Control Remoto mediante gestos

Daniel Asensi Roch
DNI: 48776120C

15 de mayo de 2024

Índice

1. Introducción **2**

2. Estado del arte **3**
2.1. Reconocimiento de Gestos 3
2.2. MediaPipe 3
2.3. ROS (Robot Operating System) 3
2.4. Evasión de Obstáculos 3
2.5. Integración de Tecnologías 4

3. Desarrollo **5**
3.1. show_camera_robot.py 5
3.2. pose_estimation.py 5
3.3. obstacle_avoidance.py y obstacle_detection.py 6

4. Conclusiones **10**

1. Introducción

En esta práctica, el objetivo es desarrollar un sistema de teleoperación para un robot utilizando gestos de la mano como comandos de control. Utilizando el framework de ROS (Robot Operating System) y la librería MediaPipe para el reconocimiento de gestos, se pretende ofrecer una interfaz intuitiva y natural para manejar el robot.

El sistema se compone de varios módulos interconectados: la captura de imágenes del operador, el procesamiento y reconocimiento de gestos, y la generación de comandos de control Ackermann para el robot. Además, se implementará una capa de seguridad basada en la detección de obstáculos, utilizando datos de un sensor LIDAR para evitar colisiones y garantizar la integridad del robot en su entorno operativo.

La metodología de desarrollo incluye la configuración del entorno de trabajo con ROS y las dependencias necesarias, la creación de nodos para la captura y procesamiento de imágenes, y la implementación de algoritmos de reconocimiento de gestos utilizando MediaPipe. Los gestos reconocidos se traducen en comandos de movimiento que se envían al robot a través de ROS. Para la capa de seguridad, se desarrollará un nodo que monitoree continuamente los datos del LIDAR y ajuste los comandos de movimiento para prevenir colisiones.

El entorno de simulación Gazebo se utilizará para desarrollar y probar el sistema en un entorno controlado. Esto permitirá validar el funcionamiento del sistema bajo diferentes condiciones y escenarios, garantizando su robustez y fiabilidad antes de una posible implementación en un robot real.

Esta práctica no solo refuerza conceptos de visión por computadora y control de robots, sino que también proporciona una experiencia práctica en la integración de múltiples tecnologías y herramientas en un proyecto cohesivo. La documentación detallada de cada paso y la justificación de las decisiones de diseño e implementación son aspectos fundamentales de este trabajo, asegurando que el proyecto sea comprensible y mantenible a largo plazo.

2. Estado del arte

El control de robots mediante gestos es un área de investigación activa en el campo de la robótica y la interacción humano-robot. Este enfoque busca crear interfaces naturales y accesibles que permitan a los usuarios controlar robots de manera intuitiva, sin necesidad de dispositivos de entrada físicos tradicionales como joysticks o teclados.

2.1. Reconocimiento de Gestos

El reconocimiento de gestos ha evolucionado significativamente con el avance de las tecnologías de visión por computadora y aprendizaje profundo. Tradicionalmente, los sistemas de reconocimiento de gestos utilizaban métodos basados en la extracción de características manuales y clasificadores como Support Vector Machines [**?**] o Redes Neuronales Artificiales . Sin embargo, estos métodos a menudo eran limitados en términos de precisión y robustez. [1]

Con la llegada del aprendizaje profundo, las técnicas de reconocimiento de gestos han mejorado considerablemente. Las Redes Neuronales Convolucionales y los Modelos de Transferencia de Aprendizaje han permitido el desarrollo de sistemas capaces de detectar y clasificar gestos con alta precisión. Entre las herramientas más destacadas en este ámbito se encuentra MediaPipe de Google, una librería que proporciona soluciones de detección y rastreo de manos en tiempo real utilizando técnicas de aprendizaje automático. [**?**]

2.2. MediaPipe

MediaPipe es una librería de código abierto desarrollada por Google que facilita la implementación de pipelines de procesamiento de medios, incluyendo la detección de manos y el reconocimiento de gestos. Utiliza modelos de aprendizaje profundo optimizados para funcionar en tiempo real, incluso en dispositivos con recursos limitados. MediaPipe Hand Tracking proporciona un modelo preentrenado que detecta 21 puntos clave en cada mano, permitiendo la identificación precisa de la posición y el movimiento de los dedos. [**?**]

2.3. ROS (Robot Operating System)

ROS es un framework flexible para el desarrollo de software de robótica. Proporciona herramientas y bibliotecas que ayudan a construir aplicaciones robóticas robustas y escalables. ROS facilita la comunicación entre diferentes módulos del sistema mediante un sistema de publicación y suscripción de mensajes, lo que es fundamental para integrar la captura de gestos y el control del robot en esta práctica. [2]

2.4. Evasión de Obstáculos

La evasión de obstáculos es una función crítica en la navegación autónoma de robots. Los sensores como LIDAR se utilizan comúnmente para detectar obstáculos en el entorno del robot. Los datos del LIDAR se procesan para identificar la presencia y la proximidad de

obstáculos, y los algoritmos de planificación de movimientos ajustan la trayectoria del robot para evitar colisiones.

