



UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño de Robot Asistente para Hospitales

T E S I S

PARA OBTENER EL TÍTULO DE

Ingeniero Eléctrico Electrónico

PRESENTA:

García Martínez Alan

TUTOR

Dr. Víctor Manuel Lomas Barrié



México, CDMX, Coyoacán, C.U.

2022

Jurado asignado:

Sitio en donde se desarrolló el tema:

Instituto de Investigaciones de Matemáticas Aplicadas y en Sistemas.

Laboratorio de Róbótica.

Ciudad Universitaria.

Universidad Nacional Autónoma de México.

Dr. Víctor Manuel Lomas Barrié

Asesor de Tesis

Los resultados y avances de esta tesis, se presentaron en:

Agradecimientos

Dedico mi esfuerzo y conocimiento a las personas que me han apoyado a lo largo de mi formación académica y formación profesional para ser la persona con la que me propuse ser un día.

Un Agradecimiento a mi madre quien me ha apoyado en multiples ocasiones para poder completar mi educación.

Un agradecimiento a mi Padre quien con su apoyo logre persistir a través de estos años para no desistir y lograr todas mis metas.

*Alan García Martínez
Álvaro Obregón Ciudad de México, 2022.*

*Cada libro, cada tomo que ves, tiene alma.
El alma de quien lo escribió,
y el alma de quienes lo leyeron y vivieron y soñaron con él*

-La sombra del viento. Carlos Ruiz Zafón.

*El futuro nunca lo vi,
se convirtió en ayer,
cuando intentaba alcanzarlo.*

-Esperanza. José Emilio Pacheco.

Índice general

Índice general	X
Índice de figuras	XI
Índice de tablas	XIII
1 Resumen	1
2 Antecedentes	3
2.1 Planteamiento del problema	3
2.2 Importancia de la róbotica para la sociedad actual.	4
3 Objetivos e hipótesis	5
3.1 Objetivo General	5
3.2 Objetivos Específicos.	5
3.2.1 Caracterización de un robot móvil con diseño omnidireccional.	5
3.2.2 Diseño del tren motriz para navegación del robot.	5
3.2.3 Diseño del algoritmo para la navegación del robot móvil.	6
3.2.4 Implementación de tópicos de ROS para el robot móvil.	6
4 Marco teórico	7
4.1 Sistema motriz de robots móviles	7

4.1.1	Rueda Mecanum	7
4.1.2	Movimiento Omnidireccional de un robot móvil	8
4.1.3	Driver controlador de motores de DC modelo Talon SR	10
4.1.4	Motor DC modelo AM802-001A	11
4.2	Microcontrolador Raspberry PI Pico	12
4.3	Raspberry PI 4 MODEL B	14
4.4	Sensores de Distancia	16
4.4.1	Sensor Ultrasonico HC-SRF05	17
4.5	Sistemas de control de robots móviles	18
4.6	ROS	18
5	METODOLOGÍA	19
5.1	Tren motriz del robot móvil	19
5.2	Protección de la electrónica de potencia	19
5.3	Sensores de distancia y retroalimentación	19
5.4	Algoritmo de control móvil	19
5.5	Configuración para el controlador Talón SR	19
5.6	Diagrama de Flujo para el algoritmo de navegación	20
5.7	Diagramas de conexión	20
5.7.1	Propuesta de Diseño del circuito.	20
6	Resultados y discusión	23
7	Conclusiones	25
8	Códigos realizados	27
Bibliografía		37

Índice de figuras

4.1	Rueda Mecanum	7
4.2	Direcciones para una plataforma compuesta de 4 ruedas mecanum.	8
4.3	Tipos de movimientos con una plataforma omnidireccional. Imagen obtenida de: www.researchgate.net	9
4.4	Driver controlador de motores Talon SR.	10
4.5	Raspberry Pi Pico.	11
4.6	Raspberry Pi Pico.	12
4.7	Driver controlador de motores Talon SR.	13
4.8	Driver controlador de motores Talon SR.	13
4.9	Sensor ultrasonico modelo HC-SR04	14
4.10	Características del dispositivo Raspberry PI 4 Model B. Imagen disponible en www.raspberrypi.com	15
4.11	Software Imager disponible para Windows y Linux	16
4.12	Sistemas operativos disponibles Raspbian.	16
4.13	Sistemas operativos disponibles Linux.	16
4.14	Sensor ultrasonico modelo HC-SRF05	17
5.1	Propuesta de los diagramas de conexión.	21

Índice de tablas

4.1 Caracterización de movimientos.	10
5.1 Caracterización de movimientos.	20

- CAPÍTULO 1

Resumen

Las necesidades de cubrir los servicios y productos han llevado a la sociedad a buscar nuevas formas de innovar y modernizar la logística para la producción de productos de todo tipo y colaborar en la participación de distintos servicios en favor de la sociedad moderna, consolidado por la industria 4.0 implicando consigo mayor productividad y eficiencia entre otras ventajas como la reducción de márgenes de error y procesos precisos y eficientes reduciendo tiempos de producción y costos de operación.

Con lo anterior en mente se implementó el diseño de control de un robot móvil haciendo uso de múltiples herramientas matemáticas y el diseño de algoritmo para la navegación adecuada con el fin de poder ser implementado en los hospitales para la asistencia a personal médico profesional para lograr optimizar múltiples procesos.

CAPÍTULO 2

Antecedentes

“La robótica se define como una ciencia que reúne diferentes campos tecnológicos, con el principal objetivo de diseñar máquinas robotizadas capaces de realizar diferentes tareas automatizadas en función de la capacidad de su software.

-EDS Robotics.

“Un robot es una máquina capaz de extraer información de su ambiente y usa el conocimiento acerca de su mundo para moverse de manera segura para realizar un propósito específico.

-1942, Isaac Asimov.

Sección 2.1.

Planteamiento del problema

La pandemia del COVID-19 durante el año 2019 nos planteó una nueva visión sobre la situación sanitaria que implicó la alta tasa de los contagios que se suscitaron durante los primeros meses de la pandemia, por lo que al tiempo de realizar análisis y estudios en las áreas encargadas para atender el COVID-19 se propuso utilizar robots para la asistencia médica en áreas específicas y automatizar tareas específicas dentro de los hospitales con el propósito de reducir el contagio entre el personal médico y los pacientes, sin embargo se requieren proponer nuevos diseños y protocolos para los robots dentro de los hospitales para poder atender en capacidad diferentes situaciones sanitarias que se puedan presentar más adelante para mejorar la atención médica.

Sección 2.2.

Importancia de la robótica para la sociedad actual.

Los constantes avances de la tecnología generan cambios en nuestra sociedad en la forma en la que llevamos a cabo nuestras actividades cotidianas, facilitando tareas que solían ser más complicadas en años anteriores. La búsqueda por optimizar y automatizar las tareas han llevado la búsqueda la innovación y evolución en los diferentes métodos de software y hardware para satisfacer la necesidad de realizar tareas automatizadas en las últimas décadas.

La robótica no solo ha ayudado a mejorar los tiempos de producción continua y de calidad para industrias de todo tipo como por ejemplo la automotriz, farmacéutica, constructora entre muchas otras industrias que emplean los brazos robóticos para realizar tareas complejas. Sin embargo no solo en las industrias han sido beneficiadas por los avances en robótica, ya que dentro del área médica los brazos robóticos han sido una herramienta de bastante demanda debido a la capacidad de llevar a cabo complejas operaciones en pacientes. Nos referimos a sistemas de brazos robóticos como el modelo Zeus y el modelo DaVinci que tomaron mucha relevancia en hospitales para realizar diferentes operaciones.

