Midterm Progress Report

ARC - Autonomous RC

Senior Capstone Project

Oregon State University

Winter 2017

Tao Chen, Cierra Shawe, Daniel Stoyer

Version 1.0

February 17, 2017

CONTENTS

1 Project purpose and goals

The purpose of the Autonomous RC (ARC) project is to determine if it is possible to build an autonomous RC vehicle using commodity components, meaning components that are relatively inexpensive and can be bought at places like Radio Shack®, Best Buy®, or on Amazon.
Our goal is to make an RC vehicle navigate autonomous to a given waypoint/location, preferably at a high rate of speed and with obstacle avoidance. Stretch goals are to make the vehicle drift around corners and parallel park.

Fig. 1. Drifting example. Image from https://autorally.github.io/

While our main goal is to have a functioning autonomous RC vehicle we also hope that we can produce instructions that RC enthusiasts can follow to produce a functioning, consumer-grade autonomous RC vehicle of their own.

2 Cierra

2.1 Current Status

Over the past six weeks, my focus has been acquiring hardware, modifying and building hardware and components, and working on the vision system. Currently, I have obtained all of the hardware we will need according to our design document minus the car. In the beginning of the term I modeled a mount that is adjustable and meant to be used with the two USB cameras we were provided. It is designed to be adjustable until we know how the cameras will need to be adjusted. After having printed, acquired the baby screws required, and assembled the camera, I found the cords to be messy. I fixed that with an expandable wiring sleeve that fits into the back of the camera case and expands in such a way that it doesn't slip out the back. The next step was calibrating the cameras, which had a very steep learning curve due to learning the libraries for openCV, which is a cross-platform vision processing library. As of now I am able to receive input from both cameras and create a disparity map. The main problem is the updates slow and even after calibrating the cameras and the depth is often distorted from the cameras trying to auto-focus. Hopefully, this week we will receive the Leddar M16, which is a Multi-element sensor module that uses 16 individual solid-state sensors to do time of flight sensing. The M16 was designed for industrial applications and the company provides a "trial" kit which

provides everything needed including the module, SDK, and C libraries for using the sensor. Due to the ability to detect multiple objects up to 150ft and providing a 45 degree view range, the M16 will probably end up replacing the stereo vision system, which isn't working as well as I would have hoped. The RC will namely need to track objects in front of it, so the 45 degree field of view should be enough. Using the M16 would also be able to allow us to switch to using a board such as the UPboard in place of the NUC, as it is only ever tracking 16 points at once. This is significantly less complex than having to run complex algorithms for pixel matching and point cloud creation that we would need to do for stereo vision.

We were given a UM7, which didn't have any headers on it, which meant we had no way to communicate with the sensory. I soldered on ends to the UM7, which was a challenge due to never having soldered and not understanding how wiring diagrams work. Currently, using the software provided by RedShift Labs, I am able to visualize the data on a Windows computer. This is rather cool, as once the major motion has gone away, if you place your finger on the sensor, the UM7 sensitive enough to notice a slight change in temperature and display your heart beat.

2.2 Left to do:

Once the Leddar unit arrives, this section could be vastly different. Most of the documentation for the unit is protected; however, it will be included when the testing package arrives. Once we have the unit in hand, I will begin testing if it will be able to replace the stereo rig. If the Leddar is able to preform to our needs, i.e. give fast enough readings at a wide enough range and we will need to see what libraries are available. In order to test whether the M16 will work, I would like to take it outside, and see if it is able to detect things such as trees, walls, curbs, and rocks.

On top of that, as I oversee the hardware component protection of this project, I plan on creating a case for any component that is currently exposed. As of now, that includes the Raspberry Pi 3 with the PXFmini shield, UM7, and a protective "roll" cage for the Intel Nuc, or possibly a smaller main computer if we are able to move away from the NUC, as the main challenge was being able to process vision. We will also need to ensure that we are properly able to power all of our extra hardware, which currently will mean adding an extra LiPo battery. As a group, we still need to be able to fully control the servos and Electronic Speed Controller (ESC), and once we know what hardware we will finally use, I will turn our mock up into either an array of components mounted onto an acrylic/polycarbonate plate, or a complete box that all components can be contained within.

