Autonomous Teleo-Reactive Multi-Rover Collaboration for Lunar Exploration and Resource Extraction

TeamL3

TeamL3 is composed of 7 aerospace professionals working together remotely from Tokyo (Japan), Ottawa (Canada), Pasadena (United States), and Sydney (Australia), bringing a wide range of expertise and experience that proved necessary to solve the complexities of the NASA Space Robotics Challenge Phase 2.
Executive Summary presentation slides

Apeksha Budhkar
Ottawa, Canada

Louis-Jerome Burtz
Tokyo, Japan

Fabian Dubois
Tokyo, Japan

Nathaniel Guy
Ramona, United States

Ashish Kumar
Pasadena, United States

Ilya Kuzovkin
Sydney, Australia

Evan Smal
Ottawa, Canada

With the collaboration and support of OffWorld Inc. and Mission Control Space Services Inc.

I. INTRODUCTION

For the Space Robotics Challenge Phase 2, NASA provided competitors with a computer simulated environment representative of the lunar surface (NASA Video Presentation). Two large landers are surrounded by randomly generated rocks, craters and regions rich in volatiles, creating a 200m by 200m exploration area (Figure 1). Three rover types, each equipped with 4 fully actuated wheels, vision sensors (stereo camera and 3D LiDAR) and simulated battery/power management system are provided. The Scout rover additionally carries a volatile sensor to detect volatile reach regions below the surface within a two meter range. The Excavator is equipped with an articulated arm and scoop to excavate regolith. The Hauler is equipped with a large bin to transport regolith and deposit it in the processing plant.

Competitors were tasked with submitting to NASA code that enables a fully autonomous team of rovers to:

1. discover volatile rich areas,
2. excavate the regolith of these regions and
3. haul it back to the Processing Plant

Points were awarded as per the mass of volatiles collected within a 2 hour time limit.

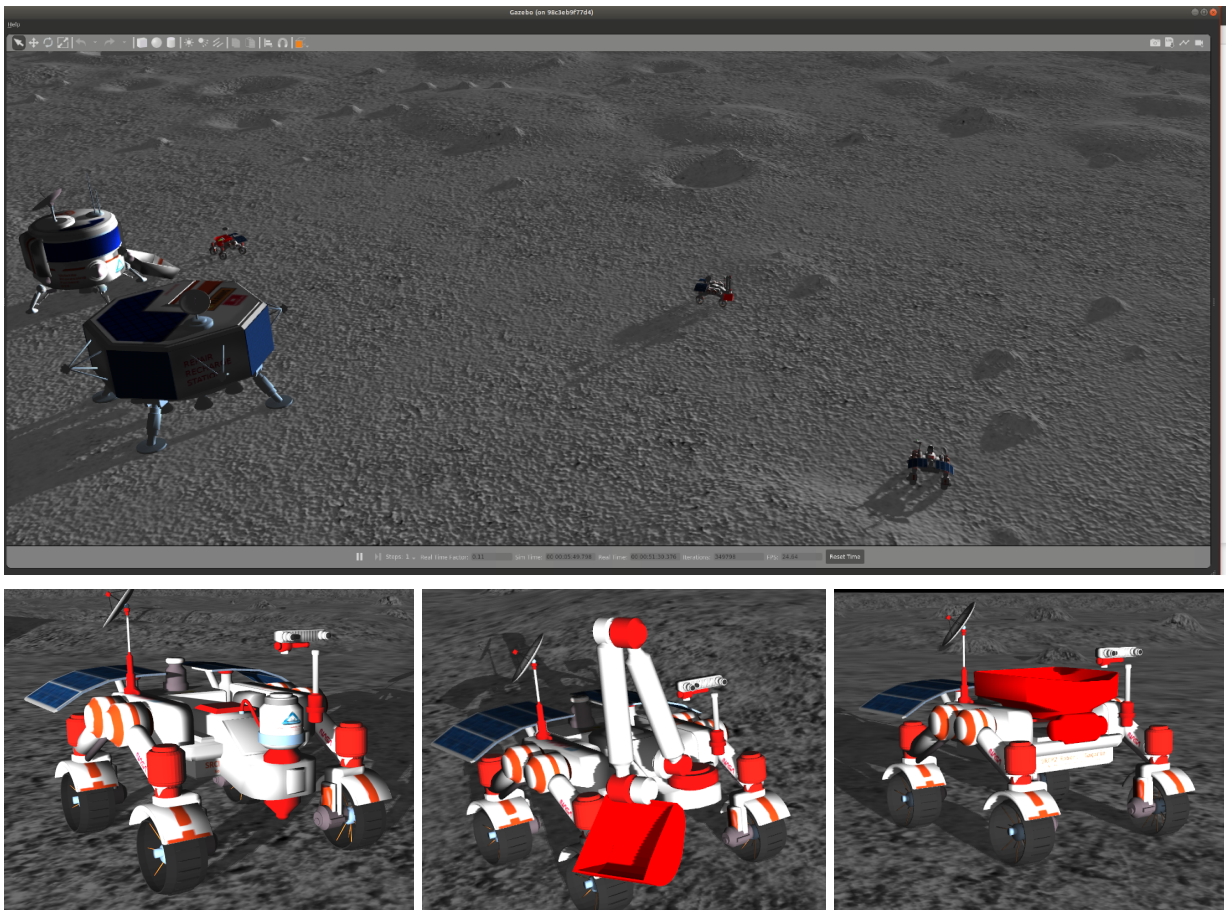

Figure 1. (top) Gazebo simulation of the lunar surface, showing two landers (Processing Plant and Repair Station) and three rovers (bottom, from left to right) Scout, Excavator and Hauler

A. Problem Formulation and Key Constraints

To guide our solution formulation, we identified the key constraints within the official rules:

1. Emphasis on collaboration in a team of heterogeneous rovers
    - A minimum of three rover types is required to detect resources (Scout rover), dig for resources (Excavator rover), and transport resources to the processing plant (Hauler rover)
    - Synchronizing the actions of a single autonomous rover in an unknown environment is hard; doing it for up to six rover is very challenging
    - Each rover has a responsibility for many individual tasks and action planning must adapt based on changing environment inputs.
    - **Strategic outcome:**
      Solution needs a flexible, modular, and reactive way to design, implement, and introspect rover behaviors (i.e., to visualize what the rovers are "thinking"): we implemented teleo-reactive Behavior Trees (Colledanchise, 2017).
2. Rover synchronisation events
    - Precise location (2 m accuracy) of a detected volatile region from Scout to Excavator
    - Precise alignment between Excavator and Hauler to drop off volatile clods from Excavator scoop to Hauler's bin (30 cm accuracy)
    - Precise alignment of Hauler with the Processing Plant for emptying Hauler's bin into the Hopper (30 cm accuracy)
    - **Strategic outcome:**
      Solution needs accurate short-range relative pose estimation capability between two rovers and between the Hauler and the Processing Plant. We trained custom machine learning models (see Section *II-G2 Short-range pose estimation: Neural Network model*).
3. Computation resources and simulation constraints
    - Low "Real Time Factor" for simulations: an entire day to run a test of 2 hours of simulation time -> slow test/fix iterations (even for smaller unit tests)
    - **Strategic outcome:**
      Solution is based on a short atomic "detect / excavate / haul-back cycle" (of 20 minutes), and a repetition of this cycle for the duration of the run.
    - Computing resources available for the grading runs are limited, unclear (from the Official Rules), and subject to change (changed 2 weeks before submission).
    - **Strategic outcome:**
      Careful consideration to select only implementations that can scale to up to 6 rovers. For us, this unfortunately immediately disqualified innovative solutions such as vision-based SLAM for absolute localization.

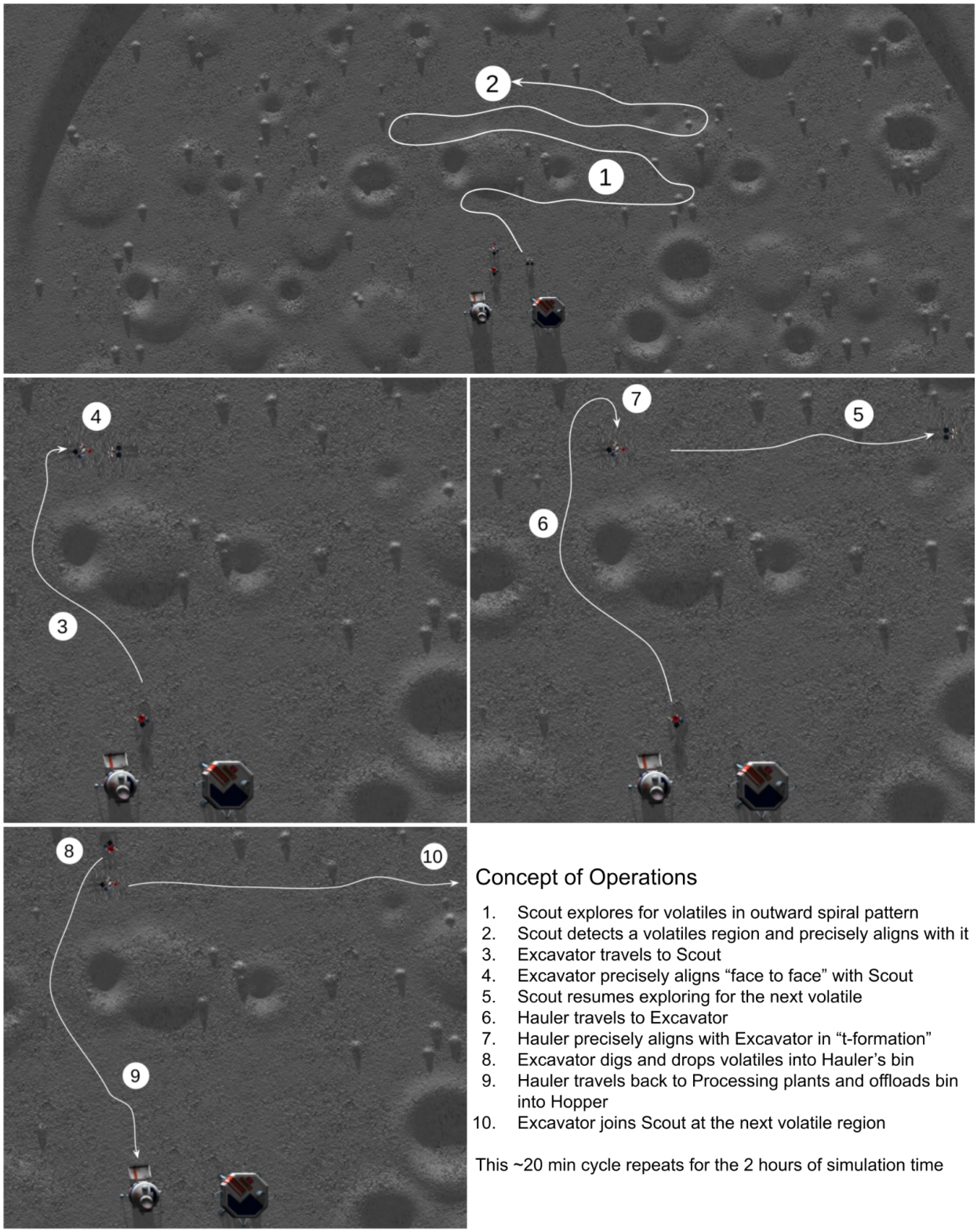

Figure 1. Concept of Operations for submission showing the 10 tasks the 3 rovers must accomplish in each scoring cycle.

