MANUAL MILIBOTS:

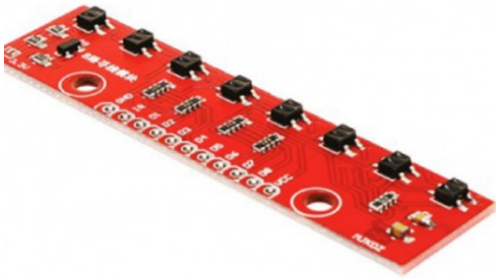
armado de velocista seguidor de línea

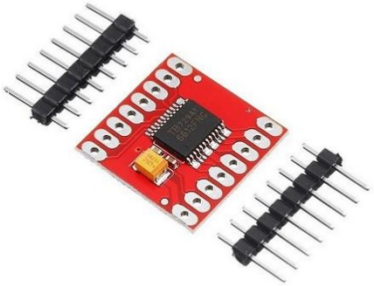
En la siguiente guía se explican los pasos necesarios para realizar un velocista seguidor de línea apto para competencia y con una base de materiales

Materiales:

los materiales mostrados en esta guía pueden llegar a tener alternativas o referencias distintas en el mercado que, por lo cual es importante consultar al docente o compañeros que cambios implicaría la selección de una variante de los componentes, para de esta manera saber si son aptos para el montaje.

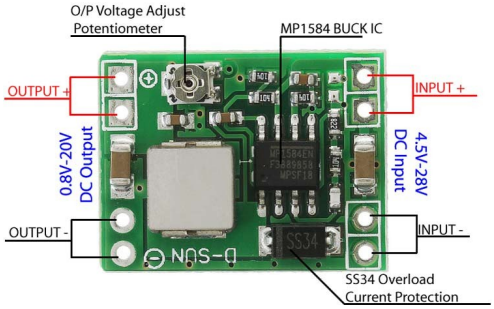
Arreglo de sensores de línea y Driver de motores tb6612:

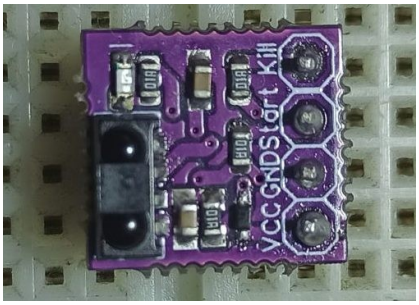




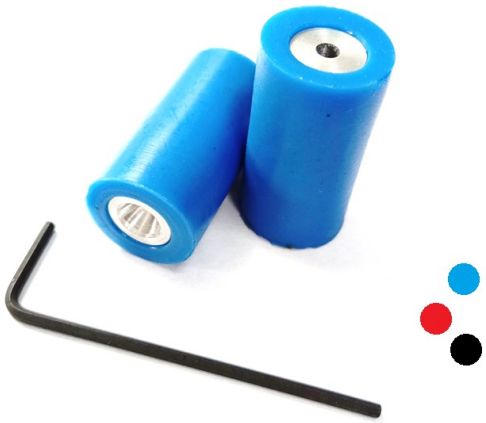
Batería tipo lipo de 7.4 v:

Conversor dc-dc: MP1584 y Modulo de inicio de competencia:

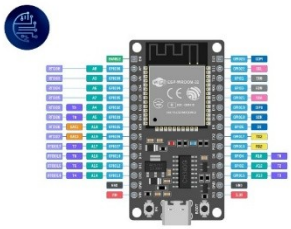




Motores n20 y llantas:



Esp32 con entrada tipo c:



PINOUT ESP32

TIPO C - 30 PINES

www.rheticspcb.com

SEMANA 1: INTRODUCCION

esta primera semana de trabajo está enfocada en conocer los componentes que serán utilizados en el montaje y sus respectivas funciones, así como principios básicos para las conexiones y cuidado de los componentes.

Objetivos:

- Comprender qué es un seguidor de línea.
- Conocer los componentes electrónicos y mecánicos.
- Instalar el entorno de programación (Arduino IDE) y preparar la placa ESP32.

Actividades:

1. Ver video o presentación de un seguidor de línea funcionando.
2. Revisar uno por uno los componentes:
   - **ESP32**: microcontrolador (el “cerebro” del robot).
   - **Motores N20 y llantas**: movilidad.
   - **Driver TB6612**: intermediario entre ESP32 y motores.
   - **Sensor QTR-8A (Pololu 960)**: detecta la línea negra sobre fondo blanco.
   - **Batería Lipo 7.4V** y conversor MP1584: fuente de alimentación.
   - **Módulo de inicio**: botón o sensor para comenzar una carrera.
3. Instalar Arduino IDE, configurar la placa ESP32:
   - Añadir ESP32 desde el **Administrador de Placas**.
   - Seleccionar puerto correcto (COM).