- Something to do...

2.3 Challenges

Problems	**Solutions**
Problems that impeded progress.	Specific actions to resolve problems.

3 TAO

3.1 Current Status

My work was mostly on simulation of the platform on the computer. My strategies was to use as much AutoRally's code as possible. The AutoRally package contains a model description of the vehicle it uses. We are not using the same vehicle for our project. But it will be good enough for the simulation. When we get our hands on our vehicle, we can adjust some of the model parameters to best fit our vehicle. The AutoRally package also has a controller that is in Python to control the steering, which uses the Ackerman steering theory, and the forward/backward motion.

As for right now, I still haven't written any code yet. I teared apart the AutoRally package and tried to understand it. I migrated a minimum amount of code to our own package such that the simulation would still run. That being said, when we are ready to build our platform and start implementation, we will use the current package as the foundation. The simulation of the platform will not drive itself yet. It has to be controlled by a Xbox controller (See figures below).

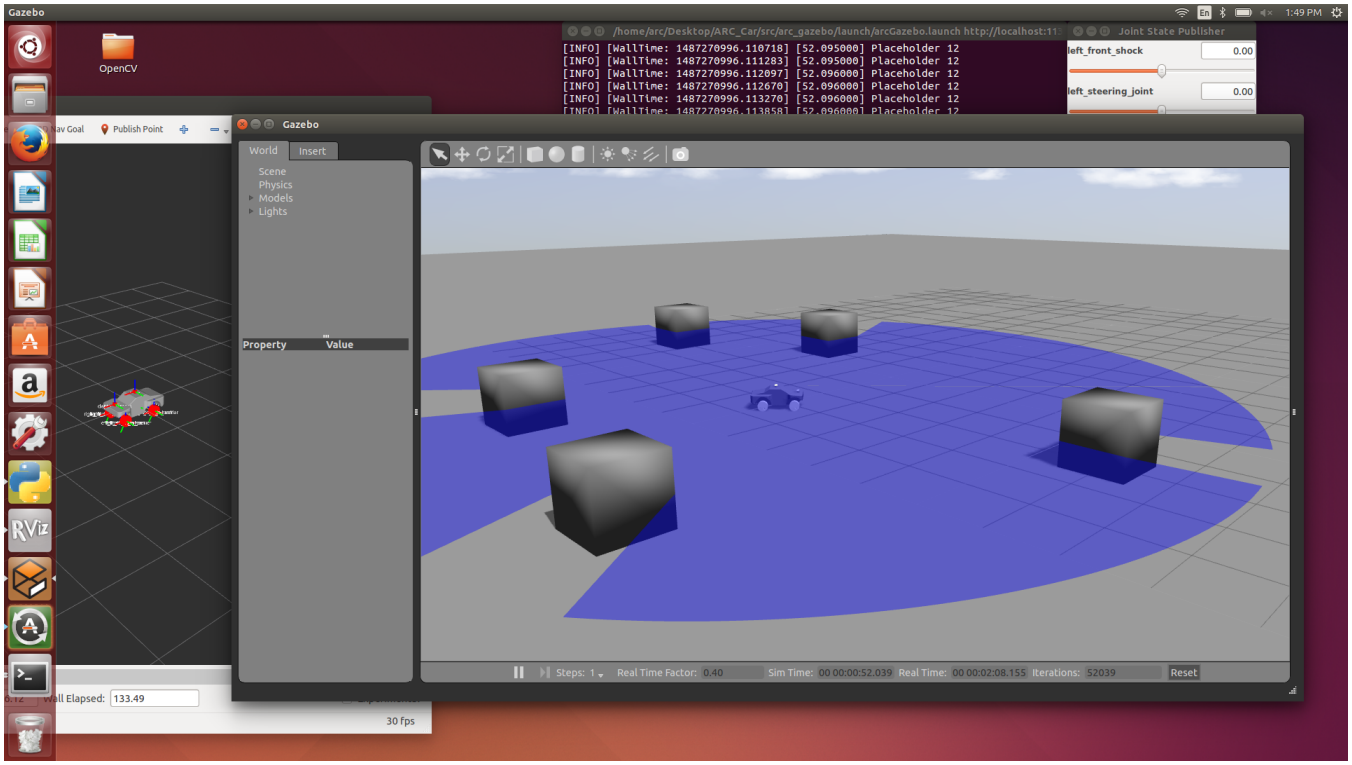

Fig. 2. Simulation on a computer

My area of focus specified in the design document are: motion model, path planning, and other algorithms. What I am doing right now is related to motion model. I need to get my hands on the actual vehicle to determine the parameters for the motion model. I am not sure if our vehicle uses the Ackerman steering model. If it does, the controller provided by AutoRally is perfect. Otherwise, I will have to assume that our vehicle does use the Ackerman steering model. Luckily, our vehicle is not as big as the one that AutoRally has. The steering model will only have minor effects on the simulation.

In order to do path planning and experience on other algorithms, I have to have the simulation running first. So I have not started this part yet. Once I have the simulation ready, I will start on the planing algorithms. I will create a publisher that subscribes to data that describes surrounding environment and publishes new waypoints accordingly. This publisher assumes the map of the world is known.

3.2 Left to do:

- To configure the simulation to accurately simulate our platform, I still edit some parameters in some confiuration and model files. For example, the size and weight of the chassis determine the innertial of the vehicle in the physical engine.
- The two sensors for vision are not working perfectly. They are outputing the right data. However, I can't visualize their outputs on Rviz. I could print out the lidar data on the terminal. I have not try the pointcloud data yet. Rviz keeps complaining that it cannot find the transform frame for the lidar and the camera. Strangly, I was able to visualize the output on Rviz if I used my old model file. I need to double check my model file and the launch files.
