Homework 2

ME 500 A4 - Prof. Tron

Thursday 22nd March, 2018

The goal of this homework is to apply analytically and implement in ROS the concept we saw in class about rigid body transformations and modeling the motion of robots. In particular, you will develop nodes that provide *odometry* (i.e., an estimate of the pose of the robot with respect to its starting position) based on either the inputs given to the robot. **You will find that the results are often in the right ballpark, but are not extremely accurate, with noise that accumulates the farther the robot travels and wheel slip introducing large errors.** This is expected. In practice, external measurements (such as those from a camera) need to be combined. In addition, you will have a chance to use ROS launch files.

General instructions

The assignment is intended to be completed in pairs using *pair programming* (remember, two people working on *one* laptop). See the syllabus for details, and see Blackboard for the group assignments.

Homework report (required). Solve each problem, and then prepare a small PDF report containing comments on what you did for each problem, including whether you tackled the optional questions.

Demo video (required). You need to prepare a one to three minutes video with the following elements:

- A foreground framing of yourself (both members of the group) saying your names and the name(s) of the robot(s).
- For each question marked as video, a short segment showing the laptop and robot demonstrating that you completed the requested work. If you could not complete a question, you must explain what were the problems that you encountered.
- A final conclusion where you explain what you learned with the activity.

Submitting multiple video segments instead of a single video is allowed but not encouraged (in this case, all the segments must be recorded in the same session, e.g., not on different days or hours apart). **Please have one person run the demonstration, and the other narrating what is happening in the video.** Alternatively, you can insert explanatory text in post-production.

Analytical derivations. To include the analytical derivations in your report you can type them in any equation editor (the LaTeX typesetting system is recommended) or clearly and neatly write them on paper and use a scanner (least preferred method).

Submission. Compress all code you wrote for the assignment into a single ZIP archive with name `<Last1><First1>-<Last2><First1>-hw<Number>.zip`, where `<First1>`, `<First2>` and `<Last1>` , `<Last2>` are the first and last names of the group, and `<Number>` is the homework number (for instance, `TronRoberto-BeeVang-hw1.zip`). Submit the report, the video and the ZIP archive through Blackboard. Both members of the group need to submit a copy of this material. Please refer to the Syllabus on Blackboard for late homework policies.

Grading. Each question is worth 1 point unless otherwise noted. Questions marked as **optional** are provided just to further your understanding of the subject, and not for credit. If you submit an answer I will correct it, but it will not count toward your grade. See the Syllabus on Blackboard for more detailed grading criteria.

Maximum possible score. The total points available for this assignment are 12.5 (12.0 from questions, plus 0.5 that you can only obtain with beauty points). Points for the video questions are assigned separately on Blackboard.

Hints Some hints are available for some questions, and can be found at the end of the assignment (you are encouraged to try to solve the questions without looking at the hints first). If you use these hints, please state so in your report (your grading will not change based on this information, but it is a useful feedback for me).

Use of external libraries and toolboxes All the problems can be solved using the basic ROS Python facilities. You are **not allowed** to use functions or scripts from external libraries or packages unless explicitly allowed.

Update the software for your robot

Before starting the assignment, follow the directions on the Blackboard Wiki page *"ROS-Bot/Updating the provided software"*.

Problem 1: Closed form trajectory solutions for differential drive robots

In this problem you will apply the theory we saw in class regarding the ODE model for our ROSBot.

