**Proyecto Final Control II:**

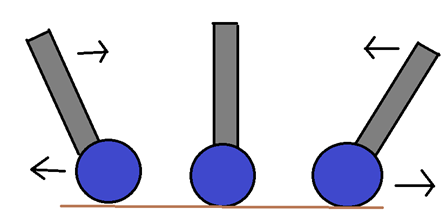
**Two-wheeled inverted pendulum**

Versión 4 - 06/07/18

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.

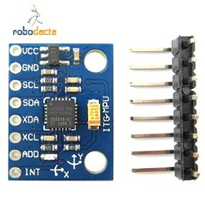
**OBJETIVO**

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***

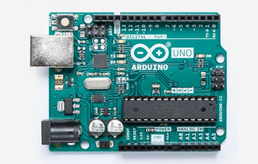
Se utilizó un motor DC el cual tiene un acople con las ruedas, estas son las más comunes para realizar proyectos con arduino. Estos motores soportan entre 3 y 12V (pero 6 a 8V está recomendado).



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

→ ***Arduino Uno***

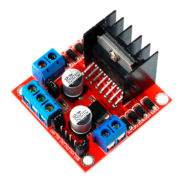
Un microcontrolador con una interfaz de desarrollo que resulta ser adecuado para la implementación del proyecto, por otro lado, resulta ser una de las placas más baratas y de facil entorno de programación con códigos abiertos.



