Desarrollo de una interfaz gráfica para el control remoto de robot explorador

Proyecto de Investigación

Por

Kevin Miramontes Escobedo

En cumplimiento a los requerimientos para la obtención de

la Especialidad de Tecnólogo en Mecatrónica

Revisor académico: Dr. Julio Cesar Solano Vargas.

Santiago de Querétaro, Qro., México, Agosto 2019.

Desarrollo de una interfaz gráfica para el control remoto de robot explorador

Proyecto de Investigación

Por

Kevin Miramontes Escobedo

En cumplimiento a los requerimientos para la obtención de

la Especialidad de Tecnólogo en Mecatrónica

Revisor académico: Dr. Julio Cesar Solano Vargas.

Santiago de Querétaro, Qro., México, Agosto 2019.

Índice general

Índice de figuras

Agradecimientos

A mis padres, **María Angelica Escobedo Castañeda** y **José Noe Miramontes Partida** por estar siempre que los necesite y darme la motivacion para salir adelante.

A mi novia, **Karen Yuliana Rojas Gonzalez** Por su apoyo incondicional, estar siempre ahí conmigo a pesar de los malos ratos y siempre apoyarme en las decisiones que he tomado hasta ahora.

A mi asesor, **Julio César Solano Vargas** por sus valiosos consejos y su apoyo en el desarrollo de este proyecto, ademas de ser un buen amigo.

A mis compañeros de área y laboratorio, **Juan Manuel Barrera Fernández**, **Francisco Jimenez Oronia**, **Daniel Moreno Orduña**, **Daniel Murrienta Alvarez**, **Omero Nicolas Olalde Mendoza**, **José María Guillen Soto** y **Edgar Saul Valencia Olvera** por su apoyo en todo este año de especialidad.

Por ultimo pero no menos importante, agradezco al **Consejo Nacional de Cienca y Tecnología (CONACYT)** y al **Centro de Ingeniería y Desarrollo Industrial (CIDESI)** por su apoyo y patrocinio para el desarrollo de este proyecto.

Resumen

En el siguiente documento se explica el desarrollo del proyecto para la creación de una interfaz gráfica para el control remoto de robot explorador, por medio de hardware de bajo costo y software libre (gratuito). Desde la creación de la interfaz gráfica esta la programación de la transmisión de datos por medio del protocolo RS-485 en el lenguaje de programación Python, también se muestra el diseño en SlodiWORKS de la carcasa para la pantalla. Además, se muestran los resultados obtenidos en base a experimentación a lo largo de los cuatro meses correspondientes para el desarrollo del proyecto, obteniendo así un prototipo que sirve como parte aguas para la actualización de una nueva generación de robots exploradores en el área de robótica de inspección industrial.

Capítulo 1

Generalidades del proyecto

1.1. Introducción

En las últimas décadas la robótica ha ido tomando un papel importante dentro de la industria, ya sea para optimizar procesos, elevar la calidad de los productos o sustituir a los humanos en tareas que requieran una gran precisión o que puedan presentar graves peligros. Por esto surge la necesidad de profundizar en dicha área, empleando recursos naturales, para impulsar el desarrollo de nuestro país [2].

A pesar de que ya se ha trabajado en algunos proyectos, la experiencia aún es poca en comparación con otros países más desarrollados, por lo cual también existe una falta de confianza por parte del cliente. Pero la necesidad de mejorar debido a la exigencia del mercado obliga a desarrollar nuevas tecnologías que nos ayuden a cumplir con nuestros objetivos [2].

Existen robots que son capaces de realizar trabajos de inspección en ambientes peligrosos, estos robots mejoran la calidad del trabajo reduciendo así los tiempos y lo mas importante es que gracias a la intervención de dichos robots el personal no corre ningún riesgo al realizar las tareas de inspección.
Los robots de inspección utilizan diversos sistemas con los que el operador actúa, uno de estos es el control remoto; el control remoto es aquel que le permite al operador manipular el movimiento del robot en las cuatro direcciones (arriba, abajo, adelante y atrás), también, algunos fabricantes permiten al operador modificar parámetros y rangos de operación tales como la velocidad, torque, etc.

El objetivo de los controles es poder hacer que el manejo del robot sea mas sencillo y efectivo, que el usuario pueda manipularlo y capturar todas las mediciones sin ningún problema, es por esto que cada vez se han ido implementando nuevas tecnologías que hacen que la comunicación humano-maquina sea mas eficiente.
Dichas tecnologías también brindan a los desarrolladores la oportunidad de hacer que la comunicación con el robot sea mas eficiente al momento de enviar/recibir y guardar los datos leídos por el robot, por lo que también permite implementar nuevas funciones que ayuden los usuarios a resolver problemas de manera mas rápida.

1.2. Planteamiento del problema

En la actualidad, en el área de energías renovables se encuentra un robot explorador para el análisis de tanques de almacenamiento. Dicho robot fue creado en el año de 2008 y cuenta con un control remoto que le permite al usuario mover al robot; el usuario es apoyado por una interfaz con un display gráfico de 128x64 el cual cuenta con funciones básicas de movimiento y programación del robot. La interfaz gráfica de la versión anterior permitía al usuario tener un control parcial del robot, por lo cual, se encuentra delimitado en muchas de sus funciones de movimiento.

En este proyecto se actualizó la interfaz gráfica con una pantalla touch de 7" para la mejora del manejo por parte del usuario, agilizar su aprendizaje de programación y aprovechar todas las características de movimiento y análisis del robot, esto para conocer de manera eficiente el estado físico del tanque de almacenamiento a inspeccionar; así como también el apoyo visual para conocer el comportamiento del robot.

1.3. Objetivos

1.3.1. Objetivos Generales

Diseñar una nueva interfaz gráfica a color táctil que sea capaz de controlar remotamente el robot explorador para medición de espesores de pared de tanques de almacenamiento y que brinde una mejor interacción con el operador.

1.3.2. Objetivos Específicos

1.- Diseñar una nueva carcasa para el soporte del control remoto.

2.- Desarrollar de una nueva interface gráfica para la programación y control de velocidad y dirección del robot explorador.

3.- Implementación de un protocolo de comunicación entre la interfaz gráfica y el robot explorador, que permita interactuar con el control de velocidad y dirección del robot.

4.- Experimentación de la interacción con el control de velocidad y dirección del robot explorador..

1.4. Justificación

Al diseñar un nuevo control remoto se obtiene los siguientes beneficios:

- Una forma más intuitiva y gráfica de manipular el robot.
- Rápido acceso a los modos de operación del robot.
- Rápido acceso a las configuraciones del robot.
- Un control más liviano con una pantalla que asegura visibilidad al usuario aun así este bajo luz que impida su visión.
- A futuro, se pretenden crear aplicaciones para graficar los datos obtenidos por el transductor y conectar una cámara al robot para realizar inspección visual.

1.5. Alcances y limitaciones

1.5.1. Alcances

Obtener una nueva interfaz gráfica capaz de interactuar con el usuario de una manera más rápida e intuitiva para la selección de los diferentes métodos de operación del “robot explorador”.

1.5.2. Limitaciones

Para poder obtener la carcasa del nuevo control se requiere de una impresora 3D, para el momento en el que este proyecto fue desarrollado se encontraba en mantenimiento, sin embargo se busco la manera de imprimirlo en otra impresora.

Capítulo 2

Fundamentos

2.1. Antecedentes

2.1.1. Tipos de robots para la inspección de tanques de almacenamiento

Dentro de las nuevas tecnologías para la inspección de tanques de almacenamiento se encuentran varios sistemas que ayudan al operador a hacer más fácil, rápida y segura la inspección. Entre los sistemas se encuentra el Spider UT (figura 2.1) [3], el cual es un vehículo de acceso remoto diseñado para realizar inspecciones ultrasónicas de medición de espesor de componentes de material ferromagnético como tanques, recipientes y tuberías de grandes diámetros, sin la necesidad de montar peligrosos y costosos andamios y sin arriesgar la integridad física de los técnicos durante la inspección.

El funcionamiento de este robot se da gracias a que se adhiere a la pared del tanque o recipiente utilizando ruedas magnéticas de alta intensidad. Las ruedas magnéticas del dispositivo son mucho más grandes en diámetro que otros diseños, lo cual es de gran utilidad para atravesar obstrucciones como las que se encuentren en los tanques de almacenamiento [1].

Figura 2.1: Robot explorador Spider [1]

Otro Sistema es el Scorpion2 (figura 2.2) [4]. Este sistema aporta importantes mejoras en la eficiencia de inspección a estructuras ferromagnético por encima del suelo, tales como tanques de almacenamiento, recipientes e instalaciones en alta mar. Las Ventajas de rendimiento de este sistema incluyen la entrega más rápida de datos, los patrones de escaneo eficientes, mayor probabilidad de detección y la capacidad de recopilar mediciones valiosas en lugares críticos reduciendo considerablemente los accidentes [4].

Figura 2.2: Robot explorador Scorpion2

También existe el RMS2 (figura 2.3), es de gran velocidad, el sistema de mapeo de la corrosión ultrasónica de alta precisión de acceso remoto diseñado para evaluar la condición de los activos como tanques de almacenamiento, tuberías, recipientes a presión y otros equipos críticos, apoyando programa de inspección eficiente que apoyan los procesos de gestión de integridad para

asegurar un funcionamiento eficaz y seguro [5].
El RMS2 es extremadamente flexible con una gama de cabezales de exploración para adaptarse a diferentes requisitos de inspección, es decir, existen varias versiones del sistema RMS2, la selección de este depende del área que se desea cubrir.

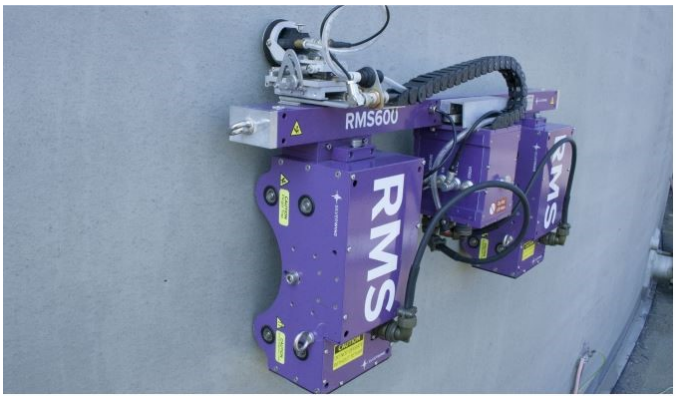

Figura 2.3: Robot explorador RMS2

Los equipos anteriormente mencionados son de empresas extranjeras (EUA y Gales) y el Scorpion2 (figura 2.2) como el RMS2 (figura 2.3) son desarrollados por la misma empresa, lo que quiere decir que sus principios de funcionamiento son los mismos.

Estos robots tienen diferentes modos de operación, por ejemplo, el robot Spider (figura 2.1) solo cuenta con modo manual mientras que el Scorpion2 y el RMS2 cuentan con manual y automático.

En México estos equipos son utilizados para la realización de tareas de insepccion en tanques de almacenamiento de hidrocarburos, por lo que el departamento de energías del Centro de Ingeniería y Desarrollo Industrial opto por desarrollar su propio robot explorador, el cual seria el primer robot medido de espesores hecho en México.
Actualmente se cuenta con dos versiones de dicho robot y en este trabajo se presenta una pequeña parte del desarrollo de la tercera version.

La primera version del robot creado en el Centro de Ingeniería y Desarrollo Industrial (figura 2.4) cuenta con un transductor para la medición de espesores de manera puntual, con un sistema para la detección de obstáculos, todo esto se controla mediante un DSP (por sus siglas en ingles

Digital Signal Processor).

Figura 2.4: Robot explorador Desarrollado en CIDESI primera generación

Por otro lado, la segunda generación de este robot desarrollado en México (figura 2.5), cuenta con un sistema de medición de manera puntual y continua, con ayuda de un acoplante. De igual manera cuenta con un sistema de detección de obstáculos controlado por el mismo procesador que la anterior generación (DSP).
Este robot cuenta con dos modos de operación, manual y automático. El modo manual da la libertad al usuario de mover el robot a cualquier dirección y variando su velocidad mientras que el modo automático es programado por el usuario para que el robot recorra cierta distancia de entre mediciones a una velocidad anteriormente establecida.

Figura 2.5: Robot explorador Desarrollado en CIDESI segunda generación

2.1.2. Control remoto de los robots desarrollados en CIDESI

Los robots creados en CIDESI cuentan con un control (figura 2.6) que permite al usuario manipular el robot gracias a que cuenta con dos joystics uno de ellos permite la movilidad hacia arriba y abajo y el otro hacia la izquierda y la derecha.
Para poder visualizar los parámetros del robot y las acciones que este hace, el control cuenta con una pantalla LCD controlada con un microcontrolador PIC 18F4550, dicho controlador contiene todos los modos de operación con los que el operador puede trabajar.

La carcasa del control esta hecha con acero inoxidable, lo cual asegura el control contra caídas, ademas cuenta con un botón de para de emergencia en caso de que el robot falle mientras esta escalando el contenedor.
Este control tiene la ventaja de ser robusto pero los usuario se han quejado de la poca visibilidad a la hora de configurarlo (dado que las pruebas se hacen bajo el sol) y lo tardado que es entrar a los diferentes menús ya que para navegar entre ellos se tienen que manejar los joystics.

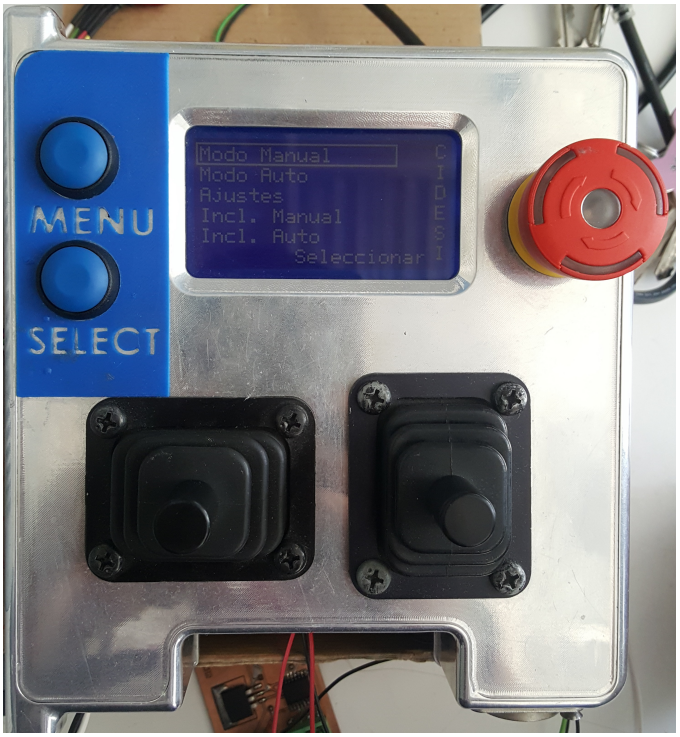

Figura 2.6: Control remoto utilizado por los robts exploradores creados en cidesi

2.2. Metodología

La metodología propuesta para la realización de este proyecto esta denotado en el diagrama de flujo que se muestra en la figura (2.7).
Se puede apreciar que durante la aplicación de la metodología existen 3 bucles en los que se realiza una experimentación, dicha experimentación puede no resultar como el proyecto lo requiere por lo que es necesario rehacer los pasos. Aunque parezca que esto puede atrasar el avance del proyecto esto esta contemplado dentro de los tiempos propuestos.

