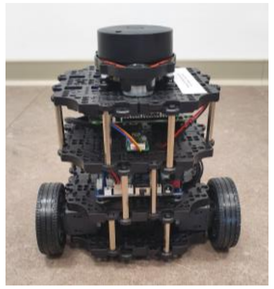
High-level control for sensor-based navigation



Tutor: Francesco Trotti
francesco.trotti@univr.it

Agenda

- High level control and sensor-based navigation
- Exercise on simulated robot and turtlebot3

High level control for sensor-based navigation

- High-level control refers to the ability of executing a **complex** task, i.e., a task that is composed of several sub-tasks.
- When we talk about high-level control, we are raising the level of abstraction with respect to the control and data processing from sensors
- The implementation of **actions** on the robot aims at creating generic primitives that can be composed to perform different complex tasks

Available solutions for High-level control

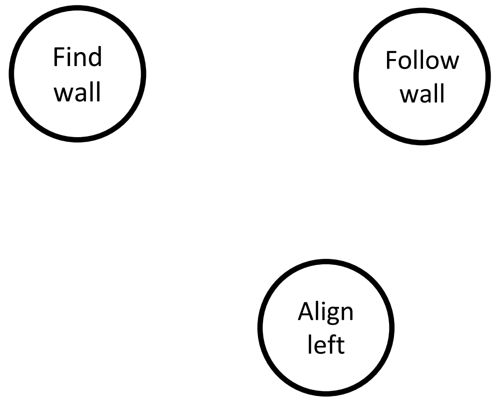
- Currently in the state-of-the-art there are several solutions for high-level control:
  - FSM (Finite State Machines)
  - Behavior-tree
  - Petri Net Plans
- There are several extensions of these tools, for example in Hierarchical FSM, a state could be a primitive or a sub-task that contains another FSM.
- We will focus on the simplest too: FSM

High-level control using FSM: “wall following”

- The aim of the exercise is implementing a "wall follower" algorithm by defining a FSM
- For the exercise we will define a list of tasks to be executed using a simple FSM
- Each state of the FSM will be a task for the robot; in our exercise primitive and task coincide because we are dealing with a fairly simple scenario
- To guarantee the correct task execution it is necessary to define the constraints and the rules that control the evolution of the FSM

Wall following as a FSM: states

- The states (robot tasks) that will compose the FSM are:
  - Find a wall
  - Follow the wall
  - Align left

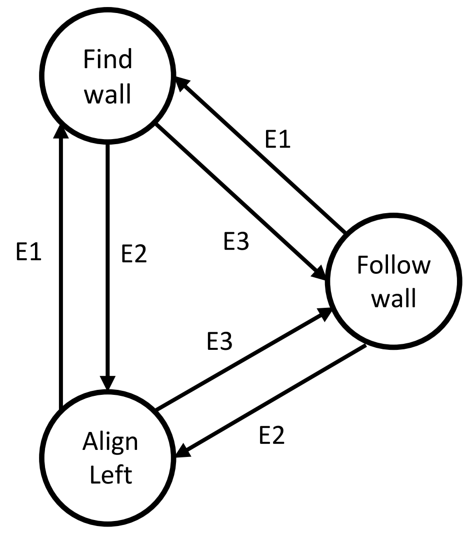


Wall follow as a FSM: transitions

- Constraints:
  - Define the rotation side of the robot to keep the wall to the right or to the left
  - Define a threshold in order to specify when a “wall” (i.e., a set of aligned laser readings) is detected and to avoid collisions with the wall
- Rules to evolve the FSM (transitions):
  - A **wall** is found if a range of lidar readings (i.e., **region**) are aligned within a given distance
  - To follow the **wall** it is necessary to align the robot
    - If the front lidar **region** detects a wall, the robot must turn left (right)
  - Follow the wall going straight and maintaining the distance to the wall
  - If front and right (left) lidar regions are inside the threshold, the robot must turn left (right)

Wall follow as a FSM

- The FSM for left rotation (keep wall to the right):
  - E1:
    - If right region is greater than Th
  - E2:
    - If front region is less than Th **and** other regions are greater
    - If front and right regions are less than Th **and** other region is greater
    - If all lidar regions are less than Th
  - E3:
    - If front and left regions are greater than Th **and** other regions are less



