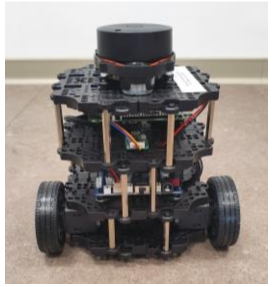
Sensor based navigation the "follow" algorithm



Tutor: Francesco Trotti
francesco.trotti@univr.it

Agenda

- Turtlebot3 sensors
- Laser scanner in ROS2
- “Follow” algorithm
- Exercise on simulated robot and turtlebot3

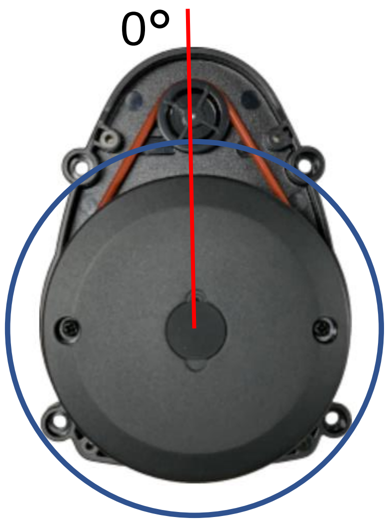
Turtlebot3 sensors

Turtlebot3 sensors

- Turtlebot3 has several sensors as:
  - Camera
  - Encoder
  - IMU
  - Lidar (Laser scanner)
- In this lesson we will be focusing on lidar

Turtlebot3 sensors - Lidar

- This lidar is a 2D laser scanner capable of sensing 360 degrees
- Scanning frequency is 5 Hz
- Ideal angular resolution 1°
- Max distance 8 meters
- Min distance 0.10 meters
- Starting angle (0 degree) is like the picture



Turtlebot3 sensors - Lidar

- Important value for a lidar:
  - Angular resolution: minimum angle that the sensor is able to distinguish
    - In other words, in how much time the sensor sends and reads a beam
  - Angle increment: angular distance between measurements
  - Max distance: maximum distance for a ray
  - Min distance: minimum distance for a ray

Laser scanner in ROS2

Lidar in ROS2

- ROS2 provides a specific message for lidar data (LaserScan message)
  - This message is generated by lidar of a real robot driver or from simulated lidar driver
  - The default topic name for lidar message is "/scan"
  - Several applications in ROS2 use this information
    - SLAM algorithm
    - Localization algorithm AMCL
    - Navigation stack
    - Collision avoidance algorithm
    - …

ROS2 Lidar message

- The lidar message provides several information:
  - Angle_min/max: start and end angles in radians
  - Angle_increment: angular distance between measurements in radians
  - Time_incrememt: time between measurements in seconds
  - Scan time: time between scans in seconds
  - Range_min: minimum ray distance in meters
  - Range_max: maximum ray distance in meters
  - Ranges: array of distance in meters
  - Intensities: intensity of the ray

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

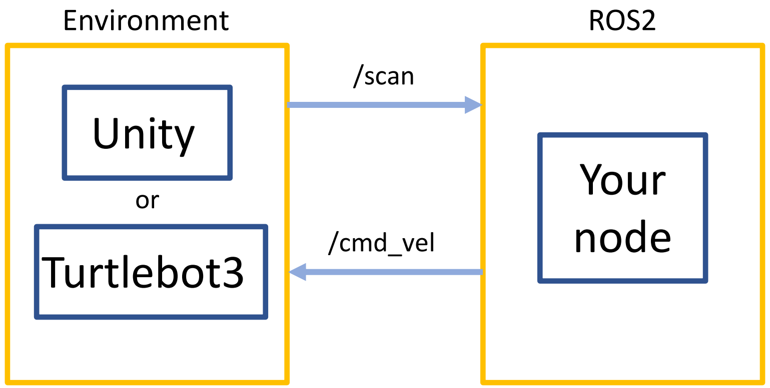
“Follow” algorithm

“Follow” algorithm

- The main idea is to develop an algorithm that moves the robot in order to “follow” the closest object in a pre-defined angle range and maintaining a pre-defined distance from it
- In order to do this, you have to:
  - Access and process the laser scan topic
  - Publish velocity command to the robot
  - Implement the algorithm to move the robot

