MEM380: Applied Autonomous Robots I

Wk 5 Assignment

Due: Monday, Oct. 29, 11:59pm EST

Objective: The objective of this assignment is three-fold: 1) investigate the use of inertial sensors (accelerometers and a rate gyroscope) for dead reckoning in a differential drive mobile robot; 2) implement a least-squares line fit solution using actual laser data; and 3) implement a least-squares circle fit algorithm using actual laser data.

Dead Reckoning via an Inertial Navigation System (INS)

Assumptions: In all of the INS experiments, we will assume that

- Our inertial measurement unit (IMU) consists of x and y-axis accelerometers, and a z-axis rate gyroscope.
- The IMU frame is coincident with the robot frame.
- Our kinematic model for a differential drive robot still applies.

1. Inertial Navigation (10 pts) Complete the function `deadReckon1_user(inputFile)` where `inputfile` is the name of a file consisting of robot accelerations in the X_R and Y_R directions (in m/s^2), *i.e.* the robot centered coordinate frame, as well as angular velocity $\dot{\theta}$. As usual, replace `user` with your Drexel login.

- Note that `inputfile` should be entered as `'fileName'` where `fileName` denotes the name of the file including the extension.
- The function provided to you does the following:
  (a) Take a mouse click input from the user and create a robot at that position with $\theta = 0$, Lines 8-10 of `deadReckon1_user()`.
  (b) Read in the inertial sensor measurements (sampled at 50Hz) from a text file, Lines 12-18 `task1_user()`.
  (c) Use the accelerations to update the global pose of the robot using the `makeRobot` and `moveRobot` functions. Take a look at `sample.m` file for an example using `makeRobot` and `moveRobot` functions. **Your portion of the code should be added after Line 36 and before Line 43 of the downloaded file**.
  (d) We will assume that the robot is starting from a stopped position.

  Some helpful MATLAB functions include `ginput` and `help`. If you don't know what `ginput` does, type `help ginput` at the MATLAB command prompt.
- You can use the sample input file `input_wk4.txt` to test the functionality of your program. If it is working correctly, the robot should move at constant speed in a circle going counterclockwise (see Figure 1). In the coordinate frame attached to the robot, *i.e.* the robot coordinate frame, the x-axis is always pointing forward.

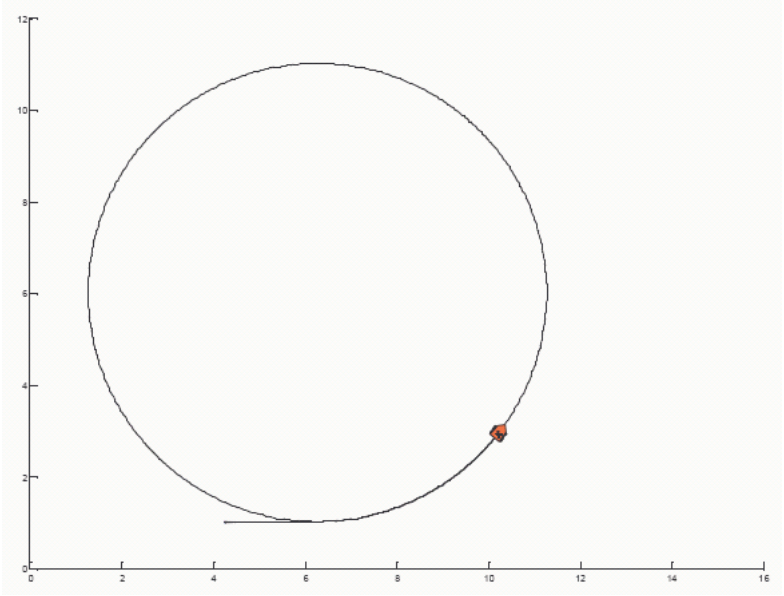

Figure 1: Schematic for closed-loop motion control.

- You should also test your program with other input files of your own creation to ensure proper functionality of your program.

Tips: My suggested route for doing this task is as follows:

(a) Express the velocity of the robot in world coordinates;

(b) Take the derivative in the world coordinates to find the vehicle accelerations in the world frame;

(c) Rotate these accelerations into the body frame using $\mathbf{R}^T = \mathbf{R}(-theta)$;

(d) Use the equations that were found in the previous step to match the sensor readings; and

(e) Numerically integrate on those value to find the trajectory of the vehicle.