Wall Follow: implementation of primitives

- The "find wall" state is the initial state of our FSM, while there is no final (or accepting) state.
- Once the transitions for the evolution of the FSM have been defined, it is necessary to implement the primitives that the robot must execute.
  - Find wall:
    - Move the robot with a defined linear and angular velocity
  - Align left:
    - Rotate the robot in place with a defined angular velocity
  - Follow wall:
    - Go straight with a defined linear velocity
- Note these primitives are general, and we can use them also for other complex tasks

Exercises

High Level Control – Exercise 1.a

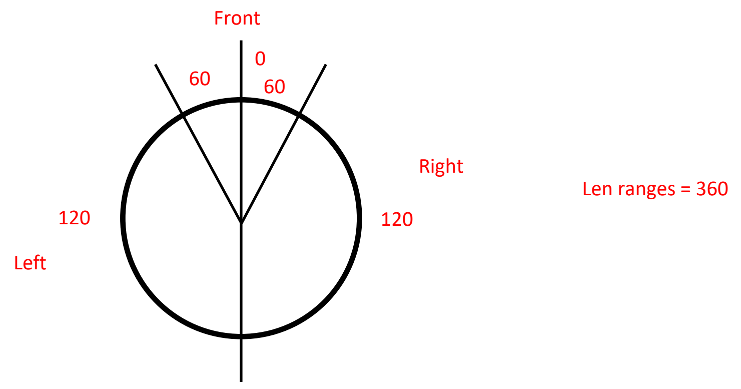
- To implement the HighLevelControl node it is necessary to use a new environment in Unity and add your template node in your workspace
- Clone the repository
  - https://gitlab.com/TrottiFrancesco/mobile_robotics_lab.git
- Open in Unity the new project.
  - In unity navigate to scene directory and here you should find a scene called "turtlebot3HighLevelControl"
  - Drag this scene in history and delete the old scene
- Copy the new package called "turtlebot3_ HighLevelControl" in "colcon_ws", build and source the environment

High Level Control- Exercise 1.a

- Implement the FSM as shown before.
- The robot must follow the wall and switch the FSM states in order to keep the wall on the right side of the robot, so the robot should align turning left
- To complete the first exercise, you must implement the primitives and the conditions to trigger the transitions between the states

High Level Control - Exercise 1.a

- An important part is defining the ranges that define the regions for the lidar, in the template you can find the struct initialization with three ranges:
  - Front
  - Left
  - Right



- We suggest to divide the length of message ranges list by the number of areas in order to obtain the number of rays per range
- We suggest to parametrize the number of rays for each region to facilitate you on the real robot

High Level Control- Exercise 1.a

- Now you have to find a correct linear and angular velocity
  - Min velocity is 0
  - Max linear velocity is 0.22 m/s
  - Max angular velocity is 2.84 for burger, 1.82 for waffle
- Knowing min and max values you can use a feasible value to move the robot (we suggest to move the robot slowly)
- Set the distance from the wall (we suggest 0.15 m)

High Level Control- Exercise 1.a

- In the template you will find:
  - Struct definition (lidar ranges and FSM states)
  - Laser scan callback
    - You will have to divide the lidar ranges in regions
  - Definition and implementation of change state function (change_state)
  - Definition of the triggers for the change state function (take_action)
    - You will have to implement the rules to evolve the FSM
  - Definition of actions/states of the FSM ("find_wall", "align_left", "follow_wall")
    - You will have to implement the actions for the robot

High Level Control- Exercise 1.a

- To run your node, you have to compile and source your environment
  - Colcon build
  - . instal/setup.bash
- Click play button in Unity
- Run your node with
  - ros2 run turtlebot3_ HighLevelControl turtlebot3_ HighLevelControl

High Level Control - Exercise 1.a

- The correct behavior should be that the robot approaches the wall, turns and follows it. When there are corners, the robot should continue to follow the wall turning to maintain the correct distance toward the correct side