B. Concept of Operations

The TeamL3 final submission consists of 3 rovers: a Scout, an Excavator, and a Hauler.

The sequence of events has been carefully developed to minimize the absolute localization requirements: a lead rover always serves as a reference point to the next rover to avoid rovers getting lost on the lunar surface. We developed a 10-task cycle (Figure 1) that orchestrates prospecting, digging, dropping, haul-back, and offloading resources to the Processing Plant. Each cycle takes about 20 minutes to complete and consists of:

- Discovering 1 volatile region
- Excavating 1 volatile region
- Returning the contents of 1 volatile region to the Processing Plant (for scoring)

Each cycle is therefore a score increase opportunity. Several cycles repeat for the duration of the allotted simulation time: up to 6 full cycles within 2 hours.

II. METHODS

A. System Architecture

Our solution to enable full autonomy on three cooperative robots consists of a tightly coupled combination of modules as shown in Figure 2:

1. Each rover is in charge of sensor processing and the low-level “infrastructure” functions that enable it to sense the environment and move safely (blue modules in Figure 2)
2. Collaborative functions orchestrate the tasks of each rover and maintain knowledge of their relative and absolute positions over time (Green modules in Figure 2)

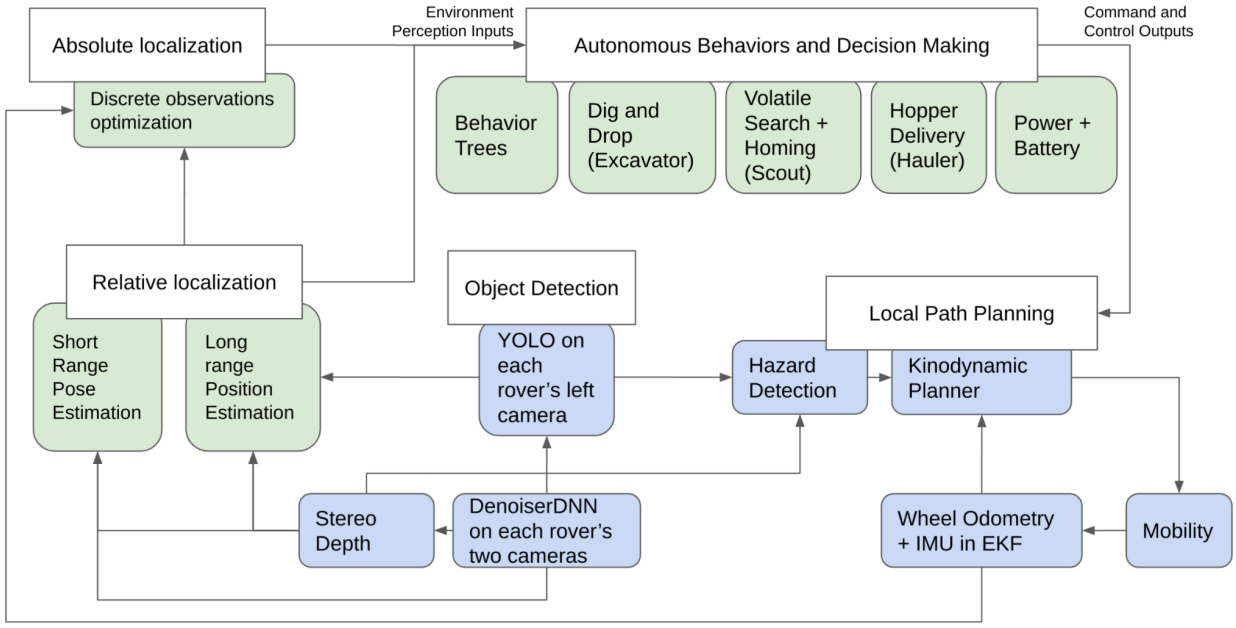

Figure 2. TeamL3 System Architecture. **Green:** Collaboration: high-level functions and those that require collaboration between 2 or more rovers. **Blue:** Infrastructure: low-level functions and those performed for each rover individually.

B. Scout Volatile Search + Homing

To find volatile regions, we defined a sequence of waypoints for the Scout to follow to optimize the coverage and speed of exploration. Together, these waypoints form the basis for Global Path Planning. Each waypoint has a position and orientation, as indicated by white arrows in Figure 3.

This Global Plan creates a series of arcs expanding outwards. Each arc is separated from the previous one by 3.8 m of distance. This is a tradeoff between exhaustive exploration and speed of exploration, ensuring overlap of sensor range (4 meters). The angular distribution is limited to +/- 70 degrees to ensure the rovers always have a good line of sight to both landers (used for localization, see Section *II-I Absolute Localization*).

Figure 3 shows the effective path taken by the Scout in blue: the Scout is following the outward arcs of the global plan while avoiding Hazards (in green, red, and grey). Note: our original plan was to make an equivalent pattern for three more rovers to work on the shadow side of the landers. However, we had to scale down due to CPU and GPU constraints not supporting our solution for 6 rovers.

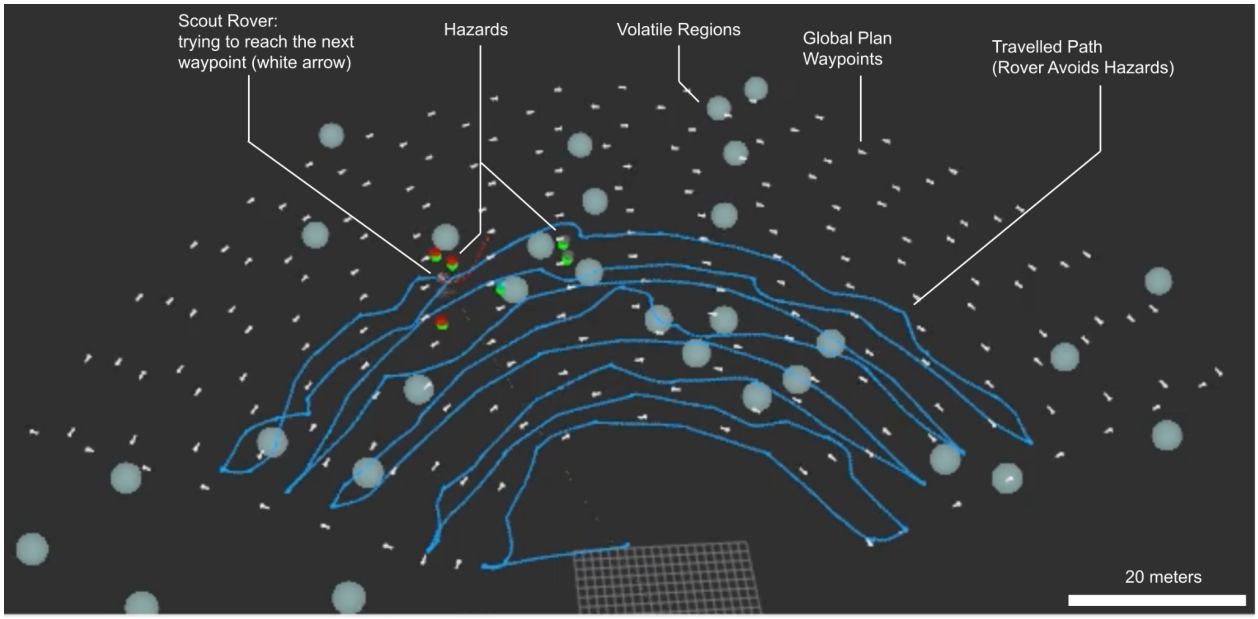

Figure 3. RViz display of the Global Plan (white arrows) and the Scout rover's path (dark blue path) while exploring for volatile regions (light blue 4 m diameter disks) and avoiding Hazards (green, red and grey disks).

For homing in on the resource, we estimate $\frac{\delta D}{vdt}$, where D is the volatile distance reported by the volatile sensor, and v is the rover velocity as estimated by odometry. If the absolute value of the ratio is below a given threshold, we conclude that the volatile is better approached by going in a perpendicular direction, and we turn the wheels 90 degrees. If the ratio is negative, the rover stops and goes in the opposite direction. This lets the rover position itself such that the sensor is located above the resource with any arbitrary tolerance limited only by sensor noise and time.

The next step is to go forward so that the rover center ends up directly above the volatile location, which allows us to rotate toward the Processing Plant for localization without significant loss of volatile localization accuracy. Finally, the Scout backs away to make space for the Excavator and stays there to act as a landmark to assist the Excavator in positioning itself at an ideal dig position.

C. Dig and Drop Excavator

The Excavator action server provides three high-level behaviors:

1. **Dig** - Lower bucket into the regolith and return to hold position.
2. **Drop** - Approach the drop location above the Hauler's bin as defined by the rover alignment "T-formation." Drop clods into the bin and return to hold position.
3. **Empty** - Empty clods in the bucket if there are no volatiles in it. Return to hold position.

Each arm trajectory is implemented by converting a set of waypoints into a joint-space trajectory that can be executed in real-time. During trajectory planning, the desired joint angles are computed once for each Cartesian waypoint in the path using an analytical inverse kinematics (IK) solver (the ikpy open source library).

To guarantee a smooth trajectory and avoid losing volatile clods during the dig and drop sequences, linear interpolation is used to generate equidistant intermediate points between each waypoint in joint-space. Joint-space planning is performed because the conventional task-space planning was determined to be infeasible due the computational resources required; a smooth path requires solving the IK at each point of the trajectory, an order of magnitude more times than at each waypoint.

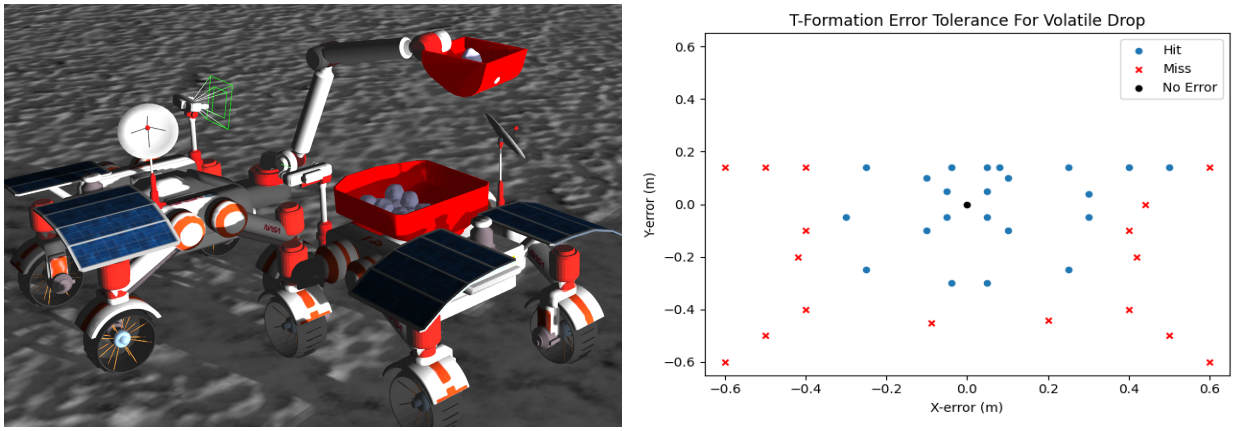

Figure 4. The image on the left shows the Excavator executing a "drop" action with a Hauler in T-formation. The image on the right shows the distribution of hits (blue: defined as at least one clod lands in the bin) and missed drops (orange) against the (x, y) alignment error of the Hauler in T-formation.

