Laboratory exercise 5

Turtlesim in C++

Name: Sérgio Miguel Rosa Estêvão **JMBAG:** 0036563035

Preparation

- **Do not: consult, read, or examine ANY materials or solutions for assignments from the previous years' editions of this course, or from your colleagues. The assignment solution must *fully* be your work. In case signs of plagiarism are detected, you will get a zero score and may be subject to be reported to the Faculty's Ethics Committee.**
- **Do: consult the teaching staff (for this assignment, teaching assistant Vlaho-Josip Štironja) via Teams DM, if you have any problems, if anything is unclear, or if you need any help with any part of the assignment.**
- Review the lecture slides about writing a ROS subscriber and publisher in C++. Review the code from the lecture, build it on your system, and use it as a template for the assignment.
- Write clean, readable, easy-to-understand code. Give meaningful names to variables. Make sure that you check that your solution runs without syntax errors and works as intended. (Remember that you can always ask for help!)

Assignments

Task 1 : Lawnmower algorithm in Turtlesim

a) For this assignment, you will have to program a turtle lawnmower algorithm. The turtle **should start from the lower left corner** and continue straight until it reaches the end and then turn in a small circular arc and continue in the opposite direction. This should go on until the turtle covers the whole area. You can assume that the position of the turtle is known (`turtle1/Pose`) and that the size of the environment is known (an 11×11 square). We are not asking you to implement a complete coverage algorithm; detect when the robot is close to the edge, and then make it turn, otherwise maintain a straight heading. Your result should resemble Figure 1.

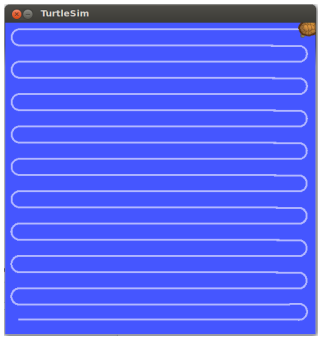

Figure 1: The result of the turtle lawnmower algorithm in TurtleSim.

In the following text box, report every command needed for reproducing your lawnmower in Turtlesim.

```
rosrun turtlesim turtlesim_node
rosrun lawnmower_turtle lawnmower_turtle_node
```

How did you manage to put the turtle in the lower left corner? Write the command(s) in the following text box.

Through the command line:
rosservice call /turtle1/set_pen 255 255 255 2 true
rosservice call /turtle1/teleport_absolute 0.5 0.5 0.0
rosservice call /turtle1/set_pen 255 255 255 2 false

b) Inside the `turtle_lawnmower` package create a new folder called `launch`. Write a launch file named `firstname_lastname.launch` which will run both turtlesim and the lawnmower node at once. One should be able to reproduce your result simply by running your launch file.

Task 2 Turtlesim chase game

In this task we will implement autonomous Turtlesim chase game employing P and PI regulators. Chaser turtle `turtle1` tries to catch `turtle2`. After catching it, `turtle2` will appear in a new random spot in the environment after a short time.

a) Place the given package turtle_game in the catkin directory and build it using `catkin_make` command. Run the launch file from the package using `roslaunch turtle_game start.launch`. Remember to source the `setup.bash` file from the devel folder after building the package within your workspace, as shown in lectures. Once you've executed the launch file, attempt to catch another turtle using the keyboard to verify that everything is working correctly. Examine the code and answer the following questions.

What is the purpose of the `threshold` variable within `turtle_game.cpp`? Would the game still operate correctly if it were set to a large value (~ 100), and why or why not?

the threshold is the minimum distance the chasing turtle has to be from the target so it can be considered as caught.
The greater the threshold, the easier it is to catch the target since the chasing turtle doesn't need to

What would happen if the `RemovePen` function within `turtle_game.cpp` was not implemented? Would the game operate as per the instructions?

The turtle movements would leave a trail, but the game would still operate normally

What is the purpose of the `GetDistance` function in `turtle_target.cpp`?

Returns the euclidean distance between the target turtle and a specific position, which in this context is the chasing turtle

b) In this task, we will implement an autonomous turtle chase game similar to Laboratory Exercise 2. The following block diagram is shown in Figure 2 will help you visualize the interaction between turtles. The role of the chaser turtle is to drive towards the chaser target by publishing the according velocity. The command velocity message type `geomtry_msgs/Twist` consist of **linear** and **angular** velocity

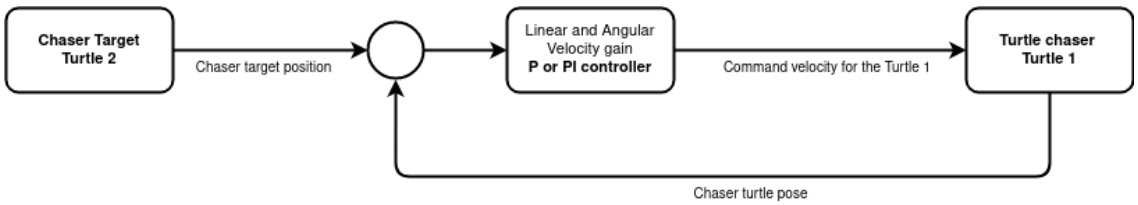

Figure 2: A block diagram illustrating how the chaser turtle (`turtle1`) is chasing the target (`turtle2`) serves as an example of a closed feedback loop control system. A feedback loop is a mechanism that compares the actual output of a system with the desired output, and adjusts the input accordingly to reduce the error and it is a common concept in robotics and automation.

