Use of Spine Articulation in Aerospace Robotics with Monocular SLAM Navigation

Micah Stembridge, Daniel Thornton, Roberto Sauzameda-Carbajal, Benjamin Carpenter, Liam Tarpey

Pueblo Community College, Southwest Campus

Melissa Watters, Faculty Advisor

Colorado Space Grant Consortium Undergraduate Research Symposium

April 13, 2026

Introduction

The rover, named Roverto, was developed to participate in the Colorado Space Grant Consortium Robotics Challenge. This challenge requires participants to design and build an autonomous rover that can traverse through multiple challenge courses with Mars-like terrain. Roverto was designed to fit within the constraints of the challenge, namely, autonomous navigation through sandy and rocky terrain, a 5kg weight limit, and a $500 prototyping budget. Another constraint of the challenge was time; our team had 7 months to design and build the rover. Our team developed two primary goals for this challenge. The first goal was to design a mechanical locomotion system that utilized articulation inspired by reptilian spines. Reptile spines were used as inspiration due to their flexibility and strength. Extensive research demonstrated that reptilian spines are highly effective in lateral motion and navigating uneven surfaces, utilizing flexibility to scale difficult terrain. The rover design has three separate body segments connected via articulation. The reptilian-based design allows the rover to maintain contact with the ground at all times. The second goal was to utilize a monocular SLAM in tandem with ultrasonic sensors to create a custom navigation system that was capable of recognizing and avoiding obstacles, as well as being aware of where those obstacles are in relation to the position of the rover. Another goal with the ultrasonic sensors was to implement fallbacks so that the robot would still be able to navigate through obstacles even when weather conditions or technical challenges impacted the performance of the camera.

Materials and Methods

The rover addressed in this paper was developed for the Colorado Space Grant Consortium Robotics Challenge and was designed to extend concepts developed in a previous robot design iteration completed for the 2025 challenge [1]. That rover consisted of two body pieces connected with a rotating articulation. The 2026 rover expands on that concept by using three main body segments connected with a spine-like articulation able to both move side to side as well as rotate to maintain ground contact. The three body segments were used to house all of the electronics, sensors, and batteries. Motor mounting holes were integrated into the sides of each of the segments in order to reduce the overall weight of the design, as well as to keep the project under budget. The body segments were designed using the Fusion CAD program and then manufactured using 3D printing. In addition to the main body segments, the wheels were also 3D printed.

Wheels were designed to be wide to allow the rover to move across the surface of the sand without sinking in, in effect “floating” on the surface of the sand. The sand at the Great Sand Dunes National Park is very fine, making it difficult to move, even with a design that works with other sand consistencies. The wheels were equipped with four small lateral slots equally spaced, with a section of clear rubber tubing conforming to each slot. The entire piece was then wrapped with a thin sheet of rubber sourced from a bike tire tube. The tubing allowed the wheel to have flexibility, and the rubber gave the wheel grip.

The robot design featured a separate front face with integrated spaces to mount two ultrasonic sensors and a camera used with the SLAM navigation system. In addition to the body segments, eight pieces of articulation, described further in the following section, were used to allow all six wheels to remain in contact with either the ground or another surface at all times. Finally, the rover body segments were fitted with concave lids to allow for as much internal volume as possible. To fabricate all of our custom rover pieces, we used the CAD modeling software Fusion 360. All of these pieces were 3D printed using a mix of Generic and Hyper PLA. The different types of PLA used were determined strictly for aesthetics, as we wanted variations in the color of the different pieces, and we had a limited selection of both types of PLA.

Articulation and Structure

The articulation pieces were partially inspired by the previous year's rover design [1]. While not a direct iteration, the articulation's design criteria remained the same: keep most or all of the wheels on the ground at all times. This year, the articulation was designed to mimic reptile vertebrae, specifically those of lizards, skinks, and snakes. Reptile vertebrae are unique in that they allow for side-to-side, "S-motion," which is useful for navigation around obstacles such as those found on the robotics challenge courses. For example, snake vertebrae have five different points of articulation that connect to neighboring vertebrae. The main central part of the vertebrae has a ball-and-socket joint that connects the vertebrae [2]. The motion allowed from these points of articulation and the ball and socket rotation are the inspiration for the design of the articulation in the rover, which allows for modular and elongated vertebrae-like connections between the three main body pieces.

The design needed to have flexibility, rotation, and be modular for easy part replacement. The articulation development consisted of ten different iterations. After first 3D printing a model of reptile vertebrae for examination, the first two iterations were experiments and proof of concept prototyping using legos and simple 3D modeling. These early prototypes allowed the team to narrow down different ideas about what the articulation should look like and how it should behave.

After these two iterations, the mimicking of reptile vertebrae as the articulation design began and was refined and developed through several more design iterations. The third iteration included the ball and socket joint, like that found in a reptile vertebra. This served as the base for the design, as well as another proof of concept. The prototype was used to confirm that a 3D printed ball and socket joint was feasible and simple enough to manufacture. The fourth iteration slimmed the ball and socket joint to a more goblet or hourglass shape that was partially hollow. This allowed for running all of the wiring going to each of the three body segments through the articulation, much like a spinal cord. One end of the articulation piece had the rounded ball, while the other end had a 4-sided socket that the ball fit into. On every side of the socket, there was a small gap that allowed the socket to flex so the ball could snap fit. This design needed improvement. The pieces needed to be easy to separate or join, so the socket had to be relatively

loose. However, this meant that if the articulation was bent too far over, the ball would slip out of the socket, and the two pieces would become disconnected.

Once again, drawing on inspiration from the reptile spine, the next iterations added prongs on the exterior of the ball and socket joints to allow but also limit motion in multiple directions. As previously mentioned, reptile vertebrae have five points of articulation to allow movement between vertebrae, but that movement is also restricted by those articulations, which is what determines how a reptile bends and moves [2]. Originally, two prongs on opposite sides of the socket were implemented in the articulation design to restrict motion and attempt to combat the disconnection of the articulation joints. The two prongs were insufficient to hold the pieces together, as the articulation pieces would slip out on the other two sides without prongs. Adding two more prongs on the empty sides of the articulation made the connection stronger, but the prongs were still too far apart, and pieces still slipped out. The sixth iteration included the addition of one more prong to all four sides of the articulation, one prong on each side of each socket gap on all four sides. The prongs now stopped the pieces from coming apart and added some motion restriction. While one problem was solved, another presented itself in determining the strength and orientation of the prongs of the simulated vertebrae.

The next iteration featured longer prongs, and each pair of prongs was fused with a filler piece, which proved to be too limiting of motion. It didn't allow the socket to flex at all, making connecting the pieces difficult. The additional length did stop the pieces from coming apart, but the prongs were too long and again limited movement. These issues were addressed with further iterations until the final design selected featured thicker prongs, which kept the pieces together well, allowed for flexibility, and reduced potential damage. The final and tenth iteration was only applied to two of the eight total articulation pieces used. It shortened the prongs that would be directly in front of the middle and back body pieces. In the previous iteration, the length of the prongs worked well when connected to another articulation piece, but they were too long and didn't allow rotation right at the connection between the articulation and body segment. This iteration solved that. For diagrams and visual representations of these iterations, refer to Appendix A.

