UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

INCORPORATION OF DEPTH FEED FOR PROTOTYPE ROVER VIDEO AND OTHER USABILITY IMPROVEMENTS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

in

MECHANICAL ENGINEERING

By

JOSEPH DAL SANTO
Norman, Oklahoma
2019

INCORPORATION OF DEPTH FEED FOR PROTOTYPE ROVER VIDEO AND OTHER USABILITY IMPROVEMENTS

A THESIS APPROVED FOR THE
SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. David Miller, Chair

Dr. Chung-Hao Lee

Dr. Kuang-Hua Chang

Acknowledgements

I would like to thank Dr. Miller, my parents, and both parties' considerable patience in supporting my completion of this thesis from a thousand miles away. I would also like to thank my committee members, Dr. Chang and Dr. Lee, and Dr. L'Afflitto, who served on my committee until he left OU.

Table of Contents

List of Figures

List of Tables

Abstract

NASA has been mandated by the US government to return to the moon within the next five years, meaning that functional technology for lunar exploration must be developed and tested in order to reach that goal. While most lunar exploration vehicles can be operated using a supervisory control scheme, utilizing a direct human control scheme would allow missions and navigation be more robust to unforeseen events and obstacles. Because any human-teleoperated lunar mission would involve significant time delay, low video resolution, and low framerate, video feeds need to be augmented to grant human navigators additional information about the rover's surroundings. This work proposes two new video feed options for an existing prototype lunar rover that utilize an Intel D435 depth camera to provide a depth image stream of the rover's surroundings and an augmented reality (AR) depth overlay on a mono RGB video stream. The video feeds proposed in this work have been proven to function within the rover's current architecture, and can correctly simulate the conditions a navigating user would experience when attempting to teleoperate the rover at a lunar distance. This work also proposes usability improvements to the User Heads-Up Display based on previous test feedback to reduce onscreen clutter for navigating users, and a data visualization program to reduce data analysis time for testers and increase clarity when presenting data.

Chapter 1

Introduction

Exploration and discovery are hallmarks of the human experience that stoke our collective and individual creativity and force our species to develop unprecedented solutions to overcome the barriers that prevent us from journeying into the missing pieces of our maps. Previous exploratory voyages during the first and second Ages of Discovery involved charting the unknown reaches of our own planet, but in the current day exploration on Earth is largely limited to the depths of the oceans, leaving the vastness of space as the primary unknown for humans to strive towards visiting [1]. While pursuing space travel may seem like a frivolity in the face of the many more terrestrial problems facing humanity, space exploration efforts are historically proven to have profound benefits for the entire world. Space exploration expands and diffuses global scientific knowledge to produce new, innovative technologies, expands the economic sphere of humanity into space as advancements create opportunities for private companies, and promotes collaboration and diplomacy between world powers in order to overcome the daunting challenges of space travel for the benefit of humanity. More abstractly, space exploration also serves as a means to inspire future generations of scientists and allows for deepened or new

understandings of humanity's place in the universe [2].

1.1 Rovers and Humans as Exploratory Teams

In order to pursue space exploration in a timely and effective manner, unmanned exploratory rovers will need to be utilized to maximize humanity's exposure to extraterrestrial objects while minimizing risks to human life. While human extraterrestrial explorers are able to adapt plans in the face of uncertainty to prevent mission failure, utilizing humans for space travel is dangerous and expensive. During space travel, astronauts risk being exposed to cancerous radiation, changing gravity fields which cause bone density loss and cardiovascular issues, and intense isolation on long journeys which can endanger their mental state [3]. Additionally, the technology and extra space required to keep a person healthy and comfortable during space travel dramatically increases the cost of the voyage. Utilizing unmanned rovers for exploratory missions circumvents this danger to human life posed by space travel, reduces the cost of missions by allowing engineers to remove life support systems and unnecessary space, and allows scientists to routinely perform delicate tasks with the precision provided by a programmable explorer. However, full autonomy throughout the entire course of a mission is currently not feasible due to the unpredictability of space exploration, so human navigators are still necessary at certain points throughout the mission. Combining the robustness of human decision making with the precision and ruggedness of an unmanned rover allows exploratory missions to be developed much more quickly and at a fraction of the cost, making space travel a more attractive and tangible option for funding.

Due to the importance of effective low-bandwidth rover navigation systems for future space travel, it is necessary to design control systems to be paired with rovers

relying on low-bandwidth communication to allow for their effective control by human navigators. One of the largest barriers to the effectiveness of potential control systems is time delay present in extraterrestrial communication, roughly 1.3s between Earth and the Moon and up to 21 minutes between Earth and Mars [4]. Many current rover navigation systems circumvent this time delay issue by using supervisory control schemes, which involve controlling remote vehicles using initial, human-given commands and closing the control loop with the rover using input from sensors the rover has access to and the environment the rover is in [5]. Supervisory control is one of the control modes used by NASA for piloting their currently deployed Mars rovers, and is additionally utilized on many other teleoperated rover endeavors such as Carnegie Mellon University's Robotics Institute, which controls a prototype lunar rover using a human navigator requesting desired travel paths and a "safeguard" program which maps surrounding terrain and prevents the rover from pursuing dangerous paths [6]. NASA's currently deployed Mars rovers also employ a form of highly delayed direct control, where explicit commands are sent to the rover for it to follow, and the success of these commands is then verified [7]. Autonomous rover control systems, like those used in older systems such as Carnegie Mellon's Ambler rover [8] and the NASA Jet Propulsion Laboratory's BISMARC rover control system [9], seek to remove the time delay consideration from the day-to-day operations of rovers entirely by allowing them to continuously act on its own in pursuit of a more complex goal. While these methods do allow for teleoperation of a rover subjected to time delay, they are not designed to allow for a human navigator to continuously pilot a remote rover under direct control. Because human guidance is still necessary for certain elements of unmanned exploratory missions, designing a rover control system which supports a human navigator to improve their ability to quickly and safely pilot a teleoperated rover would expand the kind of tasks a rover

could complete and would make it more robust to handling navigational situations where the quick, nondeterministic decision making of a human navigator is helpful.

The key component of a direct control system for a teleoperated rover is the vision system, which must allow for navigational observation rather than just scientific or artistic observation. These vision systems need to be designed to maximize the information a human navigator has at their disposal while minimizing cost, power use, and bandwidth consumption to account for the limitations of extraterrestrial travel. While a simple monocular video feed would be the solution which minimizes impact on the rover's performance, providing additional information through tools such as video effects, overlays, and Heads-Up Displays is essential for task performance by human navigators or else they are more likely to fail the task they are being asked to complete [10]. One promising video effect involves conveying spatial information to users through the use of a pair of stereo cameras, whose images can be fused together to create a 3D effect. Use of stereo cameras for remote teleoperation is shown to increase speed at completing more complex tasks [11]. This finding is also suggested by previous testing with the OU Intelligent Robotics Laboratory's prototype rover, shown in Figure 1.1, in a low bandwidth, high time delay situation. On average, when being piloted using the rover's current stereo vision system, navigators were able to drive through courses more quickly than when piloting using a monocular view. These results suggest that providing more robust spatial data, such as generating a depth map using a 3D camera or stereo vision system to show on its own or overlaid on a monocular video feed, could improve drivers' times even further.

