Marsupial Planetary Rover CS After Action Report

Outline:

I. What this is

II. Why we made this

III. ROS2 Foxy Installation
    1. Ubuntu Installation (Recommended)

IV. Raspberry Pi Setup
- Installing Ubuntu Server 20.04
- Setting up SSH
- Setting up VS Code (Optional)

V. Arduino Setup/Usage

VI. Reasons behind some of our designs, decisions & choices
- A. Using Ubuntu Server 20.04
- B. Migrating ezrassor_teleop package
- C. Not using Ros2Arduino, Rosserial, or other ROS-arduino library
- D. Usage of Hotspot on Raspberry Pi
- E. Ros2_serial_interface design
- F. Why we choose to not use PyFirmata

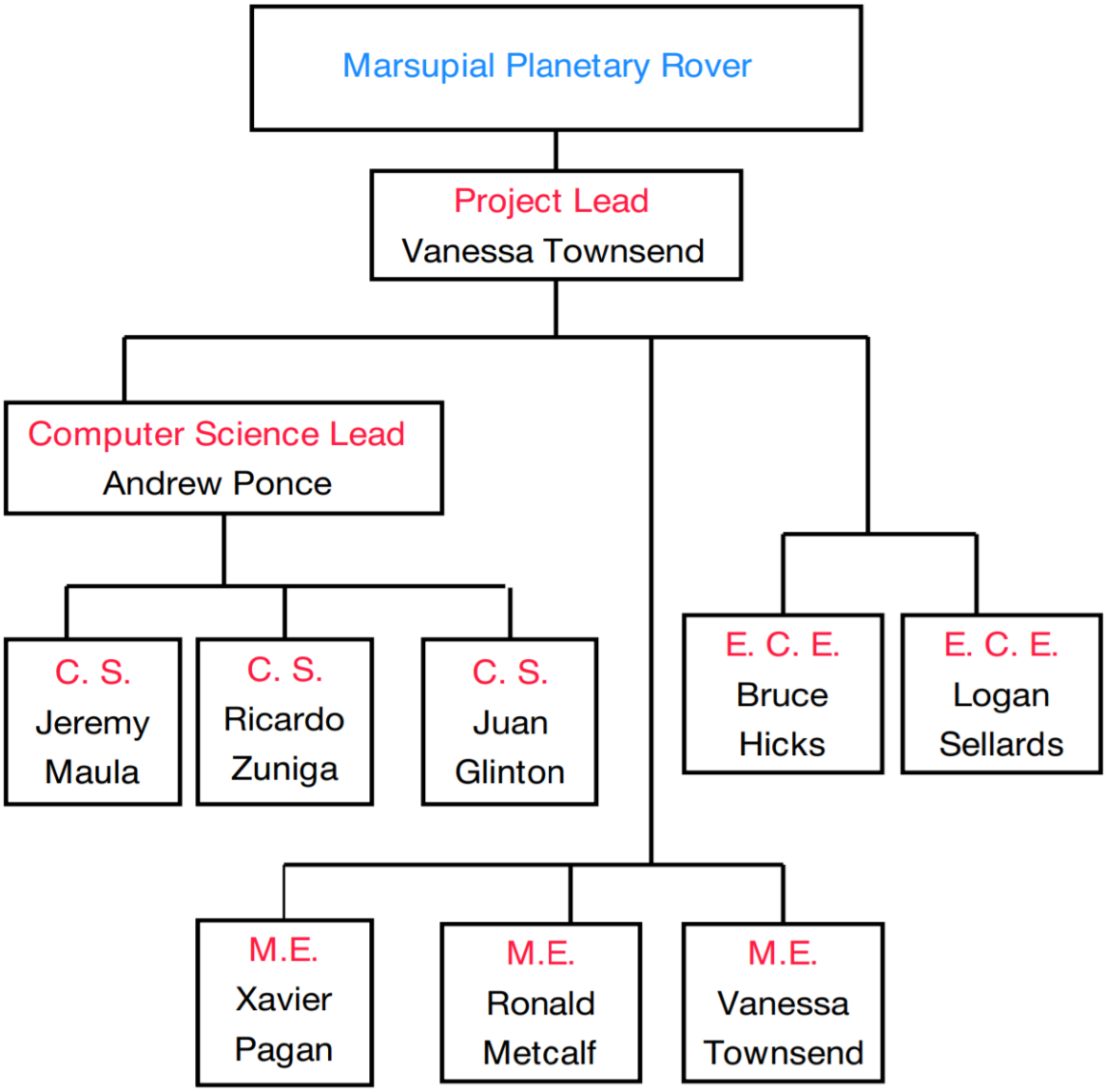
VII. Recommendations

VIII. Status up to now (Include dates)

IX. Ways to potentially improve upon our Ideas
- A. Give links to resources looked at but not used
- B. Connect teleop package with Gazebo
- C. Connect ros2_serial_interface package with ezrassor_swarm & autonomy packages
- D. Connect Ezrassor app with ros2_serial_interface