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/

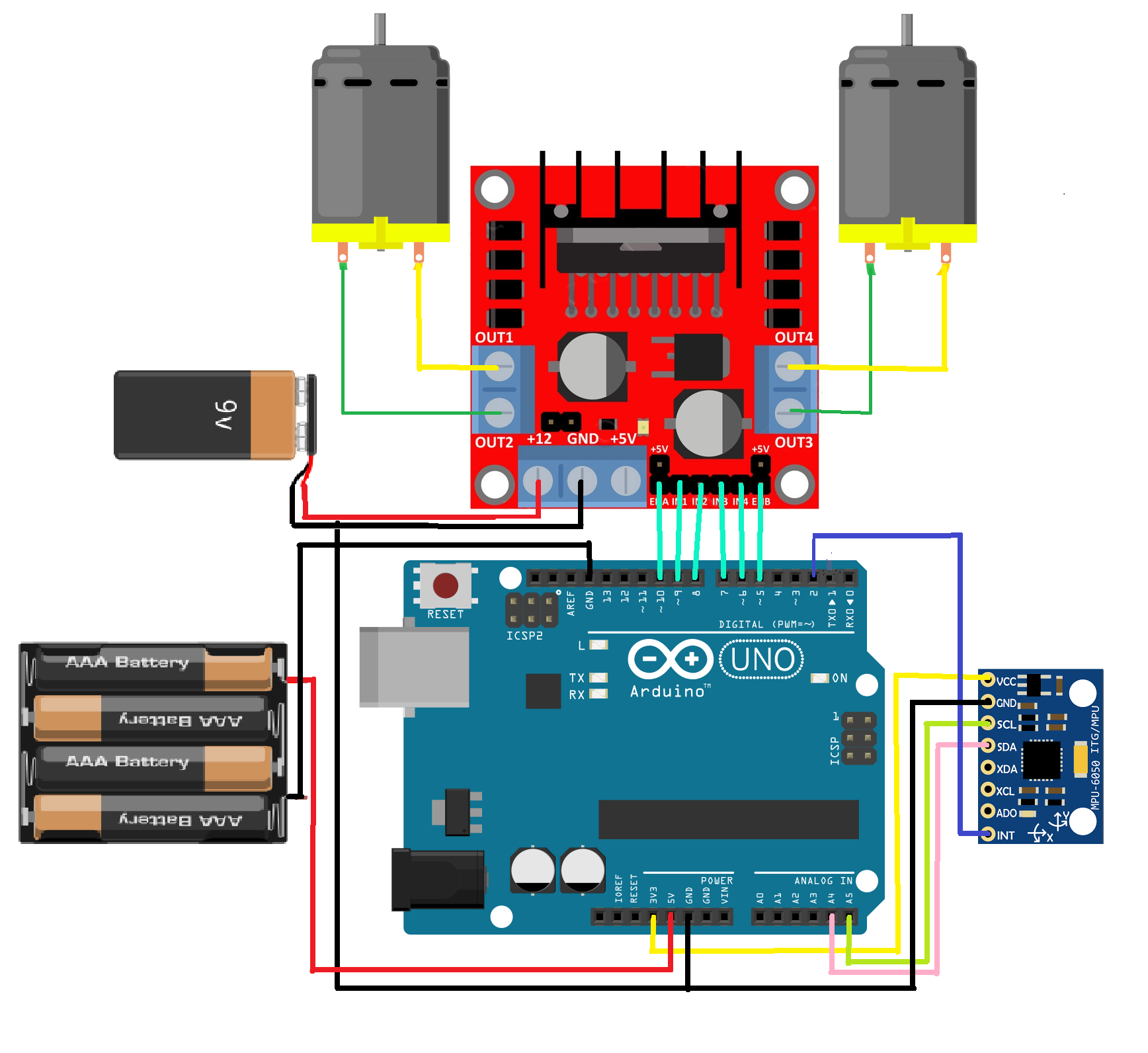
→ ***Otros componentes***

* 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.

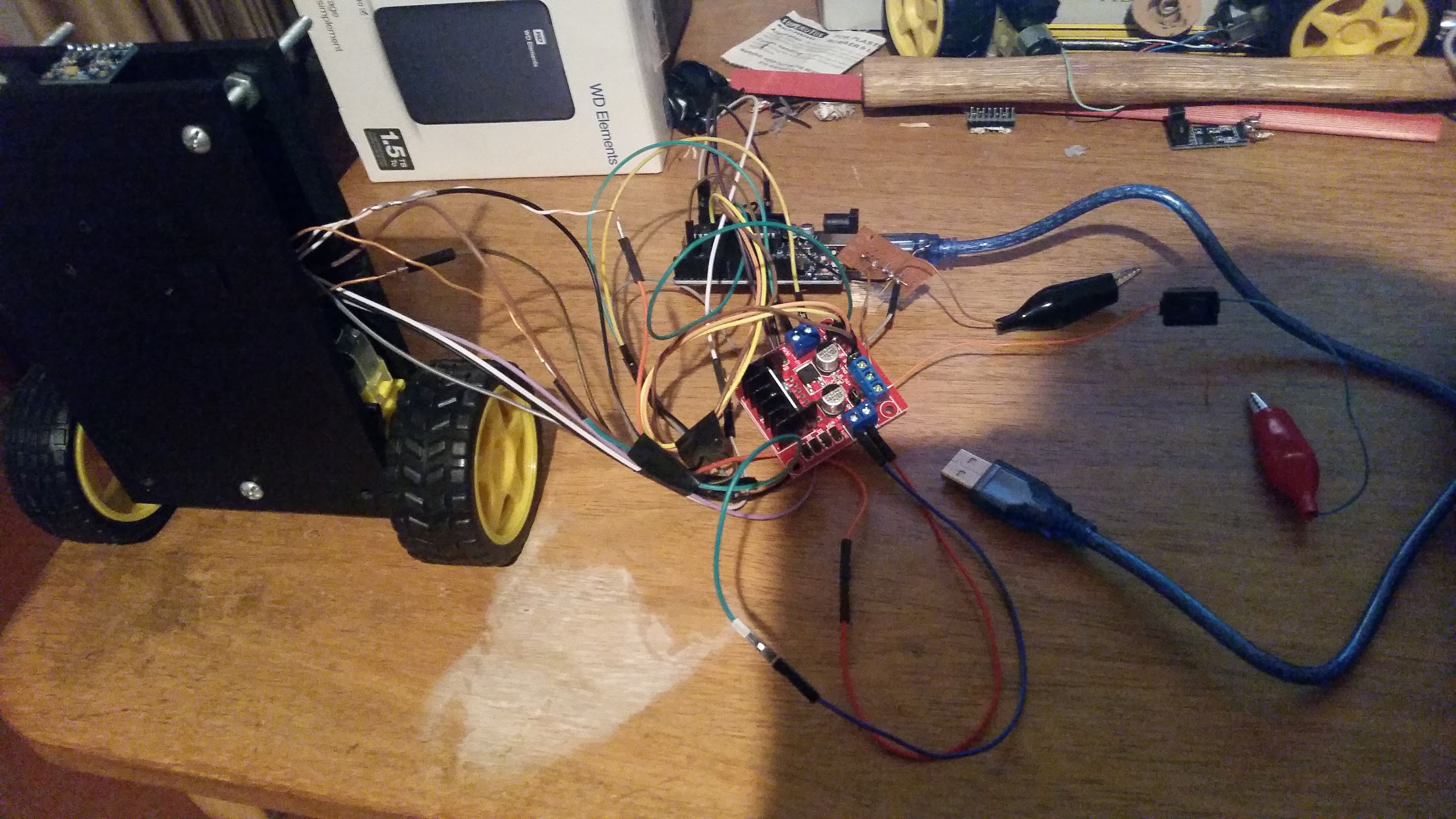
Las 4 baterías van al GND digital del Arduino (el negativo) y 5V de la parte analógica del Arduino (positivo de la batería).

