Path Planning and Deployment of Mobile Robot Networks

Shengdong Liu
University of California, San Diego
Summer 2015

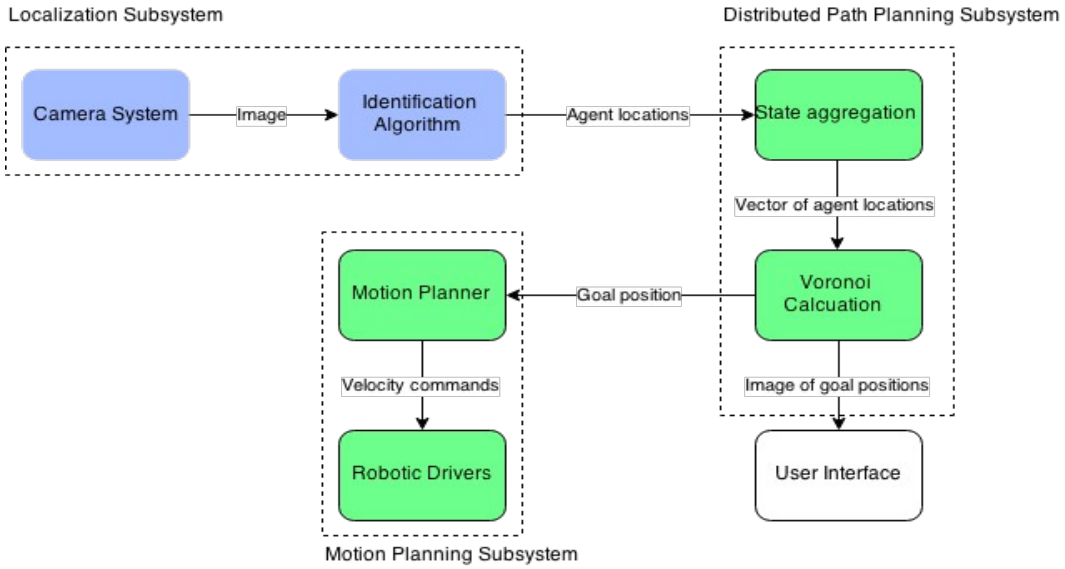
Abstract

This paper continues my work from Spring 2015 and focuses on the implementation of a path planning algorithm, Dubins Curve, on the Turtlebot. This algorithm is adopted to simulate a simple car on TurtleBot, a small ground robot consists of a mobile base, 3D Sensor, and a laptop computer. The simple car is assumed to move at constant forward speed and is constrained by a maximum steering angle, which results in a minimum turning radius. With the implementation of this algorithm, TurtleBots could be used to simulate a network of autonomous vehicles for the the setting up and testing of inter-robot communication and coordination.

1 Big Picture

Advancements in robotics technology are increasing the number of autonomous robots in our daily lives. This boom in robots calls for not only more robot-user interaction, but also more robot to robot communication. By communicating, robots can complete tasks in groups. Tasks such as localization, collision avoidance, and traffic control can be done more accurately and efficiently through inter-robot communication and coordination. The Cortes and Martinez labs at UCSD have been incubating a testbed to test potential multi-agent control algorithms for deployment. The labs currently possess ten TurtleBot platforms as a multi-agent robotic network. The open source ROS (Robotic Operating System) is used as the software platform for its publisher-subscriber model to allow for inter-robot communication and coordination. The objective of the ROS TurtleBot project is to provide a stable system to which distributed control system can be readily deployed for testing and validation on hardware. There are a plethora of interesting algorithms that could potentially be deployed using the TurtleBot network, such as cooperative task completion. Of particular interest to the group is cooperative simultaneous localization and mapping (SLAM).

The current system set up for the testbed consists of three subsystems. The localization subsystem provides the position and orientation of each of deployed TurtleBot through two overhead cameras that spend video stream to a laptop as the base station. The distributed path planning system applies multi-agent control algorithms and uses the video stream to calculates the goal position of each TurtleBot. The motions planning subsystem on each TurtleBot then uses the goal position to create a path and send velocity commands to the mobile base for navigation. Since the whole system is implement in ROS, information are easily passed on by means of publishers and subscribers between the subsystems. Visualizations of the multi-agent control algorithm are created and displayed to the users through the user interface for monitoring the correctness and effectiveness of the multi-agent control algorithm being tested.



