CSE 298 – Homework 4

Wall Follower & Feedback Control

Due: 7/30/2020 by end of day

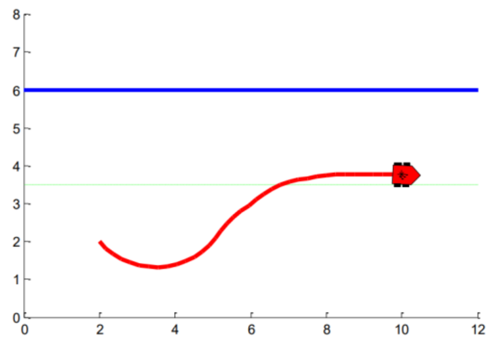

Figure 1: LS Wallfollower trajectory under PD controller

Robot Model:

1. Again, we are using a differential drive kinematic model.
2. Assume your robot is equipped with a ring of 30 sonars spaced in 12 degree increments [pi/15:pi/15:2*pi].
3. Assume a constant robot velocity of 0.5 m/s
4. Assume a MAXIMUM angular velocity of 0.25 rad/s
5. The sensor update rate is 2Hz
6. The maximum sensor range is 10 meters.
7. Assume a left-handed wallfollower (like in Figure 1) with a desired standoff distance from the wall (blue line above) of 2.5 meters (dashed green line).

Sonar Sensor Model:

1. In this problem, you will generate a sensor model for sonar measurements.
2. Assume that your sonar has a maximum range of 10 meters.
3. Write a function

   r = DetectWall(robot, a_w, b_w)

   that takes as an argument the robot struct and the slope/intercept (a_w,b_w) of the wall in the world frame, and returns an ARRAY of range measurements from the sonars. In effect, this function returns "ideal" sonar measurements. Note that the order of sonars should be from [pi/15:pi/15:2*pi] in the robot frame. Also, only sonars that are within 10 meters of the wall should return an appropriate value. If the true sonar range would be > 10 meters, you should return a -1 in the respective array index.
4. We will now perturb the true wall measurements to model both random noise and outlier measurements.
   a. Assume that the range measurements are corrupted with random zeromean Gaussian noise with standard deviation = 0.01ρ, where ρ is the actual range to the target.

    b. Further assume that 25% of the sensor measurements suffer from multipath, resulting in a range measurement error of +20-50%. You may assume that these measurements are uniformly distributed.
5. You are to implement a function rHat = FireSonar(r) that:
    a. Take the ARRAY of true ranges from DetectWall as input.
    b. Tests to see if each range was affected by multi-path error. If so, the appropriate random error is added to the actual range measurement and returned from the sensor model as rHat.
    c. Otherwise, the range measurement is corrupted with appropriate random Gaussian noise and returned as rHat.
    d. Under no circumstances should the range returned exceed the maximum detection range of the sensor. For ranges >10 m, the sonar will return a value of -1 (indicating that no target was detected).
6. Helpful functions:
    a. rand
    b. randn

Least Squares Line Fitting:

1. For this problem, we will be using the least-squares line fitter from page 20 of the Robust Estimation lecture notes.
2. Implement the following functions in Matlab

    ```
    [ a_r, b_r] = LS_Line( rho, alpha );
    [ a_w, b_w ] = Transform_Line( robot, a_r, b_r );
    ```

    LS_Line takes an array of range measurements rho and an array of bearing angles alpha and returns the least-squares solution [a_r, b_r] that corresponds to the slope and y-intercept line parameters in the robot frame. The function Transform_Line takes these parameters and robot struct as inputs and transforms these to parameters in the world frame. This will be useful for plotting lines in the world frame. NOTE: You may solve this using the solution in the notes or pseudo-inverse operation. However, you MAY NOT solve this using a built-in Matlab LS function.

Least-Squares & RANSAC Wallfollowers:

1. You are to implement a least-squares wallfollower LS_WallFollower(kP, kD, y_wall) based upon the LS_Line function above. kP and kD correspond to the proportional and derivative gains for your controller. This will control the relative position and orientation of your robot with respect to the wall described in Figure 1. y_wall denotes the y-intercept of the wall. You may assume that it is always parallel to the x-axis.
2. To regulate the position and orientation of the robot, you will implement the PD controller discussed in class during the feedback control lectures. You will need to choose appropriate gains for the controller design.
3. You will first estimate the relative position and orientation of the robot with respect to the wall. These inputs will be fed to the PD Controller which in turn will generate an output (ω) that will be used to control the orientation of the robot.

