Middlesex University London

Report on Human Pose Estimation implemented On Turtlesim

Sela Shapira
M00956836

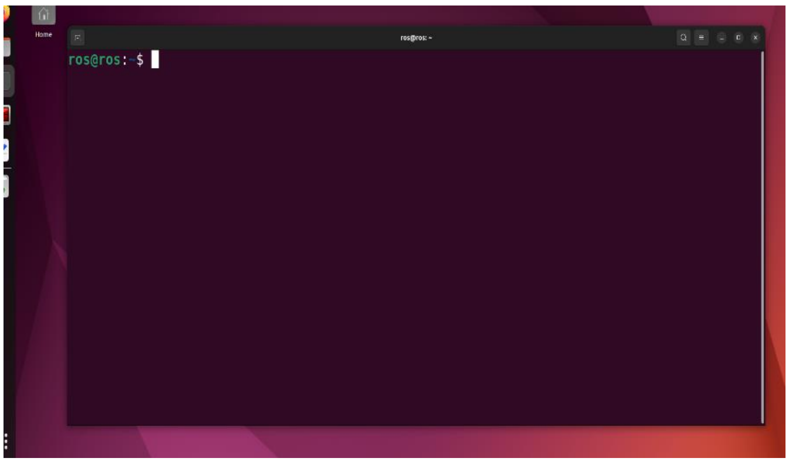
January 2024

Background:

The primary objective of this project is to employ an RGB camera to detect the skeletal structure of a human body. Subsequently, the detected body pose will be integrated into the turtlesim program, allowing the turtle to move in response to specific gestures. For instance, raising the left arm will prompt the turtle to move left, while raising the right arm will result in rightward motion. On the turtlesim screen, additional turtles are introduced, to represent obstacles that the main turtle must navigate around. A target point is designated for the turtle, and its objective is to reach this point based on my gestures. The code for this project will be crafted using Camera vision, AI, and Machine Learning techniques. The steps for achieving the task using these methods are outlined in the following report.

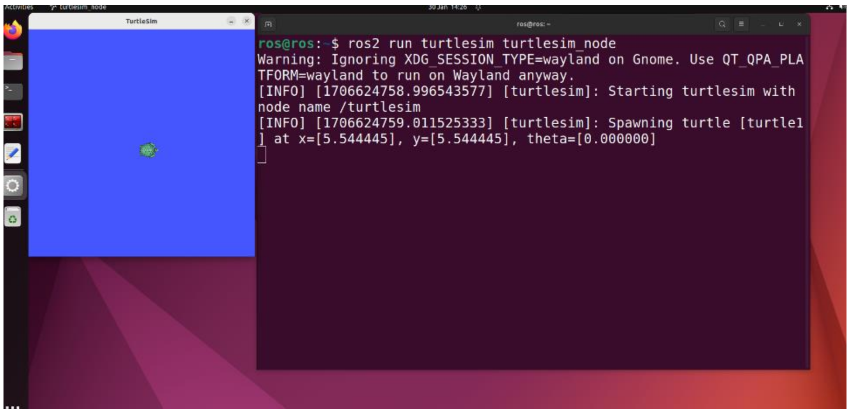
Setting up the work environment:

For this project, I utilized the Turtlesim program, a tool that displays a turtle on a screen, capable of moving in any direction. This program operates within the ROS environment. To access the program and open the Turtlesim screen I followed the following steps.



```
ros@ros:~$
```

Fig 1. Opening the terminal in the ROS environment.

