Lab 3: Inverse Kinematics

CSCI 3302: Introduction to Robotics

Report due 10/13/20 @ 11:59pm

The goals of this lab are to

- Understand how to calculate the wheel speed given a desired velocity and/or position
- Implement basic feedback control
- See the need for higher-level reasoning / path-planning

You need:

- Webots
- CSCI3302_Lab3 downloaded from the course Canvas

Overview

A robot's forward kinematics allow you to compute the final position of a robot given individual joint positions. If we wanted to be able to perform the opposite process – figuring out the joint positions required to move a robot to a desired position/configuration, we will need to utilize *inverse kinematics*. A robot's inverse kinematics provide a joint configuration that will achieve a desired pose (in ePuck's case: x, y, θ). In this lab you will implement a feedback controller that navigates the robot from its current position to a provided goal position.

Instructions

Each group must develop their own software implementation and turn in a **single report** that contains:

- The code (1 file)
- One lab report in PDF format. Please put the number of each question next to your answers, rather than turning in your answers as an essay

If your group does not finish the implementation by the end of the class, you may continue this lab on your own time as a complete group.

Part 1: Compute the Inverse Kinematics of ePuck

Use the forward kinematic relationship for a differential wheel platform from the book ($x_r{}'$ and θ_r' as a function of left and right wheel angle change) to calculate its inverse, that is, left and right wheel angle change given $\dot{x}_R$ and $\dot{\theta}_R$. **Note: you do not have to do the algebra yourself, but you should understand where the equations for $\dot{\phi}_l$ and $\dot{\phi}_r$ came from!**

Part 2: Turn-drive-turn Controller

HINT: keep track of your units! I recommend using radians, meters, and seconds as your default rotation, distance, and timing units, scaling them as necessary for the functions you use.

1. (Position Error) Calculate the Euclidean distance ρ between your current location and the goal position.

2. (Bearing Error) Calculate the angle α between the orientation of the robot and the direction of the goal position.
   For example, if the line between the robot's position and the goal position is oriented 90 degrees to the left from the robot's perspective, α is 90 degrees. If the goal is directly in front of the robot, α is 0 degrees (and so on). You should use the function atan2() for doing this to avoid divide-by-zero and needing to figure out which quadrant your solution is in. (Python's atan2 returns radians)

3. (Heading Error) Calculate the angle η between the orientation of the robot and the goal orientation.

4. In the **turn_drive_turn_control** state:
   Using <left/right>Motor.setVelocity(), create a controller that rotates in place until the robot is facing the goal position (reduce bearing error to zero), drives forward to the goal position (reduce position error to zero), then rotates in place to orient to the proper heading (reduce heading error to zero). Make sure the odometry code is updating properly!

5. Verify that your controller works by moving the target on the ground to a new position!

Part 3: Feedback Controller

This portion of the lab is more open-ended and challenging than previous labs. Be sure to ask questions as they come up so you have plenty of time to overcome any difficulties that you encounter.

1. Calculate the necessary change in robot position x_r' that is proportional to ρ
   What happens if you don't set/enforce a maximum value that x_r' can take in your IK code? Will your robot ever be able to rotate if $x_r' \gg \theta_r'$? Keep in mind that $\dot{\theta}_r$ will never be more than 2π.

2. Calculate the necessary change in robot rotation θ_r' that is proportional to α and η
   What should you do to α as the robot reduces its position error ρ near to 0?
   Should you weight it more or less heavily if ρ is large?

3. Create a proportional feedback controller that uses your error signals to compute the wheel rotations needed to make the position and rotation changes for driving to a given goal. There are many valid ways to solve this problem! Describe your solution in your report.

You will likely have to update the odometry code to accommodate this new method of moving the robot! I recommend just using left_wheel_direction and right_wheel_direction to hold the percentages, which should be in the range [-1.0, 1.0].

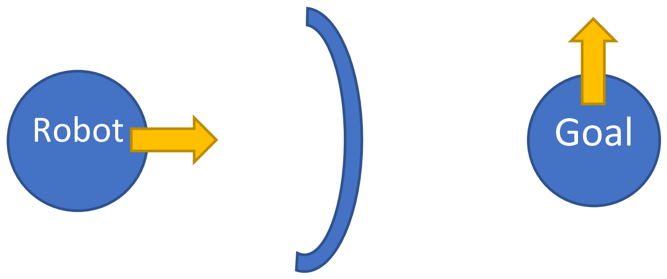
Keep in mind that the robot can only rotate its wheel a small amount during the time between loop executions, which is almost always going to be a value less than ϕ_L' and ϕ_R'. You may wish to scale the velocity argument for each motor.setVelocity function call such that they are proportional to the ratio between ϕ_L' and ϕ_R'. (e.g.,: If $\phi_L' = 1.0$ and $\phi_R' = 2.0$, Right motor should run at 100% and left motor should run at 50% of maxVelocity). I recommend scaling down the overall speed of your system so that the robot isn't always trying to move at EPUCK_MAX_WHEEL_SPEED (i.e., setting motor velocity to motor.getMaxVelocity()) so that you don't encounter too much wheel slip.

4. Set some kind of stopping criteria, such that your controller stops if your distance and heading error are less than some (small) value, e.g., a few cm or a few degrees off-target.
5. Verify that your controller works by moving/driving to the target on the map!

Part 4: Lab Report

Create a report that answers each of these questions:

1. Describe one condition where your controller from Part 2 might fail.
2. What is the role of the position error?
3. What is the role of the heading error?
4. Why include bearing error?
5. Describe how you implemented your feedback controller in Part 3.
6. Does your implementation for part 3 work? If not, describe the problems it has.
7. What are the equations for the final controller from your implementation? What are your equations for $\theta_r'\ and\ x_r{}'$ in your feedback controller?
8. What happens if you alter your gain constants (the weights for each error term)?
9. What happens if you increase these gain constants? What if they become too large?



10. What would happen if an obstacle was between the robot and its goal?
11. (Briefly) How would you implement simple obstacle avoidance using the light sensors on ePuck?
12. Describe what would happen if the obstacle-avoiding robot encountered a U-shaped object like this:
13. Roughly how much time did you spend programming this lab?

14. (*Optional, confidential, not for credit- delivered via e-mail to Prof. Hayes*) A brief description of any problems (either technical or with respect to collaboration) that came up during the lab that you'd like me to be aware of.