# 2.5- Ajustar parámetros para mejorar la detección y estabilidad del sistema.

## 1. Ajustes en la Detección de Sensores

La precisión es fundamental. Se realizaron las siguientes calibraciones:

### 1.1. Calibración del Sensor Ultrasónico (HC-SR04)

- **Problema Identificado:**

  Las mediciones de distancia iniciales del sensor no eran precisas, mostrando una desviación consistente respecto a las mediciones reales.

- **Análisis:**

  Se observó que el sensor medía consistentemente un poco menos de la distancia real y que este error aumentaba de forma proporcional a la distancia. Esto descartó un error de offset simple y apuntó a un error de escala.

- **Parámetro Ajustado:**

  `FACTOR_CALIBRACION_SONIDO`.

- **Solución Implementada:**

  Se realizó una calibración empírica midiendo distancias reales vs. las reportadas por el sensor. Se calculó un factor de corrección multiplicativo promediando la relación entre ambos valores, resultando en un `FACTOR_CALIBRACION_SONIDO` de **1.03**. Este factor se aplicó a todas las lecturas del sensor, mejorando significativamente su precisión.

### 1.2. Calibración del Sensor de Color (TCS34725)

- **Problema Identificado:**

  La detección de colores era poco fiable y muy sensible a la iluminación ambiental.

- **Análisis:**

  Se determinó que los valores RGB crudos no eran suficientes para una detección robusta. La normalización de los valores (dividiendo por el canal de brillo "Clear") y el establecimiento de umbrales específicos eran necesarios.

- **Parámetros Ajustados:**

- Umbrales de componentes de color normalizados.
- Umbral del canal de brillo (`c`) para la detección de "Negro".

- **Solución Implementada:**

  Se desarrolló una rutina de calibración donde se expuso el sensor a los colores objetivo (Rojo, Verde, Azul, Negro) bajo las condiciones de luz de prueba.

  Se registraron los valores normalizados y se definieron umbrales precisos en el código para cada color. Además, se ajustó el umbral de brillo a `c < 150` para una detección fiable de la superficie negra.

## 2. Ajustes en la Estabilidad del Movimiento

Un robot que se mueve de forma errática es ineficaz. Los siguientes ajustes estabilizaron el chasis:

### 2.1. Corrección de Rumbo con IMU (MPU9250)

- **Problema Identificado:**

  El robot se desviaba de una trayectoria recta debido a diferencias mecánicas (fricción, potencia) entre los dos motores.

- **Análisis:**

  El movimiento en lazo abierto (enviar el mismo valor de PWM a ambos motores) era insuficiente para un avance recto.

- **Parámetro Ajustado:**

  Ganancia Proporcional (`Kp`).

- **Solución Implementada:**

  Se implementó un controlador Proporcional (P) que utiliza el ángulo de Yaw del IMU para corregir la desviación.

  El `targetYaw` define el rumbo deseado, y el controlador ajusta diferencialmente la velocidad de las ruedas para minimizar el error.

  El parámetro `Kp` fue ajustado experimentalmente a **2.0**, logrando un equilibrio entre una reacción rápida y un movimiento estable sin oscilaciones.

## 3. Ajustes en la Lógica de Navegación

La "inteligencia" del robot va en su lógica de decisión. Se refinó de la siguiente manera:

### 3.1. Eliminación de Bucles de Detección en Maniobras

- **Problema Identificado:**

  Al detectar un color (por ejemplo, "Rojo"), el robot giraba, pero al seguir viendo el mismo color, volvía a ejecutar la misma maniobra en un bucle infinito, sin lograr escapar de la zona.

- **Análisis:**

  El `loop()` principal era "sin estado", re-evaluando las condiciones inmediatamente después de una acción.

- **Parámetro Ajustado:**

  La estructura del programa, introduciendo una variable de estado (`estadoActual`).

- **Solución Implementada:**

  Se refactorizó el código a una máquina de estados simple con dos estados: `NAVEGANDO` y `EJECUTANDO_MANIOBRA`.

  El robot solo busca nuevos estímulos cuando está en `NAVEGANDO`. Al detectar un color que requiere una acción, cambia a `EJECUTANDO_MANIOBRA`, realiza la secuencia completa (girar y avanzar para escapar), y solo al finalizar regresa al estado `NAVEGANDO`.

## 3.2. Corrección de la Lógica de Giros

- **Problema Identificado:**

  Las acciones para los colores Rojo y Azul estaban invertidas; el robot giraba a la derecha para el Rojo y a la izquierda para el Azul.

- **Análisis:**

  Hubo un error lógico al asignar el signo del ángulo a la maniobra deseada.

- **Parámetro Ajustado:**

  El signo del ángulo (`grados`) pasado a la función `girarGrados()`.

- **Solución Implementada:**

  Se invirtieron los signos en las llamadas a la función:

  - `girarGrados(90)` para el giro a la izquierda (Rojo).
  - `girarGrados(-90)` para el giro a la derecha (Azul).

## 3.3. Gestión del Rumbo Objetivo (`targetYaw`)

- **Problema Identificado:**

  Después de solucionar otros problemas, se notó que el robot había dejado de corregir su rumbo de forma proactiva.

- **Análisis:**

  La línea `targetYaw = currentYawAngle;` se estaba ejecutando en cada ciclo del `loop()` durante el avance, lo que causaba que el robot aceptara constantemente su propia desviación como el "nuevo rumbo correcto", resultando en un error de 0 y, por lo tanto, ninguna corrección.

- **Parámetro Ajustado:**

  La lógica de actualización de `targetYaw`.

- **Solución Implementada:**

  Se eliminó la reinicialización de `targetYaw` de los estados de avance continuo.

  Ahora, el `targetYaw` solo se actualiza intencionalmente al inicio del programa o después de una maniobra de giro, permitiendo que el controlador P trabaje constantemente para mantener un rumbo fijo a largo plazo.

## 4. Conexiones de Sensores

Respecto a las conexiones se siguieron las mismas instrucciones del repositorio de GitHub.

Lo único distinto fue realizar una conexión compartida en la protoboard, ya que el MPU y el sensor RGB comparten dos puertos: `SCL` y `SDA`.

Por lo tanto:

- Se conectaron esos 2 puertos (`SCL` y `SDA`) desde el Arduino hacia la protoboard.
- Luego, tanto el MPU como el sensor RGB se conectaron a la protoboard en paralelo, compartiendo correctamente el bus I2C.

Se utilizó la siguiente referencia para los pines:

## Sensor RGB TCS34725 (conexión en Arduino UNO):

| Pin del TCS34725 | Arduino UNO | Función |
|---|---|---|
| VIN | 5V | Alimentación |
| GND | GND | Tierra común |
| SCL | A5 | I2C reloj |
| SDA | A4 | I2C datos |
| INT (opcional) | No conectar | Interrupción (opcional) |

## Sensor MPU6500 (conexión en Arduino UNO):

| Arduino UNO | MPU-6500 | Descripción |
|---|---|---|

| | | |
|---|---|---|
| 5V | VCC | Alimentación (o +3.3V) |
| GND | GND | Tierra |
| A4 | SDA | Datos I2C |
| A5 | SCL | Reloj I2C |
| GND | AD0 | Tierra |