Assignment 2, Due Nov. 9th 2009

(Worth 15%)

October 21, 2009

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, P_1 and P_2. The goal is to get the robot to drive on the line, in the direction $P_1 \rightarrow P_2$. Once you get near P_2, 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!).

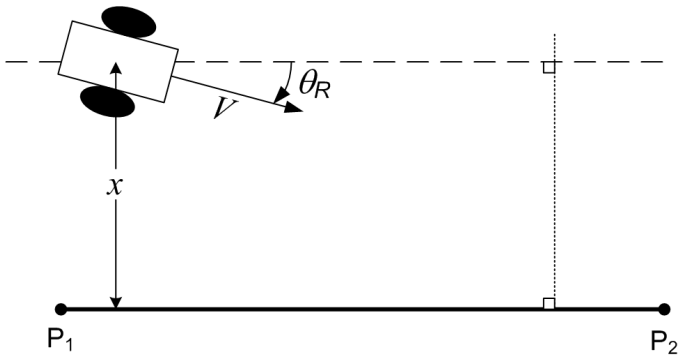


Figure 1: Schematic of the line following problem.

In terms of variable for this problem, we have :

- distance x to the line;
- desired heading of the robot θ_c, relative to the line;
- the desired rate of turn of the differential drive robot $\dot{\theta}_c$;
- actual heading of the robot θ_R, relative to the line;
- actual rate of heading change of the robot $\dot{\theta}_R$, relative to the line;
- the *fixed* forward velocity of the robot V.

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

1 Single Loop with P-D Controller

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 in Fig. 2. The leftmost 0 simply indicate we want to be at a distance 0 of the line. The grey box is a simplified model of the differential drive robot.

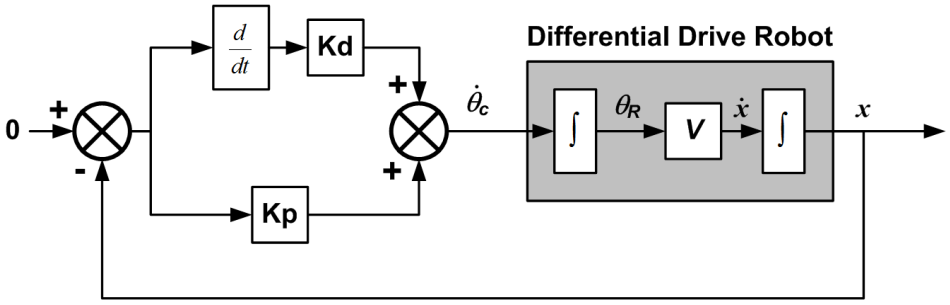


Figure 2: Block-diagram of the single loop with proportional-derivative controller.

Using player and stage, write a program that will make the robot follow a path made of 5 line segments corresponding to these locations:

- : Line 1: [-7 -5] → [5 -5]
- : Line 2: [5 -5] → [6 6]
- : Line 3: [6 6] → [-6 0]
- : Line 4: [-6 0] → [7 -7]
- : Line 5: [7 -7] → [9 -7]

The forward velocity V of the robot should be 0.2. Use the configuration `emptybox.cfg` that contains walls that are far away from those 5 lines. Note: Before you start coding, please read the **Hints** Section 5.

a) 20 pts For 4 different meaningful combinations of Kp and Kd, including one combination where $Kp = 0$, and one where $Kd = 0$:

- Record the path, and plot it on top of the line path.
- Compute the average absolute error you have.
- Plot the command $\dot{\theta}_c$ over time.
- Plot the error over time.

Discuss briefly those results.

b) 10 pts Pick a fixed value of Kp. Do 10 trials with increasing values of Kd (hint: you can do that in a for loop inside your code). Compute the average absolute error for each of the paths.

Plot the mean average error as a function of Kd. Do not plot each path!! Discuss how this error changes with Kd.

