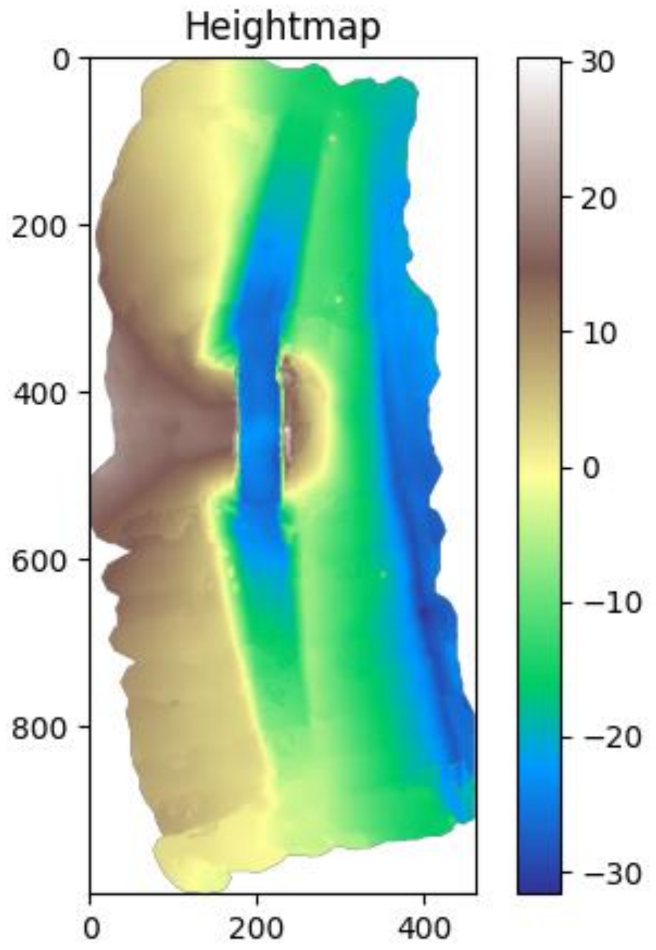
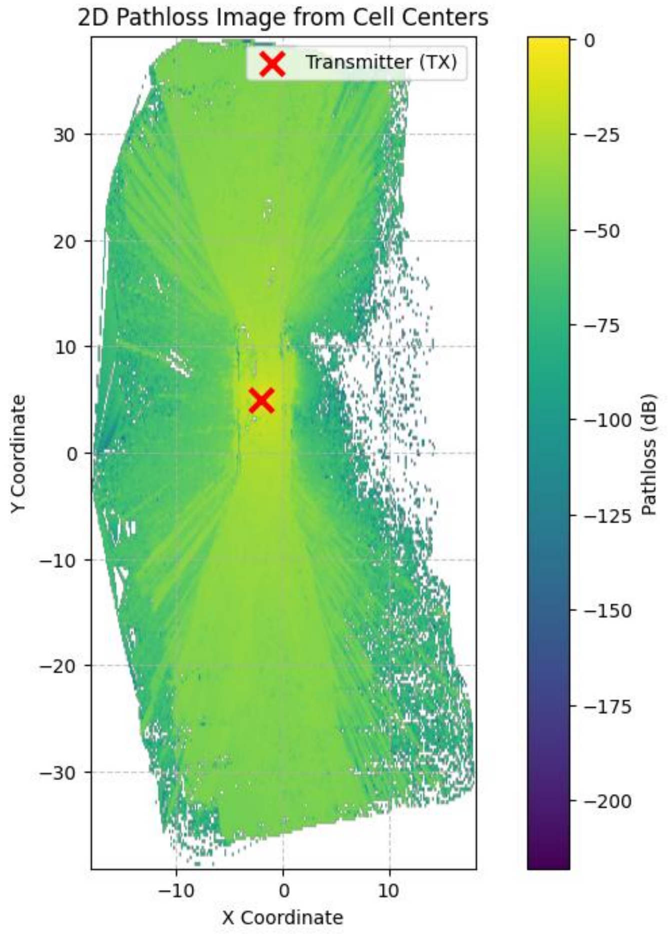
# NASA DCGR Project
# End of Internship Presentation

Alex Moscibroda

# Context

- I worked on the NASA DCGR Project, specifically on RadioLunaDiff.
- The goal of RadioLunaDiff is predicting RF signal strength on lunar terrain based on a given heightmap.
- My focus within this project was to enable real-world testing of the RF models.



