ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

UNIDAD DE GESTIÓN DE TECNOLOGÍAS

CARRERA DE ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN & AVIÓNICA

AUTOR: SUNTAXI REIMUNDO JIMMY JEFFERSON

TEMA:

IMPLEMENTACIÓN DE UN VEHÍCULO DE DOS RUEDAS AUTO-EQUILIBRADO PARA PRÁCTICAS DE MICROCONTROLADORES

Objetivo General

Implementar un vehículo de dos ruedas auto-equilibrado utilizando Arduino para prácticas de microcontroladores

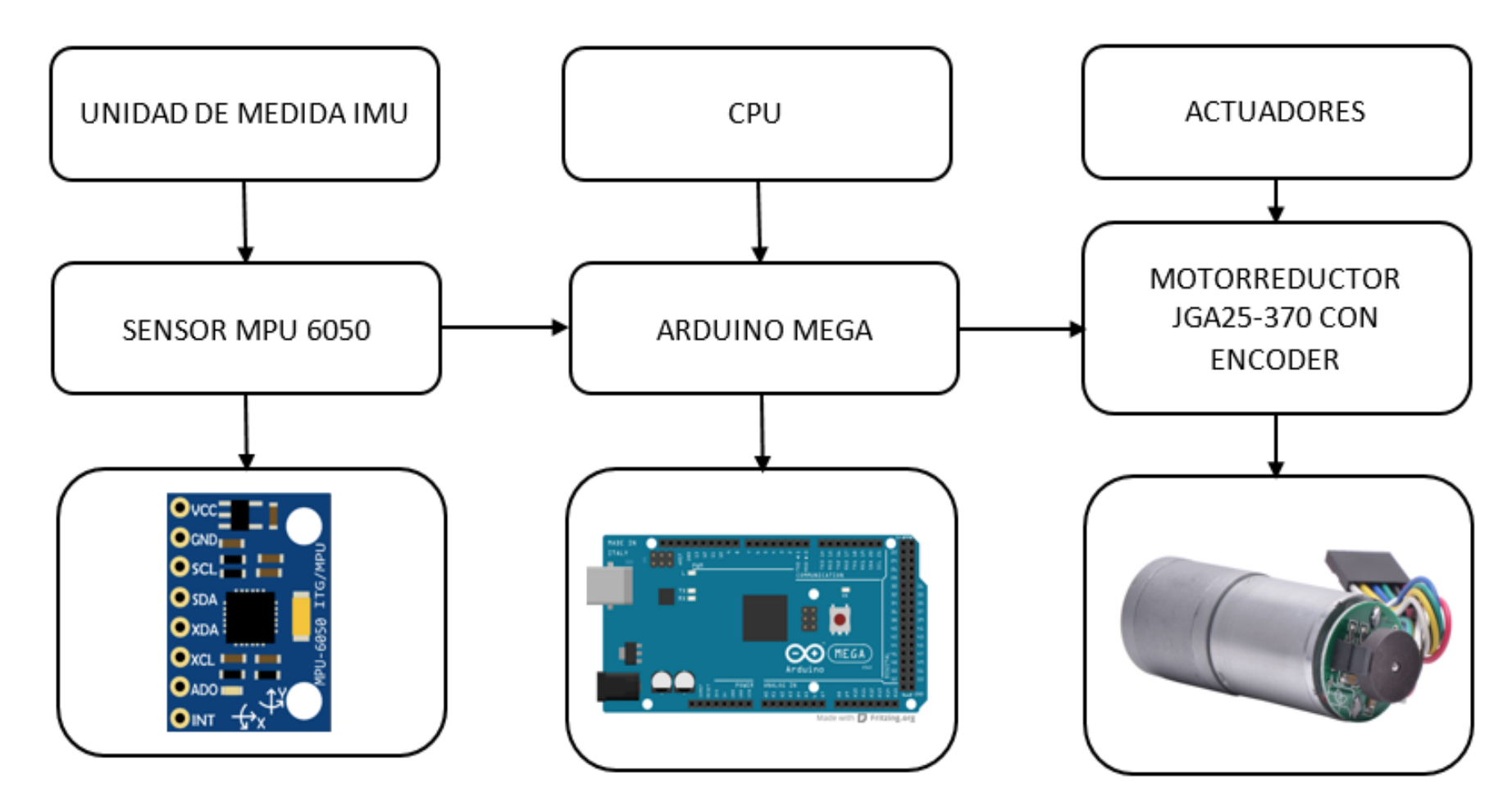
Objetivos Específicos

- Investigar el funcionamiento de una IMU mediante el análisis de sus características para una correcta aplicación en la implementación del vehículo.
- Implementar el algoritmo de control para el funcionamiento del vehículo a través de librerías existentes en la plataforma de programación.
- Realizar la sintonía PID para el control de equilibrio mediante diversos métodos de sintonización.