2.5. Integración de Tecnologías

La combinación de MediaPipe para el reconocimiento de gestos y ROS para el control y la navegación del robot representa una integración efectiva de tecnologías avanzadas. MediaPipe ofrece una solución eficiente para la interpretación de gestos en tiempo real, mientras que ROS proporciona una infraestructura robusta para la comunicación y el control del robot. La capa de seguridad añadida con la evitación de obstáculos garantiza que el sistema no solo sea intuitivo, sino también seguro y fiable en su operación.

3. Desarrollo

En este apartado, se explicarán los diferentes scripts desarrollados para implementar el sistema de teleoperación por gestos, la captura de imágenes, el procesamiento de gestos y la seguridad basada en la evitación de obstáculos. Cada script cumple una función específica dentro del sistema, y su correcta integración es esencial para el funcionamiento del sistema completo.

3.1. show_camera_robot.py

El script show_camera_robot.py se encarga de suscribirse al tópico de ROS que publica las imágenes de la cámara del entorno del robot y mostrarlas en una ventana de OpenCV, permitiendo al operador visualizar el entorno del robot en tiempo real. Para ello, primero se inicializa el nodo de ROS bajo el nombre 'image_viewer' y se suscribe al tópico /camera/color/image_raw, el cual transmite las imágenes capturadas por la cámara del robot. Cuando se recibe un mensaje de imagen, la función de callback convierte este mensaje de ROS a un formato de imagen que puede ser manejado por OpenCV utilizando la librería CvBridge. Posteriormente, la imagen se muestra en una ventana de OpenCV, actualizándose continuamente para reflejar el entorno en tiempo real. El nodo se mantiene en ejecución mediante rospy.spin() hasta que se recibe una interrupción, momento en el cual todas las ventanas de OpenCV se cierran adecuadamente para liberar recursos.

Este script es para proporcionar al operador una visualización en tiempo real del entorno del robot, lo cual es crucial para el control basado en gestos y la navegación segura del robot. La capacidad de ver el entorno del robot ayuda a tomar decisiones informadas y rápidas. La imagen generada se puede apreciar en 3

3.2. pose_estimation.py

El script pose_estimation.py tiene como objetivo principal capturar imágenes del operador, procesarlas para reconocer gestos de la mano y traducir estos gestos en comandos de control Ackermann para el robot. El nodo ROS inicializado bajo el nombre gesture_based_control se suscribe al flujo de video de la cámara, utilizando OpenCV para la captura de imágenes en tiempo real. Una vez capturada la imagen, se convierte de BGR a RGB para ser procesada por MediaPipe,

La función classify_gesture se utiliza para interpretar los puntos clave detectados y clasificar el gesto en una de las siguientes categorías: mover hacia adelante, mover a la izquierda, mover a la derecha o detenerse. Dependiendo del gesto reconocido, se generan comandos Ackermann específicos con diferentes velocidades y ángulos de dirección. Por ejemplo, un gesto de "mover hacia adelanteresultará en un comando con una velocidad de 1.0 y un ángulo de dirección de 0.0, mientras que un gesto de "mover a la izquierda.ªsignará una velocidad de 0.5 y un ángulo de dirección de 90 grados. Estos gestos se pueden apreciar en la figura 2d

Los comandos generados se publican en el tópico /blue/preorder_ackermann_cmd, permitiendo que el robot responda en tiempo real a los gestos del operador. Además, para proporcionar retroalimentación visual al operador, los puntos clave detectados y los gestos

