**Proyecto Final Control II:**

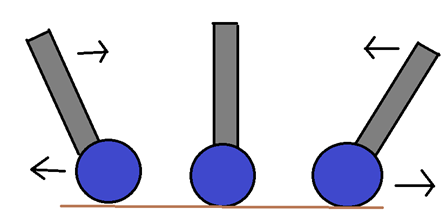
**Two-wheeled inverted pendulum**

Versión 3 - 16/06/18

Joshua Loayza

Stefani López

Esquema del proyecto:



**Fuente**: Hecho por nosotros

**RESUMEN**

El proyecto final de la materia Control II se construyó un *Two-wheeled inverted pendulum*, en otra palabras es un péndulo invertido que se balancea por 2 llantas. El ejemplo adecuado para referenciar es el equipo *Segway*:



**Fuente**: http://www.ecotoursmontreal.com/en/

El péndulo invertido es un ejemplo común del control automático y un tema innovador en la automatización. Hoy en día existen proyectos que automatizan el péndulo invertido de varias maneras pero nosotros nos enfocamos en el principio básico. Al finalizar el proyecto se espera que el sistema se equilibre automáticamente y mantenga una posición vertical.

**INTRODUCCIÓN Y ANTECEDENTES**

El proyecto está directamente relacionado con el transportador en dos ruedas *Segway*. Esta idea fue implementada en el área de investigación para el transporte ligero de personas. En el proyecto se enfocó en una modelización del Segway en versión pequeña. Para eso se usó un Arduino como conexión de hardware y software. Para el sistema de control se implementó un controlador PID.

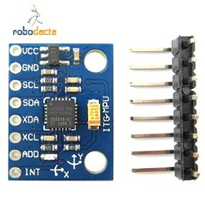
**OBJETIVO**

El objetivo de este proyecto es diseñar un auto con dos llantas el cual sea capaz de mantenerse en equilibrio verticalmente por encima del eje de rotación de este sistema.

**COMPONENTES**

**→ *MPU 6050***

Como sensor principal se usó el MPU 6050 que es compatible con Arduino. Este sensor tiene un giroscopio y un acelerómetro integrado. El giroscopio está usado para determinar la posición del auto, en específico para ver la inclinación. El acelerómetro mide la aceleración en relación con el cuerpo, lo que abre la posibilidad de usar esta aceleración sin considerar otros factores.



Fuente:<https://listado.mercadolibre.com.mx/sensor-mpu-5060>

→ ***Rueda y Motor***

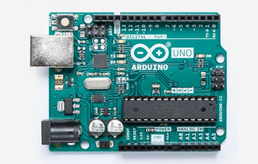
Se utilizó un motor DC el cual tiene un acople con las ruedas, estas son las más comunes para realizar proyectos con arduino. Estos motores soportan entre 3 y 12V (pero 6 a 8V está recomendado).



Fuente: https://www.prometec.net/producto/rueda-motor/

→ ***Arduino Uno***

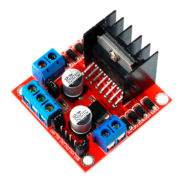
Un microcontrolador con una interfaz de desarrollo que resulta ser adecuado para la implementación del proyecto, por otro lado, resulta ser una de las placas más baratas y de facil entorno de programación con códigos abiertos.



Fuente:<https://store.arduino.cc/usa/arduino-uno-rev3>

***→ Modulo Puente H L298N***

Este driver o módulo nos ayuda a controlar 1-2 motores. Este puede controlar el giro de ambos motores al mismo tiempo y no tiene mucha complejidad.



Fuente: http://quanticaelectronics.com/catalogo/arduino-es/modulo-puente-h-l298n/