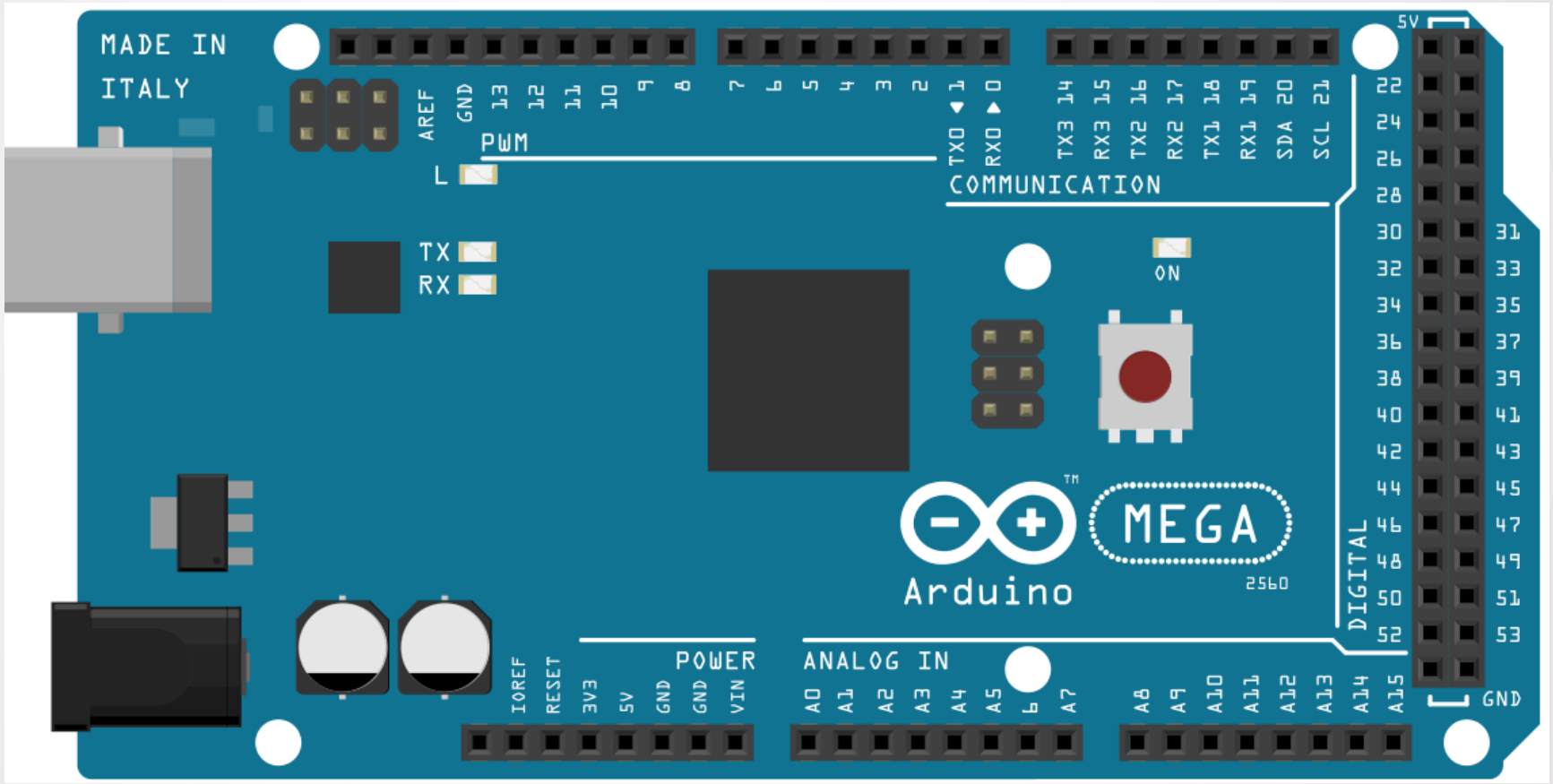
Requerimientos mínimos

- ARDUINO IDE (Software)
- ARDUINO Mega 2560
- MPU6050 (Giróscopo)
- MOTORREDUCTOR CON ENCODER
- SHIELD SainSmart (Puente H y sensores)
- CABLE DE PROGRAMACIÓN
- BATERIAS
- RUEDAS

