Gesture Controlled Robot

D. Antonio-Torres, Member IEEE,

B. Gutiérrez-Padilla, S.A. Saldaña-Millán, L.P. Carrasco-Molina, and M.A. Villareal-González

***Abstract*— Para este proyecto desarrollamos un sistema embebido el cual busca reducir el riesgo laboral del contacto del ser humano con químicos peligrosos, mediante el uso a distancia de un robot capaz de manipular dichos químicos y el cual es controlado por un sistema de visión por computadora, mediante ciertos gestos específicos y que además cuenta con un sistema de autenticación por reconocimiento de rostro como medida de seguridad.**

Index Terms— LocoBot, Visión por computadora, Sistemas Embebidos, Python, Brazo robot, Visión para robots, Peligro laboral.

I. INTRODUCCIÓN

A. Objetivo general

Desarrollar un sistema de brazo robótico controlado a distancia mediante un programa de visión por computadora, el cual utiliza gestos de mano y reconocimiento facial, para la manipulación de compuestos químicos delicados y/o peligrosos de manera remota.

B. ¿Por qué se realizará este proyecto?

La producción de productos químicos va en paralelo con el crecimiento y el desarrollo industrial en los países. Los productos químicos poseen un riesgo potencial para la salud, por fugas o derrames, ya sea por una situación accidental en las plantas de procesamiento o por errores humanos en el manejo de los mecanismos de seguridad en los complejos procesos industriales.

Desde la perspectiva de la salud pública, los efectos de las sustancias químicas pueden ser agudos o crónicos, según tipo de sustancia, propiedades fisicoquímicas, propiedades toxicológicas, cantidad liberada, tiempo de exposición y vías de exposición. (Accidentes Químicos, 2022)

Una sustancia tóxica es cualquier sustancia capaz de producir un efecto nocivo en los seres vivos, para que esto ocurra es necesario que el organismo se ponga en contacto con la sustancia tóxica, lo cual se puede presentar a través de la inhalación (lo que respiramos), la ingestión (lo que comemos o bebemos) o vía cutánea (contacto con la piel y ojos). (María Esther Arcos Serrano, 2007)

Javier Benaya, presidente de Científicos por el medio ambiente comenta lo siguiente "El riesgo químico (problemas para la salud por sustancias que se usan en la industria) pasa desapercibido por la población y por los políticos". (La exposición a sustancias químicas causa 4.000 muertes al año entre trabajadores, 2010)

Este peligro es tan oculto que cuesta que las ONG tengan datos concretos. Llorenç Serrano, de la Confederación Sindical de Comisiones Obreras, afirma que la exposición en el trabajo causa 4.000 muertes al año, que 33.000 trabajadores enferman cada año y que hay 18.000 accidentes laborales relacionados. (La exposición a sustancias químicas causa 4.000 muertes al año entre trabajadores, 2010)

Al ver estos datos, quedamos impactados. ¿Cómo es posible que después de tantos años de estudios y avance tecnológico aún no podamos evitar que sustancias químicas tóxicas como las anteriores perjudiquen la calidad de vida de los obreros trabajadores de las industrias? Con esta pregunta en mente encontramos una solución que, si se continúa, tiene un gran potencial de poder resolver este tipo de problemas que azotan a los obreros y empresas de manera global.

C. Solución a los problemas

La solución planteada es la siguiente: desarrollar un brazo robótico controlado de manera remota haciendo uso de visión por computadora para lograr que el brazo robótico pueda manipular los diferentes compuestos químicos utilizados en la industria y, de esta forma, reducir el impacto que estas sustancias tóxicas tienen en los humanos.

Imaginemos el siguiente escenario. Supongamos que tenemos que realizar una combinación de compuestos químicos que liberan una nube tóxica, con el uso de este robot el operador estaría en un cuarto aislado de los compuestos químicos realizando la manipulación de los mismos y, al mismo tiempo, ventilando el área donde se realizó la combinación de los compuestos. Esto creará un ambiente seguro donde el operador podrá mover el brazo robótico mediante gestos reconocidos por el programa de visión por computadora para que el movimiento del brazo simule un brazo humano.

En este reporte nuestro trabajo será guiarlos por los pasos necesarios para comprender y entender de una mejor manera cómo lograremos esto y además poder proponer mejoras a posteriori, para hacer de este robot un obrero más de las empresas del mundo.

D. Implementación General

Primero que nada, necesitamos conocer qué robot utilizaremos durante este proyecto. En el apartado "Locobot" podremos comprender más acerca del funcionamiento de este brazo robótico que, no solo incorpora un brazo robótico como comentado anteriormente, sino que también incorpora sensores LIDAR, Cámara 3D y ruedas que permitirán al robot en

un futuro poder incluso desplazarse por el cuarto donde sea implementado.

Sabemos que hoy en día la seguridad lo es todo cuando hablamos de nuestra información personal, e inclusive de los secretos de las empresas en que este robot pudiera laborar como obrero, es por esto que también se implementó un sistema de seguridad basado en reconocimiento facial (Ver apartado: "Reconocimiento Facial") donde solo aquellas personas que estén registradas en la base de datos del robot puedan ingresar para poder empezar a manipular el robot.