Código de prueba: blink (verificar comunicación)

```
void setup() {
pinMode(2, OUTPUT); // LED integrado del ESP32
}
void loop() {
digitalWrite(2, HIGH);
delay(500);
digitalWrite(2, LOW);
delay(500);
```

```
}
```

SEMANA 2: Armado físico del chasis

Objetivos:

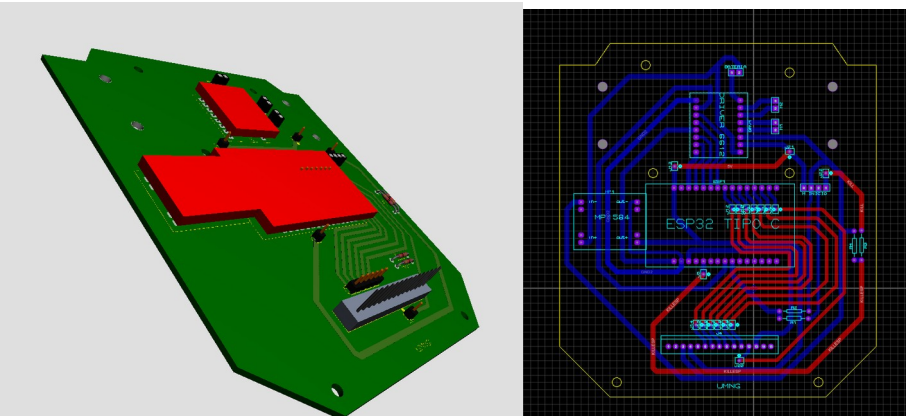
- Montar correctamente la estructura del robot.
- Alinear ruedas y fijar motores con seguridad.

Actividades:

1. Ensamblar chasis y colocar los motores.
2. Atornillar ruedas (llantas) a los ejes de los motores N20.
3. Colocar la batería en el chasis.
4. Asegurar buena distribución del peso.
5. Fijar (con soldadura) los componentes electrónicos **sin conectarlos aún a la fuente (bateria) directamente**.

Sin código esta semana. Se trata solo de la parte mecánica.

Se debe tener en cuenta que para esta semana se debe tener impresa la pcb para realizar la soldadura de los componentes, se recomienda usar regletas de pines para poder quitar y poner los componentes de una manera sencilla en aquellos que puedan ser propensos a presentar fallas, se puedan necesitar en otros proyectos, o que posiblemente deban ser probados sin el montaje en algún punto futuro(esp32, módulo de inicio, driver de motor).



SEMANA 3: Diseño CAD e impresión 3D de partes clave

Objetivos:

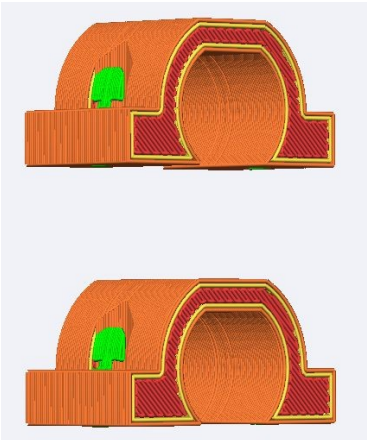
- Diseñar en CAD los soportes y piezas estructurales necesarias para fijar los sensores, motores y ruedas del robot, asegurando modularidad y facilidad de montaje.
- Fabricar mediante impresión 3D los componentes diseñados, validando su compatibilidad física con la PCB, motores y demás partes del chasis.

1. Soporte para sensor (8 sensores)

Diseño tipo marco para fijar la PCB del sensor reflectivo. Permite modificar la distancia al suelo y a los motores. Se fija con tornillos y tiene grabado “CHILLINGWINI / ADD” como referencia de montaje.

2. Soportes de motores

Soportes semicirculares con cavidad ajustada al motor y base de anclaje. Aseguran firmeza y evitan desplazamientos laterales. El diseño es robusto y fácil de imprimir.



3. Rines

Cilindros huecos con refuerzo interno, optimizados para imprimir sin soportes. Diseñados para ajustarse al eje del motor y acoplar llantas de silicona o goma.

SEMANA 4: Alimentación segura – prueba del conversor MP1584

Objetivos:

- Ajustar el conversor DC-DC MP1584 a 5V o 3.3V.
- Verificar tensión adecuada para ESP32 y sensores.