# Problem

- To enable real-world testing, we needed to:
    1. Scan locations to generate PLY mesh.
    2. Convert PLY mesh into an heightmap for RF simulation.
    3. Create an RF model to predict signal strength.
    4. Use the heightmap to predict RF signal strength in the location.
    5. Program the Astrobotic rover to traverse the location and collect RF data.
    6. Compare the RF model with real-world RF data.
- The code either did not exist or had to be heavily adapted, so I had to implement it myself.

Mapping

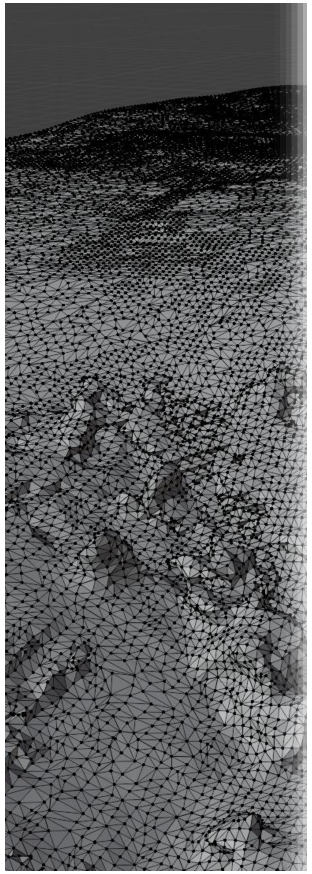
Path Planning

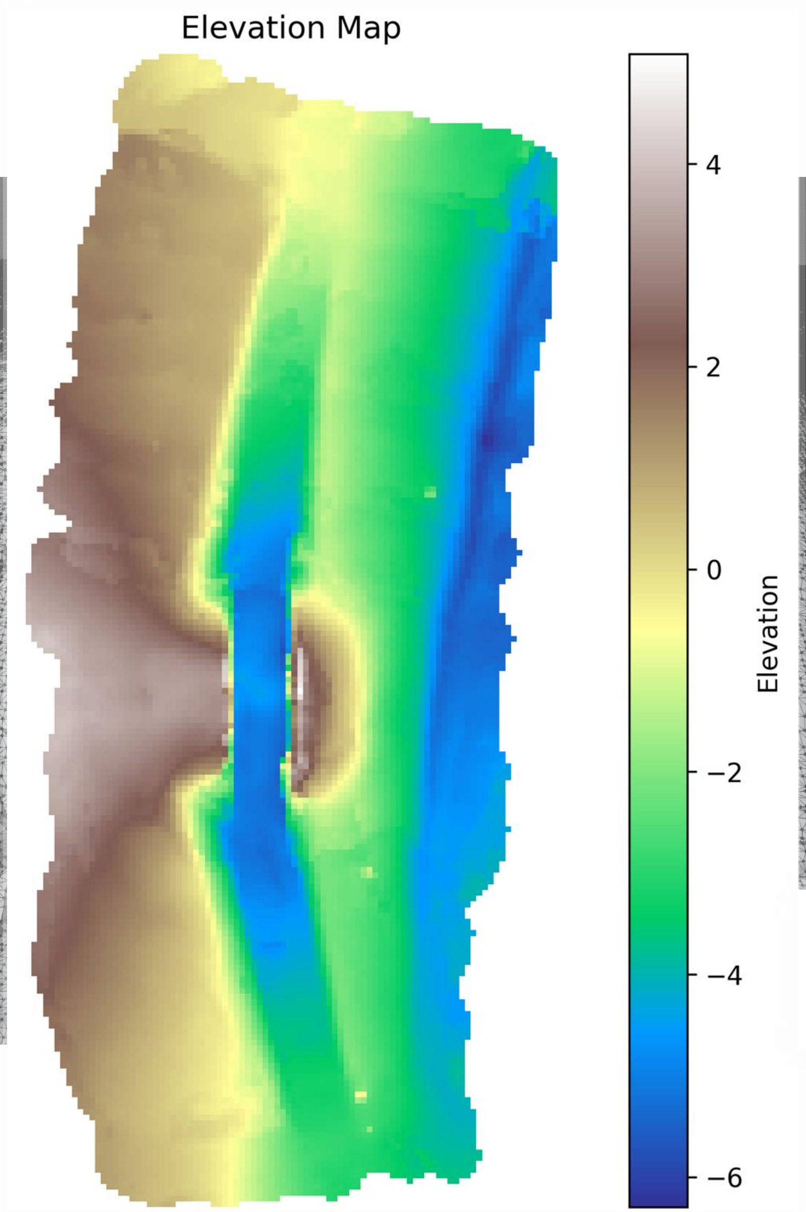
Rover Control

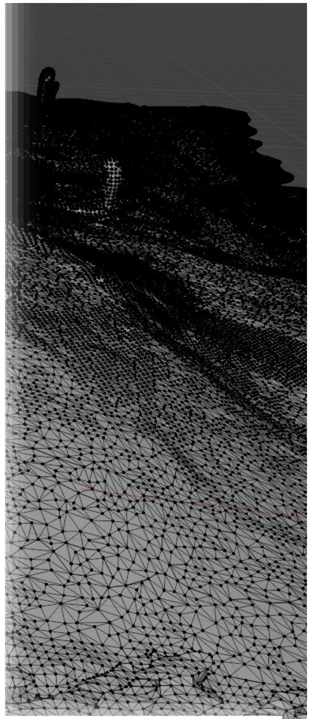
# Mapping

*— Scan and Create Heightmap —*

