



PROYECTO VISIÓN POR COMPUTADOR -
**MOVIMIENTO DE UN
BRAZO ROBÓTICO
IMITANDO EL MOVIMIENTO
DE NUESTRO BRAZO
A TRAVÉS DE UNA WEBCAM**



Juan PÉREZ FERNÁNDEZ
Esther REGO FALAGÁN
Laura TOMÁS MOCHÓN
ANDRÉS VILCHES GARCÍA

ÍNDICE

- 1. INTRODUCCIÓN Y OBJETIVOS**
- 2. ESTADO DEL ARTE**
- 3. METODOLOGÍA**
 - 3.1. Software**
 - 3.2. Hardware**
 - 3.3. Elaboración del brazo robótico**
 - 3.4. Elaboración del soporte**
 - 3.5. Programación**
- 4. EXPERIMENTACIÓN**
 - 4.1. Preparación del entorno**
 - 4.2. Servomotores**
- 5. CONCLUSIONES**
- 6. DEMOSTRACIÓN**
- 7. BIBLIOGRAFÍA**

1. INTRODUCCIÓN Y OBJETIVOS

La visión por computadora ha revolucionado el campo de la robótica, especialmente en aplicaciones de tiempo real, proporcionando a los robots la capacidad de percibir y entender su entorno de manera similar a los humanos. Esta sinergia entre la visión por computador y la robótica ha desbloqueado un sinfín de posibilidades, permitiendo que los robots realicen tareas con mayor autonomía, precisión y adaptabilidad.

Aprovechando esta revolución, se ideó hacer una simulación del robot Atom, de la película de “Acero puro” el cual simula los movimientos de la persona que tenía delante.



Imagen 1. Película 'Acero puro'

Para hacer este movimiento posible, se ha usado una raspberry pi 4 model B para poder utilizar mediapipe. Mediapipe es un proyecto de software de código abierto para el procesamiento de medios, desarrollado por Google. Para la detección se usa una webcam por usb.

Para hacerlo más visual se ha impreso en 3D un brazo de 2 grados de libertad y se ha implementado su movimiento mediante servomotores.

OBJETIVOS

- conseguir que a través de la webcam detecte el brazo
- transformar el movimiento del brazo en ángulos legibles para el servo
- obtener un movimiento similar en nuestro brazo robótico

2. ESTADO DEL ARTE

Como hemos mencionado anteriormente, nos hemos inspirado en la película Acero puro, donde se ve como el robot imita los movimientos del humano. Para la elaboración del proyecto hemos utilizado en nuestra programación Mediapipe, para conseguir el movimiento en tiempo real.

MediaPipe se lanzó inicialmente como una solución para facilitar el desarrollo de aplicaciones de visión por computador. El objetivo era proporcionar a los desarrolladores una forma eficiente y fácil de crear aplicaciones que pudieran procesar imágenes y video en tiempo real, utilizando técnicas avanzadas de aprendizaje automático.

Las primeras versiones de MediaPipe ofrecían herramientas para el reconocimiento facial, la segmentación de objetos y el seguimiento de gestos. Estas capacidades permitían a los desarrolladores crear experiencias interactivas, como filtros de realidad aumentada para cámaras de teléfonos inteligentes y sistemas de seguimiento de movimientos.

A medida que el aprendizaje automático se volvía más avanzado, MediaPipe integró modelos de inteligencia artificial más potentes. Esto permitió características más sofisticadas como el seguimiento de poses en 3D, la estimación de la profundidad y la detección de gestos con mayor precisión.

Hoy en día, MediaPipe es utilizado en una variedad de aplicaciones comerciales y de investigación. Desde la creación de efectos visuales en redes sociales hasta aplicaciones en robótica y asistencia sanitaria, su impacto es notable. La capacidad de procesar y entender imágenes y video en tiempo real ha abierto puertas a innovaciones en múltiples sectores.

3. METODOLOGÍA

3.1. Hardware

Para poder hacer este sistema en tiempo real se ha utilizado el microcontrolador Raspberry pi Model B:



Imagen 2. Raspberry pi Model B

Por otro lado, para la detección se ha utilizado una webcam que funciona por USB:



Imagen 3. Webcam

Por último, para el movimiento del brazo se ha usado 2 servomotores diymore DM996 que tienen 13 kg/N de fuerza.