2 Cascading Loops

For this section, we will use two proportional loops in cascade, as in Fig. 3. The inner loop should be a proportional controller that implements this equation:

$$\dot{\theta}_c = Kp_{inner} * (\theta_c - \theta_R) \tag{1}$$

and the outer loop would be this proportional command:

$$\theta_c = Kp_{outer} * (0 - x) \tag{2}$$

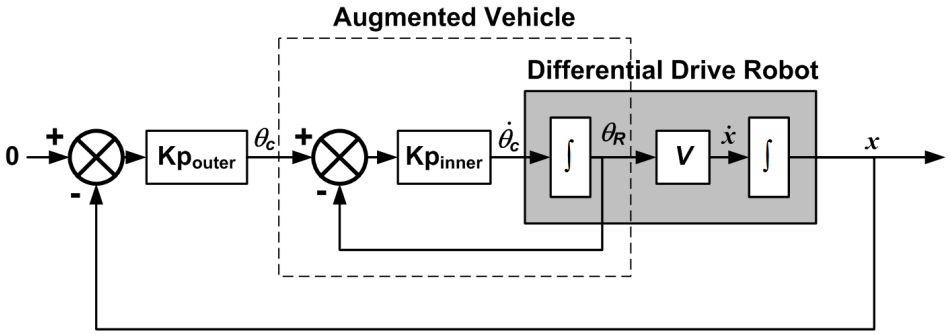


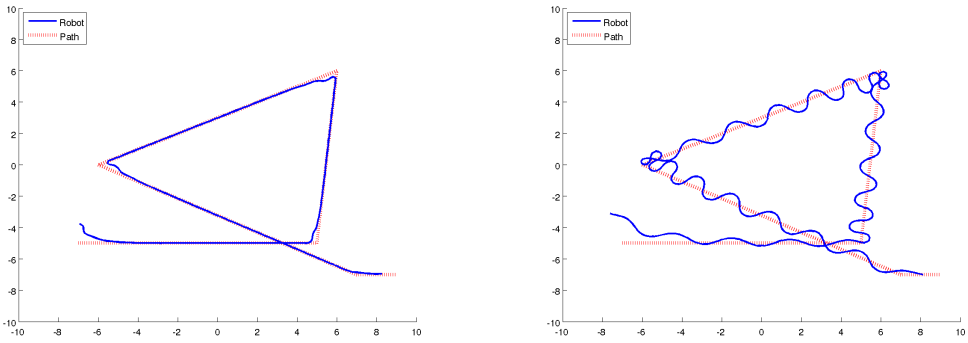
Figure 3: Block-diagram of the double cascaded loops with proportional-derivative controller.

a) 20 pts For 3 different meaningful combinations of Kp_{inner} and Kp_{outer}:

- Record the path, and plot it on top of the line path.
- Compute the average absolute error you have.
- Plot the command $\dot{\theta}_c$ over time.
- Plot the error over time.

Discuss briefly those results, and compare them to the results you had with the other controller. Overall, your response with delay should look like Fig. 4(a).

b) 10 pts Artificially add some delay in the command loop, to simulate robot imperfections. You can place the delay between $\dot{\theta}_c$ and the command you send to player. A delay is simulated by storing the information in a buffer. Let's say you want have a delay corresponding to 10 iterations, between the `in` and `out` variables. The C pseudocode would be:



(a) (b)

Figure 4: What you should get with the cascade controller a) without delays b) with delays and gains too high. Results plotted in matlab, from a CSV file saved while running player/stage.

```
out = delay[0];
for (i = 0; i < 9; i++) {
    delay[i] = delay[i+1];
}
delay[9] = in;
```

Find the values of Kp_{outer} and Kp_{inner} at which the system starts oscillating more or less out of control, for delay of 1, 2, and 4 iterations. Overall, your response with delay should look like Fig. 4(b).

c) 10 pts In this question, we will look on how this controller can be used to eliminate disturbances. The disturbance or noise we will add here will follow a uniform distribution, i.e. you have equal chance for all values.

