NASA EZ-RASSOR 2.0

Black Team

John Albury, Shelby Basco, John Hacker, Michael Jimenez, Scott Scalera

Department of Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — EZ-RASSOR is a ROS-based software suite for controlling regolith-mining rovers, such as the Mini-RASSOR: a low-cost robot meant for use in education and research. In this work, autonomous navigation for GPS-denied environments is added to EZ-RASSOR. Two methods for absolute localization, one using images of celestial bodies and the other using digital elevation models, are combined with relative localization methods. With a derived location estimate, the robot is able to navigate using a modified wedge bug algorithm. Since no maps are used, a novel object detection algorithm is used to safely avoid obstructions both above and below the ground. Functionality is tested in a simulated moon-like environment in the Gazebo simulator.**

***Index Terms* — Robotics, artificial intelligence, autonomy, planetary rovers, simulation, localization, obstacle detection, path planning.**

I. Introduction

This work aims to build on the foundation provided by the first iteration of the EZ-RASSOR project. EZ-RASSOR is a software suite designed for use on NASA's Mini-RASSOR rover, a scaled-down version of the RASSOR rover. The team that previously worked on EZ-RASSOR was a UCF Computer Science Senior Design team in Spring 2019. This team was involved in creating computer models of the Mini-RASSOR and a moon-like environment in the Gazebo simulator, building the controls and communications functionality of the rover using ROS, and implementing basic autonomy. This work aims to further develop EZ-RASSOR by adding GPS-denied navigation and improving the autonomy of the rover. The goal of this work is for the rover to be able to safely navigate autonomously from a starting location to a given target location without the aid of GPS in a simulated moon-like environment.

II. Overview of Approach

In order to navigate to a desired location, the rover must first know where it is. Determining location is also known as localization. Without the aid of GPS, there is not a simple solution to this problem. Thus, the localization component was divided into two parts: absolute localization and relative localization. Absolute localization is determining the location of the rover in a globally-known environment. With this type of localization, the rover should have the ability to be placed at any spot without prior knowledge of where it is and determine its location. GPS is a common method for absolute localization on Earth. Relative localization is determining the location of the rover relative to a starting point and involves measuring how far the rover has moved from the starting location. An example of relative localization is using the motion sensor data from a cell phone to determine how far it has moved from some initial location. After computing both, the location estimates from absolute localization and relative localization can be combined to create one estimate for the current location of the rover.

Since the robot is autonomously navigating a poorly mapped environment, it needs to safely, dynamically avoid obstacles along the way. In order to do that, there must be some method of detecting obstacles on the rover's path. The obstacle detection subsystem is responsible for identifying obstructions in front of the rover based on the available sensor data.

Once the rover knows approximately where it is and what obstacles are ahead, this information can be combined to determine the direction and distance the rover should move to reach a destination. The path planning subsystem aims to minimize the distance traveled to reach a target location while safely avoiding obstacles along the way.

Fig. 1 summarizes the high-level design of EZ-RASSOR. The following sections will provide a more in-depth overview of each component of the system.

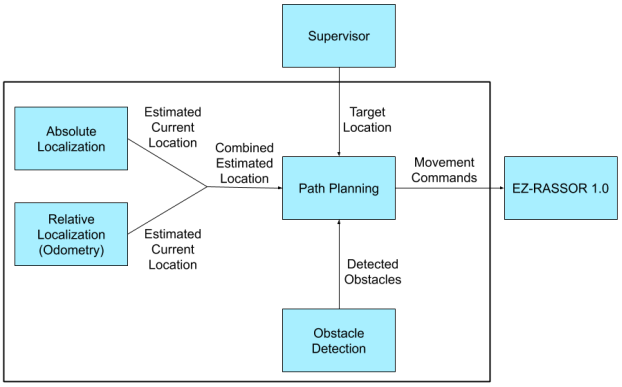

Fig. 1. A high-level overview of the technical approach. The enclosed portion of the diagram is what will be described here.

III. Relative Localization: Odometry

Odometry is the process of measuring the movement of an object. Here, odometry is used to measure how far the rover has moved from some known starting location.

