2012 NASA Robotics Academy

Autonomous Path Planning

Goddard Space Flight Center
Greenbelt, Maryland
June 4, 2012 – August 10, 2012

Principle Investigator: Thomas Flatley, Code 587
Principle Mentor: Jeffrey Hosler, Code 587
Developers: John Donahue, Julia Sichler

Abstract
Autonomous path planning in partially known or unknown environments is an important research topic. In the case of Europa and Titan, it is impossible to know exactly what is below the ice. For this reason, it is critical to have an adaptive planning system that can adjust to new sensor information. The objective of this project was to create a system to allow a rover to navigate through mars like terrain. Algorithms such as the Bug algorithm and the Ant Colony Optimization algorithm were simulated; however each had its own limitations. A combination of both algorithms has proven to be the best navigation system. At the conclusion of the summer, we were able to combine the previously researched image processing techniques, path planning systems, and current drive functions to allow the rover to navigate autonomously through the Mars yard, but it did not recognize that it had reached its goal due to errors in the PAL system.


Figure 1: The "Mars yard" – simulated Martian terrain

Table of Contents

1 INTRODUCTION

The purpose of autonomous path planning is to allow a robot to navigate through terrain relying solely on sensors built into the rover without human interference. In previous years, interns at NASA Goddard Space Flight Center have worked with the Personal Exploration Rover (PER) to develop an Adaptive Sensor Fleet to enable the rover to explore and collect data efficiently in various terrains. In order to accomplish autonomous driving the groups incorporated a compass into the rovers as well as a Precision Asset Location system. Our summer objective was to create an algorithm for the PER to navigate through a Mars-like terrain without any data regarding the environment.

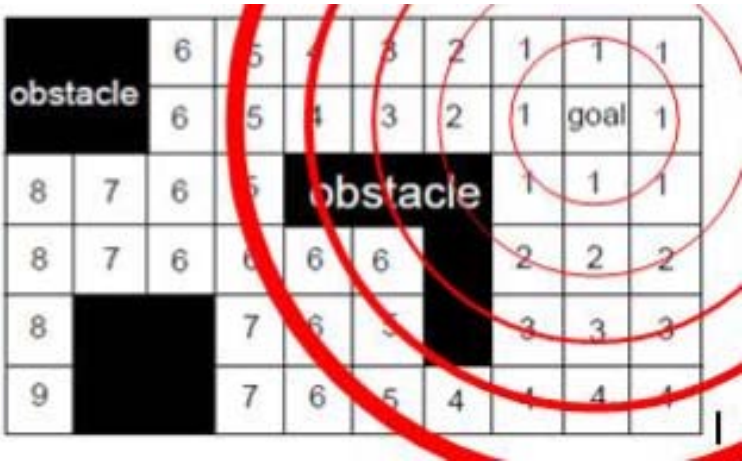
2 RESEARCH

Navigating through unknown terrain required research into path planning algorithms and strategies in order to find the best way to navigate autonomously.

2.1 Bug Algorithm

The bug algorithm draws a straight line from the initial position to the goal. This system works perfectly as long as there are no obstacles in way of the most direct route. If an obstacle is detected, the algorithm randomly chooses one direction to follow the side of the obstacle. When the obstacle is no longer present, the rover reverts back to the most direct route until it reaches the goal, or another obstacle. The disadvantage of this algorithm is that if the program guesses the wrong direction to follow an obstacle there can be a significant detour.

2.2 Wavefront Algorithm



The Wavefront Algorithm starts from an initial position and propagates outwards until it reaches the goal. It is called Wavefront because the steps appear to spread outward like a wave. The disadvantage of this algorithm is that the terrain and obstacles must be fully known.