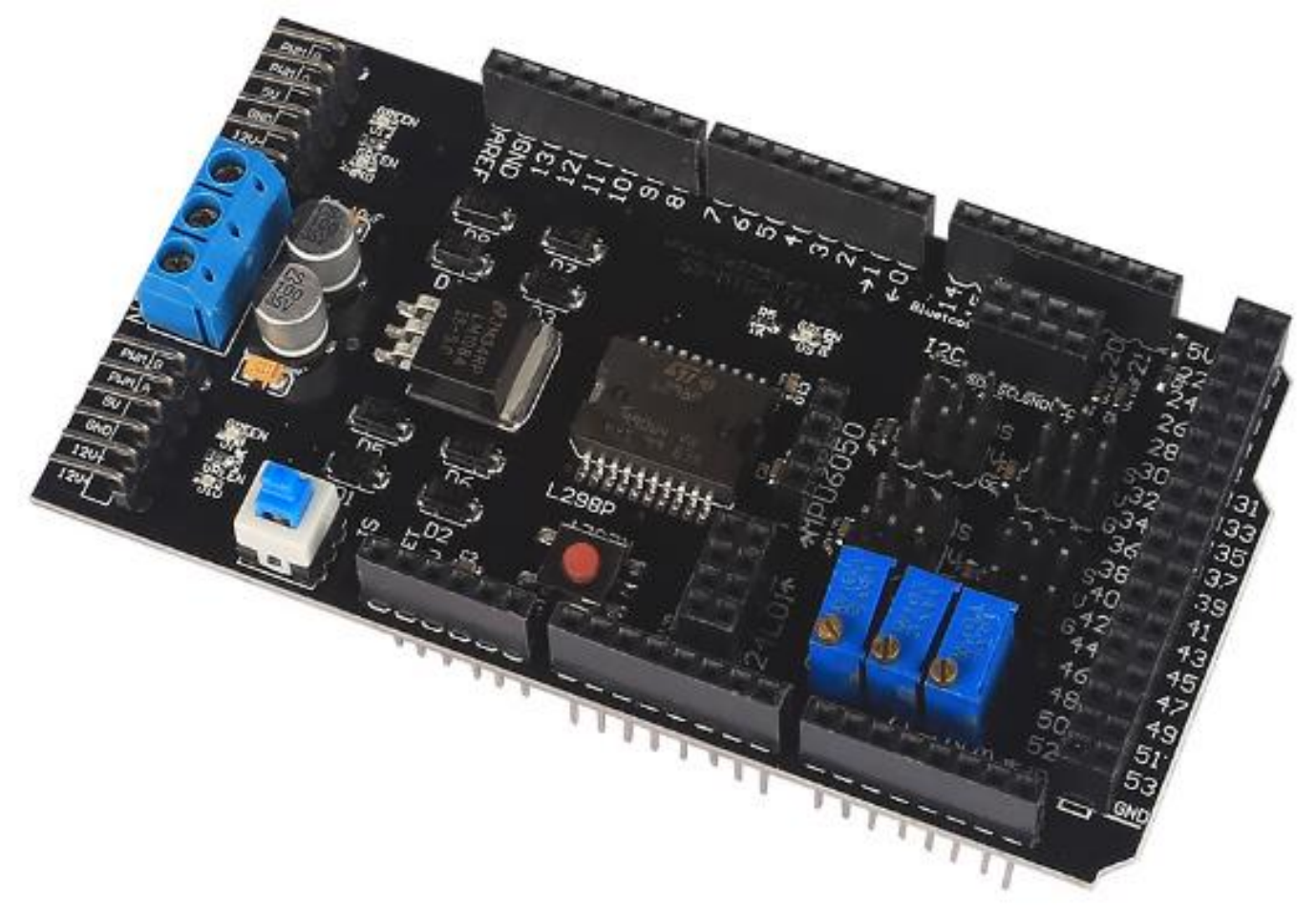
Diagrama de funcionamiento del vehículo



ARDUINO MEGA 2560



SHIELD SainSmart (Puente H y Sensores)



Puente-H (H-bridge)		
X	Y	SENTIDO DE GIRO
0	0	Freno libre
0	1	Giro Horario
1	0	Giro Anti - Horario
