Final Report -- Documentation

Team Deja Vu *(Mini Driverless Vehicle)*

Walker Finlay, Wes Olsen, Kingsley Leighton, Olivia Grimes

Walker

The transition to remote learning was disappointing, given that our specific project is so hands-on. It took a lot of the fun out of the work, but I did my best to get our robot’s software skeleton (the graph and subgraphs of ROS nodes) communicating with one another. I did not accomplish as much as I would have liked, mostly due to the large time overhead from learning ROS; which, in my opinion, is poorly documented for how idiosyncratic it can be. The evidence is that most of the progress we made was after reading ad-hoc tutorials and semi-related questions on ROS forums, as opposed to official documentation. Setting up stage and working with ROS ConstPtrs were especially weird.

That being said, the perception subgraph works soundly so far. The perception subgraph consists of 5 nodes sending empty messages. The perception subgraph nodes are defined according to our original layout at the beginning of the semester, including camera, imu, lidar, odometer, and a node responsible for aggregating and subsequently transforming sensor messages, and finally passing the custom message to the mapping node. Note that I omitted the gyro node because the IMU includes gyroscope information. All of our message needs were fulfilled by ROS common msg types, from the `sensor\_msgs` and `nav\_msgs` types. Drivers exist for almost all popular sensors on the market, so although we do not have the physical sensors to test, we can be confident that they will send the data our graph requires, and our message types will comply with the data they send.

Immediate next steps (this summer?):

* Set up perception nodes for simultaneous subscribing (from stage) and publishing (to mapping)
* Update launch file to include stage and mapping
* Define custom message type for (aggregation)--(mapping)
  + Having huge trouble with this - maybe define custom messages in their own package? Learn how CMake and package.xml work?
* Integrate Olivia’s motion model into the mapping node

Next steps beyond:

* Set up the rest of the graph (mapping/planning subgraph, locomotion subgraph) including manual locomotion via controller
* Work with physical robot
* Testing
  + Not necessarily unit-testing, but data collection, so we can debug faster when the robot is able to move

Ambitious next steps:

* Jumps and drifts
* Recognize street signs and hazards
* Improve the ROS documentation

Wesley Olsen

Most of my time on this project was spent on working with player/stage and StageRos. A lot of our time as a group was eaten up by trying to get things to run for the first time, player and stage is a very good example of this. StageRos could be used for the same purpose as player/stage (which was what we had originally worked with) which was to be able to communicate with the nodes that received the data from the simulated sensors made in our .word files. For reference. world files are essentially what the simulation is, it is a collection of models (models are every defined object in the world; the car, walls and other objects are all models) that can be interacted with. You can make the robot in the .world file itself but it is recommended to make it outside and simply “include” it so you can basically drag and drop your robot into a handful of different simulated worlds. .cfg files are what stage reads to get all of the information about the robot and its sensors and how the robot will interact in regards to our motion model. Making a working simulation required a couple basic things like a floor plan for which everything could be defined on (essentially what every other model exists on). The robot was made by mapping out an area (a rectangle in our case) and attaching a sensor which is called a ranger. Ranger sensors can be identified as many different sensors (although sometimes it doesn’t matter as long as things are coded correctly for example Lidar and Sonar are essentially the same as they both detect solid objects in the word at the range and field of view you specify the same way in a simulation). For our real robot we have a Lidar attached so we have a Lidar on the simulated robot as well. The information it picks up will work in tandem with our motion model that Olivia has worked on. In the coming semester if we are on campus physically then the plan will be to transfer the virtual simulation into a physical one where hopefully we won’t have too many hold ups as we now have a sense of how everything should work together. If we have to do the second part of our project online then we can continue to use our virtual simulation but things are a bit foggy as to what our final goal will be if we cannot meet to work on a physical car.

(editing code must be done in using sudo)

Picking up where I left off this semester to run the program again simply go to /opt/ros/melodic/share/stage/worlds

rosmake stage\_controllers (don’t have to do this again if you have done this before)

roscore$ (if this fails it is likely due to a path error which can be fixed with a similar code to the following)

export LD\_LIBRARY\_PATH=/opt/ros/melodic/lib:$LD\_LIBRARY\_PATH

rosrun stage\_ros stageros simple.word (or whatever .world file you are trying to run)

The data for the nodes can be found by using the rostopic commands (if there is a problem with finding data (there are only two things there while there should be more) it is likely the error is a result of the clock which can be fixed as follows)

rosparam set use\_sim\_time false

T. Kingsley Lehigton -

My main focus this project was getting all of the platforms, frameworks, and libraries working for each person in the group. After that my focus was helping Wes understand and implement a robot object in stage\_ros so that it published messages from its sensors, to connect with the network of nodes Walker worked on.

VirtualBox and Ubuntu -

The essential piece of software we had to use was ROS(Robot Operating System) and we needed to select an OS that ROS worked well on. We decided to use Ubuntu 18 on VirtualBox.

This presented a significant number of difficulties right away. Wes and I were unable to use VirtualBox due to some BIOS settings we had to change, which did not allow us to create a virtual drive for VirtualBox. Olivia tried to get Ubuntu to work on VirtualBox, but it crashed and never worked again, so after failing to get it to work for weeks it was decided to just have the other three members work with ROS, and Olivia would manage the motion model.

