Documentación de Proyecto

Renato Andaur - Camila Palma Caris - Felipe Valenzuela

El objetivo principal del grupo fue realizar un proyecto en el cual se pudiera controlar un brazo robótico mediante la detección de gestos, en donde, con el uso de red neuronal, intérprete estos gestos (mano, cara o cuerpo) y con ello, asociar un movimiento al robot. Además de aprender mecatrónica y programación.

Este proyecto consiste en un brazo robótico que es controlado de manera remota por el usuario mediante comandos que son dados con sus manos. Esto, dependiendo del comando, permite al brazo agarrar cosas y moverlas arriba, abajo, derecha, izquierda, hacia atrás o adelante y soltarlas.

Para este proyecto fueron necesarios los siguientes materiales:

- La estructura del brazo (impresa en impresora 3D)
- Una garra (impresa en impresora 3D)
- Cuatro servomotores
- Un arduino
- Una placa Raspberry Pi
- Cables
- Una WebCam
- Un Computador
- Pequeños objetos para que el brazo pueda agarrar.

Los comandos que recibe mediante la webcam son:

IZQUIERDA

DERECHA

ARRIBA

ABAJO

ADELANTE

ATRÁS

ABRIR LA GARRA

CERRAR LA GARRA

Algunas funciones tienen un cierto rango de funcionamiento:

1. Límite de movimiento en los ejes

El brazo robótico se mueve mediante tres servomotores que tienen cierto nivel de movilidad, puesto que estos giran de 0 a 180 grados.
El orden de los servos están dados por los tornillos de la siguiente imágen:

El primer servomotor es el de la base y que da movilidad en el plano x, tiene la libertad de moverse en un rango de 0° a 90°, empezando inicialmente en 45°. Si se le da un comando para moverse más allá de este, el servomotor no se moverá.

El segundo servomotor es el que va en lugar superior del robot y que da movilidad en el plano z principalmente (profundidad), tiene la libertad de moverse en un rango de 40° a 150°, empezando en 90°, para evitar que choque con la superficie en la que está puesto el robot. Si se le da un comando para moverse más allá de este, el servomotor no se moverá.

El tercer servomotor es el que va en la parte delantera del robot y aquel que da movilidad en el plano y, tiene la libertad de moverse en un rango de 60° a 120°, comenzando en 90°, para evitar que choque con la superficie en la que está puesto el robot. Si se le da un comando para moverse más allá de este, el servomotor no se moverá.

El cuarto servomotor es aquel responsable del movimiento de la garra.
La garra no puede abrirse y cerrarse eternamente, apróximadamente aguanta 6 veces un comando (de extremo a extremo, como de cerrada totalmente a abierta totalmente) antes que esta empiece a sobretrabajar (de todas formas este avisa cuando llega a su capacidad máxima, el servo empieza a sonar/”quejarse”). Cualquier exceso en el uso de estos comandos puede provocar que deje de funcionar y habría una pérdida valiosa de material.

2. Límite de velocidad de reacción propuesta

El brazo robótico recibe información a través de la raspi. Sin embargo, en un comienzo esta recibía muchos datos y a veces hacía que se moviera el robot en direcciones que no se le fueron dadas. Para solucionar este problema, se le implementó un tiempo de espera a la raspi, para demorar el tiempo en que envía y ejecuta, y también se le agregó al código, que se demora más tiempo en mandar la información que será procesada.

Si bien esto hace que el robot tenga un "delay" en su movimiento, se compensa al lograr hacer que este sea más preciso en sus direcciones para el usuario, solo hay que tener paciencia.

A continuación se tiene una explicación a grandes rasgos de cómo funciona el programa, esta es una explicación más técnica, para una explicación más estructural mirar el diagrama de flujo (DiagramaProyecto.pdf).

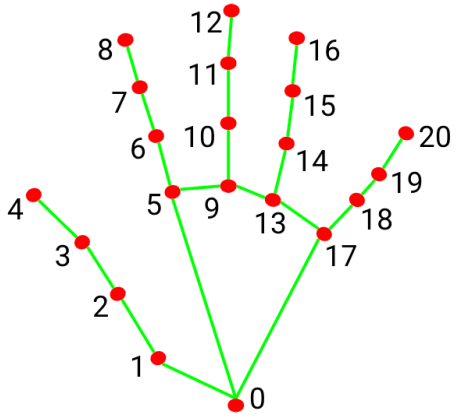
PERCEPCIÓN:

La detección de la mano está realizada con la librería "mediapipe", con esta librería se encuentran las posiciones de cada articulación de la mano.