The design of the three body segments was derived from the body segments used in the version of the rover used in the robotics challenge in 2025 [1]. Key features of the body segments include the articulation connection ports, integrated motor mounts, and integrated sensor mounts. Each segment has sections of the articulation molded into the design to allow for easy connection between the body and articulation segments. The design had the ball facing the front of the rover with all of the prongs facing the back of the rover, just like a reptile spine. Each articulation piece had a central hole in each body to allow wires to be threaded through the articulation into the other body segments. Each body segment had a ledge on the interior of the body, just above where the motors sat. Then a flat plate was placed over the motors to maximize the amount of space in each body. Each plate had sections removed near each opening to allow the condensation of wiring before it entered the narrow articulation. Motor mounting features were added to either side of the body segments to cut weight, lower the cost, and reduce the

overall number of parts needed, as shown in Appendix B. Also shown in Appendix B are an early iteration of those plates.

Wheels and Locomotion

The initial wheel designs were based on a combination of ideas taken from a number of online sources. Many sources advocated for the use of a wide wheel for fine sand [3]. A wheel with a 2.5-inch lateral surface and a 7.5-inch diameter was selected. This was intended to give the rover enough clearance to surmount just about anything that the courses threw its way. Another source provided the idea of using TPU in tandem with PLA to replicate the general design of modern vehicles' wheels [4][5]. The first design iteration was a PLA center rim with a flexible TPU "tire." TPU was intended to provide both flexibility and grip. This idea was designed in CAD using Fusion 360. The biggest problem with the original design was that it was too thick. Printing just one TPU segment would have taken well over 24 hours, much beyond the team's ability to supervise the print. The sequential iterations were aimed at decreasing the print time. While they eventually reduced the print time by a substantial margin, it was still far beyond the needed <8 hours. After several complete iterations on this base idea, the team collectively came to the conclusion that, not only was it too robust a print to manage with available resources, but that the TPU would not provide the utility that we had previously thought it might. A sample printed TPU shoe was what the design was based on, and that took much of our faith in the initial design. However, the TPU and manufacturing materials available were not nearly as similar to rubber as was believed. It wasn't flexible, nor did it provide the required grip.

At this point, the time to prepare for the competition was coming to a close. As a result, the TPU section was scrapped, and we simply used the revised PLA version with a few modifications. Flexible tubing and a rubber sheet sourced from a bicycle inner tube were added to add some traction.

By the time the physical wheels were available to test, much of the body and wiring were already assembled. This presented an unanticipated problem; there wasn't enough space between the body segments to fully accommodate the size of the wheels. This meant that when the rover took a tight turn, the wheels were at risk of hitting one another. This was less than ideal, but the time constraint precluded rewiring the whole rover to add more articulation pieces.

Electrical Components

The electrical subsystem was designed to support the rover's complex locomotion system, onboard computation, and advanced autonomy. The system is centered on the Raspberry Pi 5 and supported by a dedicated motor controller hat and a regulated power distribution network.

The rover relies on six independent DC motors for locomotion, enabling individual control and steering. Each motor is controlled either by the motor controller hat or by an additional L298N motor controller. The hat provides a direct connection to the Raspberry Pi, simplifying the motor control system. The L298N controls the two additional motors not covered by the hat, allowing extra wheels that increase the rover's locomotive capabilities [6].

Power regulation and distribution are critical aspects of the rover's electrical system. The Raspberry Pi 5 operates at 5V, while the DC motors operate at 12V. A buck converter, which steps down a higher input voltage to a lower designated voltage, is required to prevent power overload from frying the Raspberry Pi. In addition, a voltage regulator is needed to ensure that the output from the ultrasonic sensors does not harm the Raspberry Pi[7]. We used efficient cable management, color-coding the various wires running through the rover and grouping wires from the same component.

During initial system testing, thermal regulation of the Raspberry Pi became a concern. However, after further testing, it was evident that overheating was less of a concern under typical operating conditions than originally thought. To ensure reliability, an additional fan was included in the design in case overheating were to become an issue.

Autonomous navigation is handled by onboard processing on the Raspberry Pi, the rover's central computer. The system implements a monocular visual simultaneous localization and mapping (vSLAM) system, allowing the rover to map its environment and localize itself using a single camera input [8]. This system was selected for its low hardware requirements and the opportunity for advanced autonomy across many environments. This makes it well-equipped to handle the limitations of this project. Additional considerations ensured that the camera and processing unit functioned cohesively, including secure physical mounting and thoughtful cable management. The rover was also equipped with two ultrasonic sensors, which functioned in conjunction with the camera and as auxiliary autonomous detection. These sensors were positioned on either side of the camera and mounted securely to provide an optimal view for detection. These implementations support the rover’s ability to gather visual data needed for navigation while maintaining electrical organization.

Testing of the electrical subsystem was conducted throughout the build process to identify potential issues before they affected the rover. Initial testing focused on ensuring that electrical components functioned properly, including correct motor rotation, accurate visuals and data from the sensors, and proper communication between the Raspberry Pi and the motor controllers. Subsequent testing involved increasing load conditions and simulating likely environments to confirm that systems would function under these challenges. These tests proved critical in identifying issues with power stability and the rover's wiring layout. Refer to Appendix C to view some images of the electronic systems.

Software and Autonomy

As was the case for other parts of the rover design, the design was built on the ideas and results from the rover designed for the 2025 Colorado Space Grant Consortium Robotics Challenge. That design used an autonomous system based on the readings of a swiveling ultrasonic distance sensor on the front of the rover [1]. While functional, that design's simplicity limited its autonomy, and a more advanced system for this rover was explored. During the brainstorming phase, we decided to test the practicality of using a single RGB camera on the

front of the rover to assist with obstacle avoidance and, more importantly, to help localize the rover and give it a sense of direction and position relative to the obstacles around it. To accomplish this, we decided to utilize a SLAM (Simultaneous Localization and Mapping) system on our rover [9]. It was decided that additional distance sensors should also be implemented on the front of the rover as a backup system, just in case our camera-powered SLAM system misses an obstacle or fails. One other aspect that was specifically included in the software of the rover is a map of the rover's surroundings that is continuously updated and saved as the rover moves around. That way, as the operation time of the rover increases, the precision of the map and navigation will also increase.

During initial code prototyping, it was quickly determined that running the camera SLAM system, navigation and mapping system, and running all of the motors and sensors would require a lot more power than what could be provided by using an Arduino or equivalent motherboard [10]. We also discovered that running a complex SLAM system would also require running the code on top of a computer operating system [11]. Due to this, it was decided very early on to use a Raspberry Pi 5 (8 GBs RAM) for its high performance and its high level of support for different operating systems and packages [12]. We also decided to use this motherboard for its easy camera integration via the Raspberry Pi camera modules [13].