What this is:

This is an After-Action Report made by the Computer Science (CS) members of the Florida Poly Marsupial Planetary Rover Senior Design team. Our project started in Fall of 2021 and ended in Spring of 2022. Below is the team organization for the project:



The main goal of our group's project was to take the existing EZ Rassor Rover designed by NASA engineers and redesign the wheels, 3D Print, and build a marsupial planetary rover(s) that will be able to function autonomously to mine rock and ice on the moon and able to lift and dock onto a mothership rover.

Why we made this:

As the CS portion of the team, we were tasked with finding a way to make it possible to use the EZ-RASSOR software (see A1 & B4) to control the physical rover we were building. The idea was to have a Rasbperry Pi4B run the EZ-RASSOR software and an Arduino Mega 2560 control the stepper motors that would turn the wheels. The Pi was to be connected to an Arduino Mega2560 via a USB cable and would communicate with the Arduino over this, telling it when it needed to turn the motors. Additionally, it was expected that the raspberry pi would be able to communicate wirelessly with a mothership rover and be able to be controlled by this rover when needed.

At the start, we had no knowledge of ROS2 programming, Arduino programming, or Raspberry Pi programming. We didn't know exactly how EZ-RASSOR worked either. We also were not given the Raspberry Pi 4B or the Arduino Mega 2560 at the start of this project (though we were able to test with our own personal devices thankfully). Not all of us had knowledge of Ubuntu installation either. We didn't even know how the Raspberry Pi 4B would communicate with the Arduino or how it would communicate with another Raspberry Pi running EZ-RASSOR.

Despite these circumstances, we managed to figure out how to communicate between a Raspberry Pi and an Arduino and how to link that to EZ-RASSOR. The link to EZ-RASSOR came in the fact that we migrated the teleoperation package from the EZ-RASSOR software from ROS 1.0 to ROS 2.0. We then linked it to another package we built to interface with the Arduino. Finally we ran them together and demonstrated the ability for the rover to move. We also set up the Raspberry Pi 4B to act as a hotspot that could be connected to and communicated, allowing for two devices running ROS2 packages to communicate over the same network.

Because this was so difficult to figure out, we elected to leave behind this document explaining how we did everything. This document is everything

we wish we had at the start of this project. We hope that this at least makes it easier for future teams that try to follow in our footsteps.

Install ROS2 “Foxy”: (See D1 and related)

In our experience, we found that it was easiest to use ROS2 “Foxy” on Ubuntu 20.04. We mostly ran and developed our programs utilizing the ROS2- bare-bones installation, because we discovered that in fact it was possible to test most of the ROS2 EZ-RASSOR packages without the need for any of the GUI-based tools ROS2 provides. The only exceptions to this were the packages that needed the Gazebo simulation software, which were packages that we did not need to utilize in the course of our project.

Overall if you want to test the things we made (B1, B2, and B3), you don’t need to have a high-performance computer, nor should you have to do a bare-metal install of Ubuntu 20.04. The least you need is the ability to use Windows Subsystem for Linux (WSL) or a Virtual Machine with at least Ubuntu 20.04. In fact, you could run everything on Ubuntu 20.04 Server, which is simply just a shell terminal and does not come with a desktop environment (you can install one though if you ever need to). This is what we used on the Raspberry Pi 4B (we will explain that later on)

However, you should at least be able to run multiple terminals in Ubuntu in order to test ROS2 packages together. You can do this with WSL by opening multiple WSL instances, or by opening multiple terminal windows in Windows itself and running WSL on each of them.

If you’re not needing graphical tools, then WSL is by far one of the easiest ways to use ROS2 without a GUI if you have Windows 10 or 11. Using a virtual machine with Ubuntu 20.04 is also an option as well. If you have good experience with Docker, you can also look into using it. We did not use Docker, so unfortunately we can’t give advice on that :(

Raspberry Pi Setup (See C2 and E2):

If you actually need to use a Raspberry Pi for a ROS2 project, we recommend you install the 64-bit Ubuntu Server on it. We installed ROS2 “Foxy” on Ubuntu 20.04 Server on our Pi, as we felt that it was unnecessary for our Raspberry Pi to run an entire graphical desktop environment when we only needed it to run a few shell commands. This will likely decrease the workload the Raspberry Pi will be under, which is important if you want to run a lot of ROS2 packages simultaneously.