- Created **LiDAR scans** with an iPhone using the Polycam app.
- My early scans had heavy drift (the first one took around 4 hours), so I improved the process by combining multiple shorter scans together in Blender.
- Created PLY meshes of both the Urban Canyon and the UW Golf Course using this approach.
- Wrote code to **generate elevation maps from PLY meshes** and to convert elevation maps back into PLY, even for non-square maps.
- Fixed scaling and orientation issues so that the outputs were consistent and usable.





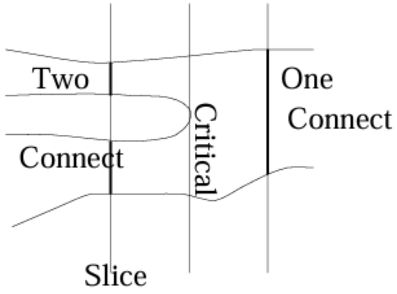



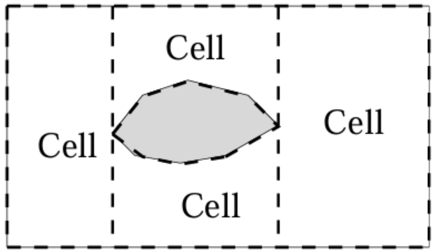
Axis ranges (m):
X: -18.13 to 18.13
Y: -2.48 to 2.54
Z: -39.18 to 39.18