***→ Armado Físico***

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



**Fuente:** Armado realizada por nosotros.



**Fuente:** Armado 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 utilizó, 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);

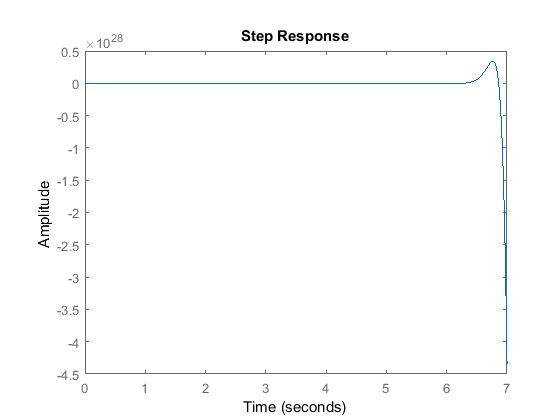
A=[0, 1, 0, 0;0, alpha, beta, -r\*alpha;0, 0, 0, 1;0, gamma, sigma, -r\*gamma];

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];



**Fuente:** Simulación del modelo matemático sin controlador realizada por nosotros.

1. **Simulación PID en matlab**

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

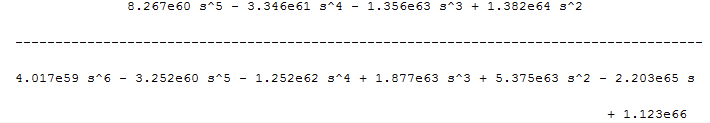
G=C\*(s\*I-A)^(-1)\*B;

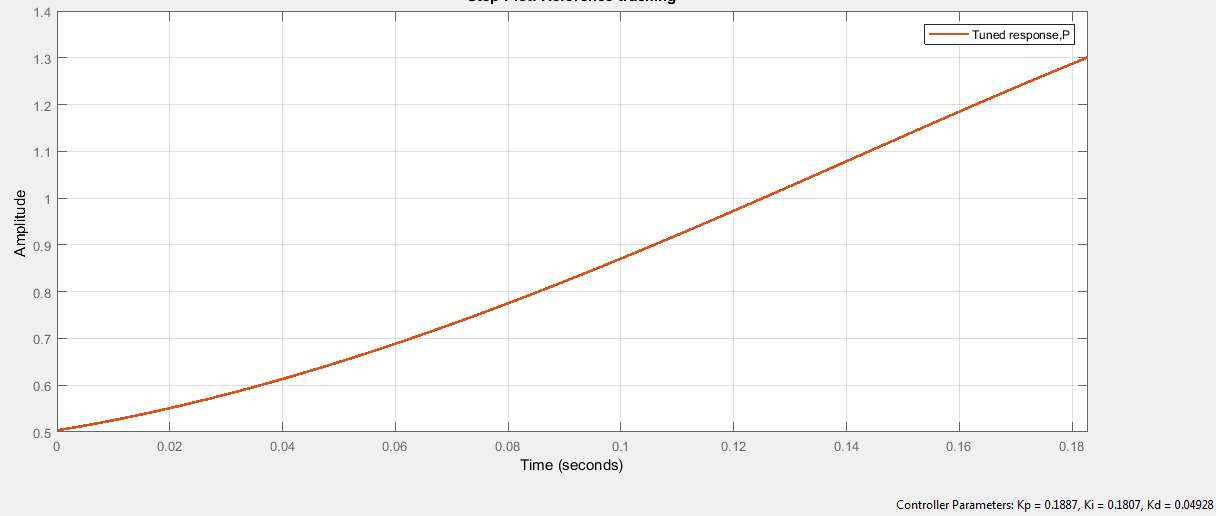
s=tf(‘s’);

step(G);

pidTuner %para tunear la funcion de transferencia G.