As mentioned previously, we had our raspberry pi use a Hotspot. We had another raspberry pi connect to this hotspot so that the ROS2 program we had running on each could communicate with the program running on the other Pi. If you want an idea of how that works, check out C5 and C6 of the resources, which will explain how that concept works. E1 and E2 will explain how you can set up a Hotspot in Ubuntu 20.04 and have other devices connect to it.

Arduino Setup:

You are going to want to install the Arduino IDE if you are going to use an Arduino. You can use either windows or Ubuntu for it (See C12 for Ubuntu)

As a tip, know that the way device drivers work on Linux is different than windows or Mac OS. Specifically Linux shows devices as filesystem entries, usually under a folder called “dev”. If you plug an Arduino into a computer running Linux, it is recognized by the name “/dev/ttyACM0”. In Windows however it might be recognized as “COM4” instead. So do keep this in mind when you are testing the ROS2 programs we built

Check out C1 to understand how serial communication works between a Raspberry Pi and an Arduino. This is the method of communication we used to control the motors on our rover.

Why we made certain decisions:

Why didn't you use PyFirmata for the interface?

PyFirmata would indeed have allowed us to use Python on the Pi to directly control the Arduino microcontroller. However, we wanted to have the Arduino and Raspberry Pi as loosely coupled as possible so that we could separate the stepper motor programming from the ROS2 & Python programming.

Why didn't you use a ROS2-Arduino library like Ros2Arduino or Micro-ROS?

The main reason why we didn't was because they were not compatible with our Arduino microcontroller (ATMega2560). Another important reason was because we wanted to make it so you didn't need to run ROS2 code on the Arduino. The way ROS2 Arduino libraries work is that they require you to run ROS2 C++ code on the Arduino itself. This would have made it more difficult for others to program the Arduino, so we elected to simplify this by building our own interface and making it so you didn't need to know ROS2 in order to program your Arduino. This actually helped a lot when it came to collaborating with the Non-CS members of our group, as we didn't need to explain how ROS2 really worked in order to explain how the Arduino would move the motors.

What about the EZRASSOR_Arduino repository (see B5)? Couldn't you have used that for the Arduino interface?

Unfortunately no. While the sketch and files in the EZ-RASSOR_Arduino repository technically would be compatible with the Arduino we were using, the If you actually

Why did you migrate the ezrassor teleop package to ROS2?

This is a fair question, as it was something we were never told we expressly had to do. But the reason we did this is because we needed a way to use ezrassor to control a physical rover. The teleop package was not migrated to ROS2 when we started, nor was there a plan to migrate it to ROS2. However we realized its potential use in our project and attempted to migrate it in hopes of maybe obtaining some of its functionality. This turned out to be beneficial, since it allowed us to demonstrate communication between different rovers running the software.

Couldn't the project still have worked if you used regular Ubuntu 20.04 instead of Ubuntu 20.04 Server?

Very likely we could have just used the regular version of Ubuntu 20.04 (the one with a desktop environment) on our Raspberry Pi and still have been able to run everything. However, as we mentioned before, we used Ubuntu 20.04 Server because it was lightweight and because the Pi didn't need to have a desktop environment on it. We wanted to give the Pi as much memory and processing power as we reasonably could, so this solution was more practical for us, especially since we knew that a decent amount of memory and processing power would be required to run all the non-simulation packages in the EZ-RASSOR software suite.

Our Recommendations:

1. Start learning ROS2 and Python (if you don't already know it) early on! We advise you start by learning the basic concepts behind ROS2, and then move on to creating your own ROS2 packages. A lot of things can be learned on the fly during your project, so the more practice the better!
2. Explore all resources available to you early on! Read the documentation available for EZ-RASSOR and make sure you understand how the software suite generally works. Get access to your microcontrollers early on so you can learn how to use them. Explore and experiment with the various GitHub repositories made by projects previous to yours. If you don't understand something don't

delay on reaching out to someone who could possibly help you. Research is important! And remember–the earlier the better!

3. Force yourself to utilize Git and GitHub to track everything you do. GitHub is a site you will use often when it comes to managing software with Git, so make sure you also become comfortable setting up repositories and managing them on GitHub.
4. Utilize the concept of “Pair-Programming” effectively. We understand how demanding a project can be, and collaboration will make a huge difference in how far you get. Try to hold sessions where you and your team members program the software you will be using. This can be in person or if you want you and your teammates can stream your screens over Discord or another communication platform and talk to each other while you develop the software.

