NEED OF ELECTRONICS

This year we have moved ahead in the field of electronics with the installing of several on board and off board tools and equipment to improve the safety, driver’s comfort, agronomics and aesthetics of our vehicle **via validations**. After discussing about the several problems that the team faces during manufacturing, testing phase, dynamic event tests and finally the ones occurring on the endurance track. We start analyzing the major and minor possibilities of innovation in them (majorly electronics solutions), a list of all the problem statements were prepared out of which some were solved this year and other were left with the purpose of research problem for next year.

The list is described below:-  
***SOLVED PROBLEMS:-***

1. Damper travel
2. Wheel alignment
3. Vibration sensor
4. GPS
5. Data acquisition system
6. Engine rpm
7. Auto-gyro kill switch
8. CVT temperature
9. CVT dyno
10. Vehicle speed
11. CVT cooling
12. Data transmission
13. Pipe cutting matrix
14. Miscellaneous (steering angle calculation, cables and connectors research data )

***INITIATED, UNSOLVED AND LEFT FOR FUTURE RESEARCH :-***

1. Working on simulation various software options like MATLAB, Proteus.
2. Force calculation on various parts like hub, knuckle, tie rod, hem joints, ball joints etc. By using strain gauge.
3. COG calculation of any object with the use of gyroscope and accelerometer.
4. Setup for parallel shaft arrangement and c2c dist. Calculation for power transmission department.
5. A complete self-analyzed system for suspension department providing them the minned data for damper travel simultaneously with the ground clearance from various positions.
6. Pressure calculation for brakes department by using either pressure sensor or ultrasonic flow meters etc.
7. Clamping force calculation for brakes department by the use piezoelectric.
8. Possibility for improvement in pipe cutting matrix code for chassis department.
9. Improved setup idea for wheel alignment system.
10. Work towards data transmission.
11. Work towards engine emissions.
12. Use of quality grade cables and connectors.
13. More ways exploration for CVT cooling.
14. More rigid setup for CVT dyno.
15. Steering effort calculation.

(NOTE: The solved problems are further categorized as per validation setup or the ones installed on board.)

|  |  |  |
| --- | --- | --- |
| **S. NO.** | **VALIDATION SETUP** | **Page No.** |
| *1* | *DAMPER TRAVEL* | *4-5* |
| *2* | *WHEEL ALLIGNMEMT* | *6-7* |
| *3* | *VIBRATION SENSOR* | *8-9* |
| *4* | *DATA ACQUISATION SYSTEM* | 10-11 |
| *5* | *ENGINE CVT RPM* | *12-13* |
| *6* | *CVT TEMPERATURE MONITORING* | *14-15* |
| *7* | *CVT DYNO* | *16-18* |
| *8* | *VEHICLE SPEED* | *19-20* |
| *9* | *GYRO KILL* | *21-24* |

CONTENTS

DETAILED DESIGN REPORT

**DAMPER TRAVEL**

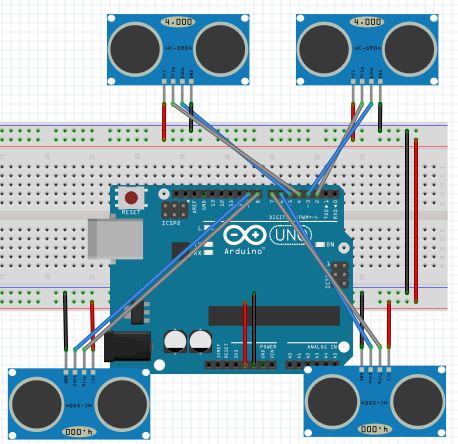
***AIM / PROBLEM STATEMENT: -*** To perform iterations determining the travel in dampers so as to study the behavior of buggy on different terrains for improvement in final overall performance.

***COMPONENTS / APPARATUS REQUIRED :-*** Ultrasonic (HC-SR04) Sensor (x4), Hose Clamp (x8), Sensor Mounting Plate (x4), Sensor Reflection Plate (x4), Arduino.

***COMPONENT DESCRIPTIVE PHOTO:-***

****  

***CIRCUIT DIAGRAM:-***

**

***THEORY / WORKING PRINCIPLE :-*** This system provides us a plot of travel in all the 4 dampers simultaneously by measuring the distance travelled by each dampers every time with respect to the original static condition using Ultrasonic sensors in dynamic conditions in a controlled testing phase environment and finally store the data using “DAT” system.

***ULTRASONIC (HC-SR04) SENSOR :-***  **Ultrasonic transducers** or **ultrasonic sensors** are a type of acoustic sensor divided into three broad categories: transmitters, receivers and transceivers. Transmitters convert electrical signals into ultrasound, receivers convert ultrasound into electrical signals, and transceivers can both transmit and receive ultrasound.

In a similar way to radar and sonar, ultrasonic transducers are used in systems which evaluate targets by interpreting the reflected signals. For example, by measuring the time between sending a signal and receiving an echo the distance of an object can be calculated. Passive ultrasonic sensors are basically microphones that detect ultrasonic noise that is present under certain conditions.

*DATASHEET :* <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>

***IMPLEMENTATIONAL SPECIFICATIONS: -*** Ultrasonic sensors is mounted on the sensor mounting plate made up of a sheet metal and tightly sealed making it waterproof. Furthermore it is hooked up to the damper at its top using a hose clamp matching to whose orientation at the bottom side of damper the Sensor Reflection Plate is mounted again with the help of hose clamp as above.