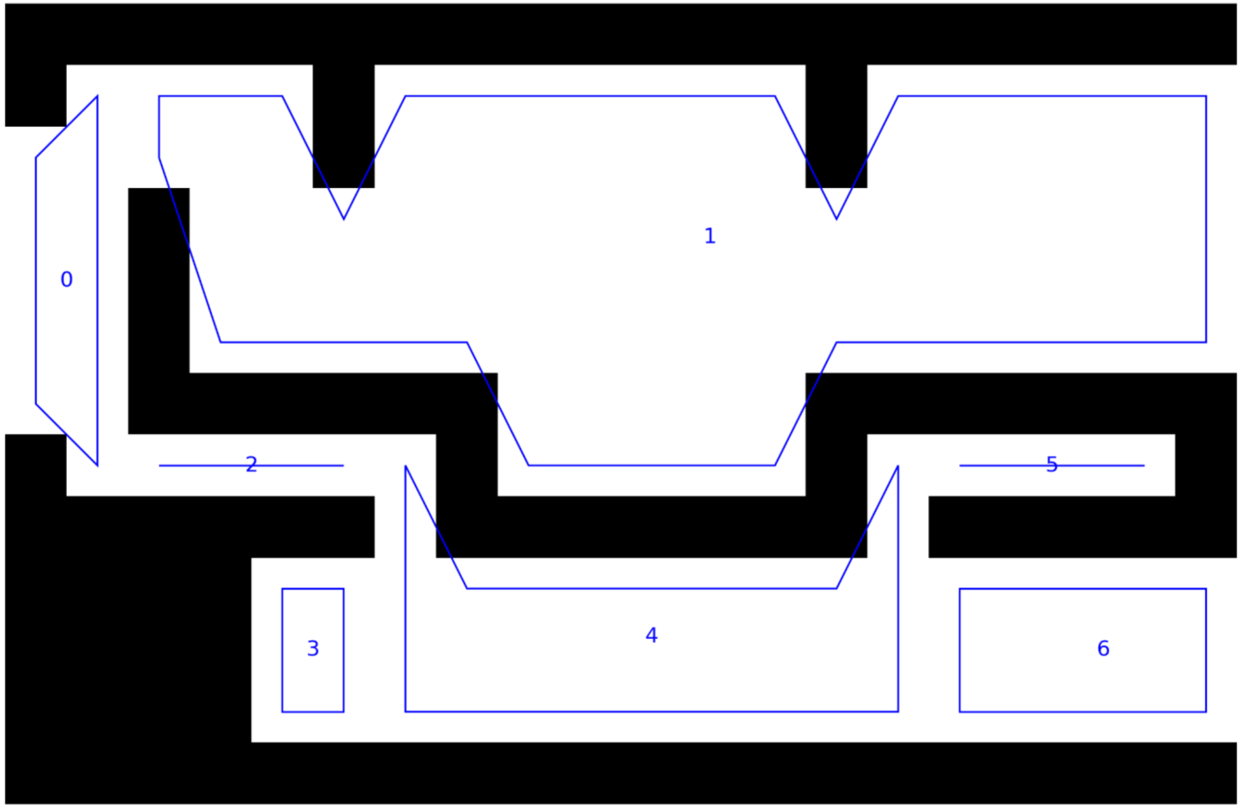
# Path Planning

*— Create Full Coverage Path —*

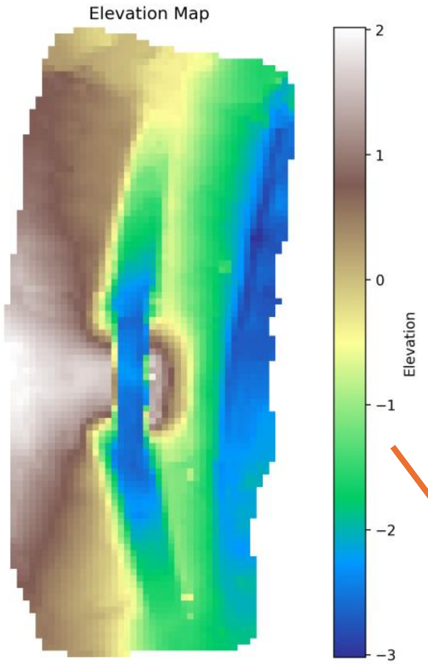
- Created the full **path planning pipeline**.