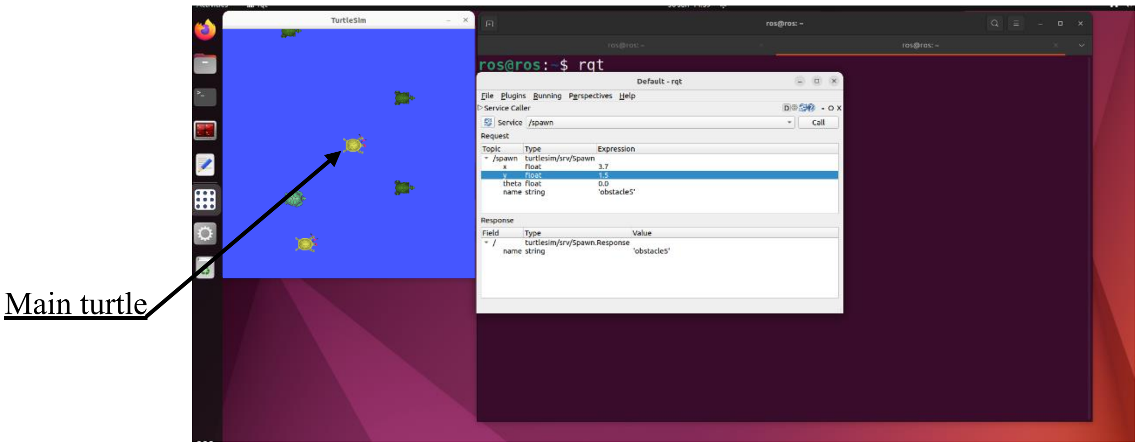


```
ros@ros:~$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLA
TFORM=wayland to run on Wayland anyway.
[INFO] [1706624758.996543577] [turtlesim]: Starting turtlesim with
node name /turtlesim
[INFO] [1706624759.011525333] [turtlesim]: Spawning turtle [turtle1
] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

Fig 2. Running the Following command to open the Turtlesim Screen.

Adding obstacles to the turtlesim screen:

The primary turtle is tasked with reaching a specific target while skilfully avoiding the obstacles along its path. To create these obstacles, I introduced additional turtles onto the screen. I utilised the /spawn command to position these obstacle turtles at specific X and Y coordinates on the screen.



```
ros@ros:~$ rqt
```

Service: /spawn

Request

Topic	Type	Expression
/spawn	turtlesim/srv/Spawn	
x	float	3.7
y	float	1.5
theta	float	0.0
name	string	'obstacle5'

Response

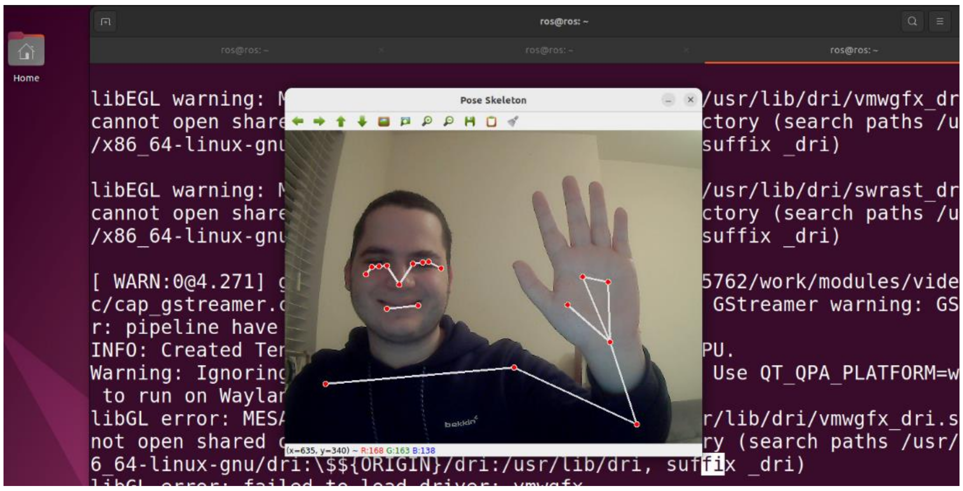
Field	Type	Value
/	turtlesim/srv/Spawn.Response	
name	string	'obstacle5'

Fig 3. Adding additional obstacle turtles on the turtlesim screen.

Running the Code for moving the turtle using Human Pose detection:

1. Teaching the camera to identify the skeleton of my body:

To enable the turtle to respond to my gestures, it was crucial to detect the skeleton of my body, at the beginning of the working progress I implemented skeleton recognition using camera vision. Turtlesim operates within the ROS environment. To run python scripts utilizing libraries for skeleton detection, I installed the necessary libraries. For detecting the skeleton, I specifically employed the 'mediapipe' library, which was installed in the ROS environment using the command: pip install [Library name].

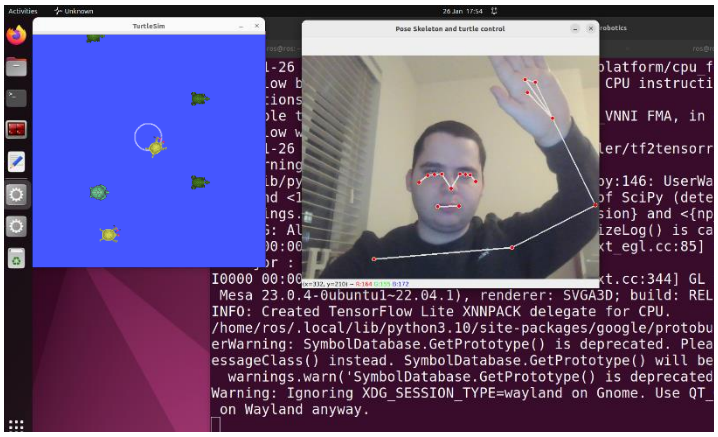


