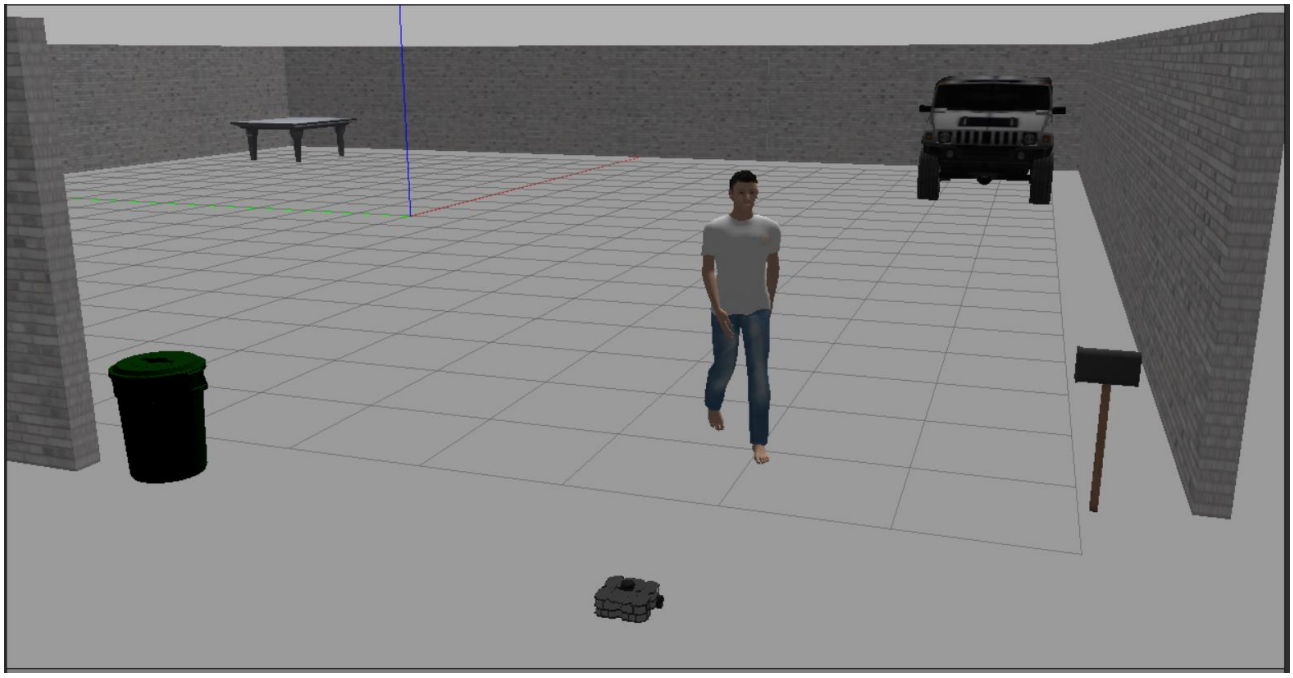
UNIVERSITÀDELLACALABRIA

Dipartimento di INFORMATICA , MODELLISTICA , ELETTRONICA e SISTEMISTICA

Master In Robotics and Automation

Person Tracking System using Robot Turtlebot3

Finale Project of Mechatronic



Professor:

Prof. Marco Lupia

Student:

Alexis Marino-Salguero

(232601)

Academic year 2022/2023

1. Introduction

Person tracking plays a crucial role in various robotics applications, such as service robotics, security, environment monitoring, entertainment. This project presents a person tracking system implemented on the TurtleBot3 robot using the ROS (Robot Operating System) framework. The goal is to enable the robot to autonomously track people in its environment, utilizing computer vision techniques and robotic control.

Benefits of the Project:

The development of this person tracking system with the TurtleBot3 robot and ROS offers several significant benefits. Firstly, it empowers the robot to autonomously interact with people, enhancing the user experience in service robotics applications. Additionally, person tracking has practical applications in areas such as security and environment monitoring, enabling the robot to function as an intelligent surveillance system.

