Senior Project Report

Alexa Hansen
Derek J. Russell
Storm Randolph

Project Advisor: Andrew Danowitz - Cal Poly Department of Computer Engineering
Client: Rich Murray - Cal Poly Department of Computer Engineering
06/15/2023

Introduction **2**
Project Overview 2

RoverX Manual **4**
Initial Setup 4
Battery Setup 5
Running Commands 6
Solar Track Algorithms 7
Power Budget System 7

Conclusion & Future Recommendations **12**

Appendices **13**
A. Config.txt file for raspberry pi 4 model B 13
B. High Level Commands for commandLayer.cpp 15
C. Use/Flow Diagram for commandLayer.cpp 17

References **18**

Introduction

Project Overview

The RoverX Senior Project has been an evolutionary process, with its roots in the RoverX Capstone project which concluded in the Winter of 2023. Our journey commenced with a core focus on hardware, which culminated in the successful creation of the Minimum Viable Product (MVP). Further information on our initial endeavors can be found in the 2022 "Alpha Report" and the 2023 "Summary of Handoff Contents/Quick Start Guide" for the RoverX project. However, recognizing the limitless possibilities of software in enhancing the rover's capabilities, our focus shifted towards this exciting frontier. Our final quarter of the senior project was dedicated to refining the source code and generating comprehensive documentation, which will serve as an invaluable guide for future rover teams.

In consultation with Prof. Murray, we embarked on the creation of a command layer. The intention behind this was to provide a user-friendly interface where high-level commands could be input and subsequently executed by the rover. The unique structure of this layer was conceived to include an outer layer written in C++, which would then interact with Python-based movement functions controlling the rover. This interaction is facilitated by an innovative "interpreter", designed to bridge the C++ and Python realms by translating C++ commands into executable Python scripts. This strategic choice was fueled by the impending transition of the rover's source code from Python to C++ or another object-oriented language by the subsequent rover team. Our aim was to provide a flexible outer layer that could be seamlessly integrated with the new C++ rover source code. In essence, we envisaged creating a dynamic platform that could adapt to future advancements, fostering an environment of continuous development and innovation for the RoverX project.

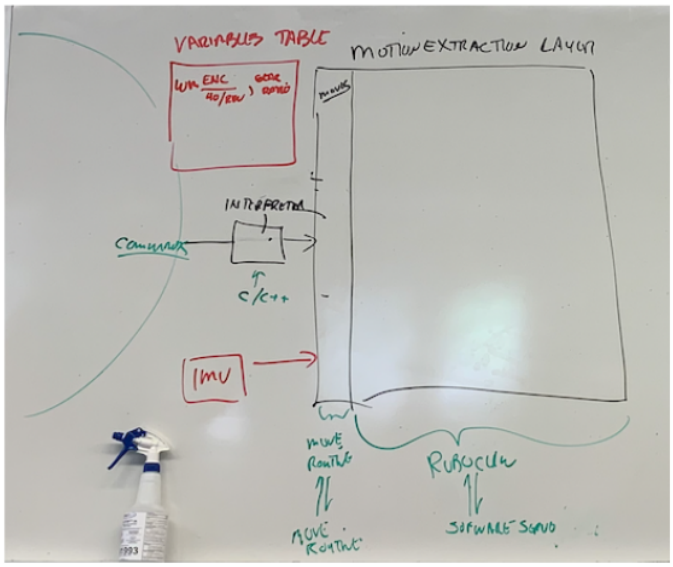


Figure 1: Initial concept design of RoverX Spring Senior Project command layer.

