Instituto Superior de Engenharia do Porto

Departamento de Engenharia Informática

Mestrado em Sistemas Computacionais Críticos

Recognition of traffic signals in Autonomous Systems

Unidade Curricular: IACOS - Intelligent, Autonomous, Cooperative Systems

Everton Matheus Oriente

Nº: 1190033 Email:1190033@isep.ipp.pt

Rafael Calai

Nº: 1220539 Email:1220539@isep.ipp.pt

Ricardo Mendes

Nº: 1201779 Email:1201779@isep.ipp.pt

December 9, 2023

Contents

1 Introduction

In the initial phase of our study, we delved into the current landscape of autonomous systems, examining how they navigate and respond to the unconventional challenges of everyday scenarios. We scrutinized the diverse strategies employed to address these issues and investigated the innovative approaches employed to surmount such challenges.

Building upon this preliminary investigation, our subsequent milestone introduced a compelling project. This project focused on the nuanced realm of gesture recognition by individuals to guide autonomous systems in their decision-making processes—deciding whether to advance, halt, turn right or turn left.

Our proposal sought to leverage the findings from our research as a cornerstone for developing an advanced gesture recognition algorithm. Subsequently, we encapsulated this algorithm within a node of the ROS 2 framework. The aim was to establish seamless communication between this dedicated node and turtlesim, a simulation platform representing a vehicle that responds to signals conveyed by an individual.

This endeavor involved the creation of a bespoke node to publish messages. Turtlesim, in turn, interpreted and executed actions based on the received commands, essentially emulating a vehicle respecting the signals demonstrated by an individual. This integration showcased our proficiency in adapting and embedding our code into the ROS 2 framework, aligning precisely with the project's stipulated requirements.

2 Architecture

To control the TurtleSim this project uses a ROS2 to create nodes and communicate, python with mediapipe and OpenCV to estimate poses, and python rclpy to control the TurtleSim. The script has two modules "action_reconition.py", "turtle_cmd.py" and the "main.py" itself that integrates them to send commands to the TurtleSim.

2.1 ROS2

ROS 2 has an architecture where nodes seamlessly communicate with each other, some nodes publish information while others consume it. Nodes in ROS 2 possess the versatility to both publish and consume, breaking free from the limitations of singular actions. Capitalizing on this publish-and-consume model, we devised our framework, intricately woven around the collaborative nature of nodes.

At the core of our architecture lie two pivotal nodes: the one ingeniously crafted by our team for recognizing signals emitted by individuals and the Turtlesim node, a fundamental component introduced during the IACOS course lectures. Turtlesim node serves as our simulation platform, emulating an autonomous system that eagerly responds to directives, such as maneuvering through a temporarily disrupted street or navigating detours.

Within the team-developed node, we've encapsulated two functionalities into a singular entity. This involves the signal recognition algorithm actively communicating with another node, the publish node. The publish node plays a dual role, receiving insights from the recognition node and, crucially, disseminating this information to the Turtlesim node. The Turtlesim node, in turn, interprets these signals, guiding its actions accordingly. Below, we present an illustrative example of the model meticulously employed by our team to articulate the intricate workings of our system.

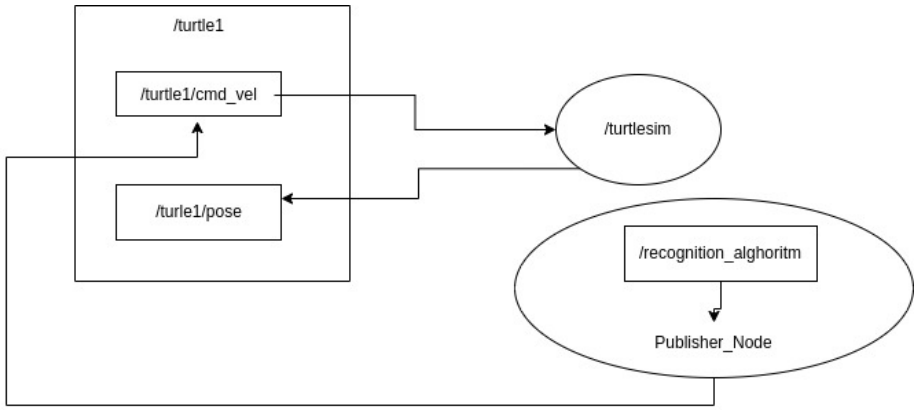

Figure 1: How the nodes communicate.

