**PID BASED LINE FOLLOWING**

What is PID?

A **proportional–integral–derivative controller** (**PID controller** or **three term controller**) is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an *error value* {\displaystyle e(t)} as the difference between a desired setpoint (SP) and a measured process variable (PV)

{Suppose the bot is supposed to be at x = 0 and the bot is at x =2 thus error (e) = 2;}

and applies a correction based on proportional, integral, and derivative terms (denoted *P*, *I*, and *D* respectively) which give the controller its name.

Why implement PID?

Line following seems to be accurate when carried out at lower speeds. As we start increasing the speed of the robot, it wobbles a lot and is often found getting off track.  
Hence some kind of control on the robot is required that would enable us to make it follow the line efficiently at higher speeds. This is where PID controller shines.

In order to implement line following one can basically start with just three sensors which are so spaced on the robot that-

1. If the centre sensor detects the line the robot steers forward
2. If the left sensor detects the line the robot steers right
3. If the right sensor detects the line the robot steers left.

This algorithm would make the robot follow the line, however, we would need to compromise with its speed to follow the line efficiently.

We can increase the efficiency of line following by increasing the number of sensors, say 5.

Here the possible combinations represent exact position like-

|  |  |
| --- | --- |
| 00100 | On the centre of the line |
| 00001 | To the left of the line |
| 10000 | To the right of the line |

There will be other possible combinations such as 00110 and 00011 that can provide us data on how far to the right is the robot from the centre of the line(same follows for left). Further to implement better line following we need to keep track of how long is the robot not centered on the line and how fast does it change its position from the centre.This is exactly what we can achieve using “PID” control.The data obtained from the array of sensors would then be put into utmost use and line following process would be much more smoother, faster and efficient at greater speeds.

PID is all about improving our control on the robot.

The idea behind PID control is that we set a value that we want maintained, either speed of a motor or reading from a sensor. We then take the present readings as input and compare them to the setpoint. From this an error value can be calculated, i.e, (error = setpoint - actual reading). This error value is then used to calculate how much to alter the output by to make the actual reading closer to the setpoint.

How to implement PID?

Terminology:

The basic terminology that one would require to understand PID are:

* Error - The error is the amount at which a device isn’t doing something right. For example, suppose the robot is located at x=5 but it should be at x=7, then the error is 2.
* Proportional (P) - The proportional term is directly proportional to the error at present.
* Integral (I) - The integral term depends on the cumulative error made over a period of time (t).
* Derivative (D) - The derivative term depends rate of change of error.
* Constant (factor)- Each term (P, I, D) will need to be tweaked in the code. Hence,they are included in the code by multiplying with respective constants.
  + P-Factor (Kp) - A constant value used to increase or decrease the impact of Proportional
  + I-Factor (Ki) - A constant value used to increase or decrease the impact of Integral
  + D-Factor (Kd) - A constant value used to increase or decrease the impact of Derivative

Error measurement:  In order to measure the error from the set position, i.e. the centre we can use the weighted values method. Suppose we are using a 5 sensor array to take the position input of the robot. The input obtained can be weighted depending on the possible combinations of input. The weight values assigned would be such that the error in position is defined both exactly and relatively.

The full range of weighted values is shown below. We assign a numerical value to each one.

|  |  |
| --- | --- |
| Binary Value | Weighted Value |
| 00001 | 4 |
| 00011 | 3 |
| 00010 | 2 |
| 00110 | 1 |
| 00100 | 0 |
| 01100 | -1 |
| 01000 | -2 |
| 11000 | -3 |
| 10000 | -4 |
| 00000 | -5 or 5 (depending on the previous value) |

The range of possible values for the measured position is -5 to 5. We will measure the position of the robot over the line several times a second and use these value to determine Proportional, Integral and Derivative values.

PID formula:

So what do we do with the error value to calculate how much the output be altered by? We would need  to simply add the error value to the output to adjust the robot’s motion. And this would work, and is known as proportional control (the P in PID). It is often necessary to scale the error value before adding it to the output by using the constant(Kp).

Proportional:

Difference = (Target Position) - (Measured Position)  
Proportional = Kp\*(Difference)

This approach would work, but it is found that if we want a quick response time, by using a large constant, or  if the error is very large, the output may overshoot from the set value. Hence the change in output may turn out to be unpredictable and oscillating. In order to control this, derivative expression comes to limelight.

Derivative:

Derivative provides us the rate of change of error. This would help us know how quickly does the error change from time to time and accordingly we can set the output.

Rate of Change = ((Difference) – (Previous Difference))/time interval  
Derivative= Kd \*(Rate of Change)

The time interval can be obtained by using the timer of microcontroller.

The integral improves steady state performance, i.e. when the output is steady how far away is it from the setpoint. By adding together all previous errors it is possible to monitor if there are accumulating errors. For example- if the position is slightly to the right all the time, the error will always be positive so the sum of the errors will get bigger, the inverse is true if position is always to the left. This can be monitored and used to further improve the accuracy of line following.

Integral:

Integral = Integral + Difference  
Integral = Ki\*(Integral)

Summarizing “PID” control-

|  |  |  |
| --- | --- | --- |
| Term | Expression | Effect |
| Proportional | Kp x error | It reduces a large part of the error based on present time error. |
| Integral | error dt | Reduces the final error in a system. Cumulative of a small error over time would help us further reduce the error. |
| Derivative | Kd x derror / dt | Counteracts the Kp and Ki terms when the output changes quickly. |

Therefore, Control value used to adjust the robot’s motion=

(Proportional) + (Integral) + (Derivative)

Tuning:

PID implementation would prove to be useless rather more troublesome unless the constant values are tuned depending on the platform the robot is intended to run on. The physical environment in which the robot is being operated vary significantly and cannot be modelled mathematically. It includes ground friction, motor inductance, center of mass, etc. Hence, the constants are just guessed numbers obtained by trial and error. Their best fit value varies from robot to robot and also the circumstance in which it is being run. The aim is to set the constants such that the settling time is minimum and there is no overshoot.

There are some basic guidelines that will help reduce the tuning effort.

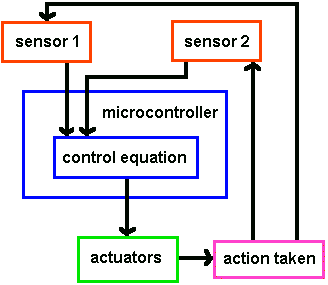
* Start with Kp, Ki and Kd equalling 0 and work with Kp first. Try setting Kp to a value of 1 and observe the robot. The goal is to get the robot to follow the line even if it is very wobbly. If the robot overshoots and loses the line, reduce the Kp value. If the robot cannot navigate a turn or seems sluggish, increase the Kp value.
* Once the robot is able to somewhat follow the line, assign a value of 1 to Kd (skip Ki for the moment). Try increasing this value until you see lesser amount of wobbling.
* Once the robot is fairly stable at following the line, assign a value of 0.5 to 1.0 to Ki. If the Ki value is too high, the robot will jerk left and right quickly. If it is too low, you won’t see any perceivable difference.  Since Integral is\*\* cumulative, the Ki value has \*\*a significant impact. You may end up adjusting it by .01 increments.
* Once the robot is following the line with good accuracy, you can increase the speed and see if it still is able to follow the line. Speed affects the PID controller and will require retuning as the speed changes.

Pseudo Code:Here is a simple loop that implements the PID control:

start:  
error = (target\_position) - (theoretical\_position)  
integral = integral + (error*dt)  
derivative = ((error) - (previous\_error))/dt  
output = (Kp*error) + (Ki*integral) + (Kd*derivative)  
previous\_error = error  
wait (dt)  
goto start

Lastly, PID doesn’t guarantee effective results just by simple implementation of a code, it requires constant tweaking based on the circumstances, once correctly tweaked it yields exceptional results. The PID implementation also involves a settling time, hence effective results can be seen only after a certain time from the start of the run of the robot. Also to obtain a fairly accurate output it is not always necessary to implement all the three expressions of PID. If implementing just PI results yields a good result we can skip the derivative part.

Shown here is the basic closed-loop (a complete cycle) control diagram:



The point of a control system is to get your robot actuators (or anything really) to do what you want without . . . ummmm . . . going out of control. The sensor (usually an [**encoder**](http://www.societyofrobots.com/sensors_encoder.shtml) on the actuator) will determine what is changing, the program you write defines what the final result should be, and the actuator actually makes the change. Another sensor could sense the environment, giving the robot a higher-level sense of where to go.