Drawing 1

2 Personal Contribution

This quarter, I continued the work Daniel Heideman and I did in the Spring 2015 for motion planning subsystem, specifically on the motion planner. Two main components of the motion planner are path planing and path following. Path planning is the process that calculates an optimal path, given the robot's current location, a goal location, and some constrains. Path following is the process that uses the optimal path as a guide to produce the speed and turning commands sent to the robot's wheels. In Spring, Daniel and I worked on implementing Dubins Curve algorithm as the path planning algorithm to simulate a simple car and implementing unicycle model proportional-integral-derivative controller for path following. This quarter, I put the two components Daniel and I worked on together and conducted some tests both in Turtlesim, and on the actual Turtlebot. In the next few sections, I'll will show some results of tests I conducted in Turtlesim as well as document how I integrated our code for motion planner to run on the Turtlebot.

3 Preliminaries

Please refer to report-Sp15-Sliu.pdf to see more on the path planner and report-Sp15-dHeideman.pdf on the path follower.

4 Methodology

4.1 Combining Efforts

Daniel and I combined our code in the Turtlesim environment and after some bug fixes

and minor tweaks, we have the path planner running. Here's a short Youtube clip showing a "turtle" in action:
Dubins curve demo
And for comparison, here's a short clip showing how the original first_turn_then_move method looks like:
First turn then move demo

So how does everything work together? He's a graph that illustrate what's how all the nodes work together:

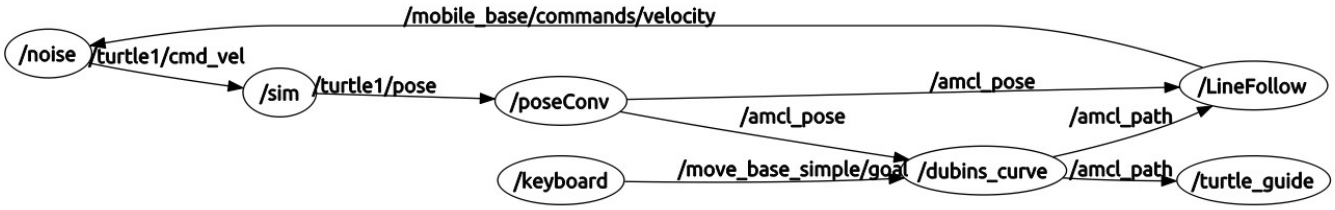


Illustration 1: Rqt graph for Dubins Path Testing in Turtlesim

There are seven nodes on the graph:

- **/sim** node displays the Turtlesim window and the simulation being ran on it.
- **/poseConv** node converts position Turtlesim publishes to the format that dubins_curve take and republish that information to the topic /amcl_pose
- **/keyboard** node is used to manually input a goal location for the turtle to navigate to.
- **/dubins_curve** node is the path planning file I worked on during Spring quarter where given the initial position of the turtle (from amcl_pose) and a goal position (from /move_base_simple/goal), it will generate a Dubins path, and publish it as an vector of positions.
- **/LineFollow** node is the path following file Daniel worked on during Spring quarter where given an vector of positions, it will follow that path using PID.
- **/turtle_guide** node creates a "guide" turtle that's been teleported (a turtlesim function) along the path to create a visual guide line of the path the turtle should be following.
- **/noise** node attempts to simulate constant and random noises that the Turtlebot might face when the code is migrated into the Turtlebots. I will cover this more in detail in the next section.

4.2 Noise Testing

Having an algorithm working in Turtlesim is quite different from getting to work in real life on a Turtlebot. So before migrating the code to the actual Turtlebot, I did noise simulation to see how the a Turtlebot might react, hence, I wrote the noise node, which adds constant and random noise to the linear and angular parts of the command velocity. I also added constant and random noise to the poseConv node to simulate camera noise. This table summaries what I found while conducting noise simulation on the Turtlesim.

Noise Source	Camera		Navigation	
Noise Type	Position	Orientation	Linear Velocity	Angular Velocity
Possible Cause	Delay (when transmitting data to Turtlebot through internet)	Misaligned ArUco marker. (ArUco markers are used in conjunction with an overhead camera to localize the Turtlebot	Hardware calibration, Road condition (slopes, bumpy road, etc.)	Hardware calibration
Noise Range Tested	Test on UCSD Protected Wifi.	0-17 Deg (3% of Pi) to either side.	0-20% increased/decreased velocity.	0-17 Deg to either side
Possible Consequence	Turtlebot lose track of the path and depending on the delay, may or may not recover.	Goes off track. The PID's attempt to correct the path usually cause the Turtlebot to run parallel to the original path at an offset distance.	Overshoot the curves at times, but most of the time can recovery.	Goes off track. The PID's attempt to correct the path usually cause the Turtlebot to run parallel to the original path at an offset distance.
Possible solution	Fast and stable internet access	Adjust ArUco maker	Calibrate hardware correctly, use accelerometer to monitor the speed.	Calibrate hardware correctly.