En lo que respecta a el área médica y en particular la cirugía robótica, se han tenido progresivos avances enfocados a la robótica utilizadas en las cirugías que presentan una mayor complicación de realizar, siendo considerada como por muchos autores como el futuro en las cirugías ya que su desarrollo ha sido rápido y ha demostrado poseer numerosas ventajas que ayudan a la mejora de las técnicas quirúrgicas gracias a el desarrollo de la industria 4.0 y el IOT.

De igual manera gracias a la cirugía robótica se han producido cambios en la práctica y en la enseñanza de la cirugía que cada vez son más comunes, a continuación, exploraremos técnicas y conceptos de la cirugía robótica.

CAPÍTULO 3

Objetivos e hipótesis

Sección 3.1.

Objetivo General

Diseñar un sistema de control para la navegación de un robot móvil sobre una plataforma omnidireccional que cumpla con funciones de asistente enfermero para hospitales designados como apoyo y atención a pacientes de COVID-19.

Sección 3.2.

Objetivos Específicos.

3.2.1 Caracterización de un robot móvil con diseño omnidireccional.

Se busca incorporar en la navegación de un robot móvil a partir de 4 ruedas mecanum para la navegación del robot asistente para lograr optimizar procesos de navegación dentro de áreas médicas específicas.

3.2.2 Diseño del tren motriz para navegación del robot.

Se propondrá un diseño en base a un circuito considerando aspectos como alimentación, protección contra picos de corriente y control del movimiento del robot móvil en base a un algoritmo de navegación y un esquema de conexiones eléctronicas.

3.2.3 Diseño del algoritmo para la navegación del robot móvil.

Se busca diseñar e implementar con la electrónica un algoritmo de programación en lenguaje de C para el microcontrolador Raspberry PI Pico para controlar el tren motriz del robot.

3.2.4 Implementación de tópicos de ROS para el robot móvil.

Se planteará implementar en conjunto con las herramientas proporcionadas por ROS2 org en uso de la distribución de ROS2 Foxy, tópicos para la comunicación serial entre la Raspberry PI4 y la raspberry Pi Pico.

CAPÍTULO 4

Marco teórico

Sección 4.1.

Sistema motriz de robots móviles

4.1.1 Rueda Mecanum

La rueda omnidireccional o rueda Mecanum a diferencia de un rueda normal tiene la característica de tener un cubo de rueda y un rodillo. La función del cubo es ser el soporte principal de la rueda mientras que el rodillo es un tambor montado en el tubo como se muestra en la figura 4.1.



Figura 4.1: Rueda Mecanum

El eje del cubo de la rueda omnidireccional y el eje del rodillo son perpendiculares entre sí, mientras que el eje del cubo de la rueda Mecanum y el eje del rodillo están en un ángulo de 45 °.

Esto permite que las ruedas se muevan en dos direcciones y se muevan holonómicamente, asegurando que puedan moverse instantáneamente en cualquier dirección.

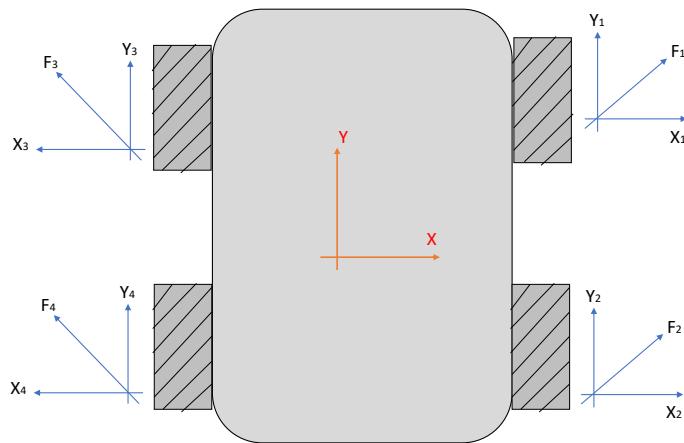


Figura 4.2: Direcciones para una plataforma compuesta de 4 ruedas mecanum.

Como se observa en la figura 4.2, al tener una dirección de giro en la rueda, por ejemplo en el eje "Y", tendremos una componente horizontal sobre un eje "X", y en consecuencia también tendremos una fuerza resultante en un ángulo a 45° .

4.1.2 Movimiento Omnidireccional de un robot móvil

El movimiento omnidireccional en la aplicación de los robots móviles es cada vez más común debido a la gran versatilidad y ventajas que propone este sistema de movimientos para los robots móviles representando una mayor libertad de movimiento para desplazarse.

Como se detalló anteriormente, el movimiento omnidireccional se debe a el uso de 4 ruedas mecanum pueden viajar en cualquier dirección desde cualquier ángulo sin girar de antemano permitiendo desplazamientos horizontales, verticales, diagonales y girar sobre un eje de referencia. Como se muestra en la Figura 4.2 tendremos 10 principales formas de movimiento.

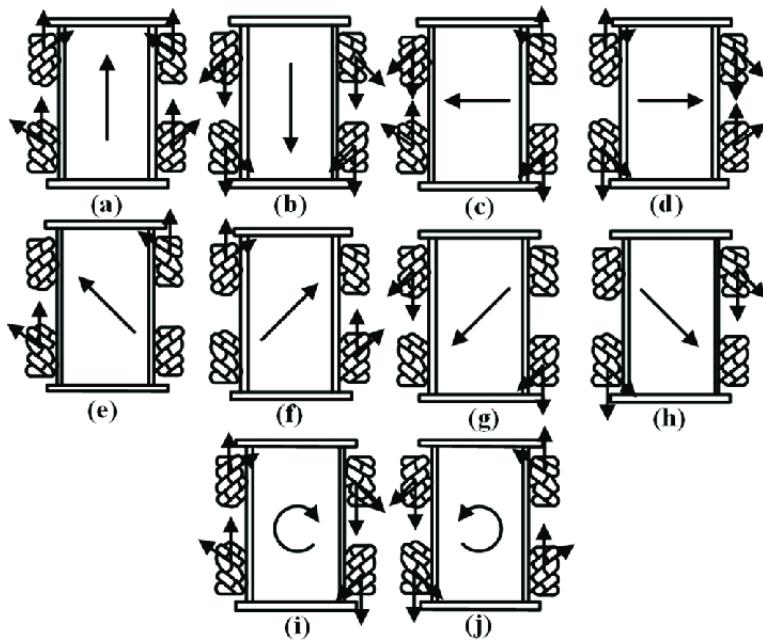


Figura 4.3: Tipos de movimientos con una plataforma omnidireccional. Imagen obtenida de: www.researchgate.net

De la Figura 4.2 a. Tenemos un movimiento hacia delante, el cual consiste principalmente en tener todas las ruedas hacia una misma dirección.

De la Figura 4.2 b. Tenemos un movimiento de reversa, con la diferencia de que es en sentido contrario al sentido del movimiento delantero.