Moreover, the integration of computer vision techniques and robotic control in this project contributes to the advancement of research in the field of autonomous robotics. The combination of visual detection and tracking algorithms, along with proportional control, establishes a solid foundation for future developments in related areas such as autonomous navigation and human-robot interaction.

In summary, the person tracking system implemented on the TurtleBot3 robot and ROS offers significant benefits in terms of autonomous interaction, security, and the advancement of robotics research. This project presents the results, methodology, and insights gained from the project, aiming to contribute to the growth and development of this exciting field.

Project Description:

The project encompasses the development of a person tracking system composed of two main nodes: the person move node and the person detection and tracking node.

The person detection node employs an OpenCV cascade classifier to detect people in the images captured by the TurtleBot3's camera. The detection information is published on a ROS topic, enabling other nodes in the system to access it.

The person move node is used to move a person to have a dynamic environment. The detection and tracking node utilizes visual tracking techniques to estimate the position of the detected person as they move within the robot's field of view. A proportional control algorithm is implemented to adjust the robot's angular velocity based on the tracking error, ensuring smooth and accurate tracking.

The steps followed during the development of the project are:

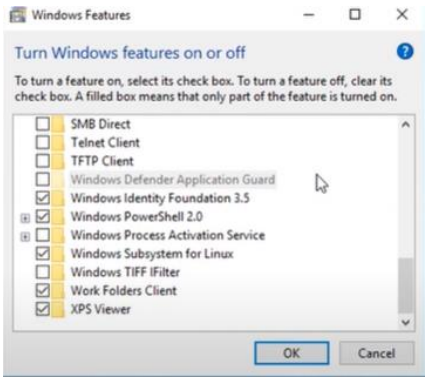
- Problem Identification
- Requirements identification
- Software and Hardware Setup
- Person_move Node
- Person detection_and_tracking Node
- Control Algorithm Implementation
- Integration and Testing

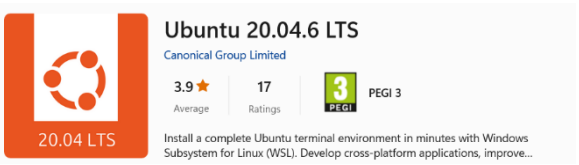
2. Setting up the work environment

As we know ROS works under the Linux operating system. For this reason, it is mandatory to install this system on our computer. This can be achieved by installing a virtual machine like VirtualBox or doing a double boot with a partition for running Ubuntu. However, in this project, it is used WSL which is the Windows Subsystem for Linux. This is a compatibility layer developed by Microsoft to run Linux executables natively on Windows 11.

Steps to integrate WSL2 in the Windows system.

Open ***Turn Windows features on or off*** and Activate Windows Subsystem for Linux





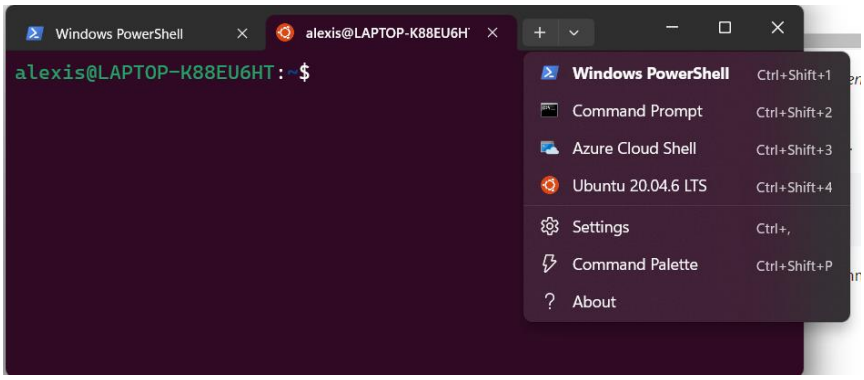
In the Microsoft store download and install the Linux distribution needed for the version of ROS in this case Ubuntu 20.04. Then, set WSL2 as default version “This is important to install ROS via WSL2”. Open PowerShell and use the following command:

>> wsl --set-default-version 2
>> wsl -l -v

```
  NAME            STATE           VERSION
* Ubuntu-20.04    Stopped         2
```

This outcome means that the previous steps were successful.

