Lab 7: Motion Control on Jinx

Sasha Belotserkovskaya, avb27@case.edu; Kristina Collins, kvc2@case.edu; Shawn Qian, xxq82@case.edu; Connor Wolfe, cbw36@case.edu

I. Introduction

This assignment demonstrates the motion control methods simulated in PS6, PS7 and PS8 on Jinx. Three control paradigms were used: Open loop control, based only on expected position; linear feedback control based on odometry; and linear feedback control based on odometry and AMCL.

II. Testing

A. Jinx Startup Procedure

- Turn on the thermal fuse by pushing the black tab up.
- Wait 10 seconds for the pre-charge of the capacitors on the output stage of the Roboteq.
- Hold down the on switch until the LED ring and battery voltage monitor turn on. (If the voltage is under 24V, charge the robot for a while.)
- Check the ESTOPs.
- Turn on the computer and log in: user/1lmiagmc!
- Run **export ROS_MASTER_URI=http://129.22.148.227:11311** on Jinx.
- Run **export ROS_IP=129.22.148.227** on Jinx.
- Run **roslaunch launchers start_jinx.launch** on Jinx.
- Run **ifconfig** on the Atlas machine, then run **export ROS_IP = <IP Address from ifconfig>**.
- Run **export ROS_MASTER_URI=http://129.22.148.227:11311** on the Atlas machine.
- Using the "manual" switch, move Jinx to the hallway, facing the lab.
- Run code.

It's a good idea to run **rostopic list** and make sure the robot is connected.

B. Demonstration Steps

1) Open Loop Control::

- **rosrun my_mobot my_mobot_lidar_alarm**
- **rosrun my_mobot my_mobot_lidar_detecting**
- **rosrun my_mobot my_pub_des_state_path_client**
- **rosrun my_mobot my_open_loop_controller**
- **rosrun my_mobot my_mobot_pub_des_state**
- **rosrun kvc2_amcl_steering glennan2_client**

2) Odometry Control::

- **rosrun my_mobot my_mobot_lidar_alarm**
- **rosrun my_mobot my_mobot_lidar_detecting**
- **rosrun my_mobot my_pub_des_state_path_client**
- **rosrun my_mobot my_mobot_pub_des_state**
- **rosrun my_lin_steering my_lin_steering_wrt_odom**
- **rosrun kvc2_amcl_steering glennan2_client**

3) Odometry Control with AMCL::

- **rosrun my_mobot my_mobot_lidar_alarm**
- **rosrun my_mobot my_mobot_lidar_detecting**
- **rosrun my_mobot my_pub_des_state_path_client**
- **rosrun my_mobot my_mobot_pub_des_state**
- **rosrun amcl amcl**
- **rosrun lin_steering lin_steering_wrt_amcl drifty_odom:=odom**[1]
- **rosrun kvc2_amcl_steering glennan2_client**

C. Tips and Notes

We had to revise our lidar alarm to have a smaller MIN_SAFE_Distance than for the Turtlebot since there is a desk immediately in front of it. We noticed it pinging at 0.4m, so we changed the min value to 0.2m.
We preferred to view each node in a separate terminal window, so note that in order to run nodes to Jinx in a new window, you must first connect to Jinx using the export ROS_MASTER_URI and ROS_IP commands from above.
Make sure to shut off Jinx after use!

III. Testing & Results

Our code and video may be accessed on Github from the following link:
https://github.com/qianxiaoye/teambacon

The video may be accessed at: https://www.youtube.com/watch?v=0noJ0gxX4uEfeature=youtu.be

We were able to navigate Jinx most of the way down the hallway, but had some difficulty with the turns in each case.

A. Open Loop Control

Jinx turns too far and runs into a wall. Granted, this is pretty common with open loop control. At least we proved the LIDAR alarm was working. And she was running straight, just...at the wall.

B. Linear Feedback Control Based on Odometry

The reason Jinx moves from side to side is because the controller gain (K_PHI in the lin_control header file) is set too high. We changed it from 10 to 6 and had better luck in the third video.

[1]This runs the node while find-and-replacing every use of "drifty_odom" with "odom." This trick is very, very useful.

C. Linear Feedback Control Based on Odometry & AMCL

Jinx doesn't oscillate as much, because the gain's been reduced in the controller node. She turns well before the end of the hallway because the path client was set to Y=-25 instead of the original -32, since that was causing Jinx to run into the end of the hallway. Jinx is able to make it to the turning point and spin around, but not spin the correct 180 degrees. At this point, we had to conclude testing due to time constraints.

IV. Conclusion

We were able to operate Jinx and execute each permutation of the code. With AMCL, we were able to get Jinx to go to the end of the hallway, but not perfectly execute the turn. With more tuning and refactoring, we could probably make the code much more robust, but stopped here because of time constraints.

Another note: In the sections using linear feedback, Jinx's yaw control worsens as she moves down the hallway. One theory proposed for this is that the pub_des_state nodes are publishing too fast, so the gap is widening between the current and desired states, messing up the linear feedback controller. In the simulation (and in real life, although not shown here) this sometimes manifests as Jinx running in circles.