Lab 3: Inverse Kinematics

CSCI 3302: Introduction to Robotics

2 pts (2%) per day extra credit for early turn-in (up to 10pts)

The goals of this lab are to:

- Understand how to calculate the necessary wheel rotations to reach a desired position,
- Implement a feedback controller,
- Better understand the need for higher-level reasoning, path-planning, and mapping.

You need:

- Webots R2022b or above
- CSCI3302 Lab 3 Base Files downloaded from the Lab 3 Assignment on Canvas

Overview

A robot's forward kinematics allow you to compute the position of a robot (or robot end effector) 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 case of a differential wheel platform such as the E-Puck or the Turtlebot: x, y, θ). In this lab you will implement a feedback controller that drive the robot from its current position along a list of waypoints until it reaches its final target.

Instructions

Read the entire instructions before diving into the code (also read all the code comments while walking through the code)

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, please continue this lab on your own time as a group.

Part 1: Compute the Inverse Kinematics of a Differential Wheel Platform

1. Find a spec sheet for the Robotis Turtlebot 3 Burger robot online. Consult the robot's specifications sheet to determine the Turtlebot's maximum speed (m/s), maximum motor speed (rad/s) [nb: **not** the maximum rotation rate of the robot itself], and axle diameter (m). You may have to print the maximum motor speed using Webots'

getMaxVelocity() function with the left or right wheel motor (e.g., some_turtlebot_motor.getMaxVelocity() to get MAX_SPEED in rad/secs). Put this information into the constants defined at the top of your controller file.

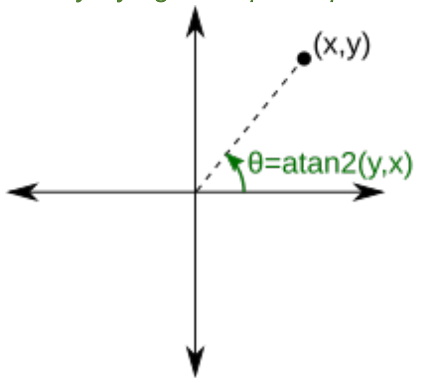
2. Use the forward kinematic relationship for a differential wheel platform from the book ($\dot{x_R}$ and $\dot{\theta_R}$ as a function of left and right wheel angle change) to calculate its inverse, that is, calculate the desired left and right wheel speed $r\dot{\phi}_l$ (vL) and $r\dot{\phi}_r$ (vR) in **meters per second** given $\dot{x}_R$ and $\dot{\theta}_R$. **You don't need to know the robot's wheel radius**, just work with wheel speeds in meters per second directly as you did in the Odometry lab or Q7(c) on HW1 (as a proportion of the maximum rotational velocity of the motor). Alternatively, the ratio of MAX_SPEED_MS/MAX_SPEED is the radius value.
(Note that you don't have the values of $\dot{x_R}$ and $\dot{\theta_R}$ yet, they will be obtained after STEP 2.1 and STEP 2.2. You can use just initialized variables for STEP 1.)
Implement these IK functions where it says STEP 1 in the controller assuming you have access to $\dot{x}_R$ and $\dot{\theta}_R$.

Part 2: Feedback Controller

1. (Position Error) Calculate the Euclidean distance ρ between your current pose and a goal pose. Implement this under STEP 2.1 in your controller.
Verify that your error is what you expect by trying a couple of points out.

2. (Bearing Error) Calculate the angle α between the orientation of the robot and the direction of the goal position. You should use the *atan2* function for this, which will work in all four quadrants. The bearing error is the angle between the robot's current orientation and the goal location. If the robot is already looking at the goal, the error should be zero. Implement this under STEP 2.1 in your controller.
Verify that your error calculation works by trying a couple of points in front of and behind the robot.

3. (Heading Error) Calculate the angle η between the orientation of the robot and a goal orientation. If the robot is already facing the goal pose's direction, the error should be zero. Implement this under STEP 2.1 in your controller.
Verify that your error calculation works by trying a couple of points in front of and behind the robot.