2.2 Main

The main.py imports turtle_cmd and action_reconition.py and creates two threads, thread consume to send actions and thread producer in witch detect if any known gesture was detected. A queue is used to send the actions from the produce to the consume.

```
from threading import *
import time
import queue
from turtle_cmd import TurtleCmdPublisher, cmd
from action_recognition import GestureRecognition

# ros2 run turtlesim turtlesim_node
def produce(queue):
    gesture = GestureRecognition(queue)
    gesture.detect_video_device()
    gesture.detec_gesture()

def consume(queue, args=None):
    cmd(queue)

def main():
    q=queue.Queue()
    t1=Thread(target=consume, args=(q,))
    t2=Thread(target=produce, args=(q,))
    t1.start()
    t2.start()

if __name__ == '__main__':
      main()
```

Listing 1: Python script main.py

2.3 Mediapipe

The MediaPipe Python framework grants direct access to the core components of the MediaPipe C++ framework such as Timestamp, Packet, and CalculatorGraph, whereas the ready-to-use Python solutions hide the technical details of the framework and simply return the readable model inference results back to the callers.

MediaPipe framework sits on top of the pybind11 library. The C++ core framework is exposed in Python via a C++/Python language binding..

2.3.1 Pose Landmaker

The pose landmarker model tracks 33 body landmark locations, representing the approximate location of the following body parts:

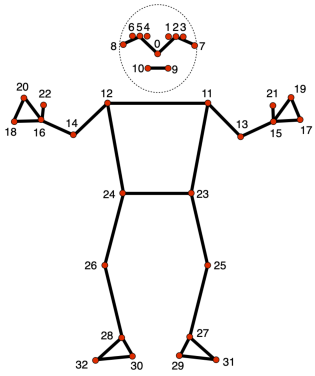

Figure 2: Pose Landmarks.

After collecting the image and get the landmarks is possible to calculate the angles using trigonometry of shoulders, wrist , hands, so on so forth. During early phase of development was decided to use this technique of identify angles and combine them to detect poses and take actions according to it. The following piece of code presents the main method that acquires image, extract landmarks, compute coordinates and angles, identify poses, and send messages via queue to TurtleSim.

```
# Public methods
    def detec_gesture(self):
        ## Setup mediapipe instance
        with mp_holistic.Holistic(min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as holistic:
            while self.__cap.isOpened():
                ret, frame = self.__cap.read()

                # Recolor image to RGB
```

```
                image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                image.flags.writeable = False

                # Make detection
                results = holistic.process(image)

                # Recolor back to BGR
                image.flags.writeable = True
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

                try:
                    # Extract landmarks
                    landmarks = results.pose_landmarks.landmark

                    coordinate = self.__get_coordinates(landmarks)
                    angle = self.__calc_angles(coordinate)
                    self.__identify_geture(angle)
                    self.__print_angles(image, angle, coordinate)
                    if self.__action == "No human detected!":
                        self.__action = None

                except Exception as error:
                    self.__action = "No human detected!"

                self.__setup_image_box(image)
                self.__draw_landmarks(image, results)

                cv2.imshow('ERR - IACOS ', image)

                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

            self.__cap.release()
            cv2.destroyAllWindows()
```

Listing 2: Python script action_reconition.py

2.4 TertleSim

To control the TurtleSim was developed a module to create a node and a callback with a timer. The code of callback is below and basically identify if there is any message in the queue and sends it to the node, if the queue is empty and the last message was not stop, it send forward to the node.

```

def timer_callback2(self):
        vel_msg = Twist()

        if not self.queue.empty():
            self.cmd = str(self.queue.get())
            self.get_logger().info("Command received: %s" % self.
    cmd)

        #Converting from angles to radians
        relative_angle = 90*2*PI/360 # 90 degrees

```

```
12          #Initialize variables
13          vel_msg.linear.x=0.0
14          vel_msg.linear.y=0.0
15          vel_msg.linear.z=0.0
16          vel_msg.angular.x=0.0
17          vel_msg.angular.y=0.0
18          vel_msg.angular.z=0.0
19
20          if self.cmd == 'R' or self.cmd == 'r':
21              vel_msg.angular.z = -abs(relative_angle)
22              self.cmd = 'F'
23          elif self.cmd == 'L' or self.cmd == 'l':
24              vel_msg.angular.z = abs(relative_angle)
25              self.cmd = 'F'
26          elif self.cmd == 'S' or self.cmd == 's':
27              vel_msg.linear.x=0.0
28          else:
29              vel_msg.linear.x=0.5
30
31          self.publisher_.publish(vel_msg)
32          self.get_logger().info('Publishing command:"%s" "%s"' % (
    self.cmd, vel_msg))
```

Listing 3: Python script turtle_cmd.py

3 Limitation of the System

The algorithm adopted for the detection and classification of human movements is not based on machine learning but rather on mathematical formulas. This raises interesting limitations and advantages that will be discussed below.