Una vez nos hayamos identificado ante el robot, procederemos a controlarlo. Para esto, se utilizó un programa de reconocimiento de gestos (Ver apartado: "Reconocimiento de Gestos"). Primero tendremos que entrenar el algoritmo para que reconozca a las personas responsables de mover el robot, ingresamos como mínimo una imagen del rostro de la persona para que el sistema nos reconozca y permita el acceso al reconocimiento de gestos para manipular el robot. Para el sistema de reconocimiento de gestos se utilizó la librería de TensorFlow, esta librería nos permitirá utilizar diferentes gestos preprogramados como lo son:

Thumbs up, Thumbs Down, Fist, Live Long, Peace, Rock, Call me, Smile, Okay, Stop

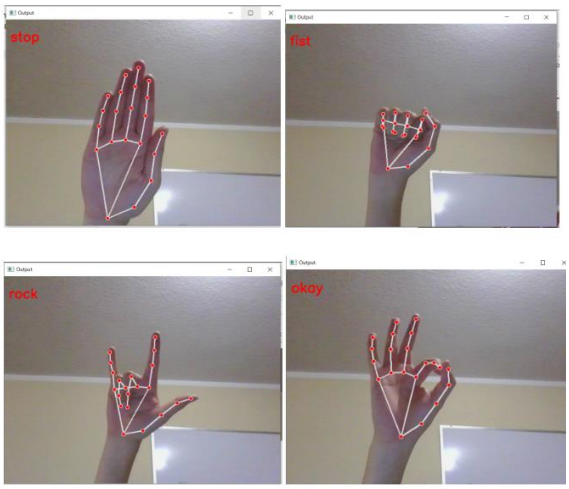

Figura 1. Ejemplos de gestos reconocidos por el algoritmo.

Gracias a esta librería y al programa de visión por computadora, podremos utilizar estos gestos para enviar una señal en forma de *byte* al LoCoBot haciendo uso de un socket (Ver apartado: "Comunicación") por conexión TCP y posteriormente recibirlos en el LoCoBot, descifrarlos y realizar la acción correspondiente previamente programada.

Con todo esto podremos hacer uso del robot en un área de trabajo de hasta 180 grados frente al robot, lo cual permitirá una manipulación más fluida y con un mejor ángulo de trabajo.

Para entender mejor el funcionamiento del robot procederemos a explicar cada uno de los componentes necesarios para su funcionamiento y las librerías que hemos ocupado para este proyecto.

II. METODOLOGÍA

A. Locobot

1. Descripción

El LoCoBot es un manipulador móvil de bajo costo desarrollado por la universidad de Carnegie Mellon, diseñado para trabajar con PyRobot y ROS. PyRobot es un paquete de Python utilizado para desarrollar proyectos de robótica, sin tener que trabajar directamente con el software del robot que se esté utilizando. ROS (Robot Operating System) es un conjunto de librerías de software y herramientas que permiten desarrollar aplicaciones en el área de robótica.

El LoCoBot está montado sobre una base Kobuki YMR-K01-W1, la cual, debido a sus características permite realizar una navegación precisa a través del espacio de trabajo.

Figura 2. Base Kobuki YMR-K01-W1

Asimismo, el LoCoBot está controlado por una PC modelo Intel NUC Mini PC, la cual cuenta con: un procesador Intel Dual-Core i3 de 8va generación, memoria RAM DDR4 de 8GB, disco duro de estado sólido (SSD) de 240GB, tarjeta gráfica Intel Iris Plus Graphics 655, así como conectividad Wifi y Bluetooth 5.0, puertos Gigabit Ethernet, HDMI, Thunderbolt 3 y USB, lector de tarjetas SD y Ubuntu 20.04.

Figura 3. Intel NUC Mini PC

De igual manera, emplea un manipulador móvil WidowX 250s y una cámara Intel RealSense Depth Camera D435.

Figura 4. a) Intel RealSense Depth Camera D435 b) WidowX 250s

Finalmente, una vez ensamblado, el LoCoBot empleado para el propósito de este proyecto luce de la siguiente manera:

Figura 5. Manipulador móvil LoCoBot empleado

2. Características

A continuación, se resumen las características del LoCoBot empleado en este proyecto:

- Cámara Intel RealSense Depth Camera D435.
- Acelerómetro, giroscopio
- Sensores de detección de esquinas y colisiones
- Base móvil Kobuki
- Brazo robótico WidowX 250s
- INTEL NUC - NUC7i5BNH
- Intel Dual Core i3-7260U 2.20 GHz Upto 3.4GHz
- 8GB Ram
- 250GB SSD
- 802.11AC WiFi / Bluetooth 4.0

Brazo robótico:

- 5 DOF (grados de libertad)
- Alcance de 550mm
- Precisión de ~1mm
- Capacidad de carga de 200g

3. Configuración

Para poder emplear el LoCoBot en nuestro proyecto, primero fue necesario aplicar algunas configuraciones iniciales. El primer paso fue realizar una actualización mediante el comando sudo apt update. A continuación, se instalaron las librerías y software con las que trabaja el robot. Estas se encuentran en repositorios de github desarrollados por Trossen Robotics (Interbotix). Una vez instalados los paquetes, se procedió a ubicar las librerías y funciones que serían empleadas en nuestro proyecto.

4. Librerías

Para el propósito de este proyecto se utilizaron las siguientes librerías y paquetes:

- **nterbotix_ros_manipulators**
  Paquetes de ROS que sirven para controlar diferentes tipos de brazos robóticos, entre ellos el que utiliza el LoCoBot (WidowX 250s).
- **interbotix_ros_core**
  Bloques de construcción necesarios para programar el robot y sobre los que se basan el resto de los paquetes para poder compilarse.
- **interbotix_ros_rovers**
  Diversos paquetes necesarios para controlar el manipulador móvil LoCoBot.
- **interbotix_ros_toolboxes**
  Wrappers de ROS y módulos de interfaz utilizados por otros paquetes del robot.