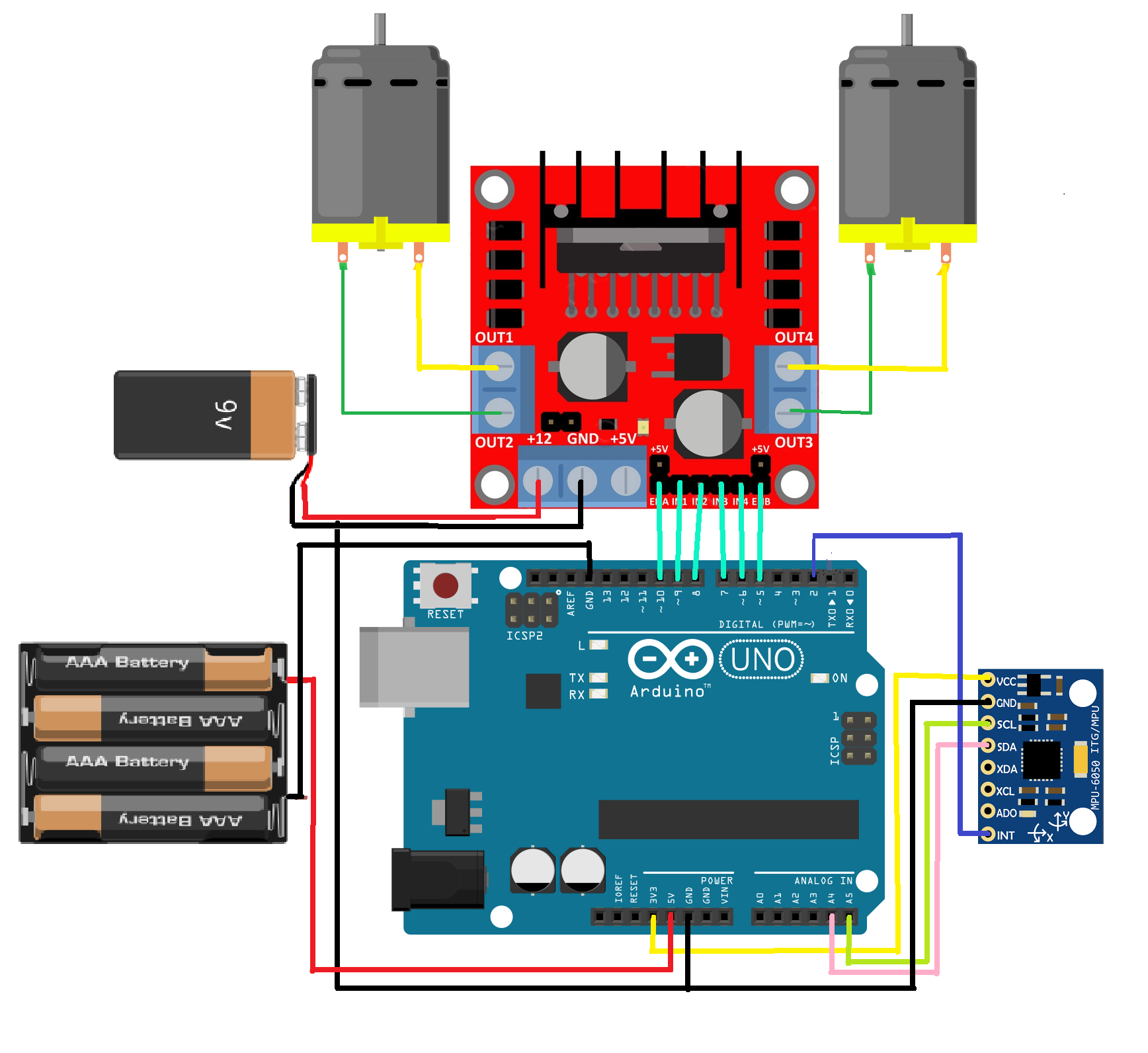
→ ***Otros componentes***

* 1 Batería 9V
* 4 Baterías 1,5V
* posiblemente una fuente DC
* Chasis ( para nuestro diseño utilizamos placas 3D de 10 x 15 cm, más 4 pernos con tuercas para ajustar )

**ARMADO**

→ ***Armado Circuital***

Se armó el siguiente circuito:



**Fuente:**circuito realizado por nosotros.

CONEXIONES:

Los 2 motores se conectan con los puertos respectivos del puente H.

Los pines D10 a D5 del Arduino se conectan a los pines EnA a EnB del puente H respectivamente,estas conexiones ayudan a controlar los motores.

El +12V del puente H se conecta con el positivo de la batería de 9V y el negativo de la batería y del puente H se van a la tierra común.

El GND de la parte analógica del Arduino se conecta con la tierra común.

La parte GND del MPU se conecta con la tierra común.

La parte de VCC del MPU se conecta con 3,3V de la parte analógica del Arduino.

El SCL del MPU va al A5 y el SDA al A4 de la parte analógica del Arduino.

El INT del MPU va a la parte digital D2 del Arduino.