Most of the changes that we made to the algorithm were based on empirical testing of different sets of parameters with the goal of improving, as much as possible, the detection performance of our system.

However, after some attempts, the parameters that were tuned to fulfill the assignment requirements showed a restrictive level of flexibility when dealing with different people. Even between different elements of our working group, the algorithm was always more likely to identify interactions of some specific morphologies than others. We also noticed that the clothes that were being worn during the testing presented some idiosyncrasies: more bulky outfits were more quickly identified.

Given the context in which this system should be applied - traffic control done by Authorized Traffic Controllers -, the usual changes in working clothes, physical differences or different movements for the same type of command are a great challenge for the system.

To contradict these restrictive outcomes, we tried to find parameters that would give a more flexible classification, but it showed an increase in false positives.

Another limitation that was intrinsic to the solution adopted is the selection criteria used to find the human in the picture if more than one is present in the field of view. The selection criteria are difficult to grasp, and in some cases, the arms of one person are attributed to another.

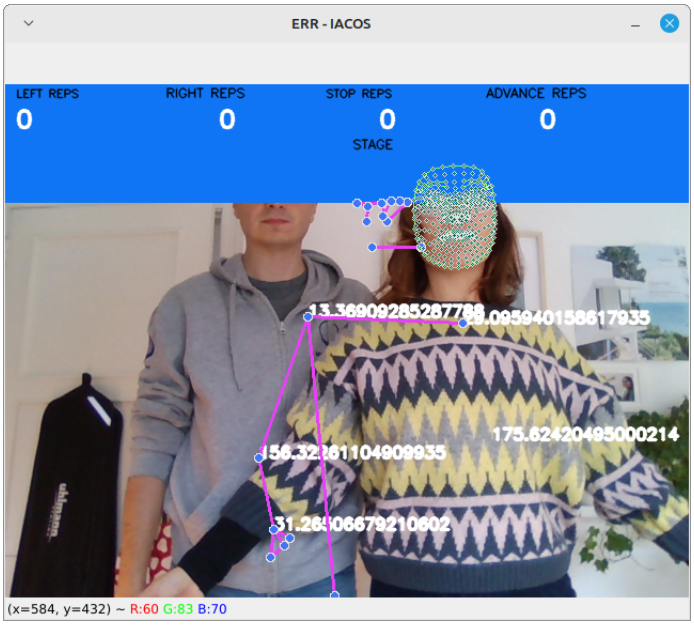

Figure 3: Many people on the screen can create some confusion.

Once more, in the context that we are talking about, being unable to identify the person responsible for traffic control is unacceptable.

The way the authorized traffic controller is framed in the picture also makes evident some other limitations of the solution.

If a person is too close to the camera in such a manner that his head is outside the angle of view, the algorithm, most of the time, doesn't identify him as a human being.

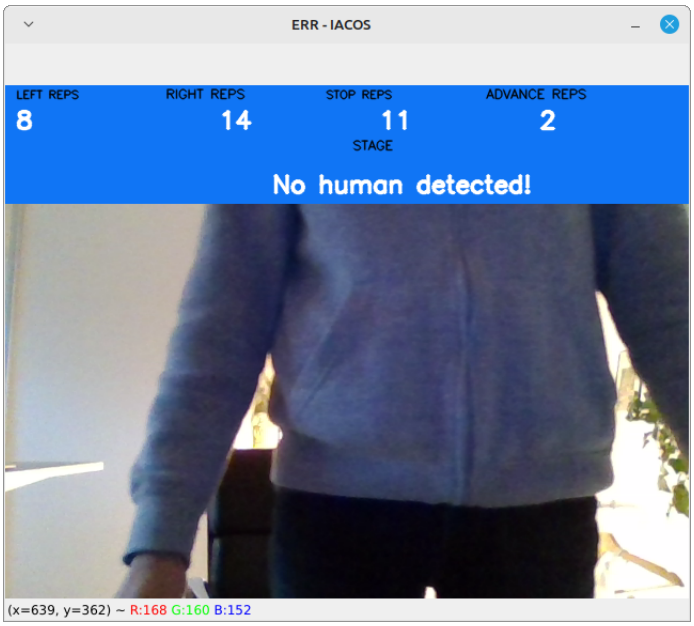

Figure 4: If the person is too close to the camera and only the torso is visible, the algorithm has some difficulty identifying them correctly.

This seems to be attenuated by the memory that the algorithm seems to have. If a person was previously completely inside the angle of view and, afterward, approached the camera and only his torso appeared, usually the system can still identify him as part of a human being.

Luckily, the facial expression doesn't play an important role; the essential elements that must be present are the torso and the full extension of both arms.

For testing purposes, we tried to see if the system would also work if a person had their back turned. The results were positive.