Figure 2: Illustration of the Wavefront algorithm

2.3 A* and D* Algorithm

Are relative path planning techniques. The A* algorithm requires a known or partially known environment. Because of the necessary terrain knowledge the A* algorithm was too inefficient. A D* algorithm does not require a known environment. However, D* algorithms are very complicated (they were too difficult to conceptualize and code in one summer project). For these reasons neither the A* or D* algorithm were used.

2.4 Ant Colony Optimization (ACO) Algorithm

Ant colony optimization algorithms focus on using multiple units to explore a given area and move in the most direct path possible from start to finish. The algorithm does this by incorporating the idea of smell. When ants walk, they leave a quickly decaying chemical enzyme. When armies of ants walk from the nest to the food source they follow the strongest scent, which is the shortest path. Because our project only had one rover we could not use the idea of a "swarm" of robots to accomplish one goal, but using the idea of applying a numerical path, which was linearly increasing worked in our project. It was because of this that the rover was able to escape from completely concave and even one-way-in one-way-out obstacles.

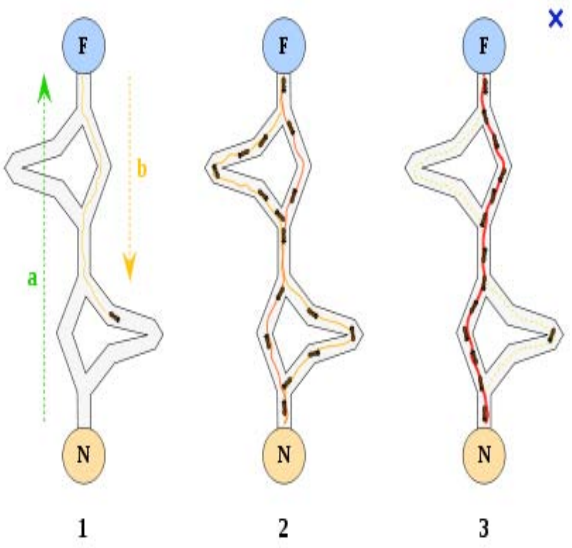


Figure 3: Illustration of ACO logic

2.5 Cost Function

Previous groups used the idea of assigning each possible step a certain cost value, and forcing the rover to go to either the highest or lowest option (depending on the design). We applied this concept of a cost function, and were consequently able to use a few of the algorithms above.

3 HARDWARE

Our project was plagued with hardware errors. Which was a positive thing in that we gained an extensive knowledge about the sensors, PAL system, and batteries; however it did delay our project. We have compiled many "pro tips" about the hardware so that the next group can spend more time path planning.

3.1 Rovers

The rovers we used were Personal Exploration Rovers (PERs) developed by the Robotics Institute at Carnegie Mellon University. The PER is a small, six-wheeled rover designed after the Mars Exploration Rovers (MERS), Spirit and Opportunity. The PER, an inexpensive, smaller version of the MERS, was implemented into museums across the country to educate the general population about the MERS. Firmware containing basic functions such as driving and turning was included with the PER and was written in Java. The PER uses a wireless card that connects to a preset router that uses a specific type of IP address. Any computer can also connect to that router and communicate with the robot using the included firmware or a command prompt terminal.

3.2 Sensors

The PER has several sensors to allow it to navigate through a simulated Mars terrain (Mars yard). Carnegie Mellon included a camera, an IR rangefinder, and an UV light. The IR rangefinder senses objects in front of the rover and determines the distance they are from the PER. The UV light was used to displayed signs of life within museum exhibits. Previous interns at NASA implemented a compass on the rear of the PER. The compass determines the direction of the rover and outputs the direction in degrees.

