# University of Hawaii Hilo

NASA Swarmathon Virtual Competition

# Team Vulkan

## Virtual Competition Technical Report

Advisor: Marc Roberts

Student Lead \ software support: Will Barden

Developer \ Tester: Alec Goodson

## Abstract:

This being our first foray into the NASA swarmathon competition, we focused on building an algorithm that will be iterated on in future competitions. The only file modified was mobility.cpp. Multiple search and return patterns were considered for implementation, one in particular being either a golden spiral or Fibonacci spiral, however, this introduces complications in logic, being heavily reliant upon initial placement and conditions. Instead, we utilized a sensor driven search algorithm. Our rover will first travel in a straight line until it detects a #0 block. Should multiple blocks be detected, the rover will position itself to grab one for definite retrieval, and push other blocks toward the collection point. This “herding” method should result in a gradual increase in collection efficiency as more blocks are detected and moved en masse to the collection point.

## Introduction:

The University of Hawaii at Hilo has never competed in the Swarmathon competition before, however our Space Robotics Club participated in the NASA Robotics Mining Competition last year. Due to our smaller student body we were unable to complete the coding portion for Swarmathon 2016. Thus our goal was to develop a working algorithm and get it into the swarmathon code for future competitions. By laying this foundation now future Vulkans from UH will be able to better compete at the national level. Including our advisor, our team was composed of only two students. We co-ordinated via email and met with the advisor on a weekly or biweekly basis. Tasks were partitioned between the two students involved, one student focusing on algorithmic development and coding, and the other student primarily working on software management, implementation, testing and troubleshooting. Tests were done on a school provided computer that met the minimum hardware requirements for simulation testing, and on student’s personal home computer with more capable hardware that allowed for more rapid simulation testing. While the random search pattern from the default mobility.cpp could vary wildly in response times based on initial conditions and luck, simulations with our algorithm got the time down to roughly 31.5 minutes with 12 blocks using the clustered pattern.

## Related papers:

Our code was developed largely by one student with software support from the other student team member. As such, outside papers weren't consulted, and much of our development arose out of trial and error.

## Methods:

Taken from pseudo-code and algorithm investigation:
When dropping off #0 at pad, when detecing #256, turn towards it.
When dropping off #0, back out until #256 is detected again. Turn 180° and resume search as rover is moving forward.

Ensure rover collects as many resources at once as is possible (pushing more blocks while holding one in claw)

-The rover can glitch when it detects #256 and thinks it needs to turn to point toward #256. It won't move forward. Make sure rover checks at least once, a few times max, turn, then moves onto mat.\

As rover is grabbing #0, stop or go forward for a moment, then back up, etc. to ensure that the block has been grabbed firmly.

When trying to pick up #0 at a roughly 45° angle, it can't pick the block up properly and continues to try over and over.
Attempt to get rover to turn around the block, then pick it up?
-When "seeing" two #0s side by side, the rover aimed between the two, and grabbed at nothing (this repeats). The rover should pick one of them, and not attempt to use the claw to grab both.

-While pushing multiple #0s toward the pad, most will sometimes get under the rover and either slow down the rover, or cause it to stop entirely. If the rover detects that it is not level, it should back out and try pushing the blocks, but only after it detects that it is again level. If the problem persists, the rover should circumvent the pushed pile and drop off the #0 it is currently holding.

Update made: changed if statement timerTimeElapsed >= {}
Was 5, now 3. For efficiency purposes