Dynamical model of ROSBot As mentioned in class, the state of our robot can be represented with a 2-D pose (R, T) (expressed as an Euclidean transformation from the body to the world reference frames), which can be parametrized with a minimum number of three variables: x_{robot} and y_{robot} for the translation, and θ_{robot} for the 2-D rotation angle. These variables can be combined in the *state*

$$z = \begin{bmatrix} x_{\text{robot}} \\ y_{\text{robot}} \\ \theta_{\text{robot}} \end{bmatrix}. \tag{1}$$

From class, we saw that the motion of the robot can be modeled with the following dynamical system:

$$\dot{z} = A(z)\begin{bmatrix} s_{\text{left}} \\ s_{\text{right}} \end{bmatrix}, \tag{2}$$

$$A(z) = \begin{bmatrix} \frac{k_{\text{lin}}}{2}\cos\theta & \frac{k_{\text{lin}}}{2}\cos\theta \\ \frac{k_{\text{lin}}}{2}\sin\theta & \frac{k_{\text{lin}}}{2}\sin\theta \\ -\frac{k_{\text{ang}}}{2} & \frac{k_{\text{ang}}}{2} \end{bmatrix}, \tag{3}$$

where $s_{\text{left}}, s_{\text{right}}$ are, respectively, the left and right commanded motor speeds (between -1 and 1, in our implementation), $A(z)$ is a matrix in $\mathbb{R}^{3\times 2}$, and $k_{\text{lin}}, k_{\text{ang}}$ are two lumped coefficients that combine physical characteristics of the motors and the robot.

Closed form solution for constant inputs Although for general dynamical system a closed-form trajectory for a given set of inputs is not easy to find, in our case we can observe that, when the control inputs $s_{\text{left}}, s_{\text{right}}$ are constant, the robot describes circular arcs. We therefore use the following ansatz for the parametric curve $z(t)$ that solves (2):

$$z(t) = \begin{bmatrix} R\cos(\Omega t - \frac{\pi}{2} + c_\theta) + c_x \\ R\sin(\Omega t - \frac{\pi}{2} + c_\theta) + c_y \\ \Omega t - \frac{\pi}{2} + c_\theta \end{bmatrix}, \tag{4}$$

where $R, \Omega, c_x, c_y, c_\theta$ are parameters of the solution.

Question 1.1. Describe, in words, the geometrical meaning of the parameters $R, \Omega, c_x, c_y, c_\theta$.

Question 1.2. Assume that the robot starts from an initial state

$$z_0 = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix}, \tag{5}$$

and is applied constant inputs $s_{\text{left}}, s_{\text{right}}$. Show that (4) is indeed a solution to (2), and obtain formulae determining the values of the parameters $R, \Omega, c_x, c_y, c_\theta$ as a function of $s_{\text{left}}, s_{\text{right}}, z_0, k_{\text{lin}}, k_{\text{ang}}$.

Question 1.3. What happens to Ω and R when $s_{\text{left}} = s_{\text{right}}$? Geometrically, why is this expected? What solution should we use instead of (4) for this case?

Problem 2: Modification of the `motor_command.py` node

The software update to the robot provides a new custom message type `MotorSpeedsStamped` in the package `me416_lab`. This message type can be imported in your Python scripts using:

```
from me416_lab.msg import MotorSpeedsStamped
```

The `MotorSpeesStamped` type is defined in the file `msg/MotorSpeedsStamped`, and contains a header for storing the time information, the left motor speed, and the right motor speed:

```
Header header
float64 left
float64 right
```

Question 2.1. Modify the node `motor_command.py` developed in Homework 1 so that the speeds that are set on the motor, whenever they are changed, are also published on the topic `motor_speeds` (type `MotorSpeedsStamped`). Use the result of the function `rospy.Time.now()` to fill in the value of the `header.stamp` field when you create the message.

Problem 3: Odometry from closed form solution

This problem uses the previous one to produce estimates of the pose of the robot. The parameters of the model $(k_{\text{ang}}, k_{\text{lin}})$ are determined in Question 3.2.

Question 3.1. Implement a node that uses (4) (and your answer to Question 1.3) to publish an estimate of the pose.

