CSCI370 Final Report

moonbees?

Julia Harvey
Derek Suzumoto
Ethan Brucker
Philip Belous

Revised December 6, 2022

CSCI 370 Fall 2022

Prof. Kelly

Table 1: Revision history

Revision	Date	Comments
First Draft	11/27/2022	
Peer Reviewed	12/5/22	
Polished	12/6/22	

Table of Contents

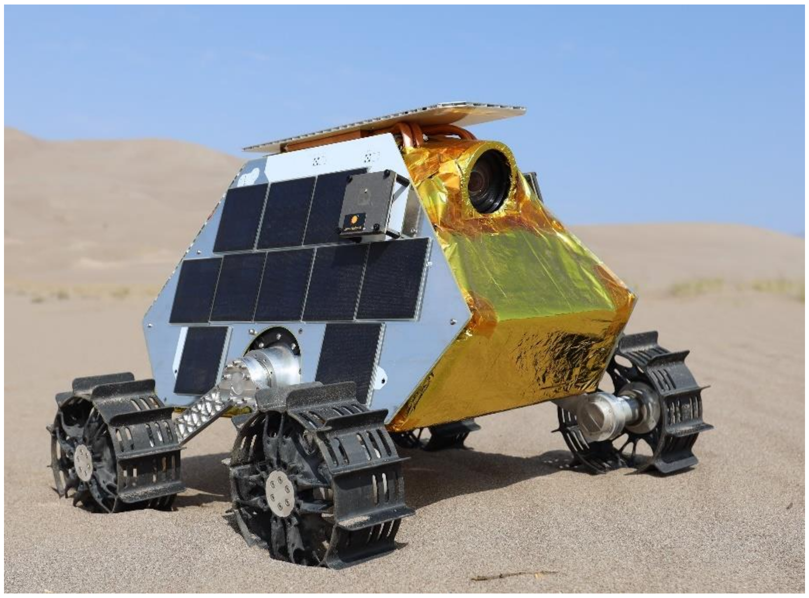
I. Introduction

Lunar Outpost Inc. is a company that focuses on developing and deploying advanced technologies that have both terrestrial and space applications. The company is comprised of engineers with experience working with NASA, defense, and commercial programs, and it is engaged in contracts with the U.S. Department of Defense, NASA, local and state government organizations, and leading research institutions. They have a few ongoing projects including MAPP, a prospector rover, Canary, an environmental monitoring system in the energy sector and municipalities across the continent, and contributions to MOXIE, the Mars Oxygen In-Situ Resource Utilization Experiment onboard the Perserverance rover.

The most relevant technology for our given project is the MAPP rover, designed to map resources and carry payloads on the lunar surface in tandem with other MAPP rovers. These rovers will work together to make mapping and payload delivery as efficient as possible. Especially in mapping, it is important to know where each rover is at every increment of time because they will be attempting to make a single complete map between all of them. This means accuracy and precision of its positioning as well as its knowledge of the other rover's positions is crucial.

While each rover will already be equipped with a positioning system, our goal was to implement robot-swarm co-localization, an additional positioning system that would be used to improve the accuracy of their location data. The idea behind this approach is that by using the video feed from the rover, we can use image processing to identify other rovers in the camera frame. After identifying them, we can then extract their 3-dimensional orientation and coordinates relative to the camera. This location data can then be used to accept, reject, or fuse with the location data of the rover swarm.

This project was passed on to us from a previous field session. Our tasks were to improve the previous team's solution, expanding their program's ability to visually identify the body panels of Lunar Outpost's rovers to handle any angle, excluding from below. We were also to work to improve the efficiency of their code so that it would run on a live-captured video stream and use the data extracted from the feed to update the robot transformation tree.



II. Functional Requirements

The program must be adapted from the previous team's field session code--which relied on ArUco markers to identify a rover and its orientation--to identify multiple rovers without said markers. Instead, the rovers will be identified via physical structure, using either a classical computer vision approach, a machine learning approach, or some combination of both.

Software must be able to:

- Identify the rover without the use of ArUco markers.
- Identify any panel of the rover from any orientation, excluding upside-down.
- Run on a live video stream, maintaining at least 7 FPS.
- Use camera information to update the ROS Transformation Tree of the robot and update its relative position.