```
libEGL warning: M                          /usr/lib/dri/vmwgfx_dr
cannot open share                          ctory (search paths /u
/x86_64-linux-gnu                          suffix _dri)

libEGL warning: M                          /usr/lib/dri/swrast_dr
cannot open share                          ctory (search paths /u
/x86_64-linux-gnu                          suffix _dri)

[ WARN:0@4.271] g                          5762/work/modules/vide
c/cap_gstreamer.c                           GStreamer warning: GS
r: pipeline have
INFO: Created Ter                          PU.
Warning: Ignoring                           Use QT_QPA_PLATFORM=w
 to run on Waylar
libGL error: MESA                          r/lib/dri/vmwgfx_dri.s
not open shared c                          ry (search paths /usr/
6_64-linux-gnu/dri:\$${ORIGIN}/dri:/usr/lib/dri, suffix _dri)
libGL error: failed to load driver: vmwgfx
```

Fig 4. Detecting the skeleton of my Body.

2. Executing the Code to move the turtle through Human Pose detection:

In the turtlesim program, the turtle's movement is guided by the publication of a 'Twist' messages.
To make the turtle respond according to my body gestures, I must adapt the python script to recognize particular gestures or movements and correlate them with specific commands for the turtle. For instance, lifting my right arm will prompt the turtle to move to the right side, lifting my left arm will make the turtle to move to the left side. While in order to make the turtle move forward and backward, I should raise and lower my arms in accordance, hands up to make the turtle move forward and hands down to make the turtle go backwards.

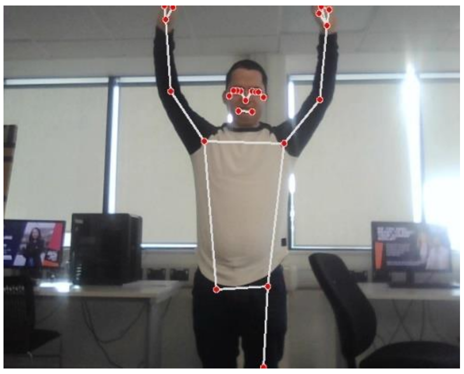