After some more experimentation, we also noticed that the defined movement for each command didn't depend on the spatial direction that the traffic controller was pointing to. For example, if the traffic controller pointed to the right with his left arm, it would be identified as a left-turn command. It seems that only the relation of the arms with the torso is being used to classify movements.

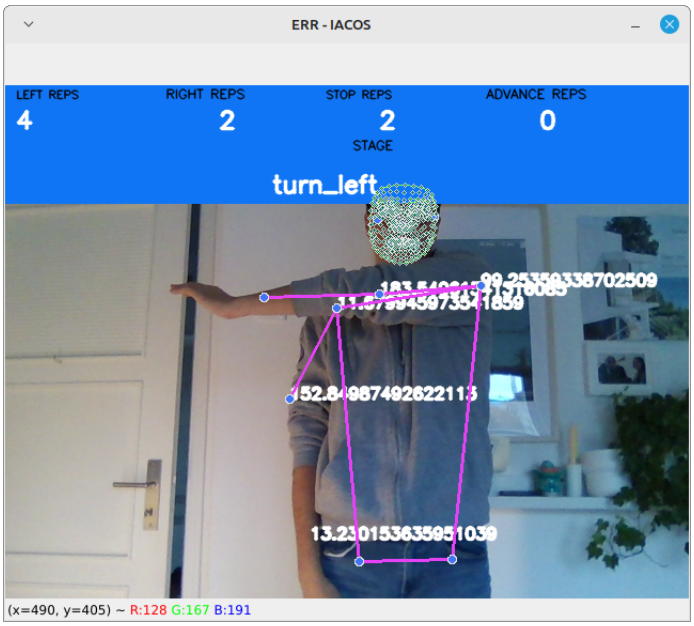

Figure 5: Turn left command when pointing to the right.

Another test confirmed that having something in our hands doesn't affect the outcome. However, bad visibility, like, for example, at nighttime, reduces the performance of the system. This is a problem closely related to the capabilities of the camera itself.

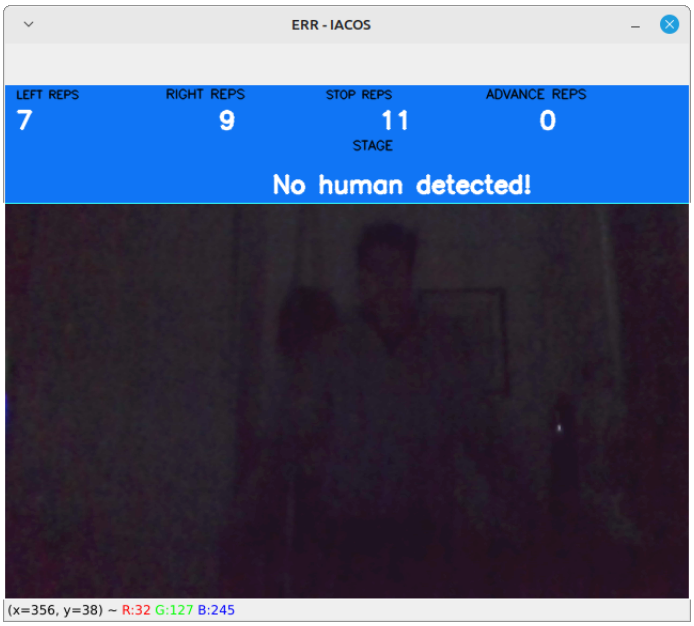

Figure 6: Bad visibility.

As a final remark, the output of this system is binary: either a movement is identified or not. This reduces the possibility of using the output to fuse it with other data from other sensors, improving the performance as a whole.

An obvious advantage when compared with the most common machine learning techniques, like convolutional neural networks (CNN), is that we can understand the why of the output for a given input to the system. Maybe here lies one of the most important challenges of our time: how to integrate safety-critical systems with life-changing technologies like black box systems incapable of justifying themselves and with an ever-increasing level of autonomy.

4 Conclusion

Upon completing the tasks involved in integrating the code for signal recognition with ROS 2, we have come to appreciate the potential of this tool for facilitating communication between disparate components. The modularity of ROS 2 allows each component to be developed independently, with the only requirement being the establishment of a means for seamless communication, achieved through topics or services. This modular approach empowers each team to work autonomously, tailoring their implementation to suit their specific needs. In the end, the singular expected outcome is either a published topic and the node consumes this topic or a service use a request and response.

This experience has provided us with valuable insights into utilizing and developing projects within the ROS 2 framework. We encountered both the advantages and challenges inherent in the framework, gaining a nuanced understanding of how nodes communicate and how the entire system interconnects to achieve more intricate tasks in the most optimal manner possible.

5 Repository

`https://github.com/rafaelcalai/IACOS/tree/main`