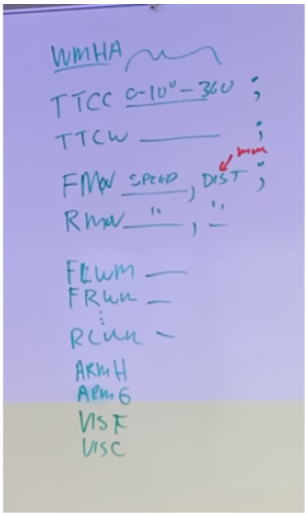


Figure 2: Initial examples of a High Level Command and their parameters

We first created a commandLayer.cpp that is a wrapper for the entire rover project that is written in C++. In this layer the user will input the command they wish to issue, and the two parameters that correspond with that command. These commands are specified in Appendix.B of this paper. This layer is in charge of accepting the user input and using that for a system function call of a Python script that will run the command. We focused our attention on setting the code up to be more command based than before, with each python file categorized by a type of movement command. Since no previous groups provided clear documentation for rover projects, we decided to also create the following manual for teams in the future.

RoverX Manual

Initial Setup

Step 1: Powering Up Rover

The rover can be powered by a hobby battery, the Stellar Power Pack, or by a power supply. Initially using the power supply to power the rover is recommended. Before turning on the rover ensure that the voltage, current, over voltage, and over current settings on the power supply match the image in figure 3.

Figure 3: **Specifically the voltage: 15 V, voltage limit: 15.1V, current: 1.8A, and Current limit 1.901A.**

Step 2: Calibrate RoboClaws 2x7A

If any RoboClaw settings have been changed each RoboClaw will need to be recalibrated. Use the BasicMicro Motion Studio App for the RoboClaw 2x7 Motor Controller found on the Basic Micro downloads page, https://www.basicmicro.com/downloads. Once the App is installed follow the instructions found in the "Calibration Document for Herbie Roboclaws" by Ryan Ozawa. These instructions are extremely important..

Step 3: SSH into the rover.

We set up the rover so that it is easier to connect to. After turning on the rover connect your wifi to roverX_1, then ssh pi@roverx.local. The password is "parched-mannish-regulate" and will need to be entered to finish the process.

Step 4: Check the config.txt file in

It is extremely important that the config.txt file in the root directory of the raspberry pi 4 matches the source code provided in Appendix A with the following commands.

- **sudo su**
- **cd ..**
- **cd ..**
- **cd boot**
- **cat config.txt**

Step 5: Installing I2C libraries

Once the rover is powered up navigate to the libi2c-master folder, enter the following command:

- **cd roverSP**
- **cd libi2c-master**
- **make**

This will make the I2C library that will be used for the project.

Battery Setup

Step 1: Charge Stellar Power Pack (SPP)

The SPP can be charged by supplying 40V with a current limit of 1A to the SOL pin (and GND to GND). The overall pack voltage can be measured with a voltmeter on the Vmux pin. Individual cell voltages can be found by measuring adjacent pins on the left side of the pack. Vmux is <u>not</u> for charging, only to check pack voltage.

Step 2: I2C Bus

If the pack shows a good voltage (25-31V) but won't communicate via I2C, check the pack is addressed with I2C on bus 4 using `i2cdetect -y 4` (this is also useful for troubleshooting the IMU/Jetson Nano I2C devices to determine if they are connected addresses). Address 0x28 should be up, indicating the pack is listening on I2C. The `test_valid_write_DSC` test case in the `test_battery.py` program on the pi to change the Short-Circuit Discharge Current Limit Timeout value to a longer period of time. The inrush current to the DC-DC converter causes the battery pack to trigger a DSC condition (Discharge Short Circuit), but by extending this time, it gives the

DC-DC converter more time to charge at a high current before the BMS thinks it is a short. The battery_test program does other things, but the test_valid_write_DSC test case provides a general framework to follow to permanently changing the configuration in EEPROM (see the ISL94203/94202 reference docs for more info on addressing scheme and the other parameters you can change). The DSC config is currently only changed in RAM (volatile) so if the pack powers down, the configuration value will be lost.

Running Commands

Step 1: Navigate to the Source Code

When the rover is powered on it will start at the first rover home screen. There are numerous directories to choose from, the one for the most current version of this project is **roverSP** and that is located in the rover directory. There is another directory rover that is located in rover/src/rover that contains the original source code that was written by the Herbie, Vega, and RoverX teams. To navigate to the executable source code is roverSP enter the following commands:

- **cd rover**
- **cd roverSP**
- **cd src**

Now all of the source code can be accessed and high level commands can be run in the terminal.

