

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

II. PLANO MECÁNICO Y DISTRIBUCIÓN ELÉCTRÓNICA

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.**

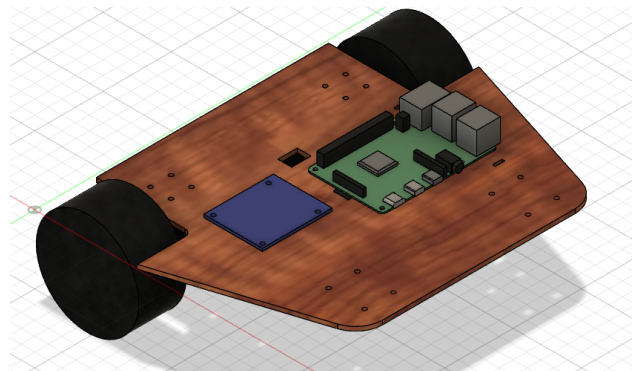


Figura 1: Diseño Fusion 360 - vista diagonal.

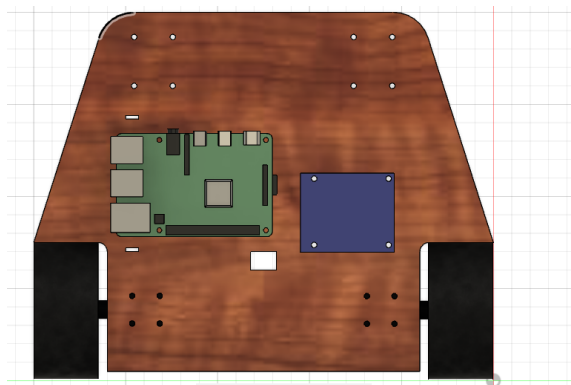


Figura 2: Diseño Fusion 360 - vista superior.

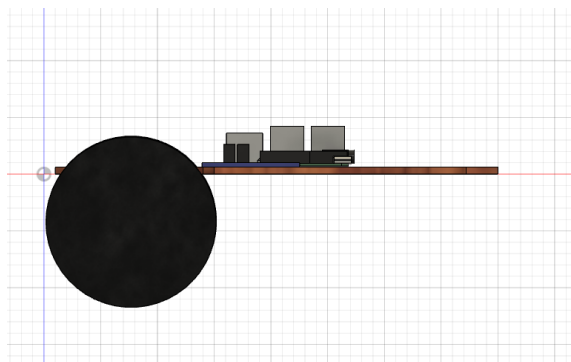


Figura 3: Diseño Fusion 360 - vista horizontal lateral.

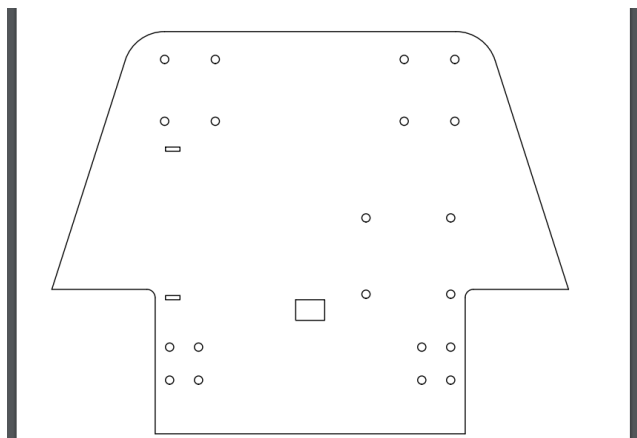


Figura 4: Plano para recorte en mdf de la base del robot.

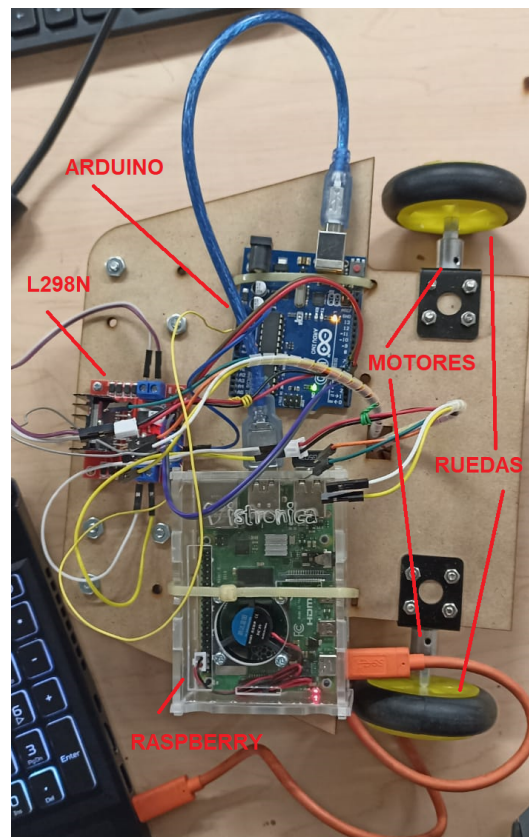


Figura 5: Plano de la distribución electrónica.