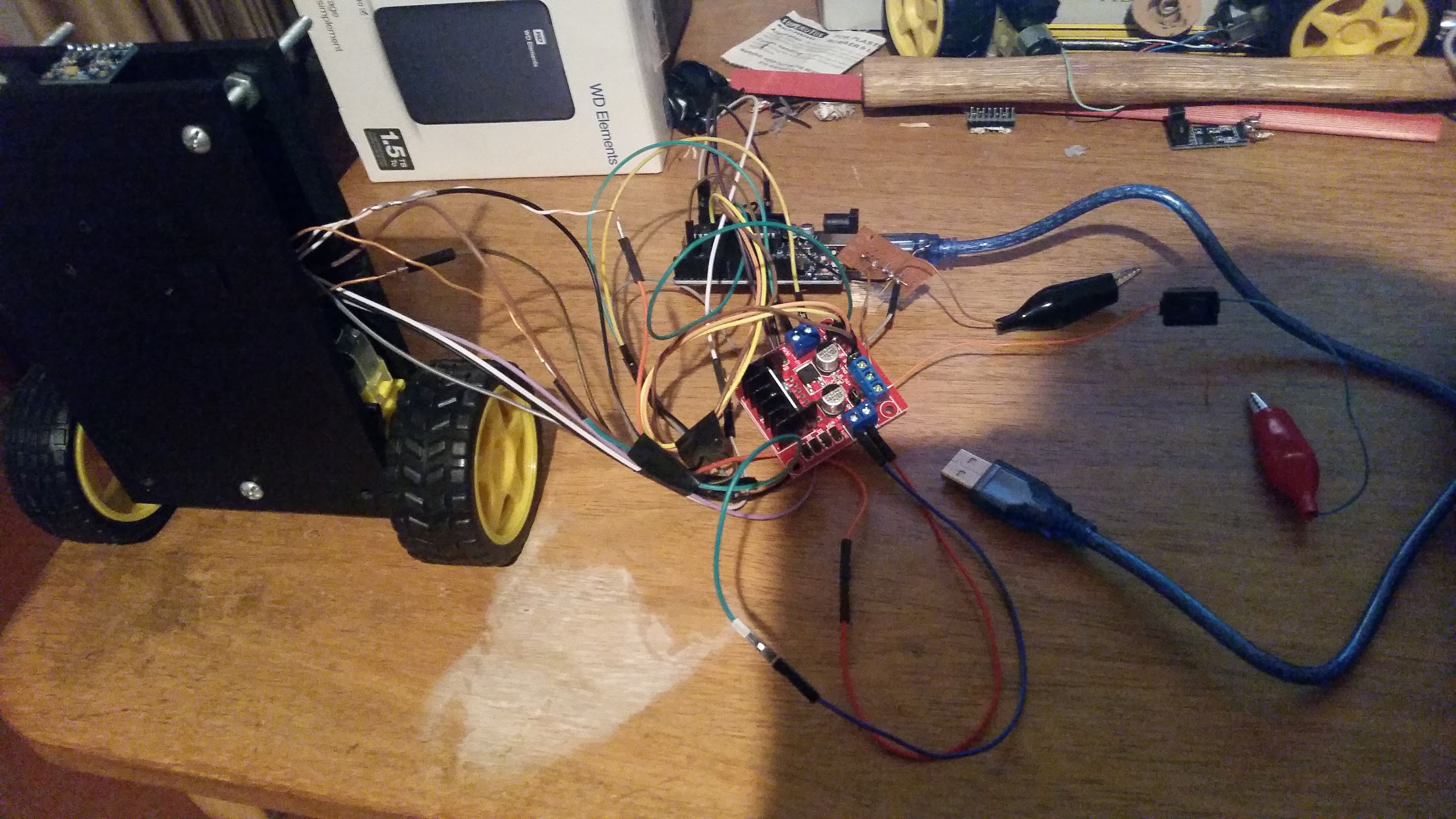
Las 4 baterías van al GND digital del Arduino (el negativo) y 5V de la parte analógica del Arduino (positivo de la batería).

**Armado Físico:**

Se unió el chasis junto con el circuito de la siguiente manera:



**Fuente:** Foto realizada por nosotros.



**Fuente:** Foto realizada por nosotros.

**Teoría involucrada**

1. **Controlador PID y LQR**

Para este proyecto se usó un controlador PID. Este es el más común que ayuda a controlar un movimiento para que llegue a un estado estable. El “P” es el factor proporcional, la “I” es el factor integral y la “D” es el factor derivativo. La función del “P” es para evaluar el error y acercarlo a 0, también acelera el tiempo de respuesta del sistema. El factor “D” es para ver la pendiente de acercamiento para que no haya mucho overshoot (se pasó el valor deseado). El “I” es para que el error no sea muy alto cuando haya mucha perturbación en los sensores (error sumado).

Aunque no se usa en este proyecto se simuló el controlador LQR que significa linear quadratic regulator. Este controlador es un controlador óptimo que intenta minimizar el costo.

1. **Simulaciones**

Aunque en este sistema no se usará una función de transferencia es bueno simular para poder estimar la salida del sistema. Para simular se usa las ecuaciones de estado tomadas referencialmente del paper Self-Balancing Robot de la universidad de Manchester escrito por Abdul Gafar.