De la Figura 4.2 c y d. Tenemos un movimiento horizontal hacia el lado izquierdo. Los movimientos horizontales tienen la característica de que los sentidos de giro están determinados por que las ruedas correspondientes al lado izquierdo y derecho deben girar en sentidos contrarios al modo horario y antihorario deseados. Es decir que en este caso tendremos las ruedas del lado izquierdo girando de manera que ambas direcciones coincidan en un vector de movimiento resultante a la dirección izquierda y derecha respectivamente.

De la Figura 4.2 e, f, g, h. Son movimientos con vectores resultantes en diagonal, resultado de tener funcionando únicamente dos ruedas omnidireccionales en sentido diagonal hacia una misma dirección, para obtener en cada caso el desplazamiento de movimiento deseado.

De la Figura 4.2 i y j. Tenemos un movimiento de giro en 360° para la plataforma del robot omnidireccional, para lograr este sentido de giro las ruedas de lado izquierdo deben seguir una misma dirección de avance, mientras que por el contrario las ruedas del lado derecho deben ir en dirección contraria para lograr este giro de 360° .

Para la caracterización de los movimientos anteriormente mencionados tomaremos como referencia el movimiento y el sentido de giro que deben tener.

Movimiento	Rueda 1	Rueda 2	Rueda 3	Rueda 4
ALTO	Sin movimiento	Sin movimiento	Sin movimiento	Sin movimiento
ADELANTE	Sentido Horario	Sentido Horario	Sentido Horario	Sentido Horario
ATRAS	Sentido Anti horario	Sentido Anti horario	Sentido Anti horario	Sentido Anti horario
DERECHA	Sentido Horario	Sentido Anti Horario	Sentido Anti horario	Sentido Horario
IZQUIERDA	Sentido Anti horario	Sentido Horario	Sentido Horario	Sentido Anti horario
DIAGONAL D1	Sentido Horario	Sin movimiento	Sin movimiento	Sentido Horario
DIAGONAL D2	Sin movimiento	Sentido Anti horario	Sentido Anti horario	Sin movimiento
DIAGONAL I1	Sin movimiento	Sentido Horario	Sentido Horario	Sin movimiento
DIAGONAL I2	Sin movimiento	Sentido Anti horario	Sentido Anti horario	Sin movimiento
Giro en sentido anti horario	Sin Horario	Sentido Horario	Sentido Anti horario	Sentido Anti horario
Giro en sentido horario	Entido Anti horario	Sentido Anti horario	Sentido Anti Horario	Sentido Horario

Tabla 4.1: Caracterización de movimientos.

4.1.3 Driver controlador de motores de DC modelo Talon SR

El driver controlador para motores de corriente directa modelo Talon SR, que se muestra en la siguiente figura. Es un dispositivo que controla la dirección de giro de los motores que utilizaremos para definir los movimientos de nuestro robot.



Figura 4.4: Driver controlador de motores Talon SR.

El funcionamiento de este dispositivo es controlar la velocidad y rotación de un motor de corriente directa mediante una señal modulada por ancho de pulsos o conocida como PWM para determinar la velocidad y sentido de giro del motor.

Como se observa en la figura 4.4, el talon principalmente tiene como características una terminal de lado izquierdo con polaridad para alimentar el dispositivo con un rango de voltaje de 6 a 28 Volts. Tiene capacidad para soportar una corriente de hasta 36 Amperes. Tiene una terminal de 3 pines de los cuales tendremos una terminal con referencia a tierra y una para la señal PWM con la que controlaremos el dispositivo.

En el centro tenemos un dissipador de temperatura al que se le puede adaptar un ventidialor. De lado derecho tenemos la terminal de salida donde se conecta al motor de corriente directa. Tendremos un cable de color verde y rojo respectivamente para identificar el sentido de dirección de giro del motor. y una intermitencia de color naranja que indica que no está actuando el motor, ya sea por la señal de modulación enviada o por que no recibe esta señal. Adicionalmente este dispositivo puede ser calibrado para ser adaptable a un microcontrolador al que se desea ocupar. Sin embargo este dispositivo tiene la característica de necesitar un intervalo de 1-2 milisegundos.

4.1.4 Motor DC modelo AM802-001A



Figura 4.5: Raspberry Pi Pico.

El motor de corriente directa modelo AM802-001A, es un motor con las siguientes características.

- Voltaje nominal de alimentación a 12 [V]
- 5310 RPM
- Capacidad de corriente de 133 [A]
- Corriente de arranque de 2.3 [A]
- Potencia máxima de 377 [W]

Sección 4.2.

Microcontrolador Raspberry PI Pico

El dispositivo Raspberry Pi Pico es un microcontrolador basado en un chip RP2040, que brinda un alto rendimiento, bajo costo y facilidad de uso debido a que permite entornos de desarrollo en Micropython y en C/C++.

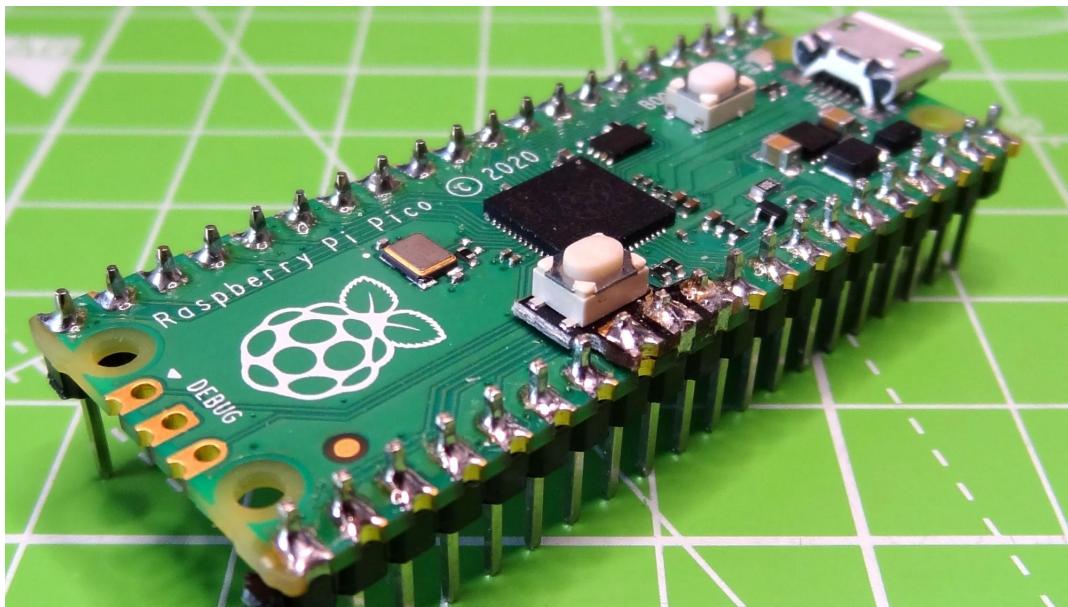


Figura 4.6: Raspberry Pi Pico.

Este dispositivo ademas de tener un costo relativamente bajo de 4 dolares, es bastante versitral ya que tiene un total de 43 pines, los cuales se detallan en la siguiente figura.