```
I0000 00:00                                        t.cc:344] GL
 Mesa 23.0.4-0ubuntu1~22.04.1), renderer: SVGA3D; build: REL
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
/home/ros/.local/lib/python3.10/site-packages/google/protobu
erWarning: SymbolDatabase.GetPrototype() is deprecated. Plea
essageClass() instead. SymbolDatabase.GetPrototype() will be
  warnings.warn('SymbolDatabase.GetPrototype() is deprecated
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_
 on Wayland anyway.
```

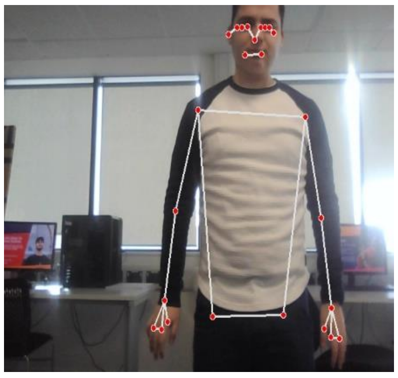
Fig 5. The turtle moves according to my body gestures. (raising my left arm)

3. Integrating a Machine Learning model for gesture predictions into the code.

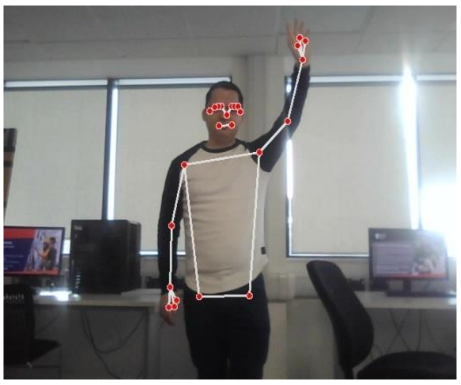
Now, the turtlesim program is responsive to the gestures identified by camera vision. To enhance the program's capability to recognize various gestures without relying on specific positional checks, I incorporated a Machine Learning model into the python script. This model predicts my current gesture, allowing the turtle to move accordingly.
To facilitate this prediction, I developed a dataset.