5. Funciones utilizadas

Para realizar el movimiento del robot, se utilizaron principalmente funciones de trayectoria y posición. A continuación, se describen los movimientos realizados por el robot, así como su relación con los gestos de la mano:

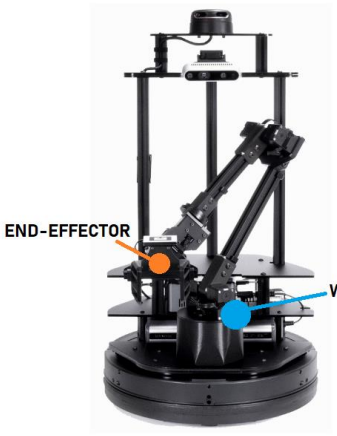

Figura 6. Manipulador móvil LoCoBot empleado

- **Rotación del brazo robótico de 180° (-90° a 90°)**
  Gesto: "Rock" y movimiento lateral de dedo. Para lograr esto, el robot recibe una señal en radianes y este dato a su vez se envía como parámetro en una función que determina la posición de la articulación "waist" (figura 6), permitiendo que el *gripper* pueda ir a cualquier ubicación en un rango de 180°.
- **Posición de seguridad**
  Gesto: "Okay".
  Posición de seguridad del robot deseada para el *end-effector* (figura 6) una vez recibido el gesto. Para esto, se definieron los siguientes parámetros: x = 0.3, y = 0.0, z = 0.15, roll = 0.0, pitch = 0.0, yaw = 0.0.
- **Trayectorias en x y z**
  Gesto: "Call me" y "Peace".
  Trayectorias realizadas en x y z para el *end-effector* (figura 6) una vez recibido el gesto. Para esto, se definieron los siguientes parámetros para el gesto "Call me": x = 0.1, y = 0.0, z = - 0.185, roll = 0.0, pitch = 0.0, yaw = 0.0. Y los siguientes parámetros para el gesto "Peace": x = -0.1, y = 0.0, z = 0.185, roll = 0.0, pitch = 0.0, yaw = 0.0. Esto quiere decir que el robot realizará el desplazamiento indicado en cada parámetro.
- **Rotación a -90°**
  Gesto: "Thumbs up".
  Una vez recibido el gesto, la articulación "waist" (figura 6) se dirige a la posición -90°.
- **Trayectoria en pitch**
  Gesto: "Thumbs down" y "Smile".
  Rotación realizada en "pitch" para el *end-effector* (figura 6) una vez recibido el gesto. Para esto, se definieron los siguientes parámetros para el gesto "Thumbs down": x = 0.0, y = 0.0, z = 0.0, roll = 0.0, pitch = 1.6, yaw = 0.0. Y los siguientes parámetros para el gesto "Smile": x = 0.0, y = 0.0, z = 0.0, roll = 0.0, pitch = -1.6, yaw = 0.0. Esto quiere decir que el robot realizará la rotación indicada para "pitch".
- **Apertura y cierre del *gripper***
  Gesto: "Fist" y "Live long".
  El *gripper* (figura 6) se cierra cuando recibe el gesto "Fist" y se abre cuando recibe el gesto "Live long".

B. Reconocimiento Facial

Cómo solicitud del profesor y para agregar una mayor complejidad al proyecto, incluimos un sistema de seguridad, mediante un programa de reconocimiento facial, que permite acceder a la parte de control por visión del robot únicamente a los usuarios previamente registrados mediante imágenes de cara completa y nombre de usuario en una base de datos.

El programa que utilizamos para el reconocimiento facial fue proporcionado por el profesor como material de clase, dicho programa está dividido en tres archivos y la base de datos. Estos están escritos en *python* y funcionan principalmente gracias a la librería de OpenCV para la adquisición de video y el módulo de "face recognition", autoría de Adam Geitgey.

Específicamente usamos la librería de OpenCV para la captura de video de la cámara web en una definición establecida, tomar un frame de este, escalar a ¼ de su tamaño para así tener una mayor rapidez en el procesamiento, pasar de BGR (formato usado por OpenCV) a RGB (formato usado en el programa), mostrar el video de salida ya con la detección de rostro, su etiquetado correspondiente y el cierre del programa al presionar la tecla "q".

El módulo "face_recognition" intentar reconocer, de una manera semejantemente al humano, la cara de un individuo mediante diferentes algoritmos de machine learning, los cuales dividiremos en 4 puntos esenciales:

Figura 7. Representación del funcionamiento del face recognition.

1. Face Detection

Para la detección de rostros usamos un método llamado Histogram of Oriented Gradients(HOG), donde primero tenemos que:

1. Pasar la imagen a blanco y negro
2. Comparar el valor de negro de cada pixel respecto a sus vecinos
3. Dibujar una flecha en dirección al vecino más oscuro
4. Promediar la dirección en cuadrados de 16x16
5. Para encontrar el rostro se busca el pedazo de la imagen que más se parezca a un patrón HOG extraído de una base de datos entrenada.

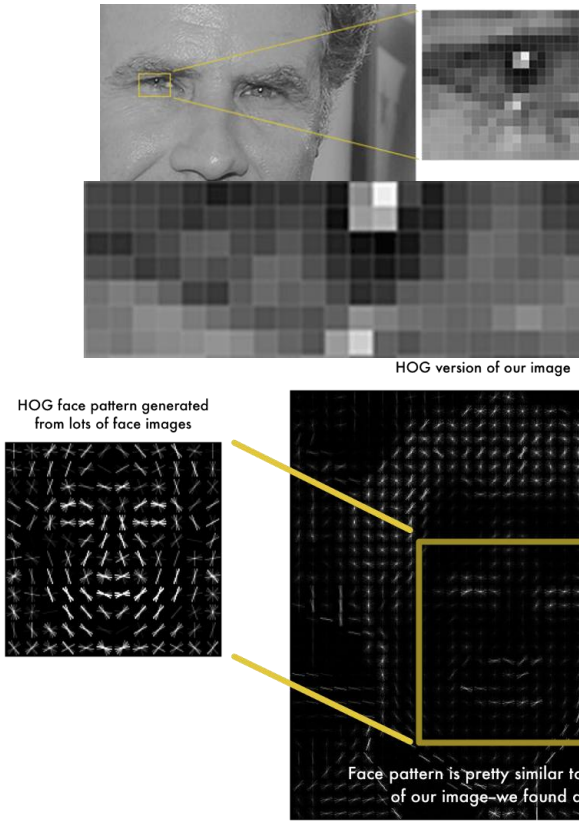

Figura 8. a) Representación de los puntos 1 y 2
b) Representación del punto 3.
c) Representación de los puntos 4 y 5.

