Randy Lewis

Cooperative Turtlebot Room Mapping

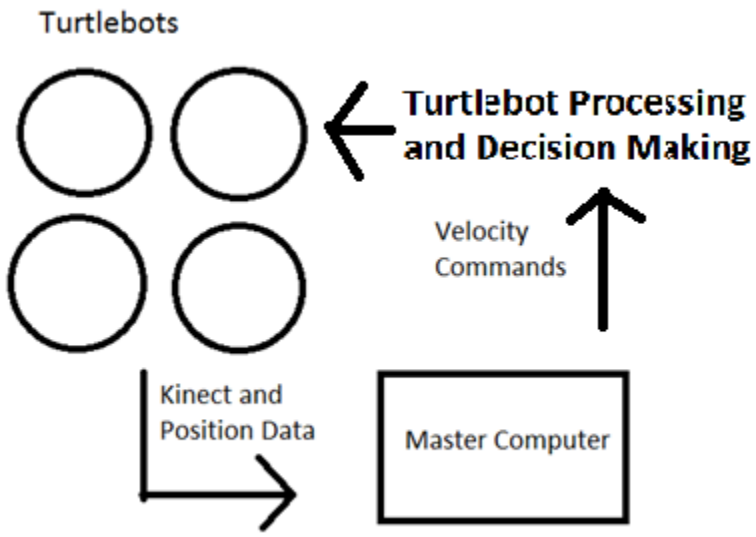
Abstract

The goal of this project was to write an algorithm to coordinate multiple Turtlebots using ROS to map a room quickly and efficiently. This quarter, besides further contributing to proper setup of the Turtlebots, I wrote a node to subscribe to the name specific Kinect data from two Turtlebots, and then to publish both of those data sets to the name general topic for use in Rviz. Overall, this quarter we continued to set up the basics mechanics of the project, but we still need to fix some glitches, and then write the algorithmic part of the program.

Big Picture

The Cortes and Martinez lab is using ROS (a library written within C++ and Python to give robot developers a generic tool for programming their robots) and Turtlebots (commercially available ROS compatible robots that are essentially upside down buckets with wheels, a laptop, and a variety of sensors) to explore cooperative coordination algorithms.

This and last quarter I have been working on a project with Daniel Heideman to use ROS to coordinate multiple Turtlebots to cooperatively map a room with their Kinect sensors and then to localize themselves in that map. An algorithm already exists to map a room with one Turtlebot (SLAM Gmapping), so the focus of the project is to harness the power of multiple robots to make the mapping process faster and more accurate.



A parallel project, pursued by Katherine Liu, aims to write a similar algorithm to map a room and localize Turtlebots in it, using one fixed overhead camera rather than the various

Kinect sensors on the Turtlebots. Although initially, both of these projects are intended to stand alone, in the end they may each be merged in order to improve each other's accuracy and ease of execution.

My Contributions

My contribution to the project this quarter included general contributions to fine tuning the setup of the Turtlebots and their operating systems, and a preliminary version of a node that subscribes to name specific Kinect data from multiple Turtlebots, then republishes that data to the corresponding name general topics for use in Rviz.

The main changes we made to the operating system setup of the Turtlebots this quarter were to update the robots to Hydro (the newest version of ROS), and then to fix some screwed up drivers relating to the Kinect and its data publishing ability. Last quarter, the Turtlebots running updated Groovy (the last version of ROS) lacked the correct nodes to publish the Kinect data they collect. Updating to Hydro solved this problem. However, updating to Hydro also caused a new problem, namely that when launching 3dsensor.launch on the Turtlebots, the operating system would give the error "no devices connected" as if the device were not plugged in. In the end, the problem turned out to be corrupted OPEN_NI drivers for the Kinect, and the problem was solved by removing specifically these drivers, then reinstalling them. Once this problem was solved, there was still another roadblock to visualizing the Kinect data in Rviz. Although when we ran 3dsensor.launch on the Turtlebots, the correct topics were published, and the operating system successfully recognized the Kinect, the actual depth data published was just arrays of 0's and NaN's (not a number). In the end, this problem turned out to be a glitch triggered by a variable called "depth_registration", and the solution was to run 3dsensor.launch with depth_registration equal to false.

After finally solving these operating system related Kinect problems, we simplified the process for running 3dsensor.launch correctly by modifying the name specific launch files created last quarter. The launch files launch minimal.launch and 3dsensor.launch inside of name tags specific to each robot, and now include the specification that the parameter depth_registration=false. Thus, running these nodes on any particular robot causes that robot to start up its mobile base and Kinect, with all the published topics prefixed by that robot's name (for example, giottobot/scan or titianbot/mobile_base/commands/velocity).

The node that I wrote this quarter works on top of the setup just described. It subscribes to Kinect data from multiple robots at once (currently the only two correctly set up robots—boticellibot and titianbot), stores each message received in a global variable, and then republishes those messages (from each robot) to the name general topics. So for example, the node subscribes to /boticellibot/scan and /titianbot/scan, storing each message in a global variable, while constantly publishing both global variables to the name general /scan topic for visualization in Rviz. This node does not yet work perfectly, and is also just a proof of concept

for the larger program as a whole. The final program will need to combine messages of the same type from different robots in some kind of algorithmic way (most likely simply splicing data together when it is about different parts of the room, and averaging data when it is about the same part of the room), and then to republish each combined message to Rviz. Currently, the node has problems with publishing old data over and over again (and not updating when new data is published), and does not contain any algorithmic combination of data.