The dataset, along with the Machine Learning model, empowers the program to detect the gestures and direct the turtle's movement without requiring the camera to discern my exact distance from it.
You can find the Machine Learning training model in a program I created called: pose_recog_model.py.
For the data collection I designed a program called Data_collect_prog.py, this program utilizes a timer, during which, the camera captures pictures of my gestures.
In total, I compiled 1175 pictures for my dataset, organized into 4 folders, each representing a different gesture.
Here is a sample of the pictures I have taken for each gesture:



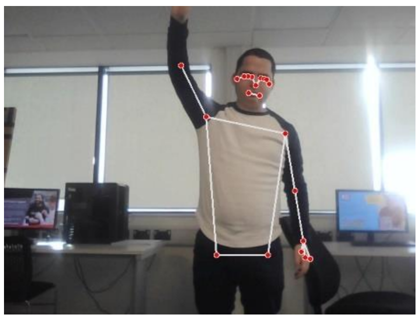
Forward



Backward



Left



Right

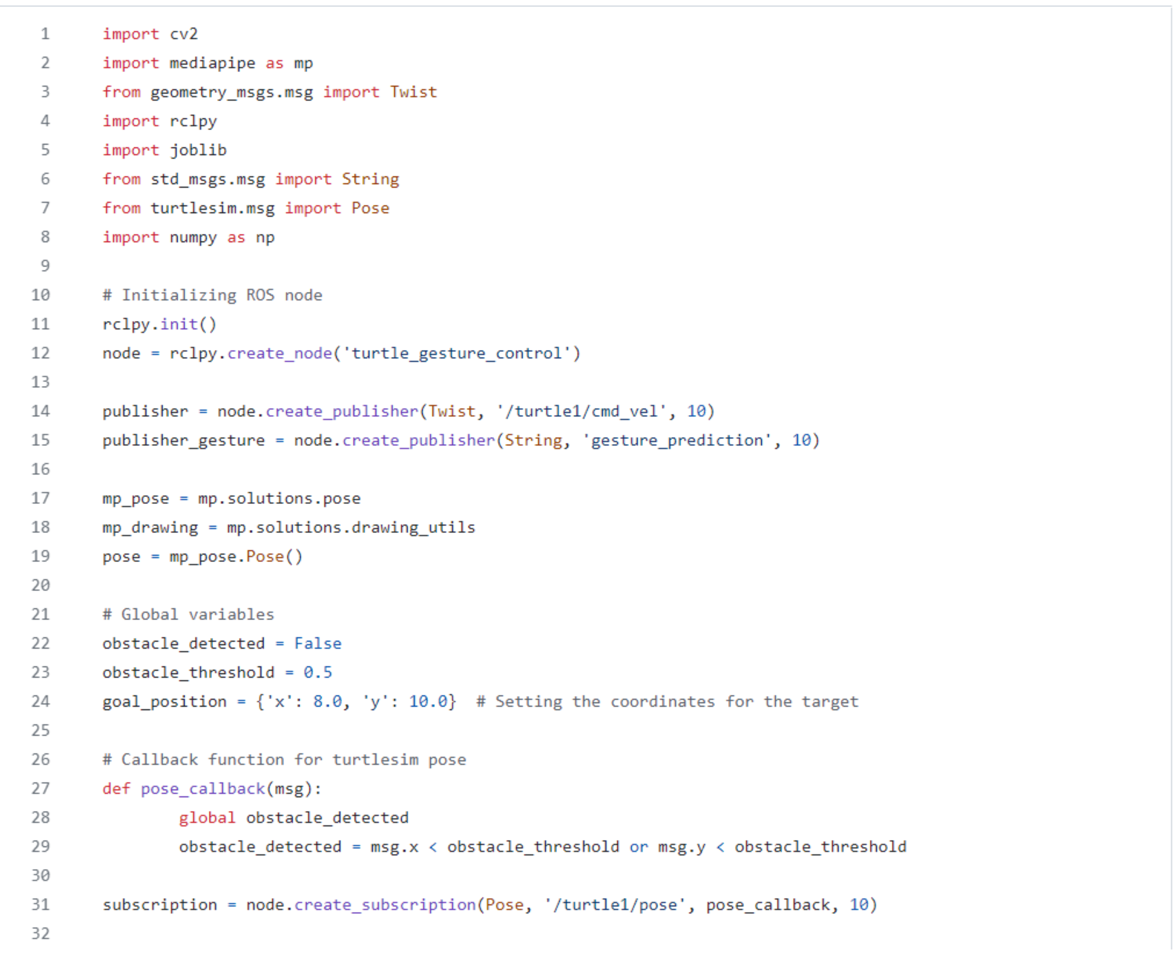
A description of all codes I used in this assignment can be seen in the GitHub repository, moreover a demonstration video can be seen in a separate link in the GitHub as well.

Described in this report is the program I wrote for the assignment called: assignment3.py

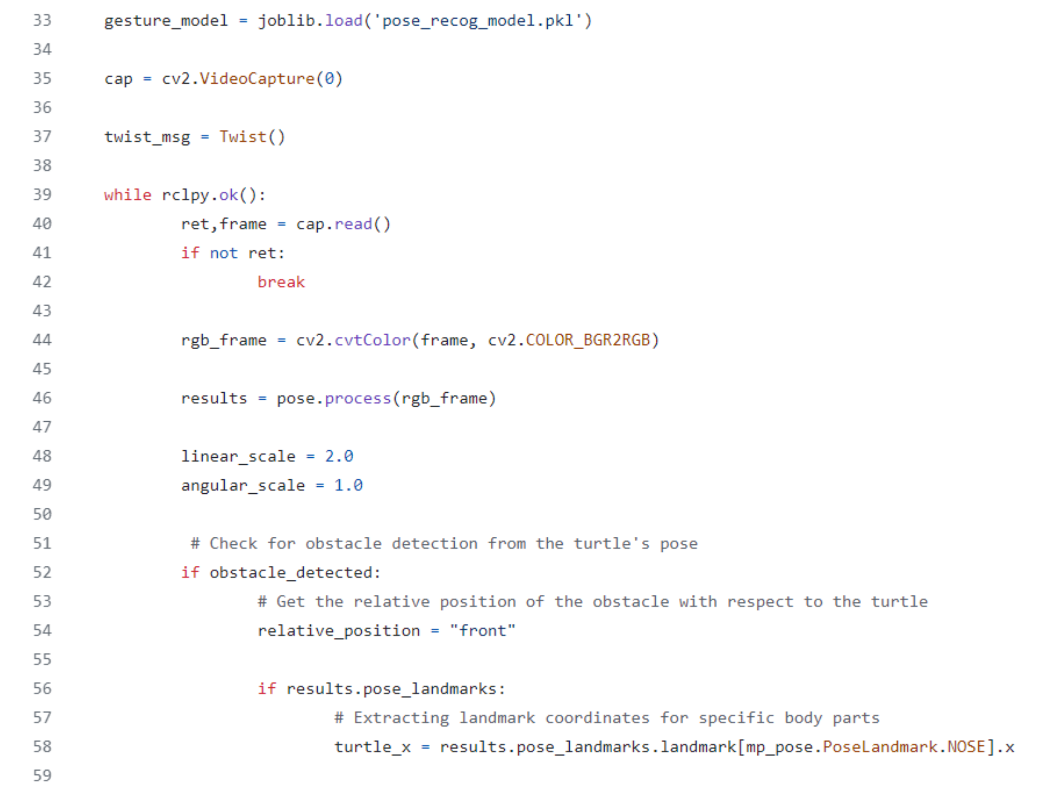
Press the following Link to enter the GitHub repository:

https://github.com/Selashap/Skeleton-pose-estimation---Project-by-Sela-Shapira/tree/main

Program Description: “assignment3.py”