**VALIDATION PHOTOS:-**

**CODE FILE : -**

#include <NewPing.h>#define SONAR\_NUM 4 // Number of sensors.#define MAX\_DISTANCE 200 // Maximum distance (in cm) to ping.

NewPing sonar[SONAR\_NUM] = { // Sensor object array.

NewPing(12, 11, MAX\_DISTANCE), // FRONT RIGHT // Each sensor's [ trigger pin, echo pin, and max distance to ping. ]

NewPing(4, 5, MAX\_DISTANCE), //FRONT LEFT

NewPing(8, 9, MAX\_DISTANCE), //REAR RIGHT

NewPing(7, 6, MAX\_DISTANCE)}; //REAR LEFT

void setup() {

Serial.begin(115200); // Open serial monitor at 115200 baud to see ping results.

}

void loop() { float a[4],b[4],h=0;

int time=millis()/100;

if(time<4)

{for(int k=0;k<4;k++)

{a[k]=a[k]+sonar[k].ping\_cm();

delay(1);h++;}

b[1]=a[1]\*4/h;

}

if(time>4)

{

for (uint8\_t i = 0; i < 4; i++) { // Loop through each sensor and display results.

delay(50); // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest delay between pings.

Serial.print("Sensor ");

Serial.print(i);

Serial.print(" = ");

Serial.print(sonar[i].ping\_cm());

Serial.print(" cm\t");

}

Serial.println();

Serial.println();

}

delay(100);

}

**WHEEL ALIGNMENT**

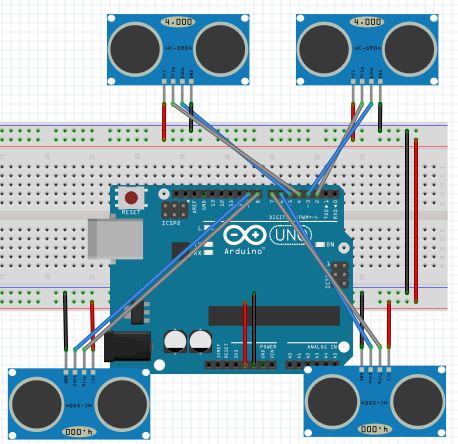
***AIM / PROBLEM STATEMENT: -*** To determine the exact orientation (of x-y plane with respect to the wheel) of hub during assembling phase to adjust the camber and toe angle according to needs.  
***COMPONENTS / APPARATUS REQUIRED -*** Ultrasonic (HC-SR04) Sensor (x4), Wheel Alignment Setup, Reference Sketch, Referential Wooden Circular Plates, Arduino.

*****COMPONENT DESCRIPTIVE PHOTO:-***

**

**

***CIRCUIT DIAGRAM :-***

**

***THEORY, WORKING PRINCIPLE AND IMPLEMENTATIONAL SPECIFICATIONS: -***This is a tool using which we can adjust the camber, and the toe angle of our ATV according to the needs in static condition.

In this ultrasonic sensors are placed in plus shape orientation with respect to ground after which 2 alternate sensors are required to provide data simultaneously so as to give an idea of the horizontal and vertical plane geometry accurately which helps in determining the plane as well. The distance from the two sensors is subtracted and inverse tangent of the output is evaluated providing the angle.

For this the buggy is placed on a reference sketch and so is the setup which then evaluates the current wheel orientation by calculating distance parameter of setup from wheel at various predefined locations which is then finally represented as wheel orientation after processing data through microcontroller.

***VALIDATION PHOTOS:-***

***CODE FILE : -***

#include <NewPing.h>

#define SONAR\_NUM 4 // Number of sensors.

#define MAX\_DISTANCE 200 // Maximum distance (in cm) to ping.

NewPing sonar[SONAR\_NUM] = { // Sensor object array.

NewPing(12, 11, MAX\_DISTANCE), // FRONT RIGHT // Each sensor's [ trigger pin, echo pin, and max distance to ping. ]

NewPing(4, 5, MAX\_DISTANCE), //FRONT LEFT

NewPing(8, 9, MAX\_DISTANCE), //REAR RIGHT

NewPing(7, 6, MAX\_DISTANCE)}; //REAR LEFT

void setup() {

Serial.begin(115200); // Open serial monitor at 115200 baud to see ping results.

}

void loop() { float a[4],b[4],h=0;

int time=millis()/100;

if(time<4)

{for(int k=0;k<4;k++)

{a[k]=a[k]+sonar[k].ping\_cm();

delay(1);h++;}

b[1]=a[1]\*4/h;

}

if(time>4)

{

for (uint8\_t i = 0; i < 4; i++) { // Loop through each sensor and display results.

delay(50); // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest delay between pings.

