Lab 2: Odometry

CSCI 3302: Introduction to Robotics

Report due Tuesday 9/29/19 @ 11:59pm

The goals of this lab are to

- Implement the forward kinematics of a differential wheel robot on the ePuck platform
- To experience and quantify sensor noise from wheel-slip, discretization, and timing violations
- To understand the concept of "loop closure"

You need:

- A functional Webots development environment
- The Lab 2 base environment folder

Overview

Odometry integrates the wheel speeds of your robot to calculate its 3-dimensional pose (x, y, θ) on the plane. In order to do this, you need to write down the forward kinematics of your robot. As your robot is non-holonomic, you need to calculate the velocity along the robot's x-axis and y-axis, as well as the rotational speed around its z-axis. You then need to transform these velocities into a global coordinate frame and add them up to calculate the robot's pose. You will find out that you cannot do this without error as the robot slips, time discretization introduces inaccuracies, and the robot's processor might not be fast enough to perform calculations while controlling its motors in real-time.

Instructions

Each group must develop their own software implementation and turn in a lab report. **You are encouraged to engage with your lab partners for collaborative problem-solving**, but are expected to understand and be able to explain everything in your write-up. 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.

Please ensure at least **one group member** submits the following to Canvas:

1) A PDF write-up answering each question from Part 4 (please put the number of each question next to your answers, rather than turning in your answers as an essay)
2) A zip file containing your entire Webots World folder for Lab 2

Note – The questions in Parts 1-3 are to help with your understanding of the lab. You do not need to answer these in your lab report.

Part 0: Loading Lab 2

To load the Lab 2 world, open Webots, go to File > Open World, navigate to where you extracted the CSCI3302_lab2 zip, enter the Worlds directory, and find CSCI3302_lab2.wbt.

Part 1: Measuring Wheel Speeds

1. Note the ePuck robot's pose at the Start Line on the floor, and record its x,z coordinates by looking at the "translation" field of the e-puck.

2. Set the robot's translation, subtracting 0.1m from the z-axis.

3. Set the controller's state to "speed_measurement" near the top of the controller code file

4. Fill in the code for the "speed_measurement" state according to the following requirements:
   a) Set the motor velocities to leftMotor.getMaxVelocity() and rightMotor.getMaxVelocity().
   b) If this is the first loop through the main control, set a the variable named measurement_start_time to the current simulation time using the robot class's getTime() function.
   c) Otherwise, if all three of the IR sensors under the robot detect a black line (signifying that you've reached the 'start' line on the map), set left and right motor velocity to 0, print out the difference between the current simulation time and the *start_time* variable, and set state to "none". This is how long it took your robot to traverse the distance you moved it.

5. Using this timing information and the distance you measured, compute the speed (in meters per second) of the robot's wheels and store this value in the global variable EPUCK_MAX_WHEEL_SPEED. This will allow you to calculate its speed in m/s without having to measure the wheel diameter. After you have completed this step, add a state transition to the "line_follower" state so your robot's program will automatically advance to part two of the lab instead of transitioning to the "none" state.
   PERFORMANCE CHECK: Make sure you're getting a speed close to 0.13m/s

Part 2: Odometry

1. Implement control code to cause the ePuck to follow the black line on the ground within the “line_follower” state. You should use +/- <motor>.getMaxVelocity() for motor velocity values.
   a) If the center ground sensor detects the line, the robot should drive forward.
   b) If the left ground sensor detects the line, the robot should rotate counterclockwise in place. (setting motors to opposite directions)
   c) If the right ground sensor detects the line, the robot should rotate clockwise in place. (setting motors to opposite directions)
   d) Otherwise, if none of the ground sensors detect the line, rotate counterclockwise in place. (This will help the robot re-find the line when it overshoots on corners!)
   HINT: You should keep track of which way each one of your robot’s wheels is moving (e.g., forward=1, backwards=-1, or stopped=0) in a separate variable to help with part 2.2, setting it in the same places you set the velocities.

2. Implement odometry code following Chapter 3 in the textbook (p. 54-55, Eqs. 3.41, 3.42). Calculate the actual distance traversed by multiplying the wheel speed with the time that the robot actually spent moving since the last odometry update. Ensure your pose is initialized (the vector [x, y, θ]) with (0,0,0) when the controller is initialized (e.g., before the “while robot.step…” loop). Use equation 3.41 to integrate your speeds into positions.
   Hint: You’ll need to keep track of the amount of simulation time that has passed in between iterations of your controller’s while loop to know how much to update your odometry. You can use robot.getTime() to get the current simulation time.
3. Print the robot’s pose in the terminal window. What values would you expect it to show when the robot arrives at the start line at the second (the third, the fourth) time? What actually happens?

Part 3: Line Following and Error Mitigation

1. Incorporate code to use the robot’s ground sensors to identify the start line, as you did in Part 1.4.c. Implement a *loop closure* mechanism to use this information to prevent your odometry error from growing each lap.

   How could you exploit this information to reduce your error?
   HINT: If the ePuck has all 3 sensors below threshold for more than 0.1 seconds, that suggests you’re passing over the start line! (Sometimes all three will momentarily pass under threshold on a sharp corner)

Part 4: Lab Report

Create a report that answers each of these questions:

1. What are the names of everyone in your lab group?
2. What happens (in terms of the robot's behavior) during the robot.step(TIME_STEP) statement?
3. What happens if you don't properly measure the elapsed time since last odometry update? What effect does this have on your position estimation, and why?
4. What is the ePuck's average speed (in m/s) when covering the 10cm distance from Part 1?
5. In an ideal world, what should the ePuck's pose show each time it crosses the starting line?
6. How did you implement loop closure in your controller?
7. Roughly how much time did you spend programming this lab?
8. Does your implementation work as expected? If not, what problems do you encounter?