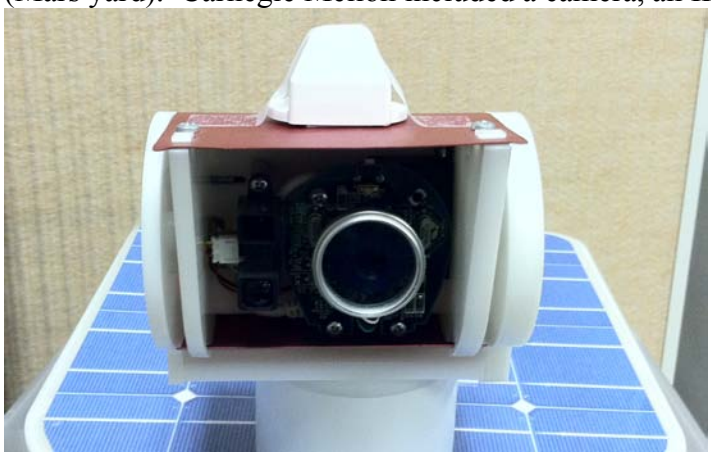


Figure 4: Picture of PER's sensors

3.3 PAL system

The Precision Asset Location (PAL) system is a positioning system created by Multispectral Solutions Inc. (MSSI). Tags placed on top of the rover, and at the final position create ultra wideband radio waves that are then sensed by receivers. Ultra wideband waves are short pulsating waves that send out packets of data. Four receivers in the corners of the mars yard use ultra wideband technology to detect these packets. The PAL hub interprets the packets by use of an optimization algorithm with the aid of a reference tag, located atop of one of the lampposts in the courtyard. The hub is able to identify outliers and averages the data to create a precise location of a tag. A new system is able to determine a location within one foot of the actual location. The hub is able to determine an accurate location of a tag if it is seen by at least three of the four receivers. Otherwise, it will determine a "presence" for the tag, but will not be able to read its location. With the aid of client software, we were able to extract the positions of the tags to use within the program.

Due to the age and conditions required by the hardware, we had difficulties using the system to its fullest extent. The age caused the system to have a two to five foot margin of error, instead of only one. Also, one of the receivers was out of commission, requiring all the remaining receivers to be able to see a tag in order for the hub to determine the location. Because of this, we were unable to see the entire Mars yard, and were limited to using only the visible area.

Instructions on the PAL system:

1. Place the reference tag on top of the lamppost in the center of the courtyard closes to the Mars yard.
   It doesn't really matter if you have the reference tag out before turning on the system, but if the tag is already out, it will take less time for the system to begin giving accurate data.
2. Turn on the router by turning on the power strip.
3. Turn on the PAL hub via the switch on the back left side.
4. Connect your computer to the "rovernet2" Wi-Fi.
5. Go into your favorite Internet browser and type in "192.168.2.100".
   This will pull up an interface for the PAL system.
6. Click on "demo". **Allow the JavaScript to run.**
7. Go into "areas" and click on "B23 courtyard".

The map will display any tag in the courtyard. In the GUI, you can control what you want to see on the map.
When you are done with the PAL system, or simply want to check something on the Internet:

1. TERMINATE ALL PROGRAMS, AND CLOSE THE INTERFACE.
2. Disconnect from "rovernet2".
3. Turn off the PAL hub and router if you are done for the day.

3.4 Compass

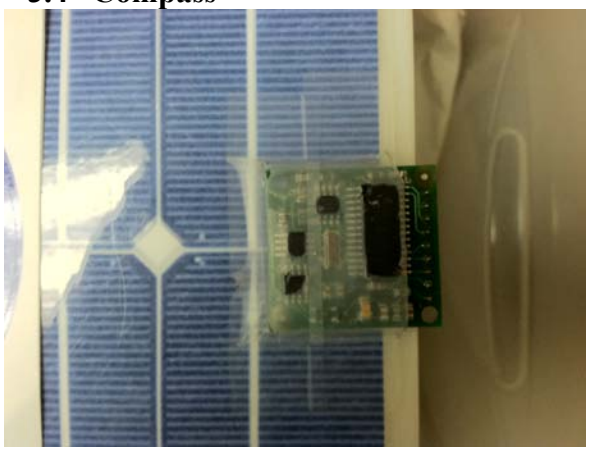


Figure 5: Picture of compass

The compass outputs units that are in degrees; however a compass has a different system than the unit circle. On a compass the units increase as the needle rotates clockwise, whereas a unit circle increases in the counterclockwise direction. We created a system that converts the compass degree value to its equivalent on the unit circle. This was done by breaking down the four quadrants then converting the compass value. The conversion was necessary because Java's trig libraries use the unit circle.