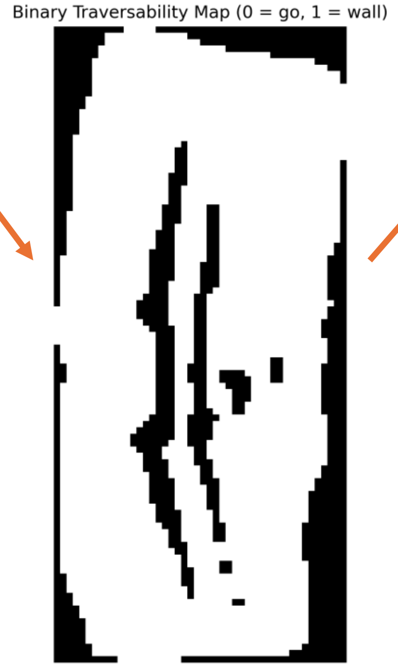



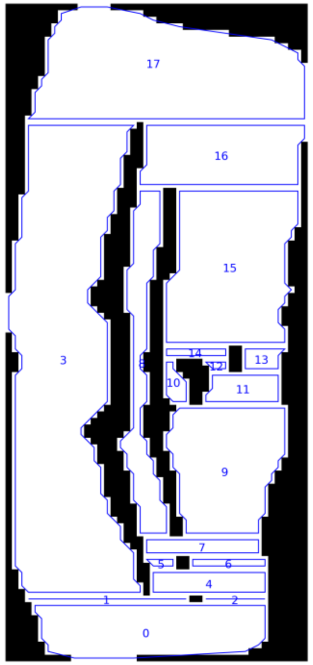
# Path Planning Pipeline



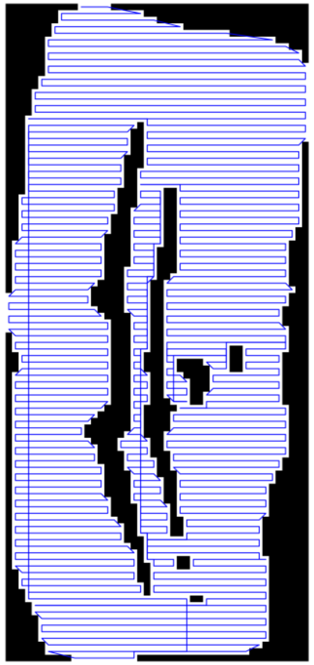

Elevation Map



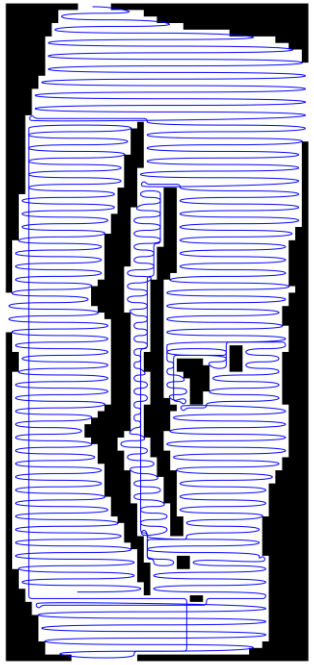

Binary Traversability Map




Boustrophedon Cellular Decomposition (BCD)