Step 2: Running High Level Commands

To begin running high level commands, compile commandLayer.cpp and then execute the program.

- **g++ -o commandLayer.cpp**

This will give you an executable that should then be run in the terminal. The user will be prompted to, “Enter RoverX High Level Command”, enter one of the commands in Appendix B. Every high level command requires two parameters, it will either be two int values or one string and one int value. When a command is not going to be using the parameters the user will just enter 0. There are no input checks in the program at the moment. It has to be inputted exactly as the manual suggests.

IMPORTANT: The first command you run upon powering up the rover is CARW (Calibrate All Rover Wheels), or else no other commands will be able to be run.

The program, commandLayer.cpp, will keep running until the command EXIT is entered.

Solar Tracking Algorithm

The goal of the solar tracking algorithm is to have the rover collect as much sunlight as possible throughout the day. The current solar tracking algorithm on the rover tracks the position of the sun during a six hour time period. The rover initially starts in its "sunrise" position for the first hour of this period. Then after this first hour, and at every additional hour after, the algorithm will first output the current position of the sun and then call the rover's tank turn method for 15 degrees left. Once this six hour period is over, the rover will tank turn back to its original "sunrise" position. The algorithm takes two different arguments: the latitude and longitude of your current location. As a default, the current latitude and longitude are set to Cal Poly's location coordinates. The following steps detail how to properly access and run the algorithm.

Step 1: Run the Solar Tracking Algorithm

To run the solar tracking algorithm, run the following command:

- **python3 solar_tracking.py**

The solar tracking algorithm takes two arguments, the latitude and longitude of the current location. These parameters can be adjusted as needed. Once the algorithm is started, it will continue until the six hour cycle has completed or it is manually stopped by user interrupts.

Power Budget System

A. Introduction

The uniqueness of the PowerMonitor script is best encapsulated in its adaptability. Designed to be elastic, it can be modulated to accommodate a spectrum of scenarios and power conditions. From vast industrial complexes with elaborate energy requirements, to small households on a quest to optimize their energy utilization, the PowerMonitor script

can be attuned to align with the context. This adaptability distinguishes it as a universally applicable instrument in the arena of modern energy management.

The significance of the PowerMonitor script, however, transcends its technical prowess. At its heart, it acts as a lighthouse guiding us towards a sustainable future. It represents a pivotal transition from reactive to proactive energy management, thrusting us towards a future where energy use is not merely monitored but optimized for efficiency and sustainability. As we continue on this path of refinement and innovation, creating tools akin to the PowerMonitor script, we are not only reshaping our energy landscapes, but fundamentally altering the destiny of our planet. Our endeavors now are investments for the future - each stride in this direction brings us closer to an existence where sustainable and intelligent energy consumption is not the exception but the norm.

B. Setting Up Jupyter Notebook for PowerMonitor Project

Step 1: Installing Python

- Before installing Jupyter Notebook and the associated Python libraries, ensure that Python is installed. Python 3 is recommended. https://www.python.org/downloads/

Step 2: Installing Jupyter Notebook with Anaconda

Anaconda is a Python distribution platform that significantly simplifies the process of installing and managing Python and its libraries. It's particularly advantageous for data science projects. Here's why:

1. **Ease of Installation:** Anaconda comes bundled with Python, Jupyter Notebook, and many other useful data science tools, eliminating the need to manage dependencies and packages separately.
2. **Package and Environment Management:** Anaconda allows you to create isolated environments that are very useful when working on projects with different package and library requirements.
3. **Comprehensive Data Science Toolkit:** Anaconda includes many pre-installed, popular data science libraries, making it a comprehensive tool for data analysis and scientific computing.
4. **Compatibility and Conflict Prevention:** Anaconda ensures compatibility between the different packages and libraries it includes. It also helps prevent conflicts between Python packages that might arise when they're installed globally.
5. **Reproducibility:** Anaconda's environment management ensures reproducibility of your data science projects. You can share your

environment configuration with others so they can recreate the same setup.