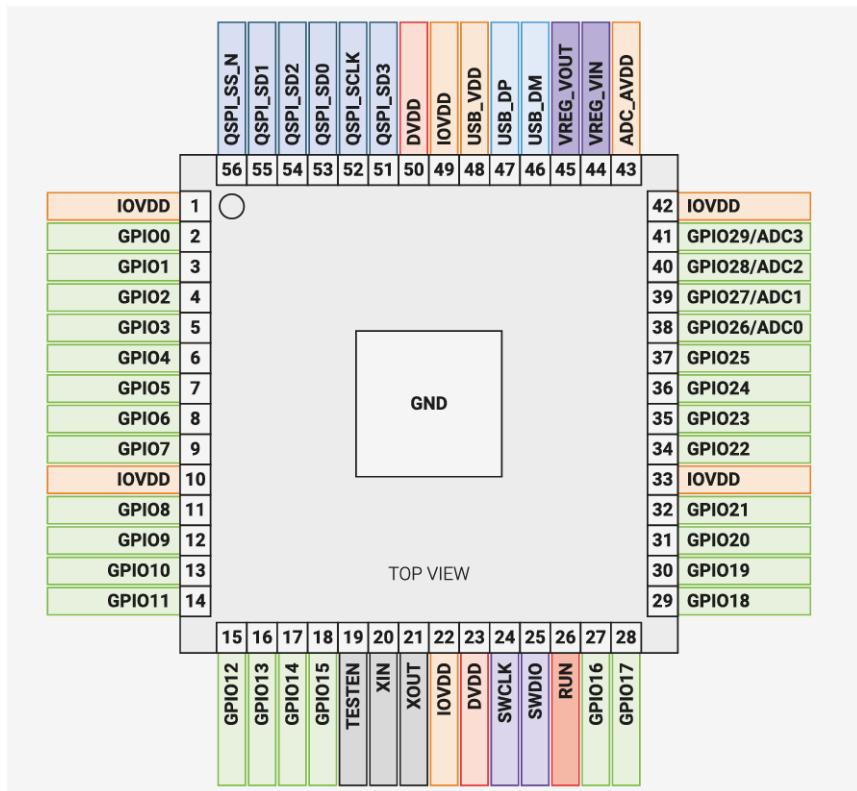


Figura 4.7: Driver controlador de motores Talon SR.

La arquitectura físicamente en el controlador se detalla ahora en la siguiente figura.

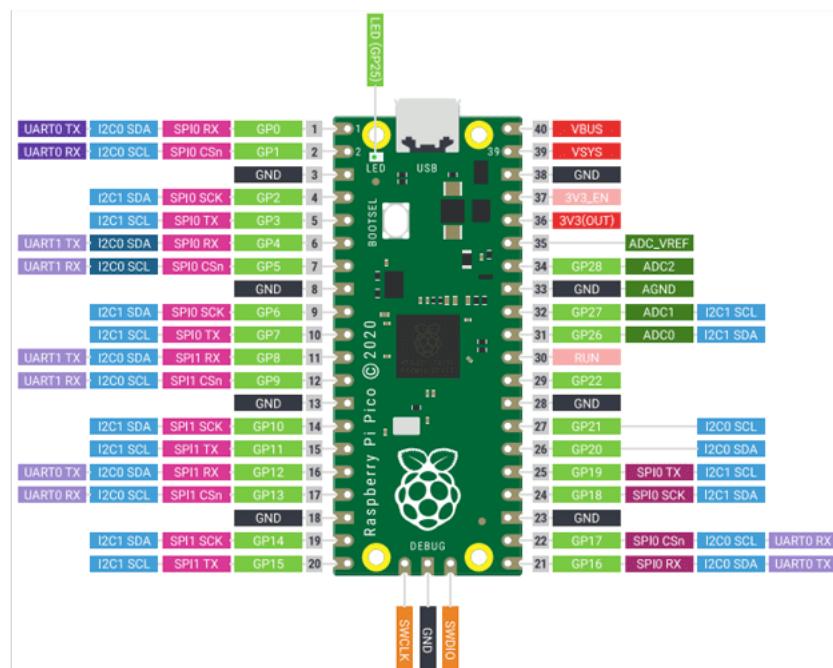


Figura 4.8: Driver controlador de motores Talon SR.

Sección 4.3.

Raspberry PI 4 MODEL B

La raspberry PI 4 MODEL B, es básicamente un dispositivo electrónico que tiene la función de ser una microcomputadora que puede trabajar en distribuciones de sistema operativo basado en Linux con distintas distribuciones disponibles, como pueden ser ubuntu, debian o como servidor de terminal.

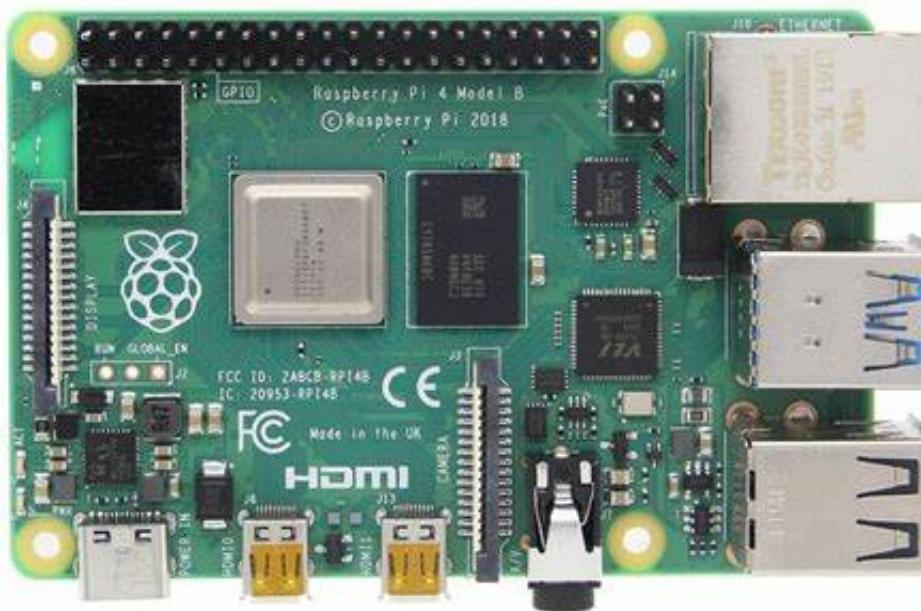


Figura 4.9: Sensor ultrasonico modelo HC-SR04

En las características técnicas tenemos:

CPU: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memoria RAM: 2GB, 4GB, or 8GB LPDDR4-3200 SDRAM (depende del modelo)
Storage: microSD card slot para cargar un SO y archivos
Conectividad: Gigabit Ethernet, dual-band 802.11ac wireless, Bluetooth 5.0, Alimentación por USB-C , dos puertos USB 3.0, dos puertos USB 2.0, dos puertos micro-HDMI, 3.5mm

4.3 Raspberry PI 4 MODEL B

audio jack

Dimensiones: 88 x 58 x 19.5 mm, 46 g.