Coverage Path

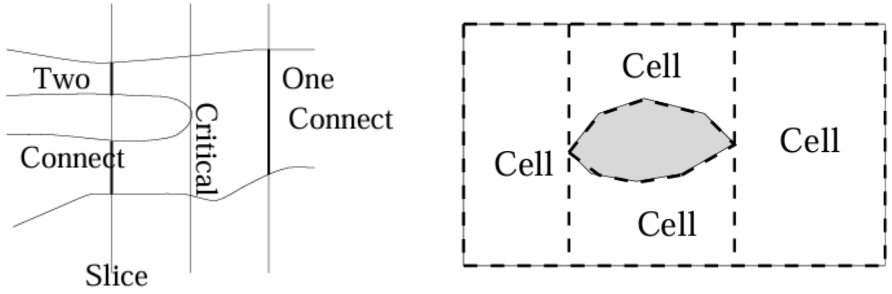


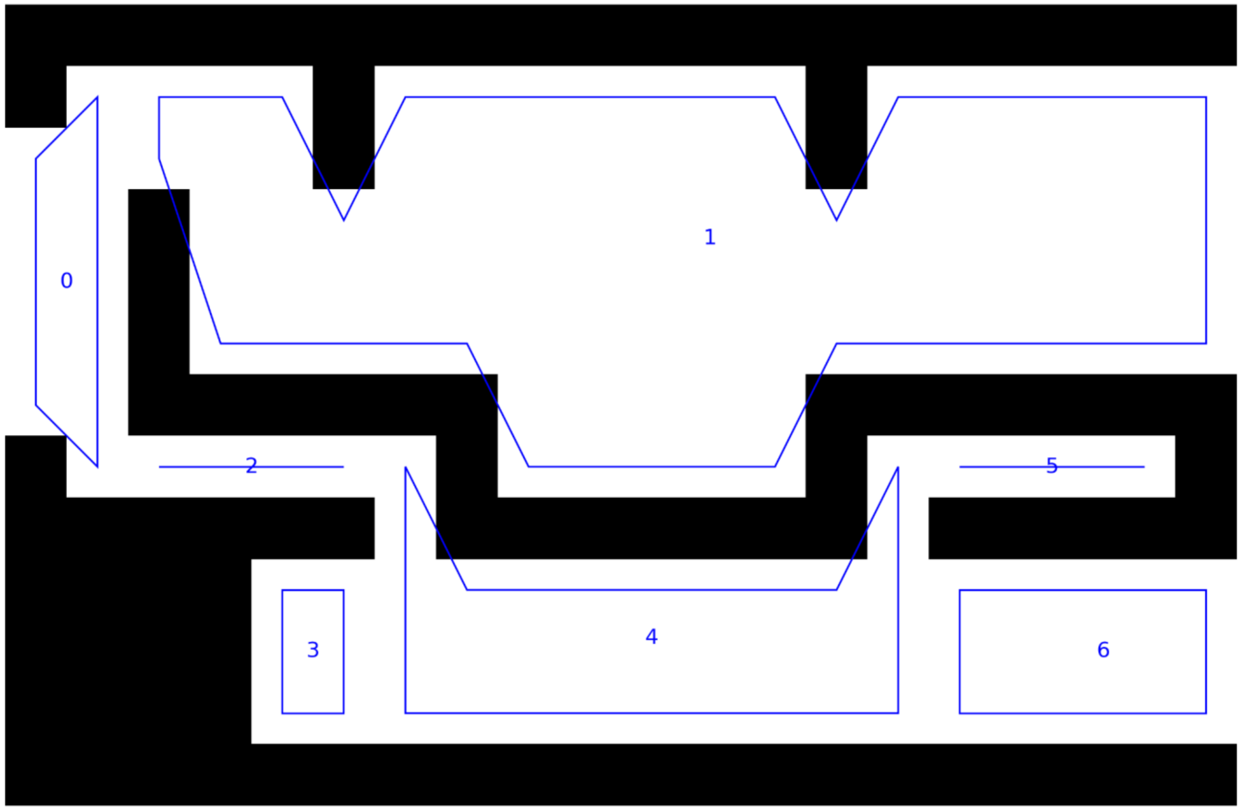
Smoothed Coverage Path (Chaikin Curve)

# Path Planning

*— Create Full Coverage Path —*

- Created the full **path planning pipeline**.
- The binary traversability maps were generated from elevation data and tuned based on the rover's mobility, including slope thresholds and slip.
- The **BCD step** divided the map into cells and created a traversal order by building an adjacency graph.
- The coverage path algorithm then created **lawnmower-style paths** inside the cells, which ensured complete area coverage.