Serial.print("Sensor ");

Serial.print(i);

Serial.print(" = ");

Serial.print(sonar[i].ping\_cm());

Serial.print(" cm\t");

}

int d = atan(((sonar[0].ping\_cm()-sonar[2].ping\_cm()))/50);

int e = atan(((sonar[1].ping\_cm()-sonar[3].ping\_cm()))/50);

Serial.print(" camber = ");

Serial.print(d);

Serial.print("\t");

Serial.print(" toe = ");

Serial.print(e);

Serial.print("\t");

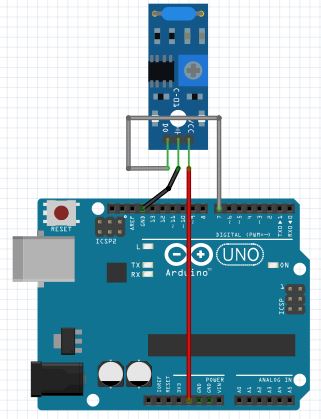
Serial.println();

Serial.println();

} delay(100) }}

**VIBRATION SENSOR**

***AIM / PROBLEM STATEMENT: -*** To study the vibrational behavior of buggy and obtain a live time graph on the serial monitor and save the same using DAQ system.



***COMPONENTS / APPARATUS REQUIRED -*** SW-420 Vibration Sensor, Arduino.

***CIRCUIT DIAGRAM:-***

***THEORY, WORKING PRINCIPLE AND IMPLEMENTATIONAL SPECIFICATIONS: -*** This is a tool made to help us understand the vibrational behavior of our buggy at various points enabling us to apply our knowledge and iterations to minimize the vibrations of buggy increasing vehicle’s performance and aesthetics.

In this a very simple hook up connection between microcontroller and the sensor is required to be established and the sensor is then mounted on the desired part where vibrations are to be calculated using double sided tape. Through this you would obtain a graph between the amplitude of vibration (on y axis) and frequency (on x axis).

***VALIDATION PHOTOS:-***

***CODE FILE :***

int LED\_Pin = 13;

int vibr\_Pin =3;

void setup(){

pinMode(LED\_Pin, OUTPUT);

pinMode(vibr\_Pin, INPUT); //set vibr\_Pin input for measurment

Serial.begin(9600); //init serial 9600

// Serial.println("----------------------Vibration demo------------------------");

}

void loop(){

long measurement =TP\_init();

delay(50);

// Serial.print("measurment = ");

Serial.println(measurement);

if (measurement > 1000){

digitalWrite(LED\_Pin, HIGH);

}

else{

digitalWrite(LED\_Pin, LOW);

}

//delay(300);

}

long TP\_init(){

delay(10);

long measurement=pulseIn (vibr\_Pin, HIGH); //wait for the pin to get HIGH and returns measurement

return measurement;

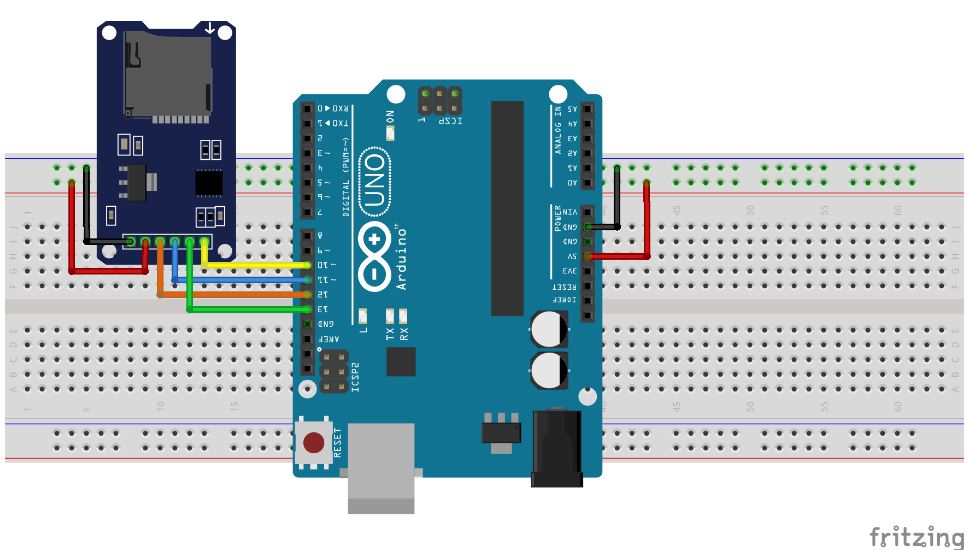
}

**DATA ACQUISATION SYSTEM**

***AIM / PROBLEM STATEMENT: -*** To apply file i/o to save the data obtained offline for records and qualitative study.

***COMPONENTS / APPARATUS REQUIRED: -*** SD card module, SD card, Arduino.

***CIRCUIT DIAGRAM:-***

**

***THEORY, WORKING PRINCIPLE and IMPLEMENTATIONAL SPECIFICATIONS:-***

**Data acquisition** is the process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that are manipulated by a computer. Data acquisition systems, abbreviated by the acronyms *DAS* or *DAQ*, typically convert analog waveforms into digital values for processing.

The components of data acquisition systems include:  
***Sensors, to convert physical parameters to electrical signals.***

Signal conditioning circuitry, to convert sensor signals into a form that can be converted to digital values.

***Analog-to-digital converters, to convert conditioned sensor signals to digital values.***

Here we have used a SD card module with micro-controller in order to record the data and save It for further use. The SD-card module is connected to the microcontroller in parallel to the existing system to record the data of that system.

***CODE FILE:-***

/\*

SD card read/write

This example shows how to read and write data to and from an SD card file

The circuit:

\* SD card attached to SPI bus as follows:

\*\* MOSI - pin 11

\*\* MISO - pin 12

\*\* CLK - pin 13[[ll[

\*\* CS - pin 4 (for MKRZero SD: SDCARD\_SS\_PIN)

This example code is in the public domain.