1	1	Freno brusco

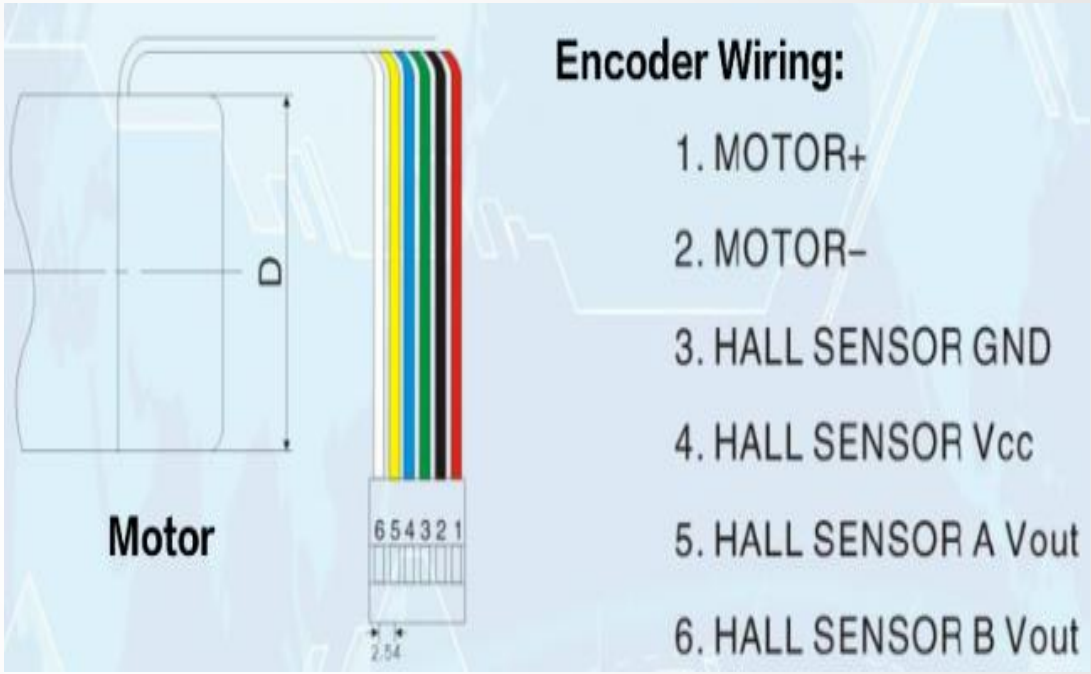
PIN SHIELD	NOMBRE	DESCRIPCIÓN
23	TN1	CONTROL MOTOR A-R
22	TN2	CONTROL MOTOR A-L
5	ENA	ENTRADA PWM A
24	TN3	CONTROL MOTOR B-R
25	TN4	CONTROL MOTOR B-L
4	ENB	ENTRADA PWM B