Due to the conditions on the moon, high-quality odometry is difficult to implement. The sediment on the surface of the moon causes high wheel slippage, which makes measuring wheel spin prone to error. Since the surface of the moon is mostly the same color, tracking the movement of visual features is difficult. These conditions necessitated the use of a combination of odometry techniques to achieve a high-quality odometry estimate.

A. Approach

To implement odometry, three methods of odometry were combined using an extended Kalman filter: wheel odometry, visual odometry, and Inertial Measurement Unit (IMU) odometry.

Kalman filtering is a mathematical technique for combining different measurements of the same variable, where each measurement has its own inaccuracies, to produce an estimate of the variable that tends to be more accurate than any single input measurement [1]. In this case, the variable being measured and estimated is the rover's pose: its position and orientation. There are many open source implementations of Kalman filters available for ROS; the *robot_localization* package[1] is used because it supports an unlimited number of odometry sources and supports multiple ROS message types. The implemented Kalman filter subscribes to data from wheel odometry, visual odometry, and IMU odometry, then produces a combined pose estimate for use in path planning.

Wheel odometry was performed based on the Twist commands, which contain a desired linear and rotational velocity used to control the wheel motors. The computations for wheel odometry were done automatically by the differential drive controller.

For IMU odometry, IMU messages are sent directly to the implemented Kalman filter. The *GazeboRosImu* plugin[2] simulates the IMU readings, with added noise to more accurately simulate real-world performance.

For visual odometry, RGB-D images—images that contain red, green, blue and depth channels—produced by the rover's onboard depth camera are used. Because the moon lacks visual features, this extra depth channel was crucial for tracking features in the environment when performing visual odometry. The RGB-D visual odometry node of the *RTAB-Map* ROS package[3] is used to implement visual odometry. This node uses the GoodFeaturesToTrack algorithm for detecting features in each frame, optical flow for matching features between frames, and RANSAC for motion prediction based on the matched features between frames [2].

B. Results

In order to test odometry, the accuracy of the odometry system at various points along a path in a simulated moon environment is measured. The environment is meant to accurately represent a typical path a rover might take to reach a dig site. The testing environment has high-slippage and uneven terrain, low gravity, a large boulder, and a crater that the rover has to maneuver around to reach its destination. The overall distance to the destination is approximately 50 meters. The results of the performance tests are shown in Fig. 2.

[1]http://wiki.ros.org/robot_localization

[2]http://gazebosim.org/tutorials?tut=ros_gzplugins#IMU(GazeboRosImu)

[3] http://wiki.ros.org/rtabmap_ros#rgbd_odometry

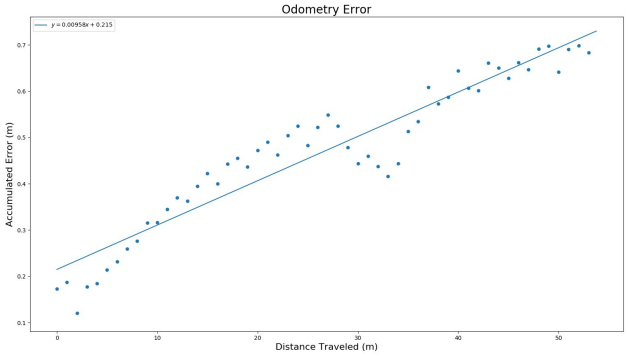

Fig. 2. Accumulated odometry error as a function of distance traveled.

The odometry system has slight error (approximately 0.2 meters) even before the rover has started moving, but the error only grows at a rate of approximately 0.01 meters per meter traveled. This means that, given a known starting location, the rover could likely travel hundreds of meters before the odometry system's accumulated error surpasses 25 meters—the requirement set for the localization components of EZ-RASSOR.

IV. Absolute Localization: Cosmic GPS

Cosmic GPS is an absolute localization method that utilizes stars, or any celestial body, for the purpose of determining a planetary position. The goal of the method is to gain a coarse position estimate that can be fed into a more precise absolute localization method, such as Park Ranger, for a finer position estimation.

A. Approach

Cosmic GPS is a system composed of a hardware and a software component.