III. Non-Functional Requirements

Software will:

- Be written in Python but maintain the ability to be translated to a C-language later.
- Use Python's OpenCV to handle image processing.
- Be ROS-compatible, for possible future integration from Lunar Outpost.

IV. Risks

This project has certain risks due to the existing technology and our skills prior to the project. The existing codebase may require extensive modification to make it more reliable and modular. Since our access to the rover is limited, it may be difficult to test code without a remote bench setup. However, we will be provided with datasets from the company, and we may be able to come in to work with the rover occasionally. In addition, we are relatively unfamiliar with the Microsoft development suite, and only 1 member of our team fully completed the Mines computer vision course before taking on the project. The rest of the team is taking the CV course concurrently. Learning computer vision concepts and getting used to the Microsoft suite will require additional time. Regarding the project's solution, we have 2 routes: traditional computer vision and machine learning. The last team on this project attempted to use machine learning and ended up down a time-consuming rabbit hole.

V. Definition of Done

The software should be able to identify the lunar rover on a live video stream and extract useful information about the relative positions of identical rovers. Specifically, for each frame in a video stream input rated at ~7 FPS, the system should be able to identify any of the rover's side panels or top face (if indeed one or more of these are in the frame) and use this detection to craft pose data (3D translation and rotation in relation to the camera used for the video stream) and intermediary development data (such as a bounding box for the most prevalent side panel and its distortion relative to complete perpendicular) for the detected rover, as well as integrate this data into an existing robotics transformation tree that can be used to digitally model the detected rover within Lunar Outpost's existing systems.

VI. System Architecture

From our highest level, our system architecture is very simple. Originally, the idea was to have the camera provide each frame of a live video feed to our OpenCV system, which would use template matching to identify the corners of the rover and extract geometry from those locations, crafting object pose from the developed coordinates. This final pose was then prepped to be fed directly into ROS. Our original system broke the pipeline into 4 small pieces, outlined in *Figure 2* below. Our final version ended up being much simpler (with additional functionality), and is outlined in its entirety in *Figure 3*. The whole system ended up comprising two stages: an optional MaskRCNN and an OpenCV-based PoseExtractor. Both are described in Section VII.