4 SOFTWARE

The software we were able to write works in the Mars. Difficulties arise from errors in the sensors. While reading other articles about autonomous path planning it appears that relying on sensor data is a very common complaint among roboticists. Our task was to plan a path in unknown terrain, so we used a combination of the Bug algorithm and the Ant Colony Optimization algorithm. In each step the rover takes into consideration: how far the next step is from the final destination, if there is an obstacle, and if the rover has explored that area (smell). All of these inputs are added to a total cost. The rover chooses its next destination based on the lowest possible cost.

4.1 Maze Function

In order to learn more about the PER and gain more experience programming in Java, we created a maze for the rover to navigate through. Paper boxes were set up for the rovers to encounter and therefore avoid. The predictability of the maze aided us in finding the errors within the program and to correct them. This enabled us to practice Java syntax and to use the various pre-programmed actions within the firmware. This program became the basis for the obstacle avoidance portion of our final program.

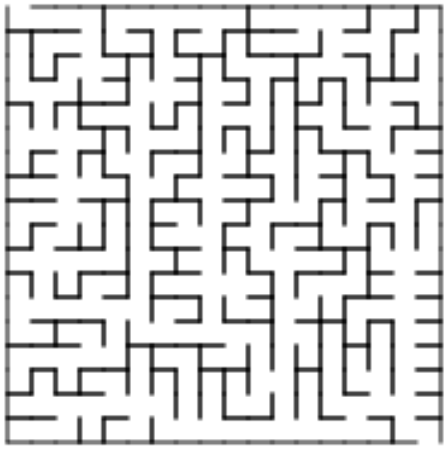


Figure 6: An illustration of a maze

4.2 RoverCerebellum

The Cerebellum instructs the rover to do comparatively simple mechanical techniques. The cerebellum is where the program begins. It first aligns the rover in the direction of the final destination. Then, the rover drives forward while scanning the horizon from negative thirty to thirty degrees at ten-degree increments. If there is an obstacle detected the location is stored in the Cerebrum. This is done so that these obstacles can be accessed later. If the rover encounters an object in its path, it switches over to the cerebrum.

4.3 RoverCerebrum

The cerebrum decides the more “thought provoking” decisions for the rover. For instance, if an obstacle is in-between the rover and the goal, it will consider which path to take using the algorithm specified in section 2.5. If the rover’s Cerebellum senses rocks that are near but not in the way, their location is stored in the Cerebrum. This creates a map of known obstacles, which is stored in the Cerebrum. Because of the Cerebrum’s

larger processing requirements, it is not faster than the Cerebellum. For this reason, the Cerebrum is only used when the rover is encountering challenging terrain.