4. To estimate the position/orientation with respect to the wall, you will:
    a. Determine the range from each sonar to the wall using DetectWall and pass the appropriate range array to FireSonar where they will be corrupted with noise.
    b. From the corrupted range measurements, form an estimate of the position and orientation for the wall relative to the robot by employing your LS_Line function. From this, you can infer the relative distance and orientation of the wall for use in your PD controller. The output of your PD controller will yield ω. Since v is constant, you can move the robot appropriately. Note that robot motion is "perfect", and updates occur at 2Hz.
    c. The initial robot position will be taken as input from the user mouse (see the ginput function). The initial orientation will be generated at random from $\theta \in [-\pi/4, \pi/4]$.

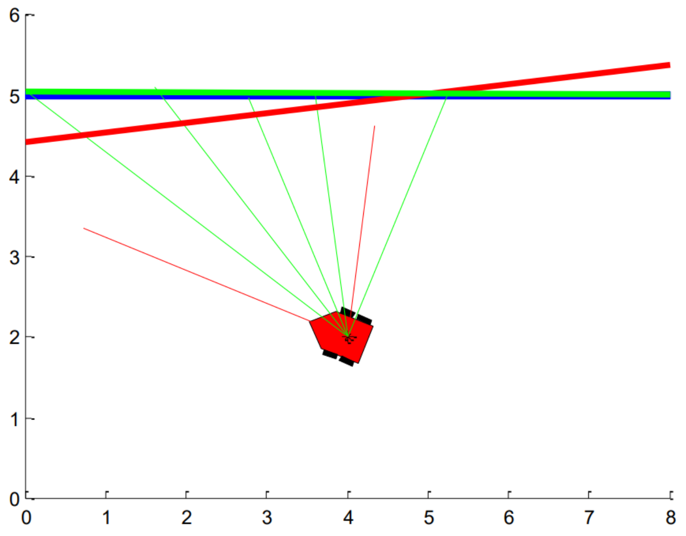

Figure 2: RANSAC Wallfollower showing least squares (red) and RANSAC solutions (green).

5. Repeat steps 1-4 by implementing a RANSAC_WallFollower(kP,kV) that will filter outliers from the sonar measurements.
    a. Determine the maximum number of sample trials n_max that you need to run RANSAC to ensure you have a 99.9% probability of choosing the correct line (you may assume a 25% outlier rate, although this is technically not quite correct).
    b. Use the subset of sonar measurements that yields the largest consensus set after n_max trials to fit the wall. In your figures, sonar measurements that were used above should be drawn green, while outliers are drawn red.
6. Also, for both the LS and RANSAC wall followers, redraw the new wall as shown in Figure 2 where the actual wall (blue), RANSAC wall (green) and LS wall (red) are all shown.
7. When you have determined that both implementations are performing properly, save images from sample trial runs with identical initial configurations for both wall followers (Figures 1-2).

Additional Questions:

1. For the following questions, set the outlier rate to 0 and the random noise to 0 in FireSonar – in other words, all the measurements from Fire Sonar should be "perfect."
    a. Run your LS_WallFollower function for a variety of gains that exhibit critically damped behavior, i.e., kd = 2(kp) 1/2 . What gains did you find were the most effective?
    b. Repeat this exercise for under-damped and over-damped gain sets. Was the behavior as expected? Show figures for each instance in your report.
    c. What happens if you set kP=0 for the controller? What about kD=0? Why?
    d. Increase the initial error in orientation for the robot beyond those covered for the design trials. What do you observe about the controller performance? Are there configurations where the controller will fail to converge to the proper distance from the wall? If so, explain why these occurs.
    e. Increase the maximum angular velocity of the robot from 0.25 to 0.5 rad/s. How does this affect controller performance?
2. Discuss differences in the performance of the LS vs. RANSAC solutions, to include convergence properties, settling time, overshoot, etc.
3. You run a series of experiments with the PD controller on an actual robot and find that the robot is always staying further away from the wall than desired. How might you change the controller to improve its performance?

Turn in

A write-up, to include images from all simulation trials, answers to all questions, as well as your Matlab source code.