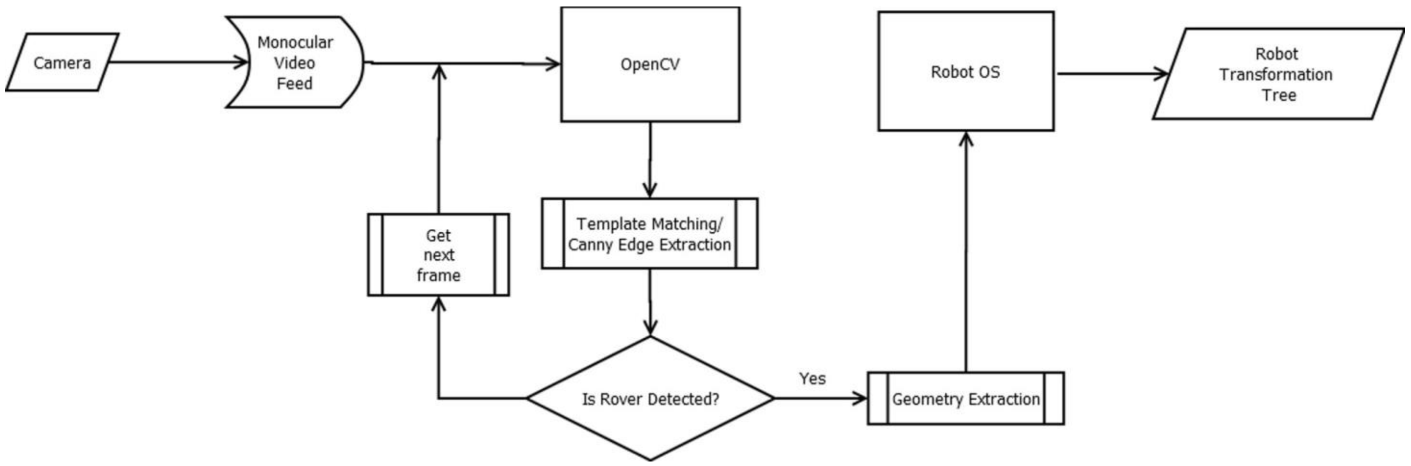


Figure 1: Original System Architecture

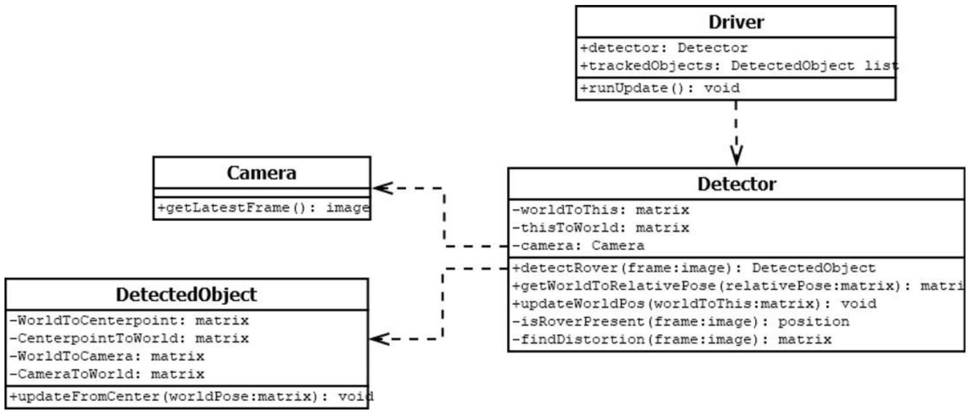

Driver
+detector: Detector
+trackedObjects: DetectedObject list
+runUpdate(): void
Camera
+getLatestFrame(): image
Detector
-worldToThis: matrix
-thisToWorld: matrix
-camera: Camera
+detectRover(frame:image): DetectedObject
+getWorldToRelativePose(relativePose:matrix): matri
+updateWorldPos(worldToThis:matrix): void
-isRoverPresent(frame:image): position
-findDistortion(frame:image): matrix
DetectedObject
-WorldToCenterpoint: matrix
-CenterpointToWorld: matrix
-WorldToCamera: matrix
-CameraToWorld: matrix
+updateFromCenter(worldPose:matrix): voi

Figure 2: Original UML Overview

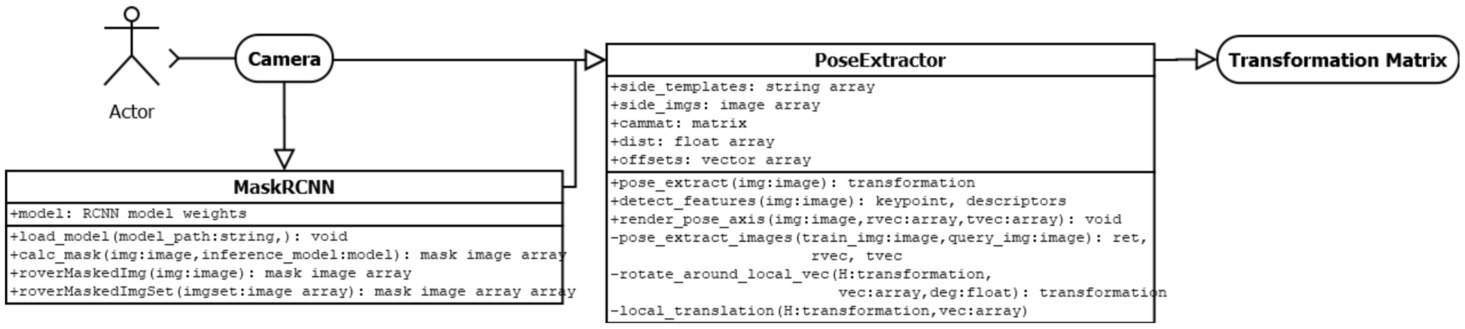

Actor
Camera
MaskRCNN
+model: RCNN model weights
+load_model(model_path:string,): void
+calc_mask(img:image,inference_model:model): mask image array
+roverMaskedImg(img:image): mask image array
+roverMaskedImgSet(imgset:image array): mask image array array
PoseExtractor
+side_templates: string array
+side_imgs: image array
+cammat: matrix
+dist: float array
+offsets: vector array
+pose_extract(img:image): transformation
+detect_features(img:image): keypoint, descriptors
+render_pose_axis(img:image,rvec:array,tvec:array): void
-pose_extract_images(train_img:image,query_img:image): ret, rvec, tvec
-rotate_around_local_vec(H:transformation, vec:array,deg:float): transformation
-local_translation(H:transformation,vec:array)
Transformation Matrix

Figure 3: Final System-Wide UML/Architecture Overview

VII. Technical Design

Our system is composed of two main elements: An ORB-based feature/template matcher with pose extraction, and an intelligent image segmentation model known as MaskRCNN.

