UNIVERSITY AT ALBANY
State University of New York

College of Engineering and Applied Sciences
Computer Science Department

Follow Feature to a Robot using Boosting Tracker and a Simple PID Controller

(Master's Project)

S R K N S Bharadwaj Nidamarthy

Advisor: Prof. Chinwe Ekenna

Abstract

The objective of the project is to implement a follow feature on a robot, using opencv tracker and this system involves using the Boosting algorithm to track an object or person's movements in subsequent frames. In this system, we used the 3d images to generate a laser scan and to determine the distance between the robot and the tracked object. Two PID controllers are used to control the linear and angular velocities of the robot to maintain the object at the center of the image frame. The distance between the robot and the object fed back into the linear velocity PID controller. The angular velocity PID controller adjusts the direction of the robot to maintain a desired orientation with respect to the object being followed. This allows the robot to smoothly track and follow the specific object. Overall, this follow feature can enhance the robot's mobility and make it more useful in applications such as surveillance where following a suspicious person in a secure location. This can help improve security and safety in public places as well as in terms of personal assistance for people with disabilities.

Acknowledgements

I would like to thank my advisor and mentor Prof. Chinwe Ekenna for giving me an opportunity to take up this project under his guidance and providing immense help in every way possible. I would also like to thank Sourav Dutta whose research work helped me immensely in understanding all the concepts. Last, but not the least, I would like to thank my friends and family for supporting me and strongly encouraging me in all of my dreams, especially academically.

1. Introduction

The "follow" feature is an important application of mobile robots that allows them to track and follow a moving target, such as a person or a vehicle. This feature can be used in a wide range of fields, from security and surveillance to logistics and transportation.The follow feature can be implemented using different techniques, such as computer vision, machine learning, and sensor fusion. In computer vision-based approaches, the robot uses cameras or other vision sensors to track the target, while in machine learning-based approaches, the robot learns from previous data to predict the target's movements. Sensor fusion-based approaches combine data from different sensors, such as cameras and LIDAR, to track the target more accurately.

The follow feature can be divided into two categories: static and dynamic. In static following, the target moves at a constant speed and direction, while in dynamic following, the target's speed and direction can change unpredictably. Dynamic following is more challenging and requires more advanced algorithms and control strategies. One of the most common techniques used to implement the follow feature is proportional-integral-derivative (PID) control. PID control is a feedback control system that calculates the error between the desired and actual position of the robot and adjusts its movements accordingly. The proportional term adjusts the robot's movements based on the current error, the integral term reduces steady-state errors, and the derivative term improves the response time and stability of the system.

To implement the follow feature using PID control, the robot needs to detect the target and calculate its position and velocity. This can be done using sensors such as cameras, LIDAR, or GPS. Once the target's position and velocity are determined, the robot can calculate the error between its current position and the target's position and adjust its movements accordingly. The PID controller continuously updates its output based on the error signal, allowing the robot to follow the target accurately and smoothly. Obstacle avoidance is another important component of person following using mobile robots. In environments with obstacles, the robot must be able to navigate around them while still following the person. This can be done using techniques such as potential fields or occupancy grids. Potential fields involve defining attractive and repulsive forces around the obstacles and using them to guide the robot's movements. Occupancy grids involve dividing the environment into a grid and using it to represent the location of obstacles. The robot can then plan a path that avoids the occupied cells in the grid.

Continuously updating the trajectory is another important component of person following using mobile robots. As the person moves, the trajectory must be continuously updated to ensure that the robot stays on track and follows the person. This requires using real-time data from the sensors and tracking algorithms to update the trajectory.

Person following using mobile robots has various potential applications. In security, it can be used to track and follow a suspicious person in a building or other secure area. In healthcare, it can be used to monitor patients or assist healthcare workers in tasks such as transporting equipment or supplies. In entertainment, it can be used to provide a robotic companion that can follow a person around a theme park or other attraction. In retail, it can be used to guide customers to products or provide assistance in a store.

The development of person following using mobile robots is an active area of research, and new algorithms and techniques are being developed to improve the accuracy and robustness of the system. For example, machine learning techniques such as deep learning can be used to improve object detection and tracking. Reinforcement learning can be used to improve the robot's control and decision-making abilities.

