A1: Robot Control

Benjamin Kuipers and Tiancheng Ge

Assigned: September 9, 2019
Challenge: September 23, 2019, 9 am, in class
Report Due: September 23, 2019, 11:59 pm

Purpose

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

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. An excel sheet with the groups assigned is uploaded on Canvas under `files/A1/teams-a1`. The template code on Canvas, at `files/A1/a1_code.tar.gz`.

For A1, A2 and A3, you'll be using the Mbot, a small but high-performance mobile robot, with the following features:

- two-wheel differential drive
- wheel encoders
- an on-board IMU
- a single rotating (5 Hz) laser range-sensor
- A BeagleBone
- A Raspberry Pi 3
- wireless communication
- 5 degrees of freedom manipulator arm
- An onboard camera

You will find a document to set up the BeagleBone and the Raspberry Pi on Canvas at `files/Specs/Mbot document.pdf`. For more hints and implementation details, check `files/A1/Assignment1 Implementation Guide.pdf`.

You will implement four controllers in this lab. One closed-loop wheel velocity controller (Task III), and three types of position controller to drive the robot in a square (Task IV, V, VI). The three position controllers are

- open-loop control
- closed-loop control with odometry
- closed-loop control with laser scans

The three position controllers will use the closed-loop velocity controller implemented in Task III.

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 `A1_doc` contains the steps 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 (x, y, θ) and other relevant information (such as left and right wheel velocity v_R, v_L) related to controlling the robot. Wheel velocity is the angular velocity of the motor times πD, where D is the diameter of the wheel.

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 (odometry) as a blue line and current pose as a blue triangle.
- The robot's estimated trajectory by laser scans (in Task VI) as a purple line and current pose as a purple triangle.
- Mark the vertices (the points where the robot turns) visited by the robot as red squares.

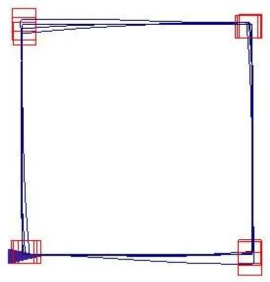

Fig. 1. An example plot showing the odometry trajectory.

Task III: Implement Closed-Loop Velocity Control

In this task, you will implement two PID controllers for v_R and v_L, respectively. The goal is to make each wheel travel at a specified set-point velocity.

Each motor takes a PWM (pulse-width modulation) duty cycle as the argument to specify the voltage. At each timestamp, the PID velocity controller calculates the duty cycle to command the motor, based on the error between the instantaneous velocity and the set-point velocity.

In the template code, the PID velocity controllers for both wheels have been implemented for you. However, you will need to tune the PID gains to make sure that each wheel can reach the set-point velocity, without much overshooting. Given the integral gain in the PID controller, the velocity controller should be able to maintain velocity at the set point, even if there is friction force, or the robot is going up and down some slopes.

Submit: For each wheel, submit 3 step-response plots and their corresponding PID gains. In each step-response plot, there are two steps: first, the step from zero velocity to the desired velocity set point, and the second step from the desired velocity back to zero.

1) A overdamped plot, where the wheel velocity reaches the set point too slowly.
2) A underdamped plot, where the velocity overshoots significantly.
3) A plot for a well-tuned controller (could be critically damped or slightly underdamped).

In total, you will need to include 6 plots in the report for this task.

Note: see whether you can produce under-damped behavior, where velocity overshoots its set point and corrects. (The robot itself may have enough damping to eliminate overshooting.)

Hint: You will need to reset the integral term in the PID controller when the set point changes, to prevent integral windup.

Task IV: Implement Open-Loop Control

In Tasks IV, V, and VI, you'll be implementing three different type of controllers. For each of these controllers, you'll be attempting to drive in a square in the environment.

In this task, you will implement an open-loop controller to drive counter-clockwise around a square with 1 meter sides.

To do so, use the velocity controller implemented in Task III, assume both motors can reach the velocity set-point immediately (which is inaccurate), and calculate the time needed to complete the translation or rotation. Apply the velocity set-point for the time needed, and then set the velocity back to zero. This controller is open-loop, in the sense that the loop is not closed for position.