The ORB-based feature detector saw many, many versions before the team settled into the final approach as it exists today. First, images of the rover's side panels are taken, warped to be perfectly orthogonal, and the real-world distances and rotations (offsets) between the center of the rover and the image are measured. Then, when the PoseExtractor is presented with a query image, it performs the following for each side panel image and offset:

First, the template and the query image are both scanned with an ORB feature detector tuned for this specific application. ORB was chosen as it was open-source and consistently faster than either SIFT or SURF, the leading competitor (and proprietary) algorithms. In addition to a built-in image pyramid that compares smaller image sections before larger ones, ORB makes use of scale-invariant features that bear binary descriptors that can be used with a Hamming matcher, further increasing its speed over the brute-force matching all but necessitated by the other algorithms.

Once these features are found for both the template and the query image, they are fed into a Hamming matcher. This reads the information accompanying each marker and matches specific rover parts to each other across the two images, discarding outliers. This is also where an image without a rover will be flagged and aborted, returning a null pose. While this is a very robust and reliable process, there is ordinarily no way of determining where these features land on the 3D rover itself. Thus, the team needed to create our own solution.

The final approach used in the system projected the 2-dimensional rover template features into 3-dimensions on a flat plane, making full use of the list of rover template images and offsets mentioned above. This allowed the detected points on the query image to be similarly projected into 3D space by simply matching the texture of the cube-like structure created by arranging the side panel images in 3D space according to their offsets.

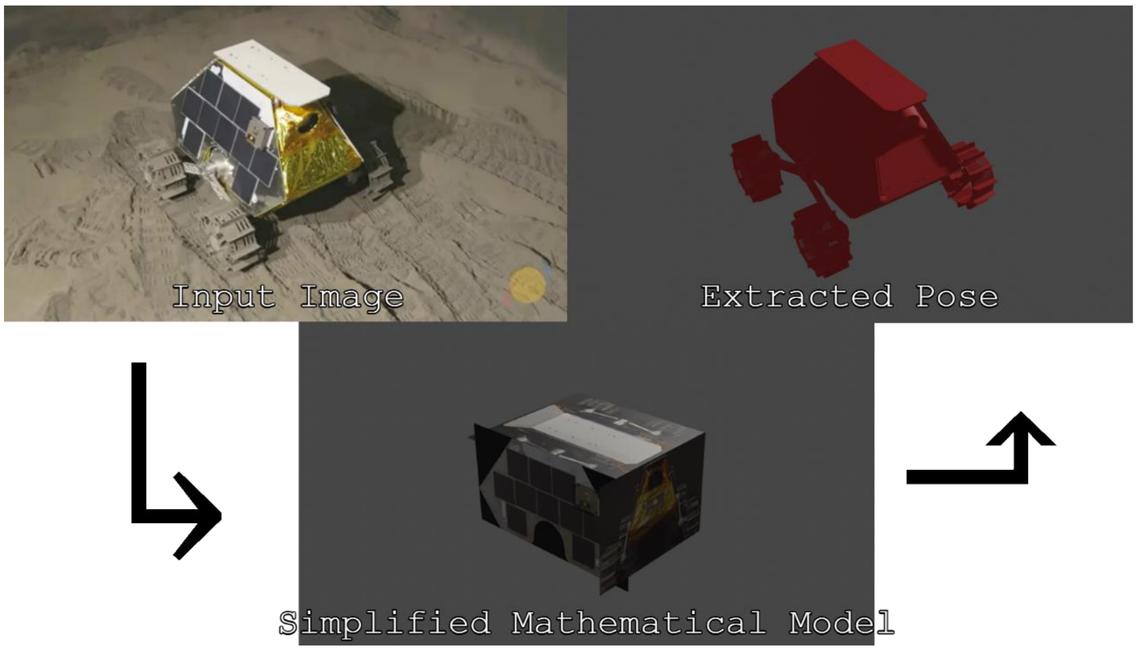


Figure 4: ORB Pose Extractor Visualization

This process worked by itself, but it was fairly inaccurate. In particular, even if a rover was detected successfully, slightly incorrectly-positioned feature matches would sometimes cause immense positional or rotational error. To help remedy this, the query image is re-projected as a plane into 3D space according to the detected transformation, rendered, and the entire pose extraction process begins once again on this new query image. This is iterated recursively until the new pose is sufficiently close to the last pose, resulting in the most accurate transformation the system can currently detect. Finally, this pose is cleaned, the offsets for the correct detected panel are applied, and the pose is fed into ROS.