> File name: `odometry_wheel_arcs.py`
> Subscribes to topics: `motor_speeds` (type `me416_lab/MotorSpeedsStamped`)
> Publishes to topics: `pose_arcs` (type `geometry_msgs/Pose2D`)
> Description: This node should keep an estimate of the state z (translation and rotation) for the robot. This estimate is initialized to $z = 0$ (all zeros). Every time a new message is received on the `motor_speeds` topic, the node updates the estimate by using the formulae from Problem 1 where the motor speeds are the ones received in the message, the initial conditions are the current estimate of the pose, and the time is obtained from the stamps in the messages. The result of every update should be published on the `pose_arcs` topic.

While you develop this question, keep $k_{\text{ang}} = k_{\text{lin}} = 1$. These constants will be tuned in the next question.

Question 3.2. Tune the constants k_{ang} and k_{lin} in the model used by:

1) performing some maneuver;

2) comparing the actual pose of the robot with the estimates produced by the node;

3) adjusting the constants until you get a reasonably good estimate (e.g., if the estimate indicates that the distance (resp., angle) of the robot is smaller than the actual one, increase k_{ang} (resp, k_{lin}).

Suggested maneuvers are: go straight forward for a fixed distance, and perform a full circle with a small radius. If the wheels slip during the maneuver, performe it again (wheel slip will be the main source of errors in the estimate). It is suggested that you record the commands for a maneuver with `rosbag record`, and then repeat the same commands with `rosbag play` while you adjust the constants. You might also find the tuning easier if you use tape marking on the floor, or a floor with tiles to give you a reference.

Question 3.3 (optional). Modify your node so that the estimate is updated and published at a regular frequency (much faster than the frequency of messages on `motor_speeds`).

Question 3.4 (video). Drive the robot in an approximately square trajectory. Show that the odometry results, after the tuning, are reasonably accurate. Please explicitly show the position of the robot and the estimate after every turn of the square.

Problem 4: Odometry from Euler integration

For this question, we will "pretend" that we do not know a closed form expression for the integration of (2), and instead we will use the general Euler numerical integration method.

Question 4.1. q:odometry-euler Implement a node that applies Euler method for the integration of (2) to publish an estimate of the pose.

> File name: `odometry_wheel_euler.py`
> Subscribes to topics: `motor_speeds` (type `me416_lab/MotorSpeedsStamped`)
> Publishes to topics: `pose_euler` (type `geometry_msgs/Pose2D`)
> Description: This node should keep an estimate of the state z (translation and rotation) for the robot. This estimate is initialized to $z = 0$ (all zeros). The node should have a subscriber that saves the last message received on `motor_speeds` in an internal variable called `motor_speeds_last_received`. At regular intervals, say, 50Hz, the node should update the estimate z by applying the Euler method on (2) using the values stored in `motor_speeds_last_received` for the inputs; after every update, the estimate should be published to `pose_euler`.

Question 4.2 (video). Drive the robot in an approximately square trajectory. Show that the odometry results, at each turn of the square, are reasonably accurate.

Problem 5: Odometry logging

Question 5.1. Implement a node that applies Euler method for the integration of (2) to publish an estimate of the pose.

> File name: `odometry_wheel_logging.py`
> Subscribes to topics: `pose_euler` (type `geometry_msgs/Pose2D`) `pose_arcs` (type `geometry_msgs/Pose2D`)
> Publishes to topics: None, but writes to files `pose_euler_log.csv` and `pose_arcs_log.csv`.
> Description: This node should open two files `pose_euler_log.csv` and `pose_arcs_log.csv` for writing. Every time a new message is received on either the topics `pose_euler` or `pose_arcs`, the node should write on the corresponding file a new line with four values, separated by commas, with the following order: current time (`rospy.Time.now()`), x_{robot} coordinate, y_{robot} coordinate, and θ_{robot} angle.

You can use the provided script `scripts/write_csv.py` as a template for writing values to a file.

Question 5.2. Make a launch file `me416_lab/launch/odometry-all.launch` that launches together the nodes `odometry_wheel_euler.py`, `odometry_wheel_arcs.py` and `odometry_wheel_logging.py`.