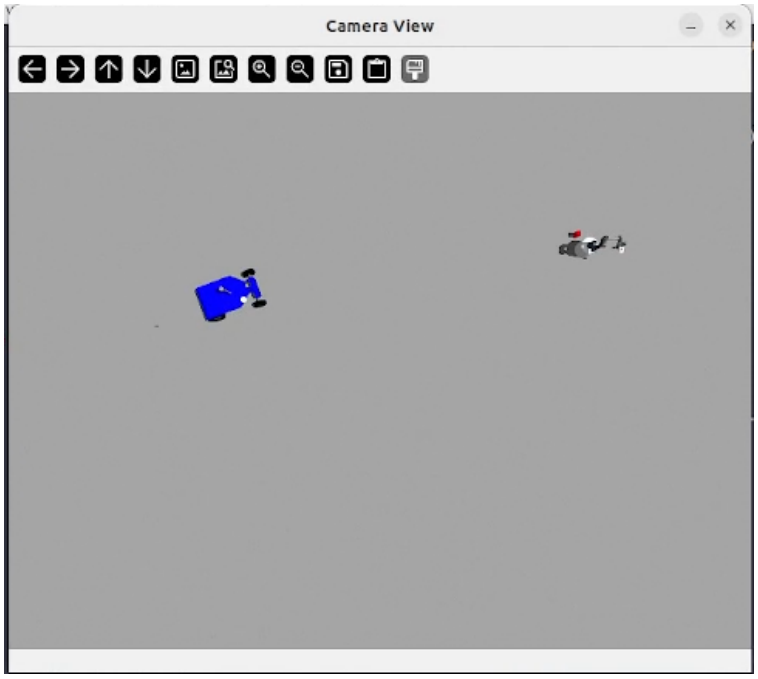


Figura 1: Imagen de muestra Script

reconocidos se dibujan sobre la imagen original y se muestran en una ventana de OpenCV. Esta ventana se actualiza continuamente para reflejar el flujo de video en tiempo real.

El ciclo de procesamiento se ejecuta a una tasa de 30 Hz para asegurar una respuesta rápida y fluida del sistema. En caso de que se reciba una señal de interrupción, el script cierra todas las ventanas de OpenCV y libera los recursos de la cámara y de MediaPipe. Este script es crucial para permitir una interacción intuitiva y en tiempo real entre el operador y el robot, aprovechando gestos naturales para controlar el movimiento del robot de manera eficaz y segura.

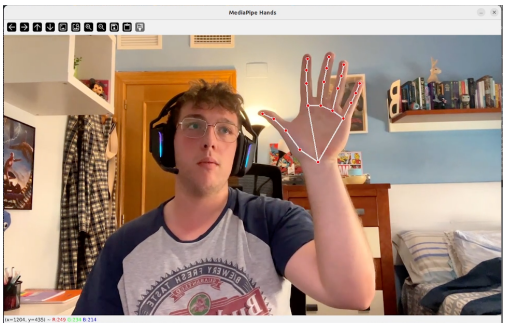
3.3. obstacle_avoidance.py y obstacle_detection.py

El script obstacle_detection.py se encarga de procesar la nube de puntos recibida de un sensor LIDAR, identificar los obstáculos en el entorno del robot y publicar esta información para su uso por otros nodos de ROS. El nodo se inicializa bajo el nombre point_cloud_filter_node y se suscribe al tópico /blue/velodyne_points para recibir la nube de puntos. La función principal del script filtra los puntos de la nube para identificar aquellos que representan obstáculos, basándose en una altura y un radio predefinidos. Los puntos que superan la altura especificada se consideran obstáculos y se publican en el tópico /obstacles. Además, se identifican y publican las zonas libres de obstáculos en el tópico /free_zone.

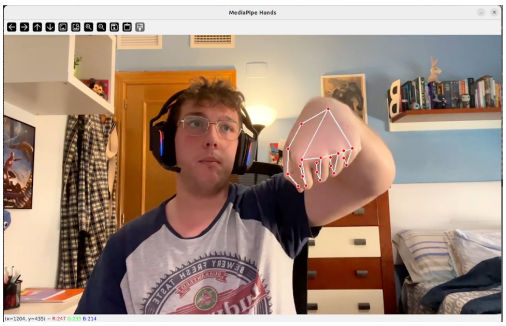
Obstacle_avoidance.py utiliza la información publicada por obstacle_detection.py para garantizar la navegación segura del robot. Inicializado como obstacle_avoidance, el nodo se suscribe a los tópicos /obstacles y /blue/preorder_ackermann_cmd, el primero para recibir datos sobre obstáculos detectados y el segundo para recibir comandos de control Ackermann.

La función obstacle_callback procesa la nube de puntos de obstáculos para determinar si hay obstáculos cercanos, y la función ackermann_callback almacena los comandos Ackermann recibidos. Dependiendo de la presencia de obstáculos, la función modify_ackermann_command decide si detener el robot o permitir que continúe con el comando original, publicando el comando modificado en el tópico /blue/ackermann_cmd.