Actividades:

1. Conectar el MP1584 a la batería.
2. Usar un multímetro para ajustar la salida del MP1584 a 5V para la alimentación de la esp32 en su pin VIN, y demás componentes que deban funcionar a dicho voltaje, los componentes que funcionen con 3.3 voltios serán alimentados mediante la salida de voltaje 3.3 de la esp32.
3. Verificar que la ESP32 encienda conectada al conversor.

No se usa código esta semana, pero debe encender el LED del ESP32.

SEMANA 5: Conexión y prueba de motores con driver TB6612

Objetivos:

- Conectar el controlador de motores TB6612 al ESP32.
- Probar que los motores giren correctamente.

Conexiones típicas:

- **Motor A**: IN1, IN2, ENA (PWM)
- **Motor B**: IN3, IN4, ENB (PWM)

Alimentación:

- VM = 7.4V (del MP1584 o batería)
- VCC = 5V (lógica)
- GND común entre ESP32, batería y driver

Código de prueba:

```
#define IN1 27
#define IN2 26
#define ENA 14

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
}

void loop() {
  // Adelante
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
```

```
  analogWrite(ENA, 150);  // velocidad media

  delay(2000);

  // Atrás
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, 150);

  delay(2000);

  // Parar
  analogWrite(ENA, 0);
  delay(2000);
}
```

SEMANA 6: Prueba del sensor QTR-8A (8 sensores analógicos)

Objetivos:

- Leer los valores del arreglo de sensores analógicos QTR-8A.
- Determinar si el robot está desviado a izquierda o derecha.
- Solo imprimir en el monitor serial, **sin controlar motores aún**.

Conexiones:

- Cada sensor del QTR-8A a un pin analógico del ESP32 los cuales ya vienen distribuidos en la pcb.
- VCC: 5V (recomendado)
- GND común con ESP32

Nota:

- El QTR-8A usa **salida analógica**, valores de 0 a 4095 en ESP32 (12 bits).

Código de prueba – lectura + sugerencia de giro:

```
// Pines para cada sensor del QTR-8A
const int sensores[8] = {34, 35, 32, 33, 25, 26, 27, 14}; // Ajusta según tu cableado
int valores[8];

void setup() {
  Serial.begin(115200);
  for (int i = 0; i < 8; i++) {
    pinMode(sensores[i], INPUT);
  }
}
void loop() {
  int sumaIzq = 0, sumaDer = 0;

  for (int i = 0; i < 8; i++) {
```

```
    valores[i] = analogRead(sensores[i]);
    Serial.print("S");
    Serial.print(i);
    Serial.print(": ");
    Serial.print(valores[i]);
    Serial.print("  ");

    if (i < 4) sumaIzq += valores[i];  // sensores 0-3
    else sumaDer += valores[i];        // sensores 4-7
  }
  Serial.print(" | Giro: ");
  if (sumaIzq > sumaDer + 200) {
    Serial.println("Izquierda");
  } else if (sumaDer > sumaIzq + 200) {
    Serial.println("Derecha");
  } else {
    Serial.println("Centro");
  }
  delay(300);
}
```

Semana 7: Integración eléctrica completa

Objetivos:

- Conectar el montaje a la bateria (motores, sensores, alimentación, ESP32).
- Verificar que no haya cortocircuitos y todo funcione simultáneamente.

Actividades:

1. Fijar todos los componentes en el chasis.
2. Verificar conexión a GND común.
3. Encender el robot y observar comportamiento (sin movimiento aún).

Código combinado de prueba (solo sensores + LED):

```
void setup() {
  pinMode(2, OUTPUT); // LED del ESP32
}
void loop() {
  digitalWrite(2, HIGH);  // Encender LED
  delay(1000);
  digitalWrite(2, LOW);   // Apagar LED
  delay(1000);
}
```

SEMANA 8: Módulo de inicio de competencia + revisión final

Objetivos:

- Instalar el botón de inicio de competencia.
- Revisar que el robot esté completamente listo para cargar la lógica en semana 9.

Código de prueba para botón:

```
#define BOTON_INICIO 4
#define LED 2

void setup() {
  pinMode(BOTON_INICIO, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  if (digitalRead(BOTON_INICIO) == LOW) {
    digitalWrite(LED, HIGH);
    Serial.println("Inicio detectado");
  } else {
    digitalWrite(LED, LOW);
  }
  delay(100);
}
```

SEMANA 9: Código Chilinwini

Objetivos:

- Implementar el código de control para el robot velocista, integrando sensores infrarrojos, control PID y lógica de arranque mediante módulos externos y Bluetooth.
- Configurar y ejecutar la calibración automática de los sensores, permitiendo al robot identificar la línea a seguir en distintas condiciones de fondo y optimizar su seguimiento.