SHIELD MPU 6050

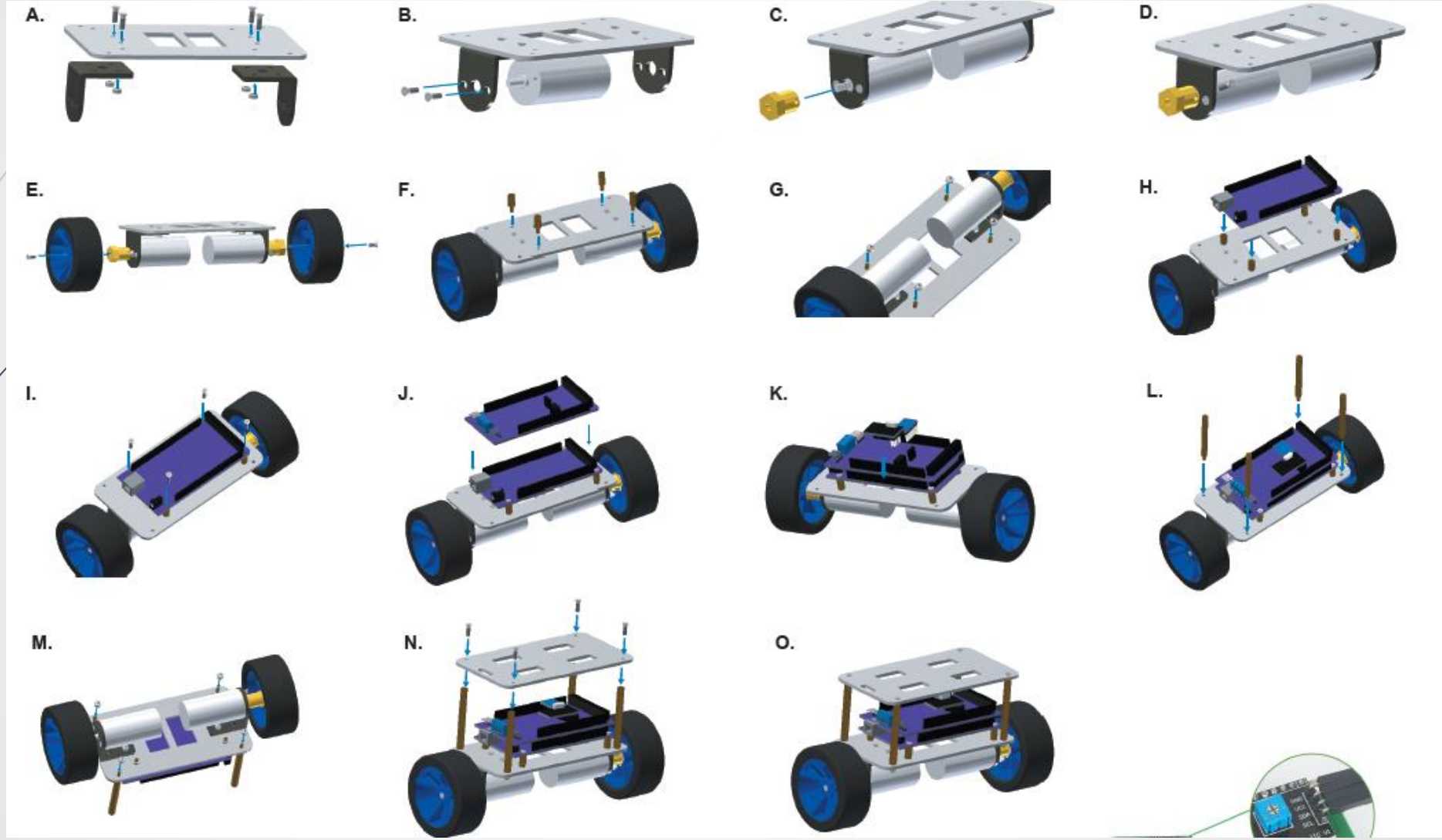
- **VCC:** Alimentación 5v
- **GNG:** Tierra
- **SCL:** I2C serial clock
- **SDA:** I2C serial data
- **XDA:** Auxiliar I2C serial data
- **XCL:** Auxiliar I2C serial clock
- **ADO:** Direcciona la comunicación I2C
- **INT:** Interrupción para ver Disponibilidad de datos