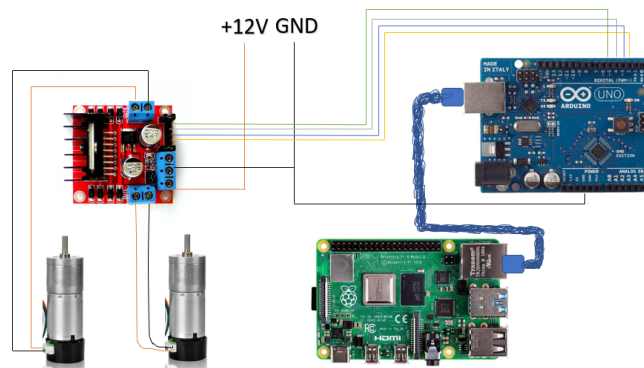


Figura 6: Conexiones entre módulos y motores.

III. 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.

IV. 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.

V. 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*.

Se escribió el nodo *flash_bot_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.

```
1 # define key press event callback
2 def on_press(key):
3     global estado_msg
4     global current_msg
5     try:
6         k = key.char # single-char keys
7     except:
8         k = key.name # other keys
9     if (k == "w"):
10         current_msg = "W"
11         estado_msg = "Arriba"
12     elif (k == "s"):
13         current_msg = "S"
14         estado_msg = "Abajo"
15     elif (k == "a"):
16         current_msg = "A"
```

```
17     estado_msg = "Izquierda"
18     elif (k == "d"):
19         current_msg = "D"
20         estado_msg = "Derecha"
21
22 # define key release event callback
23 def on_release(key):
24     global estado_msg
25     global current_msg
26
27     current_msg = "N"
28
29     estado_msg = "neutral"
```

Code Listing 1: Comando de movimiento presionando tecla.

Adicionalmente, se utilizaron las librerías *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. Una porción del código implementado se muestra en el Code Listing 2.

```
1 void messageCb( const std_msgs::String& toggle_msg){
2     current_msg = toggle_msg.data;
3     if (current_msg == "W"){
4         //Mover hacia adelante
5         digitalWrite(entrada1, HIGH);
6         digitalWrite(entrada2, LOW);
7         digitalWrite(entrada3, HIGH);
8         digitalWrite(entrada4, LOW);
9     }else if(current_msg == "D"){
10        //Mover hacia derecha
11        digitalWrite(entrada1, HIGH);
12        digitalWrite(entrada2, LOW);
13        digitalWrite(entrada3, LOW);
14        digitalWrite(entrada4, LOW);
15    }else if(current_msg == "S"){
16        //Mover hacia atras
17        digitalWrite(entrada1, LOW);
18        digitalWrite(entrada2, HIGH);
19        digitalWrite(entrada3, LOW);
20        digitalWrite(entrada4, HIGH);
21    }else if(current_msg == "A"){
22        //Mover hacia izquierda
23        digitalWrite(entrada1, LOW);
24        digitalWrite(entrada2, LOW);
25        digitalWrite(entrada3, HIGH);
26        digitalWrite(entrada4, LOW);
27    }else if(current_msg == "N"){
28        //REST
29        digitalWrite(entrada1, LOW);
30        digitalWrite(entrada2, LOW);
31        digitalWrite(entrada3, LOW);
32        digitalWrite(entrada4, LOW);
33    }
34 }
35
36 ros::Subscriber<std_msgs::String> sub("
37     Flash_Velocidad", &messageCb );
38 void setup()
```

Code Listing 2: Comando de movimiento presionando tecla.

En la Figura 7 se muestra el diagrama con la conexión entre los nodos *Flash_cmdVel* y *Serial_node*. El primero envía los comandos al topico *Flash_Velocidad* y el segundo corresponde

al subscribir el cual recibe los comandos y actúa en los motores 19 del robot diferencial.

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

Code Listing 3: Función que grafica la posición del Turtlebot.



Figura 7: Diagrama de conexiones entre nodos y tópicos.

VI. 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 al tópico *Flash_velocidad* en la que dependiendo de la acción realizada por el usuario, es decir presionar W, A, S o D, se usará la función de transformación para obtener las posiciones en X y Y. Posteriormente, se publican las posiciones al tópico *Flash_bot_position*. Estas coordenadas son posteriormente utilizadas por el nodo interfaz para mostrar gráficamente el recorrido del robot (en el marco Global).

En la Figura 8 se muestra el diagrama con la conexión entre el nodo *lectura_pos* y el tópico *Flash_velocidad*.



Figura 8: Diagrama de conexiones entre nodos y tópicos.