- To make the car drive autonomously with minimum obstacle avoidance. Currently, our package does not have the state estimator, the waypoint follower and the constant speed controller, which are all essential for atonomous driving. I need to integrate those components with sensor data to accomplish this task. All of these files resite in the core package, which are hardware specific. Since we are not using any of the hardware that AutoRally uses, I need to potentially take them apart as well.

3.3 Challenges

- Time is the issue. There are only 4 weeks left in this term. It is very hard to set a timeline for this project because we don't have a clear view of what is ahead of us. New issues arise constantly. I will try my best to complete the third item on the Left to do list in two weeks.
- I will set up the simplist obstacle avoidance procedure at first in the simulation, which is to let the car go from one point to another in a straight line but with a block in between. The vehicle should be able to go around the block to get to its destination. This requires the state estimator to know the current position of the vehicle in a global frame, as well as a waypoint follower to follow pre-defined waypoints. Waypoints are pre-defined point in the global frame. At start, the vehicle should steer towards the first waypoint and then follow the rest. A constant speed controller is also required so that the vehicle can keep its speed even the core software runs in cycles.

Problems	**Solutions**
Problems that impeded progress.	Specific actions to resolve problems.
Visualizing sensor data on Rviz.	Go through all associated files. If it still can't be resolved, I will not use Rviz for visualizing.
Getting the car to move without using a controller.	Chase down where the controller commands get sent to.
Getting a clear view of the package structure.	I need to start making documentation.

4 DAN

4.1 Current Status

Much of the work of the first half of the term has been setting up the software environment that will run our RC vehicle. We have our operating systems installed and configured on our primary computers: the laptop which is the ground control station (GCS), the Raspberry Pi 3 (RPi3) and PXFmini which is the autopilot, and the NUC which is the companion computer that handles the raw calculations. We have ROS installed and running on all platforms.
We have been working on getting the RPi3 talking with the PXFMini autopilot cape. The PXFmini autopilot is used to integrate sensor data to send to the NUC and to integrate data coming from the NUC to the vehicle controls. The autopilot also automates calibration of the vehicle and performs other convenient features. We have a system image for the RPi3, from Erle Robotics (the makers of the PXFmini) installed and are able to get the PXFmini launched and armed. This means that the autopilot is ready to take commands.