\*/

#include <SPI.h>

#include <SD.h>

File myFile;

void setup() {

pinMode(A0,OUTPUT);

digitalWrite(A0,HIGH);

// Open serial communications and wait for port to open:

Serial.begin(9600);

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

Serial.print("Initializing SD card...");

if (!SD.begin(4)) {

Serial.println("initialization failed!");

while (1);

}

Serial.println("initialization done.");

// open the file. note that only one file can be open at a time,

// so you have to close this one before opening another.

myFile = SD.open("test.txt", FILE\_WRITE);

// if the file opened okay, write to it:

if (myFile) {

Serial.print("Writing to test.txt...");

myFile.println("testing 1, 2, 3.");

// close the file:

myFile.close();

Serial.println("done.");

} else {

// if the file didn't open, print an error:

Serial.println("error opening test.txt");

}

// re-open the file for reading:

myFile = SD.open("test.txt");

if (myFile) {

Serial.println("test.txt:");

// read from the file until there's nothing else in it:

while (myFile.available()) {

Serial.write(myFile.read());

}

// close the file:

myFile.close();

} else {

// if the file didn't open, print an error:

Serial.println("error opening test.txt");

}

}

void loop() {

// nothing happens after setup

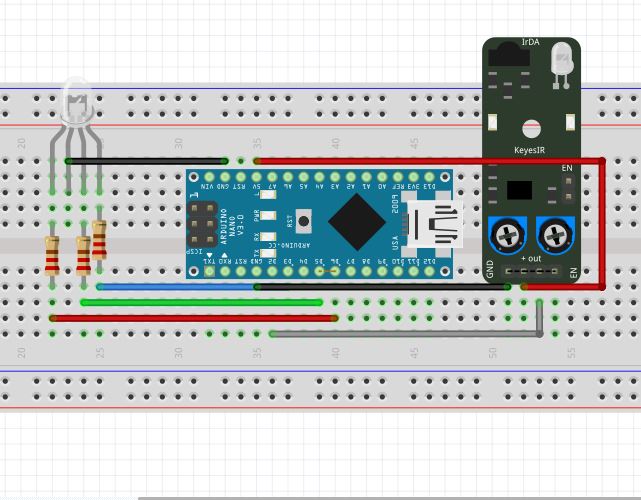
}

**ENGINE RPM INDICATOR**

***AIM / PROBLEM STATEMENT: -*** To indicate the engine rpm to driver with a colour changing LED for maintaining its RPM near around maximum torque condition.

***COMPONENTS / APPARATUS REQUIRED -*** IR Sensor, Arduino.

***CIRCUIT DIAGRAM:-***

**

***THEORY, WORKING PRINCIPLE and IMPLEMENTATIONAL SPECIFICATIONS: -*** A **tachometer** (**revolution-counter**, **tach**, **rev-counter**, **RPM gauge**) is an instrument measuring the rotation speed of a [shaft](https://en.wikipedia.org/wiki/Axle) or disk, as in a motor or other machine. Here we have used optical property of IR sensor to count the engine RPM and a colour changing LED in order to display the engine rpm with each colour encoded with certain RPM range.

***IR SENSOR:***  An IR sensor can measure the heat of an object as well as detects the motion. These types of sensors measures only infrared radiation, rather than emitting it that is called as a [passive IR sensor](https://www.elprocus.com/passive-infrared-pir-sensor-with-applications/). Usually in the infrared spectrum, all the objects radiate some form of thermal radiations. These types of radiations are invisible to our eyes, that can be detected by an infrared sensor. The emitter is simply an IR LED ([Light Emitting Diode](https://www.elprocus.com/explain-different-types-leds-working-applications-engineering-students/)) and the detector is simply an IR photodiode which is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, The resistances and these output voltages, change in proportion to the magnitude of the IR light received.

***VALIDATION PHOTOS:-***

***CODE FILE:-***

#include <SPI.h>

#include <SD.h>

File myFile;

float sp,flag1=0,flag2=0,t2=0,t1=0,ac=0;

volatile byte rpmcount;

unsigned int rpm;

unsigned long timeold;

void setup()

{

Serial.begin(9600);

attachInterrupt(digitalPinToInterrupt(2), rpm\_fun, RISING);

rpmcount = 0;

rpm = 0;

timeold = 0;

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

pinMode(A0,OUTPUT);

digitalWrite(A0,1);

Serial.print("Initializing SD card...");

if (!SD.begin(4)) {

Serial.println("initialization failed!");

return;

}

Serial.println("initialization done.");

pinMode(A0,OUTPUT);

digitalWrite(A0,1);

}

void loop()

{

myFile = SD.open("test2.txt", FILE\_WRITE);

if (rpmcount >= 3)

{

rpm = ((60000/(millis() - timeold))\*rpmcount);

timeold = millis();

rpmcount = 0;

Serial.print(rpm);

Serial.println();

Serial.println("Writing of SD card!");

myFile.print(rpm);

myFile.println();

}

myFile.close();

}

void rpm\_fun()

{

rpmcount++;

