COS 495 – Autonomous Robot Navigation

Lab 6

Point Tracking

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).

BACKGROUND

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

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

Once the new variables have been calculated, desired forward velocity *v* and desired rotational velocity *w* can be calculated. Note control gains are defined in the `Robot.h` header file.

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

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.

$$\omega_1 = \frac{r\dot{\varphi}_1}{2L}$$

$$\omega_2 = \frac{-r\dot{\varphi}_2}{2L}$$

$$w(t) = \omega_1 + \omega_2 \qquad v(t) = L(\omega_1 - \omega_2)$$

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.

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

Remember that when implementing this controller, make sure your robot never exceeds the maximum allowable velocity of 10 cm/s, and that controller gains must satisfy the following condition:

$$k_\rho > 0 \qquad k_\beta < 0 \qquad k_\alpha - k_\rho > 0$$

EXPERIMENTS

Note, use the most recent version of your code from lab 5. All coding for steps 1 through 5 will occur within the function `Robot :: TrackPoint(CWiRobotSDK* m_MOTSDK_rob)`. Recall that this function will be called for each iteration of the 20 Hz control loop. You will need your odometry localization code from lab 4. You will not need any wall positioning or odometry error characterization code.

1. **Transform to new coordinate system**

   Using the robot state estimate [`x_est y_est t_est`], and the desired state [`desiredX desiredY desiredT`], calculate the position of the robot relative to the goal position *Δx, Δy*.

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

   It is often easiest to make sure that all angles lie within $-\pi$ and π.

2. Calculate Wheel Velocities

Now that the transformation is complete, implement the control law to determine the desired velocities `desiredV` and `desiredW`. You can experiment with different gains later, but for now use `Kpho`, `Kalpha`, and `Kbeta`. Remember to reverse direction if the goal is behind the robot.

From these you can calculate the desired wheel velocities `desiredWheelSpeedL` and `desiredWheelSpeedR`.

Remember to convert these desired wheel velocites from m/s to encoder pulses / second before sending them to the robot.

You can use the `pointTracked` variable to determine if the robot is close enough to the desired state and should return to manual mode, or if the desired wheel velocities should send the commands to the robot using the function:

```
m_MOTSDK_rob->DcMotorVelocityNonTimeCtrAll
```

Part of this is coded for you in the base code.

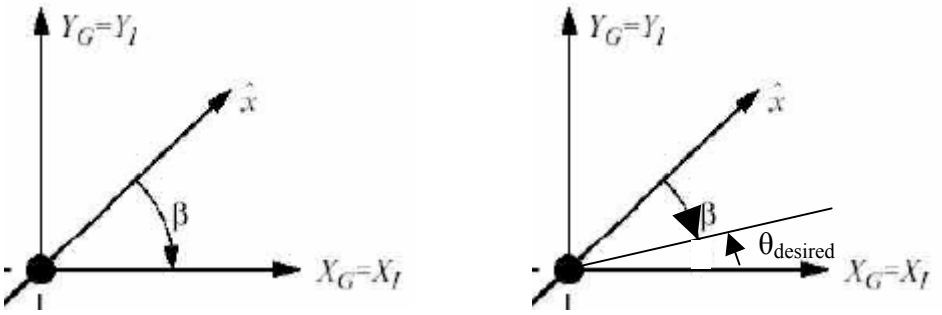
3. Track Desired Positions

After building and running the application, select *Simulation* mode. Try entering [1 0 0] in DesiredX, DesiredY, and DesiredT fields located under the *Track Point* button. Click on the *Track Point* button. The robot should move forward 1m. 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.

4. Track desired positions and orientations

To make the robot track desired orientations, the state variable β must be modified to include `desiredT`. Simply adding this 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.

5. Tracking points with the actual robot

Once you are confident the controller is working in Simulation mode, add some extra code to make sure the robot will work with the controller.

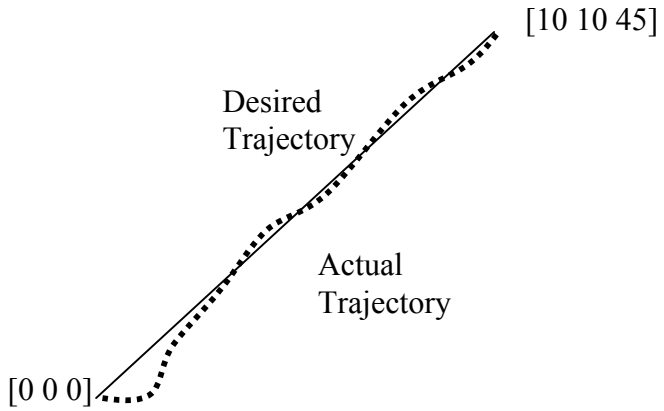
First, do a check to make sure the robot is not moving faster than 10cm/s. Next, it might be good to implement a simple P control for the case when the robot should turn on the spot. Just use an "if" statement to determine if ρ is close to 0.

6. Tracking trajectories

Using the point tracker you just developed, implement a straight line trajectory tracker that operates as follows:

When the Track Point button is pressed, the robot should follow a line that connects its current position to the position described by desiredX and desiredY. The desired orientation throughout the trajectory should be in the same direction as the line.

For example, if the robot is at 0,0,0, and the desired position is 10,10, then the desired trajectory might look something like in the figure below:



Think about whether your method word work with a curved trajectory since this might be required later. Also try to make the robot follow the trajectory smoothly without stopping and starting along the way.

To code this up, you can use the function `Robot :: TrackTrajectory(CWiRobotSDK* m_MOTSDK_rob)` located in Robot.cpp. You will have to make sure the control thread calls this function instead of `Robot :: TrackPoint`. I have also included several variables that will be of use: `x_start, y_start, x_goal, y_goal, t_goal.` See how they are set in the function `CRoboticsLabDlg::OnTrackPoint()`.

DELIVERABLES

1. Demonstration

Before the end of the final day of this lab, you must demonstrate to the Professor that your point tracker/trajectory tracker is working properly. In both simulation and X80 mode, the OpenGL window should show the robot and estimate moving towards desired goal states.

Part of your 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 Friday Novemer 18th.

2. Submissions

In a 5-10 page report, (see Lab 3 for report format requirements), present your method for tracking a straight line trajectory. Plot or sketch out some trajectory results and discuss any pros or cons of your method. The report is due Friday November 25th.