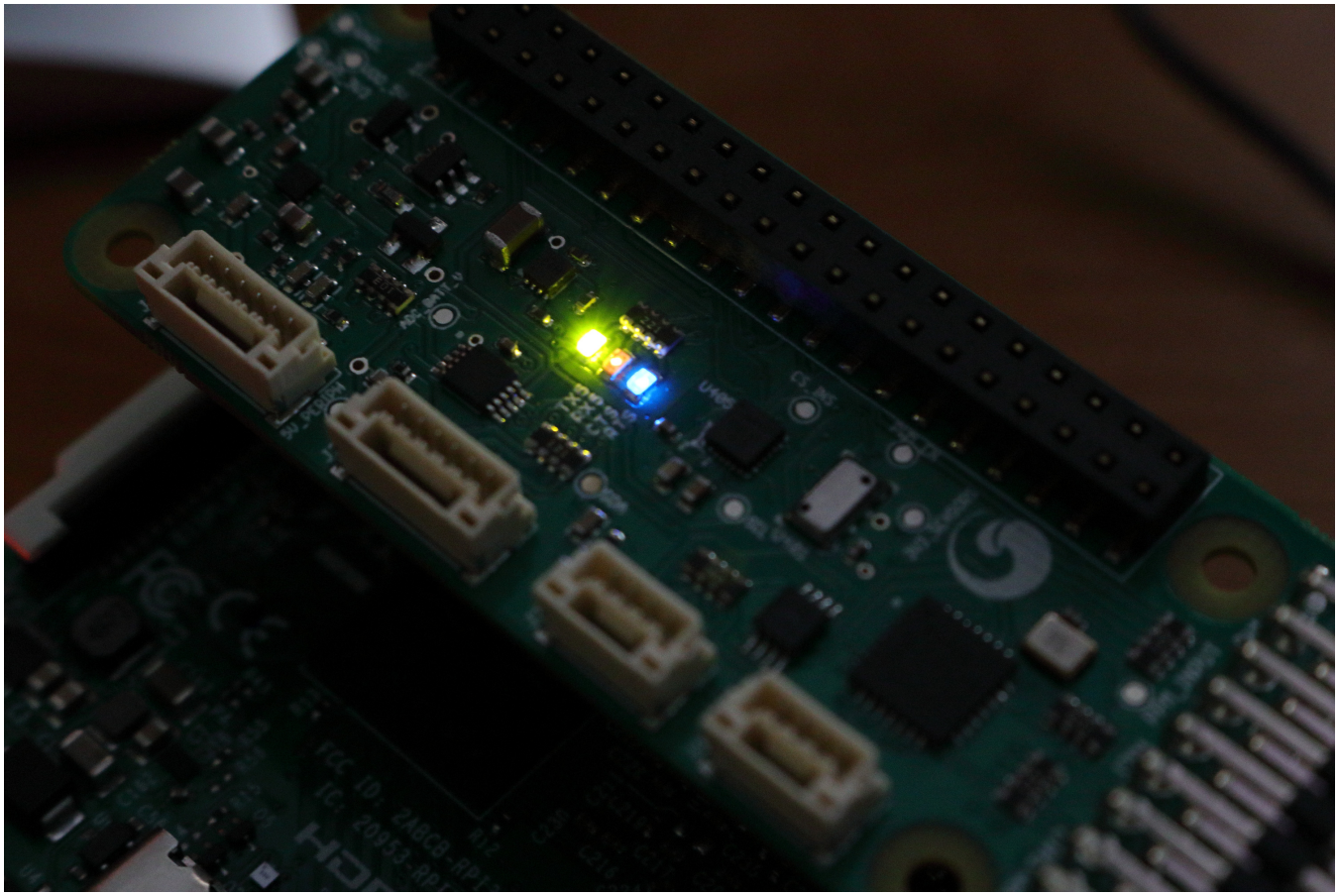

Fig. 3. PXFmini autopilot before launch and unarmed.