After preliminary prototyping, it was determined that there was an issue with linking the SLAM system, the camera input, our mapping visualization software, and the low-level machine code that runs the hardware for the sensors and the motors. In order to resolve this issue, we decided to use ROS2 (Robot Operating System 2) [14]. ROS2 runs on top of an operating system and allows for the creation of custom packages and nodes that can then publish and subscribe to different topics and actions. This allows for different parts of the software to run completely independently of one another yet still communicate and work together, even across different languages and systems. Due to this, we decided to primarily use Ubuntu 24 LTS as our OS due to its Tier 1 support for ROS2 [15]. Another reason to use ROS2 was its pre-existing over LAN debugging and visualization features. Initially, ROS2 Humble was installed on the system for testing [16], but after further evaluation, it was decided that ROS2 Jazzy would be used for its modern operation and increased customizability over ROS2 Humble [17].

After ROS2 Jazzy and Ubuntu 24 LTS were installed and tested on the Raspberry Pi, multiple different ways to connect, monitor, and control the computer remotely were evaluated. At first, we attempted to utilize a VNC connection over wifi and the internet, but eventually we decided to simply use SSH with X11 Tunnelling for our connection to the rover because we were having issues with the performance and frame rate of VNC [18]. In the end, two SSH connections were created, one could be connected via Wi-Fi, and another could be used via an Ethernet cable. The GNOME Desktop environment was installed so that we could connect the computer to a monitor if our regular SSH connections failed for some reason [19].

After installing and setting up the connection protocols, we began setting up our SLAM system for initial testing. The ORBSLAM3 package was selected due to its efficiency in creating a map and a localization estimate based on monocular camera frames [8][20][21]. We originally

tested ORBSLAM3 via the ozandmrz GitHub fork [22]. This fork was specifically created to run with easy setup on Raspberry Pi OS with ROS2 Humble, and even though this was not the setup desired for the final design, it was installed to test the performance of the SLAM and make sure that it would run well on the Raspberry Pi. After further evaluation, it was confirmed that ORBSLAM3 could run at 30 FPS with the Raspberry Pi 3 Camera Module. After confirming that the ORBSLAM3 packages could run well on the system, we began to configure a custom SLAM system to work better with the robot. Refer to Appendix D for images of the initial SLAM testing.

A custom camera publisher node was created that would access the camera connected to the motherboard and publish camera frames at 22 frames a second in a format and aspect ratio that the SLAM package could utilize. There were some slight challenges with this, however, as the default Ubuntu camera drivers are not designed to work with the special camera connection slots of the Raspberry Pi. So to solve this issue, it was necessary to install the RPICAM packages onto the Ubuntu system and make certain that our camera node would specifically use the new camera drivers [23].

After the camera publisher node was completed and tested, the ORBSLAM3 package was implemented into the ROS2 environment. This began by implementing the Mechazo11 ros2_orb_slam3 fork, which was specifically designed to be compatible with ROS2 Jazz [24]. While this package worked well with the system, it did not have the built-in features required to publish the point cloud or the rover pose that is produced by the SLAM. Instead of creating custom point cloud or pose publishers, we merely linked a different ORBSLAM3 fork that publishes the point cloud and pose. In the end, it was decided to use the ozandmrz fork for this [25]. In order to get this other package to work, it was necessary to create symbolic links within the software to connect the new forks' dependencies to the original Mechazo11 forks' dependencies that were designed to work with our ROS2 Jazzy environment. This approach ended up working very well and saving a lot of time because by linking the two ORBSLAM3 packages, we were able to run the SLAM on our system while also being able to publish the point cloud and pose data to other nodes without having to create custom publishers.

At this point, the SLAM system was tested more thoroughly, and we also began to tune its parameters to the needs of the specific rover and camera. The first step of this process involved creating a custom rpicam.yaml file that included all of the settings and configurations for the specific camera module. To make this configuration file accurate, we utilized the included ROS2 camera tuner that looks through your camera to track a chessboard, and from that data, the system can produce an accurate camera configuration file [26]. At this point, we began to focus on maximizing the efficiency of our system and using alternative debugging and visualization tools. The first step in this process involved disabling the Pangolin viewer that is activated by default with ORBSLAM [27]. From this point forward, we began using RViz2 for the main SLAM visualizer [28]. This allowed us to “see” what the robot was seeing via a different computer, allowing the rover to effectively run completely headless from this moment forward.

Please view Appendix E to view our configuration file, and Appendix F to view images of the RViz2 tool.

At this point during the development process, most of the electrical components had been assembled and tested, so once the SLAM was completed, we began creating our custom ROS2 packages that would handle all of the hardware interactions. A custom package called rovultrasonic was created that had two nodes titled publishd1 and publishd2. Both of these nodes simply publish the data gathered from the ultrasonic sensors and format the information into a topic that all of the other nodes can understand. The second custom package that was created was called motor_controller, and this package just had one node called motor_controller_node. As the name suggests, this node was in charge of actually moving the rover. The node subscribes to a topic called /cmd_vel and moves the rover according to the input it receives. This node separation was done to make debugging easier, as the navigation code will now just have to publish a velocity command to /cmd_vel in order to move the rover.

Originally for navigation, it was intended to use the pre-included ROS2 Nav2 package to handle charting courses and saving mapping data [29]. But due to time constraints, as well as some of the scale inaccuracies associated with using a monocular SLAM, we ended up creating a custom navigation package. We titled our package rovnav, and it included the following nodes: uonly and rovnav. The uonly file allows the robot to drive and navigate only using the two ultrasonic sensors as a backup in case the SLAM stops working. The rovnav node allows the rover to navigate based on data from the ultrasonic sensors, and based on the map and position published by the SLAM. View Appendix G to see our original node flow chart and our updated node flow chart.

The operation of the uonly node is fairly straightforward. This node simply tells the motors to drive in a forward direction until the range published by the ultrasonic sensors dips below a certain threshold, in which case the robot will stop, back up slightly, and turn to go around the obstacle. The rovnav file is slightly more complicated and also requires the use of another ROS2 package called Pointcloud_to_laserscan [30]. As the name implies, this node transforms the 3D point cloud published by the ORBSLAM3 into a 2D laserscan topic that is easier for the rovnav node to process quickly. One other thing to note about this is the fact that the returned 2D laser scan will also purposely exclude certain points that are considered to be a part of the ground or points that are considered to be much higher than the rover. This is done with settings within the node that describe the height of the rover, as well as where the ground is relative to the pose published by the SLAM.

The rovnav node utilizes this 2D laserscan topic alongside readings from the ultrasonic sensors to chart a path around obstacles. When the code first launches, the rovnav node tells the rover to enter what we are calling a “wiggle” cycle, where the rover will go forward and backward in very slow, controlled movements to give the monocular SLAM time to find obstacles. As soon as the node receives a laserscan and a position from the SLAM, the rover will then chart a course by splitting the laserscan into multiple separate parts and then doing a sort to discover which part of the map contains the least amount of obstacles. Once this is done, the

rover will then point itself in the direction with the least amount of obstacles and will then begin to drive forward in that general direction. During the “wiggle” phase, if no laserscan or pose topic is published by the SLAM within a certain timespan, the node will timeout and default to an ultrasonic-only mode very similar to the process described in the uonly node. When driving with the SLAM active, every 10 seconds, the rover is set to halt its motion for 5 seconds to allow the SLAM to recreate the map in case the camera was jiggled or overwhelmed with too much motion blur to keep the SLAM active. During this process, the ultrasonic sensors act as a backup, and if their readings come back underneath a certain threshold, the rover will stop, reverse itself, and attempt to navigate around the detected obstacle. This navigation system was created to utilize the positive aspects of a monocular SLAM, such as extremely fast obstacle and point recognition, while minimizing the negative aspects of using a monocular SLAM, such as inaccurate depth and the consistency issues of holding a single map while going over rough terrain.

