Lab 3: Inverse Kinematics

CSCI 3302: Introduction to Robotics

Report due 02/24/26 @ 11:59pm

Goals:

- **Kinematics:** Derive and implement the mapping between chassis and wheel velocities.
- **Control:** Implement a turn-drive-turn controller with a continuous closed-loop feedback controller.
- **Analysis:** Visualize the divergence between odometry and ground truth.

You need:

- Webots simulation
- Lab3 (lab3.wbt and lab3.py base code) downloaded from the course Canvas
- Python libraries: numpy and matplotlib

Overview

A robot's forward kinematics allow you to compute the final position of a robot given individual joint position. 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 submit a single report containing:

1. Controller code lab3.py (1 file).
2. One lab report in PDF format. (Please number your answers to correspond with the questions below, rather than writing an essay).

Note: If your group does not finish the implementation during class, you may continue this lab on your own time.

Part 1: Inverse Kinematics

1. Write out the inverse kinematics equations for a differential wheeled robot. Start with the forward kinematics equation (Eqn. 3.340 from the book). **Solve for $\dot{\phi}_l$ and $\dot{\phi}_r$ as functions of $\dot{x_R}$, $\dot{\theta}$, r and d and include this in the write up.** You can handwrite this part.

This does not allow us to deal with the case when our desired y velocity is in the robot's frame $\dot{y_R} \neq 0$.Therefore, we require trajectory tracking rather than inverse kinematics alone.

2. What are some of the ways to generate a trajectory? **Discuss some ideas with your teammates and include them in the write-up.** This should be brief.
3. For this lab, you will define the waypoints of the trajectory. Find waypoints that allow the robot to navigate counterclockwise around the arena and stay close to the black line. You can hold down shift to drag the ping pong marker around. Note the translation values and add them to the list of waypoints. The ping pong marker will be used to track the waypoints the robot is reaching for. Note that the global coordinates are different from lab 2. You can show the coordinate system in the simulator by going to View -> Optional Rendering -> Show Coordinate System.
4. **Setup:** Use numpy's atan2 and sqrt functions. We will be using the gps and compass to determine the odometry. We have also changed the robot to a supervisor node so that we can control the position of the ping pong ball. Go over the base code to familiarize yourself with the changes.

Setup Notes:

- Use numpy.atan2 and numpy.sqrt.
- We will use the GPS and Compass nodes to determine odometry (ground truth).
- The robot has been converted to a **Supervisor** node so we can control the position of the ping pong ball marker. Review the base code to familiarize yourself with these changes.

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.
   The angle should be positive if the robot has to turn left to face the goal and negative if the robot has to turn right.

3. (Heading Error) Calculate the angle η between the orientation of the robot and the goal orientation.
   Copy over your line following code and print out the error terms as the robot does line following. Do the

error terms go down as the robot gets closer to the waypoints?

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), drives forward to the goal position (reduce position error), then rotates in place to orient to the proper heading (reduce heading error).
   Clip velocity values so that the motors don't go below –MAX_SPEED or above MAX_SPEED.

5. You might run into an issue where the value of α alternates between a positive and negative value, causing the robot to turn right and then left, never reaching the desired orientation. What do you think causes this? Look at the atan2 function, where are the discontinuities? How do you ensure that the values don't alternate between the discontinuities?

Part 3: Proportional Controller

1. Create a **proportional_controller** state. Calculate the necessary change in robot position $\dot{X_R}$ that is proportional to ρ. Calculate the necessary change in robot rotation $\dot{\theta_R}$ that is proportional to α and η. Set values for left and right wheel motors accordingly.
2. Create a proportional feedback controller that uses the inverse kinematics equations with 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.
3. Replace the odometry with the code you had for lab 2. Does your controller still work? You don't have to fix it if the controller breaks but discuss what issues you face in the report.

Part 4: Lab Report

Create a report that answers the following:

1. Part 1.1, 1.2, 2.5, and 3.2.
2. What is the role of the position error?
3. What is the role of the heading error?
4. Why include bearing errors?
5. What are some of the issues you encountered for part 3? What are some of the issues you encountered when using odometry from lab 2?
6. What are the equations for the final controller from your implementation? What are your equations for your feedback controller?
7. What happens if you alter your gain constants (the weights for each error term)?
8. What happens if you increase these gain constants? What if they become too large?
9. (Briefly) How would you implement simple obstacle avoidance using the light sensors on ePuck?
10. Roughly how much time did you spend programming this lab?
11. (*Optional, confidential, not for credit- delivered via e-mail to Prof. Roncone*) 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.