2. Posing and Projecting Faces

Ya que las computadoras no reconocen como un mismo rostro cuando este está volteando en otra dirección trataremos de deformar la imagen para que sea similar a las demás mediante el algoritmo de Face Landmark Estimation, el cual con 68 puntos específicos que existen en cada rostro entrena una machine learning para que estos puedan ser encontrados en cualquier rostro y dependiendo de su ubicación podamos hacer transformaciones afines(rotación, escala, etc..) para centrar el rostro lo mejor posible.

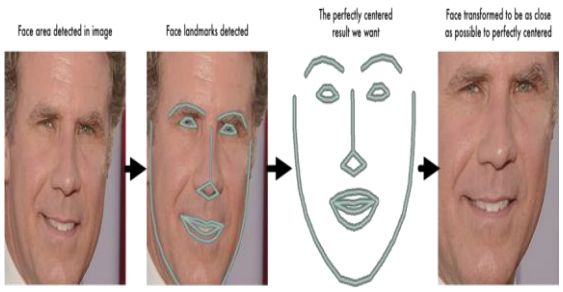

Figura 9. Ejemplificación del algoritmo de Face Landmarks y transformación afín.

3. Encoding Faces

En esta parte básicamente hacemos mediciones específicas del rostro desconocido para que pueda ser comparada y asignada a un rostro conocido con los valores más cercanos, esto lo hacemos mediante una Deep Convolutional Neural Network(CNN) la cual, gracias al entrenamiento, en el que toma 2 imágenes del mismo rostro y una tercera de otro rostro, ha decidido hacer las 128 mejores mediciones en la que las primeras dos imágenes tengan los datos más similares, mientras que la tercera los difernetes, para poder así realizar reconocimientos precisos.

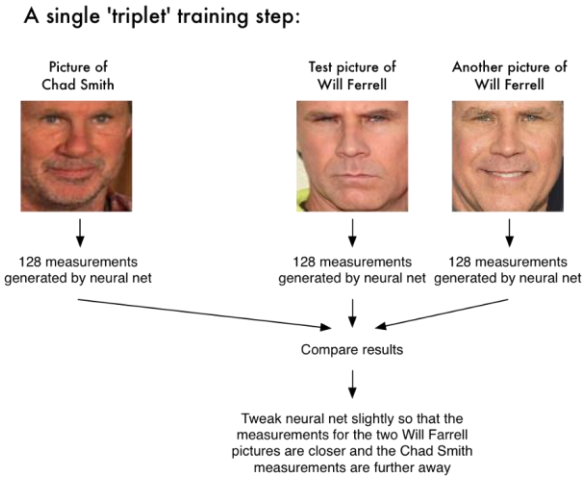

Figura 10. Ejemplificación del entrenamiento de la CNN.

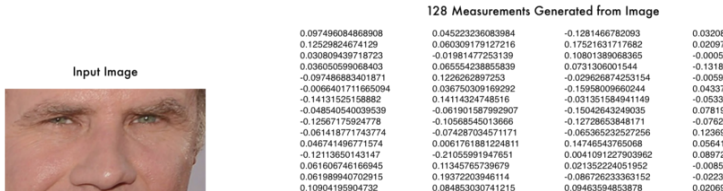

Figura 11. 128 mediciones generadas a partir de la imagen.

4. Finding the person's name from the encoding

Para encontrar a la persona a la que pertenece el rostro en nuestra base de datos se usa un Support Vector Machine(SVM) Classifier, el cual se entrena para tomar los valores de la medición de una nueva imagen y encontrar a la persona que tenga imágenes con los valores más cercanos.

5. Base de Datos

Para la creación de de la base de datos es necesario crear una carpeta con el nombre "known_faces" en la que dentro de ella se crearán carpetas independientes con el nombre de cada uno de los usuarios, dentro de ellas podrán colocar sus respectivas imágenes.

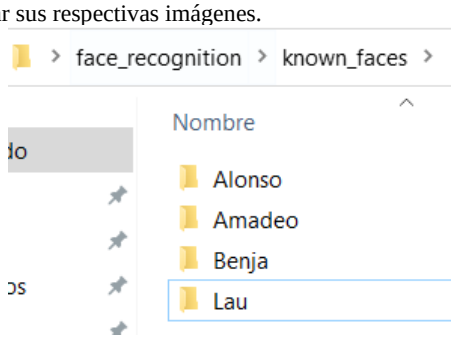

Figura 12. Ejemplificación del correcto orden de la base de datos.

6. Demostración

Una vez implementados estos pasos llegamos a un resultado en el que con unos “if” logramos condicionar el acceso al programa para controlar el robot, el cual únicamente nos será accedido si somos reconocidos por este.