MOTORREDUCTOR JGA25-370 CON ENCODER

- Cuenta con un codificador integrado que ofrece una resolución de 12 pulsos por revolución.
- **Tensión nominal:** 12 V DC constante entre los terminales del motor.
- **Fuente de alimentación (Encoder):** fuente de alimentación regulada 5VDC.
- **Velocidad nominal de carga:** 160 rpm ± 10%.
- **Potencia:** 1,25 W.
- **Corriente de carga nominal:** 250 ± 5% mA.
- **Pesos:** Aproximadamente: 100g.
- **Vida:** Sobre 500H.


Encoder Wiring:
1. MOTOR+
2. MOTOR–
3. HALL SENSOR GND
4. HALL SENSOR Vcc
5. HALL SENSOR A Vout
6. HALL SENSOR B Vout
D
Motor
6 5 4 3 2 1
2.54

CONSTRUCCIÓN DEL VEHÍCULO



PARTES PROGRAMADAS DEL Sketch

```
1  #include <Wire.h> //LIBRERIA QUE PERMITE LA COMUNICACION I2C
2  #include <I2Cdev.h> //LIBRERIA QUE PERMITE CONECCION DEL SENSOR A ARDUINO
3  #include <MPU6050.h> // LIBRERIA DEL SENSOR
4
```

```
5  MPU6050 accelgyro; // CREACION DE UN OBJETO
6
7  int16_t ax, ay, az; //VARIABLES TIPO ENTERO
8  int16_t gx, gy, gz;
9
10 #define Gry_offset 0 //VALORES POR DEFAUL, SE LOS PUEDE CAMBIAR PARA PRUEBAS
11 #define Gyr_Gain 131
12 #define Angle_offset 0
13 #define RMotor_offset 0
14 #define LMotor_offset 0
15 #define pi 3.14159
16
17 float Angle_Delta, Angle_Recursive, Angle_Confidence; //VARIABLES TIPO FLOTANTE
```

```
7  int16_t ax, ay, az; //VARIABLES TIPO ENTERO
8  int16_t gx, gy, gz;
9
10 #define Gry_offset 0 //VALORES POR DEFAUL, SE LOS PUEDE CAMBIAR PARA PRUEBAS
11 #define Gyr_Gain 131
12 #define Angle_offset 0
13 #define RMotor_offset 0
14 #define LMotor_offset 0
15 #define pi 3.14159
16
17 float Angle_Delta, Angle_Recursive, Angle_Confidence; //VARIABLES TIPO FLOTANTE
18
19 float kp, ki, kd;
20 float Angle_Raw, Angle_Filtered, omega, dt;
21 float Turn_Speed = 0, Run_Speed = 0;
22 float LOutput, ROutput, Input, Output;
23
24 unsigned long preTime, lastTime; // VARIABLES LARGAS SIN SIGNO
25 float errSum, dErr, error, lastErr;
26 int timeChange;
27
28 float Sum_Right, Sum_Right_Temp, Sum_Left, Sum_Left_Temp, Distance, Distance_Right, Distance_Left, Speed;
29
30 int TN1 = 23; // SALIDA DIGITAL A,R
31 int TN2 = 22; // SALIDA DIGITAL A,L
32 int ENA = 5;  // SALIDA PWM A
33 int TN3 = 24; // SALIDA DIGITAL B,R
34 int TN4 = 25; // SALIDA DIGITAL B,L
35 int ENB = 4;  // SALIDA PWM B
```