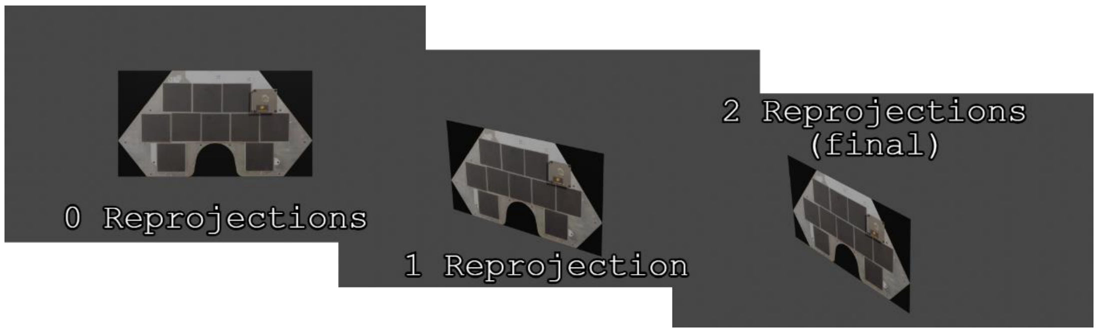


Figure 5: Reprojection Visualization

While this works by itself, MaskRCNN makes the process a lot faster and increases its accuracy.

MaskRCNN is a mask segmentation model with a backbone made of Feature Pyramid Network (FPN) and RESNet101. The idea behind this is a model which can identify the pixels of an image which are part of an object of interest, in our case a rover, and isolate it from non-rover pixels, generating a segmentation mask. Doing so allows us to reduce the size of the image our feature detector needs to inspect for possible features.

RCNN (Region-Based Convolutional Neural Network) is a neural network which modifies the older CNN model for image segmentation tasks. An RCNN breaks an image into region which it then attempts to classify as different objects. This new model lends itself better to object detection, whereas CNN is better for general image classification.
The Faster RCNN model, a modification of the older Fast RCNN, is comprised of two stages. In the first stage, the Region Proposal Network (RPN) proposes candidate bounding boxes, which essentially means it generates bounding boxes it thinks could contain the requested object. The second stage is essentially Fast RCNN. It extracts features using a RoI pool from each candidate bounding box, then performs classification and bounding-box regression. To increase the speed of the model from Fast RCNN, the features, which are used by both the first and second stage are shared, rather than being kept separate as they are in Fast RCNN.

MaskRCNN behaves in a very similar way to Faster RCNN. The first stage is the same, but the second stage has the mask output added. This branches off and runs in parallel to the portion of the model that generates the classification and bounding box. There are a few other additions needed for this to work as well as it does. One of these aspects is to modify the loss function to take into account the accuracy of the segmentation mask. To calculate loss for the MaskRCNN model, we use the equation $\text{Loss} = \text{Loss}_{cls} + L_{box} + L_{mask}$. In this case, L_{cls} is classification loss, L_{box} is loss related to the bounding box, and L_{mask} is loss related to the mask. L_{mask} is defined using average binary cross entropy loss, which avoids competition between various masks in the image by decoupling mask and classification.

In MaskRCNN, the masks are a spatial representation of our object, the rover. This is extracted using our pixel-to-pixel correspondence and prevents the loss of spatial dimensions that happens when a representation is collapsed into vector, which typically happens with bounding boxes and class labels. The RoIAlign layer was created by the creators of MaskRCNN to avoid this misalignment between RoI and the extracted features. RoIPool extracts a small feature map from RoI extracted during the first stage via quantization which introduces some misalignment which researchers found to have a major impact on mask accuracy. To avoid the misalignment that occurs as a side effect of RoIPool, the creators of MaskRCNN instead introduce a layer they call RoIAlign. RoIAlign avoids misalignment by removing the quantization stage, instead using bi-linear interpolation, where four regularly sampled locations in the current RoI bin are used to divide more accurately into bins while still keeping the alignment.