Las validaciones que se observan en la metodología, fueron realizadas tomando como ejemplo la version anterior del control y con ayuda de mi asesor, ya que no se contó con el tiempo suficiente para que el cliente fuera quien las validara.

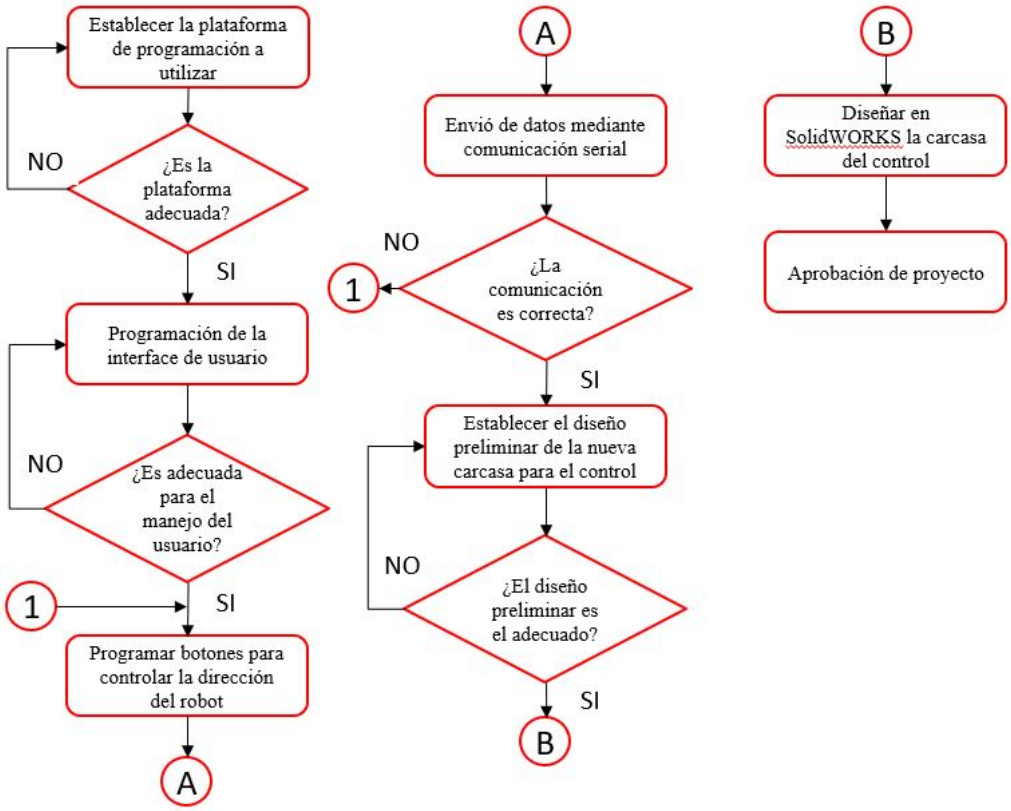

Figura 2.7: Metodología propuesta para el desarrollo del proyecto

Capítulo 3

Desarrollo

3.1. Programación de la interfaz gráfica en Windows

Los primeros pasos en el desarrollo de este proyecto fue la programación de la interfaz gráfica en Windows, simulando las dimensiones de la pantalla touch de 7 pulgadas.
Para poder hacer posible la programación en Windows se requirió la instalación de Python en su version 3.6 la cual se descargo directamente de la pagina oficial de Python [6].

3.1.1. Entorno TKinter

TKinter es una librería para la creación de interfaces gráficas en Python. TKinter resulta una herramienta bastante utilizada en la comunidad de Python gracias a su versatilidad y fácil manejo y programación de los widgets que TKinter brinda al desarrollador.
Dentro del entorno existen los widgets de botones, sliders, etiquetas entre muchas otras; para la elaboración de este proyecto solo se hizo uso de los widgets botones, sliders y etiquetas [7].

Como anteriormente se menciono, la comunidad ha utilizado TKinter para la creación de diversos proyectos generalmente relacionados con el control de entras y salidas de una Raspberry Pi. La programación de una interfaz gráfica en TKinter es relativamente sencilla e intuitiva dado que cada uno de los widgtes tiene parámetros de configuración, entre los mas comunes es tamaño, color de fondo, texto, tamaño del texto, fuente del texto, etc.

3.1.2. Programación de la Raíz de la interfaz

La raíz de la interfaz en TKinter es la base de toda la interfaz gráfica, podemos ver a la raíz como la hoja principal en la cual se va a dibujar el resto de la interfaz, es por esto que aquí se programan los parámetros importantes.

```
1  #Interface Grafica V0.2
2  from tkinter import *
3
4  #--------------------INICIO Raiz de la interface
5  Raiz=Tk()
6  Raiz.title ("Interface Grafica V0.2")
7  Raiz.resizable(False, False)
8  Raiz.config(bg="#D9D9D9")
9  Raiz.geometry("800x480")
10 cidesi = PhotoImage(file="cidesi2.png")
11 fondo = Label(Raiz, image=cidesi)
12 fondo.place(x=140, y=140)
13 #--------------------FIN Raiz de la interface
```

Figura 3.1: Programación de la raíz de la interfaz en Python con TKinter
Fuente: Elaboración propia

En la figura 3.1 se muestra la parte del código en la que se configura la raíz de la interfaz. En la linea 2 se importa la librería tkinter con todos sus elementos, en la linea 5 se le asigna la variable Raiz a la propiedad Tk() (que básicamente es el elemento principal para generar una raíz), en la linea 6 se le asigna un titulo a la ventana de la interfaz, en la linea 7 se configuro que a la interfaz no se le pudiera modificar el tamaño de la ventana, en la linea 8 se asigno el tamaño de la ventana y en las lineas 10, 11 y 12 se inserta una imagen y se posiciona en las coordenadas correspondientes de X y Y.

Una vez se programa la raíz de la interfaz se procedió a programar los botones principales para acceder a los diferentes menús dentro de la interfaz gráfica.
Dentro de la raíz anteriormente programada se tuvo que seccionar una parte para poder programar dentro de esa sección los botones principales, a esto se le conoce como frame, dicho de una manera distinta la raíz es la hoja principal y el frame es el trazo dimensionado en el cual serán dibujados ciertos elementos.

```
100  #--------------------INICIO Frame de los botones
101  FrameBoton=Frame()
102  FrameBoton.place(x=-1, y=0)
103  FrameBoton.config(bg="#0089CC")
104  FrameBoton.config(width="800", height="70")
105  FrameBoton.config(bd=2)
106  #--------------------FIN Frame de los botones
```

Figura 3.2: Programación del frame para los botones principales del los menús
Fuente: Elaboración propia

En la figura 3.2 se puede apreciar el código para la implementación del frame en donde posteriormente se programaron los botones principales.

3.1.3. Programación de menús principales

Como primer paso para la programación de los menús principales a los cuales se nombraron como modo manual, modo automático, calibrar transductor, ajustes, inclinometro manual e inclinometro automático. Estos menús son los elementales para el manejo del robot por parte del usuario.
Para poder navegar entre los menús se programaron botones, uno para cada menú al que el usuario pueda acceder. Para la programación de los botones se utilizó el widtget button de TKinter el cual nos permite modificar ciertos parámetros para darle un diseño personalizado y cual será la acción del botón una vez este sea presionado.

La configuración del widget button requiere de ciertos parámetros para su correcto funcionamiento, en la figura 3.3 se muestra el código para la programación del botón del menú modo manual. Se puede apreciar que en la línea 160 se nombro al botón como Boton Modo Manual, este botón se coloca en el frame anteriormente programado y designado a estos botones (FrameBoton) en las líneas 161 a la 165 se configuran las características de diseño como el color del botón, texto, tamaño del botón, fuente del texto y el color que toma una vez que sea presionado (esto para identificar que el botón ha sido presionado).

```
160  Boton_Modo_Manual = Button (FrameBoton,
161                              text="MODO \nMANUAL",
162                              width="10", height="3",
163                              font="Arial",
164                              bg="white",
165                              activebackground="#C0C0C0",
166                              relief=RAISED, command=Manual)
167  Boton_Modo_Manual.place(x=0, y=0)
```

Figura 3.3: Programación del botón para el menú modo manual
Fuente: Elaboración propia

En la línea 166 de la figura 3.3 se encuentra el parámetro command, este parámetro es el más importante del botón dado que dicho parámetro se iguala a una función, en este caso la función lleva por nombre Manual.
Por último en la línea 167 se coloca el botón en determinadas coordenadas en los ejes X y Y.

Para los demás menús principales faltantes (Modo automático, calibrar transductor, ajustes, inclinometro manual e inclinometro automático) el método es con exactitud el mismo la única variante para cada uno de ellos son las coordenadas en las que fueron colocados, lógicamente, dado que si las coordenadas fueran exactamente las mismas los botones estarían uno encima de otro.
En la figura 3.5 se muestra la pantalla principal de la interfaz con la raíz y el frame de los menús principales.

Figura 3.4: Ventana principal de la interfaz
Fuente: Elaboración propia

3.1.4. Programación del modo manual

En Python al programación de las funciones se denota por def():, todo el código que se requiera dentro de la función debe tener un tabulado, a diferencia del lenguaje C en el cual la funciones se denotan por void función y todo el código que se requiera va dentro de los corchetes. en el codigo 3.1 se muestra la función Manual la cual será ejecutada una vez el botón haya sido presionado.

Código 3.1: Función del modo manual

```
def Manual():
    ManualFrame=Frame()
    ManualFrame.place(x=0, y=70)
    ManualFrame.config(bg="white")
    ManualFrame.config(width="800", height="480")

    TextoManual=Label(ManualFrame, text="MODO MANUAL", font=("Arial
       ",20), fg="black", bg="white")
    TextoManual.place(x=320, y=0)

    Operacion_ManualFrame=Frame()
    Operacion_ManualFrame.place(x=0, y=100)
    Operacion_ManualFrame.config(bg="#0089CC")
    Operacion_ManualFrame.config(width="797", height="348")
    Operacion_ManualFrame.config(bd=2)
    Operacion_ManualFrame.config(relief="groove")

    Boton_Inicia_Medicion_Manual=Button(Raiz, text="MEDICION",
                                width="8", height="2",
                                font="Arial",
                                bg="#F2FA85")
    Boton_Inicia_Medicion_Manual.place(x=350, y=390)
    Cuadrante=Canvas(Operacion_ManualFrame, width="150", height="
       150", bg="#0089CC")
    Cuadrante.place(x=320, y=40)
    Cuadrante.create_line(0, 75, 150, 75)
    Cuadrante.create_line(75,0,75,150)
    Boton_Up = Button (Operacion_ManualFrame, text="", width="1",
       height="1",font=("Arial",25), bg="white")
    Boton_Up.place(x=90, y=190)
```

```
Boton_Down = Button (Operacion_ManualFrame, text="", width="1",
    height="1",font=("Arial",25), bg="white")
Boton_Down.place(x=90, y=270)

Boton_R = Button (Operacion_ManualFrame, text="", width="1",
   height="1",font=("Arial",25), bg="white")
Boton_R.place(x=690, y=225)

Boton_L = Button (Operacion_ManualFrame, text="", width="1",
   height="1",font=("Arial",25), bg="white")
Boton_L.place(x=610, y=225)

Tiempo_Bomba=Label(Operacion_ManualFrame, text="Tiempo de
   funcionamiento de la bomba: " +str(valor.get())+" Segundos")
Tiempo_Bomba.place(x=15, y=20)

Potencia_Bomba=Label(Operacion_ManualFrame, text="Potencia de
   la bomba: " +str(Valor_Potencia.get())+"%")
Potencia_Bomba.place(x=15, y=40)

Distancia_Robot=Label(Operacion_ManualFrame, text="Distancia en
    modo automatico: " +str(Valor_Distancia.get())+" CM.")
Distancia_Robot.place(x=15, y=60)
```

En general, lo que se programa dentro del la función Manual fue el entorno para el control del robot, hay cuatro botones con los que el usuario puede manipular el movimiento del robot, un botón adicional para inicializar la medición del transductor, y etiquetas que muestran el estado del robot (potencia de la bomba, tiempo de funcionamiento de la bomba y distancia automática del robot).

En retrospectiva, una vez que se presiona el botón Boton Modo Manual manda a llamar a la función Manual, la que ejecuta el código 3.1 y abre un sub-ventana nueva en donde se encuentran los botones para el control del robot.
En la figura se muestra el menú modo manual, el cuadro que se encuentra en el centro de la interfaz es un indicador gráfico que le indica al operador en qué dirección está moviendo al robot, esto gracias a que cuando se presiona alguno de los botones para darle dirección al robot una

flecha correspondiente a la dirección aparece dentro del cuadro central.

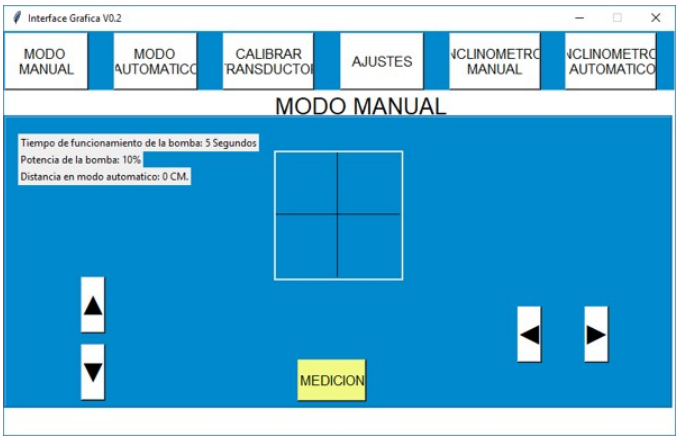

Figura 3.5: Ventana del menú modo manual
Fuente: Elaboración propia

3.1.5. Programación del menú ajustes

El menú ajustes es un menú en el que el usuario puede modificar tres parámetros de operación del robot, los cuales son:

1.- Tiempo Bomba: El robot cuenta con una bomba para la inyección del acomplante a la pared metálica y así el transductor pueda tomar las mediciones correctas y enviarlas al procesador. Dentro del menú ajustes el usuario puede configurar el tiempo en el que la bomba estará en funcionamiento dependiendo de las aplicaciones requeridas.

2.- Potencia Bomba: En este parámetro el usuario puede modificar la potencia de la bomba, tienes diversas ventajas poder manipular la potencia de la bomba una de las mas importantes es la reducción del consumo de energía por parte de la bomba cuando se requiere una mínima cantidad de acoplante.

3.- Distancia Automática: Actualmente el robot cuenta con una función en la cual se mueve cierta distancia entre medición, a esta función se le llamo como modo automático, en este parámetro el usuario puede ajustar la distancia que desee entre mediciones del robot. Esta función es muy útil dado que el usuario en lugar de estar midiendo en distancias aleatorias con esto sabe con certeza a que distancias están hechas las mediciones del robot.