6. **Platform Agnostic:** Anaconda works on all major operating systems.

Step 3: Installing Anaconda and Jupyter Notebook

1. **Download Anaconda:** Visit the Anaconda distribution page at https://www.anaconda.com/distribution/#macos. Select the Python 3 version and click on the download button.
2. **Install Anaconda:** Open the downloaded file and follow the instructions to install Anaconda. This will install Anaconda, Python, Jupyter Notebook, and a range of other tools and libraries useful for data science.

Step 4: Setting Up a New Conda Environment

- It's generally a good idea to set up a separate Conda environment for each project. This helps isolate your project and its dependencies from other projects.
- To create a new environment named "PowerMonitor", open Terminal and type: **conda create --name PowerMonitor**
- To activate this new environment, type: **conda activate PowerMonitor**

Step 5: Installing Required Python Libraries

Install the Python libraries required for your PowerMonitor project. Since you've installed Anaconda, it's recommended to use conda for package installation because it can handle dependencies more effectively:

- **conda install numpy scikit-learn matplotlib seaborn**

If a package is not available through conda, then install it using pip, Python's package installer.

Step 6: Opening up Jupyter Notebook from the Terminal

After activating the Conda environment and installing all the required packages, you can start Jupyter Notebook by typing jupyter notebook in Terminal. This will start the Jupyter Notebook server and open a new tab in your default web browser.

Step 7: Creating a New Notebook

On the Jupyter Notebook dashboard (which displays in your browser after launching Jupyter Notebook), click on the "New" dropdown button in the top right corner, then select "Python 3" under the Notebook section. A new tab will open with a new notebook.

Step 8: Importing Libraries

In the first cell of your new Jupyter notebook, import all the libraries you installed. To execute the code in the cell, press Shift + Enter.

- **import numpy as np**
- **from sklearn.model_selection import train_test_split**
- **from sklearn.tree import DecisionTreeRegressor**
- **from sklearn.metrics import mean_squared_error**
- **import matplotlib.pyplot as plt**
- **import seaborn as sns**

C. Essential Python Libraries and Frameworks

a. The script employs Python, a multifaceted programming language lauded for its applicability in scientific computation, machine learning, and data analysis. Several integral Python libraries are put into service, including:

i. **NumPy:** Facilitates array manipulation and mathematical computations.

ii. **Sci-kit Learn:** Enables data splitting, predictive modeling through linear regression, and computing mean squared error.

iii. **Matplotlib and Seaborn:** Used for data visualization through a diverse array of plots.

iv. **Datetime:** Utilized to ascertain the current day and date.

v. **Random and Logging:** Generate random values and record alerts, respectively.

D. System Architecture

The PowerMonitor script is structured around two classes, MockMLModel and PowerMonitor, which simulate power usage across a 24-hour period.

1. **MockMLModel:** Predicts power consumption variables such as the time of day, the day of the week, and specific holidays. It is designed to

replicate the behavior of a more sophisticated machine learning model.

2. **PowerMonitor:** Simulates the functioning of a power storage system, modulating power collection and consumption over time. It monitors system state, logs alerts, and visualizes power data.

Apart from these classes, a main function manages user inputs, model training, power data calculations, and the display of results.

E. Model Training and Prediction

The script adopts decision tree regression for power consumption prediction. It generates a dataset simulating power consumption over 30 days, taking into consideration peak and off-peak hours. This dataset serves as the basis for training the model, which is subsequently deployed for making predictions in the PowerMonitor class.

F. Energy Management Mechanisms

The PowerMonitor class administers an energy management system that adjusts power consumption contingent on the state of power storage and the time of day. "Missions", signifying activities that require additional power, can be added, thereby influencing power consumption patterns. When stored power falls below 20% of the initial power, the script engages an energy-saving mode, diminishing power consumption by half.

