E160 Lab 03

Point Tracking Spring 2018

1. INTRODUCTION

Robots often need to move to a point with a desired orientation. This can be difficult when the robot has nonholonomic constraints. The robot cannot simply move laterally, making it difficult to come up with a controller using intuition.

The goal of this lab is to get students to implement a closed loop controller that will drive the robot to any desired state (position and orientation). Odometry will be the sole method used for estimating the robot's state (i.e. localization). This method of localization will be replaced in the next lab.

2. BACKGROUND

As shown in class, a controller has been developed based on the following coordinate transformation:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$
$$\alpha = -\theta + atan2(\Delta y, \Delta x)$$
$$\beta = -\theta - \alpha$$

Once the new variables have been calculated, desired forward velocity v and desired rotational velocity w can be calculated. Note, control gains are defined at the top of `E160_robot.py`, but they may not be optimal values.

$$v = k_\rho \rho$$
$$w = k_\alpha \alpha + k_\beta \beta$$

Using v and w, we can determine the desired wheel velocities $\dot{\varphi}_1$ and $\dot{\varphi}_2$. The following equations were derived and are used.

$$\omega_1 = \frac{r\dot{\varphi}_1}{2L}$$
$$\omega_2 = -\frac{r\dot{\varphi}_2}{2L}$$
$$w = \omega_1 + \omega_2$$
$$v = L(\omega_1 - \omega_2)$$

Recall that this controller works well if the goal point is in front of the robot, that is if α lies between $-\pi/2$ and $+\pi/2$.

However if the goal is behind the robot, then modifications to the controller are required to give shorter more direct paths involving the robot moving in reverse. That is, we first redefine the transformation as:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$
$$\alpha = -\theta + atan2(-\Delta y, -\Delta x)$$
$$\beta = -\theta - \alpha$$

We also redefine the control law to have the robot work in reverse.

$$v = -k_\rho \rho$$
$$w = k_\alpha \alpha + k_\beta \beta$$

When implementing this controller, make sure your robot never exceeds the maximum allowable velocity of `self.max_velocity` = *0.05 m/s*, and that controller gains must satisfy the necessary conditions for stability.

3. EXPERIMENTS

Use the most recent version of the base code from the course website. Note, you will have to copy and paste your odometry over from lab 2. All coding for steps a) through f) will occur within the function `point_tracker_control()`. Note that this function will be called from `update_control()` at each iteration of the control loop. Also note that clicking the "Track Point" button on the GUI will set the robot's desired x, y, theta values stored in `state_des` to be the values in the three text boxes on the upper left part of the GUI screen. Clicking the Track Point button also resets the `point_tracked` variable to be `False`.

a) Transformation to a new coordinate system

Using the robot state estimate `state_est`, and the desired state `state_des`, calculate the position of the robot $\Delta x, \Delta y$ relative to the goal position.

Now use the equations above to calculate the state variables `pho, alpha, beta` in the new coordinate system. Remember to check if the goal is behind the robot and recalculate the variables if necessary.

It is often a good idea to make sure that ALL angles lie within $-\pi$ and π. We have created a function called `angle_wrap` to help with this.

b) Calculate Desired Wheel Velocities

Now that the transformation is complete, implement the control law to determine the desired velocities v and w, respectively represented by variables `desiredV` and `desiredW` in your code. You can experiment with different gain values `Kpho, Kalpha,` and `Kbeta`. Remember to reverse direction if the goal is behind the robot.

From desired robot velocities v and w, you can calculate the desired wheel velocities `desiredRotRateL` and `desiredRotRateR`. You may need some of the equations above.

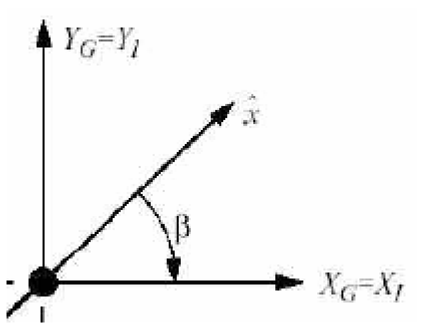
Keep good track of your units. It may be helpful to understand there are variables `encoder_resulution` and `encoder_per_sec_to_rad_per_sec` defined and set for you.

You can check if the robot is close enough to the desired state and zero the desired wheel velocities.

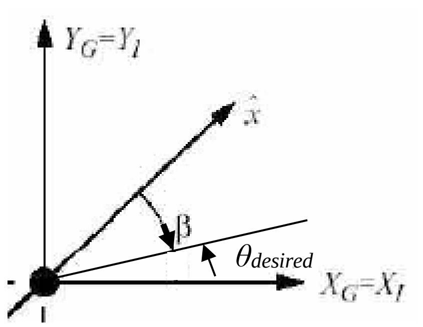
c) Track Desired Positions
Run the application in Simulation mode and set the desired state to be [0.75 0 0]. Click the "Track Point" button. The robot should move forward $0.75\ m$. Now try tracking the point *[0 0 0]*. The robot should return to the origin.

Keep trying new points to track, making sure the robot always moves to the desired locations. At this point the robot will always end with orientation 0 degrees.

d) Track desired positions and orientations
To make the robot track desired orientations, the state variable β must be modified to include the desired orientation θ_{des}. Simply adding θ_{des} to β will force the robot to track the desired orientation, (See figure below).



Old Definition:



New Definition

Now test the controller for many desired position/orientation combinations in *Simulation* mode.

e) Velocity Limits
Before having the `point_tracker_control()` function return variables `desiredWheelSpeedR` and `desiredWheelSpeedL`, be sure they travel no faster than `max_velocity` m/s. Double check your point tracking code still works after you implement this.

f) Hardware testing
Set your robot in hardware robot and make sure your point tracker still works. You may notice that as the robot gets close to the desired state, it may jitter. To accommodate this, implement a case check that sets point_tracked to be true if the robot is "close enough" to the desired state. Also, it helps to use a separate controller once the robot is within epsilon meters of the desired position. The second controller should simply robot on the spot to track the desired position using P control.

g) Tracking trajectories
Using the point tracker you just developed, implement a path tracker that enables the robot to autonomously follow a hard coded path, (this may be useful when your motion planner autonomously constructs a collision-free trajectory).

Hard code a trajectory that includes at minimum straight line path segments.

Note that you will need to create your own function that iteratively calls the point_track function.

4. DELIVERABLES

a) Demonstration
Before the end of the final day of this lab, you must demonstrate to the Instructor that your point tracker (not path tracker) is working properly. In both simulation and hardware mode, the 2D graphics window should show the robot and estimate moving towards desired goal states. Videos are not acceptable for the simulation part of the demo, the instructor will drive the robot in simulation and try to break it!

Part of your grade will be based on performance: How stable is the controller on the real robot, how close does it come to desired states, etc. Demos are due by 4:00pm Sunday, March 4th.

b) Submissions
In a 5-10 page report, present your methods and results for both point tracking and path tracking. Be sure to include the following sections: abstract, introduction, background, problem definition, control design, results, conclusion. Performance plots, indicating point tracking data that illustrates the path taken as well as tracking error are required.

Note, all lab documents in this class will follow the template found at:
http://www.ieee.org/conferences_events/conferences/publishing/templates.html

The report is due 4:00 pm, Sunday, March 4th.