For the drop action, accurate T-formation alignment is critical to successfully drop clods into the Hauler's bin. Figure 4 shows that there is an error tolerance of approximately 0.3 m from the

center of the drop location where at least one clod is dropped into the bin (configurations where the Hauler is colliding with the Excavator are not considered). If the hauler is further than 0.3 m from the center of the T-formation, all clods will be dropped on the ground.

For the dig action, Figure 5 shows the distribution of volatile clods collected per dig action as a function of the distance of the rover center from the volatile region center. The Excavator alignment with the volatile region center is critical. The maximum error is 4 m for at least 1 clod; however, this then requires 20 dig and drop sequences to deplete the volatile region which becomes a time bottleneck. The maximum error we set as a requirement to be compatible with the allotted simulation time is 2 m; this leads to at least 3 volatile clods per scoop.

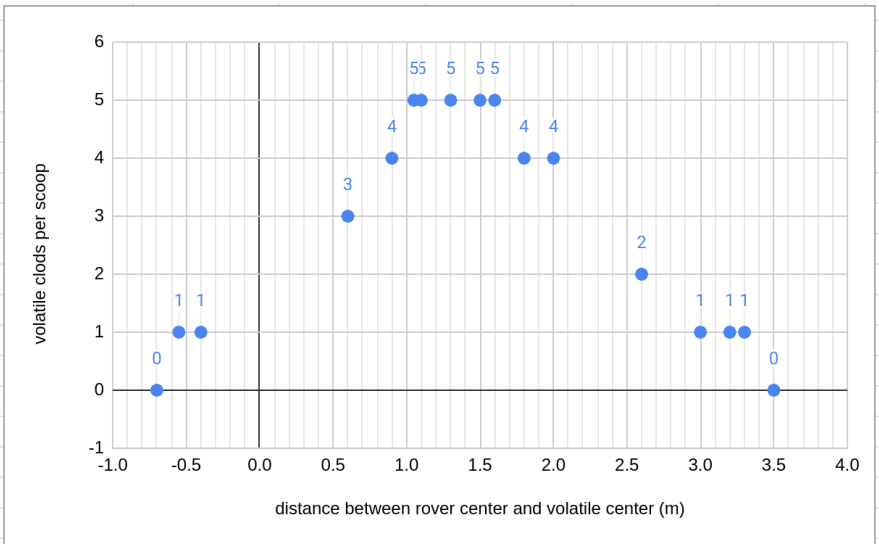

Figure 5. The expected volatiles per “dig” action as a function of distance from rover center to volatile patch center.

D. Denoising Camera Feeds with Deep Learning

Due to the presence of Gaussian noise and localized static noise in the RGB images from the onboard stereo camera (see noisy image example in Figure 7), other modules that depended on RGB images required denoising of the raw images. Since classical approaches like Median Blur filter and Gaussian Filter were not effective on the noise from the environment, we employed a modern machine learning algorithm for denoising. This algorithm is called FastDVDnet (Tassano 2020) and it is a state-of-the-art deep learning-based denoising algorithm for videos. Data collection and training of this neural network was performed during the SRCP2 qualification round and this neural network model proved to generalize well to the final round without any re-training or fine-tuning.

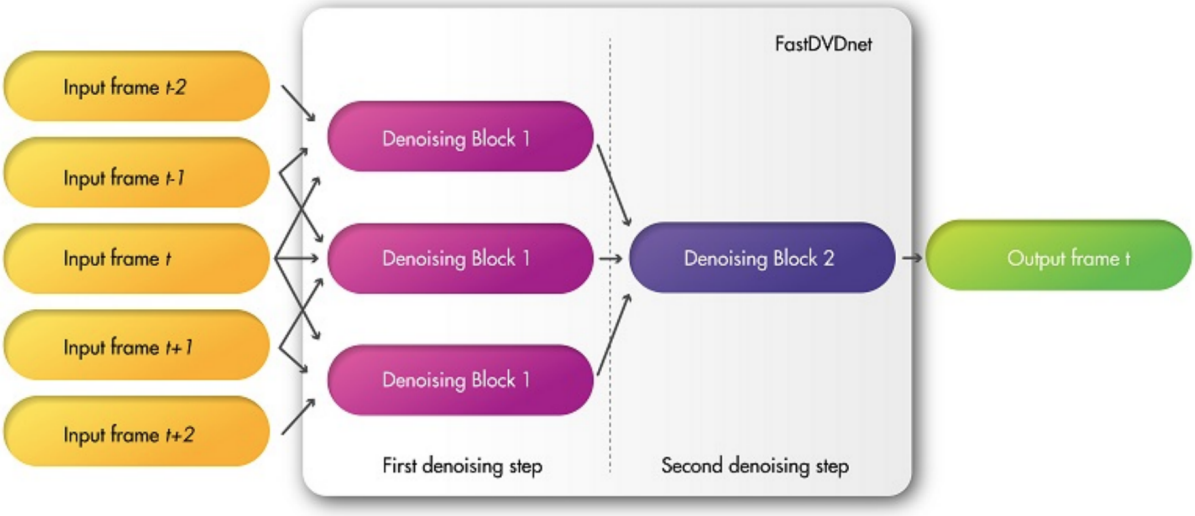

Figure 6. Diagram of inputs/outputs to denoise the middle frame: two steps successively combine three consecutive frames. Diagram reproduced verbatim from (Tassano 2020).

The FastDVDnet model performs denoising with a two-step architecture. The model takes 5 frames as an input and they are provided to 3 denoising blocks in the first step, where each denoising block takes as an input a triplet of the input frames. A noise-map or distribution of noise at the input is also included as input to each block. The denoising blocks in step 1 share the same architecture and same weight.

The output of the denoising blocks in step 1 serves as an input to a single denoising block in step 2, which has the same architecture as the denoising blocks in step 1. These denoising blocks are modified multi-scale U-nets (multi-scale encoder-decoders with skip connections). These blocks have 16 convolutional layers and a point-wise ReLU activation function is applied to the outputs of the convolutional layers in most layers, along with batch normalization. Denoising is performed separately for the left and the right cameras of each rover and an approximate time synchronization filter is applied to the input images.

FastDVDnet was successful in removing both the entire frame Gaussian noise and the frame edge static noise interference from the input images (see Figure 7). The final network requires 1.4GB of GPU memory. This is reduced from our original usage of 2.8GB by using half-precision (FP16) types. (Note that this change was necessary, despite the risk of lower numerical stability and lower accuracy, due to the low specifications of the grading machine GPU.)

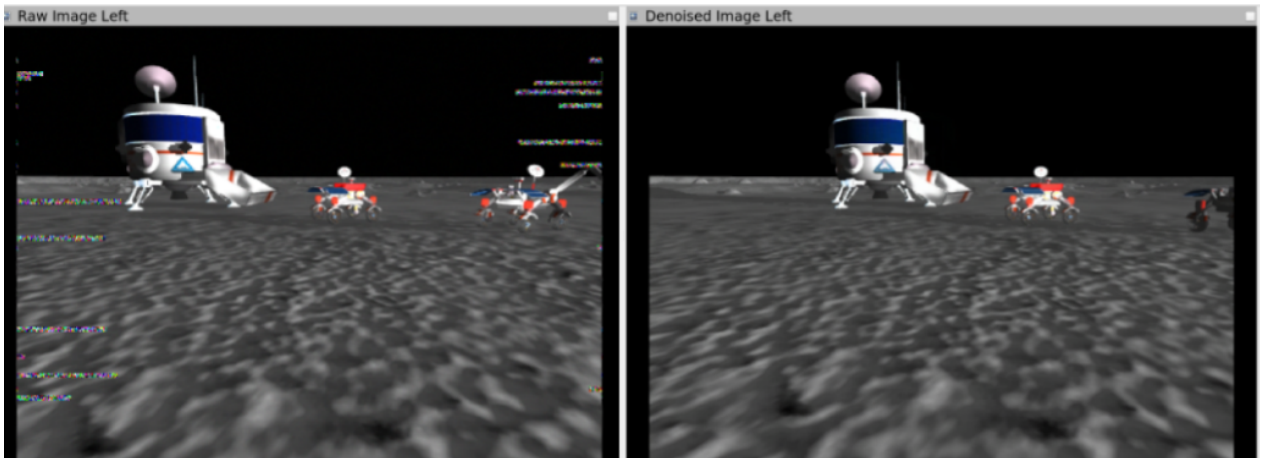

Figure 7. FastDVDnet deep neural network denoising the image on the left. The image on the right is the denoised version.

E. Point Cloud Reconstruction from Stereo RGB Feeds

The tasks of relative pose estimation (including robot-to-robot alignment), hazard detection (especially rocks and craters), and localization (including ranging the landers to re-calibrate a rover's absolute position) all depend on an accurate estimation of distances to objects in the field of view of the robot.

In our perception stack we use the stereo reconstruction technique to generate 3D point clouds from the denoised left and right camera images, for each rover. For this purpose, we implemented the stereo block matching (Konolige 2008) algorithm, which is a simple box filter-based stereo-matching algorithm that still achieves sub-pixel accuracy. We used the stereo_image_proc ROS package, which internally uses the OpenCV (Bradski and Kaehler 2008) implementation of the stereo box matching algorithm. The default parameter values from the stereo_image_proc ROS package provided the best tradeoff between point cloud density, accuracy, and noise rejection.

Stereo matching proved to be very sensitive to noise in the camera images. Removing stereo matching artifacts due to the denoising process in order to provide a clean point cloud for downstream modules was the most challenging aspect.

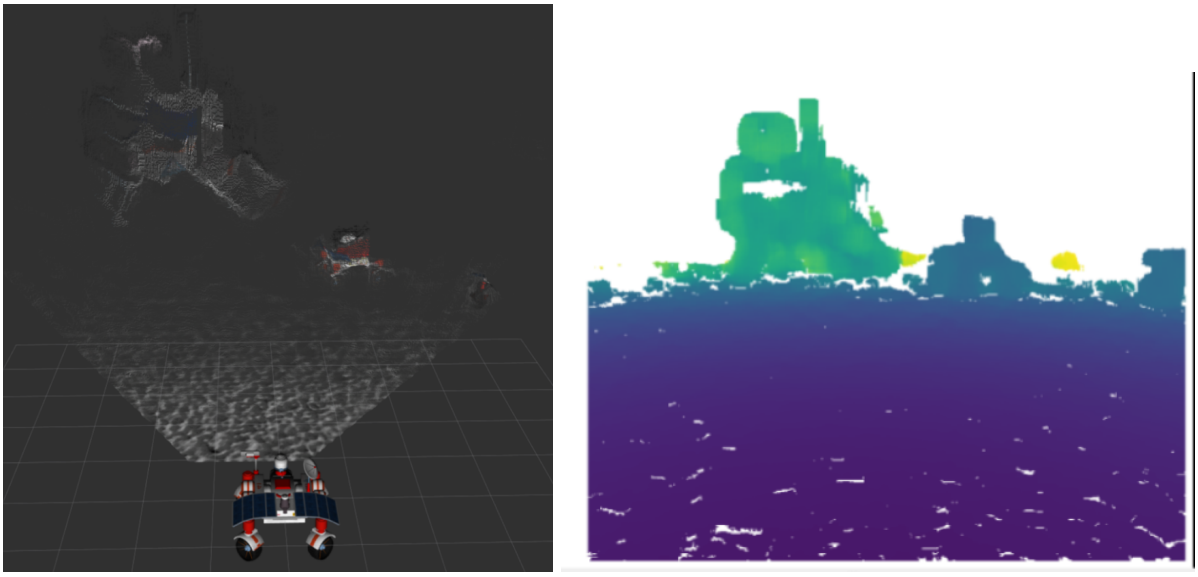

Figure 8. Stereo reconstruction using stereo block matching algorithm. The image on the left shows a 3D point cloud visualization, with colorization of each pixel in RViz. The image on the right is a depth image, where color values represent the distance to the camera (blue is closest, yellow is farthest, and white is “no data”). The first representation provides good spatial context for human eyes to identify matching artifacts; the second representation is the format the downstream algorithms use.