La función de transferencia salió de sexto grado





**Fuente:** Simulación del modelo matemático con controlador PID realizada por nosotros.

Donde: Kp=0.1887 , Ki=0.1807 y Kd=0.04928

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];

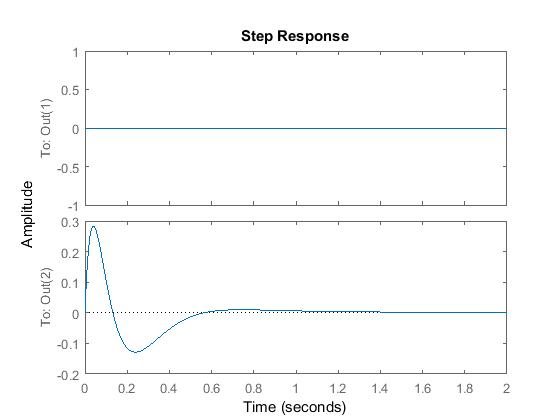
estados = {'x' 'x\_p' 'phi' 'phi\_p'};

entradas = {'r'};

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

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

step(sys\_ctl)



**Fuente:** Simulación con controlador LQR realizada por nosotros.

1. **Función de transferencia experimental**

Primero se mide la entrada y salida en el código para el autito. Este se copia y despus se puede seguir los pasos indicados. No se usó importar datos porque ocurría con error y usar los datos de la memoria directo era más rápido.

Datos=importdata('-pastespecial'); %importar workspace

Angulo=Datos(:,1); %Angulo

Voltaje=Datos(:,2); %Voltaje

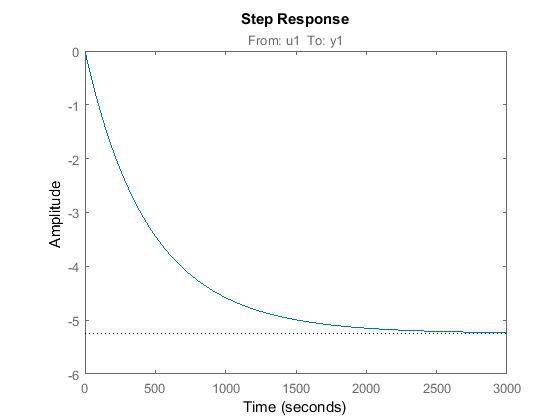
ident %TF experimental

step(tf1) %dibujar TF

pidTuner %mejorar TF

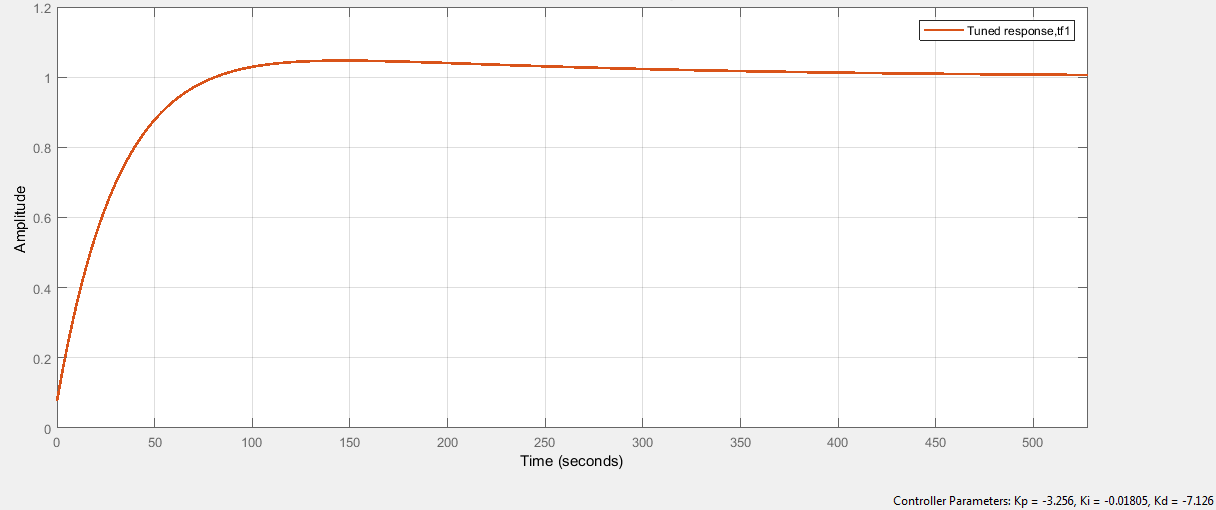
Con esto salieron la función de transferencia lineal:

y su comportamiento con STEP:



**Fuente:** Simulación sin controlador realizada por nosotros.

y con un controlador PID:



**Fuente:** Simulación con controlador PID realizada por nosotros.

Donde: Kp=-3.256 , Ki=-0.1805 y Kd=-7.126

Segun simulación el sistema no debería estabilizarse pero en la realidad si lo logra.