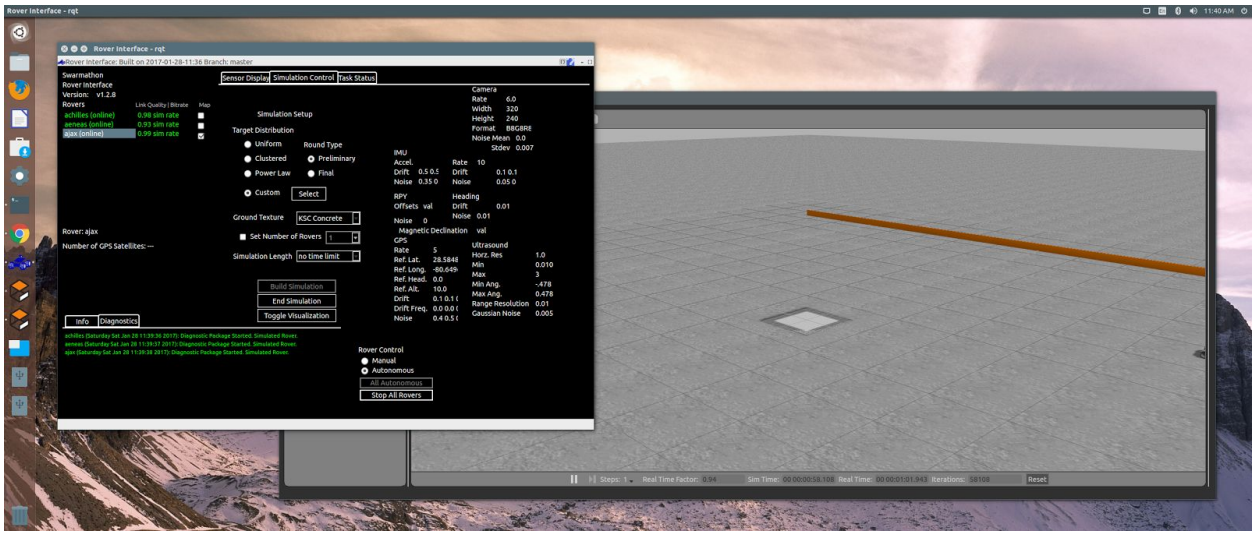
Send Drive Command (double, double); tells the rover to move/stop

o:o to stop, >= 0.0 to move forward, <= to move backwards. (0.0, a number) to rotate.