To this end, the goal of this thesis is to propose new additions to the rover piloting system which utilize depth visualization techniques and data from previous testing to

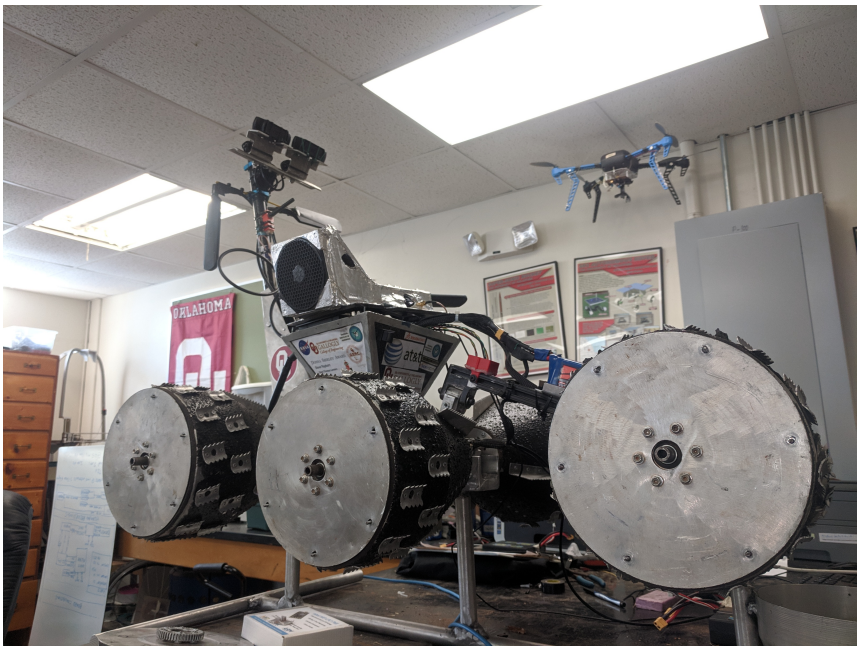

Figure 1.1: Prototype lunar rover used for teleoperation testing.

- allow the rover to transmit depth data to the user with minimal computational latency, and
- make improvements to the rover Heads-Up Display to reduce unnecessary information presented to users during testing.

1.2 Organization of Thesis

This thesis will propose a video feed improvement for human navigator control when directly teleoperating a low-bandwidth prototype lunar rover by expanding the options available for video feed recording and increasing usability of the rover's interface and functionality. Chapter 1 gives a justification for the study's relevance and outlines the primary goals that the thesis will achieve through modifications and improvements to the prototype rover's physical and software architecture. Chapter 2 is a literature review, which breaks down information on vision systems incorporating

video effects and overlays for vehicles as well as methods for depth data generation. Chapter 3 summarizes the previous work that has been done in this area in the lab involving quantitative and qualitative analysis of human teleoperation of the rover under consideration using both mono and stereo video feeds. Chapter 4 will explain the changes made to the vision system to accommodate the transmission of depth data to the user, and initial tests with the new feed to ensure functionality. Chapter 5 will explain the other usability changes made to improve the functionality of the rover, namely, to the HUD and data logger. Chapter 6 will summarize the thesis, and offer possible future testing and questions that could be explored.

Chapter 2

Literature Review

The following section presents research conducted into usability improvements for remotely piloted vehicles like the rover that the OU Intelligent Robotics Laboratory is using. This research primarily focused on improvements to the vision subsystems for the rover, looking at the effects of video overlays on user performance and methods for generating depth data using video feeds.

For the purpose of this research, video overlays are defined as any modification made to video or image data transmitted to an end user designed to improve their understanding of the elements in the data being given to them. Video overlays are an effective tool for improving user comprehension of video or image data, and allow users greater quantitative understanding of an image or video feed than unmodified video. Currently, the rover video feed employs a simple video overlay involving a Heads-Up Display to convey data about the rover's speed, position, and status.

2.1 Depth Overlays

2.1.1 Applications of Depth Mapping

The generation and use of depth data for navigation and orientation is relatively common in remote vehicle operation, and is not a new concept; successful collision and avoidance measures using stereo vision were first demonstrated by the NASA Jet Propulsion Laboratory's research robot in 1977 [12]. While depth data can be used to visualize potential navigational information, it is primarily used for autonomous navigation. The high-level system for navigation using depth maps utilized by the University of British Columbia's *José* autonomous robot is relatively common, involving using a stereo camera to generate a depth map which is then searched using a path-planning software to generate a potential path for the autonomous vehicle to follow [13].

Figure 2.1: *José*, the University of British Columbia's mobile autonomous robot. [13]

This general high-level system is then specified to fit the unique aspects of the environment being navigated. Stereo camera hardware and computers used for

processing vary and have significantly improved since *José's* creation twenty years ago, but the most unique changes for these kinds of autonomous navigation systems come from the methods used to generate and parse the produced depth maps. For example, *José* uses a trinocular array of wide-angle (90°) cameras to capture images, which are then used to generate depth maps with less pixel mismatching errors by creating traditional stereo maps between the top and left/right cameras and adding the two images together. The produced depth maps are then projected to the ground to generate top-down map views of *José's* position to allow for path planning [13].

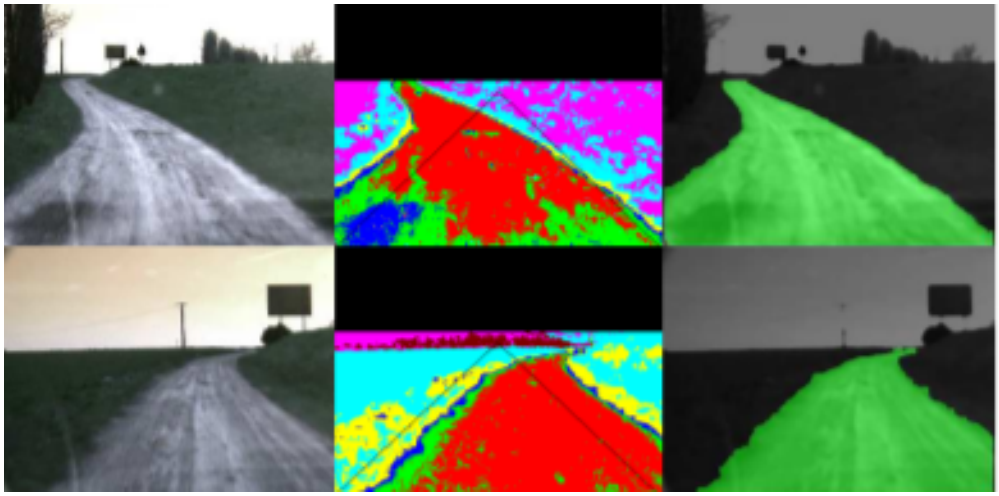

Figure 2.2: Detection of a traversable road in an image, highlighted in green, from [14] using their "free space" calculating algorithm. This algorithm calculates the disparity of objects in an image, uses this disparity to determine what areas of the image are "traversable" by a vehicle, and then uses pixel hue to group together the portions of traversable space to see which areas are likely to constitute the road, i.e. are mostly contained in the triangle placed at the bottom of the image assumed most likely to contain the road.

The same principles of using depth mapping for autonomous indoor mobile robot navigation by *José* have also been applied to outdoor navigation. One example of this presented at IEEE's 2007 Intelligent Vehicles Symposium involves using stereoscopic pair of cameras to generate a *v-disparity* map of a given image, which takes

a generated disparity map and calculates the number of pixels along a line in the disparity map with the same disparity. It then uses this v-disparity map to calculate the "free space" within a given image that corresponds to traversable ground and then analyzes this space using a color segmentation algorithm which breaks the free space in the image into color clusters that are then analyzed to see if they lie mostly within or outside of a triangle placed at the base of the image designed to represent likely road positioning. Any clusters that lie mostly within the triangle are classified as road, while clusters mostly outside of the triangle are classified as non-road [14]. This algorithm is demonstrated above in Figure 2.2.