**Terminology**   
To get you started, here are a few terms you will need to know:

**error** - The error is the amount at which your device isnt doing something right. For example, if your robot is going 3mph but you want it to go 2mph, the error is 3mph-2mph = 1mph. Or suppose your robot is located at x=5 but you want it at x=7, then the error is 2. A control system cannot do anything if there is no error - think about it, if your robot is doing what you want, it wouldnt need control!

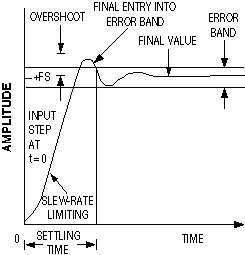
**proportional (P)** - The proportional term is typically the error. This is usually the **distance** you want the robot to travel, or perhaps a temperature you want something to be at. The robot is at position A, but wants to be at B, so the P term is A - B.

**derivative (D)** - The derivative term is the change in error made over a set time period (t). For example, the error was C before and now its D, and t time has passed, then the derivative term is (C-D)/t. Use the timer on your [**microcontroller**](http://www.societyofrobots.com/microcontroller_tutorial.shtml) to determine the time passed (see [**timer tutorial**](http://www.societyofrobots.com/programming_timers.shtml)).

**integral (I)** - The integral term is the accumulative error made over a set period of time (t). For example, your robot continually is on average off by a certain amount all the time, the I term will catch it. Lets say at t1 the error was A, at t2 it was B, and at t3 it was C. The integral term would be A/t1 + B/t2 + C/t3.

**tweak constant (gain)** - Each term (P, I, D) will need to be tweaked in your code. There are many things about a robot that is very difficult to model mathematically (ground friction, motor inductance, center of mass, ducktape holding your robot together, etc.). So often times it is better to just build the robot, implement a control equation, then tweak the equation until it works properly. A tweak constant is just a guessed number that you multiple each term with. For example, Kd is the derivative constant. Idealy you want the tweak constant high enough that your settling time is minimal but low enough so that there is no overshoot.

P\*Kp + I\*Ki + D\*Kd



What you see in this image is typically what will happen with your PID robot. It will start with some error and the actuator output will change until the error goes away (near the **final value**). The time it takes for this to happen is called the **settling time**. Shorter settling times are almost always better. Often times you might not design the system properly and the system will change so fast that it **overshoots** (bad!), causing some oscillation until the system settles. And there will usually be some **error band**. The error band is dependent on how fine a control your design is capable of - you will have to program your robot to ignore error within the error band or it will probably oscillate. There will always be an error band, no matter how advanced the system.

ignoring acceptable error band example:

if error <= .000001 //subjectively determined acceptable

then error = 0; //ignore it

**The Complete PID Equation**   
Combining everything from above, here is the complete PID equation:

Actuator\_Output = Kp\*P + Ki\*I + Kd\*D

or in easy to understand terms:

Actuator\_Output =

tweakA \* (distance from goal)   
+ tweakB \* (change in error)   
+ tweakC \* (accumulative error)

**Simplifications**   
The nice thing about tuning a PID controller is that you don't need to have a good understanding of formal control theory to do a fairly good job of it. Most control situations will work with just an hour or so max of tuning.

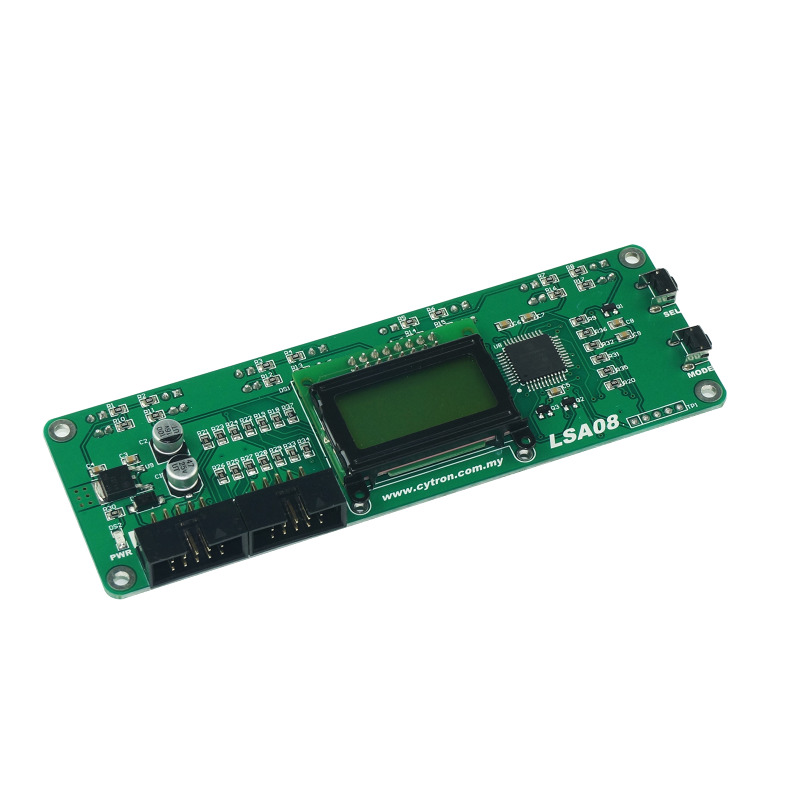
Better yet, rarely will you need the integral term. Thats right, just delete and ignore it! The only time you will need this term is when acceleration plays a big factor with your robot. If your robot is really heavy, or gravity is not on it's side (such as steep hills), then you will need the integral term. But out of all the robots I have ever programmed, only two needed an integral term - and both robots were over 30 lbs with a requirement for extremely high precision (millimeter or less error band). Control without the integral term is commonly referred to as simply PD control.

There are also times when you do not require a derivative term, but usually only when the device mechanical stabalizes itself, works at very low speeds so that overshoot just doesnt happen, or you simply dont require good precision.

**Sampling Rate Issues**   
The sampling rate is the speed at which your control algorithm can update itself. The faster the sampling rate, the higher precision control your robot will have. Slower sampling rates will result in higher settling times and an increased chance of overshoot (bad). To increase sampling rate, you want an even faster update of sensor readings, and minimal delay in your program loop. Its good to have the robot react to a changing environment before it drives off the table, anyway. Humans suffer from the sampling rate issue too (apparently drinking reduces the sampling rate, who would have guessed?).

**LSA08 LINE FOLLOWING SENSOR**

[**https://www.robotshop.com/media/files/pdf/manual-lsa08.pdf**](https://www.robotshop.com/media/files/pdf/manual-lsa08.pdf) **(user manual and datasheet)**



LSA08 (Advance Line Following Sensor Bar) consist of 8 sensors pair. LSA08 is typically used for embedded system or robots for line following task. The specially selected wavelength of **super bright green LED** as the sensor’s transmitter enables LSA08 to operate on various different colour surfaces. LSA08 is capable to operate on surface with colour of Red, Green, Blue, White, Black, Gray and possibly other colours with distinct brightness different. LSA08 has several different output modes, for the convenience of use for any system. Namely, the digital output port (8 parallel output line), the serial communication port (UART) and the analog output port.

**Features**:

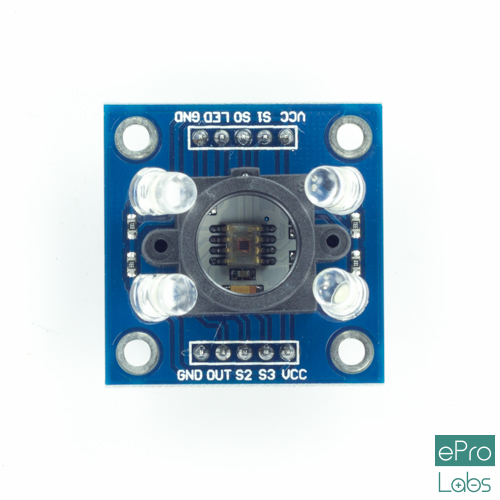
* 8 sensor pairs spaced 16mm.
* 12V input power
* On board Mode and Select button for instant configuration of LSA08
* 3 Different output mode (digital output port, UART output port, analog output port)
* LCD display unit showing 8 sensors analog value with bar chart and line position.
* Simple Auto-Calibration function to the line following surface.
* Junction Pulse (JPULSE) for detecting junction crossing and junction counting
* Power polarity protection
* Low current consumption (typically 26mA)
* Works on glossy or reflective surface
* Refresh rate up to 200Hz.