DECISIÓN:

Con las posiciones de las articulaciones es fácil detectar el comando se hace con la mano, una explicación a continuación:

Fase 1: Primero podemos ver que dedos están levantados. las posiciones que tenemos son las siguientes:



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Gracias a esto podemos ver la distancia de la punta del dedo a la palma y compararla con la distancia del punto que está bajo la punta. Para explicarlo mejor se hará con un ejemplo. Si la distancia del punto 12 al punto 0, es mayor que la distancia del punto 11 al punto 0, significa que el dedo medio está levantado. En caso contrario, el dedo estará bajado. En el código se escribe así (HandTrackingModule.py):

```python
def finger_open_list(lmList):
    #retorna una lista con los dedos que estan levantados
    fingers = [] # 0 = close , 1 = open thumb to pinkie
    t = [12, 16, 20]
    if not lmList:
        return
    #thumb no funciona taaaan bien
    if distance_points(lmList, 4, 9) > distance_points(lmList, 3, 9):
        fingers.append(1)
    else:
        fingers.append(0)
    #index
    if distance_points(lmList, 8, 1) > distance_points(lmList, 7, 1):
        fingers.append(1)
    else:
        fingers.append(0)
    #mid, anular, pinky
    for tip in t:
        if distance_points(lmList, tip, 0) > distance_points(lmList, tip-1, 0):
            fingers.append(1)
        else:
            fingers.append(0)
    return fingers
```

Podemos ver que usamos distintos puntos en la palma para cada caso, esto con el objetivo de un mejor reconocimiento.

La distancia se calcula de la siguiente manera:

```python
def distance_points(lmList, id1 , id2):
    #encuentra la distancia de dos puntos de la mano
    pointx = (lmList[id1][1], lmList[id1][2])
    point1 = (lmList[id2][1], lmList[id2][2])
    return math.dist(point1, pointx)
```

Fase 2:

Ahora que tenemos los dedos que están levantados podemos escoger una dirección dependiendo de cuales están levantados.

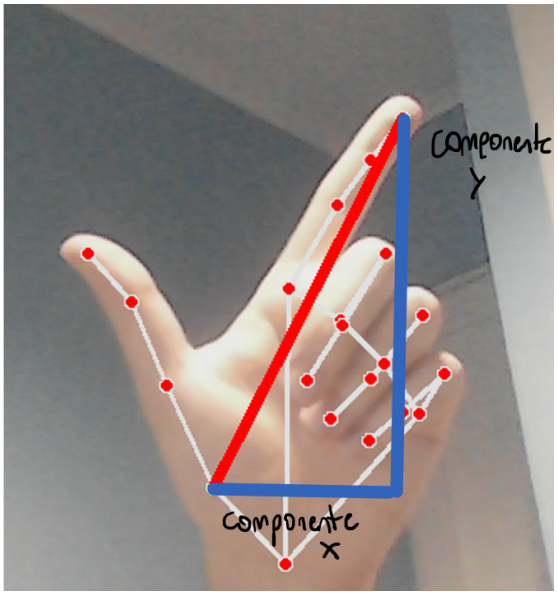
Por ejemplo si está levantado el pulgar y meñique podemos decir que la dirección es adelante. Código(apuntar.py):

```
#si se levante le pulgar y meñique la direccion es hacia adelante
elif htm.finger_open_list(lmList) == [1, 0, 0, 0, 1]:
    cur_dir = "front"
# si se levante el pulgar, indice y meñique la direccion es hacia atras
elif htm.finger_open_list(lmList) == [1, 1, 0, 0, 1]:
    cur_dir = "back"
# si se levante el pulgar y dedo medio la direccion es salir
elif htm.finger_open_list(lmList) == [1, 0, 1, 0, 0]:
    cur_dir = "leave"
# si se levante el pulgar, indice y dedo medio la direccion es abrir
elif htm.finger_open_list(lmList) == [1,1,1,0,0]:
    cur_dir = "abrir"
# si se levante el pulgar, indice, dedo medio y anular la direccion es cerrar
elif htm.finger_open_list(lmList) == [1,1,1,1,0]:
    cur_dir = "cerrar"
```

Para las direcciones de arriba, abajo, derecha e izquierda se realiza algo ligeramente distinto.