PARTES PROGRAMADAS DEL Sketch

```
60    if (abs(Angle_Filtered) < 45)
61    {
62      omega = Angle_Raw = Angle_Filtered = 0;
63      Output = error = errSum = dErr = 0;
64      Filter();
65      myPID();
66    }
67    pinMode(TN1, OUTPUT);
68    pinMode(TN2, OUTPUT);
69    pinMode(TN3, OUTPUT);
70    pinMode(TN4, OUTPUT);
71    pinMode(ENA, OUTPUT);
72    pinMode(ENB, OUTPUT);
73    pinMode(18, INPUT);
74    pinMode(2, INPUT);
75
76    attachInterrupt(4, State_A, FALLING); // I
77    attachInterrupt(1, State_B, FALLING);
```

```
81  void loop()
82
83  {
84    while (1)
85    {
86
87      Filter();
88      if ((micros() - lastTime) > 10000)// CADA 10ms TOMA LOS VALORES
89      {
90
91        if (abs(Angle_Filtered) < 45)// ACTUA SIEMPRE QUE EL ANGULO SEA MENOR A 45 GRADOS
92        {
93          myPID();
94          PWMControl();
95        }
96        else //caso contrario parar motores
97        {
98          digitalWrite(TN1, HIGH);
99          digitalWrite(TN2, HIGH);
100         digitalWrite(TN3, HIGH);
101         digitalWrite(TN4, HIGH);
102       }
103       lastTime = micros();
104     }
105   }
106 }
```

```
127 void myPID()
128 {
129 kp = 22.000; //CONSTANTE PROPORCIONAL
130 ki = 1;     //CONSTANTE INTEGRAL
131 kd = 1.6;  //CONSTANTE DERIVATIVA
132
133   // CALCULAR LOS VALORES DE SALIDA DL PID
134   error = Angle_Filtered;
135   errSum += error; //errSum =errSum + error INCREMENTO DEL ERROR (INTEGRAL ES LA SUMA DE LOS ERRORES)
136   dErr = error - lastErr; // ERROR MENOS ERROR ANTERIOR
137   Output = kp * error + ki * errSum + kd * omega; //FORMULA DEL PID
138   lastErr = error;
139   noInterrupts(); //DESACTIVA INTERRUPCIONES
140   Sum_Right = (Sum_Right + Sum_Right_Temp) / 2; //PROMEDIO
141   Sum_Left = (Sum_Left + Sum_Left_Temp) / 2;
142   Speed = (Sum_Right + Sum_Left) / 2; //DETERMINA LA VELOCIDAD EN BASE AL MOVIMIENTO DER Y IQ
143   Distance += Speed + Run_Speed;
144   Distance = constrain(Distance, -300, 300);//LIMITA L DISTAACIOA DE 300 A -300
145   Output += Speed * 70 + Distance * 0.6;// VELOCIDAD DE GUIRO
146   Sum_Right_Temp = Sum_Right;
147   Sum_Left_Temp = Sum_Right;
148   Sum_Right = 0;
149   Sum_Left = 0;
150   ROutput = Output + Turn_Speed; //VELOCIDAD DE GUIRO R
151   LOutput = Output - Turn_Speed; //VELOCIDAD DE GUIRO L
152   interrupts(); //ACTIVAR INTERRUPCIONES
153 }
```