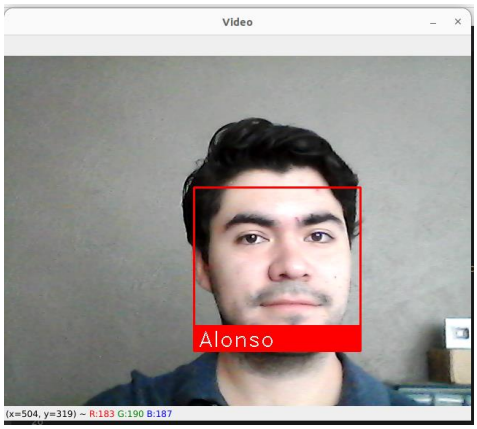

Figura 13. Ejemplificación del correcto resultado del programa.

C. Reconocimiento de mano

Se podría decir que el brazo robótico como su nombre lo indica es una recreación de un brazo humano que intenta imitar sus capacidades, y en ciertos casos mejorarlas. Actualmente los mecanismos de este tipo son controlados de diferentes maneras, desde programas automatizados con rutas específicas hasta algunos controlados por humanos con complejos controles.

El equipo creyó que una de las mejores maneras de controlar el brazo era mediante nuestro propio cuerpo, sin el uso de controles físicos, simplemente permitiéndonos tener una semejanza a la forma en que nosotros manejamos nuestro cuerpo. Es por ello que se decidió usar gestos de la mano para indicarle al robot las acciones que debería de realizar. Con esta decisión usamos las capacidades que tienen las computadoras actuales para procesar imágenes y al mismo tiempo no usamos terceros equipos costosos y difíciles de manejar.

Se encontró que el mejor método para llegar al objetivo de detectar gestos era mediante Python y su amplia variedad de librerías que nos permiten usar redes neuronales y modelos pre entrenados. Se decidió usar principalmente las librerías de MediaPipe y TensorFlow.

Primero fue necesario detectar la mano, lo que fue sencillo gracias al modelo pre entrenado de gran eficacia de Media Pipe. Esta solución ofrece la posición de 21 puntos de referencia en 3D de una mano a partir de un solo fotograma.

La documentación oficial de la librería nos permite conocer acerca de su creación y el entrenamiento del mismo. Se menciona que usa Machine Learning para el entrenamiento, que se divide respectivamente en dos diferentes modelos que en complemento generan la solución. El primer modelo consiste en un sistema de detección de la palma de la mano, y el segundo un modelo de puntos de referencia de la mano que opera en la región detectada por el anterior modelo. Dando como resultado la detección precisa de diferentes puntos clave de una mano en 3D.

El primer modelo ayuda aportando con precisión la imagen de la mano recortada, lo que facilita el proceso para el segundo modelo que se encarga de detectar las coordenadas.

El modelo usado para la detección de la palma está basado en un detector de un solo disparo (SSD: Single Shot MultiBox Detector). La detección de manos permite detectar manos en diferentes escalas, ocluidas y auto ocluidas. Para realizar esto se toman en cuenta características del contexto adicionales como el brazo, el cuerpo o la persona. Este modelo permite enfocarse primeramente en detectar las palmas ya que es más sencillo que detectar dedos articulados.

1. SSD: Single Shot MultiBox Detector

El SSD es un método para detectar objetos en imágenes usando una sola red neuronal. Esto se logra discretizando el espacio de salida en una configuración de recuadros de diferentes tamaños y escalas. La red reconoce objetos en una imagen que se encuentran dentro de una zona de la imagen y se auto ajusta para encajar un mejor recuadro (figura 14).

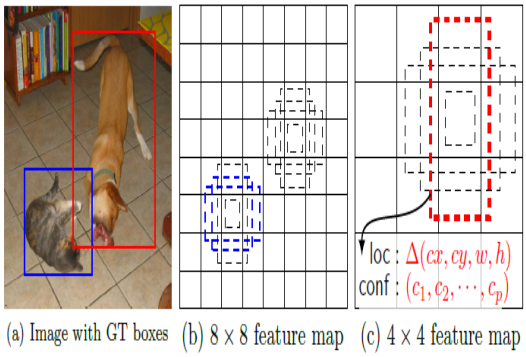

Figura 14. a) SSD solo necesita de una imagen de entrada y cuadros para cada objeto durante el entrenamiento. Posteriormente se evalúa la posición de los objetos en recuadros con diferentes escalas b) y c).

SSD es un detector más rápido que otros algoritmos como YOLO, y más preciso que Faster R-CNN. El núcleo de SSD consiste en predecir las puntuaciones de las categorías y los desplazamientos de las recuadros para un conjunto fijo de cuadros delimitadores por defecto utilizando pequeños filtros convolucionales.

SSD está basado en una red convolucional que produce una colección de recuadros y puntuaciones para la presencia de cierta clase de objetos. Algunas de las capas de la red tienen características convolucionales para disminuir el tamaño progresivamente y permitir predicciones en diferentes escalas. Otras capas están dedicadas a una serie de filtros para predecir parámetros de una posible detección de objetos. En la figura 15 se presenta un resumen de las capas utilizadas para la red neuronal SSD.

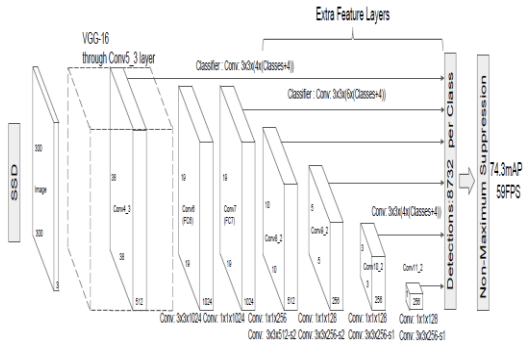

Figura 15. El modelo SSD añade varias capas de características al final de una red base, que predicen los desplazamientos a cajas predeterminadas de diferentes escalas y relaciones de aspecto y sus confidencias

El entrenamiento de esta red requiere información específica de la salida de la detección de alguna clase de objetos en una imagen. Es necesario una gran cantidad de imágenes en las que varíe el lugar de los objetos en la imagen, las escalas y los tamaños.