1. **Valores del proyecto**

syms s;

Mb=0.267; %masa vehiculo [kg]

Mw=0.0648; %masa rueda [kg]

r=0.033; %wheel radius [m]

L=0.07; %distancia rueda centro masa [m]

Ke=0.0259; %DC motor back EMF constant [Vs/rad]

Jw=0.5\*Mw\*r^2; %wheel inertia [kgm^2]

h=0.15; %altura cuerpo [m]

w=0.1; %ancho cuerpo [m]

Jb=Mb\*(h^2+w^2)/12; %body pitch inertia [kgm^2]

Km=0.0259; %torque constant [Nm/A]

R=4.4; %DC motor resistance [Ohm]

g=9.78; %gravitional constant [m/s^2]

b=0.001; %coefficient friccion [Nm\*s/rad]

alpha=2\*(R\*b-Ke\*Km)\*(Mb\*L^2+Mb\*r\*L+Jb)/(R\*(2\*(Jb\*Jw+Jw\*L^2\*Mb+Jb\*Mw\*r^2+L^2\*Mb\*Mw\*r^2)+Jb\*Mb\*r^2));

beta=-L^2\*Mb^2\*g\*r^2/(Jb\*(2\*Jw+Mb\*r^2+2\*Mw\*r^2)+2\*Jw\*L^2\*Mb+2\*L^2\*Mb\*Mw\*r^2);

gamma=2\*(R\*b-Ke\*Km)\*(2\*Jw+Mb\*r^2+2\*Mw\*r^2+L\*Mb\*r)/(R\*r\*(2\*(Jb\*Jw+Jw\*L^2\*Mb+Jb\*Mw\*r^2+L^2\*Mb\*Mw\*r^2)+Jb\*Mb\*r^2));

sigma=L\*Mb\*g\*(2\*Jw+Mb\*r^2+2\*w\*r^2)/(2\*Jb\*Jw+2\*Jw\*L^2\*Mb+Jb\*Mb\*r^2+2\*Jb\*Mw\*r^2+2\*Jw\*Mw\*r^2+2\*L^2\*Mb\*Mw\*r^2);

epsilon=Km\*r/(R\*b-Ke\*Km);

A=[0, 1, 0, 0;0, alpha, beta, -r\*alpha;0, 0, 0, 1;0, gamma, sigma, -r\*gamma];

C=[0, 0, 0, 0;0,0,0,1];

B=[0; alpha\*epsilon; 0; gamma\*epsilon];

D=[0;0];

I=[1,0,0,0;0,1,0,0;0,0,1,0;0,0,0,1];

1. **Simulación PID en matlab**

La siguiente función es para encontrar la función de transferencia con las ecuaciones de estado.

G=C\*(s\*I-A)^(-1)\*B;

s=tf(‘s’);

step(G);

pidTuner %para tunear la funcion de transferencia G

Desafortunadamente la simulación con el controlador PID concluye que el sistema no se puede controlar. Esta incontrolabilidad se debe a las aproximaciones un tanto inadecuadas De forma experimental se pudo observar que el sistema si se puede estabilizar.

1. **Simulación LQR en matlab**

Primero se verifica si el sistema se puede estabilizar.

I=zeros(4);

I(1,1)=1;

I(2,2)=1;

I(3,3)=1;

I(4,4)=1;

disp('determinante para estable');

det(s\*I-A) %El sistema es estable como los polos estan en 0

Luego se verifica si el sistema se puede controlar.

disp('rango para controlable');

co=[B A\*B A^2\*B A^3\*B]; %controlabilidad

rank(co) %El sistema es controlable como el rango es 4

Después se calcula la ganancia.

Q=[200,0,0,0;0 30 0 0;0 0 1 0;0 0 0 1];

R=1;

[K,S,e] = lqr(A,B,Q,R); %K ganancia

disp('ganancias K');

K

Ac = [(A-B\*K)];

Bc = [B];

Cc = [C];

Dc = [D];

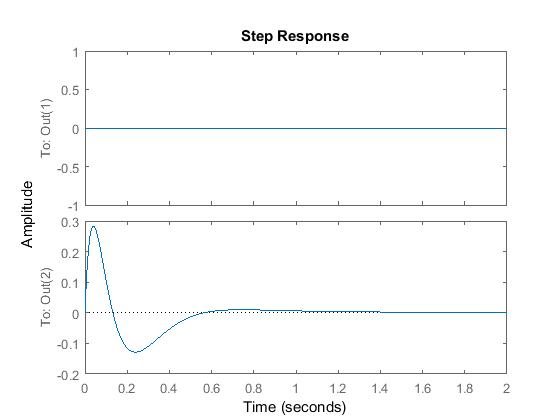
estados = {'x' 'x\_p' 'phi' 'phi\_p'};

entradas = {'r'};

salidas = {'x'; 'phi'};

sys\_ctl = ss(Ac,Bc,Cc,Dc);

step(sys\_ctl)



**Fuente:** Simulación realizada por nosotros.

1. **Función de transferencia experimental**

**Resultados**

El autito se balancea por un cierto tiempo pero despues se cae. Infortunadamente se presentó un problema con el Puente H. Al colocar 5V a la entrada el MPU funciono bien pero las llantas no se movieron con una velocidad adecuada. Al subir el voltaje para el puente H el MPU empieza a alterar su valor para cada posición y el autito ya no se puede estabilizar.