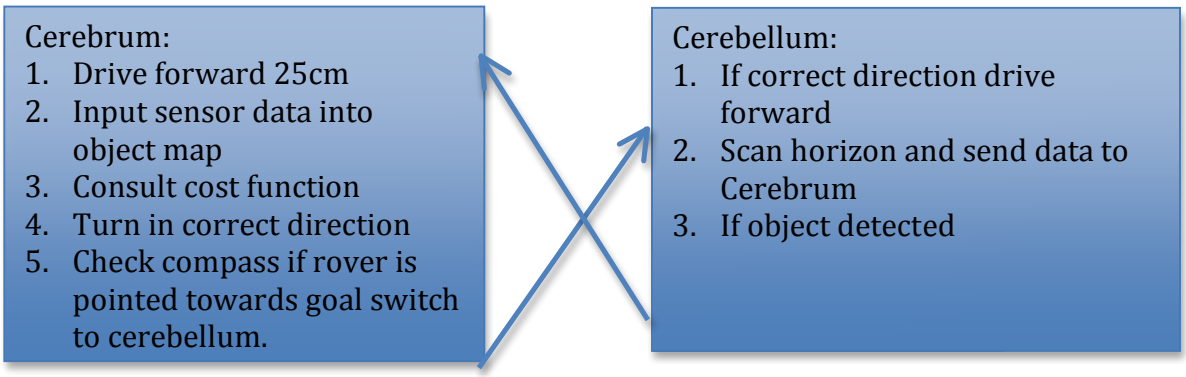


Figure 7: Diagram of Rovers logic system

5 TROUBLE SHOOTING

There are a lot of things that can go wrong. We created a list of the most common things that have happened to us, and how to fix them (or at least reduce their impact).

5.1 Calibration

When calibrating the rover:

1. Open eclipse and run calibration
2. Press the "load servo calibration rover" button. Now, you can make adjustments to the robots head angle, tilt, turning power, and distance travelled forward.
3. Make sure that there aren't unwanted negative signs before the values. Sometimes these won't be displayed, but they are there. So **if your rover starts going backwards** for some unknown reason check to make sure unwanted negatives are not in the calibration data

5.2 Sensor Error

The sensors often gave data that was either slightly incorrect, or very incorrect. We alleviated this problem by running a for-loop and taking the average of multiple sets of data. Previous groups used Kalman filters, but they proved too difficult for our summer project. Our simple averaging increased the accuracy of the compass, PAL, and IR data significantly.

5.3 PAL errors

When using the PAL, we recommend using it in the morning and whenever the sun is not on the receivers. If there are no tags on the map, check the "raw data". If you only get presence data, there is sun on the receivers and it will take time before you can use the system.

When you encounter "Error: no valid reference" in the raw data, **check that the reference tag is straight** on the lamppost. If it is, and you still have the error:

1. Close the interface.
2. **Terminate any program** that is using the PAL data. (Seriously, otherwise, the PAL will not work after you reboot it.)
3. Disconnect from "rovernet2".
4. Turn off the PAL system, then the router.
5. Wait for a few seconds, and then turn them back on.
6. Reconnect to "rovernet2".
7. Open the interface and check the raw data.

If the problem continues, there is most likely sun on one or multiple receivers, in which case you should go work on something else and by the time you're done, the sun will have moved. If you don't believe there is sun on any receiver, one of the receivers may be giving bad data. Go into the configure menu, and then into the third tab. Check that the receivers are sending data for the tags. There will be a column for each active receiver, and if one appears to have straight zeros, you might have a bad receiver. If this is the case, **talk to your mentor.** If the PAL mysteriously stops working or closes your connection, try rebooting the hub after closing any program using it.

6 FUTURE WORK

Future groups should start by working on filters so that they can get better data from the sensors. This includes the IR range finder, the compass, and the PAL system. Our code was based on the assumption that all of these things would give relatively accurate and precise data, however there was a wide range in the data collected and a large margin of error.

7 CONCLUSION

We were able to successfully create an algorithm for a PER to navigate through a Mars yard autonomously. The final program consisted of two Java classes, a basic path planning system and an obstacle avoidance system. The path planning system was able to incorporate the Bug and Ant Colony Optimization algorithms to become the "Cerebellum" of our program. Information from the PAL system and the compass enabled the system to determine the best direction for the rover to travel. The obstacle avoidance system, or "Cerebrum", was able to guide the rover around an obstacle and move back to the path planning system. The path planning system continued until the rover was within two feet of the destination, due to a margin of error for the PAL system.

References:

http://www.cs.cmu.edu/~myrover/PER/
Sapphire DART (Model H651) User's Guide pg. 6-8