For controlling mobile robots in 2D environments, such as in TurtleSim, there are three relevant components of the `geometry_msgs/Twist` messages:

- `linear.x` - v_x - the **forward** mobile robot velocity (m/s)
- `linear.y` - v_y - the **lateral** mobile robot velocity (m/s)
- `angular.z` - ω - the **turning** mobile robot velocity (rad/s)

Note: **Same as in 2. laboratory exercise, we will not use lateral velocity in this assignment. Therefore it should be left as default, zero.**

Knowing space coordinates of Turtle Sim from Laboratory exercise 2. we will revisit the control law for a P-controller for given angular and linear distances at step k:

$$v_{x,k} = K_{p,v} \cdot ||\mathbf{d}_k||$$

$$v_{y,k} = 0$$

$$\omega_k = K_{p,\theta} \cdot \text{wrap\_to\_pi}(\Delta_k).$$

Here, $K_{p,v}$, and $K_{p,\theta}$ are non-negative constant provided by the user, commonly referred to as proportional coefficients in the context of P-regulators. $||\mathbf{d}||$ and Δ represent the euclidean and angular distances between turtles, respectively while wrap_to_pi function wraps angular values to interval from - π to π before further use.

Fill in the code at the specified spaces (TODO comments in `turtle_game.cpp`) to achieve the expected behavior.

Execute the simulation using the given proportional coefficients ($K_{p,v}$, $K_{p,\theta}$) for two scenarios:

- (0.1, 2.0)
- (1.0, 2.0)

What differences do you observe between the two scenarios?

The Kp_v coeficient influences the velocity of the turtle, with 0.1 the turtle takes significant more time reaching the target then with 1.0

Hint: You do not need to rebuild your program when changing the values, as they are read at runtime from the `PI.yaml` file.

Execute the simulation using the given proportional coefficients ($K_{p,v}$, $K_{p,\theta}$) for two scenarios:

- (1.0, 2.0)
- (1.0, 10.0)

What differences do you observe between the two scenarios?

With a higher kp_theta, the chaser turtle is able to make sharp rotation more quickly, reaching the target faster compared to low kp_thetas

c) Currently, the PI controller is widely adopted in industrial application due to its simple structure, design and low-cost. PI controller will eliminate steady-state error results in operation of P-controller due to addition of integral part of controller.

Using discrete PI controller, the control law will be following:

$$v_{x,k} = K_{p,v} \cdot ||\mathbf{d}_k|| + K_{i,v} \sum_k ||\mathbf{d}_k||$$

$$v_{y,k} = 0$$

$$\omega_k = K_{p,\theta} \cdot \text{wrap\_to\_pi}(\Delta_k) + K_{i,\theta} \sum_k \text{wrap\_to\_pi}(\Delta_k)$$

where $K_{i,v}$, and $K_{i,\theta}$ stand for integral coefficients.

Edit the code according to the aforementioned control law. Notice how in the case of K_i the PI controller becomes a P controller.

Execute the simulation using the specified coefficients ($K_{p,v}$, $K_{p,\theta}$, $K_{i,v}$, $K_{i,\theta}$): (1.0, 2.0, 0.03, 0.03). Compare the results of this simulation with those from task b). What differences do you observe in the simulation outputs?

The integrator component will make the chasing target accelarate and reach the target with more speed compared to the simulation in b)

Execute the simulation using the specified coefficients ($K_{p,v}$, $K_{p,\theta}$, $K_{i,v}$, $K_{i,\theta}$): (1.0, 2.0, 1.0, 1.0). Based on the comparison with the last scenario, draw conclusions about the impact of these parameter changes on the simulation results.

The use of coeficient two levels of magnitude above the ones used in the last questions makes the chasing turtle achieve a much higher speed much quicker. This causes the chasing turtle to end up missing the target and circunvent them, due to the high speed it is unable to turn effectively to reach the target, making this a much worse choice of parameters.

Exercise submission

Create a zip archive containing **this pdf with the filled out answer** and **all other exercise files**. Organize the files in zip archive into the following structure:

- Folder **Lawnmower** containing your .cpp files, CMakeLists.txt, a screenshot of your result, and your launch file for Task 1.
- Folder **Game** containing your filled `turtle_game.cpp` file.
- This pdf with the filled out answer.

Upload on Moodle.