The physical system on the robot for obtaining images of the sky has yet to be completed, as it is outside the scope of the current work, but the software component of Cosmic GPS follows two assumptions. The first assumption is that the imaging system naturally points in the direction of the robot's zenith—the point directly above in the sky. The second assumption is that the camera is capable of capturing images of stars without false positives. That is, the star shapes in the image should be from actual stars and not hot pixels or other noise.

The software component is best thought of as a pipeline with four stages: image processing, star templating, star identification and matching, and position derivation.

The image processing stage is the shortest and involves preparing the image for star templating by reducing the effect of the random physical processes that occur during image capture and readout. For some image systems, this may require dark frame reduction and frame stacking. Otherwise, only anisotropic diffusion is needed to smooth the image to reduce noise.

Using the processed image, the star templating stage identifies the parts of the image corresponding to individual stars. This is done by first templating the image using a thresholding method to record all of the "active" pixels, then clustering these pixels into "bags" based on their connectedness. In other words, "active" pixels are associated with the same "bag" if they combine to form a contiguous area in the image. These "bags" are the stars in the image. The next step is to order these stars based on intensity and measure their angular distance from the center (zenith) of the image and determine their angle from the bearing of the image.

With the stars in the image isolated, the star identification and matching stage determines the stars' identities. This is possible because the positions of stars in the sky stay relatively fixed in relation to each other over long periods of time. This allows for the use of a star reference graph that represents 286 of the brightest stars, by apparent magnitude, as nodes and the angular distance between the stars as edges. Then, using the five brightest stars in the image, the stars' inter-angular distances are calculated—using the law of cosines and the measured angles from the previous stage—and matched with the named stars in the star reference graph using breadth-first search.

The position derivation stage involves deriving the robot's celestial position based on the celestial positions of the identified stars and their respective positions relative to the center of the image. Specifically, this is done by taking one of three identified stars in the image and calculating a circle of celestial positions around the stars' celestial position such that the points along the circle have an angular distance equal to the stars' measured distance from the center of the image. Then, these celestial positions on the circle are checked and the position most closely matching the angular distances calculated—from the centers of the three chosen identified stars to the center—in the image, is selected to correspond to the celestial position of the robot. With this celestial position, the geographic position is then determined by using the time of capture and Greenwich Hour Angle (GHA) of Aries tabulated data.

B. Results

Taking into consideration the importance of the quality of the hardware component for the accuracy and precision of Cosmic GPS, the tests showed the software performed adequately. Fig. 3 shows the results of one such test with an error of 47.4 miles.

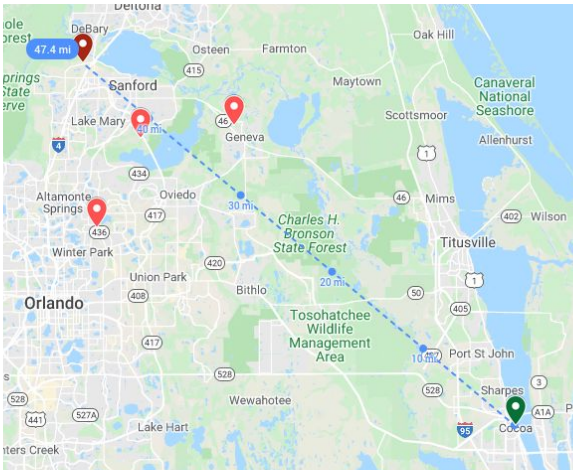

Fig. 3. The green tag marks the actual location the image was taken from. The red tags mark estimated locations.

The estimate produced during this test was within one degree of latitude and longitude from the actual position.

V. Absolute Localization: Park Ranger

Park Ranger is a term that refers to comparing the robot's surroundings environment to a digital elevation model (DEM) of the area. This method was developed to attempt to localize within a known map of the environment to provide an absolute estimate of the robot's position.