Once all of these nodes were completed and tested, a single ROS2 Launch file was created [31]. This allowed the entire system and rover to be run off a single SSH command. View Appendix H to view some more images and documents regarding the development of our SLAM system.

Results and Issues

The goal of this project was to create a fully autonomous rover under 5kg for the Colorado Space Grant Consortium. The main test of our rover happened on the date of the robotics challenge, where our rover had to drive autonomously through multiple different courses in the Great Sand Dunes National Park. During the challenge, our team faced several unforeseen challenges. Namely, an inability to connect to the rover to initiate the code. While this issue was quickly resolved, this was far from the only complication that arose. Another notable complication that was of note was the weight distribution and the consequential buckling of the articulation. The bulk of the internal weight sat in the batteries that were placed in the center compartment. This caused the middle section to slow down and pull the articulation on either side of the middle segment downwards. This, in turn, caused the back segment to push up against the articulation separating the two, causing a buckling effect and, at times, jackknifing.

Another result was that our frontal optic sensors had a tendency to assume too steep an angle to see oncoming obstacles due to the articulation pulling the front part of the rover upwards. The articulation worked well in its intended purpose. It allowed the rover wheels to remain on a solid surface at all times, even if it was an obstacle and not the ground. However, there were not enough articulation pieces between the main body segments. This caused the wheels to collide with each other and made the rover go in circles sometimes after the rover would turn. The speed of the different body segments also made the sensors point up towards the sky and not forward towards the obstacles at times. This body rotation also caused the articulation to pass its critical point, or the point where the pieces began separating.

The wheels worked far better than expected. They had a large amount of surface area on the sand at all times, but when they sank, the internal wheel supports acted almost like paddles, continuing to propel the rover. The innertube on the exterior of the wheel also performed well. The software performed admirably, even if it didn't necessarily work well alongside the hardware. The ultrasonic sensors did a very good job at detecting walls and immediate obstacles that the SLAM missed. While the ORBSLAM3 packages worked very well at creating an initial map when the rover started a course, the SLAM would often delocalize or fail partway through the course due to excessive bumping, movement, and sometimes direct lighting into the camera. However, this was not a big issue because the initial SLAM map was typically accurate enough for the rover to plan a path around all major obstacles, while the ultrasonic sensors worked well as backup and were able to give the robot the ability to easily move around any obstacles that were missed by the SLAM. Overall, while each system performed very well, these systems did not integrate well together, which led to many of the issues mentioned above. View Appendix I for some images of the rover at the robotics competition.

Conclusion and Improvements

Ultimately, the rover performed relatively well. Each system played its part as well as anyone could have hoped, but lacked overall synergy. The articulation articulated perfectly, but it also allowed for jackknifing. The code worked well in theory, but was not properly tested with the hardware of the rover. For example, the rover was intended to turn with a modified skid steering system. This worked well, but once the turning was complete, it did not correct its direction to return to a straight line of travel, causing a perpetual spiral. This could be improved by assigning each rover segment an individual IMU. The wheels were excellent. Any of the issues with them could have been corrected by simply adding more articulation in between the segments to make jackknifing more difficult. While the articulation itself worked pretty well, there was one instance when the individual vertebrae disconnected. More extensive testing could have revealed this weakness, and further iterations of the articulation pieces could have eliminated it by creating a stronger tolerance between each piece. The software worked well by itself and in theory, but when paired with the actual physical components of the rover, there were a few issues. Most, if not all, of the issues that took place could have been ironed out had more testing been conducted. Some specific improvements that could be implemented on this design include adding more cameras and or depth sensors to the SLAM, so that the map would have a much smaller chance of delocalizing. Another improvement that could be made to this design is a more specifically calculated length of articulation between each segment to assist with better steering and to avoid jackknifing altogether. In conclusion, the rover Roverto demonstrated that bio-inspired rotational articulation can definitely be a viable means of rough terrain transportation, specifically for aerospace robotics, due to its flexibility and ability to keep the rover grounded at all times. This project also demonstrates that using a monocular SLAM can be a viable means of advanced autonomous navigation for robotics, given that the SLAM system is

tuned properly, can calculate depth accurately, and has multiple backup sensors in case the SLAM system fails.

References

[1] D. Thornton, M. Stembridge, M. Mumbower, B. Carpenter, and R. Sauzameda-Carbajal, "Rotating Articulation in Robotics: Experimenting with the functionality of free rotating articulation for traversing sandy and rocky terrain," presented at the 2025 Pikes Peak Undergraduate Space Research Symposium, Colorado Springs, CO, USA, Apr. 26, 2025.

[2] *"Skeletal System,"* The Biomechanics of Snake Locomotion. [Online]. Available: https://snakelocomotion.wordpress.com/the-skeletal-system/. [Accessed: Jan 23, 2026].

[3] D. Gee-Clough, "The effect of wheel width on the rolling resistance of rigid wheels in sand," *Journal of Terramechanics*, vol. 15, no. 4, pp. 161–184, Dec. 1978, doi: https://doi.org/10.1016/0022-4898(78)90002-2.

[4] D. Stone, "David Stone," *David Stone*, 2023. https://www.davidlahrenstone.com/portfolio/rover-wheels

[5] Z. Wang and A. R. Reece, "The performance of free rolling rigid and flexible wheels on sand," *Journal of Terramechanics*, vol. 21, no. 4, pp. 347–360, Feb. 2003, doi: https://doi.org/10.1016/0022-4898(84)90026-0.

[6]Handson Technology, "L298N Dual H-Bridge Motor Driver Module Datasheet," Handson Technology, [Online]. Available: https://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf. [Accessed: Nov. 21, 2025].

[7] Circuit Basics, "How Voltage Regulators Work," Circuit Basics, [Online]. Available: https://www.circuitbasics.com/voltage-regulators/ [Accessed: Feb. 26, 2026].