Table 1

4.3 Code Integration to Turtlebot

The goal for upgrading path planning is that Turtlebots can be deployed with the algorithm for the robotics testbed. Thanks to the ROS framework, upgrading the first_turn_then_move algorithm can be done by replacing the First_turn_then_move node with the Dubins curve node which subscribes and publishes to the appropriate topics. Here is an illustration of the first_turn_then_move node with the topics it associates with:

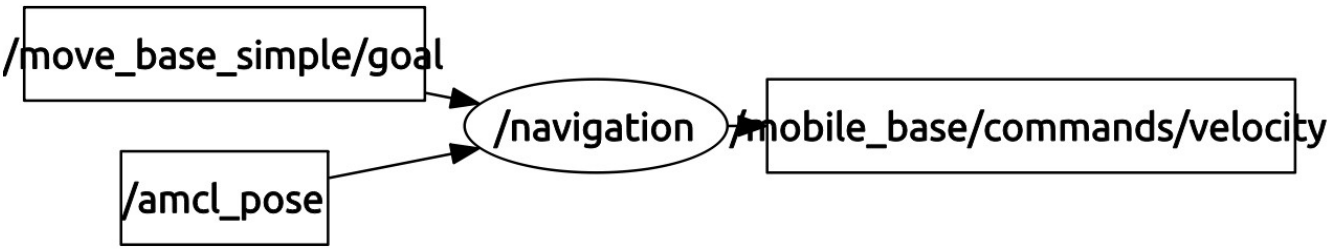


Illustration 2: First_turn_then_move node

The navigation node, which is used as a generalized name for the first_turn_then_move algorithm it contains, subscribes to the topics /move_base_simple/goal and /amcl_pose to listen to the goal position and the Turtlebot's current position respectively. It publishes the command velocity to /mobile_base/commands/velocity to navigate the Turtlebot base. Thus to ingrate Dubins curve into the system, the nodes Daniel and I worked on need to do the same. Illustration 3 shows how the dubins_curve algorithm and Linefollow PID is integrated into the system to replace the original navigation node.

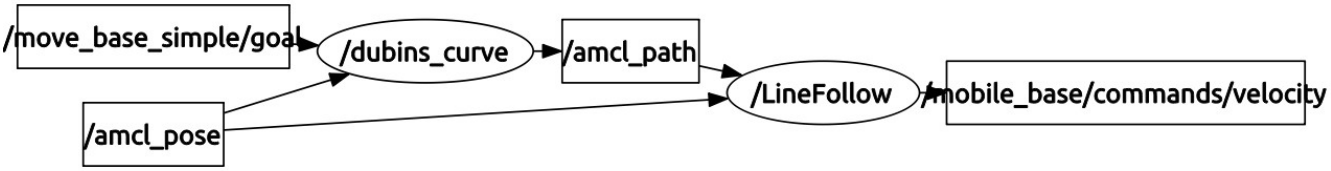


Illustration 3: Dubins curve

4.4 Set up

4.4.1 On Master Computer (Running Ubuntu 12.04)

1. If ROS Hydro is not already installed install Ros Hydro[1].
2. Clone the Github repository for ucsd_ros_project[2] on your local machine.
    1. Please follow the Getting Started[3] page on the ucsd_ros_project Github Wiki.
3. Check out the dubins_curve branch:
    1. Bring up a terminal window and navigate to ucsd_ros_project directory. (If you followed the Getting Started[3] page, then you can type "cd ~/git_catkin_workspace/src/ucsd_ros_project" at the command line").
    2. Type "git checkout dubins_curve" at the command line.
    3. Type "git pull origin dubins_curve" at the command line. It will prompt for your username and password for Github.
4. Compile the code.
    1. Type "cd ~/git_catkin_workspace" at the command line.
    2. Type "catkin_make" at the command line.
    3. If the code fails to compile, it is probably because there are missing ArUco libraries. Install the appropriate libraries and compile again.

4.4.2 On Laptops Controlling the Turtlebots (Running Ubuntu 12.04)

1. Either login to the Turtlebot Laptop physically, or ssh into the Turtlebot from the master computer by typing "ssh turtlebot@[turtlebot_name]bot.dynamic.ucsd.edu" at the command line.
2. Follow steps 1-3 from section 4.4.1 above, same as the master computer.
3. Compile the code.
    1. Type "cd ~/git_catkin_workspace" at the command line.
    2. Type "catkin_make" at the command line.
    3. If the code fails to compile, it is probably because there are missing ArUco libraries. However, the code running on the Turtlebot does not depend on those ArUco libraries. Type "catkin_make" at the command line a few more times until the remaining code is 100% compiled.
4. Add ROS_MASTER_URI of the master computer to .bashrc file.
    1. Type "vim ~/.bashrc" at the command line. (Can be open with any text editor)
    2. Add "ROS_MASTER_URI=[the_IP_address_of_the_master_computer]" to the file. (e.g. "ROS_MASTER_URI=http://mencia.ucsd.edu:11311/").
    3. Save and exit the file.