Imagen 4. Servomotores

3.2. Software

Para la detección de los puntos de referencia necesarios para calcular el señal de entrada a cada uno de los motores se ha empleado la biblioteca MediaPipe. Se han seleccionado los siguientes landmarks: Left hip, left elbow, left shoulder, left wrist. A partir de ellos, usando sus coordenadas (x,y) se pasan como argumentos a la función de **calcular_angulo**. En uso de la biblioteca de OpenCV para mostrar una pestaña con la imagen en tiempo real de la cámara. En ella, haciendo uso del método **draw_landmarks** dibujamos en la imagen de salida los puntos de referencia que detecta MediaPipe así como las conexiones entre ellos. MediaPipe segmenta muy bien la imagen por lo que no hemos tenido que preparar un entorno en concreto más allá de que esté bien iluminado.

recolorImage recibe un fotograma de la cámara y un objeto de "pose" de MediaPipe, convierte el fotograma a RGB, detecta la pose y devuelve el resultado con la imagen.

readingCamera inicializa el objeto de la cámara y configura los parámetros relacionados con la detección de pose, y muestra el fotograma procesado con puntos de referencia.

Finalmente, en la función principal se inicializa el modelo de postura de MediaPipe y se llama a la función del código principal **readingCamera**.

Además, se han importado ciertas funciones de la biblioteca de **GPIOzero**, que es una biblioteca destinada al manejo de microcontroladores, como en este caso es nuestra Raspberry Pi. Esta nos servirá para poder controlar los servomotores de manera más sencilla.

En resumen, OpenCV se usa para obtener imágenes de la cámara para visualización y MediaPipe se ha empleado para detectar poses.

3.3. Elaboración del brazo

Para la construcción del brazo se ha utilizado un modelo 3D de internet, del cual se han utilizado ciertas piezas que eran de interés para el proyecto:

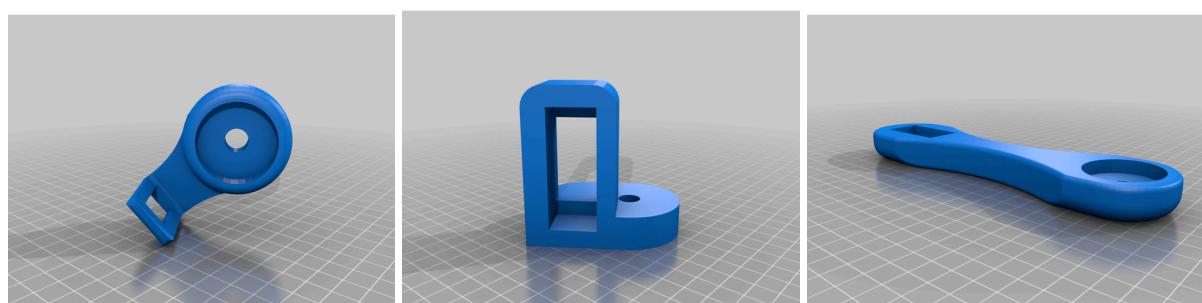


Imagen 5. Piezas modelo 3D

Para poder imprimir estas piezas se ha usado la impresora 3D anet A6 y plástico PLA:



Imagen 6. Impresora 3D y plástico PLA

Una vez impresas las piezas del brazo se han lijado y se han pintado. Posteriormente se ha hecho el montaje uniendo las piezas y los servomotores mediante tornillos.



Imagen 7. Brazo montado

3.4. Elaboración del soporte

Para la realización del soporte del brazo robótico se ha empezado cogiendo un trozo de madera para poder cortarlo y que, al ser madera y tener peso, el brazo robótico se pueda sujetar adecuadamente.

A continuación, se realizó un boceto para saber como hacer la forma del soporte. En este caso se eligió hacer una persona humana para añadirle el brazo y, para ello, se han cortado tres piezas distintas de madera.