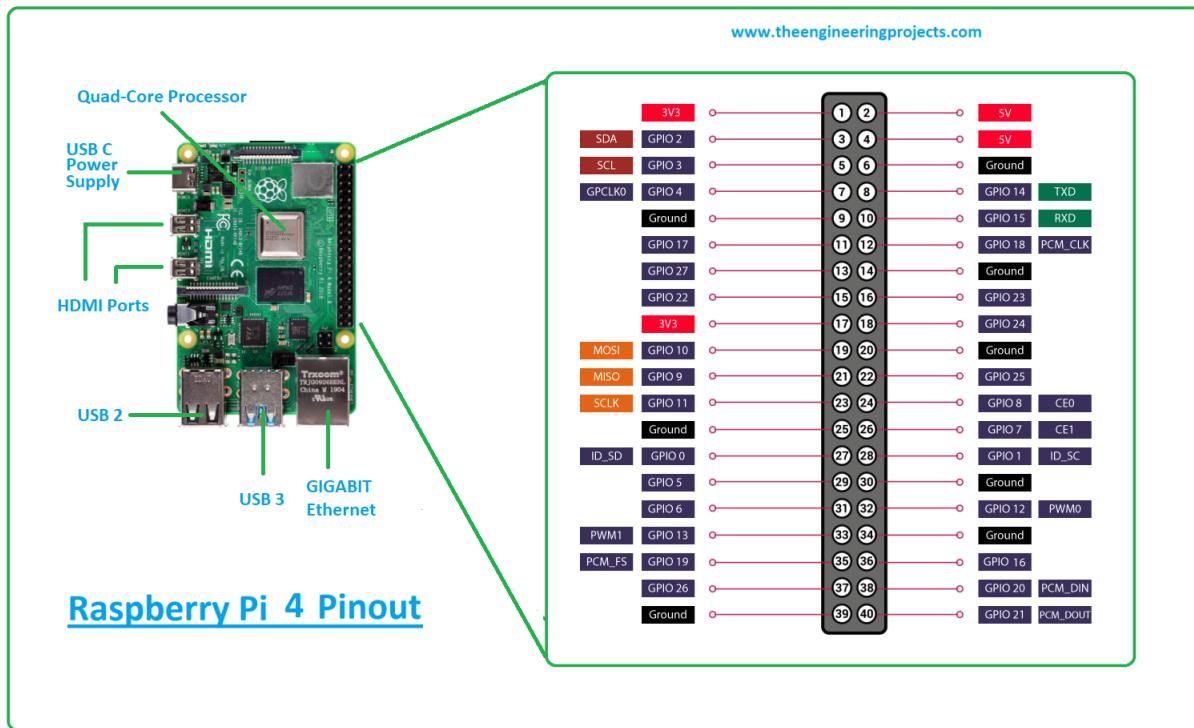


Figura 4.10: Características del dispositivo Raspberry PI 4 Model B. Imagen disponible en www.raspberrypi.com

Este dispositivo cuenta también con 40 GPIOs, que pueden utilizarse de manera conveniente para diferentes propósitos. Por otra parte, raspberry UK, proporciona un software llamado Raspberry PI Imager, con el cual se pueden cargar diferentes sistemas operativos compatibles en diferentes versiones, cada una recomendada para las capacidades de memoria RAM correspondientes a cada modelo.



Figura 4.11: Software Imager disponible para Windows y Linux

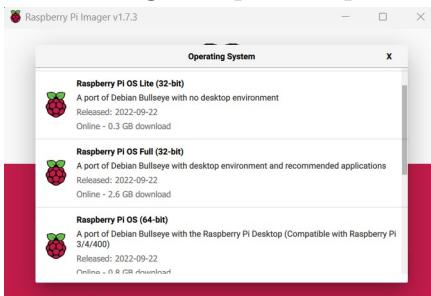


Figura 4.12: Sistemas operativos disponibles Raspbian.

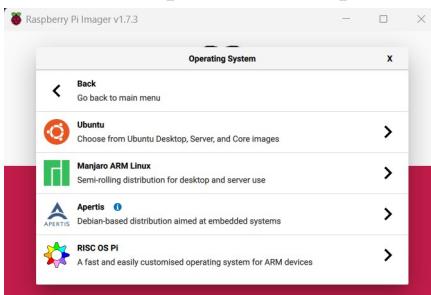


Figura 4.13: Sistemas operativos disponibles Linux.

Sección 4.4.

Sensores de Distancia

Un sensor de distancia es un dispositivo electrónico que permiten obtener el valor de la distancia mediante el uso de ondas ultrasonicas, es decir, hay una onda emitida desde un cabezal, la cual sera reflejada al entrar en contacto con alguna superficie u objeto, tomando lectura de esta misma onda el otro cabezal. Para obtener la distancia se toma como referencia el tiempo de entre la emisión y la recepción de la onda haciendo una conversión de distancia en base en la unidad de segundos que se especifique para cada modelo de sensor.

La distancia esta determinada comunmente por:

$$L = \frac{1}{2} * T * C \quad (4.1)$$

Donde T es el periodo del inverso de los milisegundos entre la emisión y la recepción del sensor y C es una constante correspondiente a la velocidad del sonido que es asociada a:

$$C = 343[m/s]$$

4.4.1 Sensor Ultrasonico HC-SRF05

El sensor ultrasonico HC-SFR05 es un sensor ultrasonico diferente a los modelos HC-SR04, cuya característica principal es que ofrece mayor rango para lecturas y cuenta fisicamente con más pines que son utilizados como pruebas de programación, por lo que generalmente solo se ocupan 4 Pines correspondientes Ground, Vcc, Trigger Pin y Echo Pin.



Figura 4.14: Sensor ultrasonico modelo HC-SRF05

Sus características técnicas son:

- Lectura de 1.7 [cm] a 4 [m]
- Alimentación de 5 [V] a 4 [mA]
- Frecuencia de 40 [KHz]

Sección 4.5.

Sistemas de control de robots móviles

Sección 4.6.

ROS

Por sus siglas (Robot Operating System), ROS es un middleware que provee librerías como controladores de dispositivos además de herramientas de visualización y comunicación para facilitar a los desarrolladores el diseño de software y creación de aplicaciones de robots. ROS tiene compatibilidad con distintos sistemas operativos como Mac OS X, Windows 10, pero principalmente con plataformas basadas en Linux.

ROS cuenta con distintas distribuciones de diferentes paquetes de ROS, tienen el objetivo de que los programadores trabajen con el código base estable para facilitar la operación de software y hardware. También existen nuevas distribuciones ROS2.

Las principales bibliotecas disponibles para cada distribución son:

roscpp: Es una biblioteca de cliente C++, a su vez una de las más utilizadas para bibliotecas de alto rendimiento en ROS

rospy: Es la biblioteca diseñada para Python del cliente ROS
roslib: Es una biblioteca de cliente LISP y generalmente se utiliza para bibliotecas de planificación.

CAPÍTULO 5

METODOLOGÍA

Sección 5.1.

Tren motriz del robot móvil

Sección 5.2.

Protección de la electrónica de potencia

Sección 5.3.

Sensores de distancia y retroalimentación

Sección 5.4.

Algoritmo de control móvil

Sección 5.5.

Configuración para el controlador Talón SR

Considerando la característica de que la salida de la raspberry tiene una salida de la alimentación de 3.3 V y el rango de la señal de pulso modulada del talon es de 1 a 2 milisegundos, se debe de ajustar el ciclo de trabajo para configurar y calibrar el voltaje de salida para caracterizar los movimientos. De manera similar a la tabla 4.1, ahora tenemos:

Movimiento	Voltaje en Rueda 1	Voltaje en Rueda 2	Voltaje en Rueda 3	Voltaje en Rueda 4
ALTO	1.6	1.6	1.6	1.6
ADELANTE	3.3	3.3	3.3	3.3
ATRAS	0.1	0.1	0.1	0.1
DERECHA	3.3	0.1	0.1	3.3
IZQUIERDA	0.1	3.3	3.3	0.1
DIAGONAL D1	3.3	1.6	1.6	3.3
DIAGONAL D2	1.6	0.1	0.1	1.6
DIAGONAL I1	1.6	3.3	3.3	1.6
DIAGONAL I2	1.6	0.1	0.1	1.6
Giro en sentido anti horario	3.3	3.3	0.1	0.1
Giro en sentido horario	0.1	0.1	3.3	3.3

Tabla 5.1: Caracterización de movimientos.