Now using the application Terminal, it is possible to choose the operating system to work.



Instal ROS Noetic on WSL2, For Ubuntu 20.04, it is possible using the following commands in.

>> sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"> / etc/apt/sources.list.d/ros-latest.list'

>> curl -sSL 'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0xC1CF6E31E6BADE8868B172B4F42ED6FBAB17C654' | sudo apt-key add -

>> sudo apt update
>> sudo apt install -y ros-noetic-desktop python3-rosdep
>> sudo rosdep init
>> rosdep update

Use following commands to *source ROS Noetic automatically* forever bash session.

>> echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
>> source ~/.bashrc

Setting up ROS for Graphical Output with WSL2 (GUI)

To run applications with graphical output, it is needed to install an X Server on Windows. For this project was used VcXsrv. After the installation, it is needed to *configure WSL2* adding to the *.bashrc* the following command:

>> export DISPLAY=$(awk '/nameserver / {print $2; exit}' /etc/resolv.conf 2>/dev/null):0

Finally, launch VcXsrv from the start menu and applied the following settings:

- Native OpenGL needs to be unchecked. Otherwise, applications such as Rviz do not run as expected.
- Disable access control needs to be checked. Otherwise, applications within WSL cannot access the x server.
- When prompted by the Windows Firewall make sure to enable access for both private and public networks.

Then the ROS is ready to run into WSL2 and visual interface is set up to work with applications like gazebo.

3. Implementation

3.1. ROS configurations

Create a workspace and Set Python 3 as the default language for the ROS package build system catkin and run the first build using the command.

>> mkdir -p ros_ws/src
>> cd ros_ws/
>> catkin_make -DPYTHON_EXECUTABLE=/usr/bin/python3

Creation a package for the project with the necessary dependencies and download the packages for the robot turtlebot3.

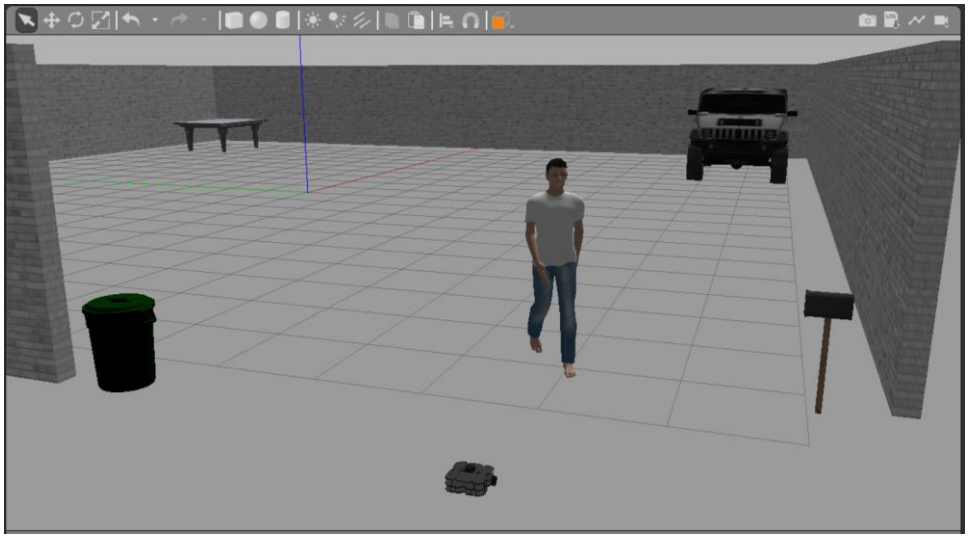
>> cd ~/ros_ws/src
>> catkin_create_pkg seguimiento_perdonas rospy
>> git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
>> git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
>> git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
>> cd ..
>> catkin_make

Now, the ROS environment is set up and the necessary packages are created and downloaded it is possible to create the necessary nodes for the tracking person project.