The autonomous navigation method proposed by Aubert et al in [14] was expanded upon to function in real-time structured environments, namely highways, to allow for real-time obstacle detection up to 55 meters away. The new detection algorithm weights the v-disparity values used in the creation of the v-disparity map, as well as incorporates a tracking algorithm to monitor the dynamic state of objects being analyzed by the algorithm. The result of this algorithm allows for the effective detection and tracking of potential obstacles and objects on the road plane with a 99% correct detection rate [15].

While these results are impressive, they are primarily applicable to structured environments like roads, which limit the amount of potential area that an algorithm will need to analyze and allow for more computationally intensive real-time analysis. Depth mapping obstacle detection has been extensively applied in unstructured environments, like natural areas and parking lots, as well, and has been specifically applied to teleoperated rovers due to the ubiquity of unstructured environments as the types of areas rovers navigate. NASA's Jet Propulsion Laboratory proposes a low-computation disparity-matching algorithm in [16] which reduces computation time by reducing the resolution of the source images before they are analyzed and

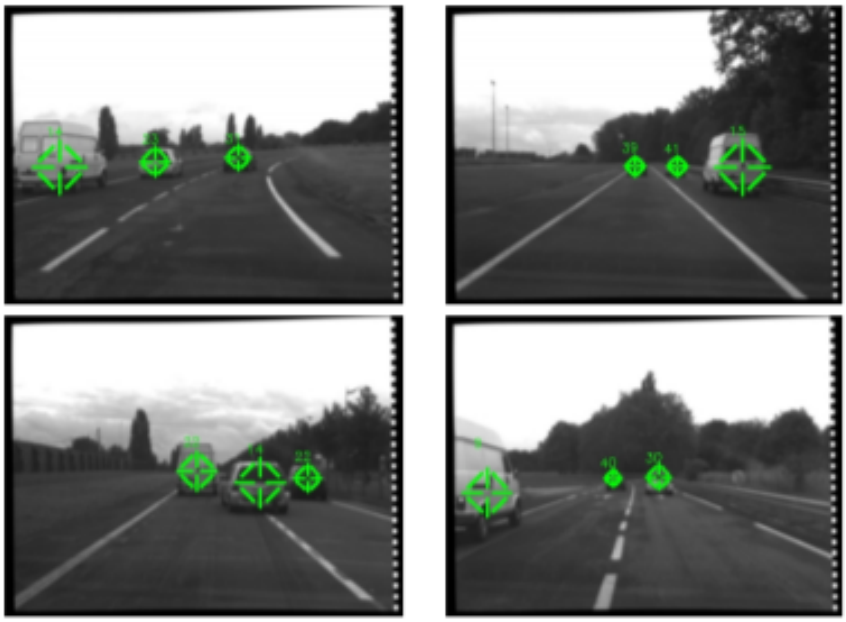

Figure 2.3: Example output from [15] showing their algorithm applied to highway images from a real database; vehicles are designated in green, and their distances from the navigating vehicle are shown faintly above the found vehicles.

rectifying the images to reduce the complexity of the pixel-matching search. This algorithm is used to generate a disparity map that is added to a local map of the area around the rover, which is then navigated to allow for the rover to reach a specified destination point using a route planner called GESTALT [16]. More modern navigational and positioning algorithms like the one utilized by DLR-RMC Robotic's Light Weight Rover Unit (LRU) use more processing power and more efficient and robust Stereo Localization and Mapping (SLAM) techniques, like the creation of a high-resolution point cloud map to use for self-localization and route planning, but the algorithms used to generate the disparity map of the area around the rover remain largely the same due to computational restrictions [17].

Inspiration and high-level ideas can be taken from the applications of depth mapping in autonomous navigation, but these systems cannot be used directly for

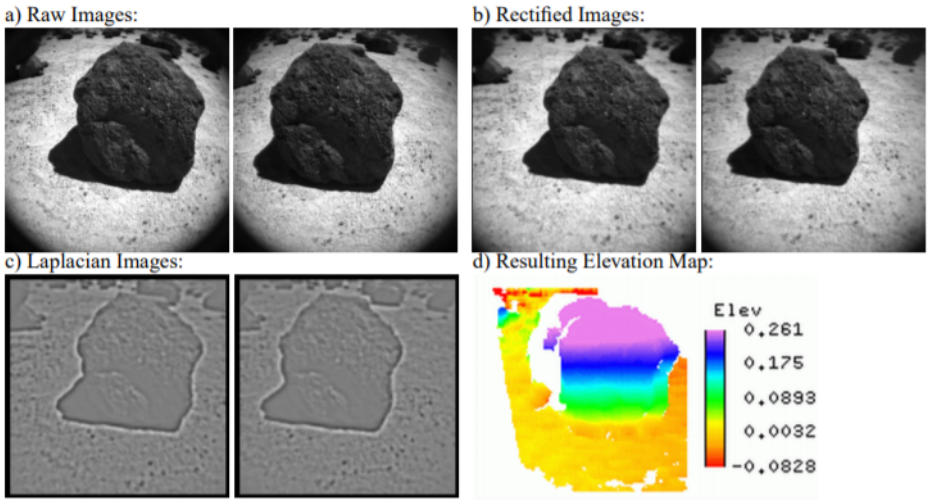

Figure 2.4: Stereo image processing performed by JPL rover in [16] to determine traversable terrain using GESTALT route planner. Similar stereo processing is performed by the LRU rover in [17] for their application.

conveying depth data to a human navigator in a meaningful way. These applications are designed to present data to human controllers for use in supervisory control over the vehicles at most, while the actual decision making and navigation is conducted by the vehicle. The robust path-planning of remotely controlled rovers is unnecessary, as said planning will be conducted directly by the human navigator. This computational advantage is countered by the necessary inclusion of a human-understandable video feed into which the depth data must be incorporated. As such, the use of a relatively simple disparity map generation algorithm, like the one used in [16], is necessary to ensure the inclusion of depth data into the human navigator's video feed doesn't create undue computational latency that would further complicate navigation.

2.1.2 Depth Mapping and Stereo Vision

Before the rover hardware and codebase could be modified to allow for streaming depth data, it was first necessary to gain an understanding of the methods used to generate depth data. Generating a depth map from images uses two images, often from identical cameras, and relies on the principle of triangulation. Triangulation allows for the determination of the 3D position of a point in an image by utilizing the fact that there must be a line which passes through both the point and the center of projection for the camera capturing the image. By using two images, the depth of a point can be determined by finding the intersection point of each image's lines and calculating the disparity between where these lines intersect with the image plane. This is shown visually in Figure 2.5.