IT CAN BE OPERATED WITH 4 OUTPUT MODES (details are provided in the links):

1. Serial with PID (<https://tutorial.cytron.io/2015/07/31/line-following-robot-using-lsa08-in-serial-mode-with-pid-controller/> )
2. Serial with Digital (<https://tutorial.cytron.io/2015/07/31/line-following-robot-using-lsa08-in-serial-mode-with-digital-output/>)
3. Analog (<https://tutorial.cytron.io/2015/07/31/line-following-robot-using-lsa08-in-analog-mode/> )

1. Digital (<https://tutorial.cytron.io/2015/07/31/line-following-robot-using-lsa08-in-digital-mode/> )

TCS - 3200



**Introduction**

TCS3200-DB Color Sensor Daughterboard is a complete color detector, including a TAOS TCS3200 RGB sensor chip, white LEDs, collimator lens, and standoffs to set the optimum sensing distance. The TCS3200 has an array of photodetectors, each with either a red, green, or blue filter, or no filter (clear). The filters of each color are distributed evenly throughout the array to eliminate location bias among the colors. Internal to the device is an oscillator which produces a square-wave output whose frequency is proportional to the intensity of the chosen color.The applications of colour sensor are Test strip reading,Sorting by color, Ambient light sensing and calibration,Color matching.click [**here**](https://wiki.eprolabs.com/images/d/d1/SEN-0019.pdf) for datasheet.

**Specifications**

* Supply Voltage:**2.7V- 5.5V**
* Communicates Directly With a Microcontroller/Arduino
* High-Resolution Conversion of Light Intensity to Frequency
* Power Down Feature
* Programmable Color and Full-Scale Output Frequency

**Pin Description**

* **VDD-**Supply voltage
* **GND-**Power supply ground
* **OUT-**Output frequency (fo).
* **S0, S1-**Output frequency scaling selection inputs
* **S2, S3-**Photodiode type selection inputs

**Working Principle of Colour Sensor**

In the TCS3200, the light-to-frequency converter reads an 8 x 8 array of photodiodes. Sixteen photodiodes have blue filters, 16 photodiodes have green filters, 16 photodiodes have red filters, and 16 photodiodes are clear with no filters.When choosing color filter, the TCS3200 can allow only one particular color to get through and prevent other color. For example, when choosing the red filter, only red incident light can get through, blue and green will be prevented.Similarly ,when choose other filters we can get blue or green light.

The type of photodiode (blue, green, red, or clear) used by the device is controlled by two logic inputs, S2 and S3.

* For **Ret** photodiode both **S2 and S3** are **Low**
* For **Blue** photodiode **S2-Low** and **S3-High**
* For **Green** both **S2 and S3** are **High**
* When **S2-High**and **S3-Low**,none of the filter is selected.

**How to connect Colour Sensor TCS3200 to Arduino Uno?**

**Hardware and Software Required**

* Colour Sensor TCS3200 Module
* Arduino Uno
* Arduino IDE(1.0.6V)

**Hardware Connections**

The colour sensor should be connected to Arduino Uno as follows:

* VCC tp 5V
* GND to GND
* S0 to digital pin 8
* S1 to digital pin 9
* S2 to digital pin 12
* S3 to digital pin 11
* OUT to digital pin 10

In addition to this,connect red,green and blue led to digital pin 2,3 and 4 of Arduino Uno board.

**Program for Colour Sensor TCS3200**

After the connections are made,upload the problem given below.Now place red colour which means any object red in colour in front of the colour sensor and as a result the red led will glow as soon as it senses the red colour. Likewise repeat the same to detect the green and blue colour.

const int s0 = 8;

const int s1 = 9;

const int s2 = 12;

const int s3 = 11;

const int out = 10;

// LED pins connected to Arduino

int redLed = 2;

int greenLed = 3;

int blueLed = 4;

// Variables

int red = 0;

int green = 0;

int blue = 0;

void setup()

{

Serial.begin(9600);

pinMode(s0, OUTPUT);

pinMode(s1, OUTPUT);

pinMode(s2, OUTPUT);

pinMode(s3, OUTPUT);

pinMode(out, INPUT);

pinMode(redLed, OUTPUT);

pinMode(greenLed, OUTPUT);

pinMode(blueLed, OUTPUT);

digitalWrite(s0, HIGH);

digitalWrite(s1, HIGH);

}

void loop()

{

color();

Serial.print("R Intensity:");

Serial.print(red, DEC);

Serial.print(" G Intensity: ");

Serial.print(green, DEC);

Serial.print(" B Intensity : ");

Serial.print(blue, DEC);

//Serial.println();

if (red < blue && red < green && red < 20)

{

Serial.println(" - (Red Color)");

digitalWrite(redLed, HIGH); // Turn RED LED ON

digitalWrite(greenLed, LOW);

digitalWrite(blueLed, LOW);

}

else if (blue < red && blue < green)

{

Serial.println(" - (Blue Color)");

digitalWrite(redLed, LOW);

digitalWrite(greenLed, LOW);

digitalWrite(blueLed, HIGH); // Turn BLUE LED ON

}

else if (green < red && green < blue)

{

Serial.println(" - (Green Color)");

digitalWrite(redLed, LOW);

digitalWrite(greenLed, HIGH); // Turn GREEN LED ON

digitalWrite(blueLed, LOW);

}

else{

Serial.println();

}

delay(300);

digitalWrite(redLed, LOW);

digitalWrite(greenLed, LOW);

digitalWrite(blueLed, LOW);

}

void color()

{

digitalWrite(s2, LOW);

digitalWrite(s3, LOW);

//count OUT, pRed, RED

red = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

digitalWrite(s3, HIGH);

//count OUT, pBLUE, BLUE

blue = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

digitalWrite(s2, HIGH);

//count OUT, pGreen, GREEN

green = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

}

<https://howtomechatronics.com/tutorials/arduino/arduino-color-sensing-tutorial-tcs230-tcs3200-color-sensor/> (arduino interfacing with TCS3200 )

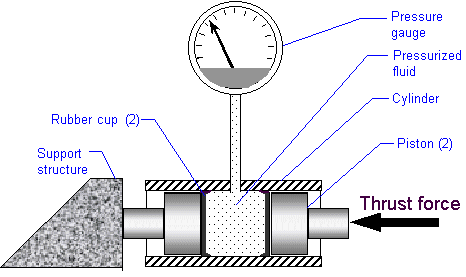
**Types of Load Cells and Sparkfun’s HX711 Amplifier Module :**

**A load cell is a physical element (or transducer if you want to be technical) that can translate pressure (force) into an electrical signal.**

So what does that mean? There are three main ways a load cell can translate an applied force into a measurable reading.

Hydraulic Load Cells

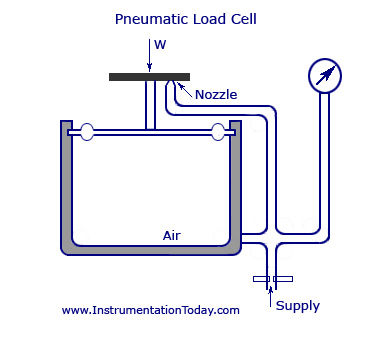
Hydraulic load cells use a conventional piston and cylinder arrangement to **convey a change in pressure by the movement of the piston and a diaphragm arrangement which produces a change in the pressure on a Bourdon tube connected with the load cells.**

[](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/hyd_dia3.gif)

*Diagram of a Hydraulic Load Cell from*[*Nikka’s Rocketry*](http://www.nakka-rocketry.net/hydlc.html)

Pneumatic Load Cells

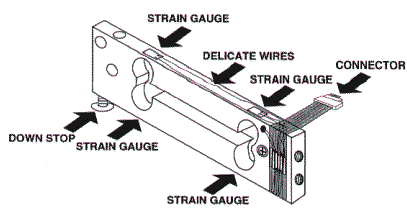
Pneumatic load cells use air pressure applied to one end of a diaphragm, and it escapes through the nozzle placed at the bottom of the load cell, which has a pressure gauge inside of the cell.