2. Modelo Hand Landmark

El segundo modelo toma como referencia el objeto detectado por SSD, en este caso la palma de una mano y realiza una localización precisa de 21 puntos clave en coordenadas 3D (ya que también detecta profundidad), como se muestra en la figura 16.. El modelo aprende una representación consistente de la postura interna de la mano y es robusto incluso para manos parcialmente visibles y auto oclusiones. Este modelo se entrenó con 30 mil imágenes en diferentes escenarios.

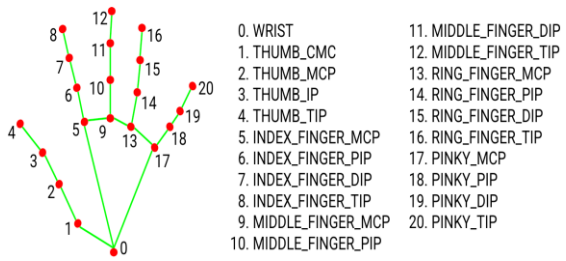

Figura 16. 21 puntos de referencia de la mano

La unión de ambos modelos nos permite distinguir la mano dentro de diferentes escalas, tamaños, escenarios y oclusiones.

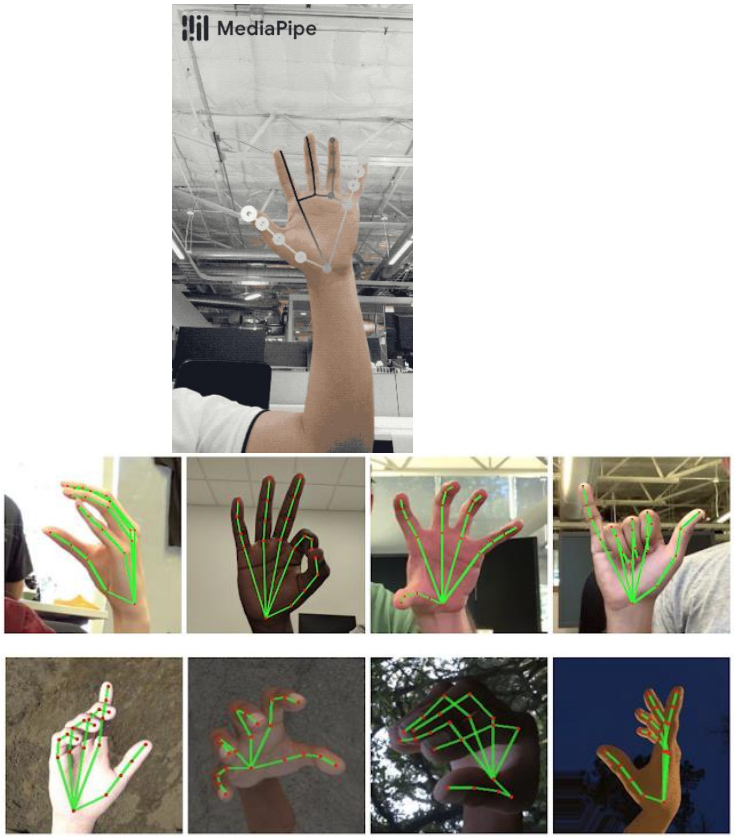

Figura 17. Los puntos de referencia de la mano 3D rastreados están representados por puntos en diferentes tonos, y los más brillantes denotan puntos de referencia más cerca de la cámara.

3. Implementación

Nuestro equipo decidió usar esta librería que nos permite detectar con alta fidelidad la mano del usuario para generar los movimientos. La mayoría de las indicaciones que se le da al robot son controladas por el reconocimiento de gestos que le indican al robot dirigirse a una posición determinada o fijar una trayectoria o movimiento (esto se explicará a profundidad en la siguiente sección), pero tambien hacemos uso de esta librería para detectar la posición de la punta del dedo índice para que el brazo robótico imita el movimiento del dedo, es decir implementamos un algoritmo que detecta la coordenadas en X, es decir horizontal a la cámara y se mapea en radianes para indicarle al robot que gire ese valor en radianes, permitiéndonos tener una mayor libertad en movimientos. Esta configuración se alcanza al hacer el gesto de Rock (de igual manera se explicará en la siguiente sección).

D. Reconocimiento de Gestos

Posteriormente implementamos un algoritmo entrenado con la librería TensorFlow para reconocer gestos, esto se logró gracias a MediaPide y su modelo pre entrenado de detección de manos y coordenadas específicas de 21 puntos clave que se explicó en la sección anterior.

En esta sección del proyecto tuvimos apoyo de un código encontrado en internet y citado como referencia, en el que una persona entrenó un modelo para detectar 10 tipos de clases/gestos diferentes: [‘okay’, ‘peace’, ‘thumbs

up', 'thumbs down', 'call me', 'stop', 'rock', 'live long', 'fist', 'smile']

Se desconoce el entrenamiento de este modelo, simplemente se implementó con el uso de MediaPipe y TensorFlow, es por ello que reconocemos la labor de esta persona y expresamos nuestra gratitud por compartir su modelo entrenado que se encuentra en la sección de referencias [3], que nos ayudó a usar los 10 gestos reconocidos para mandar indicaciones al robot.

1. Implementación

Uniendo el modelo de detección de manos con el de gestos obtuvimos el reconocimiento de cualquier mano que se presente en la cámara como se muestra en la figura 18.

Figura 18. Ejemplos de gestos reconocidos por nuestro algoritmo.

Como se mencionó en la sección anterior, el gesto de 'rock' nos permite entrar y salir de la sección de indicaciones por medio del dedo índice.