Lo primero que se desarrollo fueron los tres botones, los cuales tienen la función de navegar entre los tres diferentes parámetros del robot. Cuando uno de los botones es presionado aparece

una ventana nueva, diferente para cada parámetro o submenú. En el código 3.2 se muestra la programación del menú Ajustes y los submenús de los tres parámetros de operación del robot.

Código 3.2: Función del menú ajustes y submenus de los tres distintos parámetros

```
def Ajustes():
    AjustesFrame=Frame()
    AjustesFrame.place(x=0, y=70)
    AjustesFrame.config(bg="white")
    AjustesFrame.config(width="800", height="480")

    TextoAjustes=Label(AjustesFrame, text="AJUSTES", font=("Arial"
       ,20), fg="black", bg="white")
    TextoAjustes.place(x=320, y=0)

    Menu_AjustesFrame=Frame()
    Menu_AjustesFrame.place(x=0, y=100)
    Menu_AjustesFrame.config(bg="white")
    Menu_AjustesFrame.config(width="797", height="39")
    Menu_AjustesFrame.config(bd=2)
    Menu_AjustesFrame.config(relief="groove")

    Tiempo()#Slider de tiempo de funcionamiento de la bomba

    Boton_TiempoBomba = Button(Menu_AjustesFrame,
                                       text="TIEMPO BOMBA",
                                       width="15", height="1",
                                       font="Arial",
                                       bg="white",
                                       activebackground="#C0C0C0",
                                       relief=RAISED, command=
                                          Tiempo)
    Boton_TiempoBomba.place(x=70, y=0)

    Boton_PotenciaBomba = Button(Menu_AjustesFrame,
                                         text="POTENCIA BOMBA",
                                         width="15", height="1",
                                         font="Arial",
                                         bg="white",
```

```
                                        activebackground="#C0C0C0"
                                            ,
                                        relief=RAISED, command=
                                            Potencia)
    Boton_PotenciaBomba.place(x=270, y=0)

    Boton_DistanciaAutomatica = Button(Menu_AjustesFrame,
                                            text="DISTANCIA
                                                AUTOMATICA",
                                            width="20", height="
                                                1",
                                            font="Arial",
                                            bg="white",
                                            activebackground="#
                                                C0C0C0",
                                            relief=RAISED,
                                                command=
                                                Distancia_Automatica
                                                )
    Boton_DistanciaAutomatica.place(x=470, y=0)

Boton_Modo_Ajustes = Button (FrameBoton,
                             text="AJUSTES",
                             width="10", height="3",
                             font="Arial",
                             bg="white",
                             activebackground="#C0C0C0",
                             relief=RAISED, command=Ajustes)
Boton_Modo_Ajustes.place(x=400, y=0)
```

Una vez programado el menú y los submenús se procedió a buscar un widget con el cual el usuario pudiera manipular los parámetros del robot de una manera rápida y sencilla. En la versión anterior los parámetros eran ajustados con ayuda del joystick. Tkinter contiene un widget llamado SLIDER, los sliders son muy comunes actualmente, podemos verlos en nuestros dispositivos a la hora de subir el volumen a la música, ajustar el brillo de la pantalla, etc. El widget SLIDER de TKinter tiene configuraciones muy similares a las del widget BUTTON incluso se le puede dar una orientación vertical u horizontal dependiendo del diseño que se requiera.

El SLIDER es una barra deslizante que parte desde un valor inicial A hasta un valor final B, estos valores puedes ser configurados para que inicien en cierto valor y terminen en otro. En la figura 3.6 se muestra el código para la configuración de un SLIDER en TKinter.

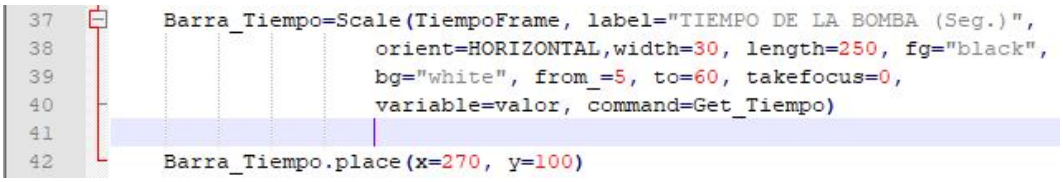

Figura 3.6: Programación del SLIDER
Fuente: Elaboración propia

En la figura 3.6 en la línea 38, el parámetro orient define cual será la orientación del slider, en este caso se igualo a una orientación horizontal, en la línea 39 los parámetros from y to indican desde que valor iniciara el slider (from) y hasta que valor será el limite (to), en la línea 40 el parámetro variable es aquel que guarda el valor en el que el slider esta colocado, esta variable valor es la que posteriormente se utiliza para enviar el dato por serial. En la misma línea 40 se aprecia el parámetro command, el mismo que es utilizados por los botones y por lo tanto se programa de igual manera en los sliders.

En la función Get Tiempo del parámetro command se crea una variable, dicha variable recibe el valor en tiempo real en el que el slider se encuentra, esto gracias al .get() que le asigna a la variable dicho valor.
Posteriormente se crea una nueva función que recibe el valor del slider igualando una variable a dicho valor recibito y convirtiendo el dato de entero a string, después para corroborar que el valor enviado por el slider corresponde al asignado la variable se imprime dicha variable en la consola del compilador.
En el código 3.3 se muestra la programación de las dos diferentes funciones, cabes señalar que en la última línea de código, valor.set(5) asigna a la variable valor el valor de 5, lo que significa que la variable siempre iniciara en 5.

Código 3.3: Funciones para la lectura del valor del slider

```
valor=IntVar()
var=IntVar()
Tiempo=IntVar()
def Get():
    Tiempo= str(valor.get())
    print(Tiempo)
```

```
def Get_Tiempo(valor):
    Tiempo= str(valor)
    print ("TIEMPO: "+(valor))
valor.set(5)
```

Por último, la programación para los otros dos submenús (Potencia bomba, Distancia Automática) es exactamente igual que la anterior mostrada. En la figura 3.7 se muestra la ventana de la interfaz en el menú ajustes y el submenú de configuración del tiempo de la bomba mientras que de el lado izquierdo se muestra el envió de los datos a la terminal del compilador comprobando que la posición del slider se encuentra en 23 mientras que los datos impresos en la terminal del compilador incrementa de uno en uno desde el 6 (dado que el slider empieza en 5) hasta el 23.

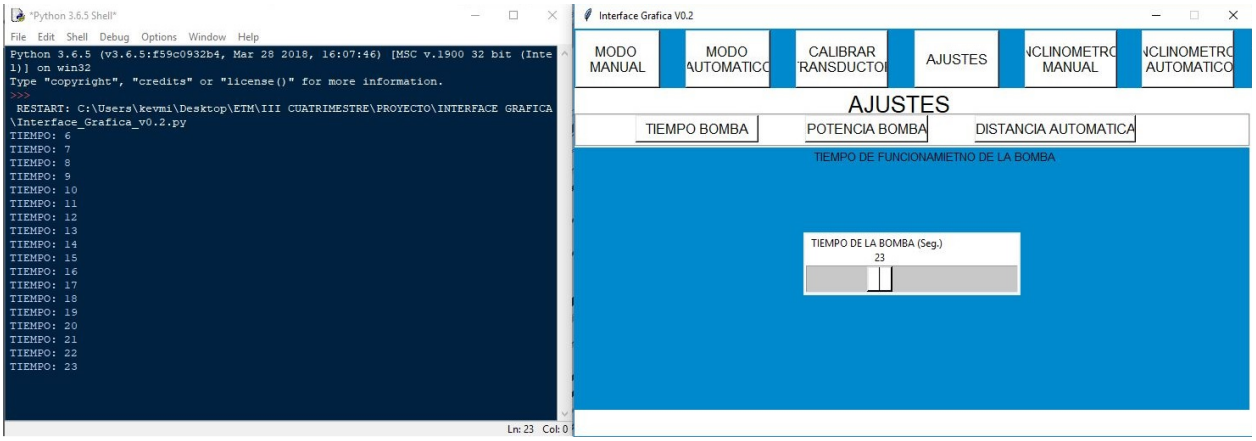

Figura 3.7: Ventana del menú ajustes en el submenú Tiempo de funcionamiento de la bomba
Fuente: Elaboración propia

3.2. implementación en Raspberry Pi 3 modelo B+

Una vez el código fue desarrollado en su totalidad en el sistema operativo Windows, se procedió a aplicar este código a la plataforma Raspberry pi, la razón por la cual se eligió una Raspberry sobre cualquier otra plataforma es por la gran capacidad que tiene y en un futuro se requieren crear nuevas funciones de almacenamiento de datos, lo cual es sencillo de hacer en dicha plataforma dado que todo el software corre desde una memoria SD la cual puede ser de diferentes capacidades, por lo tanto se tiene una amplio criterio de selección de memorias para futuras aplicaciones.

La Raspberry cuenta con gran variedad de periféricos, los cuales se pueden comprar directa-

mente en la página oficial. Para este proyecto se utilizó una pantalla de 7 pulgadas touch en la cual se visualizará la interfaz gráfica que se programó anteriormente en Windows [8].

Figura 3.8: Pantalla touch requerida para la elaboración del proyecto.
Fuente: Elaboración propia

3.2.1. Descarga del sistema operativo

Para la tarjeta Raspberry Pi existen varios Sistemas Operativos (SO) disponibles, pero es Raspbian el SO de diseño nativo para esta tarjeta. Es una distribución de Linux compilada especialmente para esta plataforma; para facilitar el proceso, existe un gestor de instalación llamado NOOBS. Puede descargarse de la página oficial de Raspberry.

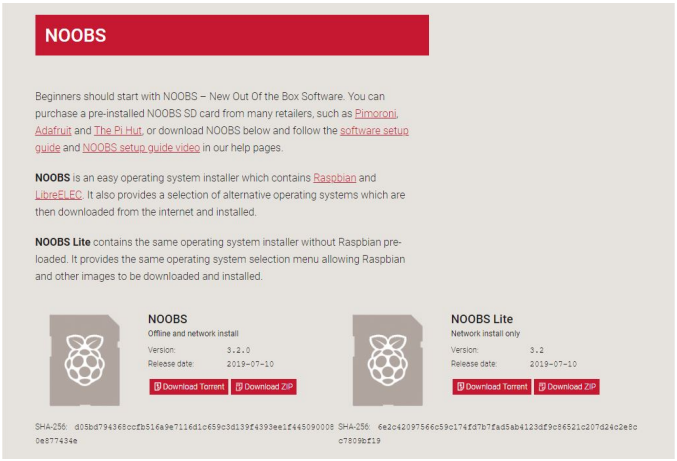

Figura 3.9: Descara del sistema operativo NOOBS para la raspberry.
Fuente: https://www.raspberrypi.org/downloads/noobs/

Noobs (New Out Of Box Software) es una aplicación que se instaló en la tarjeta de memoria, en lugar de instalar directamente un sistema operativo. Al arrancar la Raspberry Pi aparecerá

una ventana, la cual cuenta con varios sistemas operativos, e instalarlos es una tarea tan sencilla como solo hacer click sobre el sistema operativo desea y esperar a que este se descargue desde Internet y se instale en la Raspberry.

Existen dos versiones de Noobs (Figura 3.9). La normal viene precargada con el sistema operativo Raspbian por lo que si se elige instalar este sistema operativo no es necesario descargarlo dado que Raspbian mientras se esta instalando descarga Noobs de internet. En caso de elegir otro sistema, este se descargaría antes de instalarse.

La versión Lite contiene únicamente los ficheros necesarios para la ejecución de Noobs, sin Raspbian precargado, por lo cual la imagen tiene un tamaño menor. Por el contrario, cualquier sistema operativo que se elija al iniciar por primera vez la Raspberry Pi se deberá descargar al inicio.
En otras palabras, si se piensa instalar el sistema operativo Raspbian (que es el más común dentro de la comunidad de Raspberry) se debe elegir la versión completa, mientras que si se requiere otro sistema operativo es necesario descargar la versión Lite.

3.2.2. Instalación del sistema operativo

Para instalar Noobs en primer lugar, es necesario tener una computadora que tenga un lector de tarjetas SD, para poder preparar la tarjeta antes de utilizarla en la Raspberry Pi.
Posteriormente lo único que se debe de hacer es formatear la tarjeta SD y descomprimir el contenido del archivo ZIP que se ha descargado previamente en la tarjeta.

Una vez ya se haya instalado Noobs, el siguiente paso es instalar un sistema operativo lo cual es una tarea muy sencilla. Se introduce la memoria SD a la Raspberry Pi y energizamos la tarjeta.
Al iniciar aparece un menú (Figura 3.10) con los distintos sistemas operativos disponibles. Se elige el sistema operativo deseado y se pulsa la tecla I para instalarlo.

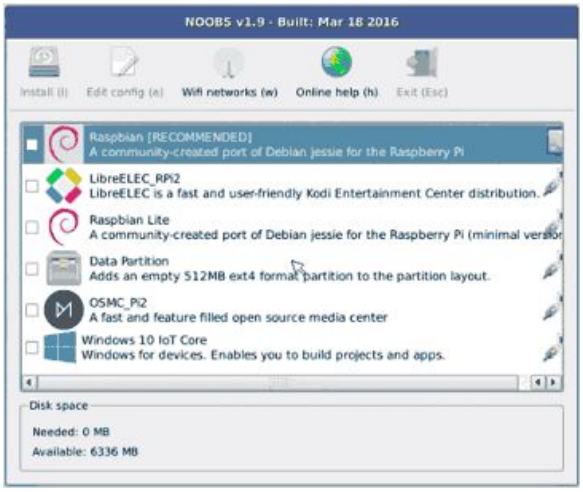

Figura 3.10: Ventana de instalación de los diferentes sistemas operativos disponibles para la Raspberry Pi.

Fuente: Elaboración propia

Una vez seleccionado el sistema operativo aparecerá una ventana (Figura 3.11) que nos indica que la instalación esta en proceso, por lo tanto, el sistema operativo se está instalando en la memoria SD. Este proceso dura alrededor de 5 minutos.

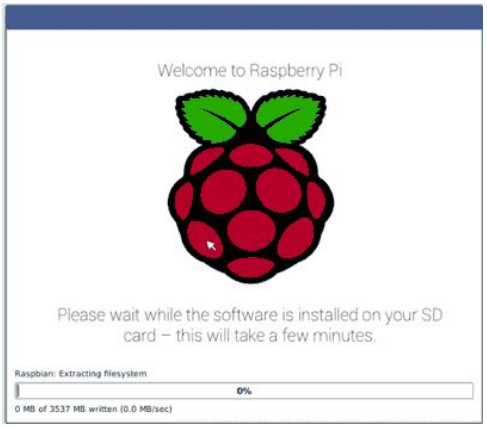

Figura 3.11: Ventana del proceso de instalación del sistema operativo.
Fuente: Elaboración propia.

Una vez finalizado el proceso de instalación la Raspberry Pi se reiniciará y al volver a iniciar se ejecutará el sistema operativo instalado.
Cabe señalar que es posible volver a acceder a Noobs en caso de que se deseé instalar un sistema operativo diferente, simplemente se pulsan la tecla SHIFT durante el arranque de la Raspberry Pi y repetir los pasos anteriormente mencionados.