[8] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multi-Map SLAM," IEEE Transactions on Robotics,
vol. 37, no. 6, pp. 1874-1890, Dec. 2021. [Online]. Available: https://arxiv.org/abs/2007.11898

[9] C. Cadena et al., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust Perception Age," IEEE Transactions on Robotics, vol. 32, no. 6, pp. 1309-1332, Dec. 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7747236

[10] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, Oct. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7946260

[11] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, eabm6074, 2022. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abm6074

[12] Raspberry Pi Ltd, "Raspberry Pi 5," Raspberry Pi Documentation, 2023. [Online]. Available: https:// www.raspberrypi.com/documentation/computers/raspberry-pi-5.html

[13] Raspberry Pi Ltd, "Camera Hardware," Raspberry Pi Documentation, 2023. [Online]. Available: https:// www.raspberrypi.com/documentation/accessories/camera.html

[14] Open Robotics, "ROS 2 Documentation," Open Robotics, 2024. [Online]. Available: https://docs.ros.org/en/jazzy/index.html

[15] Open Robotics, "REP 2000 - ROS 2 Releases and Target Platforms," ROS Enhancement Proposals, 2024. [Online]. Available: https://ros.org/reps/rep-2000.html

[16] Open Robotics, "ROS 2 Humble Hawksbill," ROS 2 Documentation, 2022. [Online]. Available: https:// docs.ros.org/en/humble/

[17] Open Robotics, "ROS 2 Jazzy Jalisco," ROS 2 Documentation, 2024. [Online]. Available: https:// docs.ros.org/en/jazzy/

[18] R. Hertzog and C. Mas, "Remote Login," in The Debian Administrator's Handbook, Freexian SARL, 2015, ch. 9. [Online]. Available: https://debian-handbook.info/browse/stable/sect.remote-login.html

[19] The GNOME Project, "GNOME Desktop," 2024. [Online]. Available: https://www.gnome.org/

[20] A. M. Kamal et al., "An In-depth Evaluation of ORB-SLAM3 on the Raspberry Pi 5: Performance, Stability, and Design Guidelines for Embedded SLAM," Preprint, Mar. 2026. [Online]. Available: https:// www.researchgate.net/publication/400559478

[21] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multi-Map SLAM," IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1874-1890, Dec. 2021. [Online]. Available: https://arxiv.org/abs/2007.11898

[22] O. Demirel (ozandmrz), "raspberry_pi_visual_slam," GitHub repository, 2023. [Online]. Available: https://github.com/ozandmrz/raspberry_pi_visual_slam

[23] Raspberry Pi Ltd, "rpicam-apps," Raspberry Pi Documentation, 2023. [Online]. Available: https://www.raspberrypi.com/documentation/computers/camera_software.html

[24] A. M. Kamal (Mechazo11), "ros2_orb_slam3 (Jazzy branch)," GitHub repository, 2024. [Online]. Available: https://github.com/Mechazo11/ros2_orb_slam3/tree/jazzy

[25] O. Demirel (ozandmrz), "ORB_SLAM3," GitHub Repository, 2023. [Online]. Available: https://github.com/ozandmrz/ORB_SLAM3

[26] Open Robotics, "image_pipeline: camera_calibration," ROS 2 GitHub Repository. [Online]. Available: https://github.com/ros-perception/image_pipeline/tree/jazzy/camera_calibration

[27] S. Lovegrove and A. J. Davison, "Pangolin: A lightweight portable rapid development library for AV and computer vision," GitHub repository. [Online]. Available: https://github.com/stevenlovegrove/Pangolin

[28] Open Robotics, "RViz2 User Guide," ROS 2 Jazzy Documentation. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Intermediate/RViz/RViz-User-Guide/RViz-User-Guide.html

[29] Open Navigation, "Nav2 Documentation," 2024. [Online]. Available: https://docs.nav2.org/

[30] ROS Perception, "pointcloud_to_laserscan: Converts a 3D Point Cloud into a 2D laser scan," GitHub repository. [Online]. Available: https://github.com/ros-perception/pointcloud_to_laserscan

[31] Open Robotics, "Integrating launch files into ROS 2 packages," ROS 2 Jazzy Documentation. [Online]. Available:https://docs.ros.org/en/jazzy/Tutorials/Intermediate/Launch/Launch-system.html

Appendix

Appendix A: Articulation Iterations

Figure 1: Fourth iteration

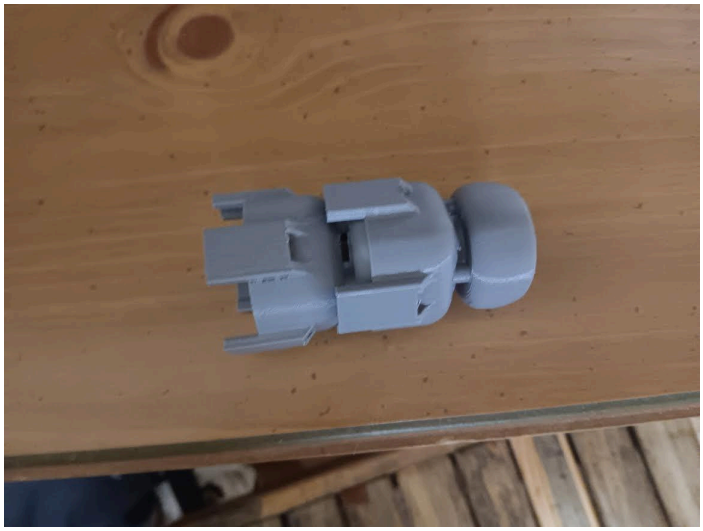

Figure 2: Seventh iteration

Figure 3: Ninth iteration

Appendix B: Rover Body Segments

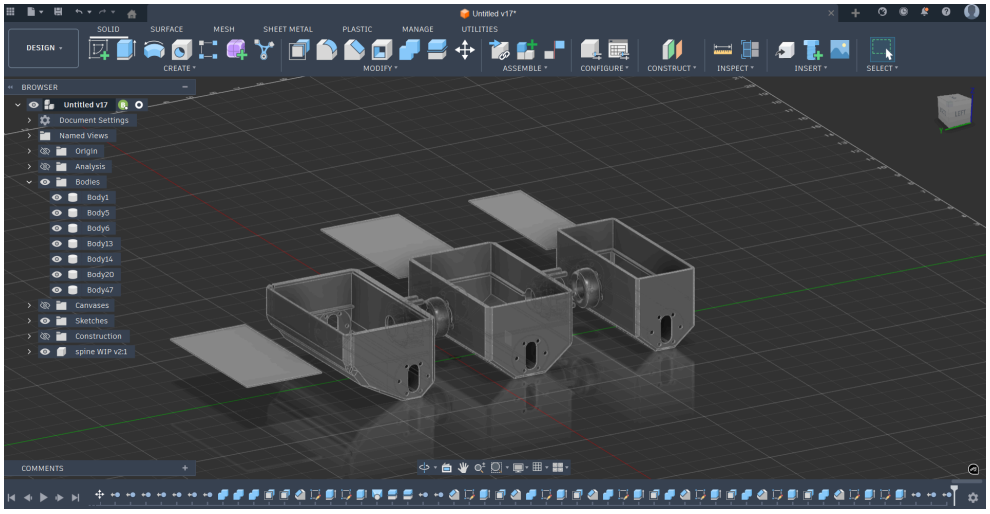

Figure 1: CAD drawing of body segments

Appendix C: Electronics

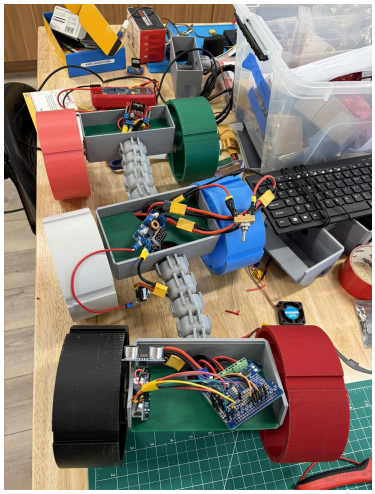

Figure 1: An image of the electronics in the rover during early testing

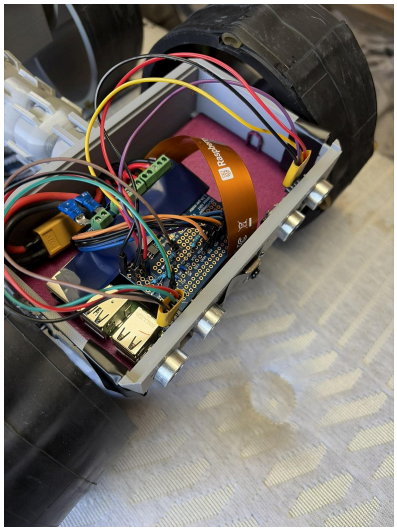

Figure 2: An image of the electronics in the front of the rover during our final week of testing