The two major sections of the model are the models used in the head and backbone. The backbone for the MaskRCNN model is a combination of FPN and RESNet101, which the researchers found had an increase in speed and accuracy. The head of the model is an extension of the head from Faster RCNN. The main difference being the addition of a branch which works in parallel with the existing outputs of class and bounding box to output a mask as seen in figure 6.

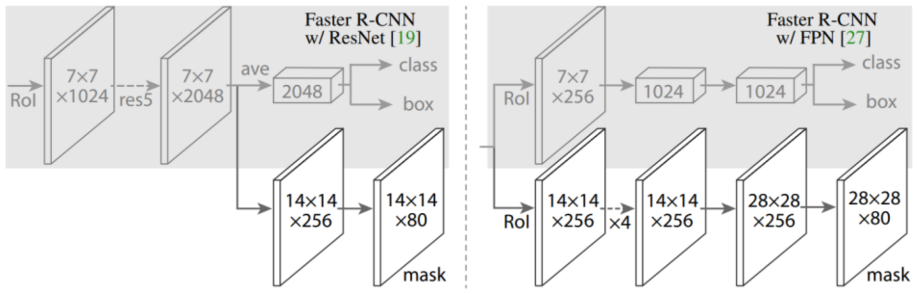


Figure 6: Architecture of the head used in MaskRCNN

In our project we incorporated a prewritten version of this model (license allowed use) with some modifications to correct a few errors and make it easier to run. The dataset was an annotated version of one provided to our team by Lunar Outpost. The annotations required a relatively tight outline of the rover, which was created using VGG Image Annotator and resulted in images similar to that seen in figure 7. The source code required to load and run the trained model was then added to the repository and the model was added to our main file.

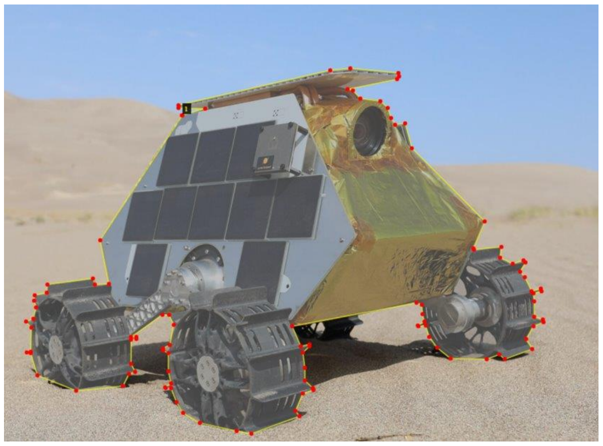


Figure 7: Annotated image of Lunar Outpost rover

VIII. Software Test and Quality

- Unit tests from the ground up, including tests for the matrix and pose math, rover 2D detected position within the frame, extracted affine transformation (no longer used), etc.
- Also built a 3D visualization suite to see and sanity-check these results in real-time.
- Once the systems were built, we also tested features and developed robust math that was double-checked through creation of an ArUco-covered rover analogue (actually just a small cardboard box).

- Human-based sanity checks and double-checks of all systems, as we're working a lot on the R&D side of things and have tried a lot of different approaches throughout the semester.

Everything in our system was extensively unit tested from the bottom up during development, and the team made sure to only add working, tested, and well-documented code to the final pipeline.

IX. Project Ethical Considerations

As with all engineering projects, the ethics of our project and its ramifications had to be considered thoroughly. The three main considerations we had to keep in mind throughout the development process of our pipeline are the stability of our code, the rigor of our test suite, and the royalties of the algorithms implemented in our code.

The stability of our product was the most important of the ethical considerations for us to uphold. Our pipeline, and the information extracted from it, would be used to make autonomous decisions for a rover navigating the surface of the moon. Due to the extreme inaccessibility of the surface of the moon, we had to ensure our software would always be outputting true, useful information. If not, we run the risk of losing a rover, sending it far off course. Not only would that be a huge financial loss to Lunar Outpost, but it would mean losing expensive payloads provided by other organizations and risking the accuracy of every experiment onboard. To prevent situations like this from happening, we implemented several minimum thresholds for valid data extraction, and worked to ensure either data was verifiable and redundant enough to be used, or not returned at all. We also designed our entire pipeline around expecting data not to be returned by default. This was a sensible design choice, as a rover is very unlikely to be in frame every second while navigating the surface of the moon. Reflecting this in our pipeline made for a stable design choice.