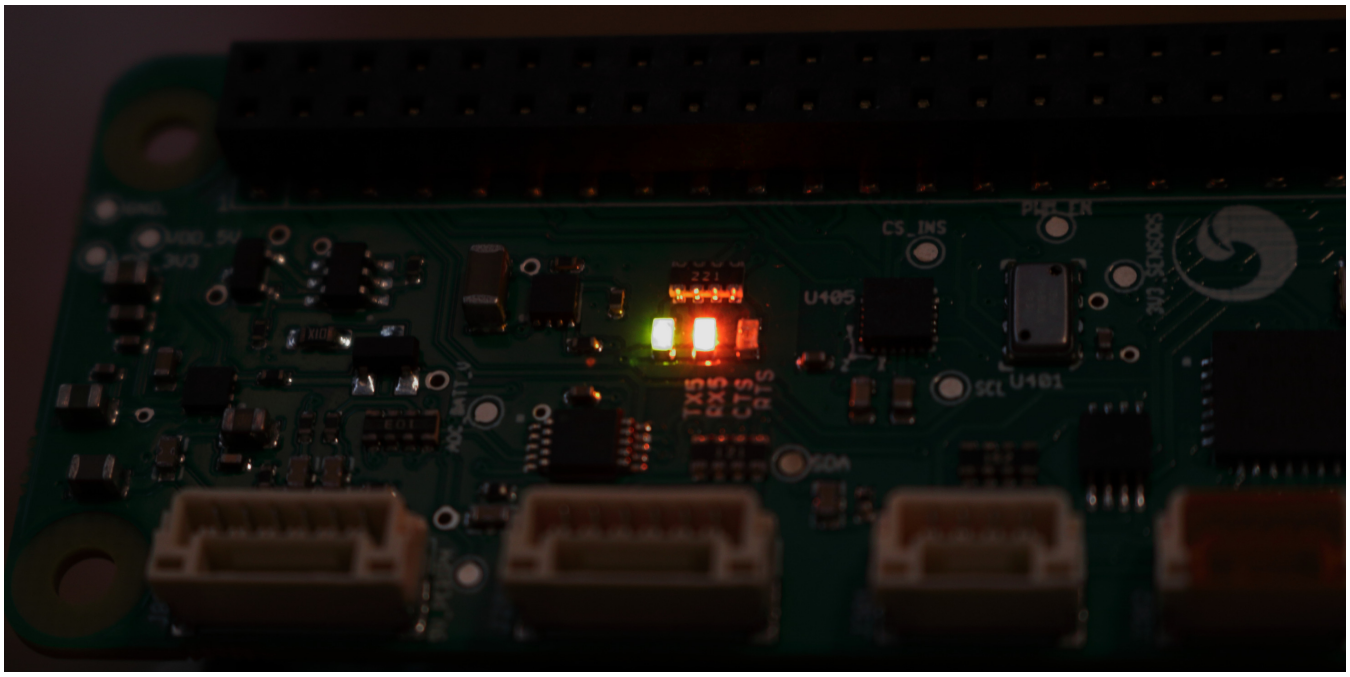

Fig. 4. PXFmini autopilot after launch and armed.

We have also been working on communicating computer-to-computer using ROS nodes. We have run successful tests over direct ethernet connections talking between ROS nodes. These tests lay the groundwork for the NUC to be able to send instructions to the autopilot.

While the following areas are all incomplete at this time. However, our accomplishments in each of them pave the way for us to reach the goal of an autonomously controlled RC vehicle.

All told, problems with the PXFmini have delayed us from actual hardware testing by about 3 weeks.

4.1.1 Image Analysis

No significant progress has been made on image analysis. We still plan to use the ROS rtabmap_ros package to handle depth-finding and environment mapping. The work Tao has been doing with ROS and Gazebo (a simulation environment) has given our virtual vehicle (running the actual software) the ability to ”see” its environment, but we do not have the analysis part of it integrated yet.