4. Type “source ~/.bashrc” at the command line.

4.5 Launching Turtlebot

4.5.1 For Master Computer

1. First terminal – roscore
   - Type “roscore” at the command line
2. Second terminal – Localization
   - Type “roslaunch turtlebot_camera_localization localization_demo.launch” at the command line

4.4.2 On the Turtlebot

1. First terminal - Deployment
   - Type “roslaunch ~/git_catkin_ws/src/ucsd_ros_project/deployment/turtlebot_deployment/launch/deploy_robot_dubins.launch” at the command line.
2. Second terminal – Goal Position via Keyboard Input
   - Type “roslaunch ~/git_catkin_ws/src/ucsd_ros_project/deployment/turtlebot_move_to_goal/launch/keyboard.launch” at the command line.

5 Tips

5.1 For Basic ROS

Going through beginner's tutorials[4] for ROS will help with understanding the structure of ROS. The particular tutorials that should be attempt before handling this project are #1-8, 11, 13, 14, 16 in the beginner's tutorials[4] section. I would also recommend reading #19 on Navigating the ROS wiki[5] as you will be using the ROS Wiki pages very often.

Understanding the ROS API for geometry_msgs and nav_msgs[6] will be of great help for updating the subscribers and publishers for motion control.

5.2 Helpful ROS Tools

ROS projects can get complicate as more nodes and topics are being added. **rqt_graph**[7] is great tool to keep track of your nodes and topics. To see the graph, type “rosrun rqt_graph rqt_graph” at the command line.

For debugging, **rostopic**[8] can be used to check to what's being published in each topic. To get information from a topic, type “rostopic echo [name_of_the_topic]” (e.g. To see the current position of the Turtlebot “leonardo”, I would type “rostopic echo /leonardo/amcl_pose” at the command line.)

6 Pitfalls

6.1 ArUco Markers Tracking

My first few attempts to test the code on the Turtlebot was unsuccessful because the camera lose track of the ArUco marker quite a lot, and the navigation algorithm was not prepared to handle that. The problem was mainly due to the lighting and the ArUco marker itself. After reprinting the ArUco marker and changing the lighting in the testing room, there's significant improvement. Also worth noting is the ekf (Extended Kalman Filter)[], which predicts the position of the Turtlebot if the camera loses tracks of the ArUco maker.

6.2 Missing Libraries

I learned to launch the Turtlebot based on the “Launching the Coverage Control Demo” page on the github wikipage of ucsd_ros_project on the lab computer “mencia”, and everything worked out fine. However, the code in the “dubin_curve” branch of the github does not contain libraries that's already install on “mencia”, thus the necessary libraries need to be download and installed for the code to fully compile on another machine.

6.3 Unit Conversion

The camera node publish the position of the Turtlebot in pixels relative to the camera, but the Turtlebot's command velocity is in meters per second. It would be better to have a unified unit for all aspects of our project, or conversion can easily occur.

6.4 Scaling of Line Follow Algorithm

As of now, the line follow algorithm is does not scale well if the size of the work environment is changed significantly (e.g. from 11m x 11m to 3m x 2m), so it would be good to find out how to automatically scale the constant in the line follow algorithm to suit different testing environments.

7 Conclusion

After putting the line following PID and Dubins curve algorithm that Daniel and I worked on together, I did some noise simulation test on Turtlesim to explore the possible causes of noises and their potential effects and solutions. With the goal of setting up the testbed for inter-robot communication and coordination, I migrated the code out of the Turtlesim environment and integrated it into into an existing deployment project to test on the Turtlebot. After some adjusting and try and error, a Turtlebot can now be launched with the dubins_curve algorithm as the path planning algorithm. However, deploying a single Turtlebot under the current condition can only serve as a prove of concept. For this path planning algorithm to work smoothly in a multi-agent deployment, flexible line follow algorithm and collision avoidance are our imminent challenge for the motion planner.

References

[1] Ubuntu install of ROS Hydro http://wiki.ros.org/hydro/Installation/Ubuntu
[2] Repository for UCSD ROS Project https://github.com/evangravelle/ucsd_ros_project
[3] Getting Started—How to set up repo for ROS https://github.com/evangravelle/ucsd_ros_project/wiki/Getting-Started
[4] ROS Tutorials http://wiki.ros.org/ROS/Tutorials
[5] Navigating the Ros Wiki http://wiki.ros.org/ROS/Tutorials/NavigatingTheWiki
[6] ROS Common Messages http://wiki.ros.org/common_msgs
[7] rqt_graph http://wiki.ros.org/rqt_graph
[8] rostopic http://wiki.ros.org/rostopic