Weekly project - Python3 for Robotics

Author: Joaquin Rodriguez

August 29, 2021

Task: Simulate a robot that goes to target

This small project is your task to practise everything what we have explained in the module Python3 for Robotics. This will set the fundations of knowledge to continue with the ROS module for Python. This project should not take you too much time, but I recommend you to start as soon as possible and not one hour before the deadline.

You have to create three classes to solve the problem:

- A Robot class
- A control class
- A user interaction class.

Your program has to simulate a Go-To-Target application. You can work in teams of 2 people MAX.

Problem explanation

The robot localization is done by a 3D vector: (X, Y, theta). See Figure 1.

1. (X,Y) are the cartesian coordinates of the center of the robot, with respect to the origin of the reference coordinates system.
2. theta is the orientation of the head of the robot with regard to the horizontal line.

The robot can move to any place in the plane. For doing so, the robot moves to a speed that can be set by software, and once set, it is considered to be constant. The robot position is updated every certain amount of time Δt, called Discretization time, and the equation of position update is:

$$X_k = X_{k-1} + V.\Delta t$$

where $X_k = (X, Y)$ is the position of the robot at the time $t = k$, and $V = (V_x, V_y)$ is the speed vector. The speed vector can be expressed in polar coordinates as:

$$V\Delta t = (V_x, V_y)\,\Delta t = (V_f.cos\,(\theta)\,, V_f.sin\,(\theta))\,\Delta t = (V_f.cos\,(\theta_{k-1} + \omega\Delta t)\,, V_f.sin\,(\theta_{k-1} + \omega\Delta t))\,\Delta t$$

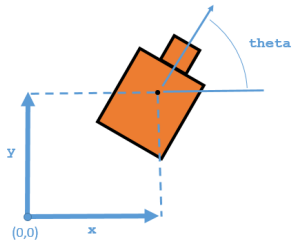

Figure 1: Robot localization sketch

$$\Rightarrow \Delta X = V\Delta t = \left(V_f.cos\left(\theta_{k-1} + \omega\Delta t\right), V_f.sin\left(\theta_{k-1} + \omega\Delta t\right)\right)\Delta t$$

If we consider the robot can rotate in place also, the previous equation means that the speed commands can be decomposed into two independent variables: The forward speed V_f and the rotation speed ω. The discretization time Δt can be considered as constant.

In control theory, the most basic controller we can use is a PID controller. PID stands for Proportional - Integrative - Derivative, and they are three components, with different coefficients, that can be used to change response of our model. The coefficients have to be set in order to avoid overshooting or oscilating responses, and that the system responds as quick as possible. To simplify the problem, we will only use the proportional controller.

To control the speed of the robot with this type of controller, we compute the speed of the robot to be proportional to the error with the target point, i.e.:

$$S_o = K.\left(P_k - P_{target}\right)$$

where S_o is the output signal (either forward speed or rotational speed). K is the constant that we can adjust in order to change the robot behavior. P_k is the actual value of the variable we want to control (distance to target), and P_{target} is the desired value we want our variable to have (if we control the distance to the target, this value must be zero). This type of controller has a problem: we will never reach the target point exactly. Each time we approach to the target, the difference between the actual value and the target value will be reduced too. Therefore, the output signal will be reduced too. If the output variable is the speed, it means the robot will slow down the speed as we approach to the target. At some point, the set speed will be so small that the robot movement will be neglictible. In summary, there is always an error in the position. The greater the controller constant, the smaller the error, but the more inestable the system will be. So, there will be a trade-off between the error we can tolerate, and the robot response we can achieve.

With the Robot class, we should be able to:

1. Query and set the robot name,
2. Query the current position,
3. Set and query the forward speed (positive number in m/s), and the rotational speed (in rad / sec).
4. We should also be update the robot position every a fixed amount of time (discretization time).
5. The Robot has a maximum speed of 0.3 m/s and 0.3 rad / sec. If higher speeds want to be set, this maximum value must be used.

The control class would be a proportional controller, and at each iteration, it computes the rotational speed, and the forward speed, separatedly. We should be able to set the proportional gains in the construction of the class, and to call a function that receives the actual position, the final position, and it computes the required speeds.

The behavior of the controller must first correct the robot orientation in place (i.e., only change the theta, while the forward speed is 0). If the theta error is smaller than ***5 degrees***, the forward speed can be non-zero, while the theta orientation stills being corrected. If at any time, the theta error becomes larger than 5 degrees, the robot must stop and correct the angle again.

In the main program, set an ***iteration time of 1 mS***, and the approximation ***error to 1 cm***.

Here you have a snippet of the main code. You can use it, and you can modify it if you need it.

```
import time

robot = myRobotClassName(robot_name, max_speed, initial_pos)
controller = myControlClassName(forward_speed_gain, rotational_speed_gain)

iteration_time_sec = 0.001
target_pos = [target_x, target_y]

# Initialize robot time:
t_k = time.time()
robot.init_task(t_k)
while True:
```

```
    if distance_to_target(actual_robot_pos, target_pos) < 0.01:
        break
    else:
        # Compute forward and rotation speed with controller
        # set speed to robot
        t_k = current_time
        time.sleep(iteration_time_sec)
        currrent_time = time.time()
        # update robot pos with t_k and current_time

# Set robot speed to zero
print("{} arrived to the target!".format(robot_name))
```