Problem 6: Analysis and comparison with motion capture

In this question you will compare the results of the two wheel odometry techniques implemented above against themselves and the ones obtained with an external positioning system (motion capture). It is suggested that you use Matlab to perform the analysis and show the results, but other software is acceptable too.

Question 6.1 (2 points). Run the `odometry-all.launch` file and the `manual_drive.launch` lauch files (the latter file is provided in the software update). Drive the robot in an approximately square trajectory. Stop all the nodes. Then copy the files `pose_euler_log.csv` and `pose_arcs_log.csv` to your laptop. Load the data in Matlab (using the command `csvread()`) or other software from each file, and make two plots: the robot angle as a function of time, and the x-y trajectory of the robo; each plot should contain two lines, one for each file; please include a legend, and include the plots in the report.

Question 6.2 (video). Comment on the results in the plots.

Question 6.3. Go to the shared folder `https://drive.google.com/drive/folders/11pY_awP6cBUlExoH_6Kj7yt2HaruWVlV?usp=sharing`. Download the motion capture data for your group (your or your teammate's name with the suffix `_converted.csv`). Repeat Question 6.1, except that:

- Instead of manually controlling the robot, use the `rosbag play` command to replay the trajectory from the last mini-hackaton (note that the command replays all the topics in real time, including any initial, middle or final pause);
- Include the results from the motion capture in the plots. Note: you might have to subtract the initial coordinates and angles from the motion capture trajectory to make it start from the same initial condition of the odometry estimates.

Question 6.4 (video). Comment on the results in the new plots. Note: the motion capture file might have missing values/inconsistent angular data due to the fact that the system could not track one or more markers at every time instant.

Hint for question 1.2: While this question might appear intimidating due to the number of parameters, but the underlying principle is as follows. To give a simpler example, assume that the differential equation was

$$\dot{x} = \cos(5t), \tag{6}$$

and the candidate solution is

$$x(t) = \sin(\Omega t) + c. \tag{7}$$

The derivative of the candidate solution is

$$\dot{x}(t) = \cos(\Omega t), \tag{8}$$

which shows that (7) is a solution to (6) with the choice of parameters $\Omega = 5$.

This, however, does not determine a full solution, because we still do not know the value of c. For this, we need to use an initial condition. Assuming, for instance, that we are given the initial condition $x(0) = 3$, by substituting in 7 for $t = 0$ we will obtain $c = 3$.

The solution to the original Question 1.2 should follow the same steps, with the difference that there are more parameters, and that instead of numbers such as 5 and 3 you have other parameters.

Hint for question 3.1: In addition to the current estimate of the pose, the node should keep in memory the value of the motor speeds and of `header.stamp` of the previous message, in order to be able to compute the duration of the time interval and the motion between two updates.

Hint for question 3.2: The `rosbag` command has the ability to record and replay individual topics. A clever approach is to perform a maneuver once, and then record and replay only the messages on the topic `motor_speed`. In this way, the robot does not even have to move while you adjust the constants.

Meta-comment

My original plan was to include an odometry estimate from an IMU. Unfortunately, Pololu has updated the IMU board during the summer, and the software that was used during the initial development of the ROSBot platform is not compatible anymore with the IMU you have. I realized this too late to code a proper fix, and, as a result, in this homework the Euler method was applied again on the wheel command instead of the IMU. However, here is, in a nutshell, what you would have learned by using the IMU:

- You need to record the bias every time you start the node, and subtract it from every measurement.
- Increased accuracy can be obtained by using the gravity vector to determine what is the "up" direction, instead of assuming to have the IMU board mounted on a level plane.
- After including the estimates for the bias and the gravity vector, the odometry estimate from the IMU for the rotation is reasonably accurate. The estimates for the linear velocity and position are not (the noise makes them diverge too quickly).