Add a uniform disturbance of range $\pm 0.4 rad/s$ to your command $\dot{\theta}_c$. Make sure to turn off the delay. Plot trajectory results for three different values of Kp_{inner}, with a ratio of ten-to-one between the smallest and biggest values. Compute the average absolute error, and discuss how this average error changes with the gain Kp_{inner}. Uniform noise can be simulated in C like this:

```
double MaxVal = 0.4; // Uniform distribution noise from -MaxVal to MaxVal
double Noise = MaxVal*2.0*double(rand())/double(RAND_MAX)-MaxVal;
```

Add now a uniform noise of range $\pm 1.0 rad$ to your reading of angle θ_R. Make sure to turn off the delay, and the noise on the command. Plot trajectory results for three different values of Kp_{inner}, with a ratio of ten-to-one between the smallest and biggest values. Compute the average absolute error. Comment on how the average absolute error change with Kp_{inner}. Can the system cope with sensor noise as easily as it copes with disturbances?

3 15 pts Wall Following

The wall-following task can be thought of as a special case of line following. This is done by basically replacing the leftmost 0 in Fig. 3 by d_{wall}, while the value θ_R correspond to the angle between the heading of the robot and a tangent to the wall, or the angle with the shortest last scan, offset by $\pi/2$ (do you see why? If you can't, draw it on paper and it should become clear). Using the simulated laser scanner, make the robot follow the left hand side wall at a distance $d_{wall} = 1.0$. You should reuse the code developed for question 2 to do this task. You should get a path like shown in Fig. 5. In addition, present the results of your wall-following algorithm using the configuration files: `cave.cfg` and `autolab.cfg`.

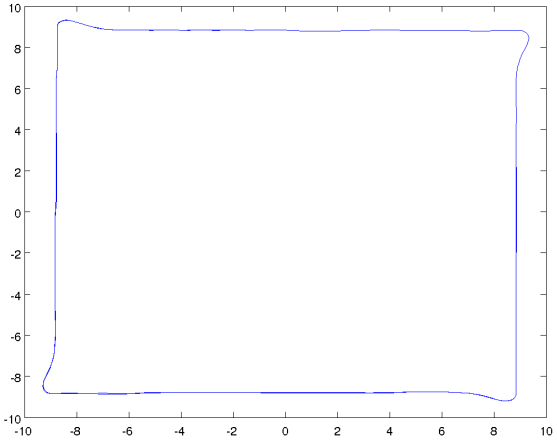


Figure 5: Path taken by the robot, using the cascade controller, when doing wall following. The robot is travelling clockwise.

4 15 pts Voronoi Graph Explorer

The controller developed to solve the trajectory following and wall following problems should be adapted to follow a Voronoi Graph edge. Following a Voronoi edge can be done by estimating the two closest obstacles, (not just points) find the mid-point, which should be used to calculate the displacement d. For orientation correction use an angle perpendicular to the line that connects the two closest obstacles. For ease of use develop first a function *Access* which guides the robot away from the closest obstacle until the robot is equidistant from two obstacles. Filter the laser points using the distance to the closest obstacle plus ϵ in order to identify distinct obstacles. When the robot approaches a 3-equidistant point the controller should switch randomly to a new Voronoi edge. Record the trace of the robot's position and presented as in Fig. 5.

Present results from the `cave.cfg`, `autolab.cfg`, and `hospital1.cfg` configuration files.

5 Hints

Direction

If your system diverges, it might be because the commands have the wrong direction. Double check that by placing the robot near the line, 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 for line P_1P_2

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 °, but the same applies for rad):

- 0° - 360° = -360° (wrong) → -360° + 360° (phase unwrap) = 0° (correct)
- 170° - -180° = +350° (wrong) → +350° - 360° (phase unwrap) = -10° (correct)

Capping the Heading Command

If the robot is far from the line, the heading command θ_c might be more than $\pm\pi/2$ rad, in which case the robot will go in the wrong direction, or start making spirals. You might therefore want to cap this command θ_c to be within $\pm\pi/2$. This way, even if the robot is initially placed heading in the wrong direction, it will turn back towards P_2 of the current line segment, and travel at an angle of $\pi/2$ towards the line.