Gesture-based teleoperation system

Javier Prada de Francisco

Universidad de Alicante, San Vicente del Raspeig, 03690, España
https://www.ua.es/es/

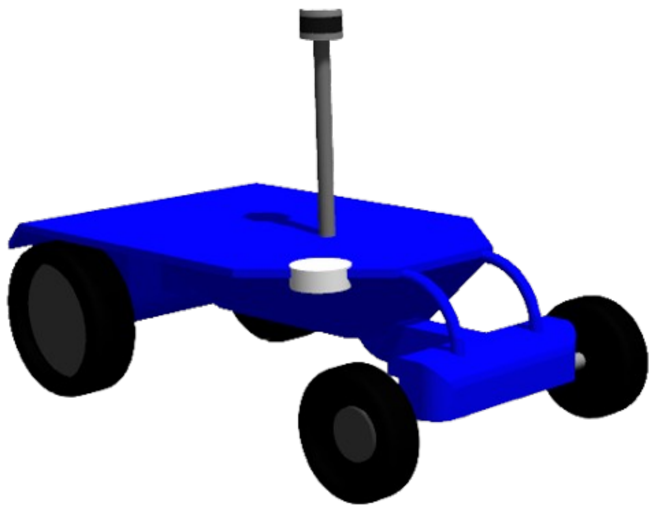

Table of Contents

Gesture-based teleoperation system 1
Javier Prada de Francisco
1 Introduction 3
2 State of art 3
3 Development 4
3.1 Show_camera_robot.py 4
3.2 Capture_video.py 4
3.3 Obstacle_avoidance.py 4
3.4 Pose_estimation.py 4
4 Experimentation 5
5 Conclusions 5

1 Introduction

In this practice, a robot teleoperation system has been developed by hand gestures, using ROS and computer vision. The main idea is that the operator can control the robot simply by hand gestures in front of a camera, without the need for physical controls.

For this purpose, the MediaPipe library has been used to detect the key points of the hand in real time. From these points, basic gestures are identified (move forward, turn left or right), which are then translated into movement commands for the robot. In addition, the project has worked on a simulation in ROS, and replaces autonomous navigation with manual control by gestures.

All the code can be found in the next GitHub repository: `https://github.com/javierpradad/Rob-tica-pr-ctica-2`

2 State of art

Human-robot interaction (HRI) has evolved significantly in recent years, seeking more natural and accessible forms of communication. Among the various alternatives, gesture control has established itself as an intuitive solution, especially in contexts where it is impractical to use physical controls or traditional interfaces.

One of the most prominent technologies in this field is MediaPipe[1], a library developed by Google that allows real-time detection of key points of the human body, such as hands, face or full posture. Its hand detection module, in particular, offers 21 reference points per hand with high accuracy, and has become a very useful tool for real-time gesture recognition applications.

On the other hand, ROS (Robot Operating System)[2] has positioned itself as the standard in robot software development by providing a modular infrastructure that facilitates communication between different system components. Its node-based architecture makes it possible to integrate sensors, vision systems, control algorithms and simulations in a simple way.

In this practice, these technologies have been integrated to build a system that allows to control a robot by means of gestures captured by camera, translated into motion commands, and processed in a simulated environment with ROS.

3 Development

3.1 Show_camera_robot.py

This script receives the images of the robot's environment through the /camera/color/image_raw topic. Here what I did was to complete the part where the image was displayed using OpenCV. I added cv2.imshow() and cv2.waitKey(1) so that a window would open and the images could be viewed in real time. This allowed me to have a view of the robot's environment while controlling it with gestures, which is key to know where it is going.

3.2 Capture_video.py

This file is responsible for obtaining the image from the operator's webcam. At first I programmed it to read the image from a video since I did not have the camera available, but as for the car control is more fluid and flexible, I modified the code to add to capture directly from the webcam, using cv2.VideoCapture(0), leaving commented the part of the video in case you want to use it in one way or another. Then, those images are converted to ROS messages and published in the /operator/image topic. With this, the system can now "see" what the operator is doing with his hands.

3.3 Obstacle_avoidance.py

Finally, this node acts as a kind of "safety filter". Here I completed the logic so that the robot does not move forward if it detects an obstacle nearby. I processed the LIDAR point cloud and, if there is any object between 0 and 1.5 meters in front of the robot, with a lateral margin of ± 0.5 meters, I force the velocity to zero. The final command is posted to /blue/ackermann_cmd, ensuring that the robot only moves when it is safe to do so.

3.4 Pose_estimation.py

Finally, in this script I completed all the necessary logic to process the image received from the /operator/image topic using the MediaPipe library. First, I transformed the image from BGR to RGB, as this is the format that MediaPipe requires to work correctly. Then, with the hand detection model enabled, I obtained the landmarks for each detected hand. From those points, I implemented a very simple classification of gestures: if the index finger is above the thumb, it is interpreted as "move forward"; if it is to the left, it is "turn right"; and if it is to the right, it is understood as "turn left". These reverse controls are due to the mirror mode of the webcam. Once the gesture was identified, I generated an AckermannDrive-like message with a specific speed and steering angle for each case, and published it in the /blue/preorder_ackermann_cmd topic. In addition, I added the visualization of the hand points and the connections between them using OpenCV, so that I could see in real time how MediaPipe interprets the operator's hand.

4 Experimentation

I started the tests by launching the show_camera_robot.py node, which allowed me to see in real time the robot's environment through an OpenCV window.

Then, I tested the capture_video.py node. Initially I ran it using a recorded video, which allowed me to verify that the system worked correctly without relying on the real-time camera. Once that was validated, I modified the code to capture directly from the webcam, and verified that the images were published correctly in the /operator/image topic.

Next, I launched pose_estimation.py to process the operator images. I checked that MediaPipe detected well the key points of the hand, and that the defined gestures (move forward, turn left and right) were recognized correctly, but that the camera having mirror mode detected the movements backwards, that is, if I point my hand to the left the car turns to the right and vice versa, so I had to reverse the directions afterwards. I also confirmed that the proper commands were generated and posted to the /blue/preorder_ackermann_cmd topic.

Finally, I ran the obstacle_avoidance.py node and performed forward gestures with and without obstacles in front of the robot. I was able to verify that, when there was an object in its path, the system stopped the robot automatically, which proved that the safety layer was working properly.

Overall, all tests confirmed that the system is functional, reacts well to the operator's gestures and takes into account the safety of the environment.

5 Conclusions

The practice has allowed me to develop a complete gesture teleoperation system using ROS and MediaPipe. I have learned to integrate different modules to capture images, detect gestures and control the robot's movement, all within a simulated environment.

The system responds well to defined gestures and the safety layer effectively prevents collisions. Although it is a simple implementation, it demonstrates that gesture control is a viable and natural option for interacting with robots.

References

1. C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C. Chang, M. G. Yong, J. Lee, W. Chang, W. Hua, M. Georg, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," *CoRR*, vol. abs/1906.08172, 2019. [Online]. Available: http://arxiv.org/abs/1906.08172
2. Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org