  client.print("X-THINGSPEAKAPIKEY: "+writeAPIKey+"\n");
  client.print("Content-Type: application/x-www-form-urlencoded\n");
  client.print("Content-Length: ");
  client.print(tsData.length());
  client.print("\n\n");

 client.print(tsData);

   lastConnectionTime = millis();

   if (client.connected())
   {
     Serial.println("Connecting to ThingSpeak...");
     Serial.println();

     failedCounter = 0;
   }
   else
   {
     failedCounter++;
```

```
    Serial.println("Connection to ThingSpeak failed ("+String(failedCounter, DEC)+")");
    Serial.println();
   }

  }
  else
  {
   failedCounter++;

   Serial.println("Connection to ThingSpeak Failed ("+String(failedCounter, DEC)+")");
   Serial.println();

   lastConnectionTime = millis();
  }
}

void startEthernet()
{

  client.stop();

  Serial.println("Connecting Arduino to network...");
  Serial.println();

  delay(1000);

  // Connect to network amd obtain an IP address using DHCP
  if (Ethernet.begin(mac) == 0)
  {
   Serial.println("DHCP Failed, reset Arduino to try again");
```

```
    Serial.println();
  }
  else
  {
    Serial.println("Arduino connected to network using DHCP");
    Serial.println();
  }

  delay(1000);
}
```