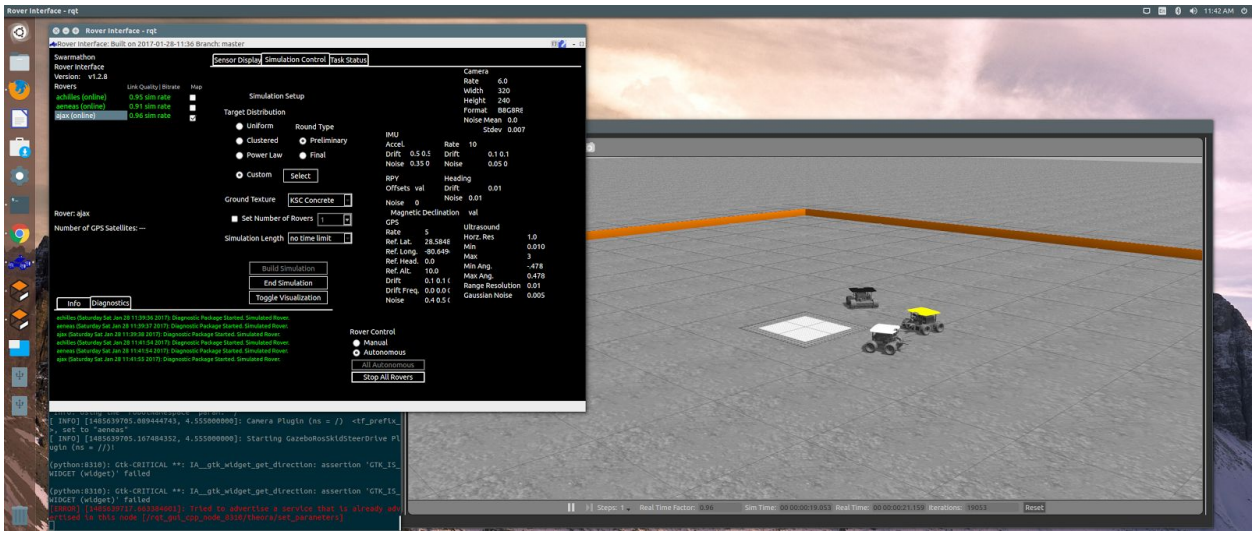
Result: Roughly 31.5 min real time

## Testing:

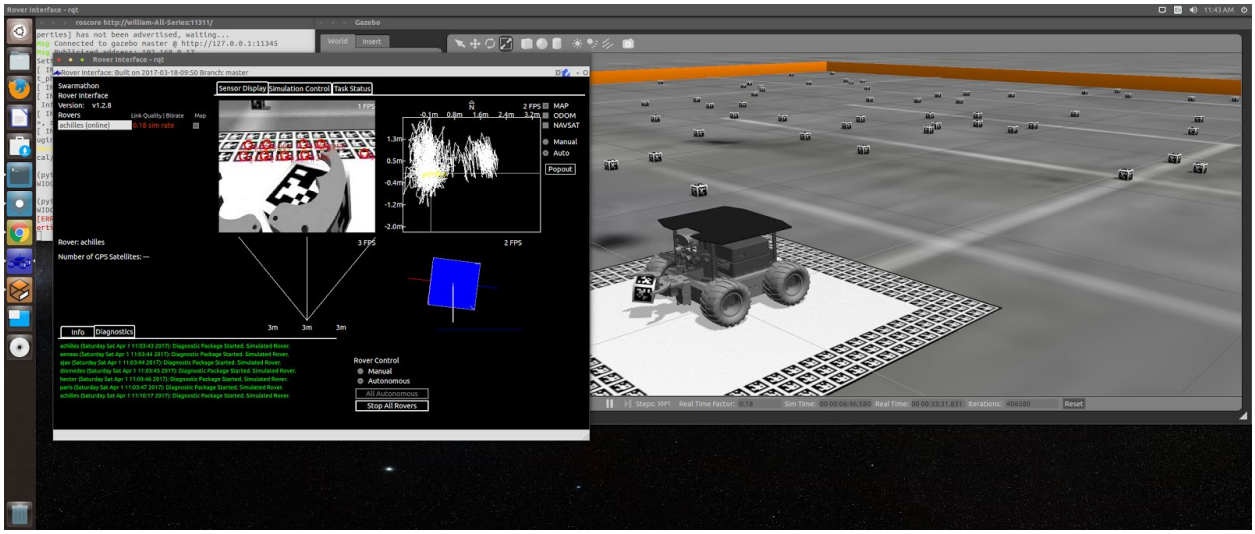
While our team was unable to produce a working non-random algorithm several tests were run by modifying the included mobility.cpp file. Specifically, several timeouts were modified, this had a small but noticeable effect on on the efficiency of certain actions. However, due to hardware and time constraints, simulations were run with an increased step count, which we believe may have lowered the accuracy of the simulation with several bugs appearing. Screenshots of the simulations can be seen below:




Simulations were mostly run with blocks placed in either a clustered or uniform pattern. We believe that a software update may have actually hindered our ability to run the simulation, as the sim rate above was inconsistent with previous tests, dropping from .9 to .18 on average.



Additional testing indicated that the number of rovers did not impact the sim rate, and frame rate was consistent across all tests. This lead us to believe that we had not run into a hardware constraint, but rather, a software constraint.



While poking at the algorithm, we simply let a single rover loose in a field of 256 block placed in a random pattern. In order for the test to run in a reasonable amount of time, the steps option was increased to 2500 for one test, and 3000 for subsequent tests.

After a period of approximately 15 minutes, the rover had located and targeted a block. After several failed attempts to grasp the block, the rover managed to snag it, and had begun the process of returning the block to the depot.

It was during the retrieval process that the rover ran into several problems. First, the rover dropped the block, something that we had not initially realized was a possibility during a

virtual run. The rover attempted to pick the block back up, and ended up running over the block several times before becoming distracted with a second block. It was this second block that the rover was able to return to the depot.

However, because the rover was not that “smart”, it seemed unable to determine where in the depot to deposit the block. The UI indicated that the rover could identify several QR codes simultaneously in the depot, and became confused, constantly turning to a drop the block in the middle of all the targets.

## Conclusion:

While our team suffered manpower losses that prevented us from delivering usable code, the two students that remained and our advisor were able to tinker through and map out the supplied code for future alteration. We foresee future Vulkan teams building on our foundation and improving our algorithm turning them from theory to reality. Most importantly, we hope that a real-world approach to a virtual search will have far more tangible results in the future. On mars, targets will follow rules not set by a random number generator, but by the forces of wind, erosion, and geology. An algorithm modeled around situational awareness potentially saves on time, which is the most precious resource any expedition, manned or otherwise, has to work with. While spiral search patterns may be mathematically efficient, unforeseen obstacles and environments require a robust algorithm that would allow a rover to change strategies as new data is acquired and key parameters are changed.