3.2.3. implementación del código en Raspberry Pi 3

Dentro del sistema operativo Raspbian que se instalo en la Raspberry vienen herramientas para el desarrollo de proyectos, por ejemplo, en Raspbian ya viene instalado Python y un entrono de desarrollo para la programacion en Python el cual se llama Tommy (Figura 3.12) Tommy es un compilador muy sencillo dado que solo cuenta con la parte donde se escribe el código, una terminal y botones de correr y detener el programa, carece de herramientas para el debuggeo, es por esta razón que la interfaz gráfica primero fue programada de manera simulada en Windows, dado que fue ahí donde las pruebas resultaron mas sencillas. El proceso para la implementación

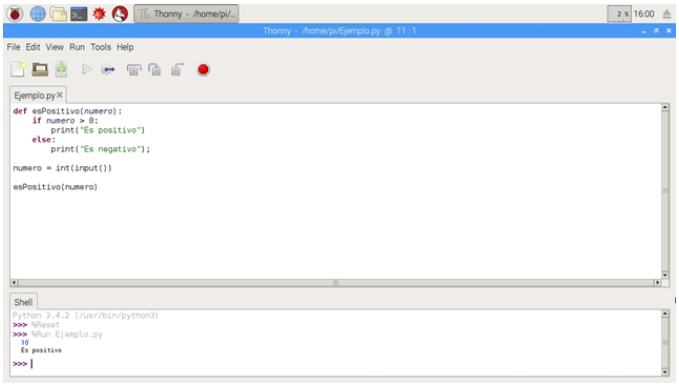

Figura 3.12: Entorno de programación Tommy.
Fuente: Elaboración propia.

del código anteriormente programado en Windows en la Raspberry es muy sencillo, como se mencionó el sistema operativo Raspbian ya contiene todas las librerías de Python, por lo tanto, lo único que se hizo fue pegar el código en Tommy y al ejecutar el código se aprecio que no se requería ninguna modificación dado que la interfaz calzaba perfecto en la pantalla tal como se muestra en la Figura 3.13.

Las pruebas que se realizaron sobre el funcionamiento del código de la interfaz sobre una pantalla táctil fueron sobre todo en los botones, corroborando que el accionamiento de estos fuera el adecuado para el manejo del control por parte del usuario y que no se presionaran varias botones al mismo tiempo evitando así accidentes.

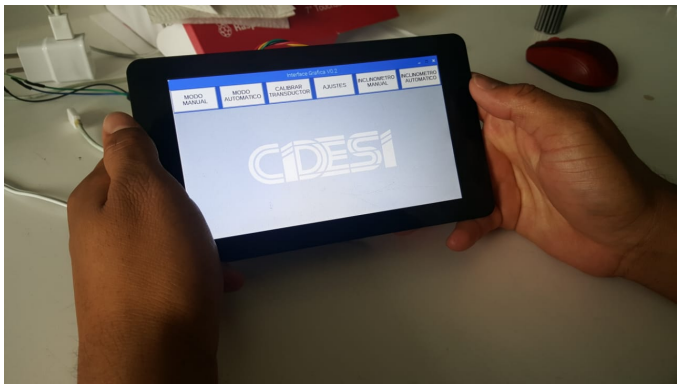

Figura 3.13: Interfaz gráfica en la pantalla touch de 7 pulgadas.
Fuente: Elaboración propia.

3.3. Comunicación Serial entre Raspberry Pi 3 y Hércules mediante protocolo RS-232

Hercules (Figura 3.14) es una útil terminal de puerto serie (RS-485 o RS-232), Terminal UDP/IP y terminal de servidor de cliente TCP/IP. Fue creado solo para el uso interno del grupo HW, pero hoy incluye muchas funciones en una sola utilidad y es de uso libre. Se puede descargar desde la pagina oficial del desarrollador: https://www.hw-group.com/software/hercules-setup-utility.

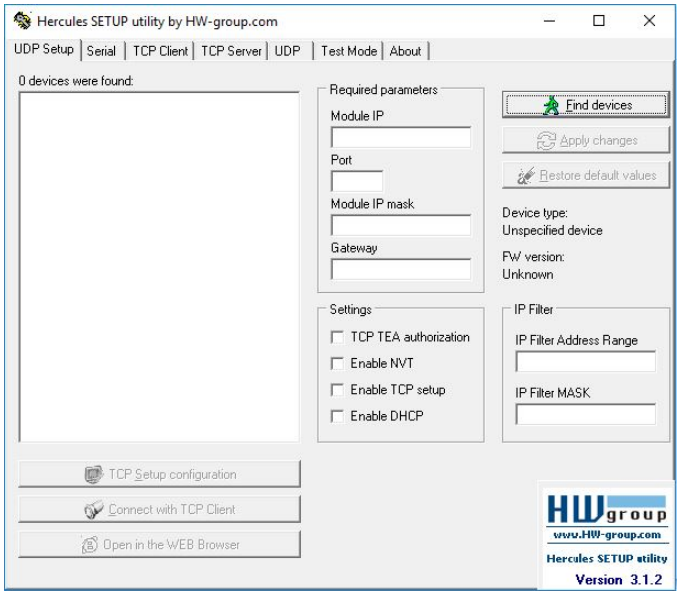

Figura 3.14: Ventana principal del software Hercules.
Fuente: Elaboración propia.

3.3.1. Instalación de librerías en Raspberry pi para la comunicación serial

Para que sea posible la comunicación serial desde la Raspberry, es muy común que la comunidad instale PYSERIAL desde Python, pero esta librería no funciona en las tarjetas Raspberry Pi 3, algunos de los síntomas que se presentan a la hora de intentar comunicarse es que la consola se queda congelada y no se logra ni enviar ni recibir ningún paquete.
Pero ¿Por qué este problema solo ocurre en la versión 3 de Raspberry?, la actualización de la tarjeta incluyo un módulo bluethoot dedicado, pasando de tener una sola UART a tener dos; una para el bluethoot y otra para los GPIO. Esta modificación implica que los pines GPIO ya no están direccionados en ttyAMA0 como ocurre en las versiones 1 y 2 de Raspbeerry, ahora pasa la interfaz UART a la dirección ttyS0, por lo tanto, se tiene que hacer modificaciones en el archivo de configuración del sistema operativo de la Raspberry.

Lo primero que se hizo fue encender la Raspberry y ejecutar en la consola de comandos para verificar los puertos disponibles, por lo que se escribe el código **dmseg/grep tty**, esta instrucción permite ver los puertos seriales conectados (véase la figura 3.15). Una vez que se verificaron los

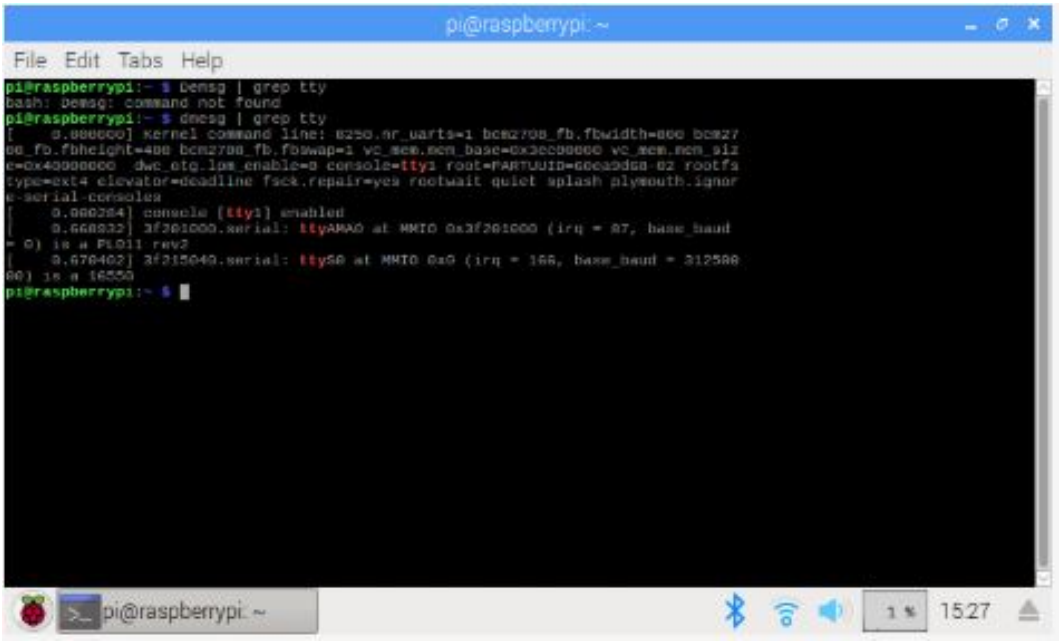

Figura 3.15: código en consola para ver los puertos serie conectados.
Fuente: Elaboración propia.

puertos, se configura la Raspberry para que Linux no use la UART para login Shell, por lo que en la consola de comandos se escribe el código **sudo raspi-config**.
Con esta instrucción se abre el software de configuración general de la tarjeta, se selecciona la

opción 5 de opciones de interfaz, posteriormente se inhabilita el login Shell, pero se mantiene habilitado el hardware de la UART (véanse figuras 3.16, 3.17, 3.18 y 3.19).

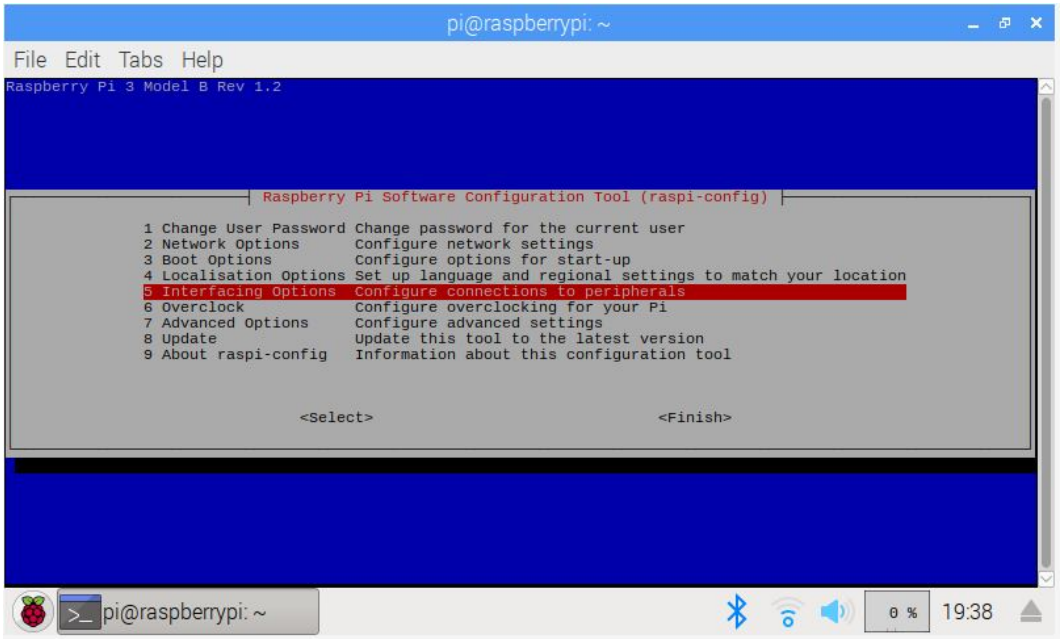

Figura 3.16: Herramienta de configuración de la Raspberry para configurar los periféricos.
Fuente: Elaboración propia.

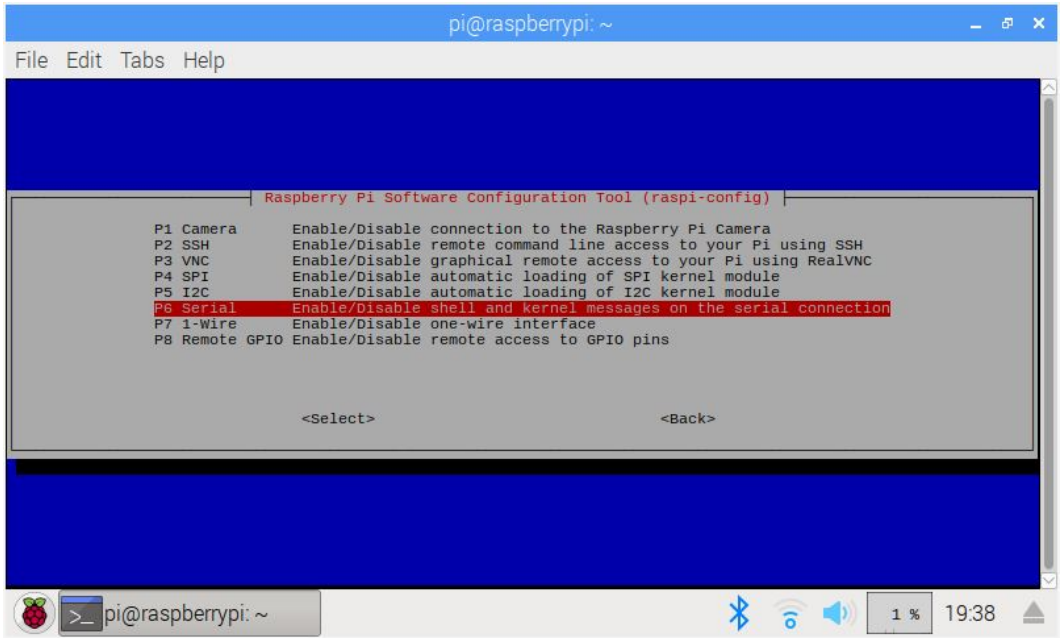

Figura 3.17: Activación/desactivación del shell y el kernel para la conexión serial.
Fuente: Elaboración propia.

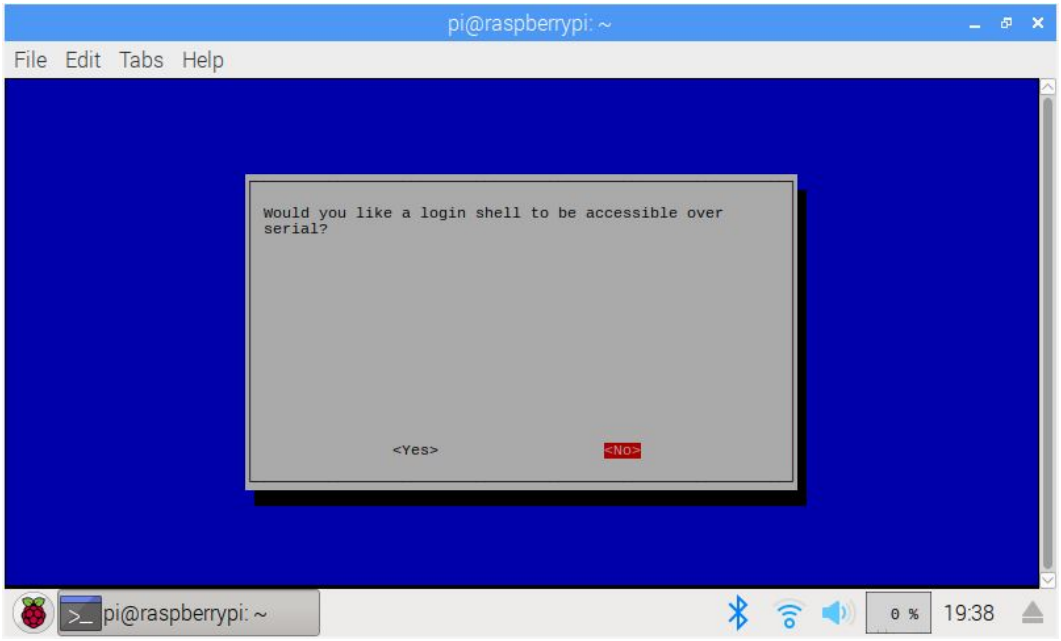

Figura 3.18: Desactivar el inicio de sesión al shell.
Fuente: Elaboración propia.