F. Convolutional Deep Neural Network for Real-Time Object Detection

The ability of a robot to operate autonomously in an unknown and unpredictable environment requires a certain level of scene understanding and visual object detection that can inform the robot about the obstacles, other robots, rogue agents, sudden changes in the environment, and more. In recent years, methods based on deep convolutional neural networks have superseded classical approaches for visual object detection and classification by achieving state-of-the-art results (Krizhevsky 2012). YOLO (Redmon 2016) is one such method that is especially suited for real-time object detection.

We collected and labeled a custom dataset of objects that are relevant to the NASA SRC challenge: the network was trained to detect 15 object classes: hills, rocks, and craters for the obstacle avoidance and path planning stack; scout, excavator, and hauler for multi-robot collaboration during the mission, processing plant and the station for long-distance localization calibration and resource delivery, and finally plant hopper, excavator axle, scout sensor, hauler bin, and excavator arm to augment precise robot alignment during operations.

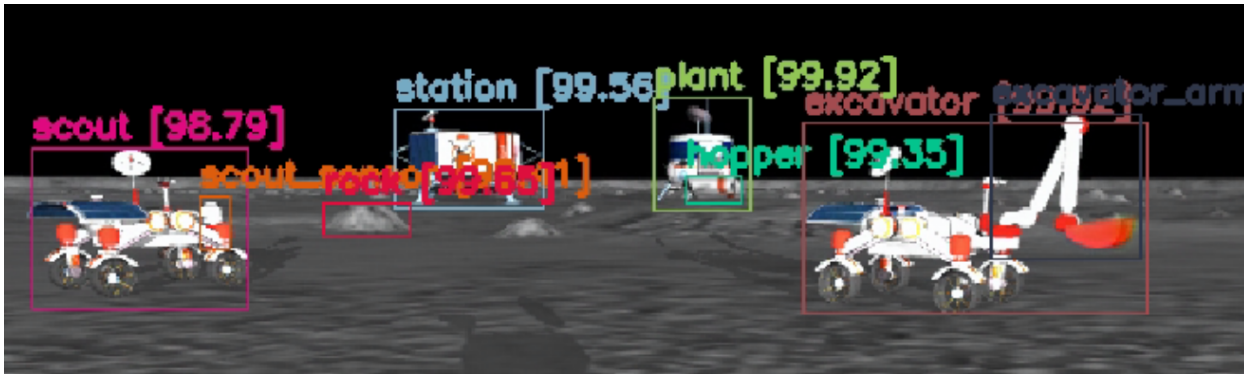

Figure 9. Deep Convolutional Neural Network trained to detect and classify objects relevant to the mission. The screenshot is taken from a live camera feed during operation.

Figure 10. Reliable object detection in varying lighting conditions.

The final network was trained for 47.4 hours (single GPU) on a dataset of 1,216 images manually annotated with ~3,000 bounding boxes. During the operational deployment, the object detection module required 1.4 GB of GPU memory and only 20% of a single CPU core, providing all 3 rovers with up to 5 Hz real-time object detection.

G. Relative Localization

A key enabler of the collaboration between rovers is the knowledge of their relative positions. We defined two main classes of relative position:

1. Long range: several tens of meters between a rover and the target object. This estimation is used for having a general knowledge of the target object, needed when starting to move in the direction of that target. For this purpose, the needed rate is low (every few seconds) and the accuracy required is low (several meters of error are acceptable).
2. Short range: less than 10 meters between a rover and the target object. This estimation is used for precise alignment, it is needed in the final stages of rovers going into formation with each other. For this purpose the needed rate is high (we implemented 2 Hz) and the accuracy is very challenging (better than 15 cm) to avoid rovers colliding with each other or volatile clods missing the mark.

1) Long-range Position Estimation: StereoRanger

Building on the modules presented above, we implemented a StereoRanger module for long distance relative position estimation between:

- Rover to Rover: when rovers have to go towards each other to meet
- Rover to Charging Station and Processing Plant: rovers use the two landers and landmarks to re-calibrate their absolute localization (see Section *II-I Absolute Localization*)
- Rover to Rocks and Craters: for Hazard Detection (see Section *II-H Local Path Planning*)

Inputs are:

- The bounding box coordinates and class names from the YOLO Object Detection module
- The 3D point cloud from the Stereo module

Outputs are:

- (x, y, z) coordinates of the target object in the rover's frame
- Statistics including number of samples used and the standard deviation to assess confidence in the estimation

The StereoRanger is generic and works with any bounding box given by the Object Detection module, as long as sufficient valid stereo point cloud data is available. The plots in Figure 11 below show the results of a typical unit test of the StereoRanger: the Scout rover is moving forward towards the Repairing Station from 70 meters away (near the edge of the simulated arena) until it reaches a point closer than 10m. This proves stable and consistent relative position estimation (x, y, z), with a relative horizontal error of about 10% over the working range.

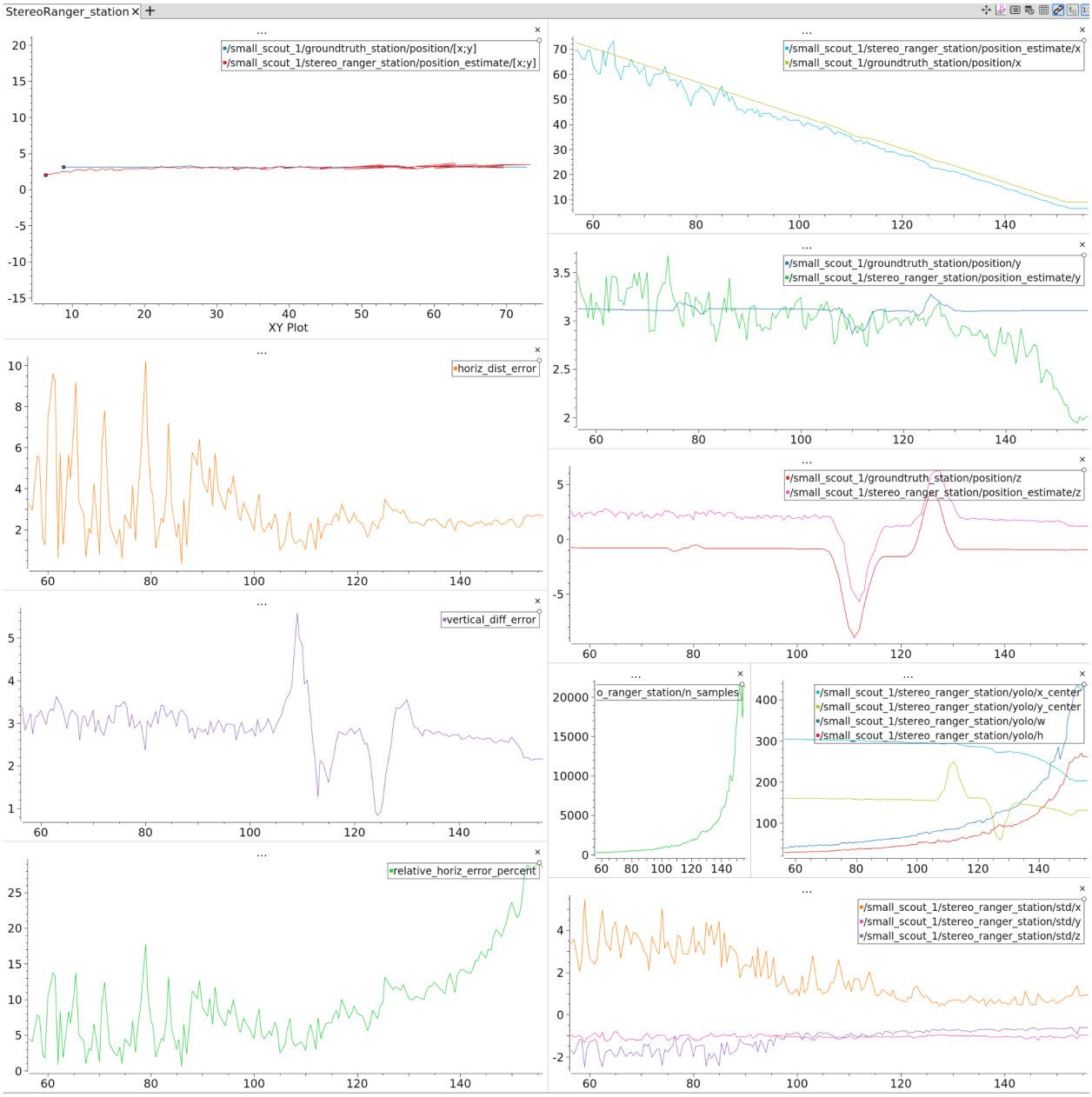

Figure 11. Time series of StereoRanger estimate while the rover approaches the Repair Station lander. Top left: view from above showing horizontal coordinates (red is ground truth, blue is estimate). Left: three error metrics. Top right: comparison of (x, y, z) coordinates with ground truth. Bottom right: n_samples = number of stereo pixels used for computing the estimate, statistics of bounding box, and standard deviation.

2) Short-range Pose Estimation: Neural Network Model

Short-range pose estimation is necessary for:

- Sharing knowledge of volatile region positions between the Scout and Excavator
- Precise alignment between the Excavator and the Hauler for the volatiles to drop from the Excavator's bucket into the Hauler's bin

- Precise alignment between the Hauler and the Processing Plant for offloading the Hauler's bin into the Hopper

For these use cases, not only position but the full six degrees of freedom (position and orientation) must have high accuracy to allow precise alignment. We developed 5 dedicated Neural Network models to cater to each of these use cases. We leveraged the ability to gather thousands of training samples from the Gazebo simulation to train the models to learn the features that correspond to relative pose.

In the figures below, we show results for the "T-formation" model: where the Hauler has to align precisely with the Excavator for the volatile clods handover. This proved to be the most difficult model to prepare, due to the proximity of the rovers and the tight tolerances needed for regolith and volatile clods to not fall on the ground.

For this model, the training data is composed of 2,400 samples. Each sample consists of:

- RGB image from the Hauler's left camera (left column)
- Depth image from the Hauler's stereo camera (middle column)
- Ground truth relative pose between the Hauler and the Excavator, taken from Gazebo (right plot)

The training samples are arranged in a "funnel" shape to cover the validity domain of the final approach of the Hauler towards the Excavator.