[](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/Pneumatic-Load-Cell.jpg)

*Diagram of a pneumatic load cell from*[*Instrumentation Today*](http://www.instrumentationtoday.com/force-transducers/2011/07/)

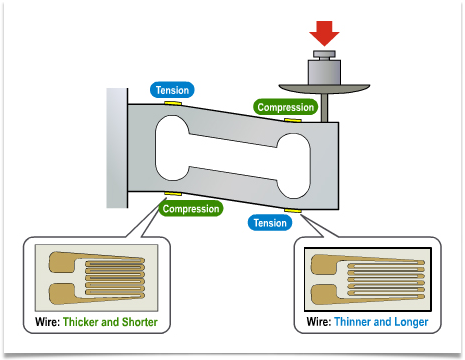
Strain Gauge Load Cells

And lastly (though there are many other less common load cell set ups), there is a strain gauge load cell, which is a mechanical element of which the force is being sensed by the deformation of a (or several) strain gauge(s) on the element.

[](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/loadcell.gif)

*Strain gauge load cell diagram from*[*Scalenet.com*](http://www.scalenet.com/applications/glossary.html)

In bar strain gauge load cells, the cell is set up in a “Z” formations so that torque is applied to the bar and the four strain gauges on the cell will measure the bending distortion, two measuring compression and two tension. When these four strain gauges are set up in a wheatstone bridge formation, it is easy to accurately measure the small changes in resistance from the strain gauges.

[](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/img0054.png)

*More in depth diagram of strain gauges on bar load cells when force is applied*

In this tutorial we will be focusing on strain gauge load cells like the ones SparkFun carries:

[](https://www.sparkfun.com/products/13332)

[Load Cell - 200kg, Disc (TAS606)](https://www.sparkfun.com/products/13332)

[In stock](https://learn.sparkfun.com/static/bubbles/) SEN-13332

[](https://www.sparkfun.com/products/13331)

[Load Cell - 50kg, Disc (TAS606)](https://www.sparkfun.com/products/13331)

[In stock](https://learn.sparkfun.com/static/bubbles/) SEN-13331

[](https://www.sparkfun.com/products/13329)

[Load Cell - 10kg, Straight Bar (TAL220)](https://www.sparkfun.com/products/13329)

[In stock](https://learn.sparkfun.com/static/bubbles/) SEN-13329

     3

Most strain gauge load cells work in very similar ways, but may vary in size, material, and mechanical setup, which can lead to each cell having different max loads and sensitivities that they can handle.

Strain Gauge Basics

A strain gauge is a device that measures electrical resistance changes in response to, and proportional of, strain (or pressure or force or whatever you so desire to call it) applied to the device. The most common strain gauge is made up of very fine wire, or foil, set up in a grid pattern in such a way that there is a linear change in electrical resistance when strain is applied in one specific direction, most commonly found with a base resistance of 120Ω, 350Ω, and 1,000Ω.

Each strain gauge has a different sensitivity to strain, which is expressed quantitatively as the gauge factor (GF). The gauge factor is defined as the ratio of fractional change in electrical resistance to the fractional change in length (strain).

(The gauge factor for metallic strain gauges is typically around 2.)

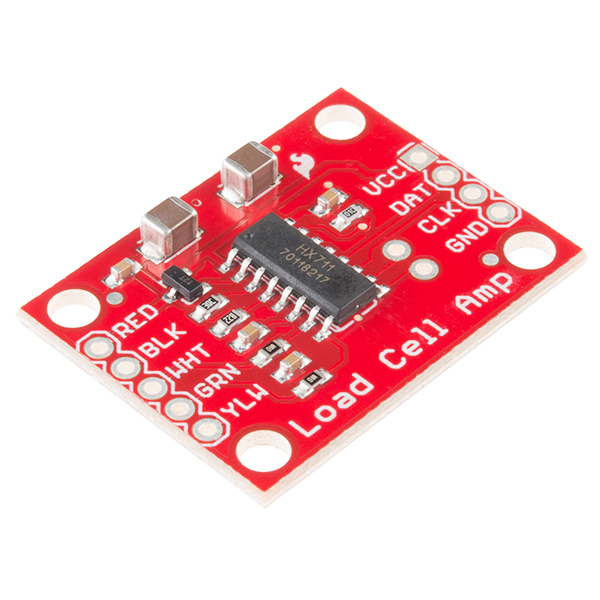
We set up a strain gauge load cell and measure that change in resistance and all is good, right? Not so fast. Strain measurements rarely involve quantities larger than a few millistrain [http://latex.codecogs.com/gif.latex?(e&space;\cdot&space;10%5e%7b-3%7d)](http://www.codecogs.com/eqnedit.php?latex=(e&space;\cdot&space;10%5e%7b-3%7d)) (fancy units for strain, but still very small). So lets take an example: suppose you put a strain of 500µε. A strain gauge with a gauge factor of 2 will have a change in electrical resistance of only

[http://latex.codecogs.com/gif.latex?2&space;*&space;(500&space;*&space;10%5e-%5e6)&space;=&space;0.1%25](http://www.codecogs.com/eqnedit.php?latex=2&space;*&space;(500&space;*&space;10%5e-%5e6)&space;=&space;0.1%25)

For a 120Ω gauge, this is a change of only 0.12Ω.

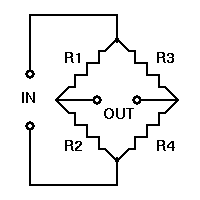
0.12Ω is a very small change, and, for most devices, couldn’t actually be detected, let alone detected accurately. So we are going to need another device that can either accurately measure super small changes in resistance (spoiler: they are very expensive) or a device that can take that very small change in resistance and turn it into something that we can measure accurately.

This is where an amplifier, such as the [HX711](https://www.sparkfun.com/products/13230) comes in handy.

[](https://cdn.sparkfun.com/assets/parts/1/0/3/9/5/13230-01.jpg)

*SparkFun’s*[*HX711 Amplifier breakout board*](https://www.sparkfun.com/products/13230)

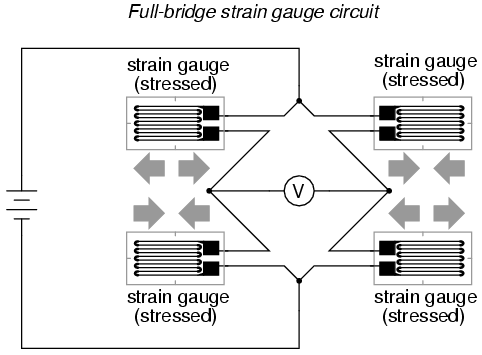
A good way of taking small changes in resistance and turning it into something more measurable is using a [wheatstone bridge](http://en.wikipedia.org/wiki/Wheatstone_bridge). A wheatstone bridge is a configuration of four resistors with a known voltage applied like this:

[](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/wheatstone_bridge.gif)

where Vin is a known constant voltage, and the resulting Vout is measured. If [http://latex.codecogs.com/gif.latex?R1/R2&space;=&space;R3/R4](http://www.codecogs.com/eqnedit.php?latex=R1/R2&space;=&space;R3/R4) then Vout is 0, but if there is a change to the value of one of the resistors, Vout will have a resulting change that can be measured and is governed by the following equation using ohms law:

[http://latex.codecogs.com/gif.latex?Vout&space;=&space;%5b(R3/(R3+R4)&space;-&space;R2/(R1+R2))%5d*Vin](http://www.codecogs.com/eqnedit.php?latex=Vout&space;=&space;%5b(R3/(R3+R4)&space;-&space;R2/(R1+R2))%5d*Vin)

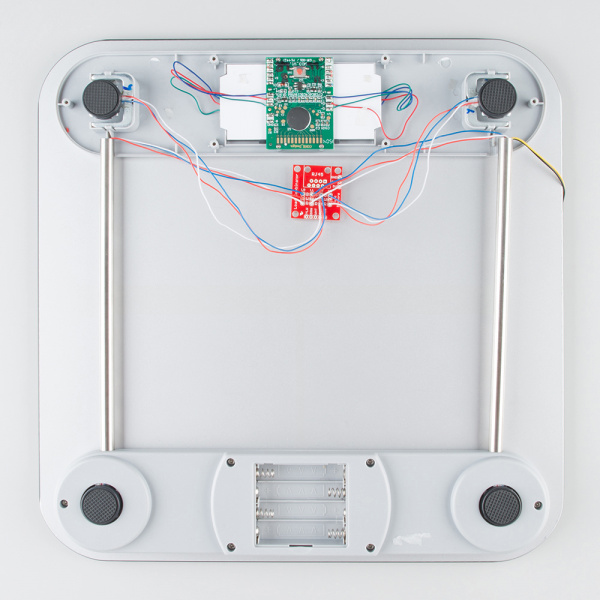
By replacing one of the resistors in a wheatstone bridge with a strain gauge, we can easily measure the change in Vout and use that to assess the force applied.