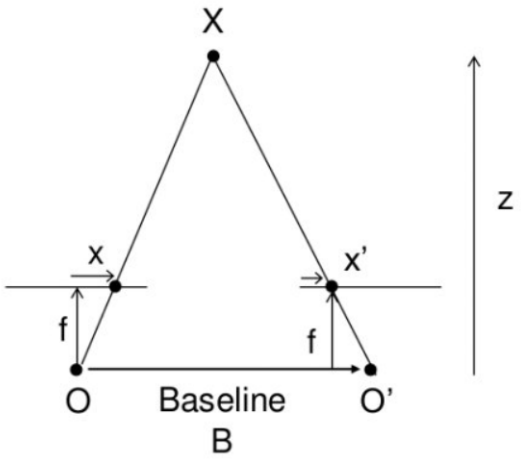

Figure 2.5: Visualization of principles used to extract depth information from two images of the same scene. [18]

x and x' represent the positions of the pixel whose depth data is being extracted from the image, both corresponding to the point X in the real world. The depth of the point in the scene is inversely proportional to the difference between these values. Thus, the closer x gets to being equal to x', the farther away that point on

the image is, and the closer x gets to being equal to $-x'$, the closer that point on the image is. This relationship is characterized by Equation 2.1 below, where Z is equal to the depth of the point, B is equal to the distance between the centers of the two cameras, and f is equal to the focal distance of the cameras. Using identical cameras means their focal distances will be the same, and allows for the simplification shown below in Equation 2.1 where a single value f can be used for the focal distance of both cameras [18].

$$hZ = \frac{Bf}{x - x'} \tag{2.1}$$

Using the above equation, two stereo images can be used to generate a depth map of a scene. However, because the images will have slight variances in perspective due to being from different positions, the position data x and x' for each point in an image will vary in both the x and y dimension, which would require a search of the entire image for each pair of points for successful triangulation. This search could be performed in a single dimension, reducing computation time, if the cameras are perfectly aligned to be coplanar, but this is often impractical to achieve and maintain, especially for a mobile, rugged application like a prototype rover. Despite this, it is still possible to reduce the image search to a single dimension for images from non-coplanar cameras after the images have been captured using a process known as stereo rectification. Stereo rectification involves rotating or warping two images until they all lie on a common image plane [19]. The necessary rotations and translations that must be applied to transform the images being searched can be obtained utilizing the epipolar geometry of the scene being observed and the two images, an example of which is shown below in Figure 2.6.

The epipolar plane shown in Figure 2.6 contains the point in 3D that a pair of pixels correspond to and the centers of both cameras capturing the point. The two

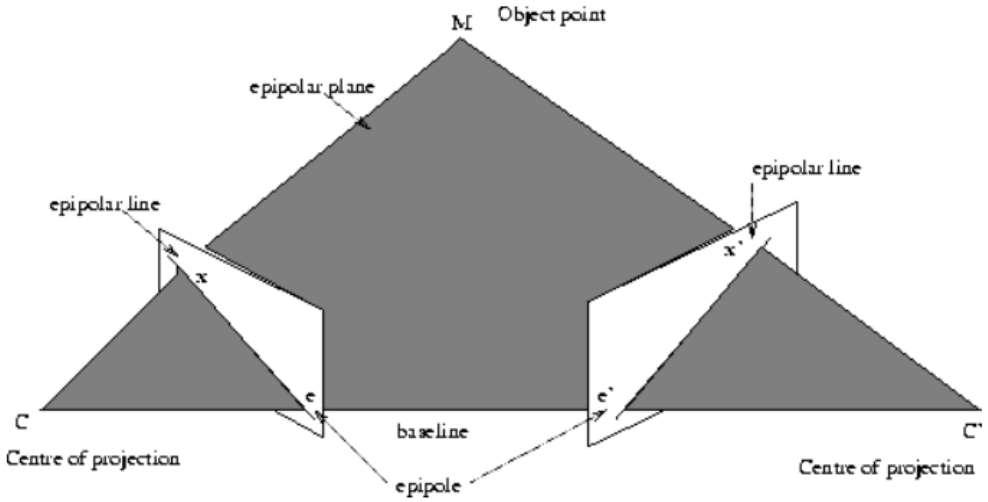

Figure 2.6: Visualization of epipolar geometry present in the capturing of a scene using stereo cameras. [20]

images intersect with the epipolar plane along epipolar lines, which intersect with the pixel coordinates of an object being observed in an image and that image's epipole. All of the epipolar lines of an image intersect with the epipole of that image, which joins the optical centers of the camera with the image plane. Epipoles also represent the position of the optical center of the other camera in the system, allowing the two cameras and their image planes to be joined on the epipolar plane via a baseline [20]. When the epipolar lines of both images are parallel to a horizontal axis represented by the baseline and the vertical coordinates of corresponding points on the images are identical, the image pair can be considered to be rectified and can be searched along a single dimension for coordinate pairs.

Performing stereo rectification on a pair of images requires performing a rotation and translation on each image to make them coplanar. Determining these operations requires a knowledge of the extrinsic and intrinsic properties of each image, with the extrinsic properties representing the position of each image frame in relation to an external fixed world frame and intrinsic properties representing the relation between

camera coordinates and the coordinates of pixels in each image [21]. Here, having identical cameras becomes an advantage again, as the intrinsic properties will be identical for both images. Thus, once the cameras have been calibrated to determine their intrinsic parameters, it is simply a matter of applying the appropriate rotation and translation to both images to orient them in the same fixed world frame. This image pair can then be searched to determine disparity between matching pixels.

2.2 Heads-Up Display Design

Changes to the user Heads-Up Display (HUD) were also considered to improve the rover's video overlay. HUDs are generally fixed information overlays that do not require users look away from their typical viewpoints to see data. HUDs were originally developed for use on military aircraft, but have since become more common on commercial vehicles [22]. While most commercial HUDs are overlaid over a user's direct line-of-sight out of a vehicle, they can be applied effectively as video overlays as well; HUDs are often used in video games as part of the game's user interface to convey information to players about their status. A HUD's capability to deliver users data without needing them to divert attention from their navigational viewpoint makes them an effective, minimalistic tool to convey live information. In a study conducted at Technical University Munich on simulated human teleoperation of satellite rendezvous and docking procedures, the inclusion of a HUD on the navigator's display increased their success at completing navigational and correctional tasks by at least 60% [10].

Including an effectively designed HUD for vehicle operation is clearly beneficial to user navigation, but ensuring that the HUD is not distracting while presenting data in a useful manner can be a difficult balance to strike. A study conducted by

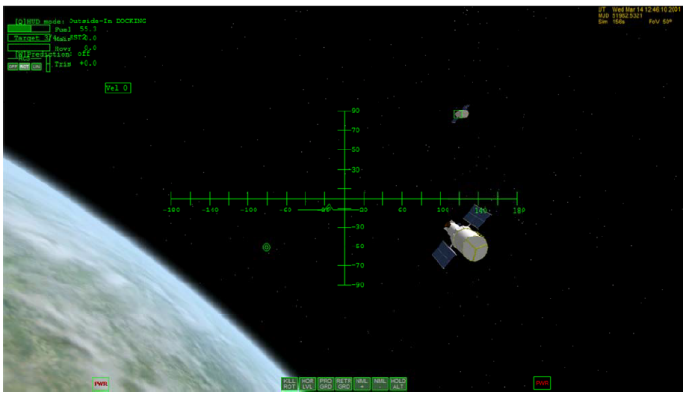

Figure 2.7: HUD used for virtual reality testing of satellite rendezvous and docking procedures [10].

the Honda Research Institute USA designed their navigational windshield overlay to minimize necessary on-screen visuals to ensure drivers stay focused on the road, and prevented the design from obstructing the driver's view [23]. The positioning and style of the information that is chosen to be presented to the user is also important. Technical University Munich also conducted a study on HUD Presentation Principles, which specified that continuous information, like speedometers and position, should be unobtrusive at all times, while discrete information, like warnings, should be eye catching but not overly distracting so as to allow for focused navigation during the display of discrete information. Additionally, as most elements of a HUD will be out of the direct line of the user, the information they present should be rendered as simple shapes designed to suggest information when they can. If this is impossible, the HUD element will necessitate a glance [24].