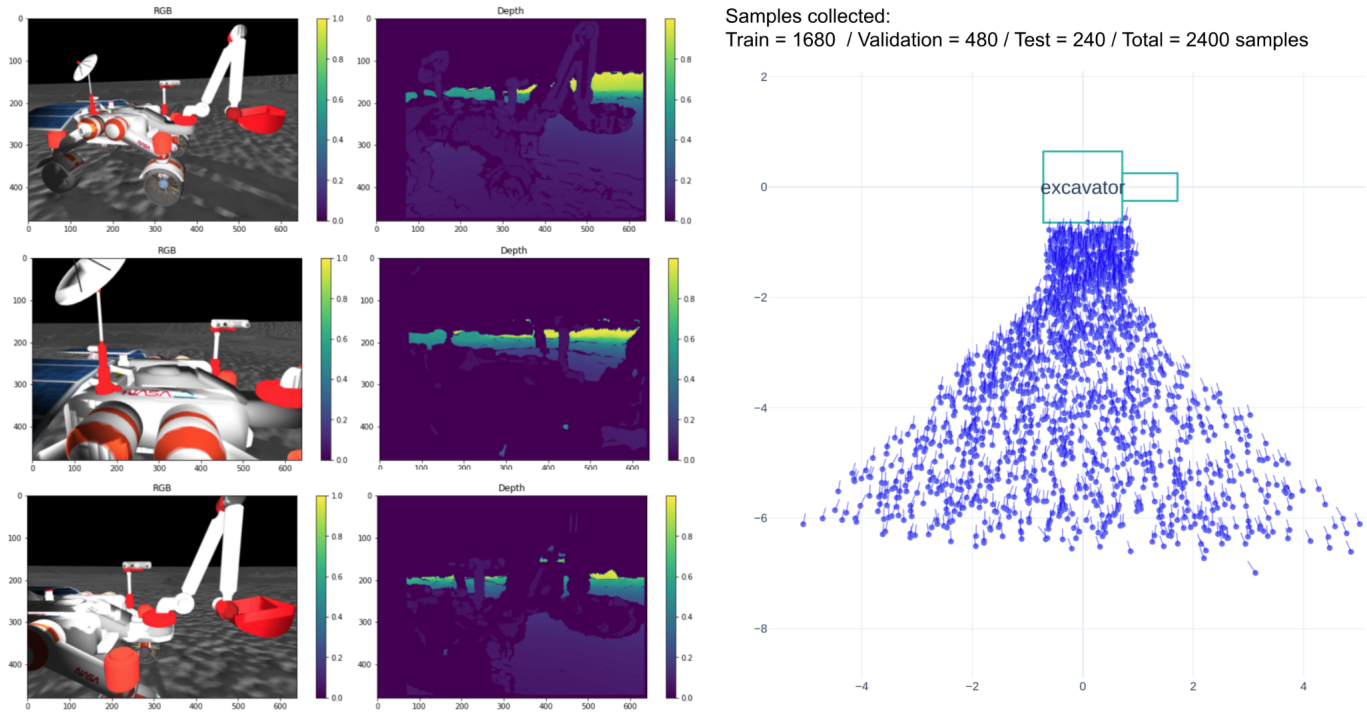

Figure 12. Training dataset for the T-formation model. Left: RGB and depth image sample. Right: all ground truth poses. Each blue dot is a training sample, showing position and orientation of the Hauler rover in the Excavator reference frame.

The results of the trained model are evaluated on a test dataset (never seen during training), shown in Figure 13 below.
At inference time, the model performs well on never-before-seen data; there is no significant bias on regressed distance, theta, and yaw. After converting from the optical frame to the rover frame (via ROS tf2 transforms), the median absolute error in the horizontal axes x and y is 17 cm. The final network consists of 5 models loaded on demand for each of the relative pose estimation use cases. The total GPU memory required is 800MB (this is reduced from the original 1.4GB by forcing a GPU memory limit on the Tensorflow virtual device. This reduces the speed of inference and increases the CPU to GPU bandwidth usage. This change was required by the lower than initially announced spec of the grading GPU)

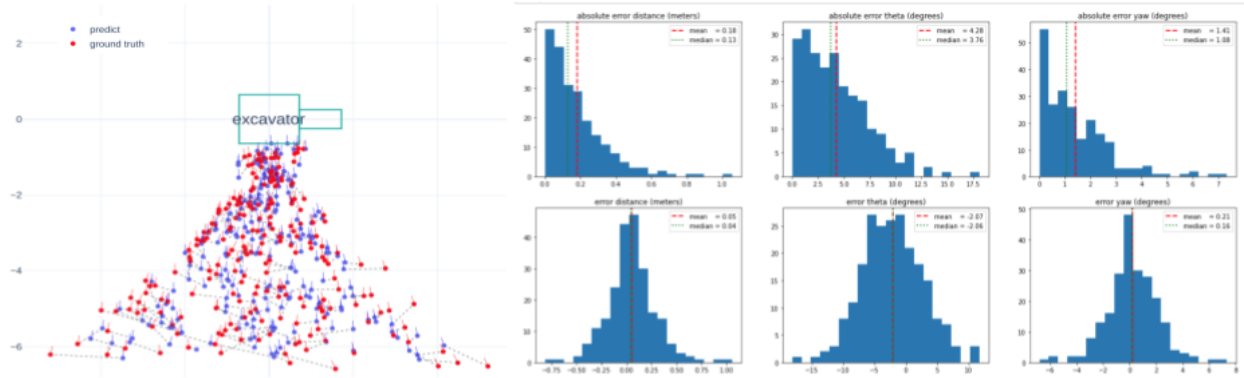

Figure 13. Left: inference results comparing estimated (blue) with ground truth (red). Right: histogram of absolute and relative errors for distance, theta, and yaw (the three variables regressed by the model).

H. Local Path Planning

Local Path Planning is performed independently on each rover, following the goals and tasks set by the higher level collaboration behaviors. In this work, Local Path Planning includes the following inter-dependent functions:

- Mobility (PID tuning + command and control)
- Rover Odometry estimation
- Hazard Detection and Avoidance
- Short-term / short distance trajectory planner

These four functions are classical robotics techniques and form the basis of a well-functioning rover.

1) Mobility

Regarding mobility, we implemented the following:

- PID tuning of the four wheels and four steering arms to be responsive and without oscillations
- Constant acceleration limitation when starting and stopping movements to avoid the rover suspension from bunching up (a phenomenon which negatively impacted rover odometry due to spurious IMU readings)

- Automatic braking: when the velocity set point on all four wheels is 0 rad/s for 3 consecutive seconds, the four wheel brakes are set to 400N force. As soon as a non-zero velocity set point is received, the brakes are set to 1N force (the minimum)
- Explicit point turn: wheel steering arms rotate to $\pi/2$ - *arctan(wheel separation width / wheel separation length)*: the rover is turning 'on the spot', with no translational movement.
- Arbitrary crab: wheel steering arms all rotate to the same arbitrary angle. If the angle is 0 rad, the rover goes straight forward; other angles make the rover move in a crab fashion

These two mobility modes are enough to cover all needs in terms of rover position and orientation. The key point is that these two modes restrict all wheels to always be rotating at the same set angular velocity. This simplifies the odometry calculations and greatly reduces wheel slip. Less wheel slip creates fewer Gazebo simulation artifacts and reduces wheel odometry drift.

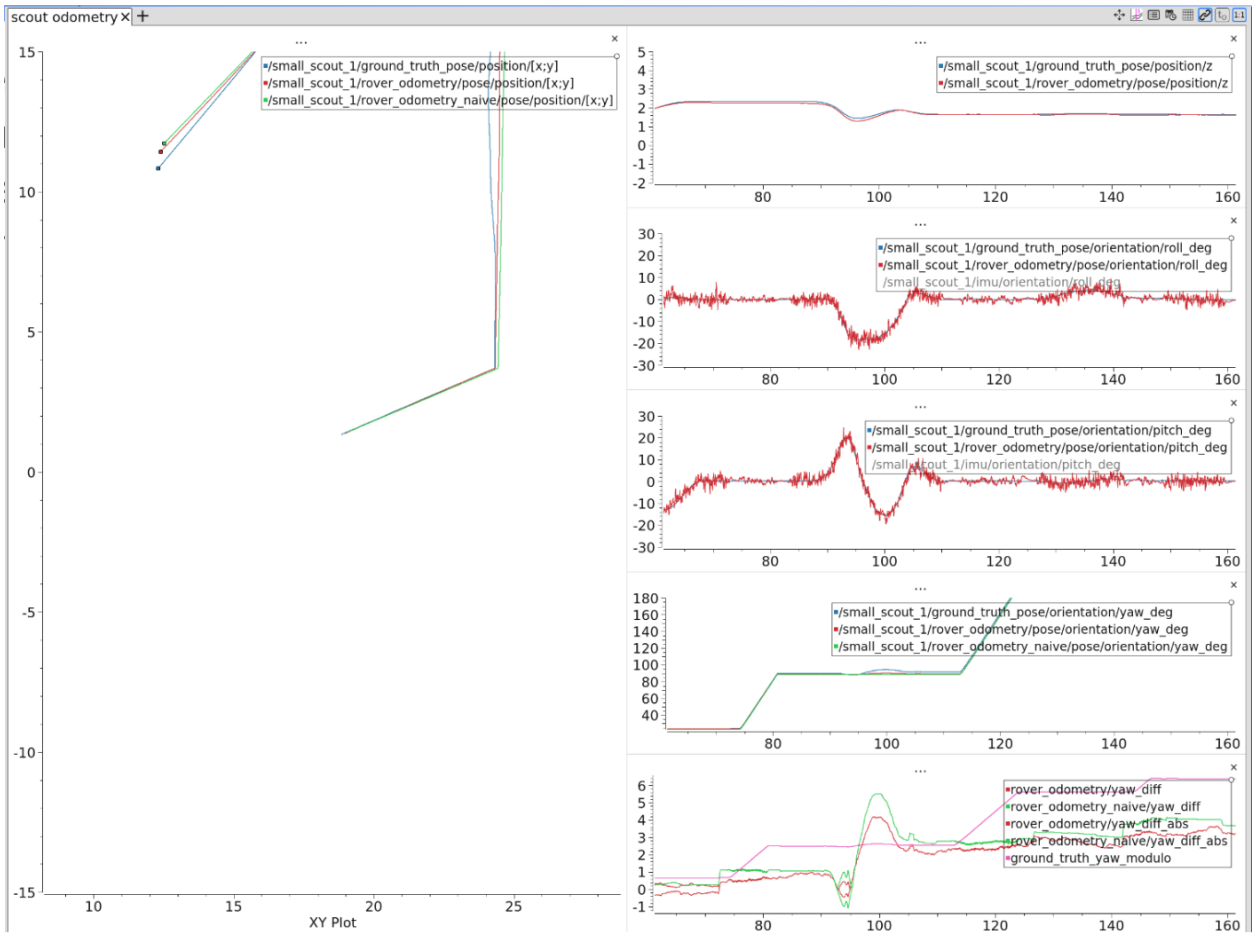

Figure 14. Rover Odometry unit test results. Left: top-down view of ground truth (red), wheel odometry (blue, largest drift), and rover odometry (green, fused wheel odometry and IMU). Right: position, roll, pitch, and yaw estimates compared with ground truth.

2) Rover Odometry

We fuse the wheel odometry computation (from wheel and steering arm encoders) with the Inertial Measurement Unit (IMU) measurements. (Due to extremely high noise on the IMU, we only fuse roll, pitch, 3 angular velocities, and 3 angular accelerations in an Extended Kalman Filter with properly tuned covariance matrices.) The generated Rover Odometry position estimate is accurate to about 1% of distance traveled. A main source of error is the yaw estimate, which drifts primarily when wheel slip occurs on slopes or obstacles. This module is implemented via the ROS robot_localization package (Moore, 2016).

3) Hazard Detection and Avoidance

The Hazard Detection and Avoidance module has the critical roles of

1. Avoiding rovers colliding with each other, with rocks, and with the two landers
2. Avoiding potential high-slip situations (e.g., going into craters)

(Note that hills are considered benign.)

