Assignment 4

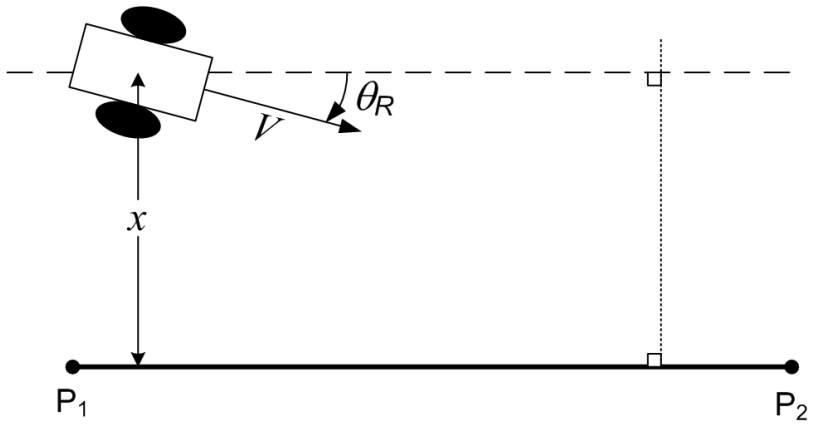
1. **50**% **(Team)** Implement three functions *translate(double d) and rotate_rel(double angle*. The translate function takes one argument the distance to be travelled in meters. The relative rotate functions take one argument the angle of rotation. Use the time-needed multiplied by the commanded velocity to calculate how much you have travel.
   In particular ensure you have installed the *robot_pose_ekf* node, and subscribe both to *odom* and *odom_combined* messages.

   Each team should implement different test patterns:
   - Straight line (1 m)
   - Rotation (relative) (+/- 30)

   Then run the robot several times for each pattern and manually estimate distance travelled and actual rotation. This will provide enough information to characterize the noise of the motion model.
   Report:
   - Actual distance travelled (manual measurement), commanded distance travelled, odom estimate, and odom_combined estimate.
   - Actual angle rotated (manual measurement), commanded distance travelled, odom estimate, and odom_combined estimate.
   - Estimate, mean error, and variance.

2. **50% (individual**) In class, we saw different components that can be used in a controller:
   a) a proportional component (P)
   b) a derivative component (D)
   c) an integral component (I)

   The goal of this assignment is to get you familiar with how to use these type of controller, and particularly in the context of robotics. As such, you will get a differential drive robot to perform a line or path following task. A line segment is defined by two points, P1 and P2. The goal is to get the robot to drive on the line, in the direction P1 → P2. Once you get near P2, you should either stop, or switch to the following next line segment, if available. What criterion you use for switching is left to you, but it should be stable: for example, if you pick a very short distance as a switching criterion, it might not work all the time (bad!).

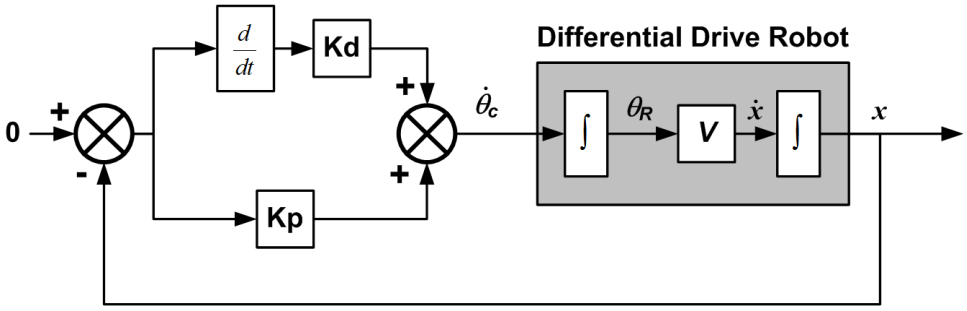

θ_R
V
x
P_1
P_2

In terms of variables for this problem, we have:

- distance x to the wall;
- desired distance x_d to the wall
- desired heading of the robot , relative to the wall θ_d;
- the desired rate of turn of the differential drive robot ω_R, (*turnrate*);
- actual heading of the robot θ_R;
- the *fixed* forward velocity of the robot V.

In the context of control theory, wall following can be defined as trying to maintain $x = x_d$. This also implicitly requires $\theta_d = 0$, otherwise the robot would drive away from the wall.

We saw in class how we can use a P-D controller to maintain the trajectory of a robot on a line, while driving at a constant velocity V. The P part was getting the robot to move towards the line, and the D part "cued" the controller to reduce the correction as it gets closer to the goal. Without the D part, the robot was oscillating constantly around the line. The block diagram for this approach is shown below. The leftmost 0 simply indicate we want to be at a distance 0 from the wall. The grey box is a simplified model of the differential drive robot.



Using ROS and stage, write a program that will make the robot follow the closest obstacle. You can either start the robot fairly close to one obstacle, or write a function that finds the closest obstacle and then moves towards it until it reaches near x_d.

Create a world (modify one of the *png* files) in such a way to demonstrate limitations of the two techniques. For example, have a corner that is too narrow, thus requiring the robot to make a big rotation; introduce a second obstacle that is closer than twice the desired distance. Discuss the resulting behavior of the two approaches.

Hints

- **Direction**
  If your system diverges, it might be because the commands have the wrong direction. Double check that by placing the robot near an obstacle, and look how your error and

commands change. If this is the source of your problem, simply multiplying by -1 the troublesome value.

- **Computing angles**
  Use atan2, not atan.

- **Phase Unwrapping**
  Remember that computations on angles are tricky. After any computation, you might want to shift the angle between $-\pi$ and π by adding or subtracting multiples of 2π. Simple examples of faulty computation if you do not do that, with the correct phase unwrapping (example given in degrees but the same applies for rad):
  - 0-360=-360 (wrong) -360+360 (phase unwrap) = 0 (correct)
  - 170 - -180 = +350 (wrong) +170 - 180 (phase unwrap) = -10 (correct)

- **Capping the Heading Command**
  If the robot is far from the wall, the heading command θ might be more than $\pi/2$ rad, in which case the robot will go in the wrong direction, or start making spirals. You might therefore want to cap this command c to be within +/- $\pi/2$.

Your assignment should consist of a pdf document that discusses your approach, difficulties and potential improvements, and a copy of your source code for verification purposes.