[](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/00431.png)

*Bar load cell wheatstone bridge example From*[*All About Circuits*](http://www.allaboutcircuits.com/textbook/direct-current/chpt-9/strain-gauges/)

Combinator Basics

Now that you have a load cell with a strain gauges hooked up to an amplifier, you can now measure force applied to your cell. For more information about how to hook up strain gauges, load cells, and amplifiers go to [our hookup guide](https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide).

[](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/3/HX711_and_Combinator_board_hook_up_guide-09.jpg)

*Bathroom scale using the*[*Load Sensor Combinator*](https://www.sparkfun.com/products/13281)*to combine twelve wires into one wheatstone bridge*

But what happens when you don’t have a load cell with four strain gauges? Or you want to measure something really heavy on something scale like?

You can combine four single strain gauge load cells (sometimes referred to as [Load sensors](https://www.sparkfun.com/products/10245))! Using the same wheatstone bridge principle, you can use a [combinator](https://www.sparkfun.com/products/13281) to combine the single strain gauge load cells into a wheatstone bridge configuration where the force applied to all four single strain gauge load cells is added to give you a higher maximum load, and better accuracy than just one, and then the combinator can be hooked up to the same amplifier for easier measuring.

For more information on hooking up load sensors go to [our hookup guide](https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide).

This is the same layout that you would find in say your home scale. There would be four strain gauge load cells hooked up to a [combinator](https://www.sparkfun.com/products/13281) and an [amplifier](https://www.sparkfun.com/products/13230) to give you your weight reading.

Resources and Going Further

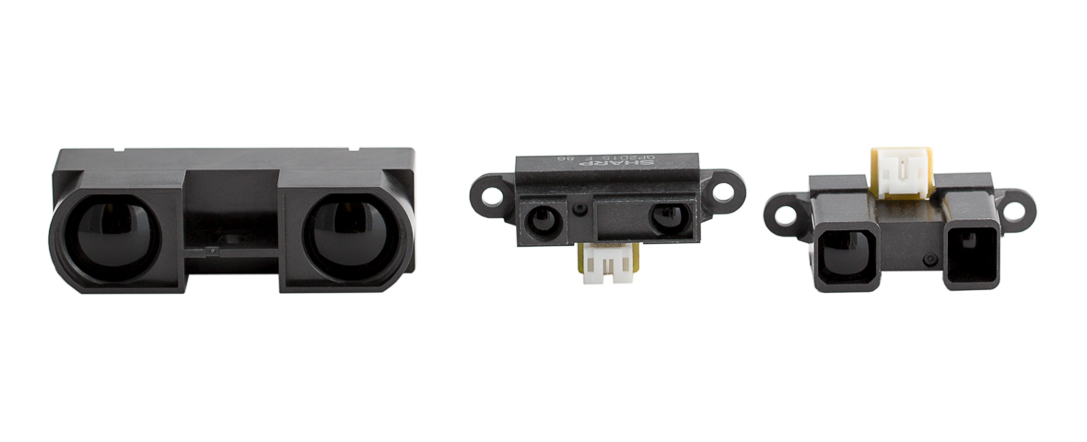
For more information about the load cells, check out the resources below:

* Datasheet
  + [50kg and 200kg, Disc/Button-Type (TAS606) (JPG)](https://cdn.sparkfun.com/assets/learn_tutorials/3/8/2/Datasheet_SEN-13331_50kg_13332_200kgDisk_TAS606_.jpg)
  + [200kg, S-Type (TAS501) (PDF)](https://cdn.sparkfun.com/assets/parts/1/2/2/3/8/TAS501.pdf)
  + [10kg, Straight Bar (TAL220) (PDF)](https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/TAL220M4M5Update.pdf)

We can also refer SparkFun’s HX711 Amplifier’s Data Sheet Here : <https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf>

About HX711 <https://www.sparkfun.com/products/13879>

SHARP IR SENSOR



# SHARP INFRARED RANGER COMPARISON

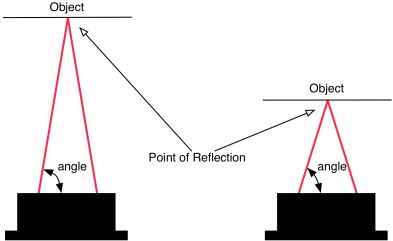
## INTRODUCTION

Sharp infrared detectors and rangers boast a small package, very low power consumption and a variety of output options. In order to maximize each sensor's potential, it is important to understand how these types of IR sensors work, their effective ranges, and how to interface to them.

## THEORY OF OPERATION

There are two major types of Sharp's infrared (IR) sensors based on their output: analog rangers and digital detectors. Analog ranges provide information about the distance to an object in the ranger's view. Digital detectors provide a digital (high or low) indication of an object at or closer than a predefined distance.

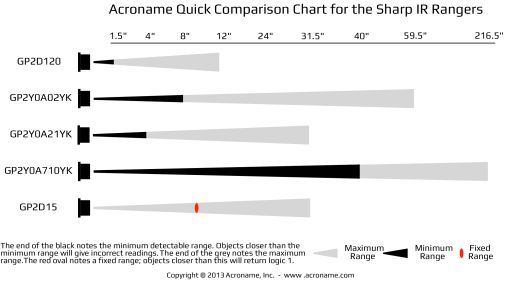
These rangers all use triangulation and a small linear CCD array to compute the distance and/or presence of objects in the field of view. In order to triangulare, a pulse of IR light is emitted by the emitter. The light travels out into the field of view and either hits an object or just keeps on going. In the case of no object, the light is never reflected, and the reading shows no object. If the light reflects off an object, it returns to the detector and creates a triangle between the point of reflection, the emitter and the detector.

  
Sharp IR detector angle of reflection arrival for near and far object

The incident angle of the reflected light varies based on the distance to the object. The receiver portion of the IR rangers is a precision lens that transmits reflected light onto various portions of the enclosed linear CCD array based on the incident angle of the reflected light. The CCD array can then determine the incident angle, and thus calculate the distance to the object. This method of ranging is very immune to interference from ambient light and offers indifference to the color of the object being detected.

## WHICH DETECTOR TO USE?

The table below characterizes each sensor by minimum and maximum ranges, as well as whether the sensor returns a varying distance value or a digital detection signal:

  
Comparison Chart for Sharp IR Rangers

## SHARP IR RANGE COMPARISON

|  |  |  |  |
| --- | --- | --- | --- |
| MODEL | OUTPUT | MIN. RANGE | MAX RANGE |
| GP2D120/GP2Y0A41 | Analog | 1.5" | 11.8" |
| GP2Y0A02 | Analog | 8" | 59" |
| GP2Y0A21 | Analog | 4" | 30" |
| GP2Y0A710 | Analog | 36" | 216" |
| GP2D15 | Digital | 9.5" | |

The [GP2Y0A710](https://acroname.com/products/r316-gp2y0a710yk.html) ('0A710'), [GP2D120](https://acroname.com/products/r146-gp2d120.html), [GP2Y0A41](https://acroname.com/products/r365-gp2y0a41sk0f), [GP2Y0A21](https://acroname.com/products/r301-gp2y0a21yk.html) ('0A21'), and [GP2Y0A02](https://acroname.com/products/r144-gp2y0a02yk.html) ('0A02') sensors offer true ranging information in the form of an analog output. The [GP2D15](https://acroname.com/products/r49-ir15.html), by contrast, provide a single digital value based on whether an object is present in it's range or not. None of the detectors require an external clock or signal. Instead, they fire continuously, requiring around 25mA of continuous current.

## MECHANICAL DIMENSIONS

All of the Sharp IR rangers are approximately the same size except the [0A710](https://acroname.com/products/r316-gp2y0a710yk.html). This sensor is larger in order to accommodate larger lenses and circuitry required for its superior range of 5.5 meters. The table below summaries the overall mechanical dimensions of each sensor. Be sure to review the data sheets for detailed dimensional drawings.