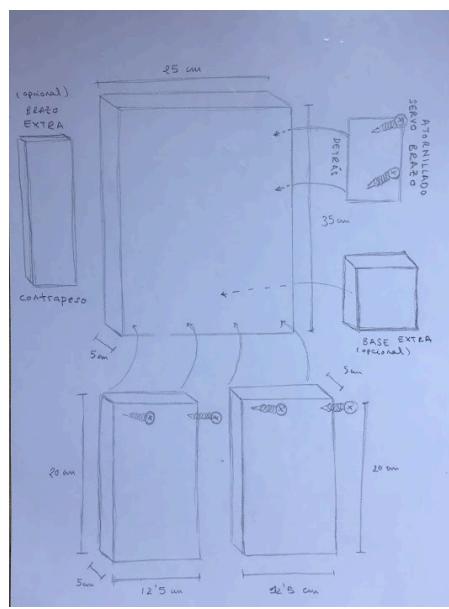


Imagen 8. Boceto soporte

La primera pieza sería un rectángulo de madera que equivale al cuerpo y las otras dos piezas que han sido cortadas se usarán para simular las piernas de la persona.



Imagen 9. Soporte montado

Estas dos piezas, además de ser un adorno, se han utilizado para que se haga contrapeso y que el soporte tenga más estabilidad a la hora de que el brazo se mueva. Para que las piezas estén juntas, se han atornillado y de esta manera poder tener una base más equilibrada.

A continuación, se ha decorado para que parezca una persona, utilizando distintos colores de goma eva y, para la cabeza, se ha utilizado una bola de poliespan decorada con rotulador. Una vez terminado todo este proceso, se ha atornillado el brazo robótico al cuerpo del monigote.



Imagen 10. Soporte decorado y finalizado

3.5. Programación

```
# LOCAL FUNCTIONS:
def calculateAngles(a,b,c,d):
    a=np.array(a) # left shoulder
    b=np.array(b) # left elbow
    c=np.array(c) # left wrist
    d=np.array(d) # left hip
    # Calculate the input for each servo
    # angulo2 -> servo1 (first joint; references: hip, shoulder, elbow)
    # angulo1 -> servo2 (second joint; references: shoulder, elbow, wrist)
    rads1=np.arctan2(c[1]-b[1],c[0]-b[0])-np.arctan2(a[1]-b[1],a[0]-b[0])
    angle1=np.abs(rads1*180.0/np.pi)
    rads2=np.arctan2(b[1]-a[1],b[0]-a[0])-np.arctan2(d[1]-a[1],d[0]-a[0])
    angle2=np.abs(rads2*180.0/np.pi)
    # We must make sure our values are ranged from 0 to 180 degrees
    if angle1>180.0:
        angle1=360-angle1

    if (angle2>180.0):
        angle2=360-angle2

    return angle2, angle1

def recolorImage(frame,pose):
    # Each of the frames of the video must be transformed to RGB by recoloring
    img = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    img.flags.writeable = False
    # making a detection -> method process to store our detection in results var
    results = pose.process(img)
    # recolor back to BGR
    img.flags.writeable = True
    img = cv.cvtColor(img, cv.COLOR_RGB2BGR)

    return results, img
```

```
def recolorImage(frame,pose):
    # Each of the frames of the video must be transformed to RGB by recoloring
    img = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    img.flags.writeable = False
    # making a detection -> method process to store our detection in results var
    results = pose.process(img)
    # recolor back to BGR
    img.flags.writeable = True
    img = cv.cvtColor(img, cv.COLOR_RGB2BGR)

    return results, img
```

```

def readingCamera(cam,pose):
    ----- Camera activation / desactivation -----
    angulo_anterior = 0
    while cam.isOpened():
        ret, frame = cam.read()
        results_recolor, image = recolorImage(frame,pose)
        # render the detections:
        #     .pose_landmarks gives us the following info: x,y,z,visibility
        #     this represents every individual points of our estimation model
        #     .pose_connections gives us the different connections of different parts of our body (wrist, elbow and shoulder most importantly)
        # We use the method draw_landmarks to draw the points detected of our body in the video feed
        mp_draw.draw_landmarks(image,results_recolor.pose_landmarks, mp_pose.POSE_CONNECTIONS)
        # we store our landmarks to get the angles related to each of the body parts related to our robot arm joints
        landmarks = results_recolor.pose_landmarks.landmark
        # Each location is a vector of points [x,y] in a 2D system so we can proceed to get the angles
        shoulder=[landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
        elbow=[landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
        wrist=[landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x, landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]
        hip=[landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x,landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].y]
        # We get the angles we'll use as input for the servos
        angle1, angle2 = calculateAngles(shoulder,elbow,wrist,hip)
        # We must convert according to our specifications
        if(angle1 <= 180): # servo1 range it's (-90, 90)
            servo1.angle = (angle1 - 90)
        else:
            servo1.angle = (180 - 90)
        if(angle2 <= 140 ): # servo2 range it's (-90,90)
            servo2.angle = ((angle2)- 90)
        else:
            servo2.angle = ((140) - 90)

        # show the image
        cv.imshow('Mediapipe Feed', image)
        k = cv.waitKey(1)
        if k != -1:
            break
    cam.release()
    cv.destroyAllWindows()
    print ('Camera quitted')