Chapter 3

Current Functionality and Previous Testing

This chapter presents a summary of the current functionality of the University of Oklahoma Intelligent Robotics Laboratory prototype teleoperated rover as it pertains to the vision and data logging subsystems, as well as the results of previous testing with this functionality that inspired the changes described in the subsequent chapters of this thesis. The overarching goal of the previous work performed using the prototype rover involved discerning the best methods to effectively convey the data necessary for a human navigator to effectively pilot a teleoperated rover in conditions of high latency and low bandwidth, simulated using an artificial time delay and deliberate video compression. Specifically, recent work has focused on ascertaining if there is an advantage gained through the utilization of a stereo video feed as opposed to a mono video feed.

3.1 Current Functionality

3.1.1 Video Subsystem

The current functionality of the video subsystem for the OU IRL prototype teleoperated rover consists of a Hardware component, which encapsulates the cameras on the physical rover and the display TV which is used during testing, and a software component, which consists of the codebase for rover control, the video streaming program used, and the HUD overlaid over the recorded video. Additionally, there is a parallel system set up to record and display video from a fisheye camera that was not used for testing. The data flow through the current system is characterized in Figure 3.1.

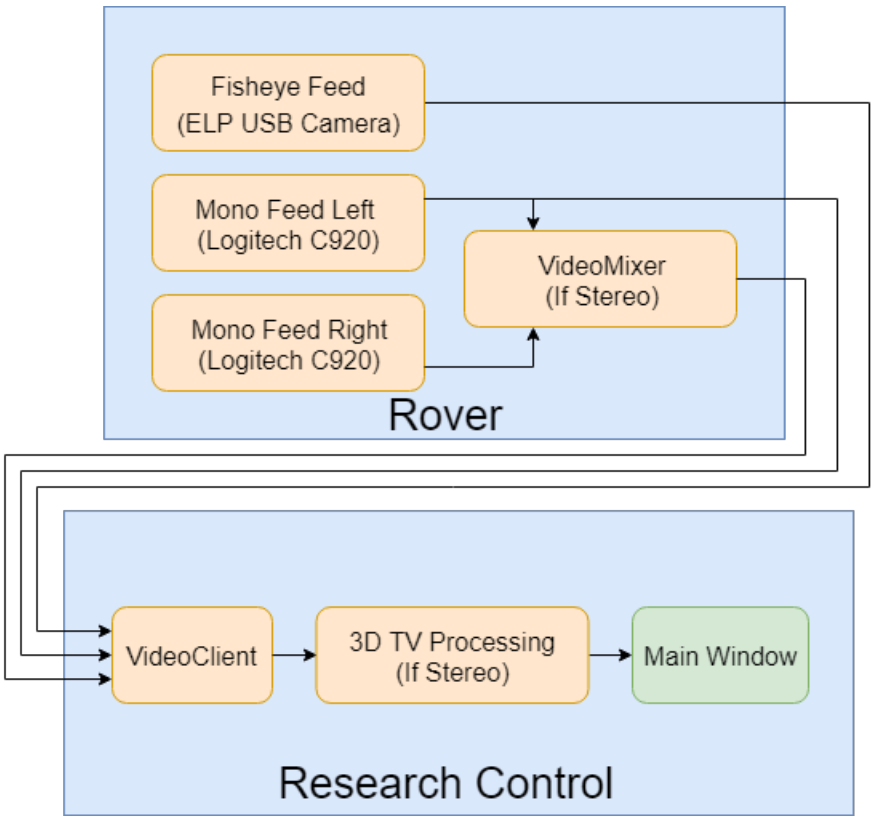

Figure 3.1: Current video pipeline utilized by rover..

Figure 3.2: Logitech C920 Pro HD Webcam currently used for video capture. [25]

The primary cameras used by the rover are a pair of Logitech C920 Pro HD Webcams, which are equipped with a 70.42°x 43.3°FOV and compatibility with Video4Linux, which allows them to interface easily with the rover's onboard computer and video streaming program [25]. Two of these cameras are mounted and aligned in parallel on the rover's mast, allowing them to capture footage of the rover's surroundings simultaneously for the production of stereo video. The other hardware component of the video subsystem is a 55" LG 4K Smart 3D TV, which is used to display the processed video to the navigator. The 3D capabilities of the TV allow for the creation of stereo video from a variety of source video options, including side-by-side images [26].

Video streaming performed by the rover is handled by pairs of video clients mission control-side which receive and display video and video servers rover-side which run sub-processes that handle the actual streaming. The streaming objects, called VideoStreamers, use an open-source video streaming platform known as GStreamer to capture, encode, and process the video data being captured. GStreamer allows the entire video pipeline to be created and defined as a single object by creating and combining a set of video streaming steps using pre-created GStreamer objects, which prevents the need for manually populating buffers unless there is no object for the action wanted [27]. GStreamer also has a library designed for allowing use

of the platform in Qt, the platform used to develop the rover codebase. This library made it easy to include compact video streaming functionality in the rover's control program [28].

The rover's current program is capable of streaming two types of video using its primary cameras: mono and stereo. When streaming mono video, as shown below in Figure 3.3, the rover only uses one of the two cameras in its main camera array. The rover uses GStreamer to send a 2D color video stream of its surroundings to mission control to be used for navigation.

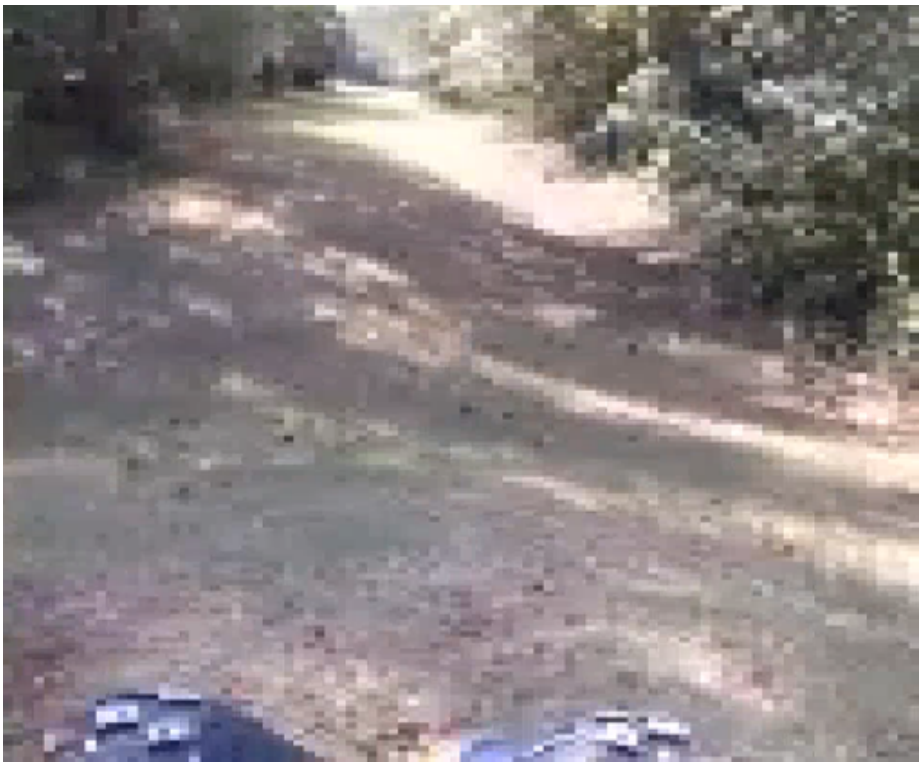

Figure 3.3: Example of mono video feed from rover, resolution = 176x140p.