Keep in mind that in the coordinate frame attached to the robot, the x-axis is always pointing forward, and the y-axis is always pointing sideways. If one naively integrates the velocity values in the input file

$$v_x(i+1) = v_x(i) + a_x(i) * dt,$$
$$v_y(i+1) = v_y(i) + a_y(i) * dt.$$

The non-negative a_y terms would produce, after a time(i) step, a velocity in the body y-axis direction. This violates what we know about the differential drive robot. It can NOT move along its own y-axis/sideways. That means there must be another source of y-axis acceleration in the system. By following the steps above you should be able to discover what that source is. For intuition - think about how a car can create a sideways acceleration and not move sideways.

2. Dead Reckoning with Bias (10 pts) In this exercise, we will test the effects of bias in inertial measurements on dead reckoning performance.

a Generate an input file which reflects the robot moving from standstill to 2 m/s in 5 seconds under constant x-acceleration, and then travels a total distance of 50 meters. Run this trial using your `deadReckon1_user` function and create a figure. You should name your script `deadReckon2a_user.m` where `user` is your Drexel login.

b Repeat the above trial by introducing bias to the measurements obtained in 2(a) above to reflect:

i Bias to the x accelerometer of $0.005g$ only

ii Bias to the y accelerometer of $0.005g$ only

iii Bias to the z accelerometer of $0.005g$ and bias to the angular velocity of 0.01 rad/s

Write a script, `deadReckon2b_user.m` to generate a figure from each trial. Use `disp()` to output your discussion of the effects of each component, to include error magnitude, rate of error growth, etc.

3. Least-squares line fit (10 pts) Read over the notes provided on the course website to Prof. Simoncelli's notes on least-squares optimization and SVD. Complete the m-file `lsLineFit_user.m`. The code provided to you does the following:

- Reads in the laser data provided in `lineData.txt` and stores it into a matrix, Lines 5-8. The data is provided in polar coordinates where the first column is the angle in rad and the second column is the measured distance in mm.
- Converts the data provided from polar coordinates into Cartesian coordinates, Lines 10-12.
- Plots the data points, Lines 14-17.
- Divides the data points into K segments or clusters based on the distance between each consecutive pair of points, Lines 19-39. If you look at the plot generated in Lines 14-17, you see that this clustering should segment out portions of the data that seem to be part of the same line.
- Fits lines to each cluster, Lines 46-50. **Note, your portion of the code will be added here.** Keep in mind that you may need to add more lines of code than provided to you.
- Draws the fitted lines to each cluster.

As usual, replace `user` with your Drexel login and submit your code. Look at the final plot generated. Do you notice anything interesting?

Extra Credit (5 pts) Can you come up with a strategy to improve the current line fitting procedure? Submit your extra credit code as a separate file named `lsLineFit_ec_user.m`. You can use your completed `lsLineFit_user.m` as the starting point.

4. Least-squares circle fit (10 pts) Read over the notes provided on the course website to the Developmental Testbed Center's (DTC) notes on least-squares fit to a circle. Complete the m-file `lsCircleFit_user.m`. The function provided is very similar to the one presented in Problem 3. Once again, the code provided does the following:

- Reads in the laser data provided in `circData.txt` and stores it into a matrix, Lines 5-8. The data is provided in polar coordinates where the first column is the angle in rad and the second column is the measured distance in mm.
- Converts the data provided from polar coordinates into Cartesian coordinates, Lines 10-12.
- Plots the data points, Lines 14-17.

- Divides the data points into K segments or clusters based on the distance between each consecutive pair of points, Lines 19-39. If you look at the plot generated in Lines 14-17, you see that this clustering should segment out portions of the data that seem to be part of the same line.
- Fits lines to each cluster, Lines 46-50. **Note, your portion of the code will be added here.** Keep in mind that you may need to add more lines of code than provided to you.
- Draws the fitted circles to each cluster.

As usual, replace `user` with your Drexel login and submit your code. Look at the final plot generated. Do you notice anything interesting?

Extra Credit (5 pts) Can you come up with a strategy to improve the fitting procedure? Submit your extra credit code as a separate file named `lsCircleFit_ec_user.m`. You can use your completed `lsLine_user.m` and `lsCircleFit_user.m` as the starting point.