```
import cv2
import mediapipe as mp
from geometry_msgs.msg import Twist
import rclpy
import joblib
from std_msgs.msg import String
from turtlesim.msg import Pose
import numpy as np

# Initializing ROS node
rclpy.init()
node = rclpy.create_node('turtle_gesture_control')

publisher = node.create_publisher(Twist, '/turtle1/cmd_vel', 10)
publisher_gesture = node.create_publisher(String, 'gesture_prediction', 10)

mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
pose = mp_pose.Pose()

# Global variables
obstacle_detected = False
obstacle_threshold = 0.5
goal_position = {'x': 8.0, 'y': 10.0}  # Setting the coordinates for the target

# Callback function for turtlesim pose
def pose_callback(msg):
        global obstacle_detected
        obstacle_detected = msg.x < obstacle_threshold or msg.y < obstacle_threshold

subscription = node.create_subscription(Pose, '/turtle1/pose', pose_callback, 10)

```



```
gesture_model = joblib.load('pose_recog_model.pkl')

cap = cv2.VideoCapture(0)

twist_msg = Twist()

while rclpy.ok():
        ret,frame = cap.read()
        if not ret:
                break

        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        results = pose.process(rgb_frame)

        linear_scale = 2.0
        angular_scale = 1.0

         # Check for obstacle detection from the turtle's pose
        if obstacle_detected:
                # Get the relative position of the obstacle with respect to the turtle
                relative_position = "front"

                if results.pose_landmarks:
                        # Extracting landmark coordinates for specific body parts
                        turtle_x = results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].x

```

```python
                # Assuming that the obstacle is directly in front of the turtle
                if turtle_x < 0.5:
                    relative_position = "right"
                else:
                    relative_position = "left"

            # Adjust turning direction based on obstacle position
            if relative_position == "front":
                twist_msg.linear.x = 0.0
                twist_msg.angular.z = 1.0  # Turn right
            elif relative_position == "right":
                twist_msg.linear.x = 0.0
                twist_msg.angular.z = -1.0  # Turn left
            elif relative_position == "left":
                twist_msg.linear.x = 0.0
                twist_msg.angular.z = 1.0  # Turn right

        else:
            # If no obstacle is detected,continuing with the gestures recognition
            if results.pose_landmarks:
                # Extracting landmark coordinates for specific body parts
                left_shoulder_y = results.pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_SHOULDER].y
                right_shoulder_y = results.pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_SHOULDER].y

                left_hand_y = results.pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_WRIST].y
                right_hand_y = results.pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_WRIST].y

                # Thresholds for gesture recognition
                threshold_up = 0.5
                threshold_down = 0.5

                # Gesture recognition logic (for the turtle movement)
                forward_gesture = left_shoulder_y < threshold_up and right_shoulder_y < threshold_up
                backward_gesture = left_shoulder_y > threshold_down and right_shoulder_y > threshold_down
                right_gesture = left_shoulder_y < threshold_up and right_hand_y < threshold_up
                left_gesture = left_hand_y < threshold_up and right_shoulder_y < threshold_up

                twist_msg = Twist()

                twist_msg.linear.x = float(linear_scale) if forward_gesture else (-float(linear_scale) if backward_gesture else 0.0)
                twist_msg.angular.z = float(angular_scale) if right_gesture else (-float(angular_scale) if left_gesture else 0.0)

                # "Go to goal" behavior
                distance_to_goal = np.sqrt((results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].x - goal_position['x'])**2
                                   +(results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].y - goal_position['y'])**2)

                angle_to_goal = np.arctan2(goal_position['y'] - results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].y,goal_position['x'] -
                                                    results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].x)

                if distance_to_goal < 0.5:
                    twist_msg.linear.x = linear_scale
                    twist_msg.angular.z = angular_scale * (angle_to_goal - np.arctan2(results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].y- goal_position['y'],
                                                    results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].x - goal_position['x']))


                # Publishing the Twist message to control the turtle
                publisher.publish(twist_msg)

                resized_frame = cv2.resize(rgb_frame, (160, 120))
                resized_frame = resized_frame / 255.0

                predictions = gesture_model.predict(resized_frame.reshape(1, 160, 120, 3))
                predicted_gesture_index = np.argmax(predictions)
                #predicted_gesture_index = tf.argmax(predictions, axis=1)[0].numpy()
                gesture_names = ["backward", "forward", "left", "right"]
                predicted_gesture = gesture_names[predicted_gesture_index]
                publisher_gesture.publish(String(data=predicted_gesture))

                mp_drawing.draw_landmarks(frame, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

        cv2.imshow('Pose Skeleton and turtle control', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    rclpy.shutdown()
```

This is the core program for this assignment. In this code, I loaded a machine learning model, named “pose_recog_model.pkl”. This model enables me to translate my gestures into turtle motion. To navigate around obstacles, I implemented a function that calculates distances based on the turtle’s nose directing it to move left or right depending on the detected obstacle. To guide the turtle towards a specific goal I incorporated the “go_to_goal” function, this function involves various calculations based on the turtle’s current location and orientation relative to the target.