E. Comunicación

Para la comunicación se creó un socket programado usando python y las librerías correspondientes para realizar dicha conexión. Los sockets son usados para mandar mensajes a través de la red, esto provee una forma de comunicación llamada IPC, por su siglas en inglés *Inter-process communication*. La red para este proyecto fue una red creada usando un *access point* a través de nuestros teléfonos celulares. Esto debido a que las conexiones realizadas por medio de la red universitaria son limitadas, incluso rechazadas por el servidor de la universidad, lo cual nos obligó a utilizar otra red para poder realizar dicha conexión. (Jennings, 2022)

El tipo más común de aplicación para los sockets son aplicaciones Cliente-Servidor, donde uno de los lados actúa como servidor y espera una o más conexiones por parte de los clientes, esto permite que un solo servidor pueda recibir más de un mensaje a la vez.

Para este proyecto, conectaremos el LoCoBot, que será nuestro servidor, para poder recibir los mensajes enviados por el cliente. El programa de visión por computadora actuará como nuestro cliente y este mandará las señales necesarias a nuestro servidor. Por ejemplo, el tipo de gesto que está reconociendo para poder manipular al LoCoBot.

Como comentado con anterioridad, se utilizará un programa de Python y las librerías necesarias. En este caso utilizaremos la API Berkeley sockets, la cual nos permitirá realizar un socket de manera sencilla y con la documentación adecuada para su sencillo manejo. (Socket Module In Python, 2022)

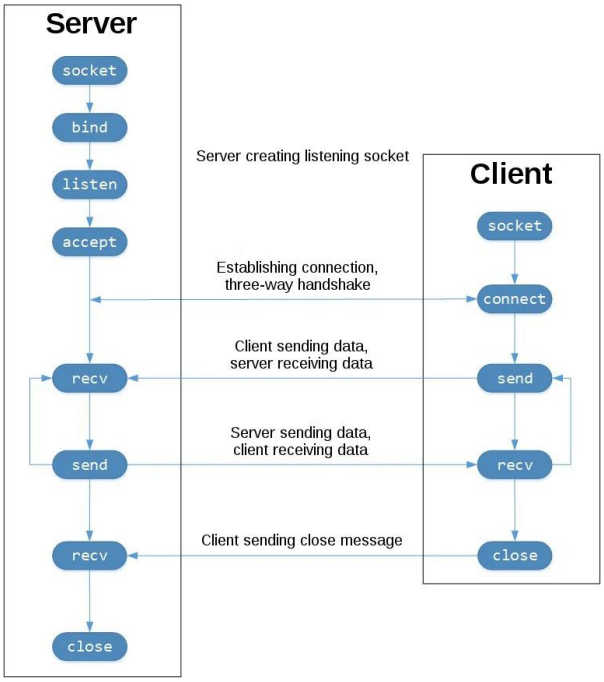

Figura 19. Conexión Cliente-Servidor haciendo uso de sockets.

En el diagrama anterior representa el comportamiento general de una aplicación cliente-servidor, donde observamos del lado izquierdo el servidor. Lo primero que

tenemos que realizar es crear un socket que nos permitirá verificar si alguien se está intentando conectar a nuestro servidor. Para esto, hacemos uso de las funciones *Listen* y *Accept* cuando hayamos detectado que un cliente desea mandar datos a nuestro servidor.

Posteriormente, realizamos la conexión entre cliente y servidor aceptando dicha solicitud, permitiendo que, tanto el cliente, como el servidor envíen y reciban datos. Este intercambio de información es el que permitirá que nuestro programa en Python se comunique con nuestro servidor y nos dé las instrucciones necesarias para manipular el LoCoBot.

Cuando la comunicación quede cerrada y el cliente ya no desee enviar más información, cerraremos la conexión del lado del cliente y, al recibir esta instrucción, el servidor procederá a realizar la misma acción. Si bien para este proyecto el socket que se utilizó fue uno muy sencillo, el uso de sockets es una tecnología que se ha ocupado por mucho tiempo y a gran escala, hoy en día lo conocemos como el internet de las cosas.

III. RESULTADOS EXPERIMENTALES Y ANÁLISIS

Los resultados obtenidos fueron más que prometedores y, a pesar de tener algunos contratiempos, estos fueron muy positivos y con una implementación muy factible para cualquier persona que quisiera replicar los mismos resultados.

Video:
https://youtu.be/VOdJB9F0hDo

En este video se muestra el funcionamiento del robot. Lo primero que observamos es al operador acercándose a la cámara para que el algoritmo de reconocimiento lo pueda reconocer y permitir el acceso. A partir de este momento el operador se encuentra en posición para empezar a realizar los gestos necesarios para mover el robot. Podemos observar que en un momento, después de realizar el gesto de “Rock”, el robot seguirá nuestro dedo índice, el cual indicará por medio de mapeo en qué posición se encuentra. El robot copia esta posición a través de conversión a radianes para girarlo a la posición indicada. Esto nos permitirá mover nuestro robot con hasta 180 grados de libertad y así poder manipular un matraz y combinar las sustancias químicas en cuestión.

Como observamos en el video, los resultados son muy positivos, ya que el robot sigue nuestras instrucciones de manera correcta y de manera clara. Es muy importante que el robot se pueda mover con libertad, pero al mismo tiempo con una precisión buena para que no ocurran accidentes, siendo que el propósito de este proyecto es limitar dichos accidentes.

