A1: Robot Control

Benjamin Kuipers and Collin Johnson

Assigned: January 12, 2016
Challenge: January 22, 2016 during lab
Report Due: January 22, 2016, 11:59pm

Purpose

The goal of this assignment is to implement control algorithms for driving the MAEbot. In this assignment, you'll get experience using odometry and laser data for closed-loop control of the MAEbot.

One point I would like to get across is that, due to cumulative error, odometry does not do a perfect job of guiding the robot around a path. This will lead into later discussion of SLAM-like ideas, where sensor data is used to re-localize the robot as it moves along.

For this assignment, you will be working in randomly assigned groups. Each group is assigned a number 1-8. Each group has a Git repository on the class server. The command for your initial cloning is: *git clone uniqname@linus.eecs.umich.edu:/repositories/a1_groupN*, where N is your assigned group number.

For A1 and A2, you'll be using the MAEbot, a small but high-performance mobile robot, with the following features:

- two-wheel differential drive
- wheel encoders
- an on-board IMU
- a fixed on-board forward-facing color camera
- a single rotating (5 Hz) laser range-sensor
- on-board ARM processor running Linux
- wireless communication

You will find some technical specs in Canvas at *Files/Specs/maebot_datasheet.pdf*.

Task I: Implement Odometry

To control the robot, you will need two pieces of information: the robot's pose and wheel velocities. Write code to calculate the robot's pose and velocity using wheel encoders. The MAEbot datasheet contains the parameters needed to complete this task.

Task II: Implement Your Visualization Tool

As part of your evaluation, you will need to display information about the robot's estimated pose in the environment and other relevant information related to controlling the robot.

Your visualization program will need draw at minimum the following information, which you can receive by subscribing to LCM messages published by programs you'll write for this assignment or provided by the staff:

- The robot's dead reckoning trajectory as a purple line and current pose as a purple triangle.
- The robot's SLAM-estimated trajectory as a blue line and current pose as a blue triangle.
- Vertices visited by the robot as small red squares.

Task III: Controllers for the MAEbot

For this task, you'll be implementing three different controllers. For each of these controllers, you'll be attempting to drive in a square in the environment. You can implement each controller in a single program with a command-line argument to choose which controller to use or as three separate programs. The choice is yours.

In either case, your program(s) should take a command-line argument specifying how many vertices of the square the robot should visit before stopping. For example, passing a 1 to the program means stopping at the first vertex the robot encounters and passing a 5 means driving all the way around the square once and stopping at the next vertex you encounter.

The MAEbot is a relatively light robot which makes the wheels slip or deflect easily. Additionally, the MAEbot has high speed motors that hinder its controllability. Thus, you might have difficulty getting the MAEbot to drive perfectly straight.

For evaluating the quality of your robot's odometry, we are providing a binary that contains an implementation of the occupancy grid SLAM algorithm that you'll be implementing in A2. To use the provided binary for creating ground-truth, you will need to create an LCM log of your robot's sensor data while it is driving. Playback the log while running the provided `maebot_slam` program. Ground-truth `maebot_pose_t` messages will be published at 5Hz on the MAEBOT_SLAM_POSE channel.

We will provide a 2m-by-2m square environment to be used for running your MAEbots.

A. Implement Open-Loop Control

Implement an open-loop controller to drive counter-clockwise around a square with $1m$ sides. Use odometry to determine when you have driven $1m$ and when you have turned $\pi/2$ radians at each vertex of the square.

When driving, used fixed motor speeds (v_L, v_R). To drive approximately straight, you'll like have $v_L \neq v_R$ due to differences in the motors, weight distribution of the MAEbot, etc.

To determine when you have completed one side of the square, store the robot's position at its current vertex (x_0, y_0) and drive until $distance((x_0, y_0), (x_t, y_t)) \geq 1.0$. Similarly,

to determine when you have rotated enough at a vertex, store the robot's initial orientation (θ_0) and turn until $\theta_t - \theta_0 \geq \pi/2$.