In conclusion, person following using mobile robots is a challenging and exciting application of robotics that has various potential applications in different fields. It requires the integration of multiple sensors, algorithms, and control techniques to develop a system that can accurately and robustly track and follow a person while avoiding obstacles. As technology continues to advance, we can expect to see further developments in person following using mobile robots, opening up new possibilities for its use in different fields.

2. Description of relevant technologies and techniques

2.1 Robot Operating System(ROS)

ROS (Robot Operating System) is a popular middleware framework used in robotics research and development. It provides a collection of software libraries and tools that simplify the development of complex robotics applications. The main goal of ROS is to provide a flexible and modular platform for developing robotics software. ROS consists of a collection of nodes, which are small software components that perform specific tasks. These nodes communicate with each other through a messaging system called ROS topics, which allows for easy sharing of data such as sensor readings and control signals. One of the key benefits of ROS is its modular architecture. Developers can create new nodes to perform specific tasks, or they can use existing nodes to build more complex applications. This modularity also makes it easy to swap out different components, such as sensors or actuators, as needed. Another benefit of ROS is its large community of developers. There are many libraries and tools available that have been developed by the community, which can save developers time and effort in building their applications. Additionally, ROS has active forums and mailing lists, where developers can ask questions and get help from others in the community.
ROS also includes a wide range of tools for visualization, simulation, and debugging. These tools can help developers test and debug their applications before deploying them on actual hardware. For example, RViz is a 3D visualization tool that allows developers to see the robot's sensors and its environment, while Gazebo is a robot simulator that can be used to test the robot's behavior in different scenarios.

ROS has been used in many robotics applications, from autonomous vehicles to industrial robots. It is also popular in research labs, where it is used to prototype and test new robotics algorithms and technologies. One potential downside of ROS is its steep learning curve. Because it is a complex system with many tools and libraries, it can take some time to learn how to use it effectively. Additionally, ROS can be resource-intensive, which may be a consideration for applications with limited processing power or memory. Overall, ROS is a powerful and flexible platform for developing robotics applications. Its modular architecture, extensive library of software components, and active community support make it a popular choice for robotics research and development.

In ROS (Robot Operating System), a rostopic is a communication channel over which nodes can publish or subscribe to messages. rostopics are used to facilitate communication between different nodes in a ROS network. To use a rostopic, a node can either publish messages to a rostopic or subscribe to messages from a rostopic. When a node publishes a message to a rostopic, other nodes that are subscribed to that topic will receive the message and can then process the data contained within it. When a node subscribes to a rostopic, it will receive messages that are published to that topic by other nodes in the network. ROS topics are typically named using a forward-slash-separated string, such as "/odom" or “/scan".

The naming convention for rostopics is generally based on the type of message being sent or received, and the topic name should be descriptive enough to indicate what type of data is being transmitted. To view the list of available rostopics in a running ROS system, you can use the "rostopic list" command in a terminal window. This will display a list of all currently active rostopics in the system.

Overall, rostopics are a fundamental component of ROS communication, allowing nodes in the network to exchange messages and share data about the robot's state, sensor readings, and other relevant information.

2.2 OpenCV Boosting Tracker

The OpenCV Boosting Tracker can be used to track objects in videos by continuously updating the position of a bounding box around the object being tracked. The bounding box represents the region in the video where the object is located, and its position is updated by the Boosting Tracker algorithm. The Boosting Tracker algorithm works by learning a set of weak classifiers that are combined into a strong classifier. The weak classifiers are used to classify the features in the image that are inside the bounding box. The strong classifier is then used to predict the location of the object in the next frame.

When using the Boosting Tracker algorithm in OpenCV, you can initialize the bounding box by specifying its location and size in the first frame of the video. Then, the algorithm updates the position of the bounding box in each subsequent frame. The updated bounding box represents the new position of the object being tracked. The bounding box can be represented in OpenCV as a rectangle with four values: the x-coordinate of the top-left corner of the rectangle, the y-coordinate of the top-left corner, the width of the rectangle, and the height of the rectangle.

Overall, the Boosting Tracker algorithm in OpenCV is a powerful tool for tracking objects in videos, and the bounding box is an essential component of the algorithm that helps to determine the location of the object being tracked.