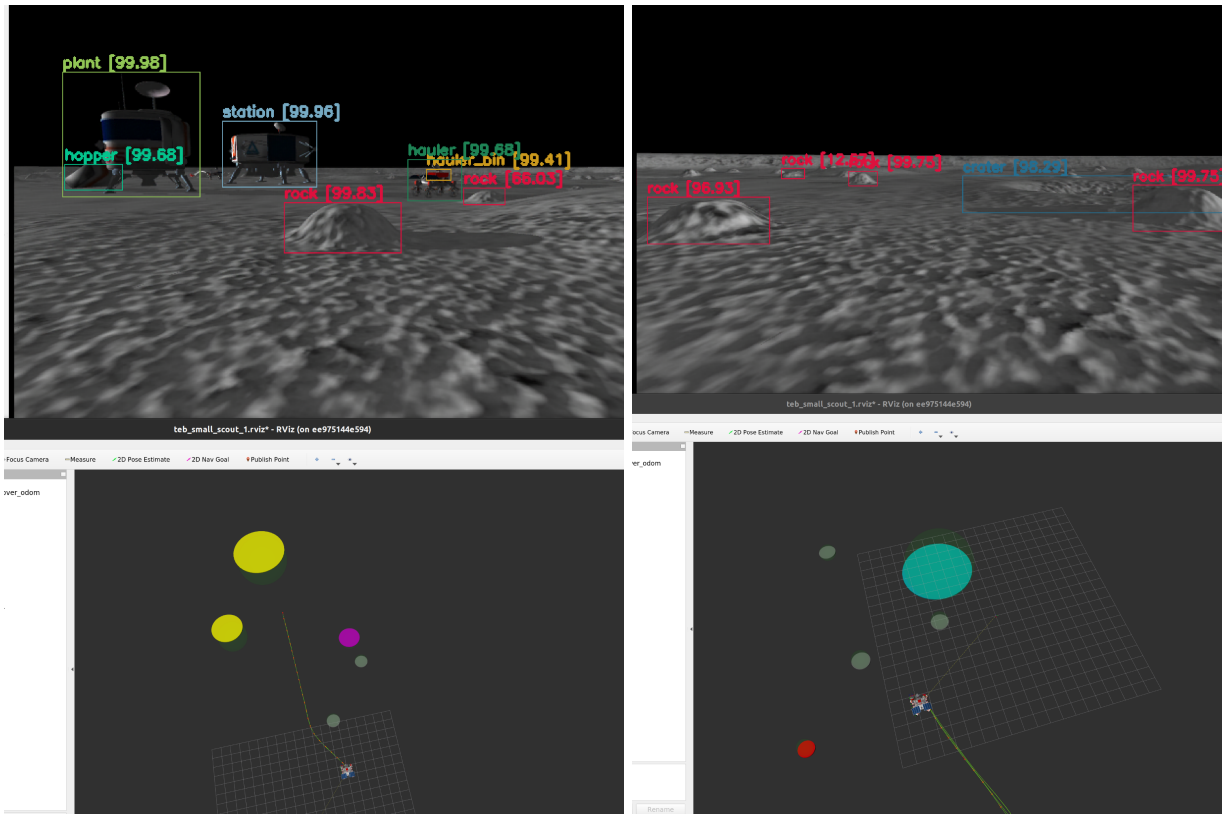

Figure 15. Yellow: landers. Pink: rovers. Grey: rocks. Teal: craters. Red: obstacles in memory. A new observation of an obstacle with the same position as a memorized one replaces the memorized one. Only obstacles closer than 10 m are memorized; this is in order to keep track of hazards (e.g., rocks) when turning around them.

The module takes the StereoRanger as input (object class names and estimated 3D coordinates) and outputs discrete rover, rock, and crater obstacles modeled as disks (with a position and best-fit diameter). These discrete obstacles are displayed in RViz and sent to the kinodynamic planner, which plans a trajectory that keeps a safe distance away from them.
The field of view of the cameras is narrow (80 degrees horizontal), so we implemented a memorization function to enable the rover to negotiate turns around hazards even when they leave the field of view (memorized obstacles are shown in red in RViz).

The Hazard Detection also serves a second role: identifying whether the area around a volatile region detected by the Scout is valid (i.e., safe for excavating). A valid volatile region is defined as:

- After resource alignment: Scout roll and pitch are lower than 5 degrees (to avoid slopes that cause rover-to-rover misalignments)
- The zone in front of the Scout is free of rocks and craters (to avoid infeasible rover-to-rover approach trajectories)

The safe zone extent is as defined in Figure 16 (top). The outputs of this volatile region validation are visualized in the Dashboard UI as shown in Figure 16 (bottom). In effect, some of the volatile regions that are detected by the Scout are under rocks or craters. This region validation ensures that the time-consuming rover-to-rover alignment sequences are only attempted when conditions are good. This strategy reduces the number of volatile regions that are excavated, but we have found that this is not a limitation, since the Scout volatile exploration strategy is efficient and total run duration time is a stronger bottleneck than the number of valid volatile regions found.

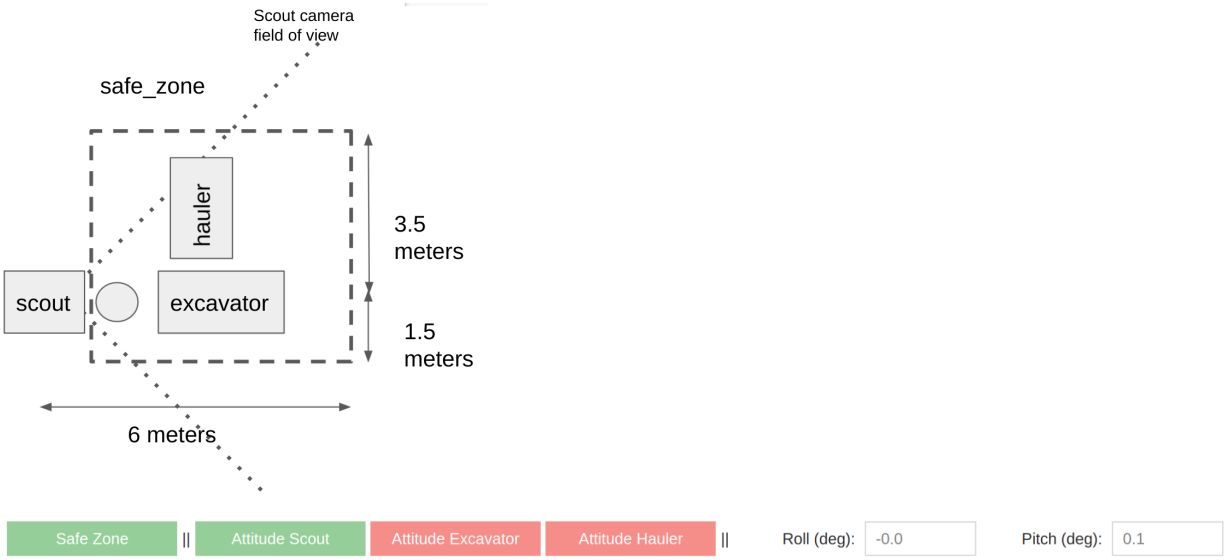

Figure 16. Top: schematic of the area in front of the Scout considered for the safe_zone computation. Bottom: user interface indicators for safe zone, attitude thresholds for each of the three rovers, and filtered rover roll and pitch.

4) Local Trajectory

Using all of the above outputs, the rover is now ready to compute a trajectory and decide which drive commands to send. We define “local trajectory planning” as:

- Short-term: plans the next 30 seconds (reduces CPU load)
- Short distance: plans the next 20 meters (reduces CPU load)
- Uses detected hazards
- Takes into account kinodynamic constraints of the rover (e.g., x, y, and theta accelerations and maximum speeds, footprint size, etc.)
- Computes the optimal path from the current position to the next waypoint at 2 Hz

The rover continuously sends the appropriate commands to the wheels and steering arms to follow this local trajectory. In Figure 17, the computed trajectory (in red) correctly avoids the detected hazards to reach the next waypoint (at right of image). Note that up to 6 homotopy classes (green paths) are optimized in parallel: this avoids the planner getting stuck in local minima (which often happens in other standard robotics planners). This module is implemented via the ROS teb_local_planner package (Rösmann 2017). We forked and modified this package for standalone execution outside of the ROS navigation stack. We intend to open-source the derivative node that we developed for community use.

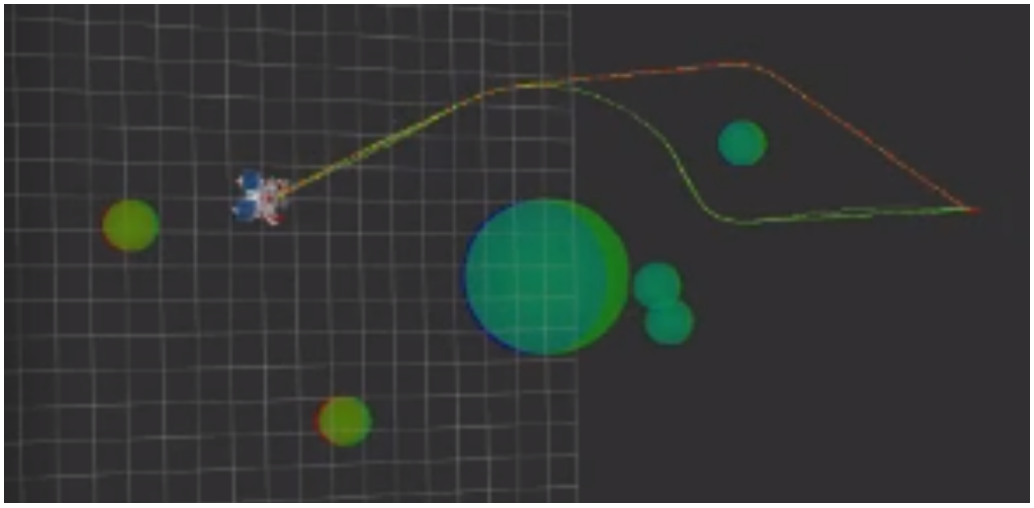

Figure 17. A visualization of local trajectory planning. The computed trajectory is the red path; homotopy classes are the green paths. Green and red circles are hazards.

I. Absolute Localization

Our absolute localization strategy relies on high semantic level observations of the environment. We initially planned to use a factor graph implementation; however, due to resource constraints and the fact that the odometry drift was reasonable, we settled on a much simpler solution.

We did relocalization at key events during the simulation, which we used to redefine the odometry frame of the rovers. The first set of event, is specific to the Gazebo simulation’s setup,

and would be replaced in a real mission with initial deployment knowledge from mission operations:

a. At the start of the simulation for the Scout and Excavator, we use the Gazebo simulation's **get_true_pose** service
b. At the start of the simulation for the Hauler**,** we use *a priori* simulation position knowledge + Scout/Excavator orientation (from a.)
c. After the first commissioning empty drop of the Hauler, we use the Gazebo simulation's **get_true_pose** service. This step is important to learn the orientation of the Hopper.

The second set of relocalization events could generalize to a real mission:

1. For the Scout: when looking at the Processing Plant on top of a volatile, we use the stereo ranging estimates (see details below).
2. For the Excavator: when it is in face-to-face alignment with the Scout, we use our expected alignment position knowledge and the Scout's absolute position from 1.
3. For the Hauler: at the bin drop, we use our expected alignment position and the position of the Hopper from c.