¿De qué trata el código? (Resumen explicado)

Este código permite que un robot velocista con ESP32 siga una línea negra utilizando 8 sensores infrarrojos. Se encarga de calibrar automáticamente los sensores, detectar la línea, calcular la posición relativa de esta línea respecto al centro del robot y ajustar la velocidad de los motores en tiempo real mediante un controlador PID. El robot puede arrancar o detenerse manualmente con pines de inicio o por comandos Bluetooth. También frena automáticamente si detecta una curva extrema.

Explicación completa (resumida)

- **Inicialización y calibración**:
  - Se definen los pines de sensores, motores y módulos de inicio.
  - Se calibra el fondo (blanco) y la línea (negro) usando el botón BOOT, y se calcula un umbral promedio para cada sensor.
- **Arranque y detención**:
  - El robot puede arrancar o detenerse mediante:
    - Un sistema físico de pines (START_PIN, KILL_PIN).
    - Comandos Bluetooth ('S' para iniciar, 'X' para detener).
- **Lectura de sensores**:
  - Se leen los valores analógicos de los sensores y se comparan con el umbral para convertirlos en valores digitales (1 si detecta línea, 0 si no).
- **Cálculo de posición**:

    - Calcula la posición de la línea como un valor ponderado entre 0 (izquierda) y 700 (derecha), usando los sensores activos.
- **Control PID**:
    - Aplica una fórmula PID para corregir el rumbo del robot ajustando las velocidades de los motores.
    - Incluye suavizado en la derivada, suma de errores para la integral y lógica para giros suaves o bruscos.
- **Frenado automático**:
    - Si detecta que la línea está muy a un lado, aplica freno diferencial para mejorar el giro.
- **Motores**:
    - Se manejan con PWM y control de dirección mediante pines H-Bridge.