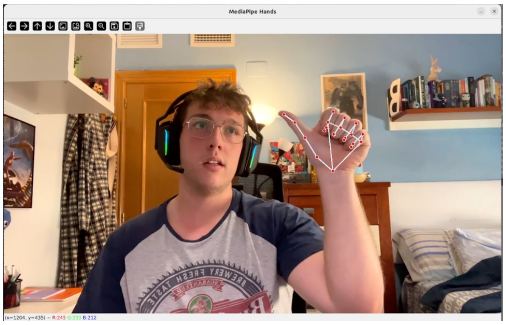
Ambos scripts trabajan en conjunto: obstacle_detection.py detecta obstáculos y publica esta información, mientras que obstacle_avoidance.py utiliza estos datos para modificar los comandos de movimiento del robot, garantizando que el robot navegue de manera segura en su entorno. Este sistema integrado asegura que el robot pueda operar eficientemente, evitando colisiones y ajustando su comportamiento dinámicamente según el entorno detectado por el LIDAR.



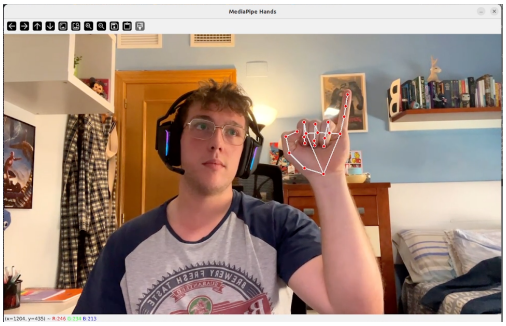
(a) Gesto Adelante



(b) Gesto Stop



(c) Gesto Izquierda



(d) Gesto Derecha

Figura 2: Gestos

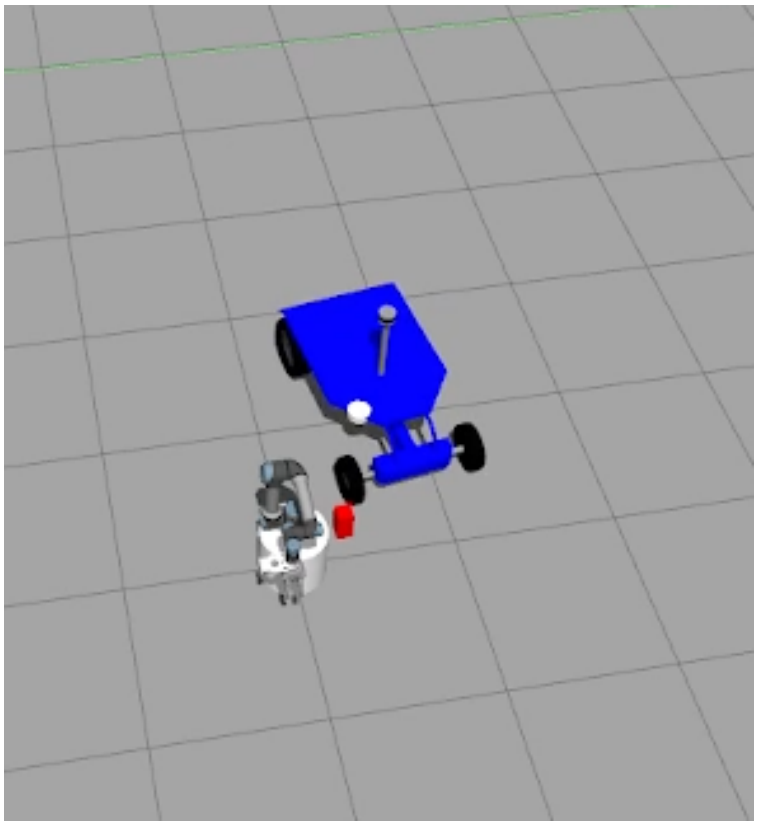


Figura 3: Robot frena antes de obstáculo

4. Conclusiones

En esta práctica hemos desarrollado un sistema de teleoperación para un robot mediante gestos, utilizando ROS y MediaPipe para el reconocimiento de gestos y la detección de obstáculos. La implementación incluyó la captura y procesamiento de imágenes, la clasificación de gestos, y la generación de comandos de control, asegurando la navegación segura del robot.

Este proyecto no solo consolidó nuestros conocimientos en visión por computadora y control de robots, sino que también nos permitió integrar múltiples tecnologías de manera efectiva. La experiencia ha sido muy enriquecedora y me ha permitido comprender mejor la interacción entre diferentes módulos en un sistema robótico.

Para ver el funcionamiento del sistema, hemos preparado un video demostrativo disponible en la siguiente URL: https://youtu.be/ee62IL55yNw

Referencias

[1] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand gesture recognition with 3d convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2015, pp. 1–7.

[2] A. Koubâa *et al.*, *Robot Operating System (ROS).* Springer, 2017, vol. 1.