VII. INTERFAZ: GRÁFICA DE POSICIÓN

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

En el Code Listing 3 se muestra la función que se encarga de graficar los datos del movimiento del robot según el marco de referencia global. Además, añade también el título de «Trayectoria del Robot», el color, y el estilo, que en este caso son puntos rojos.

```
1 def animate(i):
2     global x_values
3     global y_values
4
5     x_values.append(x_current)
6     y_values.append(y_current)
7     plt.cla()
8     plt.scatter(x_values, y_values, label= "Turtlebot",
9                 color= "red", marker= "o", s=30)
10    plt.xlim(-2.5, 2.5)
11    plt.ylim(-2.5, 2.5)
12    plt.xlabel("Posicion en x")
13    plt.ylabel("Posicion en y")
14    plt.title("Trayectoria del Robot")
15    plt.legend()
16    plt.grid(color = 'black', linestyle = '--',
17            linewidth = 0.5)
18    plt.grid(True)
19
20    plt.minorticks_on()
```

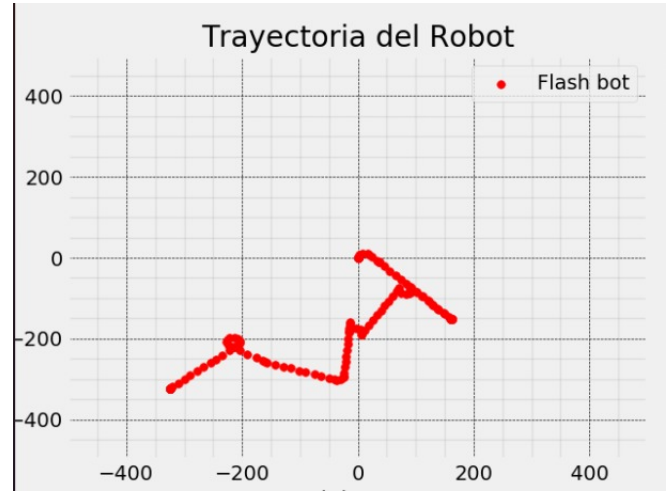


Figura 9: Trayectoria del movimiento del robot.



Figura 10: Diagrama de nodos y tópicos activos.

VIII. 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 continúa 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.

```
isaac@isaac-Lenovo-ideapad-330S-15ARR:~/catkin_ws/src/
taller_1_pkg/scripts$
isaac@isaac-Lenovo-ideapad-330S-15ARR:~/catkin_ws/src/
taller_1_pkg/scripts$ rosrn taller_1_pkg Interfaz.py
Desea guardar la ruta (S/n):s
```

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 4), crea un archivo .txt (Code Listing 5), y escribe los comandos en el archivo (Code Listing 6, Figura 14). Se le pregunta al usuario qué nombre tendrá este archivo .txt y se guarda en una ubicación (Figura 13).

```
1 def callback(vel_msg):
2     global x_current
3     global y_current
4
5     x_current = vel_msg.linear.x
```

```
6 y_current = vel_msg.linear.y
```

Code Listing 4: Recepción de los mensajes de comandos de movimiento.

```
1 if __name__ == '__main__':
2     decision = input("Desea guardar la ruta (S/n):")
3
4     if ((decision == "S")|(decision == "s")):
5         f = open("test.txt", "w+")
```

Code Listing 5: Creación del archivo .txt.

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

Code Listing 6: Escritura de los mensajes recibidos en el archivo .txt.

```
[INFO] [1645167942.157822]: Izquierda
30
30
30
[INFO] [1645167942.257974]: Izquierda
30
30
30
30
[INFO] [1645167942.357834]: Arriba
[INFO] [1645167942.457978]: Arriba
[INFO] [1645167942.557897]: Arriba
[INFO] [1645167942.657900]: Arriba
[INFO] [1645167942.757901]: Arriba
30
[INFO] [1645167942.857788]: Arriba
30
30
30
30
```

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

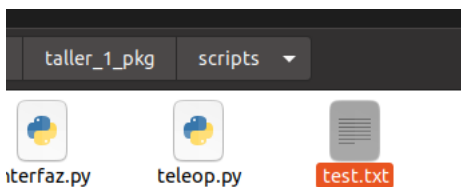


Figura 13: Archivo con los comandos, guardado en la carpeta donde se corre la interfaz.

```
25 data: "neutral"
26 data: "neutral"
27 data: "neutral"
28 data: "Arriba"
29 data: "Arriba"
30 data: "Arriba"
31 data: "Arriba"
32 data: "Arriba"
33 data: "Arriba"
34 data: "Arriba"
35 data: "Arriba"
36 data: "Arriba"
37 data: "Arriba"
38 data: "Arriba"
39 data: "Arriba"
40 data: "Arriba"
41 data: "Arriba"
42 data: "Arriba"
43 data: "Izquierda"
44 data: "Izquierda"
45 data: "Izquierda"
```

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

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 guardadas. La partida a reproducir se solicita a partir de un servicio. Para poder ofrecer el servicio, se creó el archivo reproducir_partida.srv, el cual se muestra a continuación.

```
1 string nombre_archivo
2 ---
3 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 explícito por lo cual se utiliza esta variable para indicar el momento en el que finaliza la ejecución de la secuencia de acciones.

El código 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 parámetro 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 línea 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 acción a realizar (`current_msg`) en la terminal, y se publica la variable tipo string al nodo *flash_Velocidad*.

REFERENCIAS

[1]