Taller 2

Robótica

Isaac Escobar
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá D.C
Colombia
id.escobar@uniandes.edu.co

Andrés Mugnier Z
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá D.C
Colombia
a.mugnier@uniandes.edu.co

David Pérez
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá D.C
Colombia
dp.perez@uniandes.edu.co

Luis Hernández
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá D.C
Colombia
lf.hernandezr@uniandes.edu.co

Palabras Clave—ROS, Python, simulación, nodos, tópicos, Turtlebot, robot diferencial, Raspberry Pi

I. Materiales

1. **Arduino UNO:** se utilizó este módulo para enviar las señales de control a los motores.
2. **Raspberry Pi 4 Model B:** se integró ubuntu y ROS en esta tarjeta para los protocolos de comunicación entre los comandos de movimiento y la odometría.
3. **L298n:** módulo que envía la potencia a los motores según la señal que recibe del Arduino UNO.
4. **DC Encoder Motor Robotic Car Speed Encoder 9V:** motoreductores que mueven el robot.
5. **Jumpers:** cables usados para las conexiones.
6. **Cable USB:** usado para conectar la Raspberry con el Arduino.
7. **Rueda plástica.**
8. **Rueda loca.**
9. **Tornillos, tuercas y abrazaderas plásticas.**

II. Plano mecánico y distribución eléctrónica

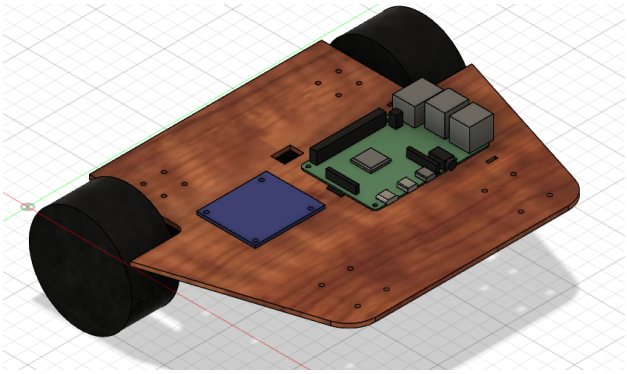

Figura 1: Diseño Fusion 360 - vista diagonal.

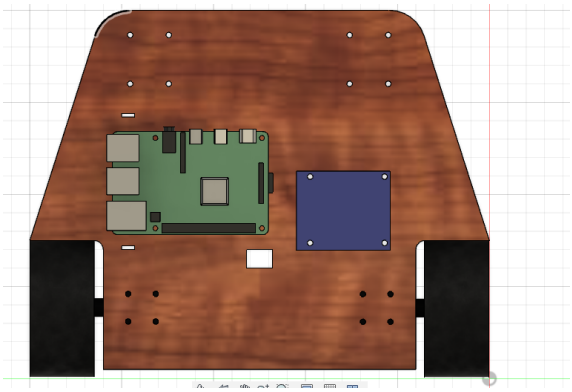

Figura 2: Diseño Fusion 360 - vista superior.

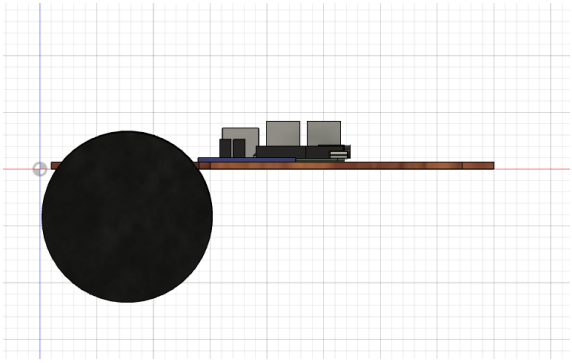

Figura 3: Diseño Fusion 360 - vista horizontal lateral.