Status of Software at the Time of Writing This (April 2022):

There were some things that we originally planned on doing in our project that ended up not being accomplished. We ideally wanted to run the ROS2 version of th ezrassor_swarm and ezrassor_autonomy packages, but given these packages were actively being migrated and upgraded by the UCF EZ-RASSOR 2.0 team, we began to realize we were not going to have enough time to test and integrate them with our rover and the software we built. This is a task we hope a future group will be able to focus on now that we have built the underlying hardware interface.

It should be mentioned that the original EZ-RASSOR (ROS 1.0 “Melodic” version) had a design flaw that required certain packages to rely on the use of the Gazebo Simulation software in order to function correctly. This issue we were told was related to the data regarding the current state of the rover, and that it was hoped that the UCF EZ-RASSOR 2.0 team would fix this issue.

What we discovered was that this issue actually stemmed from the fact that the rover’s position and orientation were derived from the Gazebo simulation. This dependency can be seen in the ezrassor_swarm and

ezrassor_autonomy as the software relies on something called “LinkStates”, which is a ROS message that provides geometric orientation and position (see gazebo_msgs/LinkStates Documentation)

We were fortunate in that we could remove the use of Gazebo in the ezrassor_teleop package when we moved that over to ROS2. We however did not work on a way to re-add the ability for the user to choose between using and not using Gazebo to simulate the ezrassor_teleop packages. As such, you will find there might be unused variables in the program, and you should see some gazebo-related functions that are commented out in the teleop_action_server, coupled with various comments stating our plan to re-introduce these functions later. We hope a future team carries on our work and elects to reimplement this functionality in the software, seeing as it was something that was present in the ROS1 version of ezrassor_teleop.

Ways to Improve or Expand upon our work:

Integrate the ros2_serial_interface with ezrassor_swarm: A major hurdle that we could not feasibly address until after we built the serial interface was how we could set up and configure multiple raspberry pis, each running ROS2 ez-rassor, to work in a swarm communication network. Communicating with one Raspberry Pi running the serial controller program and another Pi running a program that communicates with it is not a problem–they just have to be on the same Wifi or LAN network to do so. However if you want to run the serial controller program on multiple raspberry pi’s on the same network, you have a way for the software to be able to distinguish between each pi. The way you would have to do this is by making the serial controller program that runs on each Pi subscribe to differently named ROS2 topics. Say you wanted two Pis to run the software. In that case you might have the serial controller program on one Pi be subscribed to “wheel_instructions1” and have the one running on the other Pi be subscribed to “wheel_instructions2”.

Design and configure Raspberry Pi Wifi Swarm-Communication Network: The simulation software can showcase how individual rovers would work when communicating with each other. However if you want to use multiple Raspberry Pi's (or whatever you're using to run EZ-RASSOR on the rover) to communicate with one another wirelessly, they have to be on the same Wifi network.

However, this quirk is something that can generate some interesting scenarios. Let's say there's a Raspberry Pi on one main rover that is running the software and generating a Wifi network that the pi's on the other rovers are connected to. How do you account for when the Pi on one rover disconnects from the network–either because it rebooted or fell out of range? Can it reconnect to the same Wifi network?

This problem has a wide variety of solutions and can get even more complicated. You could attach a wifi-dongle to a raspberry pi which might allow the pi to connect to a wifi network while the Pi itself generates a hotspot for other pi's to connect to. The question would be how you would distinguish both the rovers and the different wifi networks

As you can see, there isn't a simple solution here. You would need to do a lot of research like we did to figure out an optimal way to implement this, as well as configure the hardware to be able to do it. We unfortunately didn't have time to even work on or test a solution to this problem. However, we did make it possible for future Senior Design groups to do so, as the serial controller software can allow you to do this.

Closing Thoughts:

It was difficult to accomplish what we did knowing very little at the beginning. That being said, we're not perfect and our work could very much be improved upon or added to. We fully expect that the software we built will be worked on and further improved by future groups. As such, we aim to at least be around or available for future teams to reach out to, because we want to ensure those who come after us can continue our work.

In that same spirit, we ask that you pay it forward to future teams by leaving behind something like our document, as well as being contactable by future teams too. Doing so helps everyone, and allows teams to carry on your work and make it even easier for them to do so.

Thanks for reading our After Action Report! We wish you luck on whatever it is you or your group might be working on!