2.3 PID Controller

PID controller is a feedback control loop that continuously adjusts the output based on the error between a desired setpoint and the current process variable. In the context of controlling the movement of a robot, the setpoint refers to the desired position or velocity, and the process variable is the actual position or velocity of the robot. The PID controller has three components: proportional (P), integral (I), and derivative (D) terms.

The proportional term is proportional to the error between the setpoint and the process variable. It generates an output that is proportional to the magnitude of the error, which causes the robot to move faster or slower depending on the magnitude of the error. The P term can be used to control the linear velocity of the robot, adjusting it proportionally to the distance between the robot and the target object.

The integral term accumulates the error over time, which helps to eliminate steady-state errors. It generates an output that is proportional to the integral of the error, which causes the robot to move faster or slower depending on how long the error persists. The I term can also be used to control the linear velocity of the robot, adjusting it to compensate for any errors in its previous movements.

The derivative term is proportional to the rate of change of the error, which helps to eliminate overshoot and oscillations in the system. It generates an output that is proportional to the rate of change of the error, which causes the robot to move faster or slower depending on how quickly the error is changing. The D term is typically used to control the angular velocity of the robot, adjusting it to compensate for any changes in the direction of the target object.

In summary, the linear velocity of the robot can be controlled using the proportional and integral terms of the PID controller, while the angular velocity can be controlled using the derivative term. The specific values of the P, I, and D terms should be tuned based on the dynamics of the robot and the target object, in order to achieve optimal performance.

2.3.1 Algorithm for PID controller

We used simple PID Controller in this system to control and movement of the robot

1.Initialize the stored data, e_prev to 0, t_prev to -100, and I to 0.

2. Set the initial control MV to MV_bar.

While True, do the following:
a. Yield MV and wait for the new values of t, PV, and SP.
b. Calculate the error e as the difference between the setpoint (SP) and the process variable (PV).
c. Calculate the proportional term, P as the product of the proportional gain (Kp) and the error (e).
d. Calculate the integral term, I as the sum of the previous integral value (I) and the product of the integral gain (Ki), the error (e), and the time difference (t - t_prev).
e. Calculate the derivative term, D as the product of the derivative gain (Kd), the difference between the current and previous error (e - e_prev), and the inverse of the time difference (1/(t - t_prev)).

f. Calculate the manipulated variable (MV) as the sum of MV_bar, P, I, and D.
g. Update the stored data for the next iteration by setting e_prev to e and t_prev to t.

3. End of the PID function.

3. Hardware and Software Requirements

3.1 Hardware Requirements

Turtlebot with 3D camera : A mobile robot platform that comes with a Kobuki base and equipped with 3D camera

Laptop or Desktop Computer: A computer with sufficient processing power and memory to run the required software tools and perform the necessary computations.

3.2 Software Requirements

Robot Operating System (ROS): A framework for robot software development that provides a set of libraries and tools to help developers create robot applications.

ROS Version: Kinetic

Laptop or Desktop Operating System: Ubuntu 16.04

Python: A programming language used for developing ROS nodes and scripts(rospy)

4. Methodology

The methodology of adding a follow feature to a robot using a 3D camera involves a few key steps that include setting up the hardware, designing and implementing the software, and tuning the control parameters to ensure optimal performance.

Firstly, the hardware setup requires attaching a 3D camera to the robot. The camera should be placed in a position that enables it to have a clear and stable view of the target object.

Secondly, a boosting tracker algorithm can be implemented to track the target object. The boosting tracker algorithm utilizes machine learning techniques to train a classifier that can identify the target object in real-time from the camera feed. The classifier is then used to track the object as it moves within the field of view of the camera.

The boosting tracker algorithm uses a feature selection approach to identify relevant features of the target object that are used to construct the classifier. This feature selection approach allows for the boosting tracker algorithm to work efficiently in real-time while still being able to accurately track the target object.