When streaming stereo video, video streams from both cameras are combined into a single image using the videomixer GStreamer element. Videomixer takes two input sources and outputs a single frame of the two source inputs combined based on user specifications [29]. In the case of the rover's stereo video feed, two parallel

images of the rover's surroundings are sent, and the final image is configured to be both images side-by-side with their width halved, the left camera image left-justified, and the right image right-justified. An example of this is shown in Figure 3.4. Once this video stream reaches mission control, it is converted to a pseudo-3D image using the 3D TV's side-by-side image 3D setting. This allows the video feed to convey a sense of depth to the navigating user while the user is wearing a pair of polarization glasses, which project the same scene to each of the user's eyes from different perspectives.

Figure 3.4: Example of stereo video feed from rover before processing by 3D TV. The overall resolution of the image is 176x140p, but the resolution of the left and right images that are combined to make the overall image is 88x140p.

In addition to the main camera video options, the rover is also capable of transmitting a fisheye view of its surroundings using a small ELP USB fisheye camera

attached to the rover mast. This video feed, shown below in Figure 3.5, is handled much in the same way that the main camera mono video feed is handled, though it has a unique pair of video clients and servers to ensure the program knows which camera to pull from when video is requested. All of these video feeds are transmitted over low bandwidth to simulate the conditions which would be present for a real-time navigable rover stream for the navigating user.

Figure 3.5: Example of fisheye video feed from rover, resolution = 640x480

3.1.2 Tester Interface and User HUD

The generated video feeds are coupled with a control interface seen only by the tester and a user HUD overlaid on the video designed to provide helpful information to the navigating user. The control interface, shown in Figure 3.6, allows the tester

to control what video feed is enabled and characteristics of the current video feed, most notably the simulated time delay on the video feed and user control input. In order to simulate the time delay that would be experienced when piloting a teleoperated rover at a great distance, like on the moon, the control interface allows for the generated video to be artificially delayed at a rate determined by the tester. The tester can also induce a separate latency on the control commands input by the navigating user to pilot the rover. This delay can be modified to ensure the navigation is realistic while still possible by factoring the processing time necessary to generate the video feed being shown and send user commands. The tester interface also allows control of the resolution of the incoming feed and the feed's framerate to better simulate the quality of video feed that a navigating user would need to use to pilot a rover at lunar distances. Other characteristics that can be controlled include the video encoding, bitrate, HUD visibility, and an optional grayscale filter, though these characteristics generally remain unchanged between tests.

The user HUD overlaid on the rover video feed, shown in Figure 3.7, is designed to convey information about the rover's speed, angle, and condition to the user in a minimalistic, largely text-free format using a combination of six widgets. The current widgets operate as follows: widget 1 in the top center shows the rover's current orientation via a compass heading; widget 2 conveys the current time delay on the feed in milliseconds; widget 3 shows both the magnitude and delay of the speed and direction commands the user is sending to the rover; widgets 4 and 5 show the current roll and pitch of the rover's three components, respectively; and widget 6 is a warning schematic designed to alert the user of the motor current draw measured by current sensors on each wheel to indicate if a wheel is stalling.

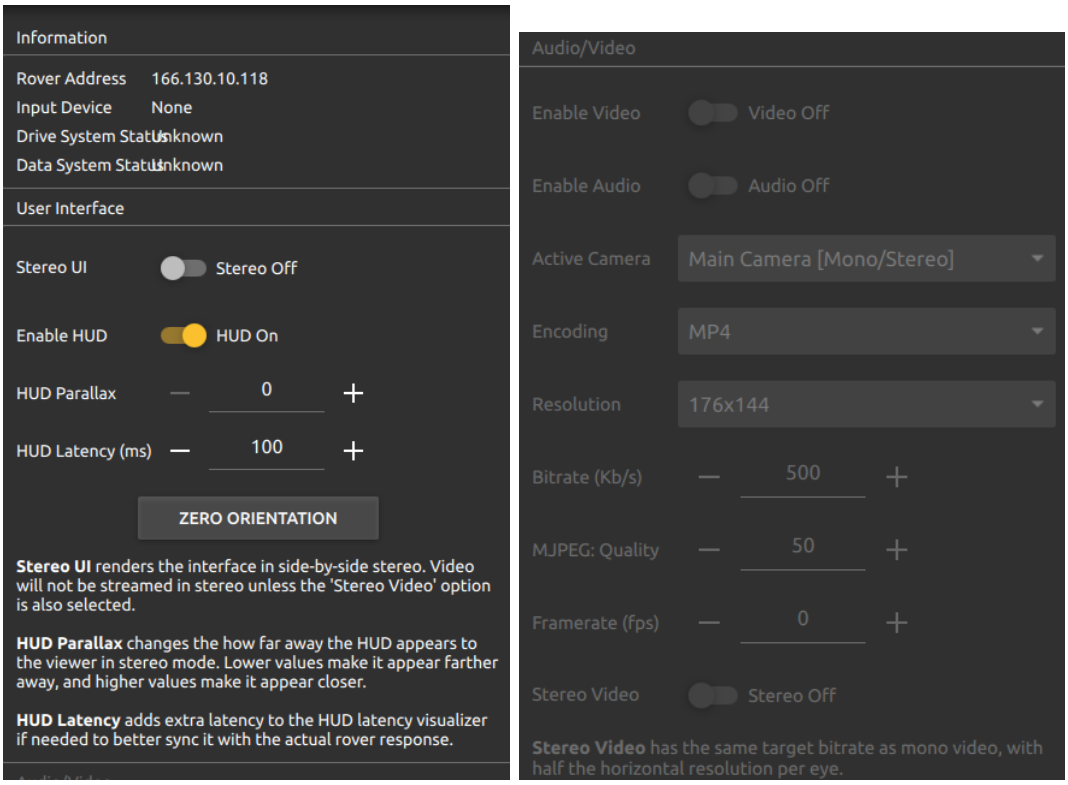

Figure 3.6: Current tester interface used to manipulate video feed characteristics. Allows for control of audio/video streaming, video mode, HUD, video resolution, video framerate, video and control input latency, HUD relative position and latency, and video encoding settings.

3.1.3 Data Logging Subsystem

The rover control program utilizes a data logging subsystem running in parallel to the video subsystem during tests in order to record data about the current set of tests both at specified intervals and when changes in the test configuration occur. The data recording can be enabled by the tester through a comments window that appears concurrently with the video feed and control window, shown in Figure 3.8.

This data is recorded by writing to three comma-separated value (CSV) files, one each for physical data, settings information, and comments from the tester. The Data file contains data related to the rover recorded at a specified interval, including

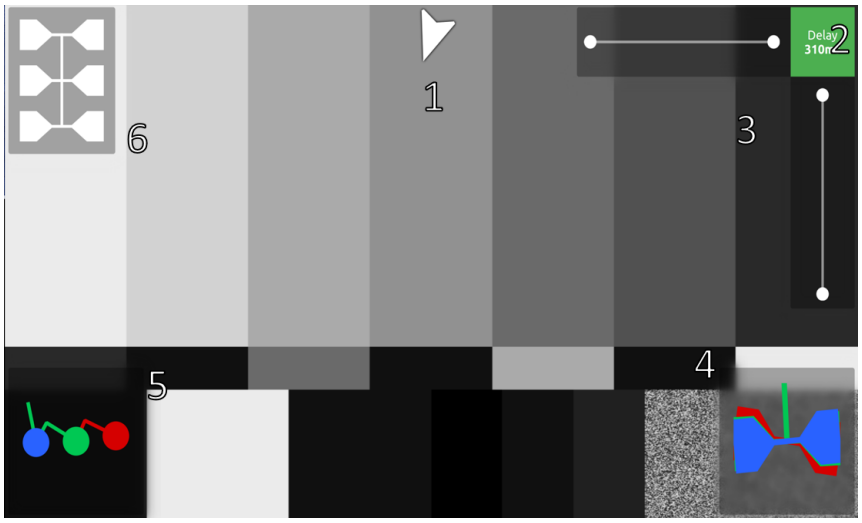

Figure 3.7: Main video window showcasing current rover HUD, and showing the positions and size of widgets overlaid on video feed to aid rover navigation.