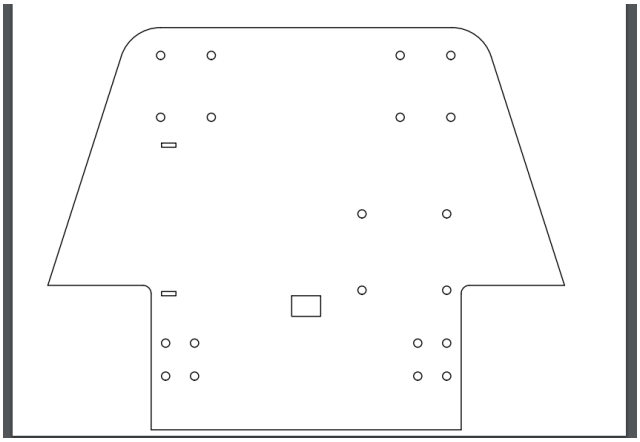

Figura 4: Plano para recorte en mdf de la base del robot.

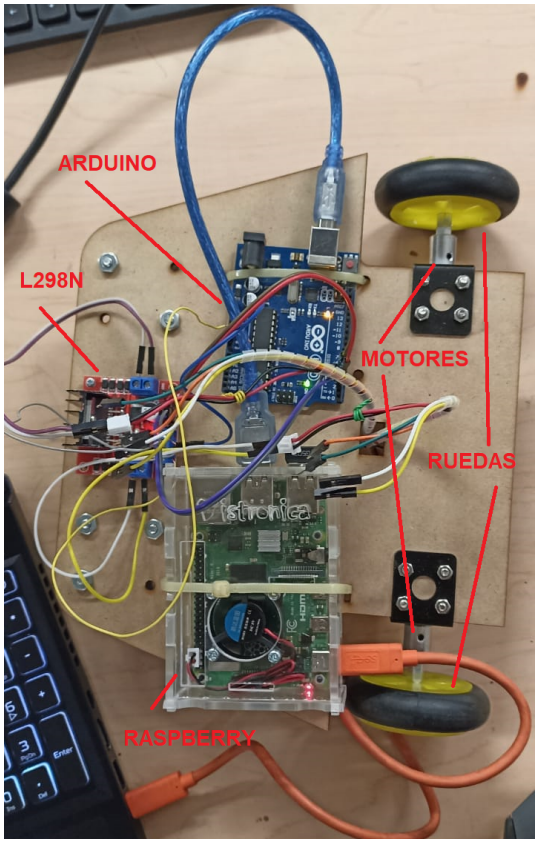

Figura 5: Plano de la distribución electrónica.

III. Movimiento: velocidad lineal y angular

Para la primera parte se creó un nodo en ROS llamado *teleop*, el cual permitió al usuario controlar el robot diferencial. Se introduce en consola las acciones que se quieren realizar con las teclas W,A,S,D. Estas se publican como string al topico *Flash_Velocidades*.

Adicionalmente, se utilizaron las librerias ros_lib de arduino, las cuales permiten comunicación entre arduino y ros. Para esto, se realizó un código en arduino que crea un nodo que se suscribe al topico *Flash_Velocidades*. Posteriormente, dependiendo del texto que obtiene del topico, envía determinadas señales al puente H, las cuales controlan la dirección y velocidad de rotación de los motores del robot diferencial.

Se escribió el nodo *teleop* como un archivo .py, donde se implementó una función que interpreta las teclas w, a, s, y d, como los comandos para el movimiento del robot. El código imprime en consola el comando que se está enviando, es decir «arriba», «abajo», «izquierda», o «derecha», según la tecla que se esté presionando. En el Code Listing 1 se muestra una porción del código para las teclas w y s, las cuales son el movimiento de arriba y abajo respectivamente. Por otro lado, también se tuvo en cuenta la acción de no presionar ninguna tecla, donde se imprimirá el mensaje «neutral» y el robot no hará ninguna acción, esto se observa en el Code Listing **??**.

```
def on_press(key):
    global vel_msg
    global current_msg
```

```
    print(speed)
    try:
        k = key.char  # single-char keys
    except:
        k = key.name  # other keys
    if (k == "w"):
        vel_msg.linear.x = speed
        vel_msg.angular.z = 0
        current_msg = "Arriba"
    elif (k == "s"):
        vel_msg.linear.x = -speed
        vel_msg.angular.z = 0
        current_msg = "Abajo"
```

Code Listing 1: Comando de movimiento presionando tecla.