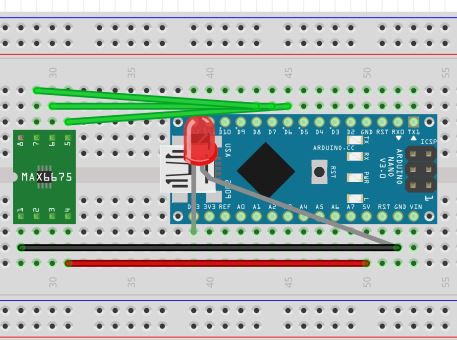
}

**CVT TEMPRATURE MONITORING**

***AIM / PROBLEM STATEMENT: -*** To indicate and observe CVT temperature

***COMPONENTS / APPARATUS REQUIRED: -*** K-Type Thermocouple, Max6675 amplifier, Arduino.

***CIRCUIT DIAGRAM:-***

**

***THEORY, WORKING PRINCIPLE AND IMPLEMENTATIONAL SPECIFICATIONS:-***

A Thermocouple is a sensor used to measure temperature. Thermocouples consist of two wire legs made from different metals. The wires legs are welded together at one end, creating a junction. This junction is where the temperature is measured. When the junction experiences a change in temperature, a voltage is created. The voltage can then be interpreted using thermocouple reference tables to calculate the temperature.

***VALIDATION PHOTOS:-***

***CODE FILE:-***

#include <max6675.h>

#include <Wire.h>

int thermoDO = 7; //DO

int thermoCS = 8; //CS

int thermoCLK = 9; //SCK

MAX6675 thermocouple(thermoCLK, thermoCS, thermoDO);

int vccPin = 3;

int gndPin = 2;

void setup() {

Serial.begin(9600);

// use Arduino pins

pinMode(vccPin, OUTPUT); digitalWrite(vccPin, HIGH);

pinMode(gndPin, OUTPUT); digitalWrite(gndPin, LOW);

pinMode(13, OUTPUT);

// wait for MAX chip to stabilize

delay(500);

}

void loop() {

// basic readout test, just print the current temp

Serial.println(thermocouple.readCelsius());

if(thermocouple.readCelsius() >200)

digitalWrite(13, HIGH);

if(thermocouple.readCelsius() >300)

digitalWrite(13, LOW);

digitalWrite(12, HIGH);

if(thermocouple.readCelsius() >360)

digitalWrite(12, LOW);

digitalWrite(11, HIGH);

delay(1000);

}

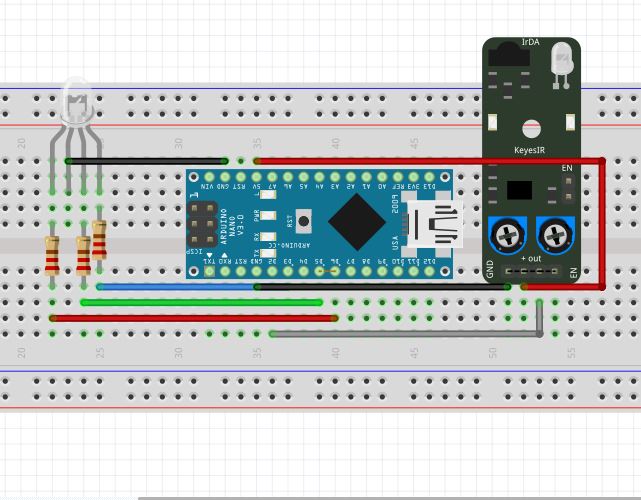
**CVT DYNO**

***AIM / PROBLEM STATEMENT:-***To indicate the engine rpm to driver with a colour changing LED for maintaining its RPM near around maximum torque condition.

***COMPONENTS / APPARATUS REQUIRED:-***Ir Sensor (x2),Arduino (x2).

***COMPONENT DESCRIPTIVE PHOTO:-***

**

***CIRCUIT DIAGRAM :-***

***THEORY, WORKING PRINCIPLE and IMPLEMENTATIONAL SPECIFICATIONS:-***A **tachometer** (**revolution-counter**, **tach**, **rev-counter**, **RPM gauge**) is an instrument measuring the rotation speed of a [shaft](https://en.wikipedia.org/wiki/Axle) or disk, as in a motor or other machine. Here we have used optical property of Ir sensor to count the engine RPM and a colour changing LED in order to display the engine rpm with each colour encoded with certain RPM range.

IR SENSOR : An IR sensor can measure the heat of an object as well as detects the motion. These types of sensors measures only infrared radiation, rather than emitting it that is called as a [passive IR sensor](https://www.elprocus.com/passive-infrared-pir-sensor-with-applications/). Usually in the infrared spectrum, all the objects radiate some form of thermal radiations. These types of radiations are  invisible to our eyes, that can be detected by an infrared sensor.The emitter is simply an IR LED ([Light Emitting Diode](https://www.elprocus.com/explain-different-types-leds-working-applications-engineering-students/)) and the detector is simply an IR photodiode which is sensitive to IR light of the same wavelength as that emitted by the IR LED. When IR light falls on the photodiode, The resistances and these output voltages, change in proportion to the magnitude of the IR light received.

*INDUCTIVE PROXIMITY SWITCH :*An **inductive proximity sensor** belongs to the category of non-contact electronic proximity sensor. It is used for positioning and detection of metal objects. The sensing range of an inductive switch is dependent on the type of metal being detected. Ferrous metals, such as iron and steel, allow for a longer sensing range, while nonferrous metals, such as aluminum and copper, may reduce the sensing range by up to 60 percent.