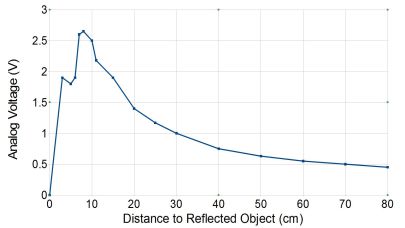
## SHARP IR DIMENSIONS

|  |  |  |  |
| --- | --- | --- | --- |
| MODEL | WIDTH (MM) | HEIGHT (MM) | DEPTH (MM) |
| GP2D120/GP2Y0A41 | 40.75 | 18.9 | 15.5 |
| GP2Y0A02 | 40.75 | 18.9 | 21.6 |
| GP2Y0A21 | 40.75 | 18.9 | 15.5 |
| GP2Y0A710 | 58.00 | 17.6 | 22.5 |
| GP2D15 | 40.75 | 18.9 | 15.5 |

  
Size comparison of [*GP2Y0A710*](https://acroname.com/products/SHARP-GP2Y0A710YK0F-PACKAGE) (left), [*GP2Y0A02*](https://acroname.com/products/SHARP-GP2Y0A02YK0F-IR-PACKAGE) (right) and [*GP2D15*](https://acroname.com/products/SHARP-GP2D15J0000F-IR-PACKAGE)/[*120*](https://acroname.com/products/SHARP-GP2D120XJ00F-IR-PACKAGE) (bottom)

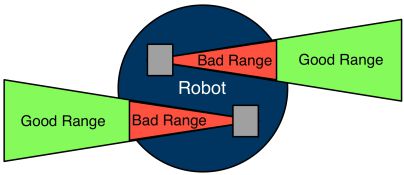
## NON-LINEAR OUTPUT

Due to the trigonometry involved in computing the distance to an object based on the reflected light incident angle, the output of these detectors is non-linear with respect to the distance being measured. The graph below shows an example of the analog output voltage vs distance to an object.

  
Example graph of Sharp IR Output Voltage vs Distance

Some interesting things to notice in this graph: first, the output of the detectors within the stated range (10cm - 80cm) is not linear but rather somewhat logarithmic. This curve will vary slightly from detector to detector so it is a good idea to "normalize" the output with a lookup table or parameterized function. This way, each detector can be calibrated to be consistent and result in data which varies nearly linearly relative to the distance to the object.

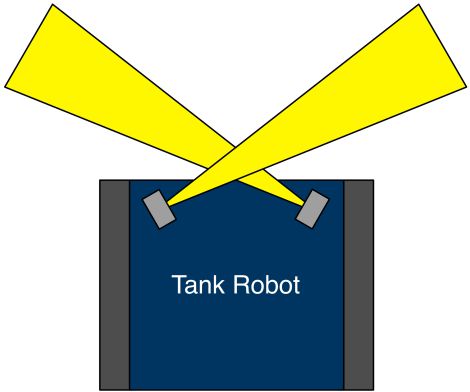
Second, notice that the output drops rapidly once the object is closer than the minimum distance (less than 10cm). As such, the output could be confused with a longer range reading. Such readings can be disastrous if your robot is slowing down as it approaches a solid object, gets below the minimum range and then misinterprets the apparently long range reading. If this errant reading is not handled correctly, the robot may drive full-speed into the object. The easiest way to avoid this is to cross-fire the detectors across the width or length of the robot as shown below.

  
Example of cross-firing detectors to avoid range errors

## BEAM PATTERN

The beam pattern for these rangers is rather consistent between types. The beam is roughly football shaped with the widest portion in the middle being about 16cm wide. This is a reasonably narrow beam pattern which makes for great ranging data when coupled with a servo to "sweep" the detector while taking readings.

When using Sharp sensors as a virtual bumper, it is advantageous to have the widest beam pattern possible in order to provide coverage for a large area such as the entire front of the robot. This can easily be accomplished using two sensors whose beams cross over one another in front of the robot (see below). The most common detector to use in this arrangement is the [GP2D15](https://acroname.com/products/r49-ir15.html).

  
Configuration of two sensors for effectively wider beam width

## INTERFACING THE SENSORS

Except for the [GP2Y0A710](https://acroname.com/products/r316-gp2y0a710yk.html), the Sharp IR sensors all use a connector called the Japan Solderless Terminal (JST) connector. These connectors have three wires: ground, Vcc, and the output. Since the sensors fire continuously and don't need any clocking to initiate a reading, interfacing to them is simple, but they continuously use power and can potentially interfere with one another when multiple detectors are used on a single robot. Interference can be avoided by keeping in mind the theory of operation of the sensors, when placing them on the robot.

The larger [GP2Y0A710](https://acroname.com/products/r316-gp2y0a710yk.html) is uses a 5-pin JST connector with two ground and two power lines. However, these lines can be soldered together provided the attached power supply is capable of delivering roughly 350mA of peak current (roughly 33-50mA continuous current). Like the other sensors, the [GP2Y0A710](https://acroname.com/products/r316-gp2y0a710yk.html) fires continuously. Be sure the review the data sheet for each sensor to ensure it is appropriate and compatible for the intended use case.

## CONCLUSION

The Sharp IR rangers and detectors are a great addition to the suite of sensors  available for robotics. They are inexpensive, are low power, fit in small spaces and have a unique range that is ideally suited to small robots in human spaces such as hallways, rooms and the mazes.

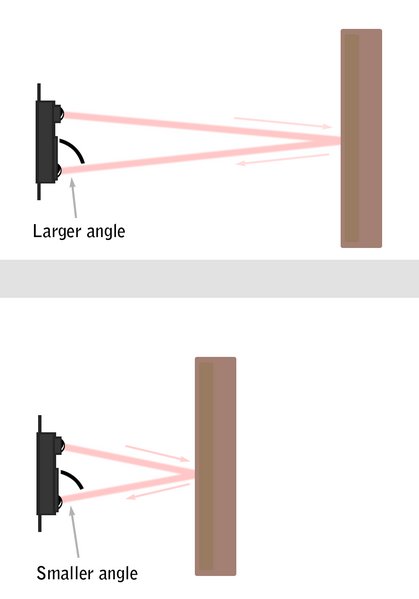
While these don't give absolute range accuracy, they offer very good information for a robot that typically deals with noisy information in the first place. Often, knowing whether a robot is close to a wall or far away is enough to make choices about what to do next.

**Comparison to a Normal IR Sensor**

A normal IR sensor can only tell if an object is bouncing back the IR light. There are times, however, when you also need to know how far away that object is. This is where the Sharp IR sensor comes into play. The Sharp sensor has a special detector that not only determines if there is light, but can also measure how far away an object is and return an analog value of the distance.

**How it works**

The detector in the Sharp IR sensor is similar to the imaging sensor found in digital cameras. Since the detector and the IR LED have a fixed distance and orientation relative to each other, the distance of an object will affect the angle at which the light from the IR LED hits the receiver. By looking at where the light hits the detector, it is possible to calculate the angle of the light and from that angle derive the distance to the object (all of which is done by the sensor itself).

[](http://education.rec.ri.cmu.edu/content/electronics/boe/ir_sensor/images/419px-SHARP_IR_operation.png)

*The Sharp IR sensor can detect object distance*

<http://www.instructables.com/id/How-to-Use-the-Sharp-IR-Sensor-GP2Y0A41SK0F-Arduin/>

ARDUINO TUTORIAL FOR SENSOR GP2Y0A415SK0F (here distance formula has to be calculated separately for each sensor in accordance to the datasheet0

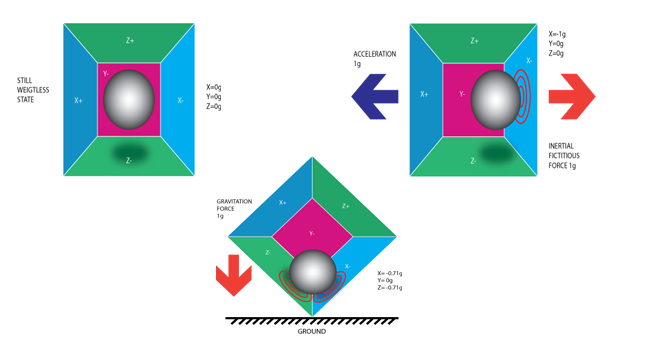
MPU 6050

MPU 6050 is an IMU (Inertia Measuring Unit) Sensor. It has an accelerometer and Gyroscope combined.