G. Data Visualization

The script generates visualizations for three dimensions of power: stored power, power consumed, and power collected, leveraging the capabilities of Seaborn and Matplotlib libraries. These visualizations facilitate the analysis of fluctuations in power consumption and collection throughout a day.

H. Logging and Alert System

The script incorporates an alert system that logs vital information pertaining to power consumption and collection. It issues alerts when power is decreasing rapidly, when power depletion is imminent, when a power outage is expected, and even when there is surplus power or idle time.

Conclusion & Future Recommendations

This project has provided us with such an amazing real world experience to implement all of our knowledge garnered at Cal Poly. We would first like to thank Rich Murray and Andrew Danowitz for providing so much guidance and experience, this project would never have gotten as far as it has without these two. We would also like to give a special thanks to our correspondence from last year's Vega rover team, thank you for accepting our late night phone calls and debugging sessions.

Given the time provided we were unable to get each of the movement commands working correctly. Currently all of our corner motors only turn left, despite being given a right turn command. We attempted to fix this within the code but even trying to brute force the turn in the correct direction did not work. I recommend the next team look into reading the current encoder values of the M2 corner motors to see what they are. It is either not reading the value correctly or not being set correctly. As a team we were unable to come up with a solution and even after reaching out to previous teams, we were unable to find a fix for this issue. The issue with the corner motors will affect every command that involves a corner motor, such as tank turns.

We would also recommend creating safe exits in python for individualMovement.py, individualCorner.py, tankTurn.py, and multipleMovement.py. These exits should not stop the entire program, but simply reset the while loop in commandLayer.cpp to ask for the next command. There are also no input validations in commandLayer.cpp, and those will absolutely need to be added for each specific command.

Another recommendation would be to modify the solar tracking algorithm so that it will be able to properly call the rover's tank turn function. As of now, the current solar tracking algorithm calls a manual tank turn function that is located in rover.py. This function requires you to manually turn the wheels into a tank turn position before running the function for it to run properly.

Appendix

A. Config.txt file for raspberry pi 4 model B

```
# For more options and information see
# http://rpf.io/configtxt
# Some settings may impact device functionality. See link above for details

# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

# uncomment this if your display has a black border of unused pixels visible
# and your display can output without overscan
disable_overscan=1

# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's size minus
# overscan.
#framebuffer_width=1280
#framebuffer_height=720

# uncomment if hdmi display is not detected and composite is being output
#hdmi_force_hotplug=1

# uncomment to force a specific HDMI mode (this will force VGA)
#hdmi_group=1
#hdmi_mode=1

# uncomment to force a HDMI mode rather than DVI. This can make audio work in
# DMT (computer monitor) modes
#hdmi_drive=2

# uncomment to increase signal to HDMI, if you have interference, blanking, or
# no display
#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
```

```
#arm_freq=800

# Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2s=on
dtparam=spi=off

# Uncomment this to enable infrared communication.
#dtoverlay=gpio-ir,gpio_pin=17
#dtoverlay=gpio-ir-tx,gpio_pin=18

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on

[pi4]
# Enable DRM VC4 V3D driver on top of the dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2

[all]
#dtoverlay=vc4-fkms-v3d

# NOOBS Auto-generated Settings:
enable_uart=1
#dtoverlay=w1-gpio
start_x=0
gpu_mem=128

# This enables uart on pins:
# 8, 10
# 27, 28
# 7, 29
# 24, 21
# 32, 33
# zero doesn't need an overlay
dtoverlay=uart2
dtoverlay=uart3
dtoverlay=uart4
dtoverlay=uart5

dtparam=i2c_arm=on
# This enables i2c on pins:
# bus1: 3, 5
# bus4: 31, 26
# bus6: 15, 16
dtoverlay=i2c1,pins_2_3,baudrate=400000
```

```
dtoverlay=i2c4,pins_6_7,baudrate=400000
dtoverlay=i2c6,pins_22_23,baudrate=400000
#hdmi_enable_4kp60=1
```

B. High Level Commands for commandLayer.cpp

Tank Turn Commands

1: TTCW(int direction, int degree): tank turn clockwise

Example of usage: TTCW right 180

2: TTCC(int direction, int degree): tank turn counter clockwise

Example of usage: TTCC leftt 360

Calibration Commands

3: CARW(): Calibrate All Rover Wheels

Example of usage: CARW 0 0

HAS TO BE THE FIRST COMMAND RAN UPON POWERING UP!