After getting Ubuntu up and running I needed to install drivers so that I could change the screen so that I could actually see what I was doing. The screen was tiny and extremely distorted. I was able to install the drivers using the following procedure:

Open terminal and enter: sudo apt install build-essential dkms

Go to *Devices -> Install Guest Additions*

Then go to the location of the .iso file and enter: sudo ./autorun.sh

Installing ROS Melodic and stage\_ros -

The next step was installing ROS which was an arduous experience. We had tried to install ROS melodic with python 3 instead of python 2.7, but it did not work. It was discovered that while the wiki on installing ros says simply build the catikin\_workspace folder with a different command using python 3 inorder to have ROS work with python 3 that was in fact not the case. All of the dependencies for ROS are in python 2.7 and inorder to get the python 3 versions of each of them or changing the environment variables to point to python 3 instead of 2.7 was a wild goose chase which ended in disappointment. Eventually we gave up on python 3 even though it would have been more convenient it was taking far too much time to work out.

The next step was installing stage\_ros. This was painful.The installation documentation everywhere is either years out of date or in pieces. There also were multiple sources for the stage\_ros plugin which got us into a lot of trouble with installing multiple or different versions. There was also a lot of playing around with environment variables in order to get everything to build correctly, as well as installing other libraries and dependencies which building stage\_ros required to build. After it was installed, all of the tutorials were also majorly out of data so we had to essentially go through a semi informed trial and error even to get the correct command to run a stage program. The best general guidelines to getting stage\_ros installed are use the github repository on the ROS wiki, and make sure the environment variables for LD\_LIBRARY, ROS\_PKG, and STAGE\_PACKAGE are correctly set so that it can be built. Each person had different issues along the way but the main issues were solved by ensuring all environment variables were correctly set. It’s also best practice to add the new value for the env var’s in your source.sh file so there is no need to reset them for every new terminal window.

Player/Stage and stage\_ros hangup:

The longest issue in the project was understanding that through all of the partial tutorials with some pieces out of date and others talking about player/stage vs stage\_ros, that we did not need player, since ros functioned as player in the player/stage dynamic, and the stage\_ros took the place of stage. Simple issue, but it took us 3 weeks to move past that issue and realize we could just focus on stage\_ros.

After The general platforms and technologies were set up for Walker, Wes, and me. I began to assist Wes in understanding Stage\_ros and how it functioned, especially the conceptual part of how the pieces fit together in stage\_ros, i.e. the different file types, objects, models, and sensors. This is all detailed in Wes’s part, but a lot of it was a collaboration between me and him.

Goals for next semester -

I would love to be able to have a more independent role in the project if possible developing the robot in stage\_ros and the node for mapping the robots environment, but if that is not what the group needs then I will continue what I am doing.

Olivia

→ a majority of what I did (technically) since being remote was create the motion model

**Currently:**

Written in C++

Simply compile and run the file to try out the motion model in its current state:

Currently, the motion model runs in a loop, asking the user to provide a steering angle and velocity, and it prints your new position, prompting you to continue moving if desired (at which point you can add a new steering angle and velocity, or quit the program)

**Next Semester:**

* Figure out a good DeltT (right now is 1 second)
* Change it around to work together with our network of nodes
  + Make it work with having to read in sensor data, and just in general with the mapping node
* Figure out how Ackermann Steering will work into this when we use the physical model
* OR: switch to using a built in mapping function from the ROS library?
  + Might be easier than creating one manually

**Goals For This Summer:**

* Learn more about ROS and try to get it running on my computer
* Understand more about how to integrate the motion model into the nodes, in case we take this route → learn about other option too (built in mapping function in ROS)
* Maybe: come to Lehigh and meet as a group to try and work with the physical robot??

Conclusion:

Our project was definitely hindered by only being able to work online with our project. That said, we are in relatively good shape to make a lot of progress next semester although what we plan to accomplish is highly dependent on whether or not we will be able to meet in person in the fall. If Lehigh is open, then the first order of business is to get our car to function properly (move, react to commands, give reliable sensor data). We believe that will be the biggest obstacle as none of us have any expertise in physical robotics, but that may change this summer as we do plan to work on our project to some extent over this time. At least one of us is also taking Montella’s Robotics course, maybe even 3 of us, so that could also help with this project come fall. After we get the car to work we will transfer our work from our simulation which should put us in a good spot as we have our motion model and nodes working. We would do this by way of using the network of nodes to interact with the physical sensors they way they interacted with the simulate ones from stage\_ros, if they do not port easily we will have to do some adapting, but regardless it should be far better than starting from nothing. After that it is a matter of improving efficiency and figuring out how to add drifting and jumping to our motion model. There is also the possibility of figuring out how to get our actual car to work even with quarantine which if we can do we can go about our fall semester plan as mentioned above**.**

If Lehigh continues to do online classes next fall and that is not an option then we will continue working with our stage\_ros simulationand make it an actual good representation of how we want our real robot to function. The next step would be to create a node which maps the environment from the data published by the sensors, hopefully this can be done using RViz, a built in mapping application in stage\_ros. Then we would create the node which based on the perceived environment decided where to go, this would have to be in conjunction with make the actual environment, either a city of some kind or a tract at least. Another possibility is to explore if something like traction and gravity can be manipulated in the simulation as to enable the possibility of drifting and jumps. Some exploration obviously has to be done to decide what we want/can do in stage\_ros. Either way next semester ends up we certainly have more to work on and also specific concrete goals that will allow us to start the semester having direction and in the case of remote classes hopefully we will do enough research to get down a more definite plan as the summer goes on as we talk with our mentor.