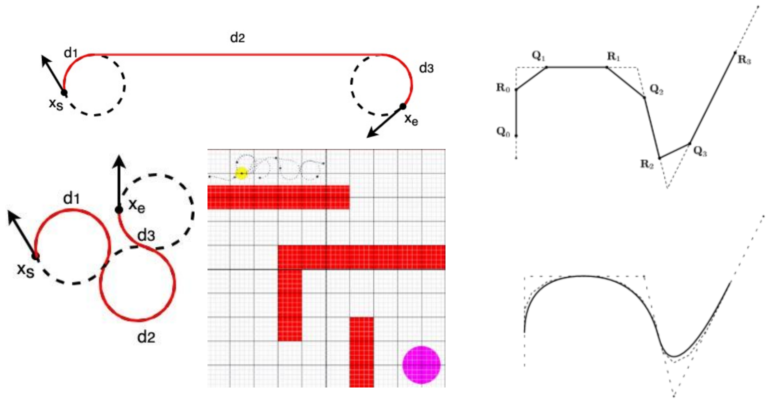


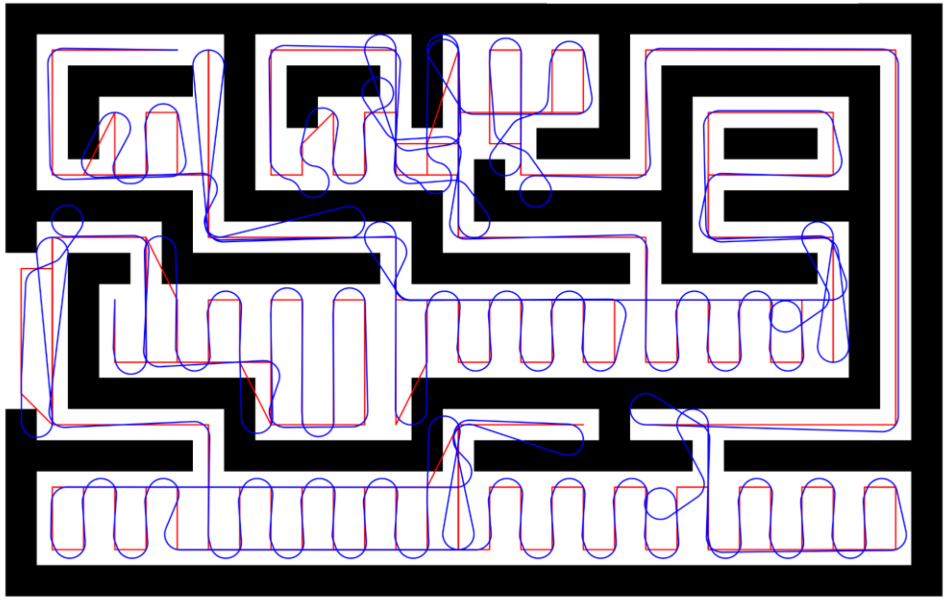



# Path Planning

*— Create Full Coverage Path —*

- Smoothing the coverage paths was one of the hardest problems I worked on.
- Implemented four different methods, including **Chaikin curve** smoothing, **Dubins paths** smoothing, **turning radius** smoothing, and **add points** smoothing.
- Findings:
  - Add points method is fine most of the time and works with the rover control.
  - Dubins path tends to work the best with rover constraints like minimum turning radius if smoothing is desired.
  - Chaikin curve removes 180° degree turns while staying close to original path.
  - Turning radius method offers no significant advantages over Dubins path.
- Recommendation: **Use add points for our coverage paths.**

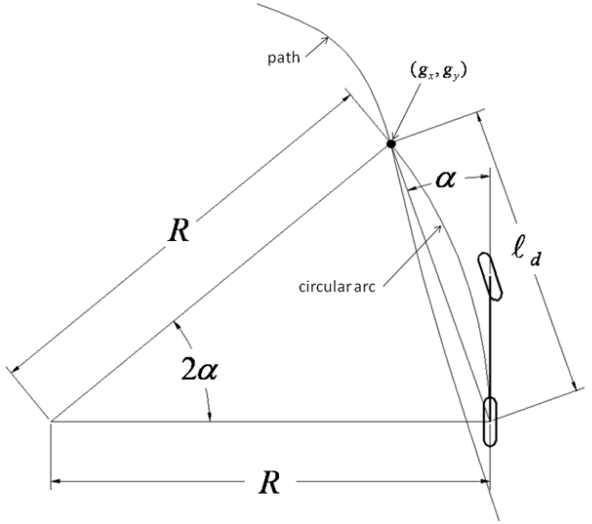


# Rover Control

*— Manual and Autonomous Area Traversal —*

- Wrote code for both manual and autonomous rover control.
- For **manual control**, implemented skid-steer driving with a PS4 controller, supporting both joystick and button modes for flexibility.
- For **autonomous control**, tested several algorithms including Stanley control and LQR, and decided to adapt the pure pursuit algorithm to work with skid-steer kinematics.
- The Jetson computer streamed position and heading data to the rover over TCP using netcat, keeping latency low and ensuring safe operation if the connection was lost.
- Although we could not fully test on the Astrobotic rover due to GPS and RealSense camera issues, **I validated the algorithm with simulated GPS inputs**.
- Finding: **Rover successfully traverses path even with steering imbalance error as large as 50% on one side.**




$$\kappa = \frac{2\sin(\alpha)}{\ell_d},\ V_L = 1 - \frac{\kappa W}{2},\ V_R = 1 + \frac{\kappa W}{2}$$