**Observaciones y Recomendaciones**

Se debe tener cuidado al elegir los miembros del grupo porque puede ser que no todos trabajan y perjudican al grupo.

Es necesario soldar los pines del MPU y verificar la respectiva continuidad en cada una de ellas para tener la seguridad de que mida los datos de posición adecuadamente.

El MPU necesita una librería especial para funcionar cual se puede descargar en el internet.

Tener cuidado con el armado del chasis, verificar que tenga un peso equilibrado, que el cuerpo del auto no choque con el piso y no roce con los motores, para mejor respuesta intentar tener un auto sumamente liviano para que la fuerza del sistema no sea una restricción ya que podría llegar a no ser controlable.

Hay que tener cuidado con el voltaje del puente H (máximo 6V) porque al colocar mucho voltaje el sensor MPU empieza a tener medidas con mucho error y variación.

**Conclusiones**

A lo largo de la implementación de este proyecto “ Self balancing”, se demostró que es un buen proyecto del área de control el cual parece uno básico pero sin embargo tiene mucha complejidad. Es necesario tomar todas las recomendaciones indicadas en el punto anterior, concluimos con un proyecto de innovación el cual tiene muchas respuestas posibles a planteamientos diferentes, como resultado se observó la respuesta del sistema de forma experimental e implementada adecuadamente, también las posibles respuestas con distintos controladores, en el cual el controlador PID demostró ser más eficaz en esta ocasión. No se logró realizar el objetivo buscado y planteado principalmente, pero por otro lado se obtuvo un avance notorio ya que se logró balancear el auto por corto tiempo, el tiempo era corto y esto se debe a la caída de voltaje que existe y por la pérdida de corriente en el driver o puente h. Recalcamos que se estudiaron todas las posibles soluciones empezando por las simulaciones, intentando con distintas compensaciones, fuentes, cambio de componentes.

**Referencias**

1. **Simulado Función de transferencia**

Arvidsson, M., & Karlsson, J. (2012). *Design, construction and verification of a self-balancing vehicle* (Doctoral dissertation, Chalmers University of Technology, 2012) (pp. 1-87). Göteborg. doi:<http://publications.lib.chalmers.se/records/fulltext/163640.pdf>

1. **Guias Segway**

Mikael Arvidsson, Jonas Karlsson. (2012). Design, construction and verification of a self-balancing vehicle. Swecia: 1.

Tmoir. (2017, October 10). Self-Balancing Upside Down Pendulum. Retrieved from <http://www.instructables.com/id/Self-Balancing-Upside-Down-Pendulum/>

Mert Kılıç. (2017, October 29). How to Make Self Balancing Robot with Arduino | Mert Arduino and Tech. Retrieved from <https://www.youtube.com/watch?v=9W5S5nqRegU>

Reactor science. (2017, March 19). How to make a self balancing robot using arduino||very easy code||reactor science. Retrieved from <https://www.youtube.com/watch?v=tGS7Vr_db98>

Castro, A. (2012). *Modeling and dynamic analysis of a two-wheeled inverted-pendulum* (Unpublished master's thesis).

**Anexos**

// This program is an inverted pendulum on a two-wheel cart

// Uses an arduino Uno or bigger one if nescessary, MPU650 Gyro and L298 2A Controller for the motors.

//T.J.Moir

//Code is provided AS-IS - no responsibility is accepted for it. Use at your own risk.

// Code modified from various instructable projects. eg http://www.instructables.com/id/Balancing-Instructable-Robot/ or http://www.instructables.com/id/Self-balancing-skateboardsegwy-project-Arduino-S/

// We use state-feedback - pole-placement here.The four states are angular position, angular velocity,angular acceleration and a 4th state for an integrator.

// you will need the I2Cdev.h library

// you will need the MPU6050.h library

// other libraries you should have from basic arduino that comes with the device software

// No steering at present

// No encoder feedback from wheels

// The control system is state-feedback - not PID 9though thre are similarities if you do the maths). I can measure position and velocity from the sensor but have to create acceleration by diffeentiating velocity.

// should really be able to get velocity directly but at pesent I synthesise it from velocity by band-limited differentiation.

///////////////////////////////////////////////////////////////////////////////////

// Wiring

///////////////////////////////////////////////////////////////////////////////

// Wire Interupt on MPU6050 to pin 2 on Arduino

// Connect SDL SDA pins on Arduino to MPU6050 - same lettering

// Various power supplies - connect the 3.3V on MPU6050 Gyro toArduino 3.3V output.

// Connect earths together and then to battery negative - important. You need three - one from each device.

// MPU6050 - lie it flat near the centre of gravity of the robot ie near middle of the axle.