El objetivo es que dependiendo de donde se está apuntando decidimos la dirección.
Para esto primero hay que ver que tanto el dedo pulgar como índice están levantados([1,1,0,0,0]).

Luego se decide si se apunta hacia una dirección vertical o horizontal. Esto se logra haciendo un vector entre la punta del índice y base del pulgar y viendo si la componente en x es mayor o menor a la componente en y.



Con esto es fácil ver a qué dirección se apunta. (en código: apuntar.py)

```python
#si se hace una pistola con la mano, se decide adonde está apuntando, dependiendo de donde apunta, la direccion que se escoge.
if htm.finger_open_list(lmList) == [1, 1, 0, 0, 0]:
    index_tip = (lmList[8][1],lmList[8][2])
    thumb_base = (lmList[1][1], lmList[1][2])
    cv2.line(img,index_tip, thumb_base, (0,0,255), 5)
    if abs(index_tip[0] - thumb_base[0]) > abs(index_tip[1] - thumb_base[1]):
        if index_tip[0] - thumb_base[0] > 0:
            cur_dir ="right"
        else:
            cur_dir = "left"
    else:
        if index_tip[1] - thumb_base[1] > 0:
            cur_dir ="down"
        else:
            cur_dir = "up"
```

Fase 3:
Ahora que tenemos la dirección se realizó un arreglo por si existe algún error al detectarla para que solo cambie la dirección si es la misma por dos frames seguidos. (Código: apuntar.py)

```python
## Arreglar errores de precision (la direccion tiene que ser igual algunos frames para cambiar.)
if counter != 1:
    if prev_dir == cur_dir:
        counter += 1
    else:
        counter = 0
else:
    if prev_dir != cur_dir:
        counter = 0
    else:
        self.direccion = cur_dir

prev_dir = cur_dir
```

Ejecución:
Ahora con la dirección encontrada se puede mandar a la raspi cada vez que cambia. (Código: pruebasRobot.py)

```python
def loop_mensaje():
    #loop para mandar la dirección a la raspi
    #Se manda el mensaje solo si es distinto al anterior
    pDir = "stop"
    while True:
        cDir = dedos.direccion
        if cDir == pDir:
            if cDir == "leave":
                #Se detiene si recibe el comando "leave"
                break
            pDir = cDir
            time.sleep(.1)
            continue
        else:
            mensaje.mandar_direccion(cDir)
            if cDir == "leave":
                print("Ya no se mueve")
        pDir = cDir
        time.sleep(.1)

    #Se termina la conexión al raspi
    mensaje.close()
```

Para hacer la conexión a la raspi se usa la librería paramiko.(Código: send_angles.py)

La raspi recibe la información a través de un input (Código:wait_comand.py)

```python
    def loop_pedir(self):
        #espera a que le llegue una dirección
        while True:
            self.direccion = input("Ingresa direccion: ")
            if self.direccion == "leave":
                self.status = False
                break
```

Finalmente la raspi escoge un servo a partir de la dirección:

```python
def loop_mover():
    #Loop que mueve al robot en la direccion actual del receptor
    while receptor.status:
        #Se escoge el servo que hay que mover(Se puede hacer más bonito)
        if receptor.direccion == "left" or receptor.direccion == "right":
            robot.mover_horizontal(receptor.direccion)
        elif receptor.direccion == "up" or receptor.direccion == "down":
            robot.mover_vertical(receptor.direccion)
        elif receptor.direccion == "front" or receptor.direccion == "back":
            robot.mover_profundidad(receptor.direccion)
        elif receptor.direccion == "abrir" or receptor.direccion == "cerrar":
            robot.mover_garra(receptor.direccion)
        time.sleep(1)
```

Finalmente se mueve. Por ejemplo:

```python
def mover_horizontal(self, dir):
    #Servo 1
    if dir == "left":
        self.pose = self.pose - self.speed
    elif dir == "right":
        self.pose += self.speed

    #Limites del servo
    if self.pose > 90:
        self.pose = 90
    elif self.pose < 0:
        self.pose = 0

    #Se le dice al arduino la nueva posicion del servo
    self.robot_serial.write_servo(1, self.pose)
```

Librerías que ocupamos:
Mediapipe: Librería para la detección de manos, hecha por google. Utiliza machine learning.
open-cv: Principalmente usada para la captura de imágenes a través de la cámara.
paramiko: Librería para mandar comandos por python a una consola a través de ssh.