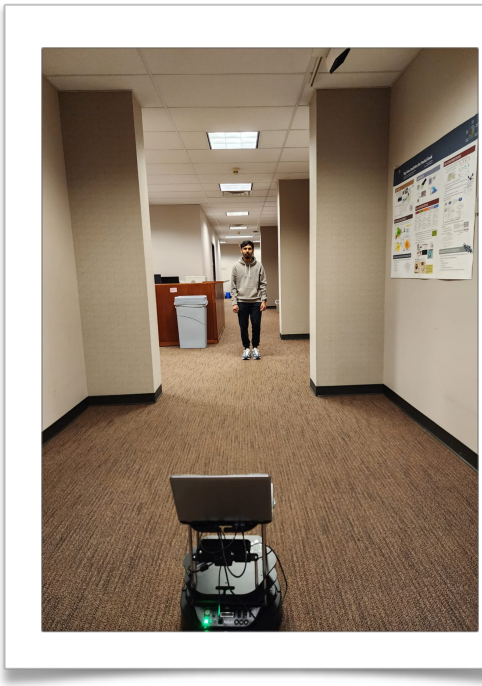
Thirdly, a proportional-integral-derivative (PID) controller can be implemented to control the linear and angular velocities of the robot. The PID controller uses feedback from the camera feed to adjust the velocities of the robot and ensure that it stays within a specific distance of the target object.

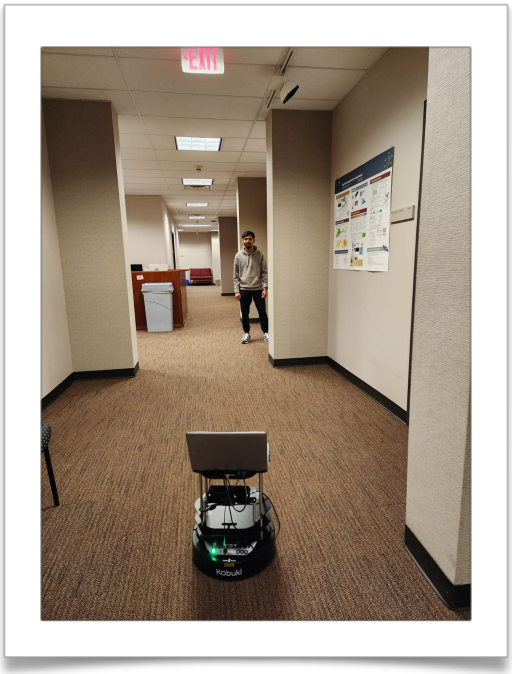
Here, the two PID controllers are used to control the linear and angular velocities of the robot to maintain the object at the center of the image frame. The distance between the robot and the object fed back into the linear velocity PID controller. The angular velocity PID controller adjusts the direction of the robot to maintain a desired orientation with respect to the object being followed. This allows the robot to smoothly track and follow the specific object.

The PID controller has three tuning parameters: the proportional gain, the integral gain, and the derivative gain. These gains are adjusted to ensure that the robot moves smoothly and accurately. The proportional gain adjusts the robot's movement in proportion to the error between the current position of the robot and the desired position. The integral gain adjusts the robot's movement to eliminate any steady-state errors. The derivative gain adjusts the robot's movement to prevent overshoot and oscillation.

5. Testing and Performance

Lastly, the follow feature is tested and evaluated to ensure that it is working as intended. During testing, robot should be able to smoothly and accurately track the target object. We tested the implementation by moving the object around the robot and observing its behavior. We also tuned the PID controller parameters to improve the performance of the robot. The gains are adjusted accordingly that the robot follows the target smoothly without oscillations or overshooting. The follow feature should also be able to handle various scenarios, such as stopping when the target object is out of range.





6. Conclusion

In conclusion, adding a follow feature to a robot with a 3D camera using boosting tracker, and PID controller can enhance the robot's capabilities and make it more useful for various applications such as surveillance, inspection, and navigation. The 3D camera provides depth information and enables the robot to detect and track a target object in real-time using a boosting tracker algorithm. The PID controller helps regulate the robot's velocity and direction, allowing it to follow the target object accurately and smoothly. Overall, the follow feature improves the robot's ability to interact with the environment and perform tasks that require tracking and following specific objects. Due to the system compatibility issues, we used OpenCV boosting tracker instead of OpenCV GOTURN tracker which will give more accurate results than boosting tracker. we leave these for future work.