$\alpha$: heading error

$\ell_d$: lookahead distance

$\kappa$: curvature

$W$: track width

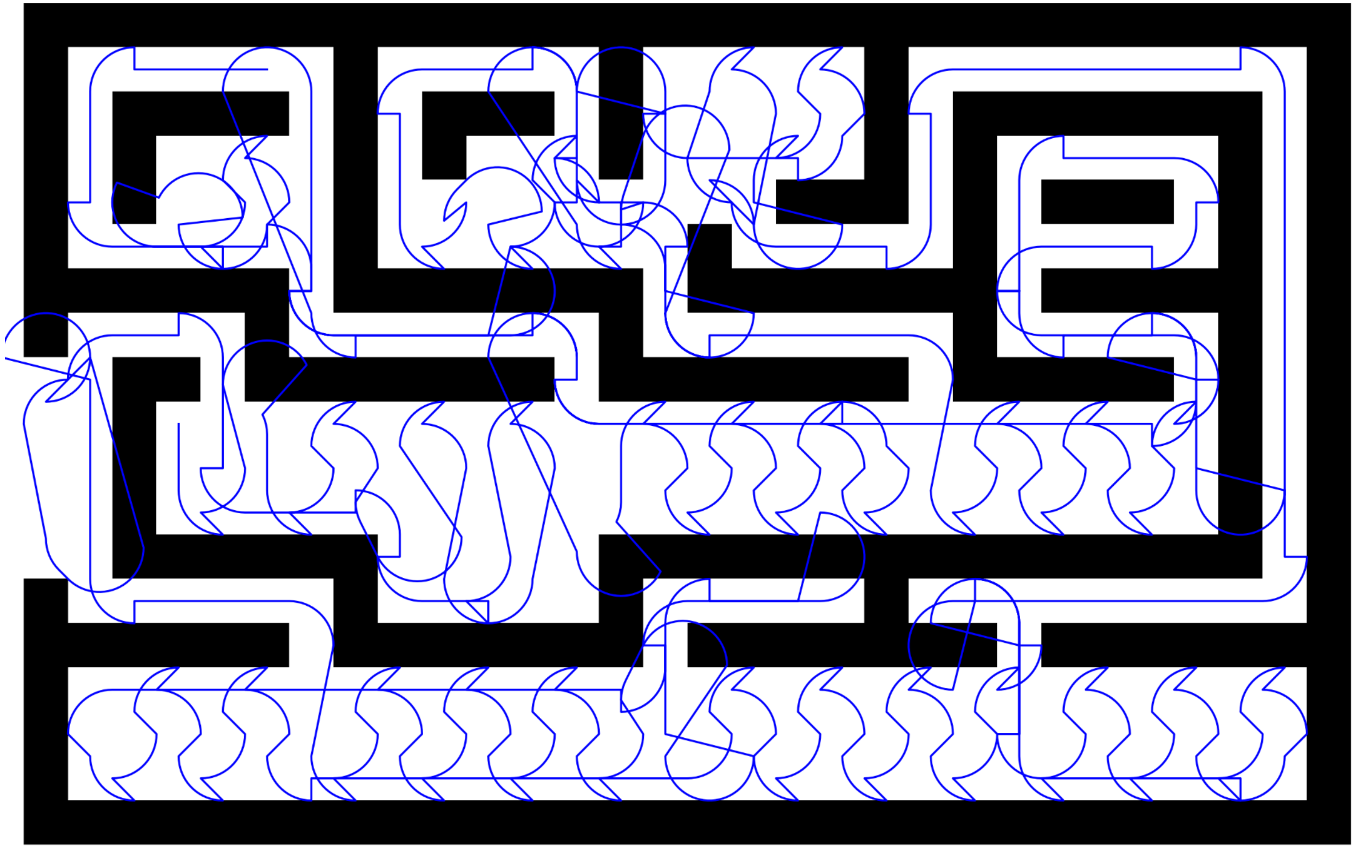
$V_L$: left wheel velocity

$V_R$: right wheel velocity

# Debugging and Issues

- Issue with BCD cells and their cell vertex lists broke my coverage path code.
- Smoothing algorithms went haywire and were hard to work with.
- There were localization problems with the rover.
  - The GPS was inaccurate.
  - The RealSense heading output didn't match the standard convention, which caused problems with the frames.

➢ Through these problems, I learned the importance of debugging early and documenting my results better.



# Code Output

- PLY meshes and elevation maps of the UW Golf Course and Urban Canyon.
- Complete code for the Astrobotic rover, including:
    - Mapping: ply_to_elevation_map.py, elevation_map_to_ply.py, etc.
    - Path Planning: cellular_decomposition.py, coverage_path.py, smooth_dubins_path.py, etc.
    - Rover Control: controller_skid_steer.py, path_tracking_position.py, gps_data_to_rover.py, etc.
- The full code is available on GitHub: [alexmosci/nasa-dcgr-astrobotic-rover](alexmosci/nasa-dcgr-astrobotic-rover).