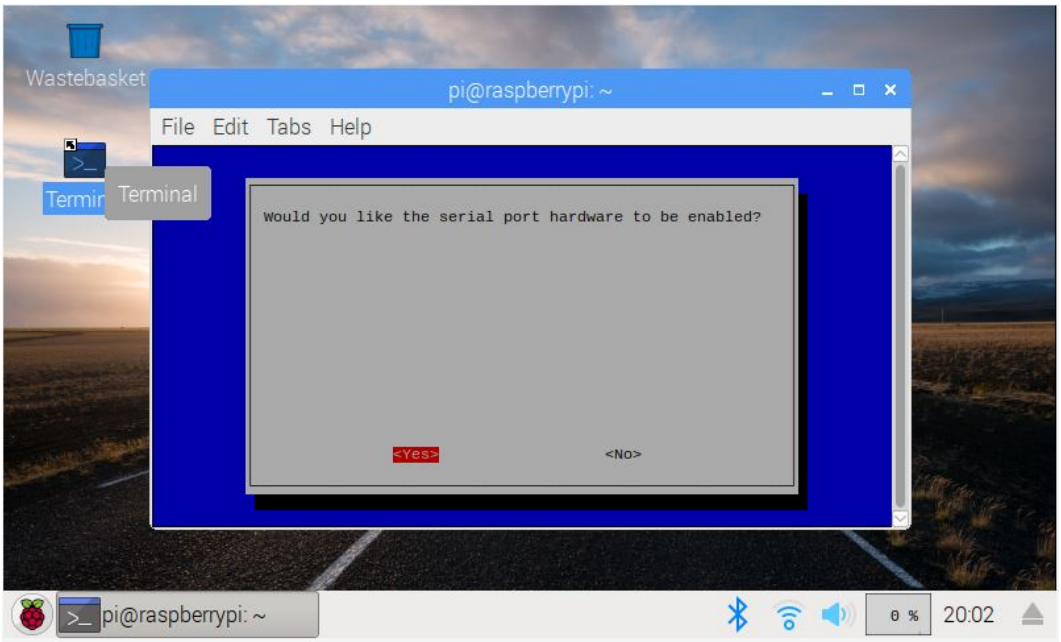

Figura 3.19: Activación del puerto serial.
Fuente: Elaboración propia.

Una vez que se realizaron los pasos anteriormente mencionados se procede a reiniciar la Raspberry.

Nuevamente se abre la consola y se configura un archivo para ajustar el reloj de la UART para ello se edita un archivo escribiendo el comando **sudo nano /boot/config.txt**. En dicho archivo se debe adicionar los siguientes comandos al final del texto **core_freq=250** y **enable_uart=1** tal como se aprecia en la figura 3.20.

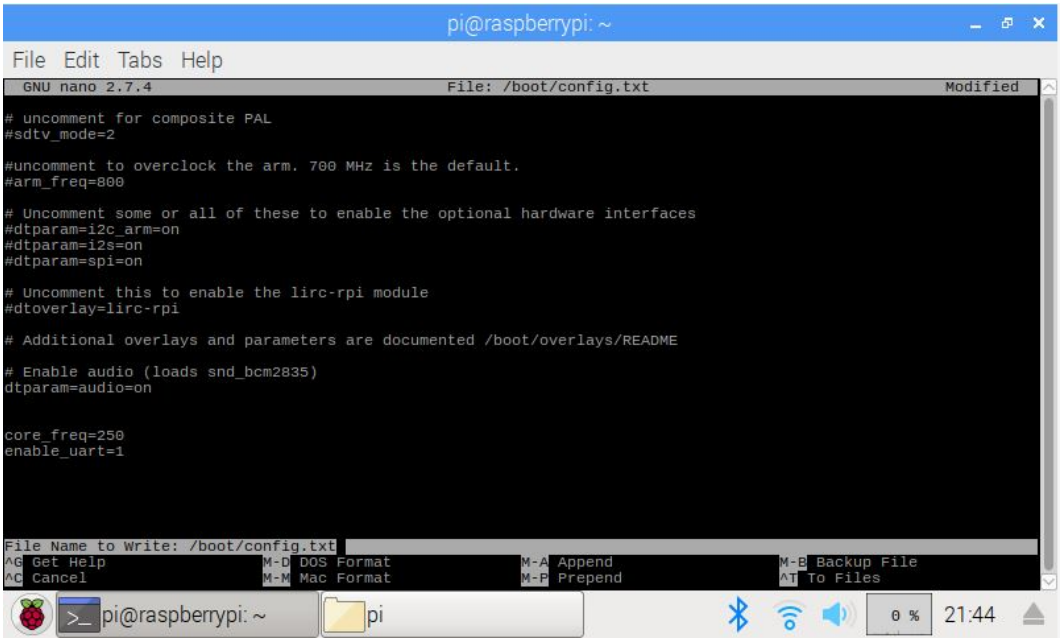

Figura 3.20: Ajuste del reloj de la UART.
Fuente: Elaboración propia.

Se guardaron los cambios hechos en el archivo saliendo de la consola presionando CTRL+C y se reinicio nuevamente la tarjeta.

3.3.2. Programación en Python para la comunicación serial con RS-232

Una vez que la instalación se completó, se procede a programar en Python el envió de los datos mediante la comunicación serial RS-232, para eso fue necesario importar la librería Serial con la línea de código **import serial**, posteriormente se configura el puerto serial con la línea de código **ser = serial.Serial(/dev/ttyS0", baudrate = 9600, timeout=1)**, donde ser es la variable que escribirá los datos.

El envió de los datos se hará mientras se presiona algún botón de la interfaz gráfica ya sea arriba, abajo, izquierda o derecha, por lo tanto y como se mencionó en la sección 3.1.4, cuando el botón es presionado manda a llamar a una función, entonces dentro de la función se escribirá el comando **ser.write(str.encode("UP")))** en el caso de que el botón presionado fuese el de

arriba.

Una vez el botón haya enviado el dato es necesario observar si dicho dato es correcto, para eso fue necesario utilizar una tarjeta TF232 para recibir el dato y monitorearlo a través del software Hercules instalado en la computadora, por lo que se tuvo que hacer una conexión entre la tarjeta FT232 y la Raspberry Pi, tal como se muestra en la figura 3.21.

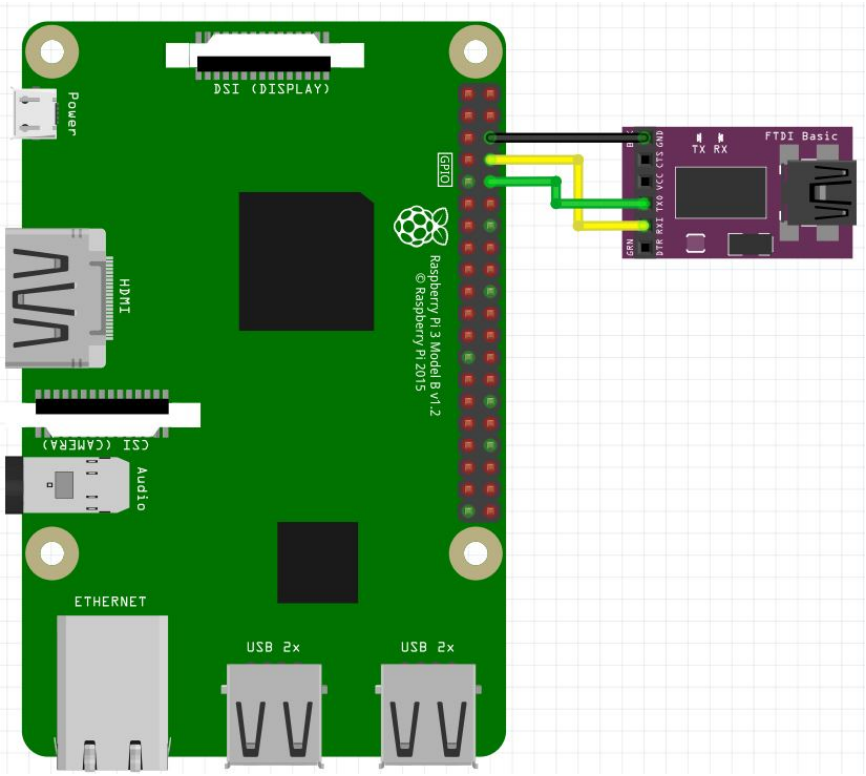

Figura 3.21: Ajuste del reloj de la UART.
Fuente: Elaboración propia.

Con la tarjeta FT232 conectada a la computadora se procedió a abrir la comunicación con el software Hercules, ajustando los paramentos necesarios como los baudios de transmisión el, el puerto COM al que esta conectada la tarjeta, por lo que cuando se presiona el botón la Raspberry envía un “UP”, y hercules lo recibe como un “UP” (véase la figura 3.22) lo cual indico que la configuración y programación para la comunicación serial es correcta.

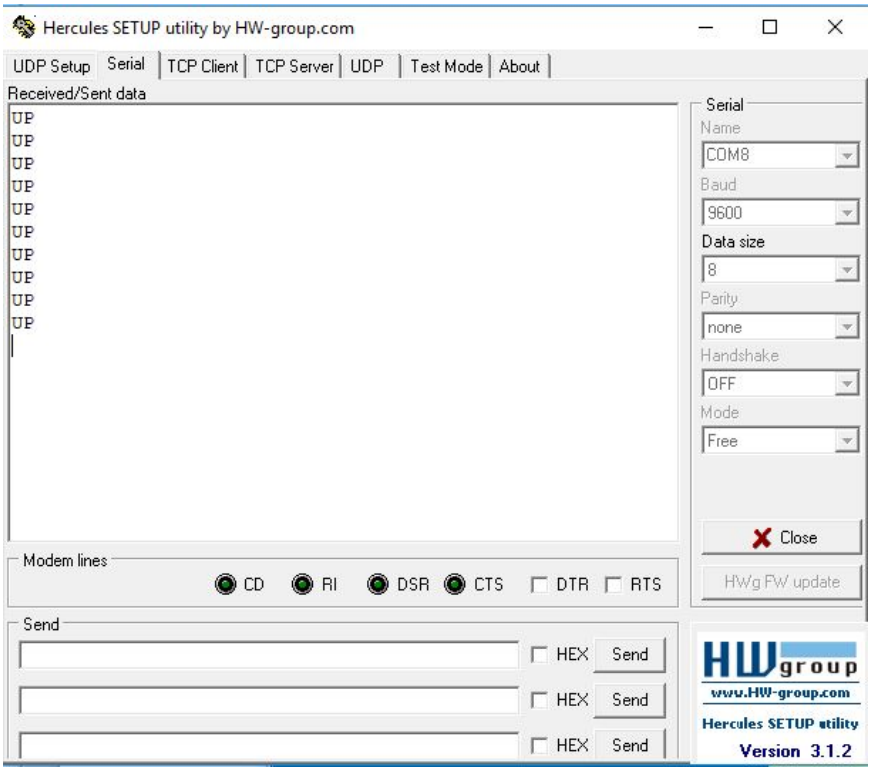

Figura 3.22: Recepción de datos por parte de Hercules desde la Raspberry.
Fuente: Elaboración propia.

El método se repite para los botones restantes (abajo, izquierda y derecha) solo que ahora el dato enviado fue “DOWN”, en el caso de el botón con dirección hacia abajo, “LEFT”, en el caso del botón con dirección hacia la izquierda y “RIGHT”, en el caso del botón con dirección a la derecha. En el código 3.4 se muestra la programación de la parte de envió de datos para el movimiento del robot.

Código 3.4: Programación para el envío de datos para el ajuste de los parámetros del robot

```
from tkinter import *
import serial
import time
import os

er = serial.Serial("/dev/ttyS0", baudrate = 9600, timeout=1)
Raiz=Tk()
nico=0
Raiz.title ("Interface Grafica V0.2")
Raiz.resizable(False, False)
```

```
Raiz.config(bg="#D9D9D9")
Raiz.geometry("800x480")
cidesi = PhotoImage(file="cidesi2.png")
fondo = Label(Raiz, image=cidesi)
fondo.place(x=140, y=140)

    def Manual():
        ManualFrame=Frame()
        ManualFrame.place(x=0, y=70)
        ManualFrame.config(bg="white")
        ManualFrame.config(width="800", height="480")

        TextoManual=Label(ManualFrame, text="MODO MANUAL", font=("
           Arial",20), fg="black", bg="white")
        TextoManual.place(x=320, y=0)



        Operacion_ManualFrame=Frame()
        Operacion_ManualFrame.place(x=0, y=100)
        Operacion_ManualFrame.config(bg="#0089CC")
        Operacion_ManualFrame.config(width="797", height="348")
        Operacion_ManualFrame.config(bd=2)
        Operacion_ManualFrame.config(relief="groove")

        Boton_Inicia_Medicion_Manual=Button(Raiz, text="MEDICION",
                                    width="8", height="2",
                                    font="Arial",
                                    bg="#F2FA85")
        Boton_Inicia_Medicion_Manual.place(x=350, y=390)

        Cuadrante=Canvas(Operacion_ManualFrame, width="150", height
           ="150", bg="#0089CC")
        Cuadrante.place(x=320, y=40)
        Cuadrante.create_line(0, 75, 150, 75)
        Cuadrante.create_line(75,0,75,150)

        def arriba():
            ser.write(str.encode("UP"))
```

```
Boton_Up = Button (Operacion_ManualFrame, text="", width="1
   ", height="1",font=("Arial",25), bg="white", command =
   arriba)
Boton_Up.place(x=90, y=190)

def abajo():
    ser.write(str.encode("DOWN"))

Boton_Down = Button (Operacion_ManualFrame, text="", width=
   "1", height="1",font=("Arial",25), bg="white", command =
    abajo)
Boton_Down.place(x=90, y=270)

def derecha():
    ser.write(str.encode("RIGHT"))

Boton_R = Button (Operacion_ManualFrame, text="", width="1"
   , height="1",font=("Arial",25), bg="white", command =
   derecha)
Boton_R.place(x=690, y=225)

def izquierda():
    ser.write(str.encode("LEFT"))

Boton_L = Button (Operacion_ManualFrame, text="", width="1"
   , height="1",font=("Arial",25), bg="white", command =
   izquierda)
Boton_L.place(x=610, y=225)

Tiempo_Bomba=Label(Operacion_ManualFrame, text="Tiempo de
   funcionamiento de la bomba: " +str(valor.get())+"
   Segundos")
Tiempo_Bomba.place(x=15, y=20)

Potencia_Bomba=Label(Operacion_ManualFrame, text="Potencia
   de la bomba: " +str(Valor_Potencia.get())+"%")
Potencia_Bomba.place(x=15, y=40)
```

```
        Distancia_Robot=Label(Operacion_ManualFrame, text="
            Distancia en modo automatico: " +str(Valor_Distancia.get
            ())+" CM.")
        Distancia_Robot.place(x=15, y=60)


    Boton_Modo_Manual = Button (FrameBoton,
                                text="MODO \nMANUAL",
                                width="10", height="3",
                                font="Arial",
                                bg="white",
                                activebackground="#C0C0C0",
                                relief=RAISED, command=Manual)
    Boton_Modo_Manual.place(x=0, y=0)
```

Una vez que los datos enviados por los botones de dirección del robot han sido visualizados en Hercules, se procede a enviar los datos desde el menú de ajustes, como se vio en la sección 3.1.5 los SLIDERS son los encargados de enviar datos de tipo entero del 0 al 100, por lo tanto, lo que seria enviado por el puerto serial fueron números enteros del 0 al 100, el inconveniente aquí fue que al ser tres SLIDERS que envían números enteros no se puede identificar cual SLIDER es el que esta enviando en ese momento, la solución a este problema fue enviar una letra diferente por cada SLIDER, y concatenar el valor numérico y así enviarlo por el puerto serial, por ejemplo, si se quiere enviar el valor del tiempo de la bomba, se escribe el código **ser.write(str.encode("T"+enviar))** donde la variable enviar toma el valor del SLIDER.