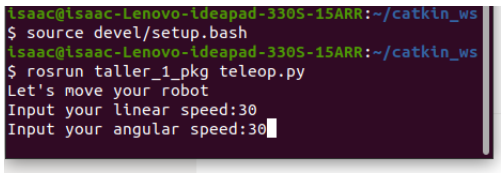

Figura 6: Nodo teleop.py en funcionamiento

En la Figura 7 se muestra el diagrama con la conexión entre el nodo *teleop*, que le envía los comandos de movimiento (velocidad angular o lineal) al tópico *turtlebot_cmdVel*, y el nodo *sim_ros_interface*, el cual se suscribe al tópico, recibe la información, y simula el robot y el entorno.

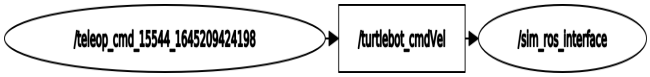

Figura 7: Diagrama de nodos y tópicos activos.

A continuación se presenta el enlace para el video de funcionamiento:

IV. Obtención de posición en el marco global

Para esta acción se diseñó un nodo encargado de transformar las velocidades en coordenadas en el marco global, para esta acción se realiza una suscripción a Flash_velocidad en la que dependiendo de la acción realizada por el usuario, es decir presionar W, A, S o D, se usara la función de transformación para obtener las posiciones en X y Y. Posteriormente, se publican las posiciones Flash_bot_position, las cuales se encargan de mostrar gráficamente el recorrido del robot (en el marco Global)

V. Interfaz: gráfica de posición

Se creó un segundo nodo, como archivo .py (Figura 8), que permitió mostrar una gráfica con la posición en tiempo real del robot en la simulación.

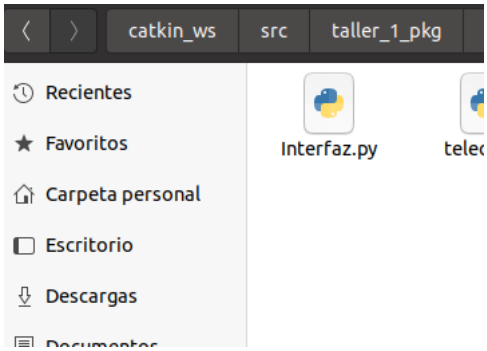

Figura 8: Nodo Interfaz.py guardado en la carpeta scripts

En el Code Listing 2 se muestra la función que se encarga de graficar los datos del movimiento del robot según el marco de referencia de la simulación. Además, añade también el título de «Trayectoria del Robot», el color, y el estilo, que en este caso son puntos rojos.

```
def animate(i):
    global x_values
    global y_values

    x_values.append(x_current)
    y_values.append(y_current)
    plt.cla()
    plt.scatter(x_values, y_values,label= "Turtlebot
    ", color= "red",marker= "o", s=30)
    plt.xlim(-2.5,2.5)
    plt.ylim(-2.5,2.5)
    plt.xlabel("Posicion en x")
    plt.ylabel("Posicion en y")
    plt.title("Trayectoria del Robot")
    plt.legend()
    plt.grid(color = 'black', linestyle = '--',
    linewidth = 0.5)
    plt.grid(True)

    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999',
     linestyle='-', alpha=0.2)
```

Code Listing 2: Función que grafica la posición del Turtlebot.

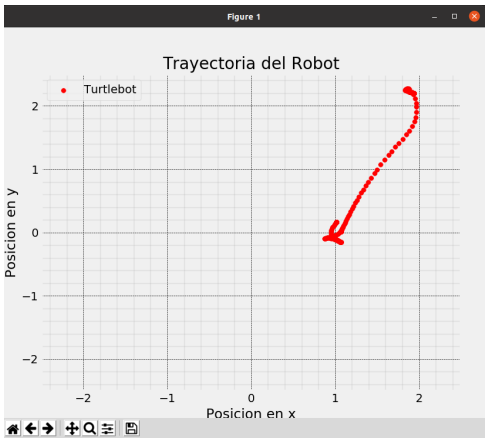

Figura 9: Trayectoria del moviento del robot.

