Taller 3

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.**
10. **3 Servomotores**
11. **Engranajes**
12. **Corte MDF para diseno de piston**

II. Plano mecánico y distribución eléctrónica

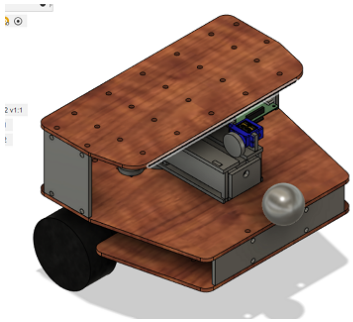
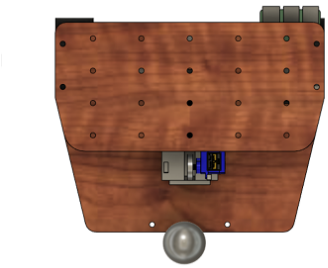


Figura 1: Diseño Fusion 360 - vista diagonal.



D

Figura 2: Diseño Fusion 360 - vista superior.

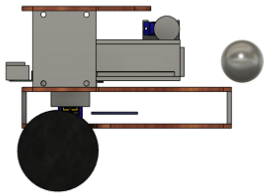


Figura 3: Diseño Fusion 360 - vista horizontal lateral.

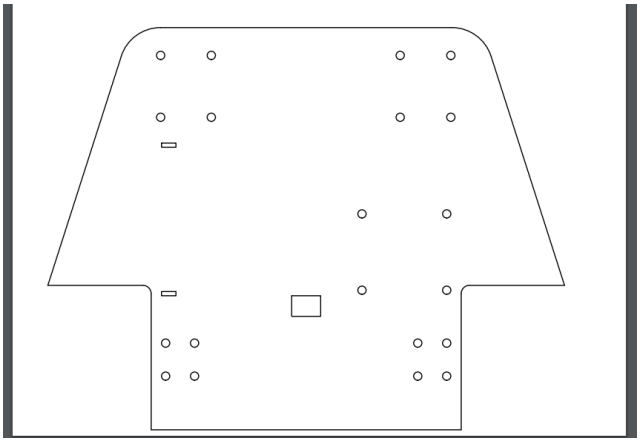


Figura 4: Plano para recorte en mdf de la base del robot.

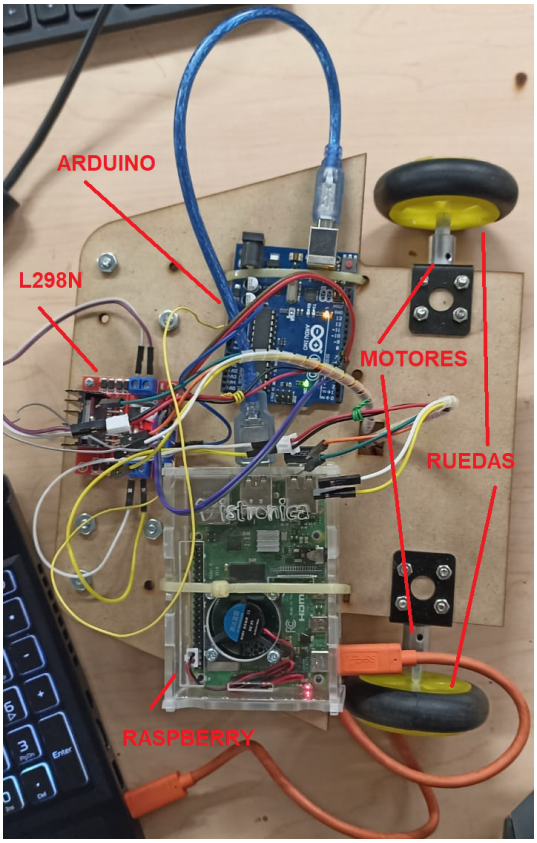


Figura 5: Plano de la distribución electrónica.

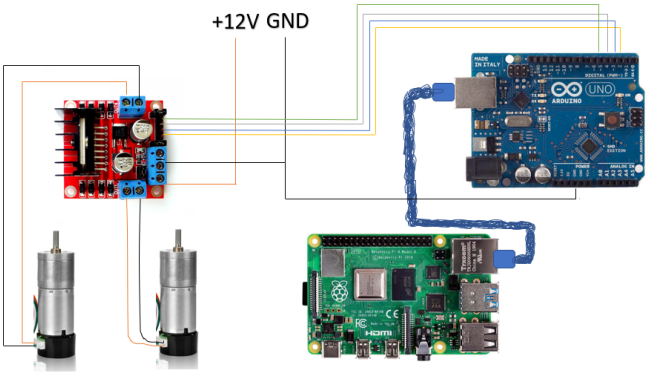


Figura 6: Conexiones entre módulos y motores.