Testing was also a massive consideration for our code. Because we won't be able to deploy full systems tests on the moon, we had to ensure our pipeline could withstand all the conditions of space, as well as perform accurately and efficiently. To do this, we were provided with a few image sets, created by Lunar Outpost, to validate our program on. We were also provided with scale-accurate 3D models and dimensioned panel schematics to help ensure the accuracy of our detector and extractor. We broke much of our testing down into these two categories for detection and extraction. First, we established an automatic test suite that would apply our detection algorithm to every image in every dataset, and compile accuracy and timing information for each set. This allowed us to highlight parts of the dataset that weren't accurately registering as a rover, as well as find out which frames took the longest to identify and gave us an idea of the general speed and accuracy our pipeline could achieve. Once we had the geometry extractor done, we also began establishing a plan to generate accurate, simulated space photos using Blender. Blender's advanced camera rendering would allow us to mimic the exact type of camera onboard Lunar Outpost's rovers and would give us the ability to render frames with precise rotation and translation values, which could then be compared to our extraction tool's output for validation. We would even have been able to use textured sky and ground models to simulate the surface of the moon and the stars above. Unfortunately, because so much of our feature detection relied on the textures of the rover, and the provided model we had did not contain any textures, we weren't able to make this dataset. In the future, if Lunar Outpost ever does create a textured model of the MAPP Rover, they themselves would be able to conduct full-scale simulated space tests easily.

Our final large consideration involved royalties. Much of the code in OpenCV's Feature Detection Library is copywritten, using algorithms that are patented by other professors and organizations. In particular, "Scale Invariant Feature Transform" (SIFT), and "Speeded Up Robust Features" (SURF) were two feature detection algorithms we were not able to make use of, as they require royalties to be paid to the developers of the algorithms. OpenCV Does provide a patent-free alternative, known as "Oriented FAST, Rotated BRIEF" (ORB). We chose to use this feature detection algorithm, as it was not only more optimized for handling RANSAC and brute-force matching, but it had no copyright issues, and did not require us to build OpenCV from source to gain access to it.

X. Results

Full-stack testing so far has yielded existence proofs of the potential viability of the system but has not yet been accurate enough to warrant deployment on the actual rover. Given the scope of the project, and the timescale we developed it on, we are happy with the product overall, and believe that it would only take simple, if not time-consuming, tweaks to get it into a perfect functioning state, most of which involve rebuilding OpenCV from source to enable GPU acceleration and hand-annotating images of the rover for the masking software to be able to detect all sides.

Our final product can generate image masks for a wide angle-range of the right-side panel. This was the easiest panel to begin training the neural network on and was the most prominently featured panel in the dataset, which is why we chose to prioritize it. This image mask can identify multiple instances of rovers in our image and is used to significantly speed up the feature transformation/matching process.

Our extractor can calculate full Homographic Transformation Matrices indicating the rotation and translation of a rover with respect to the camera, within an acceptable, floating-point error margin. It can also extract these values as rotation and translation vectors, rather than a 4x4 matrix. The math is perfectly sound, having been extensively tested on a simpler multi-ArUco-based system, and thus the only shortcomings here lie in the visual pose extraction itself.

Our detector can reach 70% detection accuracy or greater, by implementing recursive image sharpening to the feature detector. This only includes having pure templates for the left and right panels, and does not include templates for the front, top, and back. The performance for this model hovers at around 5fps, though this would be massively increased by GPU acceleration, and with the time to build OpenCV from source, and optimizing it to take advantage of the rover's onboard GPU would scale this up by a factor of 10 or more. If utilizing a direct, high-exposure time feed from the onboard camera, we would also be able to do away with the recursive image sharpening part of the algorithm, as frames would be less blurry and more precise, which would allow for greater accuracy scores and faster runtimes, as well. We also have yet to test the performance increase provided by the mask on a large-scale dataset, so the true maximum performance and accuracy have yet to be established.

XI. Future Work

For future work on our project, we would suggest expansion in several areas. To begin, the overall program has currently been developed for a limited number of the rover's sides due to time constraints so the next step would be to expand it to encompass all the requested angle from the client. In addition, signal smoothing across consecutive frames and discarding or outlier poses would make the pose extraction more stable.