3.2. Set up the world environment in Gazebo.

To implement people tracking using the Turtlebot3 robot. It was designed an environment in Gazebo, where a person can move through different points and the robot is able to follow him.

First, it is imported the objects into the turtlebot3 empty world and designed an environment according to the user as can be seen in the figure.



In order to move the person, the following node was designed.

3.3. Move person node.

The first node created is the one that allows moving the person across the environment for this goal is considering the service /gazebo/set_model_state and creating a publisher node that publicizes the state of the person in a continuous way.

First, the necessary dependencies for the publication in gazebo are imported which is *from gazebo_msgs.msg import ModelState* and *from geometry_msgs.msg import Pose*. Then, two functions are created, the first to interpolate the points of the trajectory so that the movement is fluid and the second is used so that the person always maintains the desired orientation.

```
def interpolate_points(p1, p2, t):
    x = p1[0] + (p2[0] - p1[0]) * t
    y = p1[1] + (p2[1] - p1[1]) * t
    return x, y

def calculate_orientation(p1, p2):
    dx = p2[0] - p1[0]
    dy = p2[1] - p1[1]
    yaw = atan2(dy, dx)
    return yaw
```

The next following lines of code are used to start the move_person node and indicate that it will be a publisher node with a publication frequency of 15 hertz. '/gazebo/set_model_state' is the name of the topic to which the publisher will send messages, the type of message is ModelState and dt parameter indicates the time increment for the interpolation. The trajectory indicates the points through which the person will move. These points were chosen arbitrarily to generate a trajectory without sudden changes.

```
def move_person():
    rospy.init_node('move_person', anonymous=True)
    pub = rospy.Publisher('/gazebo/set_model_state', ModelState, queue_size=10)
    rate = rospy.Rate(15)
    dt = 0.005

    trajectory = [(-9, -9), (-8, -9),(-6, -8), (-4, -8), (-1, -6), (1, -6),
```

The internal structure of the node runs until a kill sequence is called. Inside the node an object called person_state is created based on the ModelState class and the name of the person to move is assigned. Two points are extracted from the trajectory to perform the interpolation and calculate the orientation of the person. Finally, the person_state message is published.

```
    while not rospy.is_shutdown():
        person_state = ModelState()
        person_state.model_name = 'person_walking'

        p1 = trajectory[index]
        p2 = trajectory[(index + 1) % len(trajectory)]

        x, y = interpolate_points(p1, p2, t)

        person_state.pose = Pose()
        person_state.pose.position.x = x
        person_state.pose.position.y = y
        person_state.pose.position.z = 0.0

        yaw = calculate_orientation(p1, p2)
        desired_yaw = yaw + 1.5708
        person_state.pose.orientation.z = sin(desired_yaw/ 2)
        person_state.pose.orientation.w = cos(desired_yaw/ 2)

        pub.publish(person_state)
```

3.4. Detection and tracking node.

The following code creates a ROS node that uses OpenCV to detect and track a person in real time using a camera. It uses Haar Cascade classifiers for person detection and issues speed commands to the turtlebot3 robot for person tracking.

- *cv2* - The OpenCV Python module, used to process images and perform person detection.
- *Image from sensor_msgs.msg* - it is a ROS message that represents an image.
- *CvBridge CvBridgeError from cv_bridge* - Used to convert between ROS images and OpenCV arrays.
- *Twist from geometry_msgs.msg* - it is a ROS message representing linear and angular velocity.

Then is Defined a class named **DetectionAndTracking**. This class encapsulates the functionality for detecting and tracking a person.