Una vez que se envían los datos por parte de la Raspberry, estos pueden ser visualizados tal como se muestra en la figura 3.22.

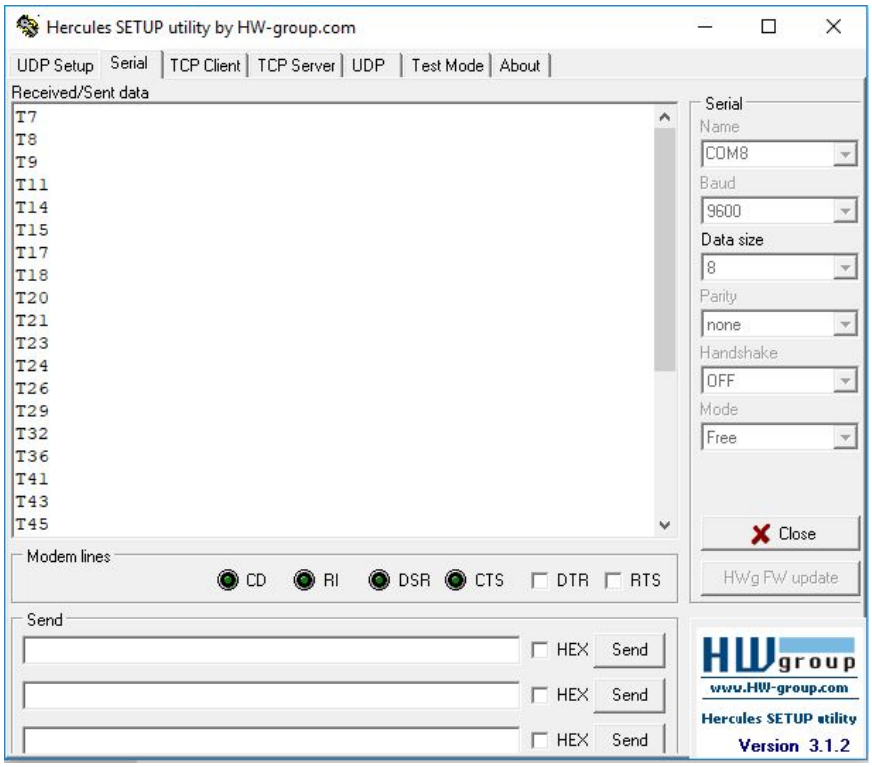

Figura 3.23: Recepción de datos de Hercules desde la Raspberry para el ajuste de los parámetros del robot.

Fuente: Elaboración propia.

En el codigo 3.5 se muestra la programación para el envio de datos por medio de los SLIDERS.

Código 3.5: Programación para el envío de datos para el movimiento del robot

```
from tkinter import *
import serial
import time
import os

er = serial.Serial("/dev/ttyS0", baudrate = 9600, timeout=1)
Raiz=Tk()
nico=0
Raiz.title ("Interface Grafica V0.2")
Raiz.resizable(False, False)
Raiz.config(bg="#D9D9D9")
Raiz.geometry("800x480")
cidesi = PhotoImage(file="cidesi2.png")
fondo = Label(Raiz, image=cidesi)
```

```
fondo.place(x=140, y=140)

    valor=IntVar()
    var=IntVar()
    Tiempo=IntVar()
    nico=0
    def Get():
        Tiempo= str(valor.get())
        print(Tiempo)
    def Get_Tiempo(valor):
        Tiempo= str(valor)
        print ("TIEMPO: "+(valor))
        enviar = str(valor)
        ser.write(str.encode("T"+enviar+"\n"))
    valor.set(5)

    def Tiempo():
        TiempoFrame=Frame()
        TiempoFrame.place(x=0, y=140)
        TiempoFrame.config(bg="#0089CC")
        TiempoFrame.config(width="796", height="309")

        TextoTiempo=Label(TiempoFrame, text="TIEMPO DE
           FUNCIONAMIETNO DE LA BOMBA", font=("Arial",10), fg="
           black", bg="#0089CC")
        TextoTiempo.place(x=280, y=0)

        Barra_Tiempo=Scale(TiempoFrame, label="TIEMPO DE LA BOMBA (
           Seg.)", orient=HORIZONTAL,
                  width=30, length=250, fg="black", bg="white",
                     from_=5, to=60, takefocus=0,
                  variable=valor, command=Get_Tiempo)
        #Guardar=Button(TiempoFrame, text="Guardar", font=("Arial",
            10), bg="white", command=Get)
        #Guardar.place(x=270, y=170)
        Barra_Tiempo.place(x=270, y=100)

    Valor_Potencia=IntVar()
```

```
Potencia=IntVar()
def Get_Potencia():
    Potencia=str(Valor_Potencia.get())
    print(Potencia)
def Get_Potencia2(Valor_Potencia):
    Potencia=str(Valor_Potencia)
    print("POTENCIA: "+(Valor_Potencia))
    enviar = str(Valor_Potencia)
    ser.write(str.encode("P"+enviar+"\n"))
Valor_Potencia.set(10)

def Potencia():
    PotenciaFrame=Frame()
    PotenciaFrame.place(x=0, y=140)
    PotenciaFrame.config(bg="#0089CC")
    PotenciaFrame.config(width="796", height="309")

    TextoPotencia=Label(PotenciaFrame, text="POTENCIA DE LA
       BOMBA", font=("Arial",10), fg="black", bg="#0089CC")
    TextoPotencia.place(x=300, y=0)

    Barra_Potencia=Scale(PotenciaFrame, label="POTENCIA DE LA
       BOMBA (%)", orient=HORIZONTAL,
                width=30, length=250, fg="black", bg="white",
                   from_=10, to=100, takefocus=0,
                variable=Valor_Potencia, command=Get_Potencia2)
    Barra_Potencia.place(x=270, y=100)

Valor_Distancia=IntVar()
Distancia=IntVar()
def Get_Distancia():
    Distancia=str(Valor_Distancia.get())
    print(Distancia)
def Get_Distancia2(Valor_Distancia):
    Distancia=str(Valor_Distancia)
    print("DISTANCIA: "+(Valor_Distancia))
    enviar = str(Valor_Distancia)
    ser.write(str.encode("D"+enviar+"\n"))
```

```
    Valor_Distancia.set(0)

    def Distancia_Automatica():
        Distancia_AutomaticaFrame=Frame()
        Distancia_AutomaticaFrame.place(x=0, y=140)
        Distancia_AutomaticaFrame.config(bg="#0089CC")
        Distancia_AutomaticaFrame.config(width="796", height="309")

        TextoDistancia_Automatica=Label(Distancia_AutomaticaFrame,
           text="DISTANCIA AUTOMATICA", font=("Arial",10), fg="
           black", bg="#0089CC")
        TextoDistancia_Automatica.place(x=0, y=0)

        Barra_Distancia=Scale(Distancia_AutomaticaFrame, label="
           DISTANCIA AUTOMATICA DEL ROBOT(CM.)", orient=HORIZONTAL,
                    width=30, length=250, fg="black", bg="white",
                       from_=0, to=100, takefocus=0,
                    variable=Valor_Distancia, command=
                       Get_Distancia2)
        Barra_Distancia.place(x=270, y=100)
```

3.4. Comunicación serial entre Raspberry Pi y LabVIEW

LabVIEW es un software de ingeniería de sistema que requiere pruebas, medidas y control con acceso rápido a hardware e información de datos. LabVIEW ofrece un enfoque de programación gráfica que le ayuda a visualizar cada aspecto de su aplicación, incluyendo configuración de hardware, datos de medidas y depuración. Esta visualización hace que sea más fácil integrar hardware de medidas de cualquier proveedor, representar una lógica compleja en el diagrama, desarrollar algoritmos de análisis de datos y diseñar interfaces de usuario personalizadas.

3.4.1. Programación del diagrama a bloques en LabVIEW para la recepción de datos de dirección del robot y ajustes

Para hacer de manera mas visual y representativa la transmisión de los datos por parte de la Raspberry Pi, se optó por desarrollar un programa en LabVIEW en el cual se recibirán mediante

la programación para recepción de datos mediante puerto serial.

Para la recepción de los datos por parte de LabVIEW se utilizo el bloque llamado **VISA configure Serial Port** y en sus terminales **VISA resource name** y **baud rate** creamos un control, el control **VISA resource name** nos permite visualizar los puertos COM que están disponibles en la computadora, ahí se selecciono el puerto COM al que esta la tarjeta FT232, mientras que en el control **baud rate** se seleccionan los baudios a los que serán recibidos los datos, en este caso se configuran a 9600.

Dentro de un ciclo WHILE, se coloco un bloque llamado **property node**, este boque obtiene y escribe las propiedades y métodos de aplicaciones locales o remotas, en la terminal **Bytes at port** se conecta un bloque llamado **VISA read**, este bloque lee los bytes en el puerto del **property node** y los muestra, para poder visualizarlos, en la terminal **read buffer** se colocó un indicador. Al final la comunicación se cierra con el bloque **VISA close**. En la figura 3.24 se muestra el diagrama de bloques.

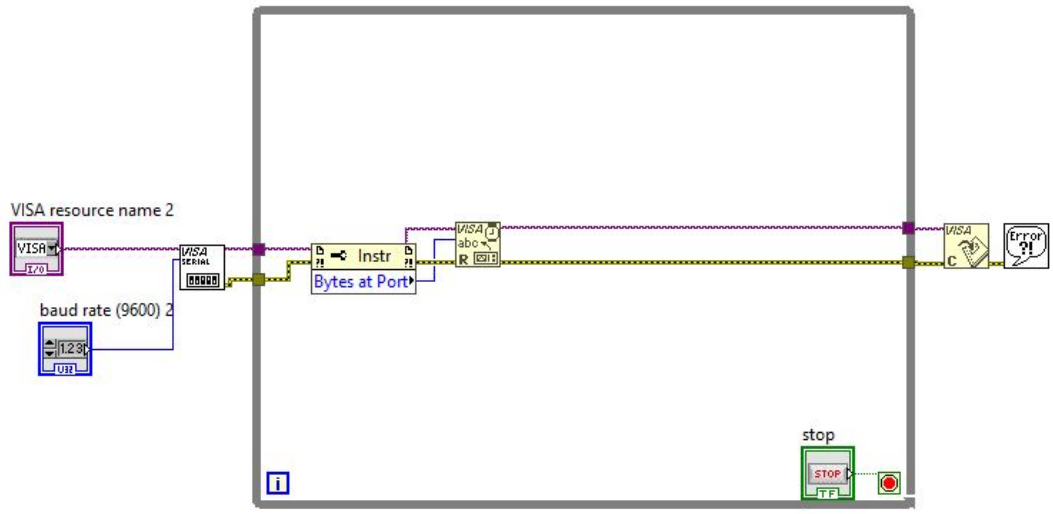

Figura 3.24: Diagrama de bloques para la lectura de datos.
Fuente: Elaboración propia.

Como se mostró en la sección 3.3.2, el dato enviado por la Raspberry es un “UP”, “Down”, “LEFT” y “RIGHT”, lo que significa que son caracteres, por lo tanto lo entregado por el buffer del bloque **read buffer** serán los caracteres UP”, “Down”, “LEFT” y “RIGHT” , para identificar cual dato era el que estaba llegando por el puerto serial y mostrarlo fue necesario igualar la salida del bloque **read buffer** con el dato de interés, por ejemplo, si se quería leer el dato “UP”, la salida del bloque **read buffer** se igualaba a una constante de tipo carácter UP, si el dato de

salida era igual al dato igualado entonces se encendería un indicador LED y el mismo método fue hecho para las direcciones faltantes. En la figura 3.25 se muestra el diagrama de bloques.

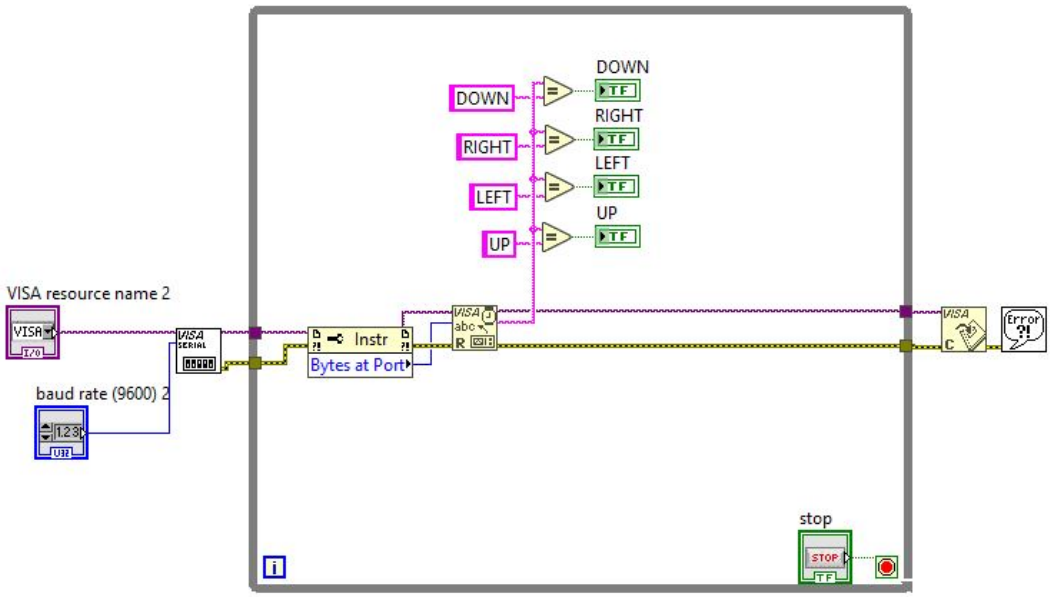

Figura 3.25: Diagrama de bloques para la lectura de datos para la dirección del robot.
Fuente: Elaboración propia.