***the two parameters must be zero (they aren't being used)

Multi-Wheel Movement Commands

4: AWMF(int distance, int speed): move all wheel motors forward

Example of usage: AWMF 300 300

5:AWMB(int distance, int speed): move all wheel motors backward

Example of usage: AWMB 300 300

6: TAFC(string direction, int degree): turn all front corner motors

Example of usage: TAFC left 30

7: TABC(string direction, int degree): turn all back corner motors

Example of usage: TABC right 15

24:TACM (string direction, int degree): turn all corner motors a specified direction

Example of usage: TACM left 20

25: RCAW(): Recenter All Wheels

Example of usage: RCAW 0 0

***the two parameters must be zero (they aren't being used)

26: EXIT(): Exit program

Example of usage: EXIT 0 0

***the two parameters must be zero (they aren't being used)

Individual Movement Commands

8: FLWF(int distance, int speed): front left wheel forward

Example of usage: FLWF 300 300

9: FLWB(int distance, int speed): front left wheel backward

Example of usage: FLWB 300 300

10: MFLC(string direction, int degree): move front left corner
Example of usage: MFLC left 15

11: FRWF(int distance, int speed): front right wheel forward
Example of usage: FRWF 300 300
12: FRWB(int distance, int speed): front right wheel backward
Example of usage: FRWB 300 300
13: MFRC(string direction, int degree): move front right corner
Example of usage: MFRC right 15