Appendix D: Early SLAM Testing

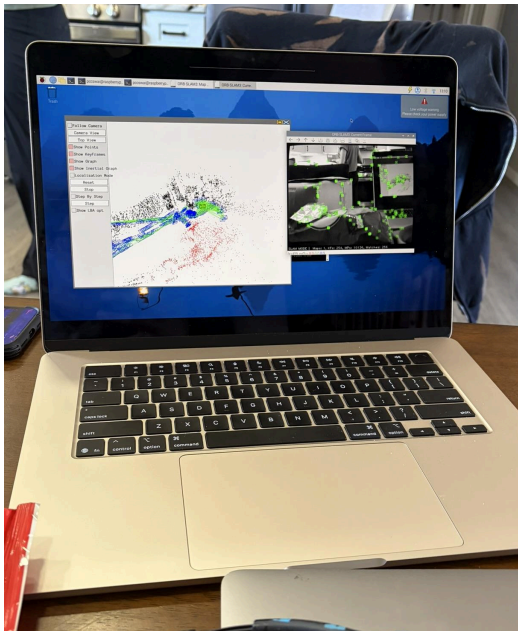

Figure 1: Early SLAM testing in the kitchen.

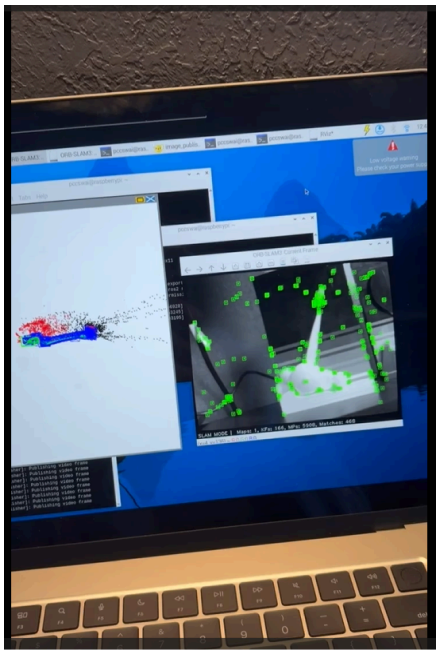

Figure 2: Early SLAM testing on the desk.

Appendix E: Camera Configuration File

```yaml
%YAML:1.0

#--------------------------------------------------------------------------------------------
# Camera Parameters
#--------------------------------------------------------------------------------------------
File.version: "1.0"

Camera.type: "PinHole"

# Camera resolution
Camera.width: 752
Camera.height: 480

# Camera calibration and distortion parameters (extracted from your data)
# fx, fy, cx, cy from camera_matrix
Camera1.fx: 940.85269
Camera1.fy: 938.10871
Camera1.cx: 383.41181
Camera1.cy: 234.01118

# Distortion coefficients: [k1, k2, p1, p2, k3]
Camera1.k1: 0.012275
Camera1.k2: 0.205665
Camera1.p1: -0.001651
Camera1.p2: 0.003243
Camera1.k3: 0.0

# Camera frames per second
Camera.fps: 22

# Color order of the images (0: BGR, 1: RGB. It is ignored if images are grayscale)
Camera.RGB: 0

#--------------------------------------------------------------------------------------------
# ORB Parameters
#--------------------------------------------------------------------------------------------
ORBextractor.nFeatures: 1200
ORBextractor.scaleFactor: 1.2
ORBextractor.nLevels: 8
ORBextractor.iniThFAST: 20
ORBextractor.minThFAST: 7

#--------------------------------------------------------------------------------------------
# Viewer Parameters
#--------------------------------------------------------------------------------------------
Viewer.KeyFrameSize: 0.05
Viewer.KeyFrameLineWidth: 1.0
Viewer.GraphLineWidth: 0.9
Viewer.PointSize: 2.0
Viewer.CameraSize: 0.08
Viewer.CameraLineWidth: 3.0
Viewer.ViewpointX: 0.0
Viewer.ViewpointY: -0.7
Viewer.ViewpointZ: -1.8
Viewer.ViewpointF: 500.0
```

Figure 1: RpiCAM.yaml

Appendix F: Rviz2

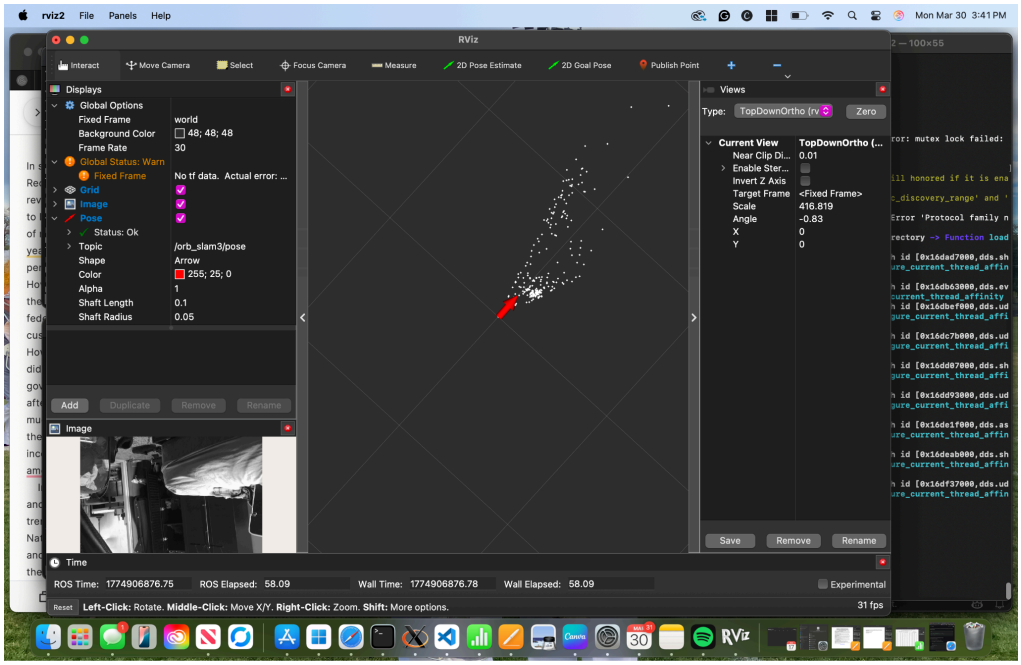

Figure 1: RViz2 running on MacOS and displaying the point cloud, laserscan, and pose generated by the SLAM