A. Approach

Particle filters are a family of algorithms, where each particle represents a position that the robot could be in. Each particle contains a weight that represents how likely it is the robot is at that position, with higher weights representing a higher likelihood. The particles are initialized such that each position on the map is equally likely, then the weights and positions of the particles are updated according to sensor measurements until the positions of the particles converge. The weights of the particles are updated by comparing a point cloud representing the surrounding environment to a known DEM. The DEM is a grid of values, where the value in each cell represents the height of the ground at the area that cell represents. Park Ranger aims to find the cell in the grid where the robot is most likely to be, so a discrete particle filter is used [3]. A discrete particle filter is different from the standard continuous particle filter in that it uses discrete integer coordinates instead of continuous floating-point coordinates.

The main steps of the particle filter implementation are as follows: initialize, predict, update, estimate, and resample.

First, the initialization step generates particles at each position of the DEM, and each particle's weight is set to $\frac{1}{n^2}$, where n is both the height and width of an $n \times n$ DEM. The likelihood function is then called between the initialize and predict steps.

The likelihood function compares a *local DEM* and a *predicted DEM* for each particle to obtain a score; the particle's weight is then updated by multiplying it by the score. The most recent point cloud is converted to a top-down view to obtain the *local DEM*, and the *predicted DEM* is a section of the DEM corresponding to what the robot would expect to see if it were at the position of the particle. The facing angle used when determining this section is the most recent heading estimate from the odometry subsystem. To obtain a score for each particle, the histogram comparison technique is used to measure the similarity between the *local DEM* and *predicted DEM*.

The second step is the prediction step, which updates the position of each particle by adding the displacement of the most recent odometry position measurement from the previous odometry position measurement. The heading of each particle is set to the most recent odometry heading estimate.

Next, the update step invokes the likelihood function to update the weights of the particles. Then, the estimation step gives an estimated cartesian position of the robot by calculating a weighted average for the x and y coordinates of the particles.

Finally, the resample step is only invoked if the *neff* (effective number of particles) is less than half of the current number of particles. The *neff* function takes in the normalized weights of the particles and returns the inverse of the summation of the normalized weights. If resampling is invoked, then residual resampling is used to determine which particles to keep and which to remove from the particle filter. Then, the weights of the particles that were kept are set to $\frac{1}{m}$, where m is the number particles after the resample step. Then, steps 2 through 5 are repeated. Once the particle filter converges to a single position, only the prediction step affects the estimate.

B. Results

Although a converging estimate was successfully derived, the accuracy and consistency of the estimate was less than ideal. The main factors that could be affecting the accuracy are: physical camera constraints, heightmap constraints, and simulation constraints.

The physical camera constraints adversely affect several components of EZ-RASSOR. The depth camera the Mini-RASSOR uses has a horizontal field of view of 74 degrees and is positioned low to the ground due to the rover's relatively small frame. This is likely the major contributor to the inaccuracy of Park Ranger because it bounds the number of usable points generated in the *local DEM*.

The heightmap constraints refer to the factors in creating the realistic moon terrain as a heightmap object in Gazebo. To create a heightmap, the choice of rasters (i.e. .jpg, .png, .tif) can affect the appearance and position of the terrain object. The ratio of the raster dimensions to the range of elevation values can also affect the terrain.

The last factor that could be affecting the accuracy of the estimate is having to test in a simulated environment. There is a noticeable latency between estimate measurements due to computation power being dedicated to rendering the simulation.

The major ways these adverse effects could be reduced are: placing the camera higher up, increasing the field of view by adding more cameras or using a different camera, and testing Park Ranger outside of simulation.

Fig. 4 shows a visualization of the weight for each position after the likelihood step.

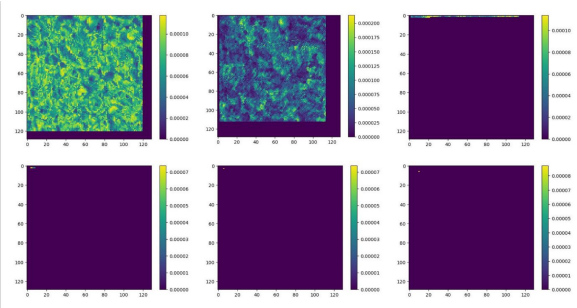

Fig. 4. For five iterations of Park Ranger, this shows the weight value for each position.