4.1.2 Radio Communication

Fig. 5. 3DR Telemetry Radio

The status of radio communication is much the same as image analysis. We do not have the physical platform ready for developing and testing radio communication. The radio component of this project should prove to be fairly trivial since radio control and telemetry communication are completely standardized and there are many examples and tutorials, both in hardware and software, to follow. We will be using the MAVLink protocol for radio communication.

4.1.3 User Interface

We have a command line interface (CLI) up and running on the GCS, the RPi3, and the NUC. We are using ROS for the CLI. Successfully installed ROS on both the GCS and the NUC. This gives us our command line interface (CLI) for issuing commands to the vehicle. We have been using the CLI to send and receive individual commands to ROS topics (a ROS topic is basically the way that ROS sends and receives commands). We are able to publish and subscribe to ROS nodes, specifically MAVros nodes, which is also part of being able to monitor and control the vehicle. We have also run python scripts that have published to ROS topics.

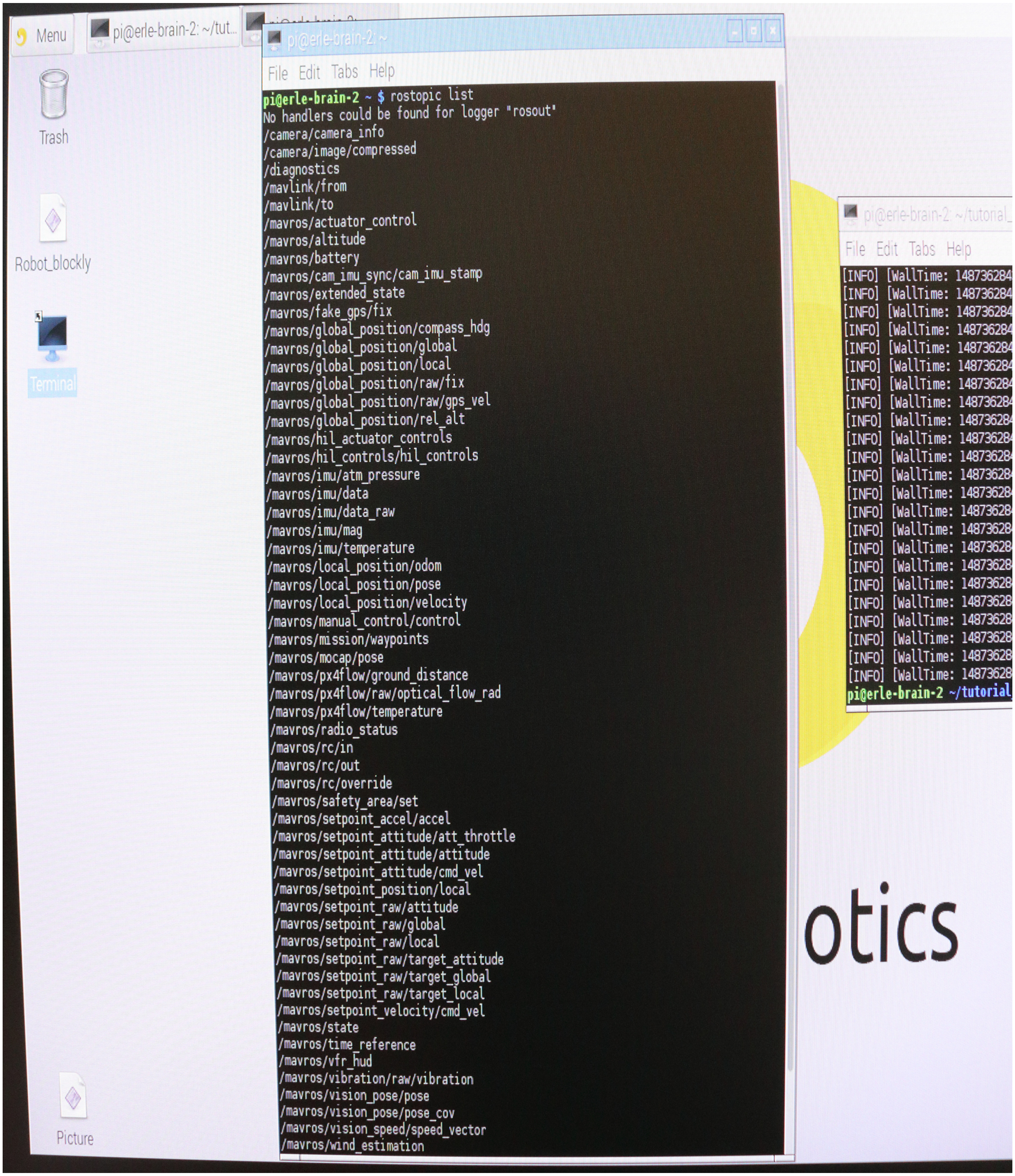

Fig. 6. Command Line Interface example of a list of running ROS topics.

4.2 Left to do:

4.2.1 Image Analysis

- Integrate data from lidar and cameras into the obstacle avoidance module.
- Perform testing and determine if rtabmap_ros will work for our needs.

4.2.2 Radio Communication

- Connect telemetry radios to the GCS and the NUC.
- Create/use ROS topics for reading the telemetry data.

4.2.3 User Interface

- Install and setup a GUI on the GCS for command and control of the vehicle. The most likely candidate at this time is ArduPilot, an open source package that allows fine vehicle control and waypoint selection.
- Some of the requirements of the GUI will be:

  hooking in to telemetry

  manual override of radio control

  manual override of autonomous operation

  ability to set waypoints
- Fill out CLI functionality. We still need to create custom convenience commands for CLI. Issuing commands to a ROS topic is pretty complicated right now.

4.3 Challenges

Problems	Solutions
Problems that impeded progress.	Specific actions to resolve problems.