Since the output of an inductive sensor has two possible states, an inductive sensor is sometimes referred to as an **inductive proximity switch**.

***VALIDATION PHOTOS:-***

***VALIDATION DATA :***

***CODE FILE :-***

#include <SPI.h>

#include <SD.h>

File myFile;

float sp,flag1=0,flag2=0,t2=0,t1=0,ac=0;

volatile byte rpmcount;

unsigned int rpm;

unsigned long timeold;

void setup()

{

Serial.begin(9600);

attachInterrupt(digitalPinToInterrupt(2), rpm\_fun, RISING);

rpmcount = 0;

rpm = 0;

timeold = 0;

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

pinMode(A0,OUTPUT);

digitalWrite(A0,1);

Serial.print("Initializing SD card...");

if (!SD.begin(4)) {

Serial.println("initialization failed!");

return;

}

Serial.println("initialization done.");

pinMode(A0,OUTPUT);

digitalWrite(A0,1);

}

void loop()

{

myFile = SD.open("test2.txt", FILE\_WRITE);

if (rpmcount >= 3)

{

rpm = ((60000/(millis() - timeold))\*rpmcount);

timeold = millis();

rpmcount = 0;

sp=0.105728\*rpm;

Serial.print(sp);

Serial.print(" ");

Serial.print(rpm);

Serial.println();

Serial.println("Writing of SD card!");

myFile.print(sp);

myFile.print(" ");

myFile.print(rpm);

myFile.println();

}

if(sp>20&&flag1==0)

{

flag1++;

t1=millis();

}

if(sp>50&&flag2==0)

{

flag2++;

t2=millis();

Serial.println("\nTime :");

Serial.println(t2-t1);

ac=8.333/((t2-t1)/1000);

Serial.println("Acceleration:");

Serial.println(ac);

myFile.println("\nTime :");

myFile.println(t2-t1);

myFile.println("Acceleration:");

myFile.println(ac);

}

myFile.close();

}

void rpm\_fun()

{

rpmcount++;

}

**VEHICLE SPEED**

***AIM / PROBLEM STATEMENT :-*** To indicate the engine rpm to driver with a colour changing LED for maintaining its RPM near around maximum torque condition.

***COMPONENTS / APPARATUS REQUIRED :-*** Hall Effect based Inductive Proximity Switch, Arduino.

***COMPONENT DESCRIPTIVE PHOTO :-***

**

***CIRCUIT DIAGRAM :-***

***THEORY, WORKING PRINCIPLE AND IMPLEMENTATIONAL SPECIFICATIONS :-***A **tachometer** (**revolution-counter**, **tach**, **rev-counter**, **RPM gauge**) is an instrument measuring the rotation speed of a [shaft](https://en.wikipedia.org/wiki/Axle) or disk, as in a motor or other machine. Here we have used optical property of Ir sensor to count the engine RPM and a colour changing LED in order to display the engine rpm with each colour encoded with certain RPM range.

*INDUCTIVE PROXIMITY SWITCH :* An **inductive proximity sensor** belongs to the category of non-contact electronic proximity sensor. It is used for positioning and detection of metal objects. The sensing range of an inductive switch is dependent on the type of metal being detected. Ferrous metals, such as iron and steel, allow for a longer sensing range, while nonferrous metals, such as aluminum and copper, may reduce the sensing range by up to 60 percent.

Since the output of an inductive sensor has two possible states, an inductive sensor is sometimes referred to as an **inductive proximity switch**.

***VALIDATION PHOTOS:-***

***CODE FILE :-***

#include <SPI.h>

#include <SD.h>

File myFile;

float sp,flag1=0,flag2=0,t2=0,t1=0,ac=0;

volatile byte rpmcount;

unsigned int rpm;

unsigned long timeold;

void setup()

{

Serial.begin(9600);

attachInterrupt(digitalPinToInterrupt(2), rpm\_fun, RISING);

rpmcount = 0;

rpm = 0;

timeold = 0;

while (!Serial) {

; // wait for serial port to connect. Needed for native USB port only

}

pinMode(A0,OUTPUT);

digitalWrite(A0,1);

Serial.print("Initializing SD card...");

if (!SD.begin(4)) {

Serial.println("initialization failed!");

return;

}

Serial.println("initialization done.");

pinMode(A0,OUTPUT);

digitalWrite(A0,1);

}

void loop()

{

myFile = SD.open("test2.txt", FILE\_WRITE);

if (rpmcount >= 3)

{

rpm = ((60000/(millis() - timeold))\*rpmcount);

timeold = millis();

rpmcount = 0;

sp=0.105728\*rpm;

Serial.print(sp);

Serial.print(" ");

Serial.print(rpm);

Serial.println();

Serial.println("Writing of SD card!");

myFile.print(sp);

myFile.print(" ");

myFile.print(rpm);

myFile.println();

}

if(sp>20&&flag1==0)

{

flag1++;

t1=millis();

}

if(sp>50&&flag2==0)

{

flag2++;

t2=millis();

Serial.println("\nTime :");

Serial.println(t2-t1);

ac=8.333/((t2-t1)/1000);

Serial.println("Acceleration:");

Serial.println(ac);

myFile.println("\nTime :");

myFile.println(t2-t1);

myFile.println("Acceleration:");

myFile.println(ac);

}

myFile.close();

}

void rpm\_fun()

{

rpmcount++;