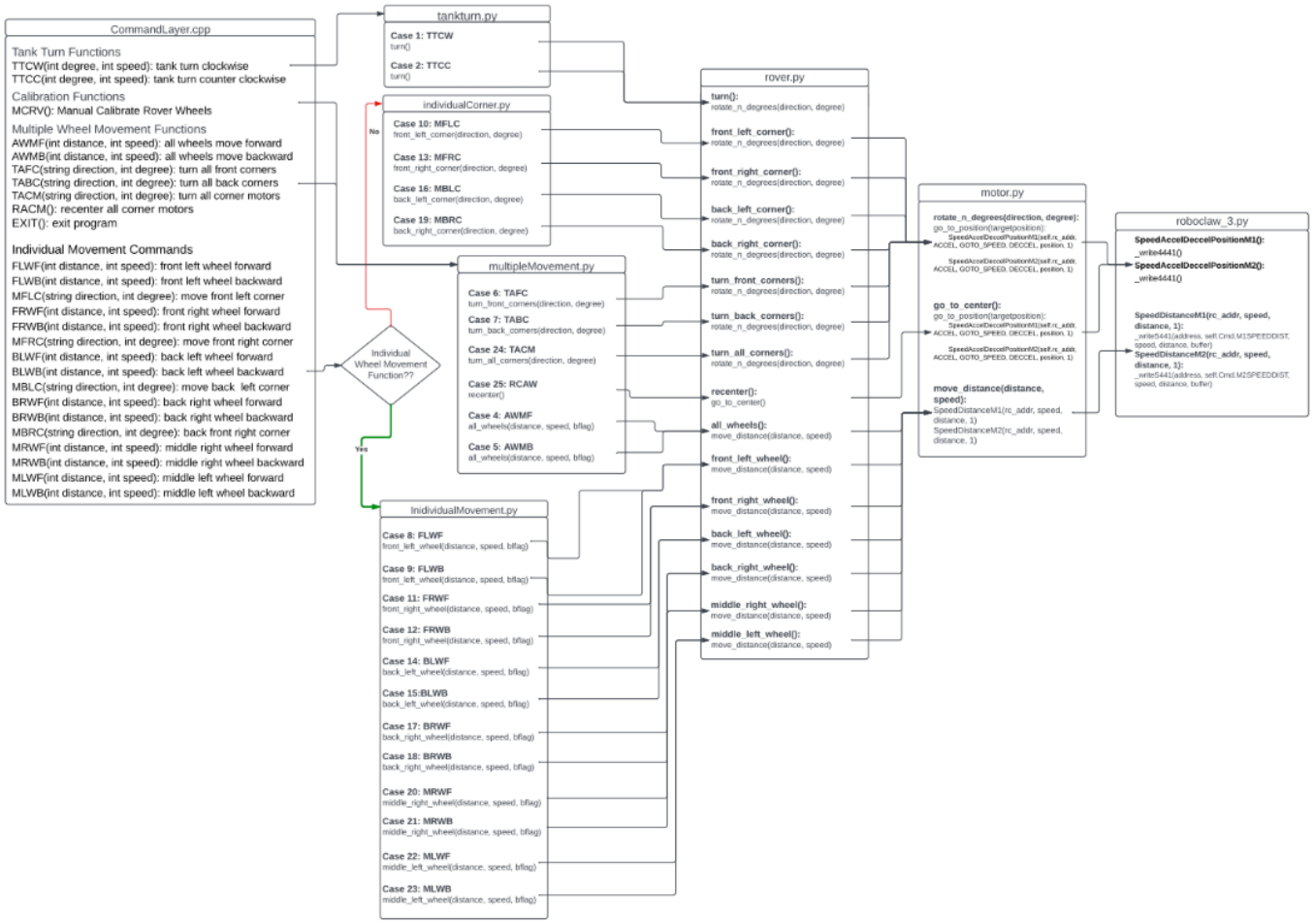
14: BLWF(int distance, int speed): back left wheel forward
Example of usage: BLWF 300 300
15: BLWB(int distance, int speed): back left wheel backward
Example of usage: BLWB 300 300
16: MBLC(string direction, int degree): move back left corner
Example of usage: MBLC right 30

17: BRWF(int distance, int speed): back right wheel forward
Example of usage: BRWF 350 350
18: BRWB(int distance, int speed): back right wheel backward
Example of usage: BRWB 350 300
19: MBRC(string direction, int degree): back front right corner
Example of usage: MBRC left 15

20: MRWF(int distance, int speed): middle right wheel forward
Example of usage: MRWF 300 300
21: MRWB(int distance, int speed): middle right wheel backward
Example of usage: MRWB 300 300

22: MLWF(int distance, int speed): middle left wheel forward
Example of usage: MLWF 300 300
23: MLWB(int distance, int speed): middle left wheel backward
Example of usage: MLWB 300 300

C. Use/Flow Diagram for commandLayer.cpp



References

Hansen, A. (2022). Reassembling the Solar Rover 1. Capstone I. California Polytechnic State University. Unpublished manuscript.

Hansen, A., Kita, E., Randolph, S., Reyna, L., & Russell, D. J. (2022). Alpha Design Report, RoverX. Capstone II. California Polytechnic State University. Unpublished manuscript.

Hansen, A., Kita, E., Randolph, S., Reyna, L., & Russell, D. J. (2023). Design Build Test 1 Milestone: RoverX. Capstone II. California Polytechnic State University. Unpublished manuscript.

Hansen, A., Kita, E., Randolph, S., Reyna, L., & Russell, D. J. (2023). Design Build Test 2 Milestone: RoverX. Capstone II. California Polytechnic State University. Unpublished manuscript.

Hansen, A., Kita, E., Randolph, S., Reyna, L., & Russell, D. J. (2023). Summary of Handoff Contents/Quick Start Guide. Capstone II. California Polytechnic State University. Unpublished manuscript.

Ozawa, R. (2020). Calibration Document for Herbie Roboclaws. Capstone I&II. California Polytechnic State University. Unpublished manuscript.