This shows that the particles start converging quite fast after even three iterations, sometimes on a position that is in a different quadrant than the actual position of the rover. Although the accuracy and consistency of the estimate could be improved through the previously mentioned methods, the Park Ranger component could be scoped itself as an independent long-term research project. Implementing this type of localization in a moon-like environment and on a robot with such a limited field of view as well as a low camera height, is a relatively novel challenge.

VI. Obstacle Detection

Obstacle Detection is necessary for safely traveling to a destination. It is especially important given that the path planning algorithm does not make use of a global map and relies solely on local data.

The Mini-RASSOR is intended to travel in moon-like environments. This means that the surface that it drives on is not flat; but, many obstacle detection algorithms assume the robot is traveling on flat ground. The challenge for obstacle detection on an uneven ground is to detect obstacles both above and below the ground without falsely detecting the ground itself as an obstacle. In the end, the obstacle detection component creates an array representing the direction of and distance of the closest obstacles in every direction the robot can see, also known as a laser scan.

A. Approach

To collect data for obstacle detection, a depth camera is used to capture the field of space in front of the robot. This data is represented by a point cloud, which follows a three-dimensional Cartesian coordinate system. These axes relate to the *right*, *down*, and *forward* directions, relative to the camera.

A calculated *distance* value for each point in the point cloud is key to the obstacle detection algorithm. This *distance* is based on the *forward* and *right* values, so that it represents the amount the robot must travel across the ground to reach that point. Then, the points are assigned a *step* value, which is a discrete angle from the line of sight of the camera [4]. The points are then grouped by their *step* value, as candidates for potential obstacles in each direction. Once the points are grouped by *step*, the groups are sorted by *distance* values. Each group iterated on, in ascending *distance* order, to find the first instance of an obstacle.

Shown in Equation 2 and Equation 3, obstacle indicators are calculated by the *hike* and *slope* values between two consecutive points in a group. Then, the *hike* and *slope* are compared to configurable thresholds based on the steepest incline and the narrowest hole to avoid. Because the use of *hike* values at larger distances can result in false positives, the maximum distance for detecting obstacles using *hike* is constrained to the

maximum distance considered for movement in the path planning. The *slope* value is intended to detect most above-ground obstacles based on the steepness of their exterior. The *hike* is intended to detect most below-ground obstacles—holes—based on how wide of a gap there is from ridge to ridge.

$$drop = down_{i+1} - down_i \quad (1)$$

$$hike = distance_{i+1} - distance_i \quad (2)$$

$$slope = \frac{drop}{hike} \quad (3)$$

Once the nearest above and below ground obstacles are individually detected, the minimum distance to each type of obstacle in every direction is stored in a laser scan. Finally, the laser scan completely describes the obstacles which the Mini-RASSOR faces.

B. Results

A simple test case is best to show the visual results of the obstacle detection algorithm. The simulated Mini-RASSOR is shown in Fig. 4 as looking at an above-ground cube to the left and a below-ground hole to the right. As seen in Fig. 5, a mostly accurate laser scan is created to detect the closest borders of both obstacles. However, it is clear that some of the hole's border near the corners is omitted from the scan. This is because either the ridges are close enough together to not be considered obstacles, or they are too far away to be considered by the path planning algorithm.

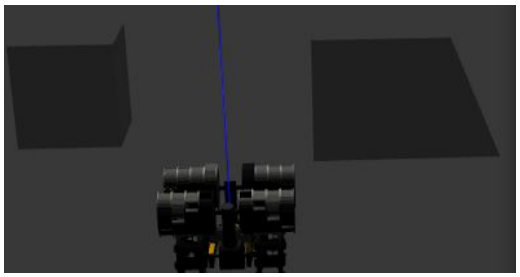

Fig. 5. Simulation of a box and hole in front of the Mini-RASSOR

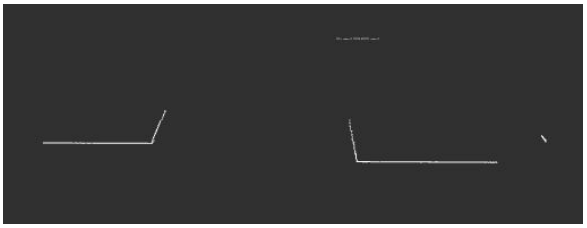

Fig. 6. Laser scan of detected obstacles in front of Mini-RASSOR

The obstacle detection algorithm is strong at identifying above-ground obstacles and is easily configurable. The closest point at which such an obstacle is unscalable by the robot is reliably reported in the laser scan.