The relocalization of the Scout at step 1 is done by optimization of the pose, based on constraints given by:

- The current odometry pose
- The relative pose observation of the Processing Plant
- The relative pose observation of the Charging Station

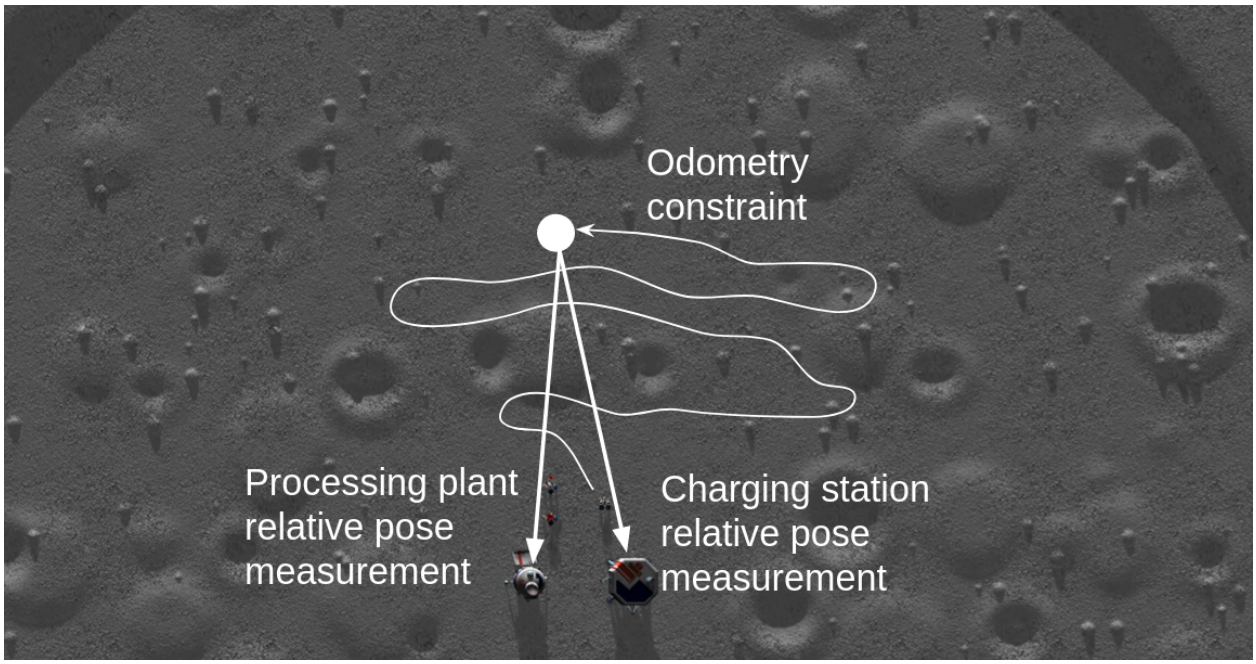

Figure 18. An overhead view of a test scenario illustrating the three constraints used for pose optimization. The Scout is at the top, in the white circle.

Weighting of the constraints is performed based on the reliability of each measurement. We optimized the exploration pattern to ensure that we can detect the 2 landers and ensure an accurate relocalization at any time. This step usually decreases the localization error by a factor of 2, enough to ensure no major drift during the duration of the exploration.

J. Behavior Trees

For high-level control and synchronization of the rovers, we use 1 behavior tree per rover, each running at 0.5 Hz.

While Finite State Machines (FSMs) have a long history of being used in robotics, their main drawback is their lack of reactivity and modularity. Behavior Trees (BTs) solve these two issues using two-way control transfer instead of one-way control transfer (Colledanchise 2017).

A behavior tree can be represented as a tree structure. Leaves are either conditions or action nodes. Other nodes of the tree are control flow nodes. A BT can itself take the role of an action in another BT, contributing to modularity.

Execution of the BT occurs at a fixed time interval, where a tick signal is generated and propagated from parent node to child node according to the control flow rules. A node can return three execution statuses: *success*, *failure*, and *running*. Execution finishes when the root node returns its execution status.

We use a minimal implementation relying on only Sequences and Fallback control nodes with the possible use of state variables. The Sequence control flow node (symbolized by ->) executes all child nodes until one node returns *failure* or *running*. If all nodes succeed, it returns *success*. The Fallback (also called Selector) control flow node (symbolized by ?) executes the child nodes until one node returns *success* or *running*. If all nodes fail, it returns *failure*.

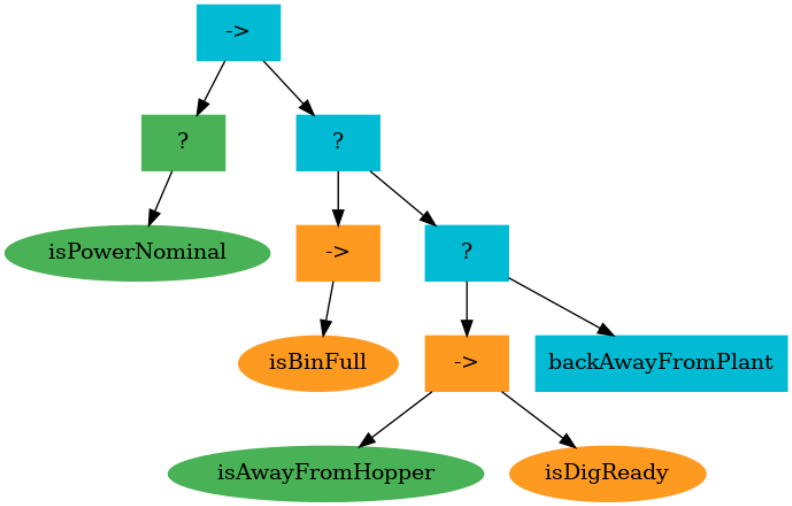

Figure 19. Example behavior tree with state visualization (running, success, failure). Watch the video linked in Section *III-C2 Scoring results* to see the full behavior trees of each rover in action.

This is sufficient to generalize decision trees, finite state machines, and teleo-reactive approaches.

We achieve a goal-oriented design, using the fact that BTs generalize teleo-reactive approaches. Implicit sequences make the design a succession of goals (postconditions) and tasks required to achieve each goal, along with their preconditions.

To improve reactivity, we try to use stateless idempotent tasks. However, it is sometimes necessary to use nodes with memory (state) to keep idempotence and prevent an action from being executed repeatedly (e.g., “move 1 meter forward” being retriggered at each tick, leading to a neverending traverse).

Some of the nodes are executing a low-level behavior controlled via odometry feedback, which can combine a localization goal with a goal tracking via the camera pan (for example, the Hauler tracking the Excavator with the camera while moving toward a relative pose goal).

K. Power and Battery

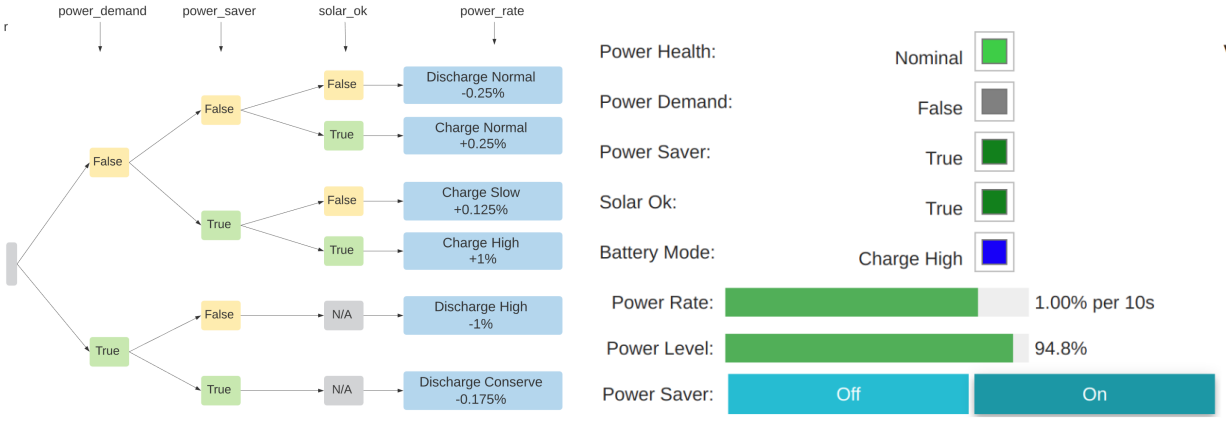

Figure 20. Left: a decision tree showing power and battery implementation for the rovers. Right: a visualization of each rover’s system_monitor topic.

From our understanding of the power and battery implementation of the rovers (as depicted in decision tree in Figure 20), we extracted the following design drivers:

- When the rover is not moving (power_demand is False) then the power system is always power-positive (regardless of solar panels viewing the Sun or not)
- Therefore, transition from nominal battery mode to Low Power will only happen when the rover is moving: during a traverse.
- Therefore, it is always safe to stop and spot turn to align the rover solar panels and achieve charge High. (I.e., there is no risk of a Low Power situation when two rovers are precisely aligned with each other, e.g., during excavating.)

We implemented a simple Power/Battery management strategy: a behavior is always monitoring the battery level (cf. visualization of each rover’s system_monitor topic in Figure 20) and taking the following two actions:

1. If the battery level falls below 26%, the rover stops, spot turns until solar alignment, and then sets the power saver. This assures Charge High mode. In 3 simulation minutes, the rover battery reaches 45%. At this point, the rover resumes its previous task.
2. As soon as the velocity set point on all four wheels is 0 rad/s for 3 consecutive seconds, the power saver is set. This ensures the rover is power-positive. In 50% of yaw orientations, this will provide Charge High. As soon as a non-zero velocity set point is received, Power Saver is unset.

Notes:

- In our strategy, the rovers are spending enough time immobile (when waiting for another rover or when in precise alignment for excavating) that with the above power/battery management system, charging is rarely needed and does not become a time bottleneck.
- We do not make use of the fast charge of the Repairing Station (lander in the center of the exploration area). The time to travel to the Repairing Station and the interruption in tasks was not worth it.
- We guarantee that the rovers should never be in Low Power or Emergency Power. This allows us to simplify our solution by eliminating the necessity of dealing with the orange and red markers in the camera images as well as reduced wheel speeds.

III. RESULTS

A. Testing Approach

Using an iterative testing approach, the team selected a series of seeds between 1000-7000 and tested the entire system in the Gazebo simulation for each seed. The team tested the system under a wide range of scenarios by leveraging seven workstations among team members to run tests in parallel across different seeds. Each test gave a chance to identify corner cases and implement fixes to make our solution more robust for the next tests.

During the testing cycle, the team tested over 50 random seeds and generated video screengrabs and anomaly reports for each to provide an assessment of the system's performance. The workflow can be summarized in the following steps:

1. Each of the 7 monitoring workstations selects a random seed for an iteration.
2. The workstations begin monitoring the rover behaviors and generate reports on the status of the run.
    a. A major difficulty: each run takes between 4 and 12 hours to complete due to the low Real Time Factor.
    b. For every unsuccessful run, the team identifies the root cause and develops fixes that are validated by unit tests and then pushed to the main branch.
3. All monitoring workstations pull the fixes available at the time of their next iteration, and the monitoring cycle starts again at step 1.