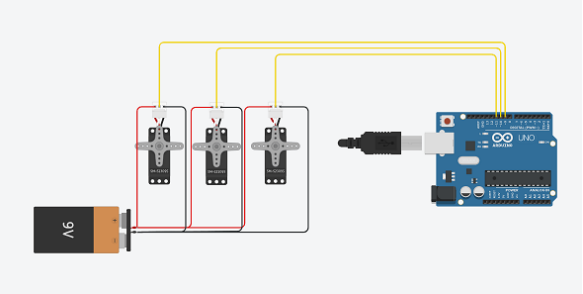


Figura 7: Conexiones entre módulos y servo-motores.

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. Se utilizaron en total 3 servos para controlar la pinza, uno de estos para rotar la base de la pinza, otro para alargar o acercar la pinza y el ultimo para abrir o cerrar esta.

IV. Funcionamiento General

El lenguaje sobre el que se esta trabajando es Python y C (Arduino), controlados bajo ROS. En este caso el diseno de la pinza se desarrolla en 4 etapas, movimiento en eje Z, eje X, cierre y finalmente imagen.

El primer movimiento se realiza a travez de la implementacion de dos engranajes, controlados a travez de un servo motor con un angulo de aproximadamente 40 grados a izquierda o derecha;

El segundo movimiento se encarga de aumentar el alcance de la pinza, teniendo asi un piston, el cual es manipulado a traves de un servomotor y un engranaje tipo cremallera.

El tercer movimiento es el encargado de cerrar la pinza y atrapar el objeto en cuestion, es de recalcar que este movimiento ya se tiene en cuenta para la grafica y por tanto no se vera reflejado un cambio en ella.

La ultima etapa, hace referencia a la imagen para el control autonomo de la pinza, esta recibe por usuario un color a elegir (Rojo, Verde o Azul), luego procede a encontrar la posicion en la imagen que puede captar y enviar los datos de la distancia a la que se encuentra de la pinza y la distancia a la que se encuentra del centro de la imagen.

Las etapas 1 a 3, tienen en conjunto su entidad grafica, con la cual es posible observar en el marco global la posicion de la pinza en el momento actual.

V.

VI. Movimiento: teleoperado

Para esta primera parte se creó un nodo en ROS llamado *robot_manipulator_teleop*, el cual permite al usuario controlar los 3 servos del brazo robotico. En consola se deben introducir las acciones que se quieran realiza, R y F para modificar el angulo, T y H para alargar o acortar, y Y y H para cerrar y abir la pinza respectivamente. Estas velocidades son publicadas como string al topico *Manipulador_Velocidades*. En caso de no oprimir ninguna tecla, el robot recibe la señal neutral y no ejecuta ninguná acción.

```python
# define key press event callback
def on_press(key):
    global estado_msg
    global current_msg
    try:
        k = key.char  # single-char keys
    except:
        k = key.name  # other keys
    if (k == "r"):
        current_msg = "R"
        estado_msg = "Sumar angulo"
    elif (k == "f"):
        current_msg = "F"
        estado_msg = "Restar angulo"
    elif (k == "t"):
        current_msg = "T"
        estado_msg = "Sumar distancia"
    elif (k == "g"):
        current_msg = "G"
        estado_msg = "Restar Distancia"
    elif (k == "y"):
        current_msg = "Y"
        estado_msg = "Abrir pinza"
    elif (k == "h"):
        current_msg = "H"
        estado_msg = "Cerrar pinza"

# define key release event callback
def on_release(key):
    global estado_msg
    global current_msg

    current_msg = "N"

    estado_msg = "neutral"
```

Code Listing 1: Comando de movimiento presionando tecla.

Al igual que para la entrega pasada, se utilizaron las librerias de rosde arduino, las cuales permiten comunicacion entre arduino y ROS. Se ejecuta un subrscruber de arduino, que recibe lo que se publica al topico *Manipulador_Velocidades* y realiza las acciones respectivas. Para facilitar el control de los sevos se utilizó la libreria Servo.h de arduino.

```

ros::NodeHandle  nh;

void messageCb( const std_msgs::String& toggle_msg){
  current_msg = toggle_msg.data;
  if (current_msg == "R"){
    //Sumar Angulo
    angulo += 1;
    ServoAngulo.write(angulo);              // tell
    servo to go to position in variable 'pos'
  }else if(current_msg == "F"){
    //Restar Angulo
    angulo -= 1;
    ServoAngulo.write(angulo);
  }else if(current_msg == "T"){
    //Sumar Distancia
    distancia += 1;
    ServoDistancia.write(distancia);
  }else if(current_msg == "G"){
    //Restar Distancia
    distancia -= 1;
    ServoDistancia.write(distancia);
  }else if(current_msg == "Y"){
    //Cerrar pinza
    pinza += 1;
    ServoPinza.write(pinza);
  }else if(current_msg == "H"){
    //Abrir pinza
    pinza -= 1;
    ServoPinza.write(pinza);
  }else if(current_msg == "N"){
    //Rest
  }
}

ros::Subscriber<std_msgs::String> sub("
    Flash_Velocidad", &messageCb );

void setup()
{
  ServoAngulo.attach(9);
  ServoDistancia.attach(10);
  ServoPinza.attach(11);
  nh.initNode();
  nh.subscribe(sub);
}

void loop()
```