//////////////////////////////////////////////////////////////////////////////

//Arduino L298 H Bridge

// D10 7

// D5 12

// D9 IN1

//D8 IN2

//D7 IN3

// D6 IN4

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// I also use two pots - you can use as many as you like but the two I use are for overall gain K and offset. The offset is for the gyro. Now and again I tim it so that the robot is vertical. Othrwise it leans over and uns away.

// some others put pots for PID (which this project is not) - which is convenient I suppose. I did it in software by tial and error which takes longer.

//So I use A0 and A1 on the Arduino which are A/D inputs. the wiper pat of the pots get wired to these pins A0 for gain and A1 for offset adjust. The other two leads on the pot are connected to regulated 5V from the L298 and 0 volts.

// Do not connect the pots to the main battery power supply since it is unregulated and will drift.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "I2Cdev.h"

#include <SoftwareSerial.h>

#include "MPU6050\_6Axis\_MotionApps20.h"

//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV\_ARDUINO\_WIRE implementation

// is used in I2Cdev.h

#if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE

#include "Wire.h"

#endif

MPU6050 mpu;

#define OUTPUT\_READABLE\_YAWPITCHROLL

bool dmpReady = false; // set true if DMP init was successful

uint8\_t mpuIntStatus; // holds actual interrupt status byte from MPU

uint8\_t devStatus; // return status after each device operation (0 = success, !0 = error)

uint16\_t packetSize; // expected DMP packet size (default is 42 bytes)

uint16\_t fifoCount; // count of all bytes currently in FIFO

uint8\_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars

Quaternion q; // [w, x, y, z] quaternion container

VectorFloat gravity; // [x, y, z] gravity vector

////////////////////////////////////////////////////////////////////////////

// Overall Gain is set from Pot 1 on A0 A/D convertor

static int pot1Pin = A0;

//Offset Pot on pin A1 so that it doesn't drift in one direction and stays steady.

static int pot2Pin=A1;

/////////////////////////////////////////////////////////////////////////////

int STD\_LOOP\_TIME = 1; //10mS loop time (100Hz) // code that keeps loop time at 10ms per cycle of main program loop Won't run any faster!

int lastLoopTime = STD\_LOOP\_TIME;

int lastLoopUsefulTime = STD\_LOOP\_TIME;

//float euler[3]; // [psi, theta, phi] Euler angle container

float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

int16\_t gyro[3]; // [x, y, z] gyro vector

/////////////////////////////////////////////////////////////////////////////////////////

// These gain values can be calculated rom the dynamic model of the system. If you don't have one then you need to guess! Educated guess that is. Start with Kp only and thn add Kd and Ka. Doesn't take much integral action.

// that's why Ki is small since we need good damping. Not good without Ki since it won't stay straight up and tends to flop back and forward due to stedy-state error. All control systems need integrators with very few exceptions.

// State 1 gain - popotional

float Kp = 15;

//Integral gain for integral state 4

float Ki = 0.3;

//Derivative gain for state 2

float Kd = 12;

// state gain for acceleration state 3

float Ka=10;

/////////////////////////////////////////////////////////////////////////////////////////

float currAnglep=0;

float currAngle = 0;

float angular\_rate=0;

float angular\_ratep=0;

float angular\_acceln=0;

float angular\_accelnp=0;

float setpoint = 0; // zero radians setpoint since it is vertical.

float control\_out=0;

float error=0;

float last\_error=0;

float integrated\_error=0;

float integrated\_errorp=0;

float est\_angle;

float est\_anglep;

float y\_angle\_comb;

float aa\_const = 0.998; //this means 0.2% (1-0.998)\*100 of the accelerometer reading is fed into angle of tilt calculation with every loop of program (to correct the gyro).

//accel is sensitive to vibration which is why we effectively average it over time in this manner. You can increase aa if you want to experiment.

//too high though and the board may become too vibration sensitive.

// Overall gain is read from external pot going into A0 ie Analogue Input 0. Although not absolutely nescessary it servs as a soft-start. Start the pot at zero volts and increase as requied.

float K = 0;

//offset is read from second pot going into A1 ie Analogue Input 1

float offset=0; // 9 degrees offset from vertical for this gyro

// Sampling rate is 100Hz is set later in this code - checked with Scope

float fs=100;

float dt=1/fs; // sampling interval in seconds

boolean timeflag=LOW; // for debug with scope

////////////////////////////////////////////////////

// First-OrderLow pass filter cut-off 1/(2\*pi\*tau) Hz, unity gain. ie 1/(1+s.tau)

//Cut-of freq in Hz, sampling freq fs=100Hz. This is for a steep roll-off to prevent resonances and noise shaking the thing apart.

float fcut=14;

float tau=1/(2\*3.14\*fcut);

float gnum= (1-(2\*tau/dt));

float gden = (1+(2\*tau/dt));

float filt\_par=gnum/gden;

// gain is 1/ gden

float filt\_gain=1/gden;

float filt=0;

float filtp=0;

float filt\_out=0;

/////////////////////////////////////////

// Filter to filter the Gyro - not too much or phase shift will cause instability. We use an IIR filter and not an FIR Savitsky Golay filter.