4. Calculate $\dot{x_R}$ and $\dot{\theta_R}$ from ρ and α, η respectively. These functions will define your "feedback control" law. Start with very simple expressions, such as $\dot{x_R}$ being simply proportional to ρ and $\dot{\theta_R}$ being proportional to α. You are welcome to include conditional logic to switch how you handle these error signals based on their current values. Implement this under the comment for STEP 2.2. *Verify that the robot turns in the correct direction first (otherwise you have a sign error along your angle calculation), then check whether it approaches the goal.*

5. By now you should have used your sources of error together with your feedback control law to compute $\dot{x_R}$ and $\dot{\theta_R}$. You should already have the robot's wheel rotational velocities (vL and vR) needed to head toward

the goal as a function of $\dot{x_R}$ and $\dot{\theta}_R$(i.e., apply the inverse kinematics equation results, but as velocities) from STEP 1.

6. Implement logic to scale your motor speeds such that they move **proportionally** to their target velocities. If the right wheel needs to rotate twice as much as the left wheel, then the right wheel could be set to MAX_SPEED and the left wheel could be set to half of that value. This will remove the problem of our robot not being able to rotate its motors far enough in the 32ms timestep that passes with each iteration of the main control 'while' loop. Do this under the comment for STEP 2.3. Make sure to set your robot's motor speeds to these values at the bottom of your while loop! *This will require some careful consideration if you would like your robot to be able to have negative wheel rotation as well!*

7. Test whether your wheel rotation speeds exceed MAX_SPEED in either the positive or negative direction and clamp the values at +/- MAX_SPEED if they do. Implement this under the comment for STEP 2.4. Note that if you correctly perform STEP 2.3, then this step will just act like a safety check.

8. Create a stopping criteria, such that your controller stops (set motors to velocity 0) if your distance error is less than some (small) value (e.g., a few cm off-target) and your heading error is less than some reasonable value (e.g., $\frac{\pi}{6}$ radians in either direction). *The yellow circle on the ground is centered at [1.5m, 1m] relative to the robot's starting position. Feel free to move this around to test your controller.*

9. Create an array with goal (x,y) locations ("waypoints") and use these with your stopping criteria to replace the robot's current goal as it approaches a target. You can use however many waypoints you want. For example - We used 6 waypoints in our solution (This should be done before the While loop). *You do not have to manually calculate your waypoint headings (each waypoint should have a heading that points in the direction of the next waypoint), as you can calculate it by finding the bearing error between the currently targeted waypoint and the one that follows. The final waypoint heading does not matter.*

10. Verify that your controller works by moving/driving the Turtlebot robot ***into the kitchen*** **without hitting any obstacles along the way.** Your waypoints should be chosen in such a way that you avoid all obstacles and move through the free space. You should pass through the kitchen door and enter the kitchen, landing somewhere on the green target cylinder visible on the ground.

Part 3: Lab Report

Create a report that answers each of these questions:

1. What happens when you set the "gains" (source-of-error coefficients) of your controller too low? Describe your observations both for distance and bearing.
2. What happens when you set the distance threshold for your stopping criteria (selecting the next waypoint from your list) too low? What happens when you set it too high?
3. Describe how you implemented your feedback controller in Part 2.
4. Does your implementation for Part 2 work? If not, describe the problems it has.

5. 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?
6. What happens if you do not limit the wheel speeds to positive or negative MAX_SPEED? Think about what the robot will do in this case and what your odometry will predict.
7. What would happen if an obstacle was between the robot and its goal?
8. (Briefly) How would you implement simple obstacle avoidance using distance sensors if they were placed around the Turtlebot?
9. Roughly how much time did you spend programming this lab?

10. (*Optional, confidential, not for credit- delivered via e-mail to the Professor (TAs CCed)*) 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.