Code Listing 2: Lectura de comandos desde arduino

VII. Obtención de posición en el marco global

En este caso, dado que ya se tenia diseñado un mapeo de la posicion del robot, el cual es un nodo que recibe coordenadas x,y y las plotea en una grafica que se actualiza a cierta frecuencia. Se modificó el nodo Pos.py, el cual recibe los comandos R, F, T, etc y de acuerdo a la velocidad del brazo, hace una actualizacion de las coordenadas x, y de este en el marco global. Este nodo publica las coordenadas tanto del brazo como del robot y el nodo interfaz.py se encarga de realizar y actualizar la grafica.

En la Figura 8 se muestra el diagrama con la conexión entre el nodo *lectura_pos* y el topico $manipulator_velocidad$.

Figura 8: Diagrama de conexiones entre nodos y topicos.

VIII. 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 5 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.

```
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 3: Función que grafica la posición del Turtlebot.

La conexión de los nodos se muestra a continuación.

Figura 9: Diagrama de nodos y tópicos activos.

La grafica obtenida se puede observar en el anexo, donde se encuentra un video de la operacion de los puntos 1 y 2 del taller.

IX. Movimiento automatico: Llegar a una posicion especifica

Para este numeral se realizó un nodo encargado de traducir las coordenadas x, y en un angulo y magnitud respecto a la posicion inicial del brazo (centrado, angulo 90 grados). Después de tener estos valores, se identificó para una velocidad especifica, cuanto era el cambio del angulo y la distancia por cada comando enviado al arduino. Se realizó entonces una conversion del angulo y la distancia al numero de comandos que debian ser enviados, en clicks, para llegar a cada una de estas posiciones.

El nodo creado se se subscribe a un topico que envia las coordenadas a alas que se quiere llegar y realiza los calculos necesarios para determinar cuantos comandos enviar al nodo de arduino.

```
def callback(vel_msg):
    global current_msg
    global position_msg
    global mag
    global ang

    #print("el mensaje es"+current_msg)
    x_current = float(str(vel_msg.data).split(":")
    [0])
    y_current = float(str(vel_msg.data).split(":")
    [1])


    mag = np.sqrt(x_current^2 + y_current^2)
    ang = np.arctan2(y/x)

    #Se define el numero de comandos a enviar
    distancia_clicks = mag/(0.02) #Constantes para
    realizar la conversion
    ang_clicks = ang/1.73 #Constante para realizar
    la conversion


    for i in distancia_clicks:
        pub.publish("T")
    for i in ang_clicks:
        if ang <= 0:
            pub.publish("F")
        else:
            pub.publish("R")
```

Code Listing 4: Codigo para enviar el brazo a una posicion especifica

Como se puede evidenciar, se transoforman las coordenadas y luego se envian señales al arduino de acuerdo a la magnitud y el angulo objetivo.

X. Archivo: control autonomo

El control autonomo de la pinza se encuentra soportado por el algoritmo del nodo disenado para la camara, el cual de acuerdo a un color especificado por el usuario enviara datos de distancia de la camara y lejania del centro de la camara. Esta informacion en forma de coordenadas sera enviada a traves de un topico, el cual una vez dentro del volumen de trabajo se procedera a generar movimiento de la siguiente manera. En primer lugar el servomotor que maneja el eje Z, se encargara de centrar el objetivo en el eje X de la camara, una vez realizado se entregaran datos de distancia al objetivo. En segundo lugar, obtenido

```

# Start a while loop
while(1):

    # Reading the video from the
    # webcam in image frames
    _, imageFrame = webcam.read()

    # Convert the imageFrame in
    # BGR(RGB color space) to
    # HSV(hue-saturation-value)
    # color space
    hsvFrame = cv2.cvtColor(imageFrame, cv2.
    COLOR_BGR2HSV)

    # Set range for red color and
    # define mask
    red_lower = np.array([136, 87, 111], np.uint8)
    red_upper = np.array([180, 255, 255], np.uint8)
```

```
    red_mask = cv2.inRange(hsvFrame, red_lower,
    red_upper)

    # Set range for green color and
    # define mask
    green_lower = np.array([25, 52, 72], np.uint8)
    green_upper = np.array([102, 255, 255], np.uint8
    )
    green_mask = cv2.inRange(hsvFrame, green_lower,
    green_upper)

    # Set range for blue color and
    # define mask
    blue_lower = np.array([94, 80, 2], np.uint8)
    blue_upper = np.array([120, 255, 255], np.uint8)
    blue_mask = cv2.inRange(hsvFrame, blue_lower,
    blue_upper)
```

Code Listing 5: Codigo para detectar colores y distancia

REFERENCIAS

[1]