A. Problemas y Soluciones

Al principio se tuvieron diferentes problemas en cuanto al reconocimiento de gestos. Si bien el programa en Python de visión por computadora reconoce los gestos que nosotros le enviábamos, en ocasiones existía el problema que detectaba otro gesto que nosotros no queríamos mandar, como por ejemplo al momento de que nosotros hiciéramos el gesto de “Thumbs up”, el programa nos detectaba un “Fist” o un “Rock”, lo cual no nos permitía enviar los valores correctos a nuestro robot. Una forma de resolver este problema fue haciendo uso de un contador, donde nos tenía que reconocer el mismo gesto por una cierta cantidad de tiempo para que el programa lo detectara y mandará la señal correspondiente.

Otro de los problemas que se nos presentaron durante la realización de este proyecto fue el tiempo de respuesta del programa de visión por computadora en Python. Este problema se puede observar durante la ejecución del programa de reconocimiento facial, donde al iniciar dicho programa notamos un decremento en los frames que puede capturar la cámara, con un retraso de aproximadamente cinco segundos. Esto supuso un problema ya que para poder ingresar al robot es necesario el reconocimiento facial del usuario y este puede llegar a tardar en obtener los resultados si la persona se encuentra muy alejada de la cámara cuando iniciamos el programa.

También observamos el mismo tipo de problema al momento de ejecutar el seguimiento del dedo índice por parte del robot. Sin embargo, se logró reducir el tiempo de reacción que tarda el robot en recibir y mandar el dato adecuado.

B. Próximas Versiones

Como equipo hemos contemplado seguir realizando este proyecto para el siguiente semestre, ya que creemos que este robot tiene mucho más que dar y, al darnos cuenta de los problemas que tuvimos durante la realización de este proyecto, creemos que estos se pueden resolver. Es por esto que a futuro planeamos resolver estos problemas, además de agregar un control PID para el brazo robótico, así como incorporar más elementos con los que cuenta el LoCoBot, para hacer uso de todas sus herramientas.

IV. CONCLUSIÓN

Para finalizar este reporte retomemos los objetivos planteados en la introducción de este reporte. En cuanto a la manipulación de objetos, podemos observar en el video incluido en la sección de “Resultados y análisis” que en efecto la manipulación de objetos fue posible de manera muy exitosa, ya que podemos mover el matraz como originalmente se planteó.

Si bien los resultados fueron exitosos, aún queda mucho más por hacer. Como se comentó en el apartado anterior, los avances a futuro todavía pueden ser muchos y, al mejorar el robot, podemos utilizarlo más allá del movimiento de matraces. Podríamos tener este robot de manera automatizada, para que pueda reconocer los compuestos que se van a verter y los manipule de manera automática tan solo con introducir el compuesto que queremos como resultado final. También se podría utilizar este mismo proyecto para otros campos de la industria y no solo enfocarse en el área de químicos.

Gracias a este proyecto, pudimos aplicar muchos de los temas que hemos visto a lo largo de la carrera, así como también en las materias de este semestre, específicamente hablando de “Visión para robots”. Donde, a pesar de que no utilizáramos muchos de los elementos que vimos en clase, esta nos sirvió como una base fundamental para comprender de mejor manera el funcionamiento de las librerías que se utilizaron durante este proyecto y poder identificar cuáles podrían ser los avances a futuro. Por ejemplo, el reconocimiento de objetos y poder medirlos fue algo que se investigó, pero nos dimos cuenta que no era necesario para el objetivo que se tenía al momento. Sin

embargo, podríamos proponerlo para una segunda edición de este proyecto.

V. REFERENCIAS

1. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. (2015). SSD: Single Shot MultiBox Detector. Junio 2022, de ARXIV Sitio web: https://arxiv.org/abs/1512.02325
2. Google. (2016). MediaPipe Hands. Junio 2022, de Google Sitio web: https://google.github.io/mediapipe/solutions/hands.html
3. Unknown. (2021). Real-time Hand Gesture Recognition using TensorFlow & OpenCV. Junio 2022, de TECHVIDVAN Sitio web: https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/
4. Geitgey Adam. (2016). Modern Face Recognition with Deep Learning. Junio 2022, de Medium Sitio web: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78
5. *Accidentes Quimicos*. (2022, 6 5). Retrieved from Organización Panamericana de la Salud: http://www.saludydesastres.info/index.php?option=com_content&view=article&id=329:3-3-1-accidentes-quimicos&catid=114:3-3-amenazas-tecnologicas&lang=es
6. *La exposición a sustancias químicas causa 4.000 muertes al año entre trabajadores*. (2010, Noviembre 23). Retrieved from El Pais: https://elpais.com/sociedad/2010/11/23/actualidad/1290466808_850215.html#:~:text=Lloren%C3%A7%20Serrano%2C%20de%20CC%20OO,hay%2018.000%20accidentes%20laborales%20relacionados.
7. María Esther Arcos Serrano, C. I. (2007). *Riesgos Quimicos.* Mexico. doi:978-970-821-006-5
8. *Socket Module In Python*. (2022). Retrieved from Pythontic: https://pythontic.com/modules/socket/introduction
9. Jennings, N. (2022, Febrero 21). *Socket Programming in Python* . Retrieved from Real Python: https://realpython.com/python-sockets/
10. Trossen Robotics. (2021). Interbotix X-Series LoCoBots. marzo 20, 2022, de Trossen Robotics Sitio web: https://www.trossenrobotics.com/docs/interbotix_xslocobots/index.html
11. Open Robotics. (2020). locobot. marzo 20, 2022, de Open Robotics Sitio web: http://wiki.ros.org/locobot
12. Facebook Research. (2020). PyRobot. marzo 20, 2022, de Facebook Research Sitio web: https://pyrobot.org/docs/overview.html
13. Trossen Robotics (Interbotix). (2021). Interbotix. marzo 20, 2022, de Trossen Robotics Sitio web: https://github.com/Interbotix#trossen-robotics-interbotix