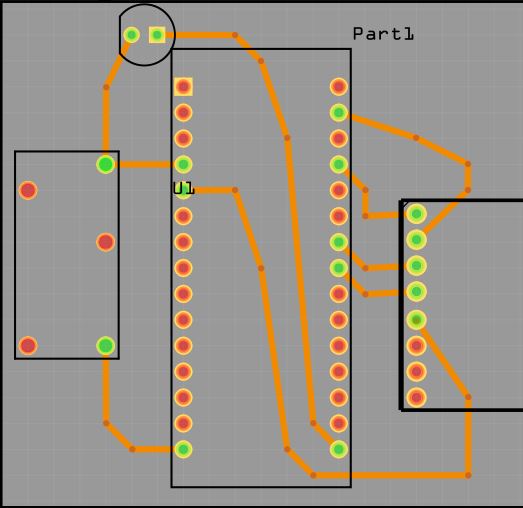
}

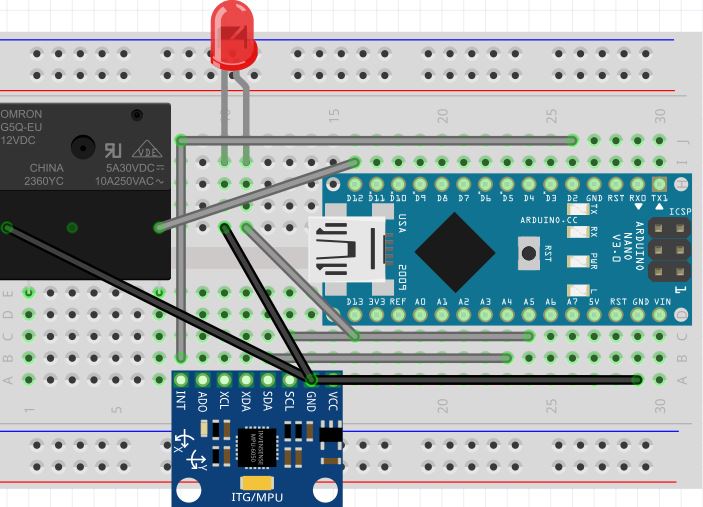
**AUTO-GYRO KILL SWITCH**

***AIM / PROBLEM STATEMENT :-***whenever there is a topple condition or roll over than first thing which driver is supposed to perform is engine kill but in case driver is unable to perform the kill than it can be hazardous so as to avoid such condition we invented an autonomous gyro kill setup.

***COMPONENTS / APPARATUS REQUIRED:-***mpu6050(gyro+acc), Arduino UNO, rubber padding and foam, aluminum case, duct/cello tape, relay(5v,7amp), battery.

***COMPONENT DESCRIPTIVE PHOTO :-***

***CIRCUIT DIAGRAM :-***

**

***THEORY / WORKING PRINCIPLE:*** This works on principle of gyro and accelerometer as it is also named as inertial moment unit then we have to use acc during the body moving and with its help we can calculate angle and calculate the position and when to turn relay on or off.

***GYRO (MPU-6050) SENSOR : -*** It is a inertial unit which contains gyro+acc+temp unit and by performing Kalman filter and its calculations and changing the output as desired we can attain the desired outcome.

***DATASHEET :***<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>

***IMPLEMENTATIONAL SPECIFICATIONS* :-** Initially we had built a basic code with raw filters , calculated the angle then we noticed the drift of angle then we changed our code to final filters and noticed no drift on using acc but as soon as we installed it on ATV we noticed the angle drift of 20-40o.  
 We removed the counter from digit and converted it to time based and found results better then also there came a problem of drift and after getting these problems we performed research on drones, which work on MPU then came to know that after using Kalman filter. We completed our task and then after taking help of power transmission dept. found the best position that will be suitable for obtaining vibrations.

***VALIDATION PHOTOS:-***

***CODE FILE :-***

#include <Wire.h>

#include <Kalman.h>

#define RESTRICT\_PITCH // Comment out to restrict roll to ±90deg instead - please read: <http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf>

int j=0,s=0;

double a;double b;double c;

int time,i=0,h=0,l=0,g=0;

float d,e,f;

Kalman kalmanX; // Create the Kalman instances

Kalman kalmanY;

/\* IMU Data \*/

double accX, accY, accZ;

double gyroX, gyroY, gyroZ;

int16\_t tempRaw;

double gyroXangle, gyroYangle; // Angle calculate using the gyro only

double compAngleX, compAngleY; // Calculated angle using a complementary filter

double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

uint32\_t timer;

uint8\_t i2cData[14]; // Buffer for I2C data

// TODO: Make calibration routine

void setup() {

Serial.begin(9600);

Wire.begin();

#if ARDUINO >= 157

Wire.setClock(400000UL); // Set I2C frequency to 400kHz

#else

TWBR = ((F\_CPU / 400000UL) - 16) / 2; // Set I2C frequency to 400kHz

#endif

i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz

i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro filtering, 8 KHz sampling

i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s

i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g

while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers at once

while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope reference and disable sleep mode

while (i2cRead(0x75, i2cData, 1));

if (i2cData[0] != 0x68) { // Read "WHO\_AM\_I" register

Serial.print(F("Error reading sensor"));

while (1);