speed, angle, power draw, latitude, and longitude, as well as timestamps for each recorded data point. Data related to the configuration of the video being broadcast to the navigating user is stored in the Settings file, and is recorded along with a timestamp whenever a change occurs in the configuration of the video mode being broadcast to the user. The Comments file contains timestamped comments from the tester recorded in the comments window, and is populated as new comments are created in a similar change-based manner to the Settings file. While the change-based data population of the Settings and Comments files make them both helpful and easy to understand, the interval-based sampling necessary for the Data file can lead to the creation of large, unwieldy files, as data is sampled and stored every 50 ms as long as data logging is enabled during testing sessions, which can run for up to an hour.

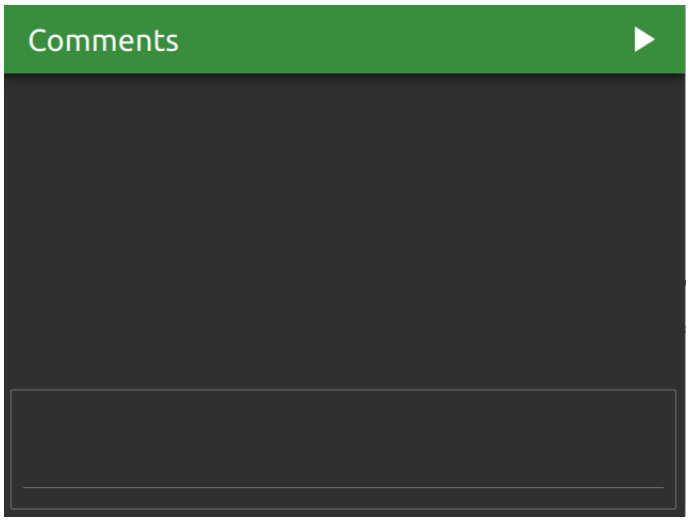

Figure 3.8: Comments window used by testers - play button allows tester to toggle data logging, and comments can be recorded in the box at the bottom.

3.2 Previous Testing Results

Initial testing to compare the results of human navigation of a prototype rover under significant latency and bandwidth restrictions involved fourteen test subjects, and entailed comparing the rover's previously described mono view option with its previously described stereo video option, which involved using a 3D TV to blend two side-by-side images of the same scene to help convey a sense of depth to the navigating user. The administered test involved a test subject partnered with a test giver in a remote location navigating the rover through a collection of predetermined courses, with two additional test givers on-site to prevent any catastrophic damage to the rover and help with transportation and repositioning between tests. During the test, the navigating user was expected to control the rover while being subjected to simulated speed-of-light delay, about three seconds, for operating a vehicle at a lunar distance. This delay manifested itself as the delay imposed on the user's perceived control inputs to the rover and the delay in the video feed showing the rover's location and surroundings. Additionally, the video resolution and framerate were

reduced to 176x140p resolution and 2 frames per second to reflect the bandwidth restrictions that would apply to video transmissions in a lunar application. The administered tests began with the tester in the remote location explaining the goal of the test and the methods used to control the rover to the navigating user, which then transitioned into a brief control familiarization period to prevent the newness of the controls and latency from significantly impacting the results of the first recorded course navigation. The user then navigated one of two courses, shown in Figure 3.9, either clockwise or counterclockwise using either a mono or stereo video feed. This was repeated for a total of four tests so that the user navigated each course, one clockwise and one counterclockwise, with both mono and stereo video. The order in which the course configurations were navigated, as well as the order in which the video feeds were introduced, was randomized for each participant to minimize the impact of familiarity with the control scheme and the courses on navigation time.

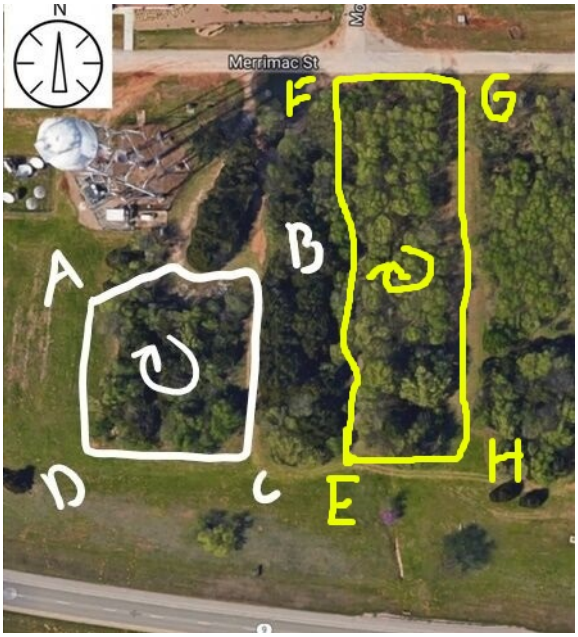

Figure 3.9: Example of the courses used in previous testing with the prototype rover, showing course configurations ABCDA and EFGHE.

The navigating users had a maximum of 900 seconds to complete each course, after which the testing would be stopped and the next course would be started. Emergency stops which required the intervention of the on-site testers were also recorded. After each test, the user was asked to complete a short questionnaire which gathered qualitative data on the user's impressions of the video feed and widgets' impact on their ability to complete the course. In total, each test took from 1 hour to 1 hour and 30 minutes depending on the performance of the navigating user. Data was collected from fourteen test subjects who each attempted to navigate through all four course configurations using either mono or stereo video to determine the effectiveness of both video feeds at assisting their navigation.

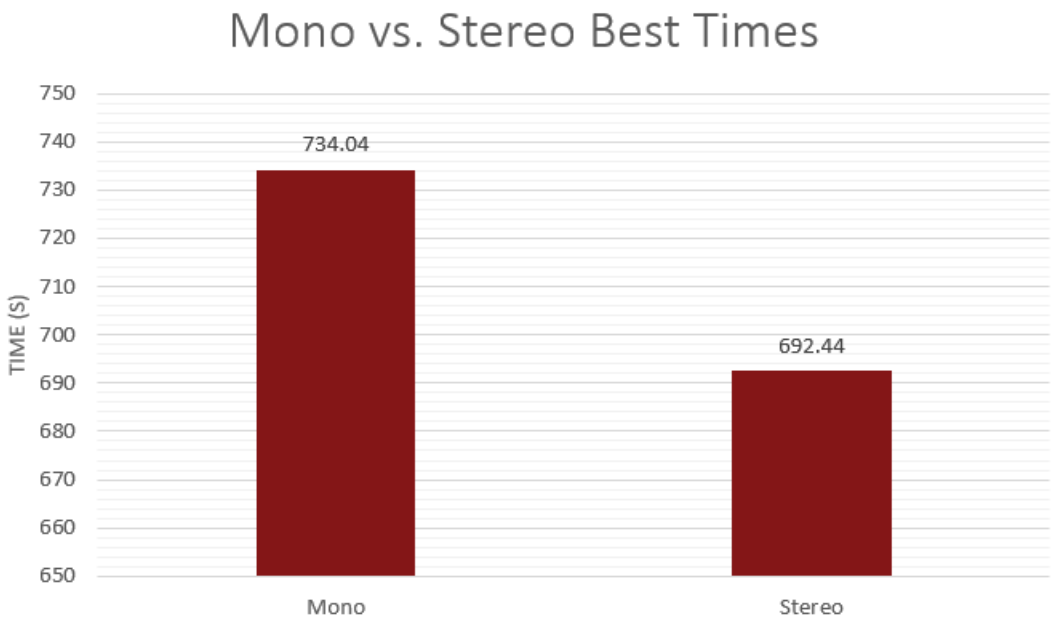

Figure 3.10: Average course completion time, organized based on if the user's best time was when utilizing stereo or mono video (N=14). Total user course times derived from average of all user course times for each user in each category