**RESULTADOS**

El autito se balancea por un corto tiempo. Desafortunadamente 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. Aún se tiene la incertidumbre de este efecto, el porqué de la falta de relación y la caída de voltaje existente.

**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.

Se recomienda usar otro tipo de microcontrolador como el raspberry.

Se recomienda averiguar a fondo todo acerca sobre el MPU, ya que este sensor resulta muy complicado de manejar, es un sensor no preciso, un sensor que presenta mucho ruido.

Para la parte eléctrica, tener en cuenta los cambios repentinos de corriente y voltaje, intentar tener fuentes estables.

**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 y 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 (Codigo)**

#include <PID\_v1.h>

#include <LMotorController.h>

#include "I2Cdev.h"

#include "MPU6050\_6Axis\_MotionApps20.h"

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

#include "Wire.h"

#endif

#define MIN\_ABS\_SPEED 20

MPU6050 mpu;

// MPU control/status vars

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

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

//PID

double originalSetpoint = 173;

double setpoint = originalSetpoint;

double movingAngleOffset = 0.1;

double input, output;

int moveState=0; //0 = balance; 1 = back; 2 = forth

// Aqui calibrar PID

double Kp = 16;

double Kd = 1.3;

double Ki = 8;

PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

// Aqui velocidad de los motores

double motorSpeedFactorLeft = 1;

double motorSpeedFactorRight = 1;

// Aqui cambiar los pines del motor

//MOTOR CONTROLLER

int ENA = 10;

int IN1 = 9;

int IN2 = 8;

int IN3 = 7;

int IN4 = 6;

int ENB = 5;

LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4, motorSpeedFactorLeft, motorSpeedFactorRight);

//timers

long time1Hz = 0;

long time5Hz = 0;

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

void dmpDataReady()

{

mpuInterrupt = true;

}

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

// initialize serial communication

// (115200 chosen because it is required for Teapot Demo output, but it's

// really up to you depending on your project)

Serial.begin(115200);

while (!Serial); // wait for Leonardo enumeration, others continue immediately

// initialize device

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

mpu.initialize();

// verify connection

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

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

// load and configure the DMP

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

devStatus = mpu.dmpInitialize();

// Aqui calibrar los valores del MPU en posicion calibrada

mpu.setXGyroOffset(852);

mpu.setYGyroOffset(58);

mpu.setZGyroOffset(-53);

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

//852, 58, -53

// make sure it worked (returns 0 if so)

if (devStatus == 0)

{

// turn on the DMP, now that it's ready

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

mpu.setDMPEnabled(true);

// enable Arduino interrupt detection

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

attachInterrupt(0, dmpDataReady, RISING);

mpuIntStatus = mpu.getIntStatus();

// set our DMP Ready flag so the main loop() function knows it's okay to use it

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

dmpReady = true;

// get expected DMP packet size for later comparison

packetSize = mpu.dmpGetFIFOPacketSize();

//setup PID

pid.SetMode(AUTOMATIC);

pid.SetSampleTime(10);

pid.SetOutputLimits(-255, 255);

}

else

{

// ERROR!

// 1 = initial memory load failed

// 2 = DMP configuration updates failed

// (if it's going to break, usually the code will be 1)

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

Serial.print(devStatus);

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

}

}

void loop()

{

// if programming failed, don't try to do anything

if (!dmpReady) return;

// wait for MPU interrupt or extra packet(s) available

while (!mpuInterrupt && fifoCount < packetSize)

{

//no mpu data - performing PID calculations and output to motors

pid.Compute();

motorController.move(output, MIN\_ABS\_SPEED);

}

// reset interrupt flag and get INT\_STATUS byte

mpuInterrupt = false;

mpuIntStatus = mpu.getIntStatus();

// get current FIFO count

fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)

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

{

// reset so we can continue cleanly

mpu.resetFIFO();

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

// otherwise, check for DMP data ready interrupt (this should happen frequently)

}

else if (mpuIntStatus & 0x02)

{

// wait for correct available data length, should be a VERY short wait

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

// read a packet from FIFO

mpu.getFIFOBytes(fifoBuffer, packetSize);

// track FIFO count here in case there is > 1 packet available

// (this lets us immediately read more without waiting for an interrupt)

fifoCount -= packetSize;

mpu.dmpGetQuaternion(&q, fifoBuffer);

mpu.dmpGetGravity(&gravity, &q);

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

//#if LOG\_INPUT

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

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

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

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

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

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

//#endif

input = ypr[1] \* 180/M\_PI + 180;

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

//Serial.print(input);

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

//Serial.print(output); // Aqui la salida de voltaje

Serial.print("\n");

}

}