7. References

1. https://www.ros.org/
2. https://jckantor.github.io/CBE30338/04.01-Implementing_PID_Control_with_Python_Yield_Statement.html
3. https://pyimagesearch.com/2018/07/30/opencv-object-tracking/
4. http://wiki.ros.org/rostopic
5. http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

7. Appendix

7.1 Python Code

turtlebot.py

```
#!/usr/bin/env python

import rospy
import cv2
from sensor_msgs.msg import Image, LaserScan
from cv_bridge import CvBridge
import copy
import math
import time
from geometry_msgs.msg import Twist


class TrackerNode:
    def __init__(self):
        self.tracker = cv2.TrackerBoosting_create()
        # Creates a Boosting Tracker
        self.cv_image = None
        rospy.Subscriber("/robot2/camera/rgb/image_raw", Image, self.image_callback)
        # subscribes to a rostopic which provides raw RGB image data from the robot's camera
        rospy.Subscriber("/robot2/scan", LaserScan, self.laser_callback)
        # subscribes to a rostopic which provides laser scan data
        self.vel_pub = rospy.Publisher('/robot2/cmd_vel_mux/input/teleop', Twist, queue_size=1)
        # publishes the velocity commands to the robot's rostopic
        self.laser_ranges = []
        self.laser_angles = []



    def PID(self,Kp, Ki, Kd, MV_bar=0):
        # initialize stored data
        e_prev = 0
        t_prev = -100
        I = 0

        # initial control
        MV = MV_bar

        while True:
            # yield MV, wait for new t, PV, SP
            t, PV, SP = yield MV

            # PID calculations
            e = SP - PV
```

```
            P = Kp*e
            I = I + Ki*e*(t - t_prev)
            D = Kd*(e - e_prev)/(t - t_prev)

            MV = MV_bar + P + I + D


 # update stored data for next iteration
            e_prev = e
            t_prev = t




    def image_callback(self, data):
        bridge = CvBridge()
        image = bridge.imgmsg_to_cv2(data, desired_encoding='passthrough')
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        # image = cv2.flip(image, 1)
        self.cv_image = copy.deepcopy(image)
        # the camera image is converted to a OpenCV format
        # if self.cv_image is not None:
        #     print("Image received in callback function")
        # cv2.imshow("Image Window", self.cv_image)



    def laser_callback(self, data):
        self.laser_ranges = data.ranges
        self.laser_angles = [(data.angle_min + i * data.angle_increment) for i in
range(len(data.ranges))]
        # laser scan data is stored
        # self.laser_angles = data.angles

    def run(self):


        angular_vel_controller = self.PID(0.005, 0.0, 0.0)
        linear_vel_controller = self.PID(0.05, 0.0, 0.0)
        angular_vel_controller.send(None)
        linear_vel_controller.send(None)
        start_time = time.time()
        # the two PID controllers are intialized, one for controlling the robot's angular velocity and
one for controlling the linear velocity


        if not rospy.is_shutdown():
            frame = self.cv_image
            flag_exit = False

            while self.cv_image is None:
                # if self.cv_image is not None:
                #     print("Frame has some value")
                print("Image not received yet”)
                time.sleep(1)
```

```
        # cv2.imshow("Image Window", frame)
        bbox = cv2.selectROI("Tracking", self.cv_image, False)
        self.tracker.init(self.cv_image, bbox)
        # select the ROI and initialized the tracker

        while not rospy.is_shutdown() and not flag_exit:
            frame = copy.deepcopy(self.cv_image)

            # Track object in current frame
            success, bbox = self.tracker.update(self.cv_image)
            current_time = time.time()
            if success:
                frame_h, frame_w, _ = self.cv_image.shape
                frame_cy = int(frame_h/2)
                frame_cx = int(frame_w/2)
                x, y, w, h = [int(i) for i in bbox]
                cv2.rectangle(self.cv_image, (x, y), (x + w, y + h), (0, 255, 0), 2)

                # Calculate center of bounding box
                cx, cy = int(x + w/2), int(y + h/2)

                if abs(cx-frame_cx) > 10:


                    # next(angular_vel_controller)
                    angular_vel = angular_vel_controller.send([start_time - current_time, cx,
frame_cx])

                    cmd_vel = Twist()
                    cmd_vel.linear.x = 0
                    cmd_vel.angular.z = angular_vel
                    self.vel_pub.publish(cmd_vel)
                    print("Angular Velocity: {}".format(angular_vel))
                    # the robot's angular velocity is published to a rostopic using twist message



                else:
                    # Convert center coordinates to cartesian and polar coordinates
                    # cartesian_coords = [cx, cy]
                    # polar_coords = [math.atan2(cy, cx), math.sqrt(cx**2 + cy**2)]

                    # Match polar coordinates with laser scan ranges using index
                    if len(self.laser_angles)>0:
                        # idx = min(range(len(self.laser_angles)), key=lambda i:
abs(self.laser_angles[i]-polar_coords[0]))
                        # range_at_polar = self.laser_ranges[idx]
                        range_at_polar = self.laser_ranges[(len(self.laser_ranges)-1)/2]
                        print("Range at Polar Coordinates: {}".format(range_at_polar))
                        if math.isnan(range_at_polar):
                            range_at_polar = 5

                        # next(linear_vel_controller)
                        linear_vel = linear_vel_controller.send([start_time - current_time, 1,
range_at_polar])
```

```
                    cmd_vel = Twist()
                    cmd_vel.linear.x = linear_vel
                    cmd_vel.angular.z = 0
                    self.vel_pub.publish(cmd_vel)

                    # Print cartesian and polar coordinates, as well as the range at the polar
coordinates
                    # print("Cartesian Coordinates: ({}, {})".format(cx, cy))
                    # print("Polar Coordinates: ({}, {})".format(polar_coords[0], polar_coords[1]))

                    print("Linear Velocity: {}".format(linear_vel))
                    # the robot's linear velocity is published to a rostopic using a twist message



                else:
                    cmd_vel = Twist()
                    cmd_vel.linear.x = 0
                    cmd_vel.angular.z = 0
                    self.vel_pub.publish(cmd_vel)
                    print("Waiting for laser scan data...")
                    # waiting for laser scan data

            else:
                cmd_vel = Twist()
                cmd_vel.linear.x = 0
                cmd_vel.angular.z = 0
                self.vel_pub.publish(cmd_vel)
                cv2.putText(self.cv_image, "Tracking Failed", (100, 80),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)
                # if the object tracking fails, then it displays the message "Tracking Failed" on the
image window
            cv2.imshow("Image Window", self.cv_image)
            cv2.waitKey(1)

            start_time = current_time


    def cleanup_function(self):
        cmd_vel = Twist()
        cmd_vel.linear.x = 0
        cmd_vel.angular.z = 0
        self.vel_pub.publish(cmd_vel)

if __name__ == '__main__':
    rospy.init_node('tracker_node', anonymous=True)
    node = TrackerNode()
    rospy.on_shutdown(node.cleanup_function)
    node.run()
```