When driving straight, use fixed wheel velocities $v_L = v_R$. When turning, use fixed wheel velocities $v_L = -v_R$

Submit:

1) A description of your implemented open-loop controller, including the wheel velocities used.
2) A screen shot of the Vx visualization of odometry after driving three times around the square.

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

Task V: Implement Closed-Loop Control with Odometry

Using the robot's odometry and wheel velocities feedback, implement closed-loop control to drive counter-clockwise around a square with 1 m 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 $(0, 0)(1, 0)(1, 1)(0, 1)(0, 0)$....

In Task IV, we did one level of abstraction: assume that the wheel velocities can follow the velocity commands exactly, and abstract away the lower level PWM control commands. In Task V and VI, we will do one more level of abstraction. Let v and ω be the desired forward and angular velocity set points of the robot. Then

$$v = (v_R + v_L)/2$$
$$\omega = (v_R - v_L)/B$$

where B is the wheel base.

We will use v and ω as the motor command to control the robot, and abstract away the left and right wheel velocities. To do so, we can express the set-point wheel velocities in terms of v and ω, and reuse the velocity controller in Task III.

$$v_R = v + \omega B/2$$
$$v_L = v - \omega B/2$$

In this task, design and implement two controllers: one for driving straight, and the other one for turning, using the odometry and instantaneous velocities feedback.

For driving straight, the controller is similar to the wall follower discussed in the lecture. It should try to minimize instantaneous angular velocity feedback as the robot travels, while minimizing the estimated distance error (from the odometry) with respect to the goal point.

When turning, the controller should make sure the robot can reach its desired heading angle based on odometry estimation, and it should minimize the instantaneous forward velocity feedback of the robot as making the turn.

Display the odometry of your robot on the GUI. Even though the robot might not drive in a square in the real world, the odometry trajectory may still be a "nearly perfect" square, since it is an illusion to the robot itself.

Submit:

1) A description of your implemented closed-loop odometry controller.
   Provide formulas to calculate the desired set-point forward velocity v and angular velocity ω at a timestamp, using instantaneous velocities feedback and the error terms estimated by odometry at that timestamp.
2) A screen shot of the Vx visualization of odometry after driving three times around the square. Compare it with the odometry visualization in Task IV.

Task VI: 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 2m-by-2m square environment.

Utilizing what you have in Task V, write a controller that attempts to keep the robot 0.5 m 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.

As the robot is driving straight, it will get the distance to the wall on the right, by identifying the correct local minimum in the laser scan. The wall-following controller will keep a fixed distance from the wall, making the robot travel parallel to the wall.

To terminate the wall-following travel, the robot needs to detect the wall in front of it. To make the counter-clockwise turn at the corner, the robot needs to track two perpendicular walls.

Submit:

1) 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.
2) A screen shot of the Vx visualization of odometry after driving three times around the square.
3) A screen shot of the Vx visualization of the trajectory in the (x, y) frame of positions *in the world*, estimated by laser scans after driving three times around the square. Compare it with the odometry trajectory.

Bonus Points: Write the code that estimates the world frame coordinates of the robot by laser scans in a way, such that Tasks IV and V can reuse the code. Take screen shots of the Vx visualizations of the world frame trajectory of Task IV and V. Compare them with the world frame trajectory in Task VI. For each screen shot, 5 bonus points out of the total 100 points of this report will be rewarded.

Challenge: Robot Control

We will be holding a Mbot race in class on Monday September 23.

Each group will have three runs to race their Mbot 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!

Your grade for the assignment is based only on successfully completing three laps around the environment, not on whether or not you win. Winning the Challenge just gives you bragging rights.

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 uploaded to Canvas in the `files/A1/report template` folder to use as an example. You can use Overleaf or Sharelatex to create files simultaneously as a group.

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.)

Compress all the code into a tar.gz file, `team0X_a1.tar.gz`, submit it together with the report. For each team, one submission is sufficient.

Certification [required for credit, 0 points]

Print or write the following on a sheet of paper:

”I participated and contributed to team discussions, and I attest to the integrity of the report.”

Each team member should sign the paper (physically, not digitally), and a photo of the paper should be included with your report. By doing so, you are asserting that you were materially involved in the team work, 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. The staff will use their discretion based on the specific circumstances you identify, and we are happy to discuss potential problems prior to the report due date.

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