}

delay(100); // Wait for sensor to stabilize

/\* Set kalman and gyro starting angle \*/

while (i2cRead(0x3B, i2cData, 6));

accX = (int16\_t)((i2cData[0] << 8) | i2cData[1]);

accY = (int16\_t)((i2cData[2] << 8) | i2cData[3]);

accZ = (int16\_t)((i2cData[4] << 8) | i2cData[5]);

// Source: http://www.freescale.com/files/sensors/doc/app\_note/AN3461.pdf eq. 25 and eq. 26

// atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2

// It is then converted from radians to degrees

#ifdef RESTRICT\_PITCH // Eq. 25 and 26

double roll = atan2(accY, accZ) \* RAD\_TO\_DEG;

double pitch = atan(-accX / sqrt(accY \* accY + accZ \* accZ)) \* RAD\_TO\_DEG;

#else // Eq. 28 and 29

double roll = atan(accY / sqrt(accX \* accX + accZ \* accZ)) \* RAD\_TO\_DEG;

double pitch = atan2(-accX, accZ) \* RAD\_TO\_DEG;

#endif

kalmanX.setAngle(roll); // Set starting angle

kalmanY.setAngle(pitch);

gyroXangle = roll;

gyroYangle = pitch;

compAngleX = roll;

compAngleY = pitch;

pinMode(12, OUTPUT); pinMode(11, OUTPUT);

timer = micros();

}

void loop() {

/\* Update all the values \*/

while (i2cRead(0x3B, i2cData, 14));

accX = (int16\_t)((i2cData[0] << 8) | i2cData[1]);

accY = (int16\_t)((i2cData[2] << 8) | i2cData[3]);

accZ = (int16\_t)((i2cData[4] << 8) | i2cData[5]);

tempRaw = (int16\_t)((i2cData[6] << 8) | i2cData[7]);

gyroX = (int16\_t)((i2cData[8] << 8) | i2cData[9]);

gyroY = (int16\_t)((i2cData[10] << 8) | i2cData[11]);

gyroZ = (int16\_t)((i2cData[12] << 8) | i2cData[13]);;

double dt = (double)(micros() - timer) / 1000000; // Calculate delta time

timer = micros();

// Source: http://www.freescale.com/files/sensors/doc/app\_note/AN3461.pdf eq. 25 and eq. 26

// atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2

// It is then converted from radians to degrees

#ifdef RESTRICT\_PITCH // Eq. 25 and 26

double roll = atan2(accY, accZ) \* RAD\_TO\_DEG;

double pitch = atan(-accX / sqrt(accY \* accY + accZ \* accZ)) \* RAD\_TO\_DEG;

#else // Eq. 28 and 29

double roll = atan(accY / sqrt(accX \* accX + accZ \* accZ)) \* RAD\_TO\_DEG;

double pitch = atan2(-accX, accZ) \* RAD\_TO\_DEG;

#endif

double gyroXrate = gyroX / 131.0; // Convert to deg/s

double gyroYrate = gyroY / 131.0; // Convert to deg/s

#ifdef RESTRICT\_PITCH

// This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees

if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {

kalmanX.setAngle(roll);

compAngleX = roll;

kalAngleX = roll;

gyroXangle = roll;

} else

kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter

if (abs(kalAngleX) > 90)

gyroYrate = -gyroYrate; // Invert rate, so it fits the restriced accelerometer reading

kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);

#else

// This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees

if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {

kalmanY.setAngle(pitch);

compAngleY = pitch;

kalAngleY = pitch;

gyroYangle = pitch;

} else

kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate the angle using a Kalman filter

if (abs(kalAngleY) > 90)

gyroXrate = -gyroXrate; // Invert rate, so it fits the restriced accelerometer reading

kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter

#endif

gyroXangle += gyroXrate \* dt; // Calculate gyro angle without any filter

gyroYangle += gyroYrate \* dt;

//gyroXangle += kalmanX.getRate() \* dt; // Calculate gyro angle using the unbiased rate

//gyroYangle += kalmanY.getRate() \* dt;

compAngleX = 0.93 \* (compAngleX + gyroXrate \* dt) + 0.07 \* roll; // Calculate the angle using a Complimentary filter

compAngleY = 0.93 \* (compAngleY + gyroYrate \* dt) + 0.07 \* pitch;

// Reset the gyro angle when it has drifted too much

if (gyroXangle < -180 || gyroXangle > 180)

gyroXangle = kalAngleX;

if (gyroYangle < -180 || gyroYangle > 180)

gyroYangle = kalAngleY;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

time=millis()/1000;

float x=kalAngleX;

float y=kalAngleY;

if(time<5)

{{a=a+x;b=b+y;

delay(1);h++;}

d=a/h;e=b/h;

if(l==0){

Serial.print("WAIT A MOMENT CALIBRATING.......");

Serial.println("");

Serial.println("GYRO KILL FOR XCELERATORS");l++;}}

if(time>5){

x=x-d;y=y-e;

Serial.print(x); Serial.print("\t");Serial.print(y); Serial.print("\t"); Serial.println("");

if((x>90&&x<250)||(y>90&&y<250))

{j++; Serial.print(j);

}

if((x<90||x>250)||(y<90||y>250))

{j=0; Serial.print(j);}

if(j>30){

{ Serial.print("KILL");

Serial.print(s);

Serial.println(" ");

digitalWrite(12,HIGH);

digitalWrite(11,HIGH);

j=0; }

delay(5000);s=0;}delay(500);

}}