Sección 5.6.

Diagrama de Flujo para el algoritmo de navegación

Sección 5.7.

Diagramas de conexión

5.7.1 Propuesta de Diseño del circuito.

En la siguiente figura se muestra la propuesta de los diagramas de conexión del circuito.

5.7 Diagramas de conexión

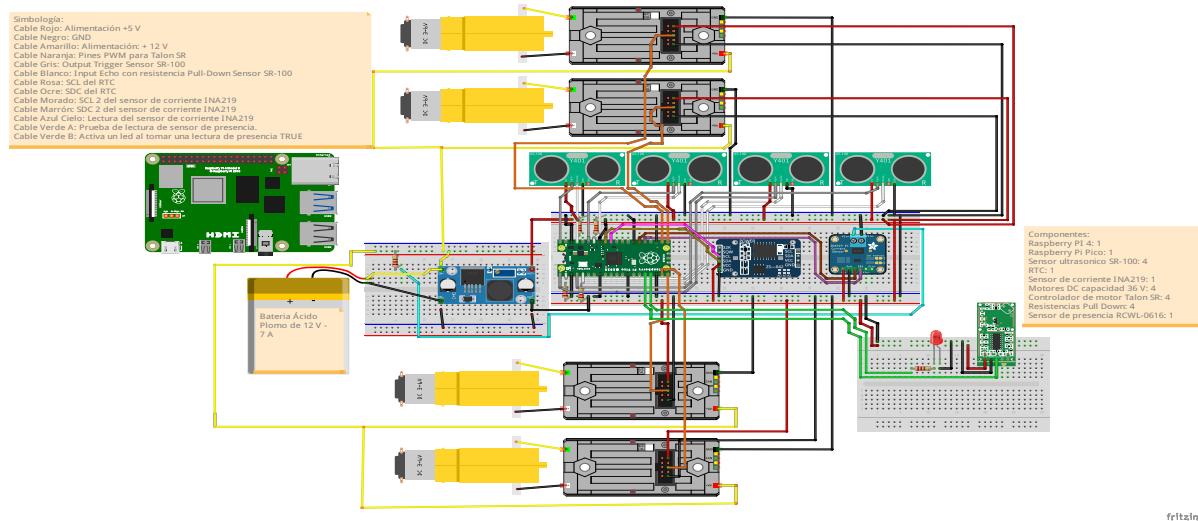


Figura 5.1: Propuesta de los diagramas de conexión.

—CAPÍTULO 6—

Resultados y discusión

CAPÍTULO 7

Conclusiones

CAPÍTULO 8

Códigos realizados

Código de Prueba para un push button.

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"

#define BUTTON_PIN 15

int main() {
    stdio_init_all();
    gpio_init(BUTTON_PIN);
    gpio_set_dir(BUTTON_PIN, GPIO_IN);
    gpio_pull_up(BUTTON_PIN);

    while(true) {
        if(!gpio_get(BUTTON_PIN)){
            printf("Button pressed\n");
        }
        sleep_ms(250);
    }
}
```

Código Utilizado para un Pin convertidor de lectura análogo Digital y control de encendido de un Led por señal PWM.

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"

#define POT_PIN 26
#define LED_PIN 14

long map(long x, long in_min, long in_max, long out_min, long out_max)
```

```

{
    return (x-in_min)*(out_max - out_min) / (in_max - in_min) + out_min;
}

int main()
{
    stdio_init_all();
    adc_init();
    adc_gpio_init(POT_PIN);
    adc_select_input(0);
    gpio_set_function(LED_PIN, GPIO_FUNC_PWM);
    uint slice_num = pwm_gpio_to_slice_num(LED_PIN);
    pwm_set_wrap(slice_num, 255);
    pwm_set_chan_level(slice_num, PWMCHAN_A, 0);
    pwm_set_enabled(slice_num, true);
    while(1)
    {
        uint16_t result = adc_read();
        long pwm_value = map(result, 0, 4095, 0, 255);
        printf("Raw: %d \t PWM: %d \n", result, pwm_value);
        pwm_set_chan_level(slice_num, PWMCHAN_A, pwm_value);
        sleep_ms(50);
    }
}

```

Código Utilizado para el Display LCD16X02:

```

// Programa para Display LCD 16x02 con modulo I2C
// Incluimos las bibliotecas a utilizar
#include <stdio.h>
#include <string.h>
#include "pico/stlplib.h"
#include "hardware/i2c.h"
#include "pico/binary_info.h"

/* Example code to drive a 16x2 LCD panel via a I2C bridge chip (e.g. PCF8574)

GPIO 4 (pin 6) -> SDA on LCD bridge board
GPIO 5 (pin 7) -> SCL on LCD bridge board
5 V (pin 36) -> VCC on LCD bridge board
GND (pin 38) -> GND on LCD bridge board
*/
// Comandos
const int LCD_CLEARDISPLAY = 0x01;
const int LCD_RETURNHOME = 0x02;
const int LCD_ENTRYMODESET = 0x04;