As noted in lecture, you can expect the results of your open-loop controller to be pretty bad, so don't worry.

Submit: A description of your implemented open-loop controller, including the wheel velocities used. Provide screen shot of your Vx visualization after driving around three times around the square.

Try adjust your wheel velocities so the robot can drive straight enough to get around the square three times, but don't spend too much time if you can't get your robot to make three passes before it hits a wall. Save your time for tuning the subsequent controllers.

B. Implement Closed-Loop Control with Odometry

Using the robot's odometry and wheel velocities, implement closed-loop control to drive counterclockwise around a square with $1m$ sides, like with your open-loop controller above.

The initial pose – (x, y, θ) – of the robot is $(0, 0, 0)$. Thus, the sequence of vertices to visit as the robot drives around the square is $(0, 0)(1, 0)(1, 1)(0, 1)(0, 0)$....

You should start by implementing individual velocity controllers for each wheel so the MAEbot can drive approximately straight on its own.

You will likely find it useful to create another controller that uses an error term calculating using the robot's position relative to its current goal vertex. For example, you can slow down as you approach a vertex so you don't overshoot.

Submit: A description of your implemented closed-loop odometry controller. Make sure to include information on how you selected the gains. Provide a screen shot of your Vx visualization after driving three times around the square.

C. Implement Closed-Loop Control with Laser Scans

Use laser scan data to control the robot by maintaining a fixed distance from the walls in a simple square environment.

Write a controller that attempts to keep the robot $0.5m$ from the walls of the environment. At the corners, where two walls are approximately equidistant, turn $\pi/2$ and begin following the next wall. Repeat these steps until the robot has driven to the number of vertices specified from the command line.

Submit: A description of your implemented closed-loop laser-based controller. Make sure to include information on how you found the local minima in the laser data and what you used for your error signal in your laser-based controller. Provide screen shot of your Vx visualization after driving three times around the square.

Challenge: Robot Control

We will be holding a MAEbot race during lab on January 22.

Each group will have three runs to race their MAEbot three laps around the square environment. The fastest successful run will be used for each group. A successful run requires the robot to stop within a certain distance of the starting position at the end of the final lap, which will be marked with a piece of masking tape.

You can use any controller you want during the challenge, and you can change parameters between runs. A good strategy would have you start conservatively to ensure one successful run before throwing caution to the wind and zooming around the course!

The winning group will receive an edible prize.

Your grade for the challenge is based only on successfully completing three laps around the environment, not on whether or not you win.

Report: Robot Control

Each group should produce a report on your project, formatted using LaTeXin IEEE two-column conference format. This style can be found in the ieeeconf package.

The .tex file associated with this assignment is included in the repository in the *docs* folder to use as an example.

LaTeXis available on the computers in EECS 2334, on CAEN machines, and can be installed for Windows, Linux, or Mac.

Use Vx to produce the needed figures, and then include them into the report, which is submitted as a PDF file. Each figure should include a detailed caption, explaining the method, conventions, and significance of the information in the figure. The main body of text should refer to each figure, but should not duplicate the caption explanation. (Think of the figure and its caption as a module being used by the main body to communicate the message of the paper.)

Certification [required for credit, 0 points]

Print or write the following on a sheet of paper:

"I participated and contributed to team discussions on each problem, and I attest to the integrity of each solution. Our team met as a group on [DATE(S)]."

Each team member should sign the paper (physically, not digitally), and a photo of the paper should be included with your problem set. By doing so, you are asserting that you were materially involved in each answer, and that you certify that every solution complies with the collaboration policy of this course. In the event of a violation of the course's guidelines, your certification will be forwarded to the honor council.

If the statement requires qualification (i.e., it's not quite true), please describe the issue and the steps you took to mitigate it. If a team member did not materially contribute to a particular solution, then their score on that problem will generally be reduced. The staff will use their discretion based on the specific circumstances you identify, and we are happy to discuss potential problems prior to the problem set due date.

If your team works together, as required, this will be painless! :)