Did not get RPi3 erle image until late (week 4/5).	Getting the image allowed us to attempt to set up the PXFmini on the RPi3.
The first erle image sent was corrupted, it took about a week waiting for a new image.	Erle sent a new image.
The environment from Erle seems to be very fragile/unstable. We have run into different problems each time we have re-installed the image they gave us. Our client is have yet a different problem in a separate project where they are trying to use the PXFmini.	This has not been resolved.
The PXFmini instructions/tutorials were inconsistent/incorrect. This has created quite a few problems, which has further delayed getting our autopilot running.	We have been working with our client to solve some of the issues with installing and running the PXFmini autopilot.
The RPi3 WiFii and ethernet were not configured to connect to the internet.	Researched how to set the RPi3 to connect to the internet via ethernet. It turned out that this problem was caused by how Erle set up their OS image. Found the proper settings to connect to the internet via either WiFi or ethernet. However, if WiFi is set to connect to the internet, external computers cannot talk to the PXFmini via WiFi, same for ethernet.
The PXFmini does not respond to commands via MAVros.	MAVros is the ROS api to MAVLink, the protocol for commanding the PXFmini. While we are able to publish (send commands to) and subscribe (get data from) the appropriate topics, the PXFmini does not respond as would be expected. The motors do not activate. We have found a forum posting stating that a certain variable (SYSID_MYGCS) needs to be set to 1 in order to override the RC in control to allow computer control of the PXFmini. It looks like the variable can only be set through a GUI, such as ArduPilot. We will try installing ArduPilot on the GCS and see if we can set SYSID_MYGCS to 1 and see if that makes any difference.
My personal laptop died during week 1	I received a replacement in 2 days and it took another 2-3 days to set my environment back up.
Installing Ubuntu 14.04 in dual boot did not go well. It took about a week to get installed properly along side Windows 10.	I wanted a native install of Ubuntu 14.04 (the OS we wanted to use with ROS) on my system to streamline development. Finally got the OS working properly