```

```
const int LCD_DISPLAYCONTROL = 0x08;
const int LCD_CURSORSHIFT = 0x10;
const int LCD_FUNCTIONSET = 0x20;
const int LCD_SETCGRAMADDR = 0x40;
const int LCD_SETDDRAMADDR = 0x80;

// Banderas para entrada del Display
const int LCD_ENTRYSHIFTINCREMENT = 0x01;
const int LCD_ENTRYLEFT = 0x02;

// Banderas para el cursor del Display
const int LCD_BLINKON = 0x01;
const int LCD_CURSORON = 0x02;
const int LCD_DISPLAYON = 0x04;

// flags for display and cursor shift
const int LCD_MOVERIGHT = 0x04;
const int LCD_DISPLAYMOVE = 0x08;

// flags for function set
const int LCD_5x10DOTS = 0x04;
const int LCD_2LINE = 0x08;
const int LCD_8BITMODE = 0x10;

// flag for backlight control
const int LCD_BACKLIGHT = 0x08;
const int LCD_ENABLE_BIT = 0x04;

// By default these LCD display drivers are on bus address 0x27
static int addr = 0x27;

// Modes for lcd_send_byte
#define LCD_CHARACTER 1
#define LCD_COMMAND 0

#define MAX_LINES 2
#define MAX_CHARS 16

/* Quick helper function for single byte transfers */
void i2c_write_byte(uint8_t val) {
#endif
    i2c_write_blocking(i2c_default, addr, &val, 1, false);
#endif
}

void lcd_toggle_enable(uint8_t val) {
    // Toggle enable pin on LCD display
```

```
// We cannot do this too quickly or things don't work
#define DELAY_US 600
    sleep_us(DELAY_US);
    i2c_write_byte(val | LCD_ENABLE_BIT);
    sleep_us(DELAY_US);
    i2c_write_byte(val & ~LCD_ENABLE_BIT);
    sleep_us(DELAY_US);
}

// The display is sent a byte as two separate nibble transfers
void lcd_send_byte(uint8_t val, int mode) {
    uint8_t high = mode | (val & 0xF0) | LCD_BACKLIGHT;
    uint8_t low = mode | ((val << 4) & 0xF0) | LCD_BACKLIGHT;

    i2c_write_byte(high);
    lcd_toggle_enable(high);
    i2c_write_byte(low);
    lcd_toggle_enable(low);
}

void lcd_clear(void) {
    lcd_send_byte(LCD_CLEARDISPLAY, LCD_COMMAND);
}

// go to location on LCD
void lcd_set_cursor(int line, int position) {
    int val = (line == 0) ? 0x80 + position : 0xC0 + position;
    lcd_send_byte(val, LCD_COMMAND);
}

static void inline lcd_char(char val) {
    lcd_send_byte(val, LCD_CHARACTER);
}

void lcd_string(const char *s) {
    while (*s) {
        lcd_char(*s++);
    }
}

void lcd_init() {
    lcd_send_byte(0x03, LCD_COMMAND);
    lcd_send_byte(0x03, LCD_COMMAND);
    lcd_send_byte(0x03, LCD_COMMAND);
    lcd_send_byte(0x02, LCD_COMMAND);

    lcd_send_byte(LCD_ENTRYMODESET | LCD_ENTRYLEFT, LCD_COMMAND);
```

```

        lcd_send_byte(LCD_FUNCTIONSET | LCD_2LINE, LCD_COMMAND);
        lcd_send_byte(LCD_DISPLAYCONTROL | LCD_DISPLAYON, LCD_COMMAND);
        lcd_clear();
    }

int main() {
#ifndef i2c_default || !defined(PICO_DEFAULT_I2C_SDA_PIN) || !defined(PICO_DEFAULT_I2C_SCL_PIN)
    #warning i2c/lcd_1602_i2c example requires a board with I2C pins
#else
    // This example will use I2C0 on the default SDA and SCL pins (4, 5 on PICO)
    i2c_init(i2c_default, 100 * 1000);
    gpio_set_function(PICO_DEFAULT_I2C_SDA_PIN, GPIO_FUNC_I2C);
    gpio_set_function(PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C);
    gpio_pull_up(PICO_DEFAULT_I2C_SDA_PIN);
    gpio_pull_up(PICO_DEFAULT_I2C_SCL_PIN);
    // Make the I2C pins available to picotool
    bi_decl(bi_2pins_with_func(PICO_DEFAULT_I2C_SDA_PIN, PICO_DEFAULT_I2C_SCL_PIN,
                               I2C_SDA, I2C_SCL));
    bi_enddef();

    lcd_init();

    static char *message[] =
    {
        "Ejemplo LCD16X02", "Raspberry Pi Pico",
    };

    while (1) {
        for (int m = 0; m < sizeof(message) / sizeof(message[0]); m += MAX_LINES)
            for (int line = 0; line < MAX_LINES; line++)
                lcd_set_cursor(line, (MAX_CHARS / 2) - strlen(message[m + line]));
        lcd_string(message[m + line]);
    }
    sleep_ms(2000);
    lcd_clear();
}
}

return 0;
#endif
}

```

Código Utilizado para Calibrar el Talon SR:

```

//incluimos todas las bibliotecas que vamos a usar con las funciones para PICO
#include <stdio.h>
#include "pico/stlplib.h"
#include "hardware/gpio.h"

```

```

#include "hardware/adc.h"
#include "hardware/pwm.h"

//Definimos los nombres de los GPIO que vamos a usar
#define POT_PIN 26
#define LED_PIN 14
#define CAL_PIN 15 // Este pin se usara para calibrar el Talon

//EL PWM utiliza el reloj del sistema , cada pulso tiene una frecuencia de 125 M
//el TALON SR funcione debemos tener una frecuencia de 1-2 ms con este valor sal
//Si seleccionamos 2 ms como frecuencia el numero de ciclos es: 2ms/8ns = 250,0
//Por esta razon debemos usar el divider de PWM si queremos 633Hz que es un per
//Divider = (125000000/(4096*frecuenciaDeseada))/16

//existen 8 slices para el PWM cada slice tiene dos salidas PWM,
//en total hay 16 salidas PWM, y todos los GPIO pueden controlarlos
//Para el GPIO 14 es el PWM_A7, Para el GPIO 15 ES EL PWM_AB

// creamos una funcion para que funcione el pwm desde 0 a 180

//El ADC es de 12 bits por lo que el maximo valor es de 4095
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
    return (x - in_min)*(out_max-out_min) / (in_max-in_min) + out_min;
}

int main(){

    int frecuenciaDeseada = 300;
    float Divider = (125000000/(4096*frecuenciaDeseada))/16; //= 3.81

    long ciclos = 65535;//numero de ciclos del wrap y numero de ciclos maximo

    //iniciamos la biblioteca stdio para que funcione todo el programa
    stdio_init_all();

    //Le damos reloj al ADCA
    adc_init();


---


    INICIALIZACION ADC

```

```

adc_gpio_init(POT_PIN); // iniciamos el PIN 26 correspondiente al ADC que
adc_select_input(0); // Activamos el canal 0 del ADC para el potenciómetro
//canal 0 corresponde a pin 26
//canal 1 corresponde a pin 27
//canal 2 corresponde a pin 28
//canal 3 corresponde a pin 29, aqui se encuentra el sensor de temperat

//----- INICIALIZACION PWM LED-----
gpio_set_function(LED_PIN, GPIO_FUNC_PWM); // seleccionamos la funcion
//numero de pin correspondiente a LED_PIN, la funcion sera PWM

//----- INICIALIZACION PWM TALON-----
gpio_set_function(CAL_PIN, GPIO_FUNC_PWM); // seleccionamos la funcion
//numero de pin correspondiente a CAL_PIN, la funcion sera PWM

//LO de abajo significa que seleccionamos el PWM 7, ambos canales A y B
uint slice_num = pwm_gpio_to_slice_num(LED_PIN);

//Utilizamos la funcion que dividira el reloj del sistema
pwm_set_clkdiv (slice_num , Divider);

//”ciclos” es el valor maximo antes de reiniciar el contador a 0
pwm_set_wrap(slice_num , ciclos);

pwm_set_chan_level(slice_num , PWM_CHAN_A,0); //Empezamos en 0 el nivel
//RECORDEMOS QUE ESTAMOS EN EL CANAL A
pwm_set_chan_level(slice_num , PWM_CHAN_B,0); //Empezamos en 0 el nivel
//RECORDEMOS QUE ESTAMOS EN EL CANAL B

pwm_set_enabled(slice_num ,true); //Habilitamos el PWM

//loop principal
while (1)
{
    uint16_t result = adc_read(); //Leemos el resultado del ADC del pot
    long pwm_value = map(result , 0, 4095, 0, ciclos); //convertimos el
    float voltaje = pwm_value*3.3/65635;

    printf("Raw: %d \t PWM: %d \t Voltaje PWM: %.1f \n", result , pwm_
    //imprimimos el valor del resultado del ADC y el valor para el PWM
    //para visualizar valores en pantalla usamos sudo minicom -b 115200

    //SELECCIONAMOS CON PWMVALUE LA CANTIDAD DE CICLOS QUE QUEREMOS
    //EN ALTO DENTRO DEL PERIODO DE 180, EJ, SI pwm_value ES 10 PUES 10
}