Team Contact Info:

Andrew Ponce (Andy):
Email: aponce7336@floridapoly.edu or andy.ponce@outlook.com
GitHub Username: AnrdyShmrdy
Discord: Anrdy#6221

Jeremy Maula:
Email: jmaula6418@floridapoly.edu
GitHub Username: jmaula99
Discord: Ymerej#5485

Juan Glinton:
Email: jglinton7589@floridapoly.edu
GitHub Username: tinysoccerball
Discord: TinySoccerBall#2184

Ricardo Zuniga:
Email: rzuniga8612@floridapoly.edu
GitHub: CowardlyPLeb
Discord: CowardlyPleb#1009

Resources Index (At the Time of Writing this)

A. Florida Space Institute
   A1. https://github.com/FlaSpaceInst/EZ-RASSOR/wiki
   A2. FSI Robotics Initiative – Florida Space Institute

B. Important Github Repositories
   B1. https://github.com/AnrdyShmrdy/ros2_arrow-key_example
   B2. https://github.com/AnrdyShmrdy/ros2_serial_interface
   B3. https://github.com/AnrdyShmrdy/ezrassor_teleop
   B4. https://github.com/FlaSpaceInst/EZ-RASSOR
   B5. ffermo/EZ-RASSOR_Arduino (UCF 2020 Black & Gold team's Arduino files)
   B6. https://github.com/AnrdyShmrdy/MarsupialCSTeamResources
   B7. https://github.com/AnrdyShmrdy/pyserial_arduino_controller

C. Raspberry Pi & Arduino Resources
   C1. Raspberry Pi Arduino Serial Communication - Everything You Need To Know - The Robotics Back-End
   C2. How to install Ubuntu Server on your Raspberry Pi | Ubuntu
   C3. How to Connect Raspberry Pi over Serial
   C4. Example of Raspberry Pi Serial communication with Arduino
   C5. Running ROS2 on Multiple Machines, Including Raspberry Pi
   C6. Robotics Back-End Raspberry Pi Tutorials
   C7.GPIO Programming on Raspberry Pi
   C8.Coding on Raspberry Pi remotely with Visual Studio Code - Raspberry Pi
   C9. Remote Development on Raspberry and Arduino with VS Code - Joachim Weise
   C10. How To Program Arduino From The Raspberry Pi Command Line – Siytek
   C11. All Libraries - Arduino Libraries
   C12. Install the Arduino IDE | Ubuntu
   C13: control-arduino-with-python-and-pyfirmata-from-raspberry-pi

D. Ros 2 Resources
   D1. ROS2 Foxy Ubuntu Install Guide
   D2. ROS2 Foxy How-To Guides
   D3. ROS2 Foxy Tutorials
   D4. rclpy api documentation
   D5. Actions — rclpy 0.6.1 documentation
   D6. Topics vs Services vs Actions — ROS 2 Documentation: Foxy documentation
   D7. Creating an action — ROS 2 Documentation: Foxy documentation
   D8. ROS2 Foxy Creating Your First ROS2 Package

D9. ROS2 Foxy Using Parameters In A Class Python
D10. ROS2 Foxy Writing A Simple Python Publisher And Subscriber
D11. Writing a simple service and client (Python) — ROS 2 Documentation: Foxy documentation
D12. Creating a workspace — ROS 2 Documentation: Foxy documentation
D13. Understanding ROS 2 actions — ROS 2 Documentation: Foxy documentation
D14. Writing an action server and client (Python) — ROS 2 Documentation: Foxy documentation
D15. About Ros Interfaces
D16. Creating custom ROS 2 msg and srv files — ROS 2 Documentation: Foxy documentation
D17. Creating a launch file — ROS 2 Documentation: Foxy documentation
D18. Automatic Addison's ROS2 Actions Example
D19. ROS2 Foxy Action Client+Server examples
D20. examples/rclpy/actions at master · ros2/examples
D21. ROS2 Foxy program/concept examples
D22. Package that Inspired ROS2 Serial Interface

E. Ubuntu/Linux
E1. Ubuntu Network Manager Documentation
E2. Configure WiFi Access Points with Network Manager's CLI
E3. Add a User to a Group (or Second Group) on Linux
E4. Setting up a Static Ip Address Ubuntu 20.04 Raspberry Pi
E5. How to overwrite the previous print to stdout in python? - Stack Overflow
E6. bash - Hide current working directory in terminal - Ask Ubuntu

F. Our ECE Parts
F1. Nema 17
F2. Stepper Motor Driver
F3. Raspberry Pi 4B
F4. ATmega2560 (Arduino Mega 2560)

G. Other
G1. Handling the keyboard — pynput 1.7.6 documentation
G2. pynput - pynput.pdf

Note: See the html file containing the above and any other resources not mentioned in our GithHub repo: https://github.com/AnrdyShmrdy/MarsupialCSTeamResources