"Follow" algorithm

- In your code you will have:
  - A subscriber to Laser scan topic (/scan)
  - A publisher for robot velocity (linear and angular)
  - A control loop to calculate the new command for the robot
    - You can implement the algorithm that you prefer: e.g., set a velocity that is linear or quadratic with respect to the error in distance, define different control law for different distances, …



"Follow" algorithm

- In control loop you will implement the new command for the robot
  - You have to read the scan message and save the information
    - Laser ranges
    - Angle min/max
    - Angle increment
  - Define the laser angle ranges in which you want to run the "Follow" algorithm
    - We suggest a range between 45 and -45 degrees
      - In ranges list these angles correspond to the position 0:45 for 45 and ranges.len – 45:ranges.len-1 for -45
    - In this way the robots will follow the object in the [-45,45] angle range

"Follow" algorithm

- Define the distance thresholds that the robot will consider
  - The max distance from the object (we suggest 0.60 meters)
    - This is necessary to avoid that the robot follows all objects
  - The min distance from the object (we suggest you 0.20 meters)
    - This is necessary to avoid collision
  - The max linear velocity (0.22 m/s)
  - The max angular velocity (2.80/-2.80 burger – 1.82 waffle)
    - This two thresholds are necessary to avoid extreme velocity values that may result in unstable behaviours
- Now we have to take the interest laser data from the *ranges* data stucture
  - Loop over ranges and save the value
    - Check if data is none before saving (np.nan_to_num numpy function do this)

"Follow" algorithm

- For each distance value in *ranges* in [-45,45] get the angle and save it
  - To obtain the angle you have to do:
    - angle = angle_min + (index * angle_increment) this return angle value in radians for positive angle (e.g 45 deg)
    - angle = angle_min + (index * angle_increment) – 6.28 this return angle value in radians for negative angle (e.g -45 deg)
- Now we have:
  - A list of distances in a defined ranges
  - A list of respective rays angles

"Follow" algorithm

- To compute the liner velocity command you can use the minimum of the distance list
  - To obtain the command you have to apply a very simple control law, e.g.:
    - Liner_vel_x = min_dist^2
- To compute the angular velocity:
  - Get the index of the minimum distance value
  - Use it to find the respective value in angle list
  - (Assume that the two lists were generated at the same time)
  - Obtained the angle of ray with minimum distance, the command will be:
    - Angular_vel_z = min_angle^2 * np.sign(min_angle) (remember the sign if you use the quadratic function)

"Follow" algorithm

- If everything is correct you can publish the cmd_vel message
- The correct behavior should be that the robot follows the nearest obstacle and stops when it arrives at minimum distance

Exercises

Exercises

- Clone the repository
  - https://gitlab.com/TrottiFrancesco/mobile_robotics_lab.git
- In Unity Hub open the new cloned project with the update
- In colcon_ws copy the new package called "turtlebot3_follow"
- Two main exercise
  - Implement the “follow” algorithm in simulation “Unity”
  - Test the “follow” algorithm on the real robot

Exercise 1

- If you click play button in the Unity scene you will see a yellow cube
- With your mouse you can drag the cube in order to use it as a target for the “follow” algorithm
- In ROS2 package (turtlebot3_follow) in folder "turtlebot3_follow" you will find a python script called "follow.py"
  - The follow.py script is a partial implementation of the “follow” algorithm it that you have to complete.
- In Unity the laser scan is an ideal lidar with the same turbtlebot3 features

Exercise 1

- To build and run you code you have to:
  - Build (colcon build)
  - Source (. install/setup.bash)
  - ros2 run turtlebot3_follow turtlebot3_follow
- In Unity the robot will have to follow the yellow cube
- If you move the cube the robot will have to follow it

Exercise 2

- Try your algorithm on real robot
  - Run bringup on turtlebot3
  - Run on your pc the “follow” node
  - Put an obstacle in front of the robot and move it

References

- Lidar
  - https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_02/
- ROS 2
  - Message
    - https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html