```

```

//180 SERAN 1 , LOS DEMAS SERAN 0

//INDICAMOS QUE USAREMOS EL NUMERO DE SLICE 7 PORQUE LO DECLARAMOS ARRIBA

//INDICAMOS QUE USAREMOS EL CANAL A
//INDICAMOS QUE USAMOS EL VALOR DEL PWM DEL POTENCIOMETRO
pwm_set_chan_level(slice_num , PWMCHAN_A, pwm_value);

//INDICAMOS QUE USAREMOS EL CANAL B
//INDICAMOS QUE USAMOS EL VALOR DEL PWM DEL POTENCIOMETRO
pwm_set_chan_level(slice_num , PWMCHAN_B, pwm_value);

//RETARDO DE 50ms para visualizar los datos en la pantalla
sleep_ms(100);

}

}

```

Código Utilizado para los sensores Ultrasónicos:

```

// Programa de sensor ultrasonico
// INcluimos las librerias a utilizar

#include <stdio.h>
#include "pico/stlplib.h"
#include "hardware/gpio.h"
// Definios los GPIOs
uint trigPin=2;
uint echoPin=3;

// definimos un tiempo de salida
int timeout = 26100;
// Definimos una funcion para inicializar los Gpios
void setupUltrasonicPins(uint trigPin , uint echoPin)
{
    gpio_init(trigPin );
    gpio_init(echoPin );
    gpio_set_dir(trigPin , GPIO_OUT);
    gpio_set_dir(echoPin , GPIO_IN);
}
// uint64
uint64_t getPulse(uint trigPin , uint echoPin )
{
    gpio_put(trigPin , 1);
    sleep_us(10);

```

```

        gpio_put(trigPin , 0);
        uint width = 0;
        while(gpio_get(echoPin) == 0) tight_loop_contents();
        while(gpio_get(echoPin) == 1)
        {
            width++;
            sleep_us(1);
            if(width > timeout) return 0;
        }
        return width;
    }
    int getCm(uint trigPin , uint echoPin)
    {
        uint64_t pulseLength = getPulse(trigPin , echoPin);
        return pulseLength / 29 / 2;
    }
    int getInch (uint trigPin , uint echoPin)
    {
        uint64_t pulseLength = getPulse(trigPin , echoPin);
        return (long)pulseLength /74.f /2.f;
    }
    // Definimos la funcion principal
    int main()
    {
        stdio_init_all();
        setupUltrasonicPins(trigPin , echoPin);

        while(true)
        {

            if (getCm(trigPin , echoPin) <= 6){
                printf("\n Distancia minima: \t %d \t cm \t Alto",
                sleep_ms(500);
            }
            else if (getCm(trigPin , echoPin) >= 7 && getCm(trigPin , echoPin) <= 16){
                printf("\n Distancia cercana: \t %d \t cm \t Girar",
                getCm(trigPin , echoPin));
                sleep_ms(500);
            }
            else if (16 <= getCm(trigPin , echoPin)){
                printf("\n Distancia Segura: \t %d \t cm \t Avanzar",
                sleep_ms(500);
            }
        }
    }
}

```

Bibliografía

- [1] WHO, “WHO Coronavirus Disease (COVID-19) Dashboard”. [En línea]. Disponible en: <https://covid19.who.int/>.
- [2] “Covid-19 México, Datos Abiertos Dirección General de Epidemiología”. [En línea]. Disponible en: <https://datos.covid-19.conacyt.mx/>
- [3] M. Igoe y V. Chadwick, “After the pandemic : How will COVID-19 transform global health and development ?”, Devex, núm. April, pp. 1–9, 2020.
- [4] V. M. Lomas-Barrié, M. Peña-Cabrera, y T. Alcantara-Concepcion, “AYUDAME 1.0”, 2020. [En línea]. Disponible en: http://www.leai4.iimas.unam.mx/?page_id=101.[Consultado : 07 – jul – 2020].
- [5] H. Durrant-Whyte y T. Bailey, “Simultaneous localization and mapping: Part I”, IEEE Robot. Autom. Mag., vol. 13, núm. 2, pp. 99–108, 2006.
- [6] Bailey y H. Durrant-Whyte, “Simultaneous localization and mapping (SLAM): Part II”, IEEE Robot. Autom. Mag., vol. 13, núm. 3, pp. 108–117, 2006.
- [7] S. Gatesichapakorn, J. Takamatsu, y M. Ruchanurucks, “ROS based Autonomous Mobile Robot Navigation using 2D LiDAR and RGB-D Camera”, 2019 1st Int. Symp. Instrumentation, Control, Artif. Intell. Robot. ICA-SYMP 2019, pp. 151–154, 2019.
- [8] R. Liu, J. Shen, C. Chen, y J. Yang, “SLAM for Robotic Navigation by Fusing RGB-D and Inertial Data in Recurrent and Convolutional Neural Networks”, 2019 IEEE 5th Int. Conf. Mechatronics Syst. Robot. ICMSR 2019, pp. 1–6, 2019.
- [9] P. Henry, M. Krainin, E. Herbst, X. Ren, y D. Fox, “RGB-D mapping: Using depth cameras forense 3D modeling of indoor environments”, en Springer Tracts in Advanced Robotics, 2014, vol. 79, pp. 477–491.
- [10] F. Endres, J. Hess, J. Sturm, D. Cremers, y W. Burgard, “3-D Mapping with an RGB-D camera”, en IEEE Transactions on Robotics, 2014, vol. 30, núm. 1, pp. 177–187.
- [11] M. Labbé y F. Michaud, “Online global loop closure detection for large-scale multi-session graph-based SLAM”, en IEEE International Conference on Intelligent Robots and Systems, 2014, pp. 2661–2666.

- [12] X. Liu, B. Guo, y C. Meng, “A method of simultaneous location and mapping based on RGB-D cameras”, en 2016 14th International Conference on Control, Automation, Robotics and Vision, ICARCV 2016, 2017.
- [13] V. Lomas-Barrie, M. Peña-Cabrera, y J. Durán-Ortega, “Determining humanoid soccer player position based on Goal detection”, en Proceedings of the 2015 International Conference on Artificial Intelligence, ICAI 2015 - WORLDCOMP 2015, 2015, pp. 61–65.
- [14] P.-C. Mario, L.-J. Ismael, R.-C. Reyes, y C.-C. Jorge, “Machine vision approach for robotic assembly”, Assem. Autom., vol. 25, núm. 3, pp. 204–216, 2005.
- [15] IFR Press Release. Industrial Robots: Robot Investment Reaches Record 16.5 billion USD. Shanghai, Frankfurt, Sep 18, 2019 [consultado 9 Mar 2019] Disponible en: <https://ifr.org/ifr-press-releases/news/robot-investment-reaches-record-16.5-billion-usd>
- [16] Posibilidades de la robótica y perspectivas de la robótica en la medicina <https://www.revistadyna.com/busqueda/posibilidades-y-perspectivas-de-robotica-en-medicina>
- [17] Ramos, A. C. (2001). Estado del arte en cirugía robótica. Revista Mexicana de Cirugía Endoscópica, 2(2), 109 112.
- [18] Cornejo Aguilar, J. A., Cornejo, J., Vargas, M., Sebastian , R. (2019). La revolución de la cirugía robótica en latino américa y la futura implementación en el sistema de salud del Perú. Revista de la Facultad de Medicina Humana, 19(1), 16. 16.[10] R.H. Taylor A Perspective on Medical Robotics Proceedings of the IEEE.,