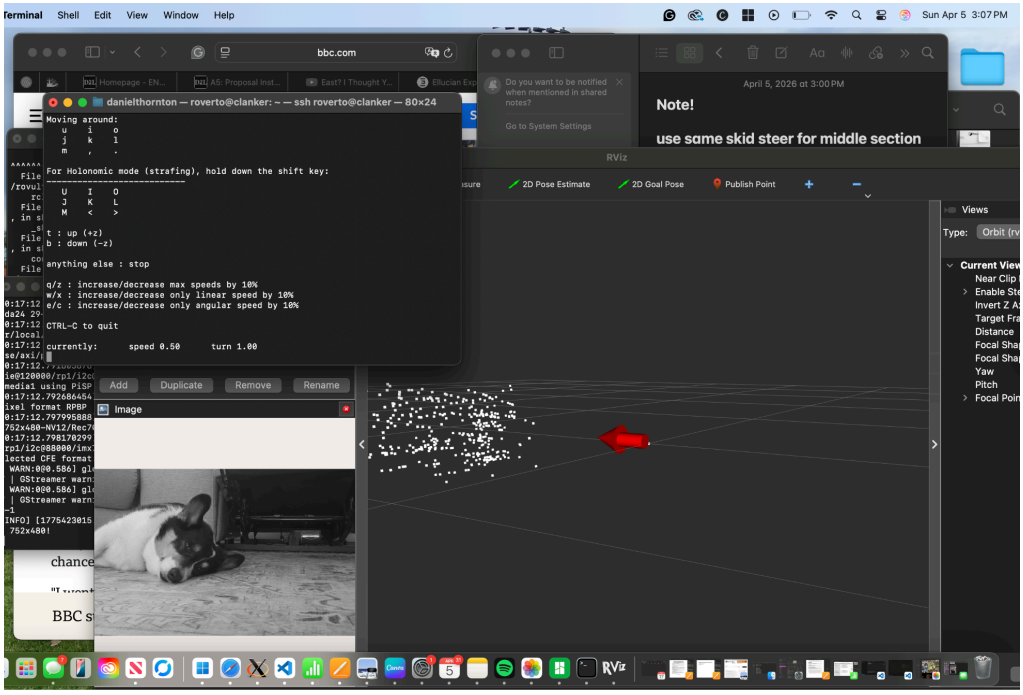

Figure 2: RViz2 displaying point cloud and pose from SLAM while the robot is facing a dog on the ground.

Appendix G: Node Flow Charts

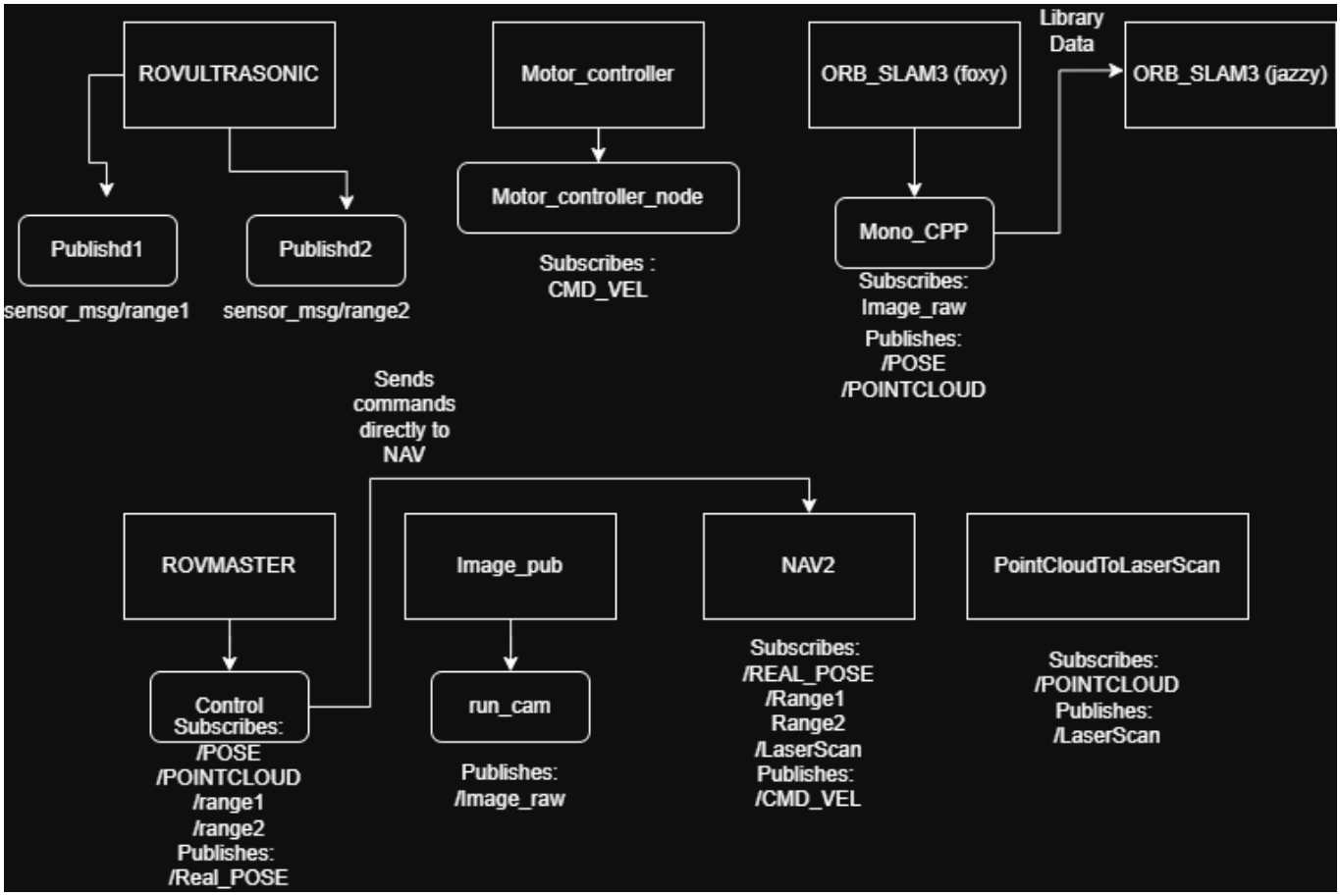

Figure 1: Original Node Flow Chart (includes Nav2 references)