High Level Control - Exercise 1.b

- The second exercise is similar to the first but now the robot has to keep the wall on the left side
- For this exercise you must create a new state called "align_right" and you must implement the rules to do the transition to the new state
- In your code you should create a boolean variable to decide which side of wall you want follow.
- In this way, before executing the node you can choose the rotation direction

High Level Control - Exercise 1.b

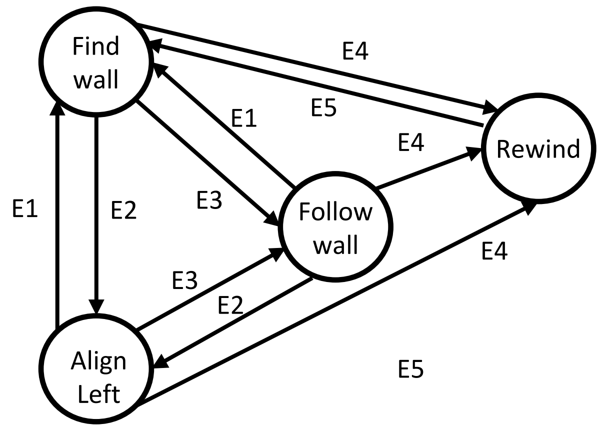
- If everything is correct if you set your boolean variable to keep the wall on the left side the robot have to turn on the right side and follow the wall
- In other case if you set your boolean variable to keep the wall on the right side the robot have to turn on the left side and follow the wall
- Note that you can implement this as you prefer but creating a different FSM for keeping the wall to the left maybe the easiest way.

High Level Control - Exercise 1.c

- The third exercise is based on adding a new state called "rewind" in the FSM and a new condition to do the transaction
- This new state allows the robot to reverse the path created before with the standard execution of the FSM
- During the normal execution of your FSM, you should save the velocity command and when you click a keyboard key the robot will have to stop, turn of 180 degree and come back executing the velocity commands in reverse order and with the opposite angle
- When you click another keyboard key or the list of the saved velocity is ended the robot should go out of the "rewind" state and return on the "wall follow"

High Level Control - Exercise 1.c

- You have to add a state in the FSM that provides the saved velocity to the robot (be careful because the direction is opposite)
- You have to add a new trigger to go in the new state, click of keyboard key (we suggest to implement a thread to do the key click detection and save the click on a class variable)
- You have to add a new trigger to go back to the wall follow FSM, consider adding also a safe condition to avoid collisions
- The FSM will be:



High Level Control - Exercise 1.c

- The correct behavior will have to be the following: the FSM runs normally and when clicking a key, the robot stops and comes back with the old command but in the opposite direction. When you click the other key or the list of the saved velocity is empty, the robot restart with the "wall follow“ behaviour.

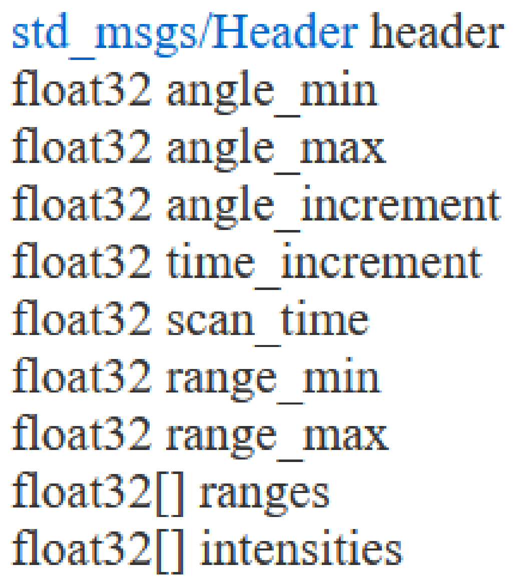
High Level Control – turtlebot3

- If in simulation the behavior is correct, you can try the three exercises on the real robot
  - Connect to the robot with ssh
  - Run bring-up on turtlebot3
  - Run your node in your PC

References

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
```

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
  - Create package
    - https://docs.ros.org/en/foxy/Tutorials/Creating-Your-First-ROS2-Package.html
  - Pu-Sub node
    - https://docs.ros.org/en/foxy/Tutorials/Writing-A-Simple-Py-Publisher-And-Subscriber.html