PS4 Report

Code Repository:

Our group's repository is located at the following URL:
https://github.com/ShaunHoward/cwru-ros-pkg-hydro-gamma

The packages that were used for this assignment were the robot_commander and sensor_utils packages located in the catkin/src/cwru_376_student/ directory. Each of the packages contains a README and well-commented source code.

Changes between cwruBot and Jinx:

Only one change was really necessary to the run robot code on cwruBot. Many changes were required in order to run the code on Jinx because Jinx is using real hardware and cannot safely travel as fast as the robot in the simulator.

In order to run the code on cwruBot, we had to change the subscription in our lidar sensor node, “lidar_alarm” to “/laser/scan”.

In order to run the code on Jinx, we had to change both the lidar sensor node subscription as well as many properties and functions of the velocity profiling node, “vel_profiler”.

The following change was made to the lidar sensor node “lidar_alarm”:

1. In lidar_alarm, change subscription to “/laser/scan” in vs. “/base_laser1_scan” (for Jinx)

The following changes were made to the velocity profiling node “vel_profiler” in order to use the code between cwruBot and Jinx:

1. The maxAcceleration constant was decreased
2. The maxVelocity constant was decreased
3. We modified the moveOnSegment() function because some calculated values were no longer used and thus became unnecessary in the code
4. We changed the segment lengths and rotation phis to execute when moving robot in the hall according to the distances and rotations we needed (for different environments)
5. The following ternary operators were changed in the trapezoidal velocity profiling methods for forward motion and rotation because they were causing the speed to ramp up instead of ramp down when the maxVelocity was reached:
    1. Ternary operator from > to < in callback.odomVelocity > scheduledVelocity else if clause in trapezoidal slow down function
    2. Ternary operator > to < in fabs(callback.odomOmega) > scheduledOmega else if clause in turn slow down function
6. We added many turning cases (8) that allow phi to transition naturally between positive and negative radians in order to continue on a rotation segment and consistently increment its delta phi. If negative phi values, we used the absolute value of differences in order to maintain a positive delta phi increment. The delta phi increment was added to the phiCompleted in order to

track how many radians the robot had rotated at a given iteration.
    1. We added a last odom callback phi variable in our header file in order to track the last phi and get the difference between the last and current callback phis to determine the direction of turning and the delta phi.
7. We re-factored some code so that move segments and rotation segments are initialized when the move or rotation functions are called.
8. We added many comments and organized the code so that it is more understandable and reusable for others.

Discussion of Results:

Through experimentation with Jinx on the rollers and in the hallway, we have gathered recommended speeds and accelerations. The recommended values are as follows:

- const float maxVelocity = 0.5; //1m/sec is a slow walk
- const float minVelocity = 0.1; // in m/sec; if command velocity too low, robot won't move
- const float maxAcceleration = 0.5; //1m/sec^2 is 0.1 g's
- const float maxOmega = 1.0; //1 rad/sec-> about 6 seconds to rotate 1 full revolution
- const float minOmenga = 0.1; //in rad/sec
- const float maxAlpha = 1; //1.0 rad/sec^2-> takes 1 sec to get from rest to full omega
- const float changeInTime = 0.05; // choose an update rate of 20Hz; go faster with actual hardware
- const float maxSafeRange = 2.5; //start slowing down when object is within this range of robot (units in meters)
- const float minSafeRange = 0.5; //units in meters

These values are appropriate due to friction and other factors that cause the robot not to accelerate from exactly 0 m/sec^2.

We also found certain values necessary for ramping forward and rotation velocities up and down. These values are as follows:

- float accelerationTime = maxVelocity / maxAcceleration; //...assumes start from rest
- float decelerationTime = maxVelocity / maxAcceleration; //(for same decel as accel); assumes brake to full halt
- float accelerationDistance = 0.5 * maxAcceleration * (accelerationTime * accelerationTime); //distance required to ramp up to full speed
- float decelerationDistance = 0.5 * maxAcceleration * (decelerationTime * decelerationTime); //same as ramp-up distance
- float turnAccelTime = maxOmega / maxAlpha; //...assumes start from rest
- float turnDecelTime = maxOmega / maxAlpha; //(for same decel as accel); assumes brake to full halt
- //float turnAccelPhi = 0.5 * maxAlpha * (turnAccelTime * turnAccelTime); //same as ramp-up distance

- float rotationalAccelerationPhi = 0.5 * maxAlpha * (turnAccelTime * turnAccelTime);
- float rotationalDecelerationPhi = 0.5 * maxAlpha * (turnDecelTime * turnDecelTime);

As far as repeatability, the journey is definitely repeatable. Each time the software runs, it will have various inaccuracies in turning. For example, one inaccuracy was in the rotation phi. We have an average error of about +.02 radians when turning. When turning negatively, this means we had an error of -.02 radians. Therefore, when we want to turn -1.57 radians, the robot rotation will vary between -1.57 radians and -1.59 radians. Another inaccuracy we had was overshooting the desired segment length by a small fraction, i.e. +0.1, during forward moves. Both of these inaccuracies were inherently caused by the lethargy of the odometer callback refresh rate as well as factors like friction and other program inaccuracies. These other inaccuracies exist in the ramping up and down of velocity for both forward movement and rotation. The velocity will initially increase past the maxVelocity and/or maxOmega during the execution of the forward movement and rotation functions. Then the velocity will normalize after it is run through the trapezoidal velocity profiling algorithms again on the next iteration of the odometer callback. By the end of segments of either forward moves or rotations, the robot will slow down approximately to its goal, but often overshooting by the discussed factor.

Overall, the code produced for this project is reusable for other dead-reckoning movement assignments and will also work with lidar and estop sensors. In order to minimize inaccuracies, we could add more precise calculations or upgrade the robot hardware so the odometer refreshes faster. There could be other possible tweaks to make the program more accurate including different acceleration and deceleration equations. In conclusion, despite the small inherent inaccuracies, the journey is most definitely repeatable and more precise values could be used when commanding the robot to move forward and rotate in order to make the robot follow a more precisely coordinated path with dead-reckoning.