7.2 Code Explanation

The code is a ROS node written in Python that uses OpenCV to track an object in real-time and uses the data from the laser scanner to control the movement of the robot based on the position of the object in the frame.
The node subscribes to two topics - "/robot2/camera/rgb/image_raw" to receive image data from the camera and "/robot2/scan" to receive laser scan data from the laser scanner. It then initializes a Boosting tracker from OpenCV and tracks the object in the frame using the tracker.
Once the object is tracked, the node calculates the center of the bounding box around the object and compares it with the center of the frame. If the object is not at the center of the frame, it calculates the angular velocity required to move the robot towards the object using a PID controller. The angular velocity is published to the "/robot2/cmd_vel_mux/input/teleop" topic to control the movement of the robot.
If the object is at the center of the frame, the node calculates the polar coordinates of the object and matches it with the laser scan ranges to get the range at that point. It then uses a PID controller to calculate the linear velocity required to move the robot towards the object. The linear velocity is published to the "/robot2/cmd_vel_mux/input/teleop" topic to control the movement of the robot.
The code also includes two PID controller functions - one for calculating the angular velocity and the other for calculating the linear velocity. These functions take in the PID constants and the setpoint as input and yield the manipulated variable (MV) which is the velocity required to move the robot in the desired direction. The functions use the current time, the process variable (PV) and the previous error to calculate the proportional, integral, and derivative terms of the controller.

7.3 Execution

Firstly, we need to connect the laptop to the robot and then we have to launch the following files in the Terminal individually

roslaunch turtlebot_bringup minimal.launch

roslaunch turtlebot_bringup 3dsensor.launch

rosrun tutorial turtlebot.py (Here, the tutorial is the package name)