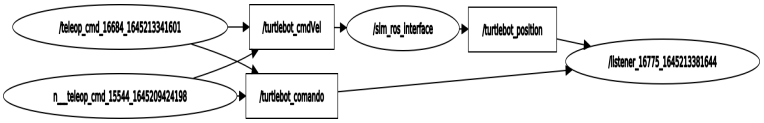

Figura 10: Diagrama de nodos y tópicos activos.

VI. Archivo: almacenamiento de posición

Se modificó el nodo de la interfaz (*Interfaz.py*) para tener la funcionalidad de que se le pregunte al usuario, a través del cmd (Figura 11), si desea guardar el recorrido del robot. En el caso de marcar «n» no sucede nada y la simulación continua normalmente. En el caso de marcar «s» se crea un archivo .txt con los comandos de movimiento que recibe el robot en el tiempo que dura la simulación.

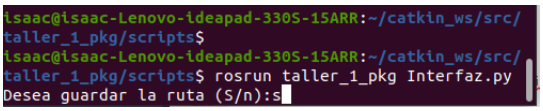

Figura 11: Corriendo Interfaz.py en el cmd.

Cuando la simulación está activa el nodo *Interfaz* recibe los comandos de las teclas para el movimiento del robot (Code Listing 3), crea un archivo .txt (Code Listing 4), y escribe los comandos en el archivo (Code Listing 5, Figura 14). Se le pregunta al usuario qué nombre tendrá este archivo .txt y se guarda en una ubicación (Figura 13).

```
def callback(vel_msg):
    global x_current
    global y_current

    x_current = vel_msg.linear.x
    y_current = vel_msg.linear.y
```

Code Listing 3: Recepción de los mensajes de comandos de movimiento.

```
if __name__ == '__main__':
    decision = input("Desea guardar la ruta (S/n):")

    if ((decision == "S")|(decision == "s")):
        f= open("test.txt","w+")
```

Code Listing 4: Creación del archivo .txt.

```
def callback2(current_msg):
    global decision
    if ((decision == "S")|(decision == "s")):
        f.write(str(current_msg)+"\r\n")
```

Code Listing 5: Escritura de los mensajes recibidos en el archivo .txt.

Figura 12: Lista de comandos durante la simulación en el cmd.

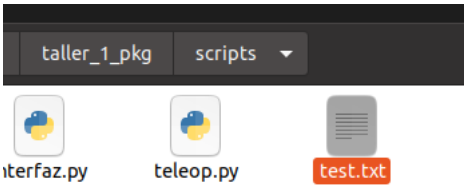

Figura 13: Archivo con los comandos, guardado en la carpeta donde se corre la interfaz.

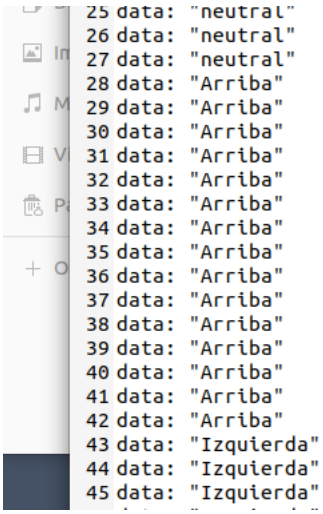

Figura 14: Lista de comandos durante la simulación en el archivo .txt.

A continuación se presenta el enlace para el video de funcionamiento de las dos últimas secciones:
https://youtu.be/OgQ296ToB9k

VII. Funcionamiento General

El robot integra una serie de códigos en Python y en C (Arduino) los cuales son llamados a través de ROS. En

primera instancia se tiene el control del robot, en el que a través de las teclas W, A, S, D se manipula la dirección del robot, de forma que cada vez que se presiona o suelta una tecla se envía constantemente, estas son recibidas por el Arduino a través de una suscripción para el inmediato movimiento de los motores.