```

```

#-----
#----MAIN-----
#-----

# variable to visualize our poses
mp_draw = mp.solutions.drawing_utils
# imports our pose estimation model
mp_pose = mp.solutions.pose
# get the camera feed
cam = cv.VideoCapture(0)
mdc = 0.5 # parameter of minimal detection confidence(accuracy)
mtc = 0.5 # parameter of tracking accuracy

with mp_pose.Pose(min_detection_confidence=mdc, min_tracking_confidence=mtc) as pose:
    # Camera activation
    readingCamera(cam,pose)

```

4. EXPERIMENTACIÓN

4.1. Preparación del entorno

En primer lugar se tiene que introducir el sistema operativo en la Raspberry pi, en el caso de este proyecto se ha utilizado Raspberry Pi OS x64 , que es un sistema linux que está en desarrollo. Una vez instalado en una micro SD se introduce en la raspberry y se configura el sistema.

Cuando se configure, se comprobará la versión de python instalada, que en el caso de este microcontrolador es la 3.11. Por consiguiente se instalará opencv para poder utilizar Mediapipe, es una buena práctica crear entornos virtuales en python para no tener conflictos entre librerías, para ello se utiliza virtualenv que se instala con la instrucción: **sudo apt install python3-pip python3-venv**.

Una vez instalado, para crear un entorno virtual se usa la instrucción: **python3 -m --system-site-packages venv [nombre entorno virtual]**. “**--system-site-packages**” se utiliza para que el entorno pueda acceder al resto de librerías del sistema, ya que como se están usando servomotores es necesario acceder a *gpiozero*. Cuando se quiera entrar a este entorno virtual se usará: **source [nombre entorno virtual]/bin/activate**. Entonces ya se podrá instalar opencv sin preocuparse de que genere conflicto.

4.2. Servomotores

Los servomotores se tienen que conectar directamente a la Raspberry, sin embargo se les tiene que proporcionar una tensión externa, ya que con la alimentación de la Raspberry no tienen suficiente para funcionar. Por lo tanto el esquema electrónico quedaría de la siguiente manera:

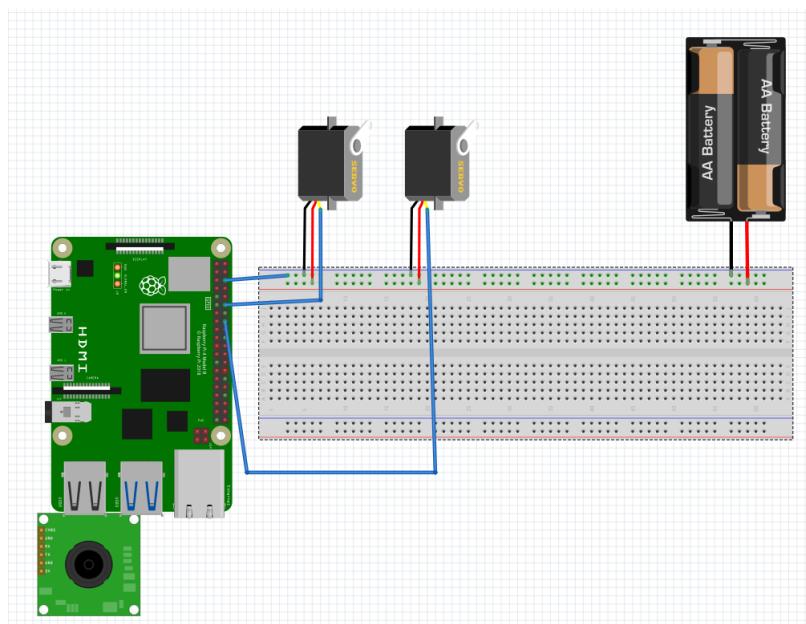


Imagen 11. Esquema electrónico

Sin embargo a la hora de programar los servos en la raspberry, se tiene que ajustar el PWM de estos. Para ello hay que obtener esa información del fabricante. El problema de estos servomotores es que son una imitación de los TowerPro MG996R, por lo tanto hay que acceder a la información que nos proporciona el fabricante. De esta se obtiene que el pwm tiene un DutyCycle de 500 a 2500 μ s. Además indica que dependiendo de la tensión aplicada estos funcionarán a 9Kg/N con 5 V y aplicando una tensión de 6V a 13 Kg/N. En el caso de este proyecto con aplicarle una tensión de 5 V será suficiente.

Una vez ajustado eso se observará que a la hora de asignarle una posición al servomotor este temblará. Como el servomotor no es completamente original, el pulso que le manda la raspberry no estará del todo ajustado porque el DutyCycle de estos servos varía un poco. Para solucionar esto se puede usar la librería pigpio factory, esta librería convierte los pulsos de la raspberry mandados por ella en pulsos mandados por el hardware.

Una vez configurados los servos ya se pueden implementar con el resto de la programación.

5. CONCLUSIONES

Este proyecto demuestra el avance de la visión por computador asociada a la robótica, especialmente en aplicaciones de tiempo real. La integración del software de MediaPipe y OpenCV en un hardware accesible como la Raspberry Pi sumado a la acción de control sobre un brazo de 2 grados de libertad impreso en 3D, reflejan un salto significativo en la interacción hombre-máquina.

Como conclusión, se puede observar que gracias a los conocimientos adquiridos en esta asignatura se ha conseguido realizar este proyecto de una manera satisfactoria y se demuestra que la visión por computador es una parte vital de la robótica actual.

6. DEMOSTRACIÓN

Vídeo demostración del funcionamiento del brazo robótico:

<https://drive.google.com/file/d/1Kf32MQrvJNQ6avcA0I3AukaAEM1dKzql/view?usp=sharing>



Imagen 12. Brazo robótico terminado

7. BIBLIOGRAFÍA

Para la información sobre la raspberry es mejor buscar directamente en su web debido a que está constantemente actualizándose y mucha de la información está obsoleta.

Referencia: "Documentación de Raspberry Pi". Disponible en:
<https://www.raspberrypi.com/documentation/computers/>

Referencia: "Pigpio - Raspberry Pi GPIO Interface". Disponible en:
<https://abyz.me.uk/rpi/pigpio/>

Referencia: "Documentación de Python - Biblioteca 'venv'". Disponible en:
<https://docs.python.org/es/3/library/venv.html>

Referencia: "Hoja de Datos de MG996R". Disponible en:
<https://www.alldatasheet.es/datasheet-pdf/pdf/1131873/ETC2/MG996R.html>

Referencia: "Raspberry Pi Guide - Uso de Cámaras Web USB". Disponible en:
<https://raspberrypi-guide.github.io/electronics/using-usb-webcams>

Referencia: "Pose landmark detection guide". Disponible en:
https://developers.google.com/mediapipe/solutions/vision/pose_landmarker

Referencia: "MediaPipe Holistic". Disponible en:
<https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md>

Referencia: "Sistema de representación del movimiento de una persona en arreglos vectoriales". Disponible en:
https://repository.javeriana.edu.co/bitstream/handle/10554/65099/attachment_0_Sistema-de-representaci%C3%B3n-del-movimiento-de-una-persona-en-arreglos-vectoriales.pdf?sequence=1&isAllowed=y