The below-ground obstacles are slightly problematic with potentially an abundance of false positives. However, this is well minimized by configuring the hole-diamter threshold as large as is acceptable. This results in less of a hole being detected; but, the center, closest points of the hole can still be detected.

VII. Path Planning

Path Planning is a necessary component for the robot to safely and optimally navigate its environment. Because the Mini-RASSOR is meant to travel large, open spaces to reach a dig site, no maps were used when planning paths. All choices made by the robot when determining a path to take are determined using only information about the robot's local environment.

A. Approach

Since the robot is navigating using only information it can gather about its local environment, an algorithm that works well in unknown environments, wedge bug, is used. Wedge bug is inspired by the way insects navigate and uses sensors to navigate through a mapless environment [5]. The state of the robot's surroundings is represented by the laser scan created by the obstacle detection algorithm. Each laser scan contains information about the 74-degree window directly in front of the robot.

The robot navigates by moving a small, configurable distance, then stopping when it either travels this distance or encounters an obstacle in its way. Each time the robot stops moving, the path planning subsystem evaluates the most recent laser scan published by the obstacle detection subsystem. Then, using a heuristic function, the path planning algorithm determines the best direction to move toward. This function measures the difference between the robot's current facing angle and the angle to the

destination; the path planning algorithm aims to minimize this value. Additionally, a safety function is used to determine whether the robot could move safely in a direction for some configurable distance. The direction that has the lowest heuristic value and is safe is used. If none of the directions in the current wedge are safe to move towards, the robot turns to evaluate adjacent wedges until a safe direction is found. The path planning algorithm halts once the robot reaches its destination.

B. Results

The path planning algorithm is successfully able to avoid small rocks, large boulders, craters, and steep slopes in a simulated environment. Determining paths to take using only a 74-degree horizontal field of view was a challenge. Despite these limitations, the robot is still able to successfully navigate autonomously. However, the speed of the robot is a drawback. The physical speed of the robot was reduced to ensure that it did not lose traction and fall down a hill or into an obstacle. The path taken by the robot was not always optimal. This was a design decision, as it was determined early on that the safety of the robot is more important path optimality, so a large amount of buffer space is used when determining the safety of moving in a direction to ensure that the robot has adequate space to avoid clipping any obstacles in the area. This led to some situations where the robot could realistically fit between two obstacles, but the path planning algorithm chose to navigate around the obstacles instead.

VIII. Conclusion

In total, the requirements set for the project have all been successfully met. The odometry system is accurate enough that the rover's predicted location stays well within the required 25 meters on all simulated tests. Cosmic GPS surpassed expectations in real-world tests. Park Ranger was implemented successfully, but the accuracy and consistency of the predicted location was limited by the height and field of view of the camera as well as other factors.

In simulation, the robot is able to travel on a near-optimal route to a destination using only a minimal amount of information about its surroundings. It successfully avoids dangerous obstacles both above and below the ground.

Acknowledgement

The authors wish to acknowledge the assistance and support of Mike Conroy of the Florida Space Institute and Kurt Leucht of the National Aeronautics and Space Administration.

References

[1] R.E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, 1960.

[2] M. Labbe, and F. Michaud, "RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation," *Journal of Field Robotics*, pages 8-10, 2019.

[3] B. van Pham, A. Maligo, and S. Lacroix, "Absolute Map-Based Localization for a Planetary Rover," *12th Sympusium on Advanced Space Technologies and Automation in Robotics*, May 2013.

[4] M. Ghani, and K. Sahari, "Detecting negative obstacle using Kinect sensor," *International Journal of Advanced Robotic Systems*, 2017.

[5] S. L. Laubach, and J. W. Burdick. "An Autonomous Sensor-Based Path-Planner for Planetary Microrovers," *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1999.