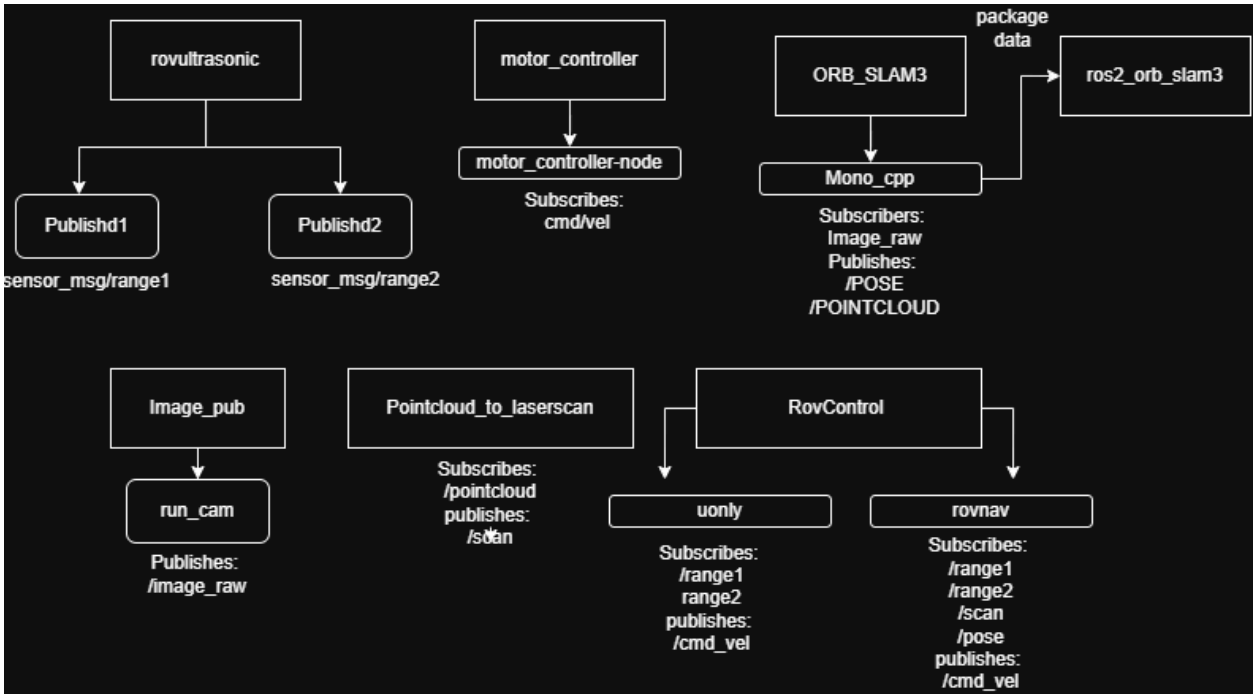

Figure 2: Modified Node Flow Chart (includes our custom navigation nodes)

Appendix H: Other Images related to the development of our SLAM system

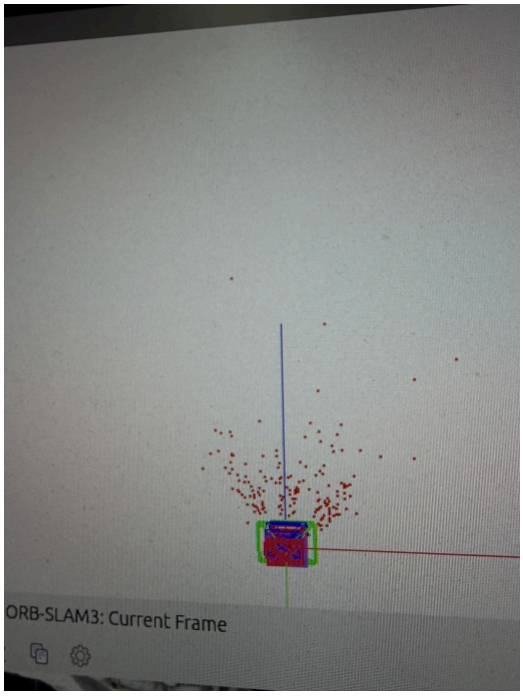

Figure 1: An image of the Pangolin 3D viewer.

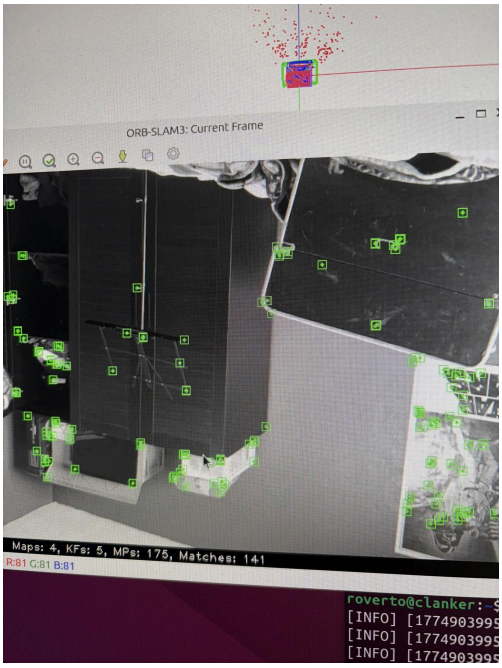

Figure 2: An image of the SLAM detecting ORB points on a camera frame.

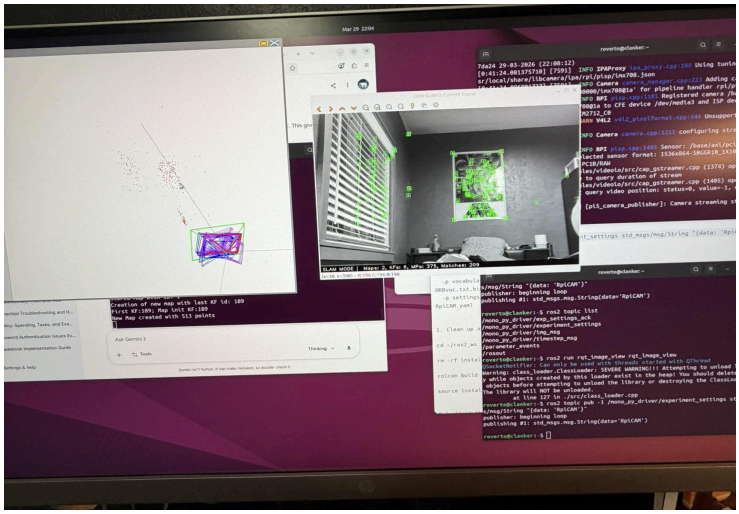

Figure 3: Multiple Nodes and visualization software running on the Raspberry Pi

Appendix I: Project Results.

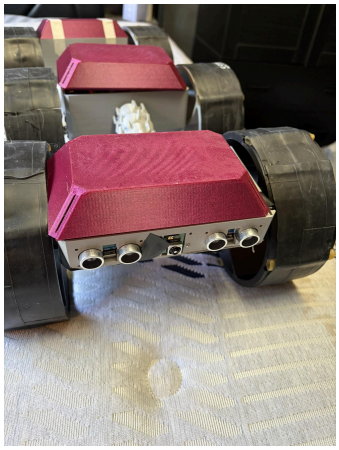

Figure 1: an image of the rover a few days before the competition.

Figure 2: An image of the rover the day before the competition.

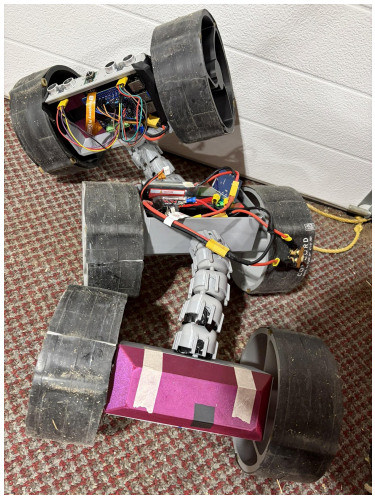

Figure 3: an image of the rover's articulation working during testing