Después de enviados los datos para la dirección del robot, se procedió a crear un diagrama de bloques para la recepción de los datos enviados para el ajuste de los parámetros del robot, como se mencionó en la sección 3.3.2, los datos enviados son números enteros del 0 al 100, esto por los tres SLIDER, lo que hace que al momento de recibir los datos no sea claro de que SLIDER fueron enviados, por lo que al mover un SLIDER los datos de los otros dos eran modificados, para eso se concateno una letra al dato enviado, dicha letra fue diferente para cada SLIDER.

Cuando LabVIEW recibe el dato, con ayuda del bloque llamado **replace substring**, este bloque tiene como función insertar, eliminar o reemplazar un substring, por lo tanto la terminal **string** se conecta directamente a la terminal **read buffer** del bloque **VISA read**, en la terminal **substring(“”)** se creo una constante con la letra que se quiso leer, por ejemplo, si envio los datos del tiempo de la bomba, el puerto serial recibirá una T seguida del numero en tiempo real del SLIDER, osea T40 (en caso de que el número sea 40), entonces en la constante se le coloca una “T” y en la terminal **lenght (len. of substring** se creo una constante de 1, esto porque se le indica que solo lea la primera posición de todo el string completo, o sea, la “T”.

Una vez llega el dato string al bloque y se lee la primera posición de este, es hora de compáralo con todos los demás, evaluar cual letra es la que fue leída y diferenciar entre sliders.

Usando el ejemplo anterior con la letra “T”, para evaluar si es la letra T, en la terminal **replaced substring** se iguala a la letra que se desea evaluar (“T”), y si es verdadero entonces se envía a un **case structure**, en el cual dentro del caso “T” se coloca el slider de Labview correspondiente al tiempo de la bomba, en caso de que la letra no sea “T”, evalúa si es otra de las que se envían por serial, por ejemplo, “P”, si es verdadero en “P”, entonces el **case structure** se ira al caso “P” y moverá el slider correspondiente de la potencia de la bomba, si es falso en “P”, evaluara si es “D”, y si es verdadero ira al caso “D”. En caso de que ninguno sea verdadero, el caso “DEFAULT” del **case structure** será ejecutado y no se moverá ningún slider en LabVIEW. En la figura 3.26 se muestra el diagrama de bloques.

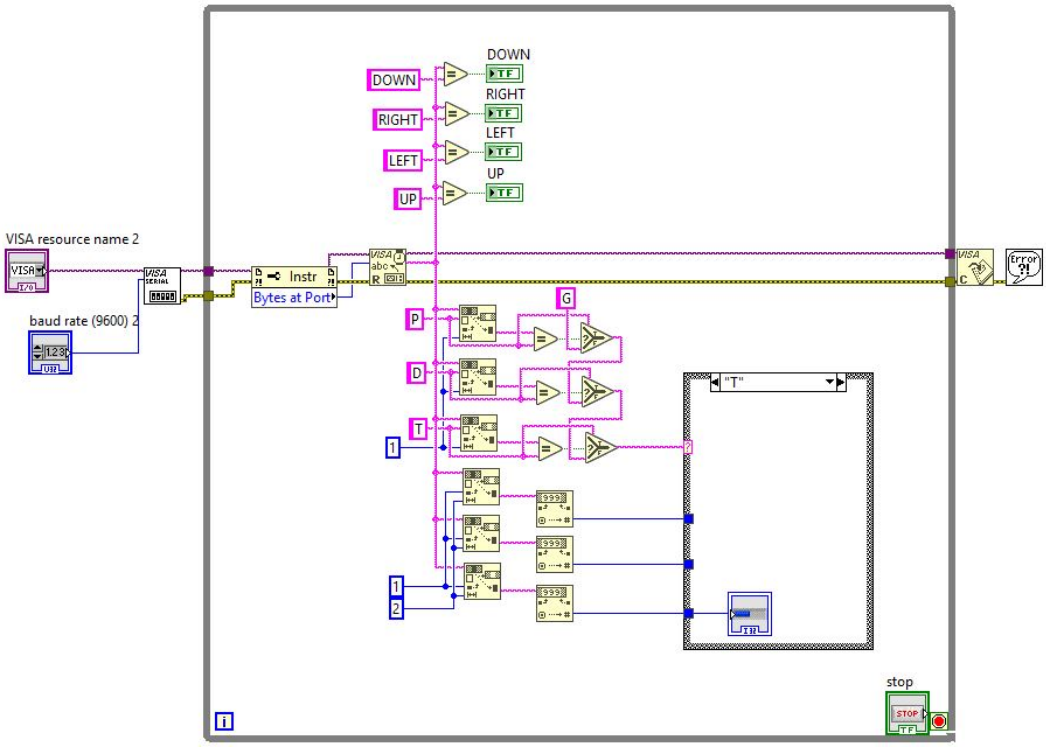

Figura 3.26: Diagrama de bloques para la lectura de datos para el ajuste de parámetros de funcionamiento del robot.

Fuente: Elaboración propia.

3.4.2. Interfaz gráfica en LabVIEW

Para poder visualizar de una manera más ordenada los datos recibidos, se creó una interfaz, con el fin de mostrar gráficamente como es que se recibían los datos, en la interfaz podemos observar que hay cuatro indicadores, uno para cada dirección y tres sliders, uno para cada SLIDER de la interfaz grafica de la Raspberry. En la figura 3.27 se observa dicha interfaz.

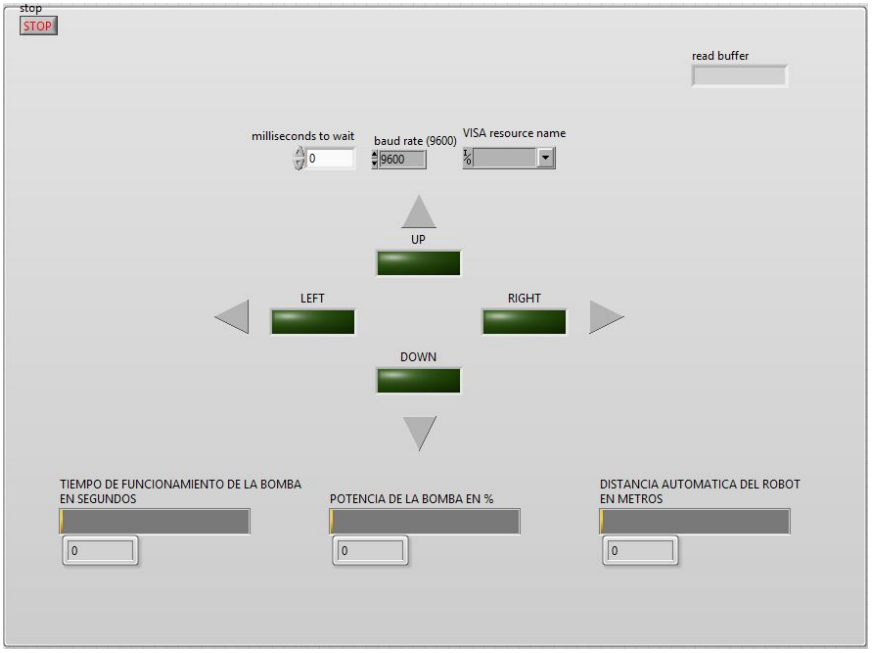

Figura 3.27: Interfaz para la visualización de los datos recibidos en LabVIEW.
Fuente: Elaboración propia.

3.5. Comunicación Serial entre Raspberry Pi 3 y Hércules mediante protocolo RS-485

Una vez validada la comunicación serial RS-232, se procedió a implementar la comunicación serial, pero ahora en con el protocolo RS-485. El porqué del cambio de protocolo es justificado, ya que el protocolo RS-232 es para comunicaciones en rangos cortos, mientras que el protocolo RS-485 es para comunicaciones remotas llegando a alcanzar hasta los 1200 metros de transmisión y recepción de datos. El robot explorador sube tanques de almacenamiento de mas de 500 metros por lo que el protocolo RS-232 no es el apto para implementarlo en el robot, pero si para hacer pruebas de laboratorio.

En la table 3.1 se muestran las diferencias mas importantes entre el protocolo RS-232 y el RS-485.

Cuadro 3.1: Diferencias entre los protocolos RS-232 y RS-485. [9]

Concepto	**RS-232**	**RS-485**
"1"Lógico de la salida (Emisor)	-5 / -15 (volts)	-1.5 / -5 (Volts)
"1"Lógico de la salida (Receptor)	-3 / -15 (volts)	-0.2 / -7 (Volts)
"0"Lógico de la salida (Emisor)	5 / 15 (volts)	1.5 / 5 (Volts)
"0"Lógico de la salida (Receptor)	3 / 15 (volts)	0.2 / 12 (Volts)
Tercer estado en la salida	No	Si
Conexion entre emisor y receptor	No equilibrada (Emisor y Receptor)	Diferencial (Emisor y Receptor)
Máxima tensión aplicada a la linea de salida	+- 25(Volts)	-7 / 12 (Volts)
Número de emisores y receptores en una linea	1 Emisor 1 Receptor	32 Emisores 32 Receptores
Velocidad máxima	20 Kbps/15m	10 Mbps/12m
Velocidad media	-	1 Mbps/120m
Velocidad mínima	-	100 Kbps/1200m

3.5.1. Circuito de prueba para comunicación RS-485

Para realizar la comunicación RS-485, fue necesario diseñar un circuito. Para el diseño se utilizo el circuito integrado **LTC485** el cual es un transceptor de línea/bus de baja potencia diseñado para el estándar de datos multipunto, el cual fue utilizado de aplicación típica, tal como lo muestra la imagen 3.28 [10].

El circuito **LTC485** cuenta con 8 pines, de los cuales el pin 8 es Vcc y el pin 5 es GND. Los pines 1 (**RO**) y 4 (**DI**) van conectados a los pines **RX** y **TX** de la Raspberry respectivamente. En los pines 2 **RE** y 3 **DE** se conecta un pin del GPIO de la Raspberry, este pin cuando esta en alto (1 lógico) habilita la transmisión de los datos por RS-485. En los pines 6 (**B**) y 7 (**A**) se conecta el cable convertidor USB-485. En la figura 3.29 se muestra el circuito que se utilizó para la comunicación RS-485.

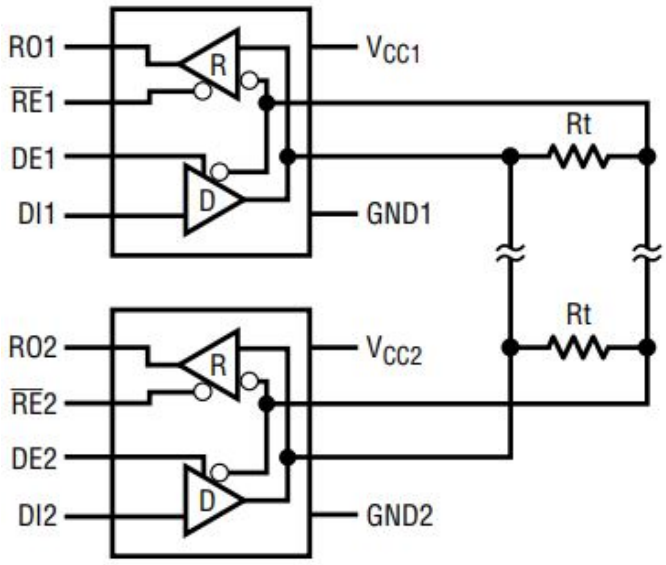

Figura 3.28: Aplicación típica del circuito LTC485.
Fuente: [10].

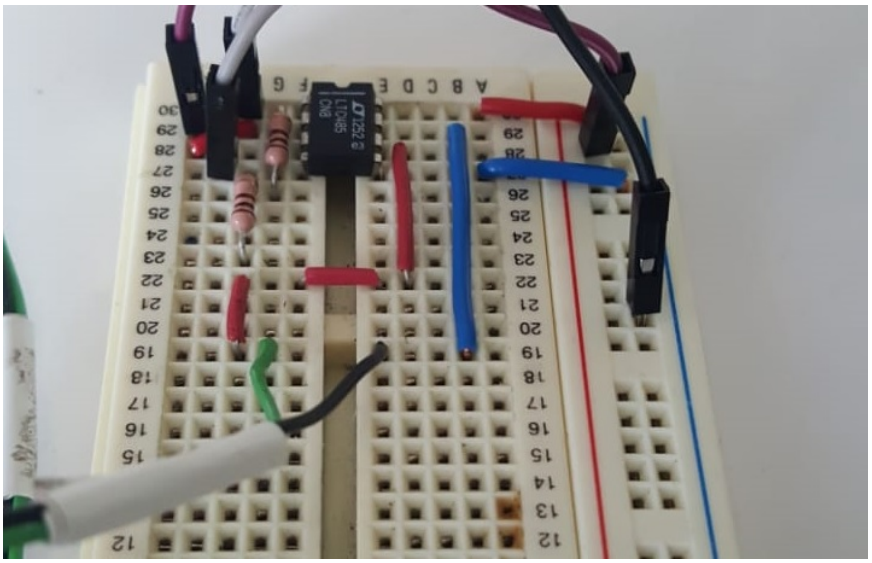

Figura 3.29: Circuto electronico para la comunicacion serial RS-485.
Fuente: Elaboración propia.

3.5.2. Programación para el envió de datos por el protocolo RS-485

La programación en Python para la transmisión de datos por el protocolo RS-485 fue relativamente sencilla, ya que se utilizó el mismo método de programación que en el protocolo RS-232, solo que se añadió un pin del GPIO para habilitar la transmisión de datos por medio de RS-485,

además de que se añadieron algunas configuraciones, como **GPIO.setmode(GPIO.BOARD)** el cual indica que trabajaremos con el numero de pines que viene impreso en la tarjeta, **GPIO.setup(7,GPIO.OUT** el cual indica que se utilizara el pin 7 como salida y su estado inicial será encendido o en 1 lógico, esto porque siempre estará habilitando la recepción de datos por parte del circuito LTC485. En el código 3.6 se muestra la programación para el envío de datos por RS-485.

Código 3.6: Programación para el envío de datos por el protocolo RS-485

```
from tkinter import *
import serial
import time
import os

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7,GPIO.OUT,initial=GPIO.HIGH)

ser = serial.Serial(port="/dev/ttyS0",
                    baudrate = 9600,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,
                    bytesize=serial.EIGHTBITS,
                    timeout=1)
Raiz=Tk()
Raiz.title ("Interface Grafica V0.2")
Raiz.resizable(False, False)
Raiz.config(bg="#D9D9D9")
Raiz.geometry("800x480")
cidesi = PhotoImage(file="cidesi2.png")
fondo = Label(Raiz, image=cidesi)
fondo.place(x=140, y=140)
```

3.6. Diseño de la carcasa para el control remoto

El desarrollo de la carcasa se baso en las tablets modernas, donde se eliminaron los joysticks y los botones físicos que el control de la generación 2 tenia, ya que se encuentran dentro de la interfaz gráfica. Aunque si el cliente requiere que el manejo del robot sea por medio de joystick, se le pueden agregar a la carcasa y mantener el fácil manejo de la interfaz gráfica [11].