// The reason is that the Savitsky Golay filter is non-minimum phase and hence has a lot of negative going phase which de-stabilises loop. If we make the cut-off

// of our IIR filter high enough it will do a small amount of filtering with less phase-shift. Never use FIR filters in a control loop. The unity gain bandwidth

// of this loop is about 1Hz, so 14 times higher at 14Hz cut-off (-45 degrees phase shift there) gives us only -45/12 = -3.2 degrees phase shift at 1Hz - very roughly since it's not linear phase. You can move this up but best not to move it down in frequency.

float fcutg=fcut;

float taug=1/(2\*3.14\*fcutg);

float gnumg= (1-(2\*taug/dt));

float gdeng = (1+(2\*taug/dt));

float filt\_parg=gnumg/gdeng;

// gain is 1/ gdeng

float filt\_gaing=1/gdeng;

float filtg=0;

float filtgp=0;

float filt\_outg=0;

///////////////////////////////////////////

unsigned long loopStartTime = 0;

// ================================================================

// === INTERRUPT DETECTION ROUTINE ===

// ================================================================

volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone high

void dmpDataReady() {

mpuInterrupt = true;

}

// ================================================================

// === INITIAL SETUP ===

// ================================================================

void setup() {

// join I2C bus (I2Cdev library doesn't do this automatically)

#if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE

Wire.begin();

TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)

#elif I2CDEV\_IMPLEMENTATION == I2CDEV\_BUILTIN\_FASTWIRE

Fastwire::setup(400, true);

#endif

Serial.begin(115200);

while (!Serial);

Serial.println(F("Initializing I2C devices..."));

mpu.initialize();

Serial.println(F("Testing device connections..."));

Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

Serial.println(F("Initializing DMP..."));

devStatus = mpu.dmpInitialize();

mpu.setXGyroOffset(220);

mpu.setYGyroOffset(76);

mpu.setZGyroOffset(-85);

mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

if (devStatus == 0) {

Serial.println(F("Enabling DMP..."));

mpu.setDMPEnabled(true);

Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));

attachInterrupt(0, dmpDataReady, RISING);

mpuIntStatus = mpu.getIntStatus();

Serial.println(F("DMP ready! Waiting for first interrupt..."));

dmpReady = true;

packetSize = mpu.dmpGetFIFOPacketSize();

} else {

Serial.print(F("DMP Initialization failed (code "));

Serial.print(devStatus);

Serial.println(F(")"));

}

// for dc motor 1

pinMode(10, OUTPUT);

pinMode(9, OUTPUT);

pinMode(8, OUTPUT);

// dc motor 2

pinMode(7, OUTPUT);

pinMode(6,OUTPUT);

pinMode(5,OUTPUT);

////////////////////////////////////////////

// to find sampling time from a scope

pinMode(12,OUTPUT);

//////////////////////////////////////////////

// external Pots

// A0 for Pot Gain adjust.

pinMode(pot1Pin, INPUT);

// This one is for Pot offset for gyro pot into A1- an adjust for vertical so it doesn't lean

pinMode(pot2Pin,INPUT);

}

// ================================================================

// === MAIN PROGRAM LOOP ===

// ================================================================

void loop() {

if (!dmpReady) return;

while (!mpuInterrupt && fifoCount < packetSize) {

}

mpuInterrupt = false;

mpuIntStatus = mpu.getIntStatus();

fifoCount = mpu.getFIFOCount();

if ((mpuIntStatus & 0x10) || fifoCount == 1024) {

mpu.resetFIFO();

Serial.println(F("FIFO overflow!"));

} else if (mpuIntStatus & 0x02) {

while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

mpu.getFIFOBytes(fifoBuffer, packetSize);

fifoCount -= packetSize;

#ifdef OUTPUT\_READABLE\_YAWPITCHROLL

mpu.dmpGetQuaternion(&q, fifoBuffer);

mpu.dmpGetGravity(&gravity, &q);

mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

mpu.dmpGetGyro(gyro, fifoBuffer);

//Serial.print("ypr\t");

//Serial.print(ypr[0] \* 180/M\_PI);

// Serial.print("\t");

//Serial.print(ypr[1] \* 180/M\_PI);

//Serial.print("\n");

//Serial.println(ypr[2] \* 180/M\_PI);

#endif

//XXXXXXXXXXXXXXXXXXXXX Set Sampling rate to 100Hz: XXXXXXXXXXXXXXX

lastLoopUsefulTime = millis() - loopStartTime;