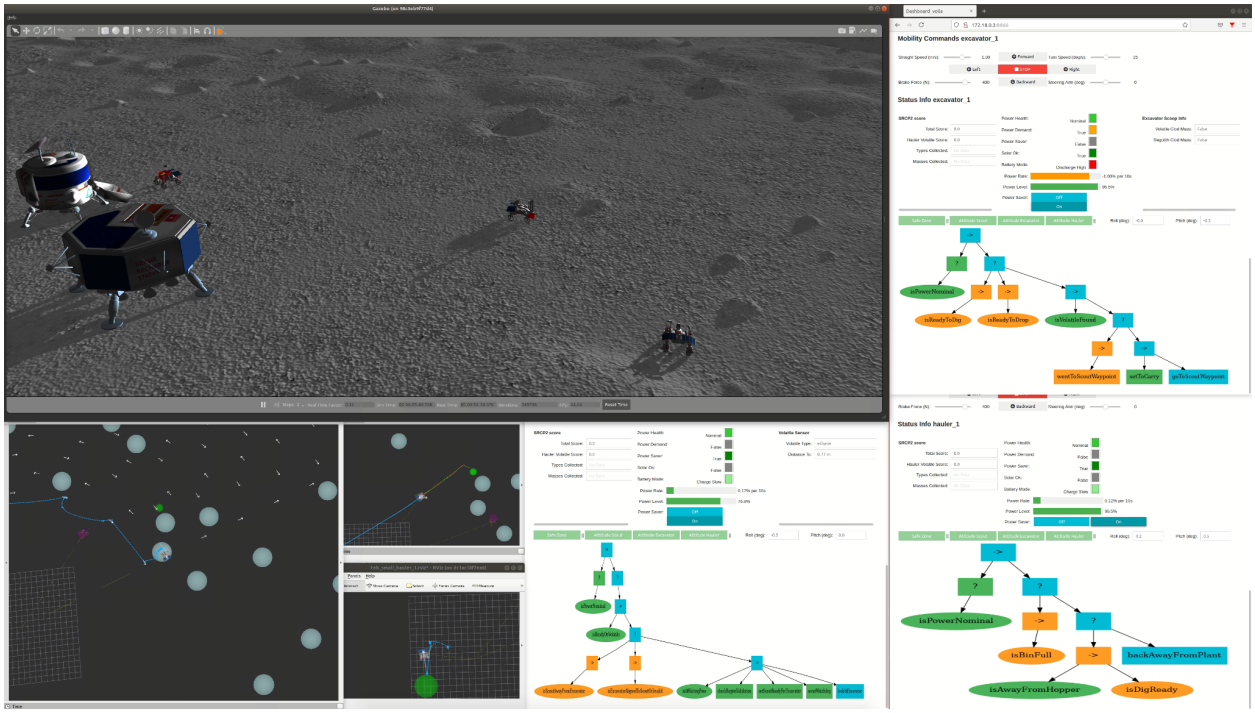

Figure 21. Example of a test visualization setup, showing the Gazebo simulated lunar surface (top left), the Voila and ipywidgets dashboard user interfaces (right) and RViz panels (bottom left) for each rover.

B. Visualization of Rover Telemetry and Inner Workings

A key design element of TeamL3's development and testing strategy relies on visualization (example test setup in Figure 21). We developed:

1. A dashboard user interface for each rover using the Voila and ipywidgets frameworks (right)
2. An RViz panel for each rover (bottom left)

Together, these six graphical elements allow the operator to monitor rover telemetry and behavior during development and testing. They provide an invaluable window into the inner workings of the algorithms. At a glance, all the inputs and outputs of the cascade of modules is understandable. The behavior tree visualizations especially give a sense of what the AI managing the rovers is "thinking."

These visualizations eliminate guesswork when trying to identify the root cause of a failure symptom. This low debugging cognitive load and the deep diagnostics capability empowered our team members to make rapid iterative progress during development and integration testing.

C. Integration Results

The team's integration approach proved successful in identifying and addressing errors in the rover behaviors. First, all modules were unit tested by themselves in representative conditions (e.g., two rovers aligning with each other). Next, full system integrated tests began. At the beginning, rovers would get stuck after completing just a handful of tasks as unforeseen situations arose. Synchronization, accuracy, and sensor noise issues were then addressed via iterative improvements. This allowed the rovers to complete all their tasks in a cycle and efficiently transition to the next cycle.

1) Watchdog implementation

Testing revealed the impressive complexity and variety of situations that the rovers face. Taking a page from aerospace design experience, we added a watchdog timer to the autonomous behaviors of our rovers. This implementation addresses anomalous situations where one or more rovers would get "stuck," unable to complete their current task or move on to the next task.

The watchdog sets a 40-minute timer and looks for success criteria to reset the timer:

- Valid volatile region found by the Scout rover
- Successful depletion of a volatile region by the Excavator rover
- Successful offloading of volatile clods from the Hauler rover's bin into the Processing Plant Hopper

When the timer expires, the rover behaviors are reset: each rover restarts its tasks from the beginning. This strategy was successfully tested to be working and to allow a “stuck” rover to recover and a scoring cycle to be attempted again.

2) Scoring results

To better explain how our rover team works together to accomplish the tasks necessary for scoring, we prepared an accelerated video (viewable on YouTube through this link: https://bit.ly/SRCP2TeamL3)

The video spans a single scoring cycle showing:

- All three rovers working together
- 20 clods of sulfur dioxide excavated from a volatile region and dropped into the hauler’s bin
- Hauler offloading all clods into the Processing Plant Hopper
- The outcome of this single cycle is 18 points (2 clods minimum scoring quantity for sulfur dioxide volatile type).
- At the end of the video, the three rovers efficiently transition to the next cycle. These cycles repeat until the end of the run.

The video is accelerated for the viewer’s sanity. In actuality, one scoring cycle takes 20 simulation minutes to complete (on average). For the NASA judging, the run will be 2 simulation hours. We therefore expect 6 cycles, yielding a total score up to 120 points. The actual score will depend on the seeds used for grading.

IV. CONCLUSIONS AND DISCUSSION

The system architecture and the implementation of the modules presented above have produced exciting results within the boundary conditions of the NASA Space Robotics Challenge Phase 2. The TeamL3 solution deploys a team of 3 heterogenous rovers working in sync to complete ISRU relevant tasks in a computer-simulated environment representative of the Lunar South pole.

Looking ahead towards implementation in a real, upcoming NASA mission scenario, the TeamL3 solution is especially innovative in three complementary areas:

A. Innovative integration of deep learning methods into the robotics stack
B. Innovative autonomous multi-rover collaboration with behavior trees
C. Innovative visualization and development methodology for autonomous robotics

We aim to open-source the reusable modules of our solution.

A. Innovative Integration of Deep Learning Methods into the Robotics Stack

The TeamL3 solution incorporates methods based on machine learning to replace otherwise laborious manual specification of operational conditions. Machine learning methods have demonstrated remarkable results in vision-based applications in and outside the field of robotics and are becoming mature enough to be tested and deployed in an operational mission scenario. The NASA Space Robotics Challenge gave us a great platform to put this conviction to test and we are happy to report that the results confirmed our expectations.

The three modules based on machine learning are described in more detail in Section *II. Methods*. They rely on deep convolutional neural networks to help robots make sense of noisy and complex visual data.

The first module, DenoiserDNN, was trained on pairs of noisy/clean images and learned to recover a clean image from the noisy one. A successful implementation of this method allowed downstream modules to operate on noiseless images, leading to better precision, efficiency, and robustness during operation.

The second module is the object detection and classification node. It implements a custom-trained YOLO network to locate a predefined set of objects in the camera images in real time. The successful execution of our concept of operations relied on this module to identify the two landers, other rovers, and obstacles in the environment and execute the behavioral logic that led to successful execution of the mission objectives. Real-time obstacle detection eliminated the need for precise mapping and allowed rovers to execute predefined routes while dynamically re-planning short-distance navigation trajectories.

The third module provides an estimation of the relative position and orientation of a rover and a

known target object. It implements a fully convolutional neural network to learn the features of the target object that can be used to estimate the relative pose of the object. This module allowed for precise alignment of rovers with each other and with the Processing Plant lander, a necessary prerequisite for the ISRU tasks.

The expected outcome from replacing classical methodologies with learning-based methods was that this would allow greater robustness and adaptability to variations in the environment without the need to manually specify and hardcode all potential configurations an environment might have or to engineer complex heuristics. This expectation was confirmed and is a strong indication of the readiness of machine learning methods to play their part in live robotic operations, especially for vision-based tasks where plenty of data is available.

B. Innovative Autonomous Multi-Rover Collaboration with Behavior Trees

Most of the robotics literature for the past 20 years has used finite state machines to implement robot actions. However, while conceptually simple, these are difficult to maintain and validate as the complexity of the agents in the systems grows.

From the start of the challenge, TeamL3 identified the complexity of multi-robot collaboration as a key design driver. We chose to implement Behavior Trees because they generalize finite state machines and decision trees while providing increased modularity and reactivity.

Behavior Trees are currently primarily used in computer games to define the actions of non-playable characters, but interest in using them for robotics (in both academia and industry) is increasing rapidly (Colledanchise, 2017). Following the good results of the qualification phase, we have already open-sourced the Behavior Tree library and associated visualization tools that we developed.

C. Innovative Visualization and Development Methodology for Autonomous Robotics

A strong focus of TeamL3's development was to create an "explainable" solution. For every module, we created visualizations of inputs and outputs and combined them in an intuitive user interface. These visualizations enable both increased iterative development speed and quality assurance.

In traditional complex systems, many components are "black boxes," raising the difficulty of troubleshooting or understanding novel situations. In contrast, the approach of TeamL3 has been to foster a teaming between robots and humans (both developers and operators). We expose in greater detail how human-robot teaming is applicable to upcoming Lunar exploration

missions, both during development and during operations, in our recent paper (Burtz et al. 2020). This paper is based on the solution TeamL3 developed for the qualification round of the NASA Space Robotics Challenge. For the final phase, we continued to build on these concepts with the results presented in *Section III*.

D. Acknowledgments

We would like to express our sincere thanks to all of the developers, engineers, project managers, and visionaries at NASA and NineSigma who made the Space Robotics Challenge possible. We are very thankful to have had the opportunity to compete with the other brilliant teams to advance the state of the art in autonomous planetary robotics.

We would also like to thank our collaborators OffWorld Inc. and Mission Control Space Services Inc., who contributed not only resources and advice, but also invaluable team members with whom we achieved what would not have been possible otherwise.

Finally, we offer our gratitude to the developers of the many open-source tools and the ROS/Gazebo communities which enabled our development.

V. REFERENCES

Bradski, G and Kaehler, A 2008 *Learning OpenCV: Computer vision with the OpenCV library*, O'Reilly Media, Inc.

Burtz, LJ, Dubois, F, and Guy, N 2020 'Human-Robot Teaming Strategy for Fast Teleoperation of a Lunar Resource Exploration Rover', *International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Colledanchise M and Ögren P 2017 *Behavior Trees in Robotics and AI: An Introduction.* arXiv preprint, arXiv:1709.00084.

Konolige, K, Agrawal, M, Bolles, RC, Cowan, C, Fischler, M and Gerkey, B 2008 'Outdoor mapping and navigation using stereo vision', *Experimental Robotics* (pp. 179-190). Springer, Berlin, Heidelberg.

Krizhevsky, A, Sutskever, I and Hinton, GE 2012 'Imagenet classification with deep convolutional neural networks', *Advances in neural information processing systems*, *25*, pp.1097-1105.

Moore, T and Stouch, D 2016 'A generalized extended kalman filter implementation for the robot operating system', *Intelligent autonomous systems*.

Redmon, J, Divvala, S, Girshick, R and Farhadi, A 2016 'You only look once: Unified, real-time object detection', *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

Rösmann, C, Hoffmann, F and Bertram, T 2017 'Integrated online trajectory planning and optimization in distinctive topologies', *Robotics and Autonomous Systems*, Vol. 88, 2017, pp. 142–153.

Tassano, M, Delon, J and Veit, T 2020 'Fastdvdnet: Towards real-time deep video denoising without flow estimation', *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*