***How Does IMU Interfacing Work?***

IMU sensors usually consist of two or more parts. Listing them by priority, they are the accelerometer, gyroscope, magnetometer, and altimeter. The MPU 6050 is a 6 DOF (degrees of freedom) or a six-axis IMU sensor, which means that it gives six values as output: three values from the accelerometer and three from the gyroscope. The MPU 6050 is a sensor based on MEMS (micro electro mechanical systems) technology. Both the accelerometer and the gyroscope are embedded inside a single chip. This chip uses I2C (inter-integrated circuit) protocol for communication.

***How Does an Accelerometer Work?***



*Piezo Electric Accelerometer*

An [accelerometer](https://en.wikipedia.org/wiki/Accelerometer) works on the principle of the piezoelectric effect. Imagine a cuboidal box with a small ball inside it, like in the picture above. The walls of this box are made with piezoelectric crystals. Whenever you tilt the box, the ball is forced to move in the direction of the inclination due to gravity. The wall that the ball collides with creates tiny piezoelectric currents. There are three pairs of opposite walls in a cuboid. Each pair corresponds to an axis in 3D space: X, Y, and Z axes. Depending on the current produced from the piezoelectric walls, we can determine the direction of inclination and its magnitude.

***How Does a Gyroscope Work?***



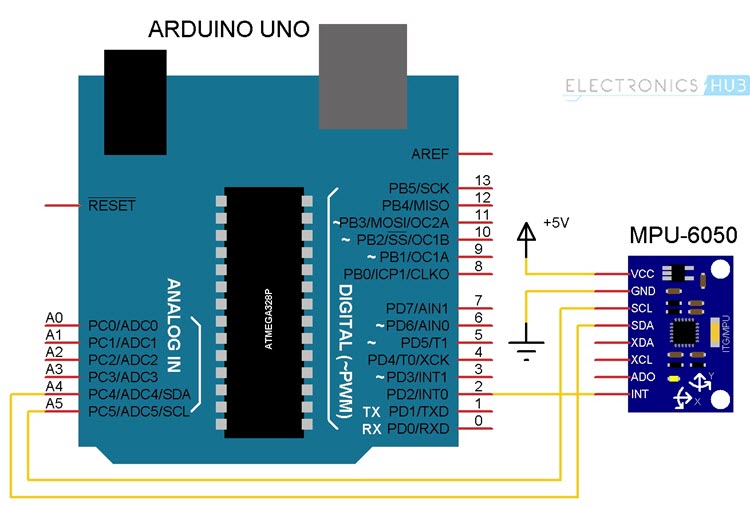
Gyroscopes work on the principle of Coriolis acceleration. Imagine that there is a fork-like structure that is in a constant back-and-forth motion. It is held in place using piezoelectric crystals. Whenever you try to tilt this arrangement, the crystals experience a force in the direction of inclination. This is caused as a result of the inertia of the moving fork. The crystals thus produce a current in consensus with the piezoelectric effect, and this current is amplified. The values are then refined by the host microcontroller.

Interfacing MPU6050 with an Arduino

MPU6050 uses an I2C communication protocol:

We initially need to download the MPU6050 Library for interfacing, which is available at <https://github.com/jarzebski/Arduino-MPU6050>

**NOTE:** In I2C Communication, the MPU-6050 always acts as a slave.



The SDA(Standard Data) and SCL(Standard Clock) are connected to the Analog Pins 4 and 5 respectively , the interruption pin is connected to D2

Arduino : MASTER and MPU6050: Slave

The three axes values respectively for gyroscope and accelerometer are to be initialized

I am attaching the basic code of MPU 6050 on the github repository along with leveling algorithm(Self Balancing) using MPU 6050

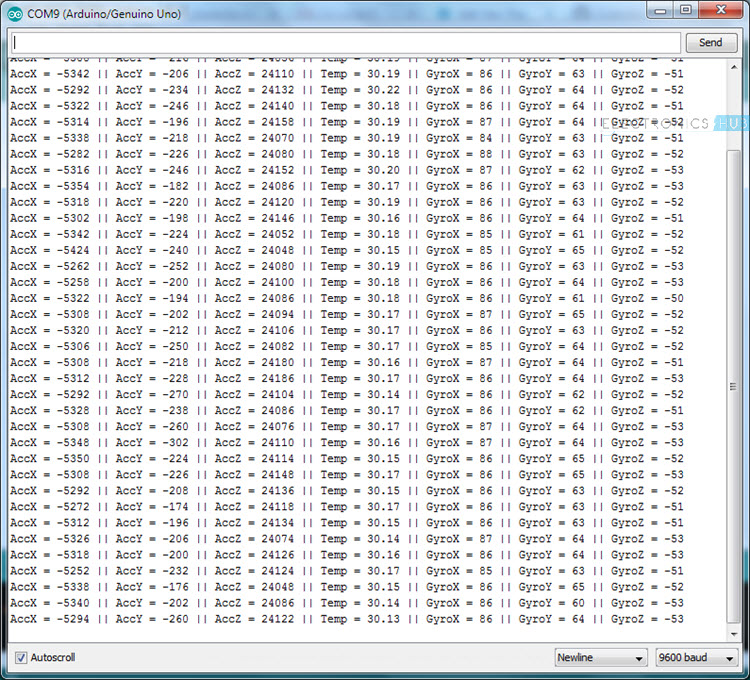
### Reading RAW Values from MPU6050

Before uploading the actual program, we will first see a simple program to read the raw values from the Accelerometer, Gyroscope and the Temperature Sensor. Simply connect the SCL and SDA wires of the MPU6050 to the corresponding I2C Pins of Arduino (A4 and A5) and upload the following code.

|  |  |
| --- | --- |
|  | #include<Wire.h> |
|  | const int MPU6050\_addr=0x68; |
|  | int16\_t AccX,AccY,AccZ,Temp,GyroX,GyroY,GyroZ; |
|  | void setup(){ |
|  | Wire.begin(); |
|  | Wire.beginTransmission(MPU6050\_addr); |
|  | Wire.write(0x6B); |
|  | Wire.write(0); |
|  | Wire.endTransmission(true); |
|  | Serial.begin(9600); |
|  | } |
|  | void loop(){ |
|  | Wire.beginTransmission(MPU6050\_addr); |
|  | Wire.write(0x3B); |
|  | Wire.endTransmission(false); |
|  | Wire.requestFrom(MPU6050\_addr,14,true); |
|  | AccX=Wire.read()<<8|Wire.read(); |
|  | AccY=Wire.read()<<8|Wire.read(); |
|  | AccZ=Wire.read()<<8|Wire.read(); |
|  | Temp=Wire.read()<<8|Wire.read(); |
|  | GyroX=Wire.read()<<8|Wire.read(); |
|  | GyroY=Wire.read()<<8|Wire.read(); |
|  | GyroZ=Wire.read()<<8|Wire.read(); |
|  | Serial.print("AccX = "); Serial.print(AccX); |
|  | Serial.print(" || AccY = "); Serial.print(AccY); |
|  | Serial.print(" || AccZ = "); Serial.print(AccZ); |
|  | Serial.print(" || Temp = "); Serial.print(Temp/340.00+36.53); |
|  | Serial.print(" || GyroX = "); Serial.print(GyroX); |
|  | Serial.print(" || GyroY = "); Serial.print(GyroY); |
|  | Serial.print(" || GyroZ = "); Serial.println(GyroZ); |
|  | delay(100); |
|  | } |

[**view raw**](https://gist.github.com/elktros/3f6a07bcc1220e899b0fec615f5b3a6e/raw/9eb694f7c25bb75bd1a7e98e2f0d23c4d829d58a/simple_raw_mpu_6050.ino)[**simple\_raw\_mpu\_6050.ino**](https://gist.github.com/elktros/3f6a07bcc1220e899b0fec615f5b3a6e#file-simple_raw_mpu_6050-ino) hosted with  by **[GitHub](https://github.com/)**

If you open the serial terminal, you will get the raw values from the Accelerometer and Gyroscope and calibrated Temperature from the Temperature Sensor. The data looks some thing like this.



As you can see, reading the raw values from the MPU6050 sensor is easy but this data and we need to perform additional calculation on this data to get the Yaw, Pitch and Roll.