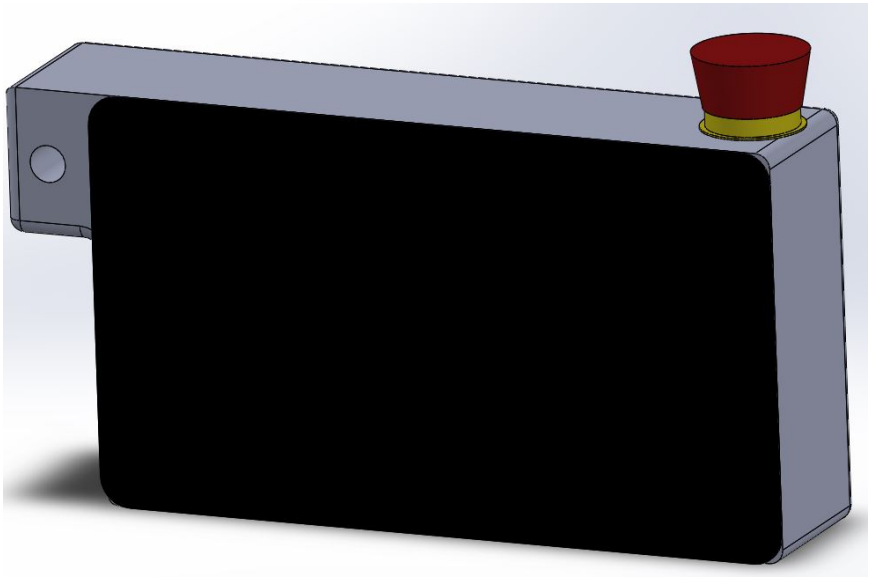

Figura 3.30: Parte frontal de la carcasa con del control remoto.
Fuente: Elaboración propia.

En la figura 3.30 se muestra la parte frontal del control remoto, desarrollada en Solid works.

En la carcasa solo se tiene la pantalla, el botón de paro de emergencia y una muesca, esta muesca tiene como objetivo asegurara que el dispositivo no se caiga cuando el operador este manipulándolo, en el hueco que se puede observar de lado izquierdo en la figura 3.30 se coloca una cinta de seguridad que va amarrada a la muñeca del operador, así, cuando se presione el paro de emergencia o se resbale el control, quede sujete a la muñeca del operador y no sufra daños.

En la figura 3.31 se muestra la parte trasera del control, como se puede observar esta completamente sellado, mientras que contiene cuatro tornillos para el ensamble de la carcasa y la pantalla, dado que la misma pantalla cuenta cuatro orificios para como se muestra en la figura 3.32.

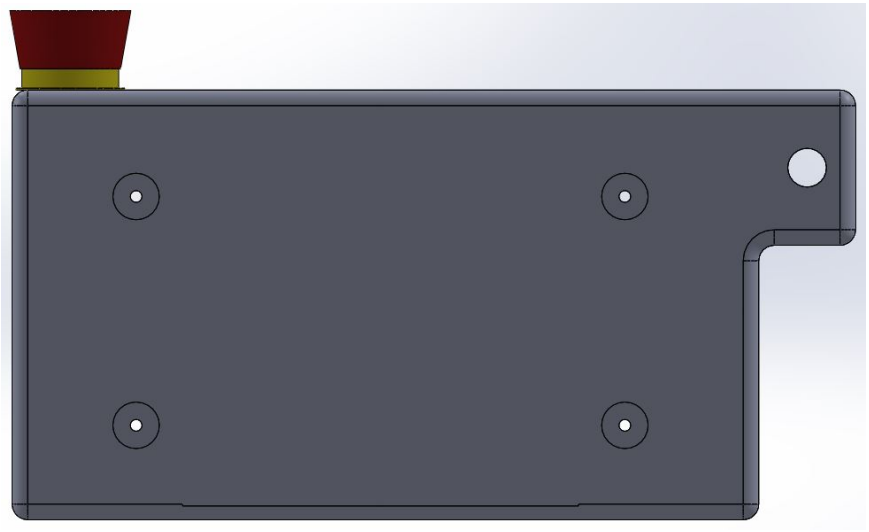

Figura 3.31: Parte trasera de la carcasa con del control remoto.
Fuente: Elaboración propia.

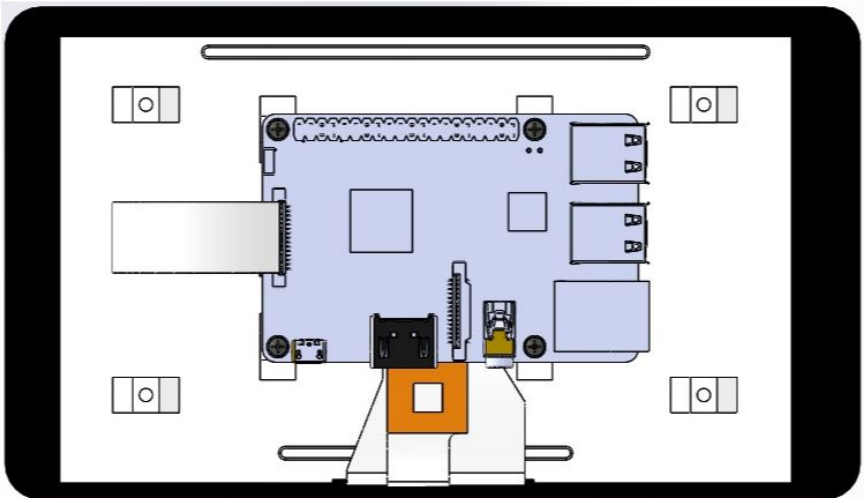

Figura 3.32: Modelado de la pantalla touch.
Fuente: Elaboración propia.

Como se puede apreciar en el diseño, cambio completamente la forma del control remoto con respecto a su antecesora, dando así una mejor presentación y forma de sujeción que harán más confortable la manipulación del robot.

Capítulo 4

Resultados

Como parte final de proyecto fue validarlo mediante las pruebas físicas correspondientes, en este capitulo se muestran algunas imágenes de los resultados obtenidos a lo largo de la experimentación.

4.0.1. Experimentación de comunicación serial entre Raspberry Pi y la computadora mediante el protocolo RS-485

Como experimentación se probó la comunicación serial entre la Raspberry Pi y la computadora con el protocolo RS-485. El software que se encargó de la recepción por parte de la computadora fue LabVIEW, con la interfaz grafica que se mencionó con anterioridad en la sección 3.4.2. En la figura 4.1 se puede apreciar la experimentación con los botones para el control de dirección del robot, donde se observa que mientras se mantiene presionado el botón enciende el indicador en la interfaz de LabVIEW, indicando así que el envío y recepción de datos funciono adecuadamente para la parte del control de dirección del robot.

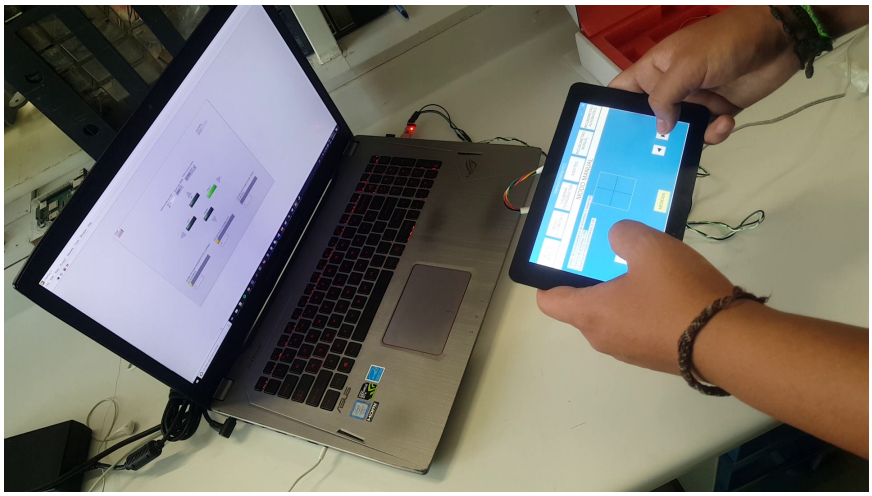

Figura 4.1: Comunicación serial entre la interfaz y la computadora para el control de movimiento del robot.

Fuente: Elaboración propia.

Una vez que se valido el envío de datos entre la interfaz y la computadora para el control del movimiento del robot, se procedió a realizar la comunicación para el ajuste de los parámetros del de funcionamiento del robot. Como se mencionó en la sección 3.1.5, la herramienta para el ajuste de los parámetros se utilizan sliders, por lo tanto, en la interfaz de LabVIEW se colocaron slider para visualizar que los datos enviados por los sliders de la interfaz coincidan con los leídos por la computadora. En la figura 4.2 se aprecia la experimentación con los sliders, donde se puede notar que toman los valores enviados, de igual forma indicando que el envío y recepción de datos para el ajuste de los parámetros de funcionamiento del robot funciona adecuadamente.

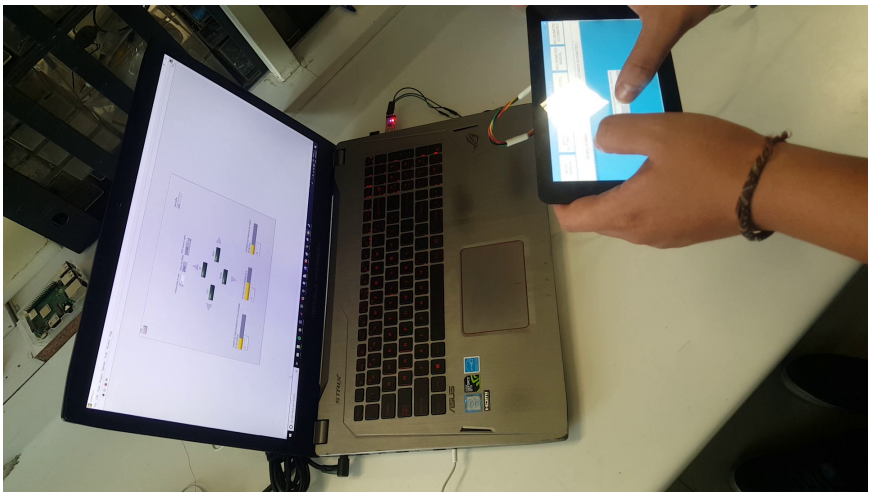

Figura 4.2: Comunicación serial entre la interfaz y la computadora para el ajuste de los parámetros del funcionamiento del robot.
Fuente: Elaboración propia.

4.0.2. Comunicación entre la interfaz y el robot explorador

El robot explorador es controlado por un DSP (por sus siglas en inglés, Digital Signal Processor) al cual la interfaz gráfica debe comunicarse mediante algún protocolo de comunicación (debe recordarse que el protocolo de comunicación es el RS-485). Dentro del DSP hay una programación mas compleja la cual recibe los datos enviados y asigna al robot una tarea, dicha programación corrió a cargo del Dr. Julio César Solano Vargas, quien ha trabajado con dicho código desde el año 2015. En la figura 4.3 se muestra la recepción de los datos por parte del DSP.

Es importante señalar que por motivos de falta de tiempo no se pudieron hacer pruebas de movimiento, por lo que no fue posible manipular el robot con la interfaz gráfica, aunque el hecho de que el DSP reciba los datos enviados desde la interfaz gráfica asegura que es posible controlar el robot con la interfaz, por lo tanto, podemos concluir en que la programación de la interfaz y del protocolo de comunicación fue exitoso.

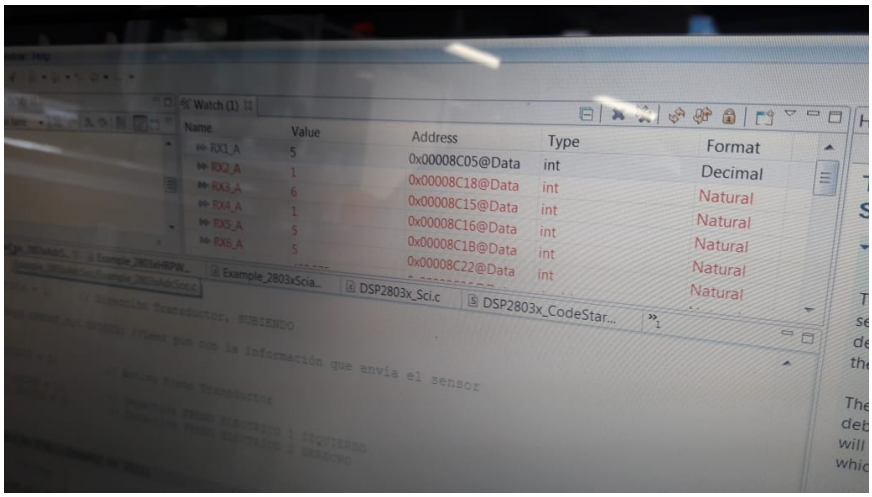

Figura 4.3: Recepción de datos por parte del DSP enviados por la interfaz gráfica.
Fuente: Elaboración propia.

Capítulo 5

Conclusiones

Mejorar la versión anterior del control equipándola con una pantalla más grande, a color y táctil, además de mejorar la visibilidad al usuario, trae consigo bastantes beneficios que se pueden implementar en un futuro, por ejemplo, la integración al robot de una cámara para realización de inspección visual por medio de Python y OpenCV, graficar datos de medición y guardar dichos datos en una base de datos, entre otras funciones.

La elaboración de este proyecto trajo consigo bastantes experiencias que a su vez dieron experiencia en el campo de la investigación y desarrollo de proyectos, como estudiante recién graduado de ingeniería, había huecos que se pretendían llenar y bases que se querían mejorar, sin duda la especialidad tecnólogo en mecatrónica brindo las herramientas necesarias para lograr el objetivo de completar satisfactoriamente este proyecto, además, dejo ver un poco de lo que es la carrera de investigador y dio bases que pueden servir de ayuda para el curso de la maestría en ciencia y tecnología.

Si bien por mi parte el proyecto a concluido, dejo a las nuevas generaciones un tema de investigación bastante interesante, que pueden complementar, innovar y concluir por completo. Hay una infinidad de herramientas que se pueden aplicar dentro de este proyecto, incluyendo métodos de control mas avanzados, periféricos que ayuden a la inspección y herramientas para la iteración humano/robot.

Bibliografía

[1] LLOG SA de CV. Araña ultrasonica, Agosto 2019.

[2] Julio César Solano Vargas. Desarrollo del sistema de control de un robot orientado a la medición de espesores de pared de tanques de almacenamiento. 2002.

[3] LLOG SA de CV. Detector de fallas por ut, Agosto 2019.

[4] Silverwing. Scorpion2 and swift, Agosto 2019.

[5] Silverwing. Automated ultrasonic corrosion mapping, Agosto 2019.

[6] A. Fernández Montoro. *Python 3 al descubierto*. RC Libros, Madrid, 2012.

[7] J.W. Shipman. *Tkinter 8.4 reference: a GUI for Python*. New Mexico Tech Computer Center, 2013.

[8] Raspberry Pi. *Raspberry Pi Official Beginner's Guide*. Raspberrypi.org, Estados Unidos, second edition, 2019.

[9] Néstor Gabriel Forero Saboya. Normas de comunicación en serie: Rs-232, rs-422 y rs-485. *INGENIO*, 2012.

[10] Linear Technology. *LTC485 Low Power RS485*. Linear Technology Corporation, Estados Unidos.

[11] J. Fernández López and F. Diaz del castillo Rodríguez. *Manual de practicas CAD utilizando el programa SOLIDWORKS*. UNAM, Mexico, 2015.