PARTES PROGRAMADAS DEL Sketch

```
155 void PWMControl()
156 {
157   if (LOutput > 0)
158   {
159     digitalWrite(TN1, HIGH);//ACTIVA MOTOR A R
160     digitalWrite(TN2, LOW);
161   }
162   else if (LOutput < 0)
163   {
164     digitalWrite(TN1, LOW); //ACTIVA MOTOR A L
165     digitalWrite(TN2, HIGH);
166   }
167   else
168   {
169     OCR3A = 0;//DESACTIVA CONTROL PWM TIMER 3
170   }
171   if (ROutput   0)
172   {
173     digitalWrite(TN3, HIGH); //ACTIVA MOTOR B R
174     digitalWrite(TN4, LOW);
175   }
176   else if (ROutput < 0)
177   {
178     digitalWrite(TN3, LOW); //ACTIVA MOTOR B L
179     digitalWrite(TN4, HIGH);
180   }
181   else
182   '
183     OCR0B = 0;//DESACTIVA CONTROL PWM TIMER 0
184   }
185   OCR3A = min(1023, (abs(LOutput * 4) + LMotor_offset * 4)); //MANDA SEÑAL PWM POR EL PIN5 MULTIMPLICA POR 4 POR LA RESOLUCION
186   OCR0B = min(255, (abs(ROutput) + RMotor_offset)); // MANDA SEÑAL PWM POR PIN 4
187 }
```

CONCLUSIONES

- El entorno de Arduino es multiplataforma que permite la integración de diferentes partes y sensores siendo versátil en el desarrollo de controladores embebidos.
- Las librerías son de gran ayuda ya que nos sirven como punto de partida para el desarrollo de proyectos más sofisticados.
- El MPU 6050 proporciona datos obtenidos del movimiento en los tres ejes de los cuales solo se utilizan datos de dos ejes debido a los grados de libertad del vehículo.
- Arduino es compatible con una amplia gama de sensores gracias a la plataforma de código abierto ya que no solo maneja estándar propietario sino que también ocupa muchos otros estándares abiertos.

CONCLUSIONES

- La comunicación I2C permite la conectividad entre más dispositivos mediante solo la utilización de dos líneas de datos y una de tierra como es el caso del MPU6050.

- La aplicación del método de ganancia límite para sintonía del PID permite establecer los tres términos de ajuste del controlador a partir del procedimiento de calibración establecido.

- Una correcta utilización del centro de gravedad favorece a la implementación del algoritmo de control.

RECOMENDACIONES

- Tener en cuenta los pines de polaridad de la Shield SainSmart ya que en caso de un corto circuito o mala polarización la tarjeta sufrirá daños irreparables al igual que Arduino Mega por el motivo que las dos se encuentran conectadas.

- Incluir las librerías necesarias a usar ya que Arduino no cuenta con algunas de ellas y son exclusivas de los fabricantes para lo cual se deberá descargarlas desde la página de los desarrolladores.

- Cambiar los parámetros de ganancia y offset del MPU6050 en la programación solo si presenta datos erróneos.

RECOMENDACIONES

- Tomar en cuenta el tipo de comunicación que poseen los dispositivos externos a utilizar ya que Arduino cuenta con la mayoría de interfaces de comunicación.

- Revisar que exista una conexión de referencia a tierra entre los dispositivos que encuentren intercomunicando físicamente.

- Realizar procedimientos de calibración adecuados para sintonizar el PID ya sea por tanteo, ganancia limite o ajuste automático.

- Tener cuidado de la posición del centro de gravedad al ubicar los dispositivos utilizados y así evitar que el vehículo pierda estabilidad.

Gracias