// millis() returns in milliseconds - time since program started

if (lastLoopUsefulTime < STD\_LOOP\_TIME) {

delay(STD\_LOOP\_TIME - lastLoopUsefulTime);

}

lastLoopTime = millis() - loopStartTime;

loopStartTime = millis();

//Serial.print(lastLoopUsefulTime);

//Serial.print(",");

//////////////////////////////////////////////////////

// Toggle this flag to measue sampling interval from pin 12. Only for debug

//timeflag=!timeflag;

//digitalWrite(12,timeflag);

/////////////////////////////////////////////////////////

//XXXXXXXXXXXXXXXXXXXXXX end of loop timing control XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Overall gain control via external pot

K = map(analogRead(pot1Pin), 0, 1024, 0, 255);

//offset Pot

offset = map(analogRead(pot2Pin), 0, 1024, 0, 255);

// angle of tilt in degrees - print it out the serial port and check when the gain is zero (to trim offset) or when it's running poperly.

currAnglep=currAngle;

currAngle = (double)((ypr[1] \* 57.29)+9+0.02\*offset); // 9 degrees is my offset but yours may be different. If you put a pot then the offset voltage from it can trim it out.

Serial.print(currAngle);

Serial.print("\n");

// now filter this to get a steadier reading. Ye cannae re-write the laws o physics captain. Dinnae filter too much o it will cause too much phase shift.

filtgp=filtg;

filtg=-filtgp\*filt\_parg+ filt\_gaing\*(currAngle + currAnglep);

angular\_ratep=angular\_rate;

angular\_rate = -((double)gyro[1]\*0.007633); // velocity of tilt - note minus sign due to gavity.

// now integrate angular\_rate to give a second estimate of angle from the Gyro. Use a simple Euler Integration but don't bother putting the sample interval since

// it can be omitted and the output scaled later - it is only a gain term after all. of the form new\_value=old\_value+dt\*input

est\_anglep=est\_angle;

est\_angle=constrain(est\_anglep+angular\_rate,-255,255);

////////////////////////////////////////////////////////

//Complementary Filter - we combine both estimates to stop drift problems. Mostly gyro but a bit of Accelerometer.

// first part is low pass and second is high-pass filter. Low pass filter the Gyro and high-pass filter the Accelerometer.dt=0.01second sampling interval

filtg=constrain((float)(aa\_const\*(filtg + angular\_rate\*dt) +(1- aa\_const)\*est\_angle),-255,255);

///////////////////////////////////////////////////////

// differentiate velocity to give acceleration which is another state. Simple crude Euler differentiator (first difference) used here is not a great idea. You should never

// use a pure differentiator since it amplifies noise and resonant frequencies. I low-pass filter the output therefore. first order only because 2nd ode gives too much phase-shift and de-stabilises the loop.

angular\_accelnp=angular\_acceln;

angular\_acceln=angular\_rate-angular\_ratep;

// now filter this with a low-pass filter

filtp=filt;

filt=-filtp\*filt\_par+angular\_acceln + angular\_accelnp;

filt\_out=filt\_gain\*filt;

////////////////////////////////////////////

// Apply state-feedback + integral action control law to Motors

Drive\_Motor(updateSpeed());

// Serial.print("speed: ");

//Serial.println(updateSpeed());

}

}

float updateSpeed() {

// K is overall gain potentiometer for trial and error setting up. can also act as a soft-start - start it at zero, hold it vertical and wind up the gain.

// at the same time you can tweek the offset pot so that it doesn't lean to on side and tries to run off.

// setpoint is zero - setpoint minus IIR filtered Gyro value

error=setpoint-filtg;

// integrated error is the fourth state

// now integrate angular\_rate to give a second estimate of angle from the Gyro. Use a simple Euler Integration but don't bother putting the sample interval since

// it can be omitted and the output scaled later - it is only a gain term after all. of the form new\_value=old\_value+dt\*input

integrated\_errorp= integrated\_error;

integrated\_error = constrain(integrated\_errorp + Ki\*error,-64,64);

// filtg is the filtered Gyro sensor IIR output to give a steadier reading - kind of inertia.

control\_out=K\*0.04\*(integrated\_error -(Kp\*filtg + Kd\*angular\_rate + Ka\*filt\_out));

return constrain(control\_out, -255,255);

}

float Drive\_Motor(float torque) {

//Serial.print("torque: ");

// Serial.println(torque);

if (torque >= 0) { // drive motors forward

digitalWrite(9, LOW);

digitalWrite(8, HIGH);

digitalWrite(7, LOW);

digitalWrite(6, HIGH);

} else { // drive motors backward

digitalWrite(9, HIGH);

digitalWrite(8, LOW);

digitalWrite(7, HIGH);

digitalWrite(6, LOW);

torque = abs(torque);

}

// PWM for the dc motors - same drive for both at pesent since no steering.

analogWrite(10,torque);

analogWrite(5,torque);

}