In the class constructor **__init__,** the ROS node is initialized, and the necessary publishers and subscribers are created. The Haar Cascade XML file for lower body detection is loaded, and variables for tracking and counting the number of frames without a detected person are initialized.

```
class DetectionAndTracking:
    def __init__(self):
        rospy.init_node('detection_and_tracking', anonymous=True)
        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber('/camera/rgb/image_raw', Image, self.image_callback)
        self.cmd_vel_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)

        self.no_person_counter = 0
        self.max_no_person_count = 5

        self.cascade_path = '/home/alexis/ros_ws/src/seguimiento_personas/scripts/haarcascade_lowerbody.xml'
        try:
            self.cascade = cv2.CascadeClassifier(self.cascade_path)
        except cv2.error as e:
            rospy.logerr('Error loading cascade: {}'.format(e))
```

The **image_callback** function is the callback for the image subscriber. It processes the received image, detects a person using the Haar Cascade classifier, and performs person tracking based on the detected person's position. If no person is detected for a certain number of frames, the robot's movement is stopped.

```
def image_callback(self, msg):
    try:
        self.cv_image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
    except CvBridgeError as e:
        rospy.logerr(e)
        return

    gray = cv2.cvtColor(self.cv_image, cv2.COLOR_BGR2GRAY)
    detections = self.cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    if len(detections) > 0:
        self.no_person_counter = 0
        if self.target_person is None:
            self.target_person = detections[0]
        else:
            min_distance = float('inf')
            closest_person = None

            for detection in detections:
                x, y, w, h = detection
                center_x = x + w // 2
                center_y = y + h // 2
                distance = abs(center_x - self.cv_image.shape[1] // 2) + abs(center_y - self.cv_image.shape[0] // 2)

                if distance < min_distance:
                    min_distance = distance
                    closest_person = detection

            self.target_person = closest_person

        self.track_person()
    else:
        self.no_person_counter += 1
        if self.no_person_counter >= self.max_no_person_count:
            self.stop_robot()
```

The **stop_robot** function publishes a zero linear velocity to stop the robot and a small angular velocity to rotate in its own center in this way the robot can find the person. Finally, it is created a **track_person** function that calculates the error between the person's position and the center of the image, then based in this error the angular and linear velocity of the robot is adjusted from the error_x and error_y respectively

```
def stop_robot(self):
    twist_msg = Twist()
    twist_msg.linear.x = 0.0
    twist_msg.angular.z = -0.05
    self.cmd_vel_pub.publish(twist_msg)

def track_person(self):
    x, y, w, h = self.target_person
    target_center_x = x + w // 2
    target_center_y = y + h // 2
    error_x = target_center_x - self.cv_image.shape[1] // 2
    rospy.loginfo(f'error x {error_x}')
    error_y = target_center_y - self.cv_image.shape[0] // 2
    twist_msg = Twist()
    twist_msg.linear.x = 0.2 *error_y
    twist_msg.angular.z = -0.001 * error_x
    rospy.loginfo(f'ang vel {twist_msg.angular.z}')
    self.cmd_vel_pub.publish(twist_msg)
```

4. Conclusions

In this project, it has been developed a person tracking system using computer vision techniques. The system consists of two main components: a person move node and a person detection and tracking node. The person detection node utilizes the Haar cascade classifier to detect the lower part of the people in the input image. Once a person is detected, the turtlebot3 tracks the person's movement by the error generated in the frame of the detected person and the center of the image.

The Haar cascade classifier has proven to be effective in detecting the lower part of the people in various environments. The tracking algorithm based on proportional control has provided smooth and accurate tracking of the target person.

This project has demonstrated the potential of computer vision techniques in enabling real-time person tracking applications. By combining person detection and tracking, we can effectively monitor and follow individuals as they move within the robot's field of view. This has numerous practical applications, such as surveillance, human-robot interaction, and social robotics.

Furthermore, the project highlights the importance of parameter tuning and system optimization. By adjusting parameters like scale factor, minimum neighbors, and minimum size, we can fine-tune the detection and tracking performance to suit specific scenarios. This flexibility allows the system to adapt to different environments and varying conditions.

Finally, the developed person tracking system provides a solid foundation for further research and application development in the field of person tracking. By leveraging computer vision techniques and intelligent algorithms, we can enhance the capabilities of robots and enable them to interact with humans in a more intuitive and responsive manner.