Both of these nodes are located on the mencia computer on the "randy" account in the directory /catkin_ws/src/turtlebot_room_mapper/src/, and can be viewed and run from there. Also, the code is included at the end of this report under the "referenced code" section.

Methodology

Kinect data subscriber/republisher

1) All the desired topics are imported (such as, for example, nav_msgs/Odometry, and sensor_msgs/Laser_Scan) for later use in the program.

2) The global variables that will store Kinect data messages for specific Turtlebots are defined.

3) The program defines "callback" methods for each individual robot's data topics that will later be run each time the master computer receives a message over this topic from a robot. The callback methods print the data to the screen with the appropriate robot and topic name, and store each message inside the corresponding global variable.

4) Inside the main class, subscriber objects are created for each robot for each desired topic corresponding to the callback methods. These objects start to continually subscribe to their topics (and thus store and overwrite messages in the global variables) upon creation.

5) Next to each pair of subscriber objects (one for each robot per topic), corresponding publisher objects are created (one per topic).

6) Inside an infinite loop, each subscriber object continually publishes whatever data is stored inside the global variable for that topic for both robots.

7) The loop is exited when the user presses ctrl-c.

The final version of this program will include a method of algorithmically combining corresponding topics in between steps 5 and 6.

Tips

There are several tips I would give to anyone attempting to take over the project where we have left off:

1) Remember to add these lines to end of your bashrc file:

source /opt/ros/groovy/setup.bash //(or hydro or whatever version you're using)

source ~/catkin_ws/devel/setup.bash

If you forget these lines, things don't work, and it is very difficult to figure out what's wrong. Putting them in your bashrc runs them every time you open a new terminal window, so you basically never have to deal with them again.

2) Remember to modify your Cmakelists file to include paths to each node that you write. You would think ROS would automatically know where to look for them in compilation, but if you don't put these two lines in your Cmakelists file, trying to compile with "catkin make" will not work properly:

add_executable(<desired node name> src/<actual c++ file>.cpp)

target_link_libraries(<desired node name> ${catkin_LIBRARIES})

3) Make full use of the ROS tutorials and the ROS answer service contained on the ROS website here: http://wiki.ros.org/ROS/Tutorials and here: http://answers.ros.org/questions/. Although Google has been helpful to me in this project, it has not even come close to the combined power of these two sites to provide me with examples and answers to specific questions.

4) If things aren't working the way you expect, disconnect from the Turtlebots, close all open terminals, and restart both the master computer and the Turtlebots before assuming the problem is more complicated. There are many problems that go away mysteriously after following some or all of these steps.

Pitfalls

There are several pitfalls that have gone unsolved this quarter. The first was the inability to successfully rename all the transform topics in the name specific Turtlebot launch files. A line was added to change the "tf_prefix" parameter, which is supposed to change the prefix of all the transform topics, but did not work for /odom for some reason that is as of yet unknown. Because of this, when running multiple robots in Rviz with their name specific launch files, the robot models do not load correctly. One visually glitch robot loads, which moves when either robot is moved (since both robots are publishing their transform odometry messages to the same topic). This is a significant problem because multiple Turtlebots need to be able to work together in Rviz for the current approach to the program to work.

The second unsolved problem (encountered last quarter and still unsolved) was the inability to find a launch file command on the central computer that would launch other launch files or nodes on the Turtlebots (such as, most importantly, the name specific base and camera startup launch files). The only way to run these files is currently either to remotely connect to

each Turtlebot using ssh, and run the programs from the command line, or to physically open the netbook laptops and run them from there. This is another vexing problem, because, without human setup, it eliminates the possibility of a stand-alone program on the master computer mediating all the Turtlebots and the possibility of dividing the processing power needed to run the final program between the various computers. This problem seems to be due to an error on the part of Clearpath Robotics, since Evan said that he remembered being able to launch launch files on one computer from a launch file on another computer in Fuerte (a version of ROS preceding Groovy and Hydro), but that Groovy seemed to lack this capability.

Referenced Code

Kinect Data Subscriber/Republisher:

-can be found in svn in folder lab/in-progress/SLAM/kinect_data_condensor_republisher.cpp

Conclusion/Future Direction

This quarter, we continued to figure out most of the basic setup requirements to write the actual bulk of the program. The Turtlebots Kinect sensors are now functional to the point where we can actually subscribe to accurate data coming from them, and the node that serves as a proof of concept for the final program is mostly written.

The final program needs to launch the appropriate startup files on the individual Turtlebots, to subscribe to all their Kinect data, and then to control the robots while publishing a map to Rviz. In order to accomplish this, both previously mentioned pitfalls need to be solved, and the actual algorithmic part of the program needs to be written.