 MPU6050 sensor, there is a special processor called DMP or Digital Motion Processor that is embedded on the same chip as the accelerometer and gyro. The use of this DMP is that it can be programmed with a firmware for performing complex calculations on the data from the sensors.

But there is no clear documentation about the DMP from InvenSense’s side and as a result we are missing out on making fast calculations on the sensor’s data directly on the chip.

Jeff Rowberg and others has done an excellent job in reverse engineering the DMP related information from the I2C signal analysis.

**Uploading the Code to Arduino and Testing MPU6050**

Before uploading the code, we need to download two libraries for Arduino. They are I2Cdev and MPU6050. The download links and the official GitHub links are given below.

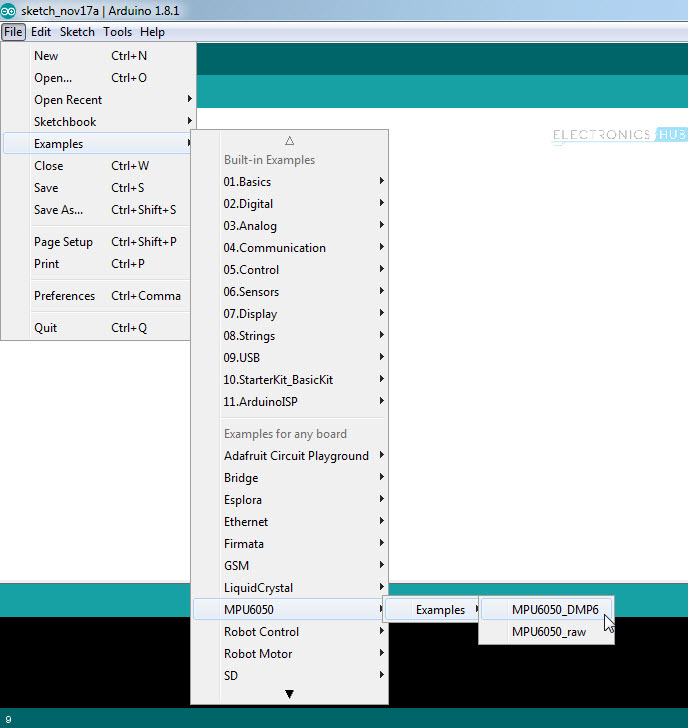
I2Cdev: [**DOWNLOAD I2CDEV LIBRARY**](https://www.electronicshub.org/wp-content/uploads/2017/11/I2Cdev.zip) or visit GitHub [**LINK**](https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/I2Cdev)

MPU6050: [**DOWNLOAD MPU6050 LIBRARY**](https://www.electronicshub.org/wp-content/uploads/2017/11/MPU6050.zip) or visit GitHub [**LINK**](https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050)

Download the MPU6050 Library and extract the content by unzipping the downloaded file. You will get a folder with name “MPU6050”. Copy this folder and paste it in the libraries folder of Arduino.

In my case, it is located at “C:\Program Files (x86)\Arduino\libraries”. Do the same thing for I2Cdev library.

If everything goes well, open Arduino IDE and navigate through the following path: File -> Examples -> MPU6050 -> Examples -> MPU6050\_DMP6 and open the example code MPU6050\_DMP6.



Upload this code to Arduino (assuming that you have already made the connections as per the circuit diagram) and once the code is uploaded, open the serial terminal.

Set the baud rate in the serial terminal to 115200 and you will get the following text.

“*Initializing DMP…*

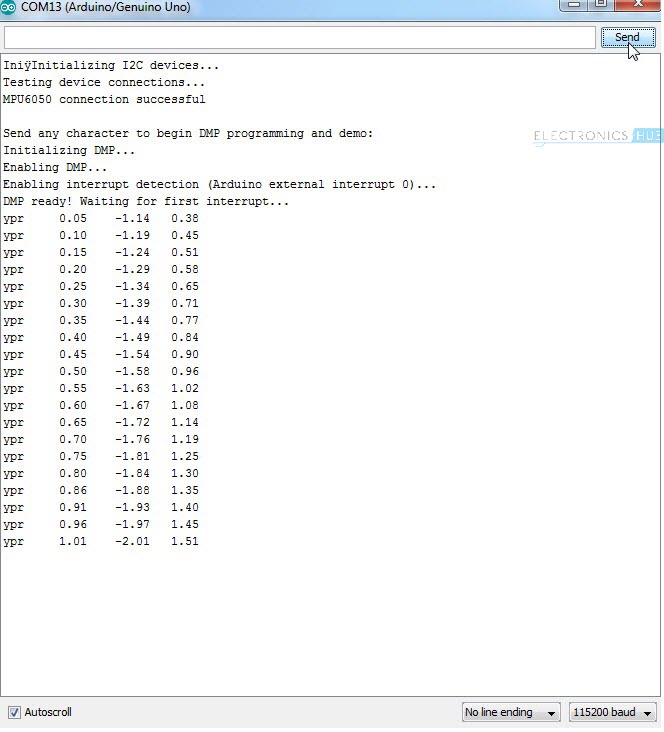
*Initializing I2C devices…*

*Testing device connections…*

*MPU6050 connection successful*

*Send any character to begin DMP programming and demo:*”

If you don’t get any data or still getting garbage data, just reset the Arduino. If you look at the last sentence, it say “Send any character to begin DMP programming and demo”. So, type any character like 1 or a in the serial monitor and send it. As soon as you hit enter, you can start seeing the Yaw, Pitch and Roll (ypr) values on the serial monitor.



**NOTE:** During this time, keep the MPU6050 on a stable and horizontal surface. Also, wait for 10 seconds for values from the MPU6050 to be stabilized.

**3D Modeling in Processing based on values from Interfacing Arduino and MPU6050**

In the next step of the project, we will take a look at 3D modeling the MPU6050 Sensor using Processing IDE, where you can view the 3D representation of the data from the sensor. Processing is a programming language and IDE that is developed for electronic arts and visual design. In fact, the Arduino IDE is also based on the Processing programming language.

To download the Processing IDE, visit this [**LINK**](https://processing.org/download/). Download and Install Processing IDE using the given link.

After downloading the Processing IDE and installing it (simply unzip the contents from the downloaded zip file), you need to download a library for Processing called “Toxi”. You can download the Toxi Library from this [**LINK**](https://bitbucket.org/postspectacular/toxiclibs/downloads/).

I’ve chosen the “toxiclibs-complete-0020” file. After downloading this file, extract the contents to a folder named “toxiclibs-complete-0020”.

Copy this folder and paste it in the libraries folder of the Processing. In my case, it was “C:\Users\Ravi\Documents\Processing\libraries”.

After copying the folder, you are now ready for 3D Modeling. First, you need to upload the previous Arduino code (MPU6050\_DMP6) with few modifications.

Open the MPU6050\_DMP6 (the example program which we uploaded earlier) in the Arduino IDE. Scroll down to the line that says the following.

#define OUTPUT\_READABLE\_YAWPITCHROLL

Comment this line by adding double forward slash in front of it.

//#define OUTPUT\_READABLE\_YAWPITCHROLL

Also, find the line that says  //#define OUTPUT\_TEAPOT  and uncomment it by removing the double forward slash. Now, you can upload the code. What we modified in the code is instead of sending the data to the serial terminal, we are forwarding it to the Processing IDE.

Now, open Processing IDE and click on File -> Open. Now, navigate to the folder where the MPU6050 library is installed for Arduino. Open the Processing example with name “MPUTeapot”.

In my case, the location for this example is C:\Program Files (x86)\Arduino\libraries\MPU6050\Examples\MPU6050\_DMP6\Processing\MPUTeapot.

This program has a provision for automatically selecting the PORT Number to which Arduino is connected. The line in the code associated with this is

String portName = Serial.list()[0];

Another way to specify the PORT number of Arduino is to manually enter the COM Port number. To do this, comment the above line and uncomment the following line and replace the COM Port number with appropriate COM Port Number to which Arduino is connected to.

//String portName = “COM4”;

Before hitting the run button in Processing, make sure that the Serial Monitor of Arduino IDE is closed. Now, click on run button in the processing IDE. You will get a window with a plane like structure. Wait for 10 seconds before 3D modeling the MPU6050.

The movements made by the MPU6050 can be seen through the 3D Object on the screen.