En cuanto al entorno grafico se hace uso de una transformación, en la que a partir de las velocidades lineales de cada rueda se arma la trayectoria en el marco global que esta recorriendo el robot.

Finalmente, para la repetición de una partida, se genera la pregunta al usuario en la que decidirá si guardar o no los siguientes movimientos a realizar. En caso de que así sea se procederá a crear un archivo de texto, bajo el nombre que el usuario decida, en el que se almacenaran todos los movimientos que se realicen. Una vez guardado, será posible llamar el servicio que hará lectura del archivo de texto indicado por el usuario, el cual naturalmente replicará los movimientos guardados en el archivo permitiendo también observar la grafica de movimiento, si el usuario así lo desea.

VIII. Integración Parte Electrónica y Mecánica

Para facilitar la comunicación con motores y próximamente sensores, se hizo uso de un Arduino Uno el cual funciona como intermediario entre la Raspberry y las partes mecánicas. La comunicación funciona de forma que cuando la Raspberry da una orden de movimiento, es decir se presiona una tecla o se reproduce un servicio de partida guardada, el Arduino indica que motores deben arrancar. Es importante mencionar que para el correcto funcionamiento de los motores y por protección del Arduino se hizo uso de un puente H, el cual alimenta los motores a través de una batería.

IX. Archivo: reproducción de movimiento

Para este numeral, se creó el nodo /turtle_bot_player, el cual toma el archivo .txt anteriormente creado y a partir de este reproduce la secuencia de acciones guaradadas. La partida a reproducir se solicita a partir de un servicio. Para poder ofrecer el servicio, se creo el archivo reproducir_partida.srv, el cual se muestra a continuación.

```
string nombre_archivo
---
string resultado
```

Figura 15: Archivo srv, nodo turtle_bot_player

Como se puede observar, se define como argumento de entrada un string correspondiente al nombre del archivo, y como salida un string correspondiente al resultado. En este caso no se necesita un resultado explicito por lo cual se utiliza esta variable para indicar el momento en el que finaliza la ejecución de la secuencia de acciones.

El codigo principal del nodo que ofrece el servicio se realiza dentro de la función handle_reproducir_partida() la cual se muestra a continuación.

```
def handle_reproducir_partida(req):
    global current_msg
    global vel_msg
    global content
    global contador
    global speed
    global angular
    pub = rospy.Publisher('turtlebot_cmdVel', Twist, queue_size=10)
    nombre_archivo = req.nombre_archivo
```

Figura 16: Función principal, nodo turtle_bot_player

Al correr el nodo, se ofrece el servicio reproducir_partida", cuyo callback corresponde a la función mencionada anteriormente. Como se puede ver, la función recibe como parametro la variable req, de la cual se obtiene el nombre del archivo txt a reproducir. Posteriormente, abre el archivo txt y obtiene de este los valores de velocidad lineal y angular, los cuales se encuentran en las últimas 2 filas del archivo. Acto seguido, obtiene un vector con las acciones guardadas en cada linea del archivo y realiza un recorrido iterativo, con frecuencia 10Hz, para ejecutar cada acción secuencialmente. Este recorrido se observa a continuación.

```
content = content[:-2]
iteraciones = len(content)
print(iteraciones)

    # MAIN LOOP
while not rospy.is_shutdown():
    for i in range(iteraciones):
        move()
        rospy.loginfo(current_msg)
        pub.publish(vel_msg)

        contador += 1

        rate.sleep()
    break
```

Figura 17: Recorrido principal, nodo turtle_bot_player

Como se puede ver, en cada iteración se corre la función move, la cual actualiza las variables globales current_msg y vel_msg a partir del vector content y posteriormente se imprime la accion a realizar (current_msg) en la terminal, y se publica la variable tipo twist (vel_msg) al nodo turtle_bot_cmdVel.

A continuación se presenta el enlace para el video de funcionamiento:

https://youtu.be/YTHalJ98Npo

Referencias

[1]