Código completo (adaptarlo según su necesidad):

```
#include <BluetoothSerial.h>  // Bluetooth para ESP32
BluetoothSerial SerialBT;

#define BIN1 27
#define BIN2 26
#define PWMA 13
#define AIN1 14
#define AIN2 12
#define PWMB 5
#define blink 2
```

```
#define START_PIN 18 //Start MODULO DE INICIO
#define KILL_PIN 4   //Kill MODULO DE INICIO

#define CALIB_BUTTON 0 //Boton de la ESP32
#define VelocidadBase 240

const int sensorPins[8] = {25, 33, 32, 35, 34, 39, 36, 15};
int lectura_fondo[8];
int lectura_linea[8];
int umbral[8];
int valoresDigitales[8];
long sump = 0, sum = 0;
int pos = 0, poslast = 350;

//MAÑAS - TRUCOS
  int ultimaDireccion = 1; // 1 = derecha, -1 = izquierda

//
int setpoint = 350;
float KP = 0.3, KD = 15, KI = 0.2;//KP=0.1, KD=20
int vel = VelocidadBase;                    //Velocidad maxima
int veladelante = VelocidadBase;
int velatras = VelocidadBase;

int error1 = 0, error2 = 0, error3 = 0, error4 = 0, error5 = 0, error6 = 0;
```

```
int proporcional = 0, derivativo = 0, integral = 0, diferencial = 0, last_prop =
0;

volatile bool startSignal = false;

void verificarModuloInicio() {
  bool kill = digitalRead(KILL_PIN);
  bool start = digitalRead(START_PIN);

  if (kill && start && !startSignal) {
    startSignal = true;
    digitalWrite(blink, HIGH);  //  LED encendido: en marcha
    SerialBT.println(" Robot iniciado");
  }

  if (!kill && !start && startSignal) {
    startSignal = false;
    digitalWrite(blink, LOW);   //  LED apagado: detenido
    Motores(0, 0);              //  Frenar motores
    SerialBT.println(" Robot detenido");
  }
}

void setup() {

  SerialBT.begin("ESP32_Robot");
```

```
  Serial.begin(115200);
  pinMode(START_PIN, INPUT_PULLDOWN);
  pinMode(CALIB_BUTTON, INPUT_PULLUP);
  pinMode(blink, OUTPUT);

  for (int i = 0; i < 8; i++) pinMode(sensorPins[i], INPUT);

  DRIVER_init();
  Motores(0, 0);

  while (digitalRead(CALIB_BUTTON));
  for (int i = 0; i < 40; i++) { fondos(); parpadeo(); delay(50); }
  parpadeo(); delay(1000); parpadeo();

  while (digitalRead(CALIB_BUTTON));
  for (int i = 0; i < 40; i++) { linea(); parpadeo(); delay(50); }
  promedio();

  SerialBT.println("Calibraci\xF3n completada.");
}

void loop() {
  //DESCOMENTAREAR PARA USAR BLUETOOTH
  if (SerialBT.available()) {
  char c = SerialBT.read();
  if (c == 'S') startSignal = true;
```

```
  else if (c == 'X') startSignal = false;  // 'X' para detener
}

  verificarModuloInicio();
  if (startSignal) {
    leerSensoresDigitales();
    calcularPosicion();
    frenos();
    PID();
  }
}

void fondos() {
  for (int i = 0; i < 8; i++) lectura_fondo[i] = analogRead(sensorPins[i]);
}

void linea() {
  for (int i = 0; i < 8; i++) lectura_linea[i] = analogRead(sensorPins[i]);
}

void promedio() {
  for (int i = 0; i < 8; i++) {
    umbral[i] = (lectura_fondo[i] + lectura_linea[i]) / 2;
    SerialBT.print(umbral[i]); SerialBT.print("\t");
  }
  SerialBT.println();
```

```
}

void leerSensoresDigitales() {
  Serial.print("Lecturas:\t");
  for (int i = 0; i < 8; i++) {
    int lectura = analogRead(sensorPins[i]);
    valoresDigitales[i] = (lectura > umbral[i]) ? 1 : 0;
    Serial.print(lectura); Serial.print("/");
    Serial.print(valoresDigitales[i]); Serial.print("\t");
  }
  Serial.println();
}

void calcularPosicion() {
  sump = sum = 0;
  for (int i = 0; i < 8; i++) {
    sump += valoresDigitales[i] * (700 - i * 100);
    sum  += valoresDigitales[i];
  }

  if (sum == 0) {
    // No se detectó línea
    pos = (ultimaDireccion > 0) ? 700 : 0;
  } else {
    pos = sump / sum;
    // Actualizamos dirección solo si hay buena lectura
```

```
    ultimaDireccion = (pos > setpoint) ? 1 : -1;
  }

  poslast = pos;
}

void PID() {
  proporcional = pos - setpoint;

float alpha = 0.4;  // entre 0.1 (más suave) y 0.5 (más rápido)
derivativo = alpha * (proporcional - last_prop) + (1 - alpha) * derivativo;

  integral = error1 + error2 + error3 + error4 + error5 + error6;
  last_prop = proporcional;
  error6 = error5; error5 = error4; error4 = error3;
  error3 = error2; error2 = error1; error1 = proporcional;

  diferencial = (proporcional * KP) + (derivativo * KD) + (integral * KI);
  diferencial = constrain(diferencial, -vel, vel);
  if (abs(proporcional) > 350) {
  diferencial *= 10;  //  Aumenta la fuerza del giro
  }

  int velocidad_actual = vel;  // ⚙ Base
  if (abs(proporcional) > 250 && abs(proporcional) <= 350)
  velocidad_actual = vel - 50;  //  Curva moderada
```

```
  else if (abs(proporcional) > 350)
  velocidad_actual = vel - 100;  //  Curva cerrada


  if (diferencial < 0)
    Motores(velocidad_actual, velocidad_actual + diferencial);//DEBE GIRAR A LA IZQUIERDA
  else
    Motores(velocidad_actual - diferencial, velocidad_actual);//DEBE GIRAR A LA DERECHA
}

void frenos() { //350 pos medio linea en la mitad - 0 linea a la izquierda - 750 linea a la derecha
  if (pos <= 200)
  Motores(-(veladelante+90), veladelante);
  else if (pos >= 500)
    Motores(veladelante, -(veladelante+90));
}

void Motores(int left, int right) {
  MotorIz(left); MotorDe(right);
}

void MotorIz(int value) {
  digitalWrite(AIN1, value >= 0 ? HIGH : LOW);
  digitalWrite(AIN2, value >= 0 ? LOW  : HIGH);
```

```
  ledcWrite(0, abs(value));
}

void MotorDe(int value) {
  digitalWrite(BIN1, value >= 0 ? LOW : HIGH);
  digitalWrite(BIN2, value >= 0 ? HIGH : LOW);
  ledcWrite(1, abs(value));
}

void parpadeo() {
  digitalWrite(blink, HIGH); delay(100);
  digitalWrite(blink, LOW); delay(100);
}

void DRIVER_init() {
  pinMode(AIN1, OUTPUT); pinMode(AIN2, OUTPUT);
  pinMode(BIN1, OUTPUT); pinMode(BIN2, OUTPUT);
  ledcSetup(0, 20000, 8); ledcAttachPin(PWMA, 0);
  ledcSetup(1, 20000, 8); ledcAttachPin(PWMB, 1);
}
```