In terms of future work for the MaskRCNN section, the dataset for training needs to be created for all sides and a new model trained based on that. In our current setup, the training was on a laptop with relatively low processing power so training on 200 images took around 30 hours. However, research indicates that with a better system (8 GPU), the model can train in around 32 hours with 35 thousand images. Another aspect to the expansion of the dataset includes modifying the training images to have different backgrounds, lighting conditions, examples without a rover, along with many other modifications to make the model more robust.

The last major aspect for future work would be to improve the time required for the program to run. To do this, the ideal solution to start would be to make use of the GPU, rather than running it on the CPU as we are doing now. This requires recompiling OpenCV on compatible hardware, such as the Nvidia GPU, in addition to possibly using OpenCL or OpenGL.

XII. Lessons Learned

We learned many lessons over the course of this project. Since the project was highly focused on researching different methods and tools that we could combine into a functional solution, we learned how important it is to leverage previously created tools to speed up and improve the creation process. To merge these different tools (developed by us or by a 3rd party), we found it was important to keep a modular structure that would allow us to inject and extract code segments simply and elegantly without having to redesign the structure. This allowed us to easily test an idea, quickly evaluate its viability, and then either discard it for another approach or proceed with it as part of the system. Such rapid-fire testing showed us how much can be accomplished in such a small amount of time and how valuable modularity is. Finally, with so many modules coming in and out, you can easily get lost in the code, and the solution is documentation. While the organization of the code makes it easy to figure out where you are in the code, we found that documenting the code as it's written adds clarity no amount of organization can give. Rather than just giving the 'what', it can also give the developer the 'why', which is often lost over time. Having code that doesn't need documentation is great but having code that doesn't need documentation yet does have documentation is better.

XIII. Team Profile

Julia Harvey

Role: Snack Consumer

Julia Harvey is a senior at the Colorado School of Mines majoring in the Computer Science Robotics and Intelligent Systems track.

Derek Suzumoto

Role: Meme Guy

Derek Suzumoto is a senior at Colorado School of Mines, majoring in General Computer Science Applications.

Ethan Brucker

Role: Long Hair

Ethan Brucker is a senior at the Colorado School of Mines, majoring in Computer Science with a focus on Robotics and Intelligent Systems, minoring in Computational and Applied Mathematics, and pursuing a Master's in Computer Science (Also RIS).

Philip Belous

Role: Emotional Support

Philip Belous is a senior at the Colorado School of Mines, majoring in Computer Science with a focus in Robotics and Intelligent Systems and pursuing a Master's in Quantum Engineering – Software.

References

Lunar Outpost Robot Swarm Co-Localization Project Proposal - https://cs-courses.mines.edu/csci370/FS2022F/Proposals/LunarOutpost2.pdf

Definition of Average Binary Cross-Entropy Loss - https://insideaiml.com/blog/BinaryCross-Entropy-1038

Research paper on Mask RCNN - https://arxiv.org/abs/1703.06870

Real-Time Detection of Textured Objects - https://docs.opencv.org/3.4/dc/d2c/tutorial_real_time_pose.html

Dense 6D Pose Estimation - https://www.arxiv-vanity.com/papers/1902.11020/

Image Segmentation with Watershed Algorithm - https://docs.opencv.org/4.x/d3/db4/tutorial_py_watershed.html

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Mask	*Extracts a region of interest (ROI). In our case, the region which contains the rover.*
Mask Region-Based Convolutional Neural Network	*A model which uses region identification to place a mask over the desired object in an image.*
RoI	*Region of Interest. In our case, a section of the image which may contain the object we are seeking.*
ROS (Robot Operating System)	*Tool suite used to standardize robot software development*
Average binary cross entropy loss	*Loss calculated using the negative average of the log of corrected probabilities used for classification.*

Backbone	*A term used to refer to the network used to extract features.*
Head	*The portion of a network that accepts input data. In our case, the model accepts RoIs.*
Pixel-to-pixel alignment	*The alignment between the pixels of two images. In our case, typically between RoI and its features.*
Fast RCNN	*Model which improved upon RCNN by creating a convolutional feature map from one iteration over the image.*
Region Proposal Network (RPN)	*Convolutional network which predicts object bounds/regions.*
Bounding Box Regression	*Technique to refine/predict bounding boxes.*
Spatial representation	*Representation of object in relation to an image.*
Quantization	*Process of mapping infinite, continuous values to a discrete set of values.*