Figure 3.10 Shows the average course completion time for navigating users, organized based on if the user's best time was using mono or stereo video. All collected data was weighted identically when being averaged together to determine user per-

formance. Users who were more proficient at using stereo video were able to complete courses an average of 42 seconds faster than those who were more proficient with mono video. The average time for stereo-proficient users was also below the total average course time, 722.92 seconds, while mono-proficient users' average time was above the total average course time. The lowest overall course time of 425 seconds was also achieved by a stereo-proficient user when navigating with stereo. The results of these tests show that the most proficient drivers are able to drive most effectively when using stereo video, and were capable of achieving the lowest times among all courses and users. As proficient navigators will be able to be selected for teleoperating lunar rovers, these results indicate promise for the use of stereo vision to mitigate the impact of the time delay, latency, and bandwidth implicit in the test conditions for these users.

Qualitative Average Scores, N=14, Maximum=3	
Mono Average	2.2, STDEV=0.6
Stereo Average	1.9 STDEV=0.7
Widget 1 (Compass)	2.8 STDEV=0.4
Widget 2 (Latency)	2.2 STDEV=0.4
Widget 3 (Directional Inputs)	3.0 STDEV=0.0
Widgets 4 and 5 (Rover Roll and Pitch)	2.1 STDEV=0.7
Widget 6 (Current Draw)	2.2 STDEV=0.4

Table 3.1: Average qualitative scores for both video options and all widgets used in testing.

Both the video options and the widgets, whose numbers correspond to the guide set out in Figure 3.7, were analyzed in the post-course surveys using a qualitative three-point score of "Harmful" to "Helpful," with a one ranking as "Harmful", a two ranking as "No Impact", and a three ranking as "Helpful." and the numbers shown in Table 3.1 indicate the average of those scores for all tests. Average scores for both mono and stereo vision indicated that both video feed options were neither

harmful or helpful to user performance. Widget 1, the compass, and widget 3, which showed the magnitude and delay of rover control inputs, were both rated as extremely helpful to navigation by users, receiving scores of 95% and 100% of the maximum score, respectively. Thus, it is unlikely that their inclusion in the user HUD negatively impacted user performance. Users indicated that both video feeds, widget 2, the latency visualizer, widgets 4 and 5, the roll and pitch indicators, and widget 6, the warning schematic, all had neither a positive or negative impact on user performance. However, several users stated that widgets 4 and 5 were actively unhelpful during their navigation efforts, and only served to clutter the screen and obscure potential obstacles. While the low feedback directed at widgets 4 and 5 could be the result of ongoing IMU malfunctions which caused the widgets to poorly represent the rover roll and pitch, this feedback was suitable to force considering a redesign of the user HUD shown in Figure 3.6 that would minimize on-screen clutter and provide users with only widgets that aided navigation.

Chapter 4

Video Feed Changes and Expansions

This chapter presents a breakdown of the alterations made to the video subsystem for the prototype lunar rover, the strategies and methodologies employed to realize the transmission of depth data to the navigating user, and the necessary changes to other subsystems on the rover and within the code base to ensure that the video system upgrades were compatible with the device's operation. These video feed changes had to provide this data in a clear, understandable manner while at once not impacting the latency in the video pipeline to an unpredictable degree due to the low bandwidth of the communication between the rover and mission control. In order to realize these changes to the video feed, two additional options were added to the rover's potential video feeds:

- A pure depth data view, where the depth map generated by the rover cameras is streamed as-is to the navigating user, and
- A depth data overlay view, where the depth map generated by the rover cam-

eras is analyzed to highlight key areas in a mono video frame to alert the navigating user to potential hazards.

Incorporating these views required the inclusion of three primary components into the existing video pipeline - a filter which would read the incoming depth data and highlight potentially dangerous obstacles on a mono video feed, known as the Pixel Influence Filter, a disparity map generation method, and an image transform algorithm which would transpose the obtained disparity map to align with the mono video feed and put the disparity data in the context of the world frame the rover is navigating through. These views were tested to ensure functionality with the rover and clarity under normal operating conditions.

4.1 Pixel Influence Filter

While using the camera subsystem on the rover to produce a disparity map to help visualize depth data is mandatory for both depth-based camera views, an additional filter is necessary to properly represent the depth data as an overlay on an unaltered mono camera stream. In order to do this, the Pixel Influence Filter (PIF) was created, a GStreamer plugin which uses the functionality of a pixel thresholding filter to create a map to replace pixels in an image in a manner unobtrusive to the existing video pipeline.

In order to understand the functionality of the PIF, the functionality of pixel thresholding filters (PTFs) in general must first be understood. PTFs change the pixel values in an image to a maximum or minimum based on their value relative to a given threshold to create binary, or two-color, images. While pixel thresholding can be applied to three-channel RGB images, the simplest application comes from utilizing single-channel grayscale images and a single thresholding value t or pair of

values t_{lower}, t_{upper}. While this simplifies the thresholding calculation, it does require the thresholded images to be converted to grayscale, which involves averaging the three-value RGB representation of each pixel in a color image. Pixel color in RGB images are represented of arrays of three numbers from 0-255 in the format $[R, G, B]$ where higher values correspond to higher percentages of red, blue and green color in the pixel, respectively. Pixels in grayscale images are instead represented by a single $0 - 255$ value corresponding to a pixel's brightness, with pure white as most bright (255) and pure black as least bright (0), so the color values for pixels in an RGB image must be averaged together into a single value to generate a corresponding grayscale image. This conversion from pixel color values to pixel brightness, or intensity as it is commonly called, is performed in this application using OpenCV's cvtColor() method. cvtColor() transforms an input image from one color space to another, and is used in the PTF to transform an input RGB image into a grayscale image to search for a desired pixel intensity [30]. cvtColor() does this by averaging the 24-bit color representation of every pixel in an image into a single value using the conversion

$$Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{4.1}$$

where R, G, and B are the numeric values for each of the color channels of a pixel on the source RGB image and Y is the resulting intensity of the corresponding pixel on the generated grayscale image. 0.299, 0.587, and 0.114 are weights defined by OpenCV which sum to 1, ensuring that the range of intensity on the generated grayscale image is the same as the intensity range of each of the color channels on the source RGB image [30].

PTFs take an input image and analyze all of its pixel data, comparing each pixel's

intensity to see where it falls relative to the thresholding value(s). Depending on the structure of the algorithm, it will then alter the pixel data in the image to create either a foreground (white) or background (black) pixel based on each pixel's intensity compared to the thresholding value(s), and output the resulting binary image. The most common PTFs of this type for single value thresholds are *threshold above*, which makes all pixels with intensities greater than or equal to the threshold value maximum intensity, or white, and all other pixels minimum intensity, or black, and *threshold below*, which performs the inverse operation [31]. An example of these operations is shown in Figure 4.1.

Figure 4.1: Example of a pixel thresholding filter applied to a grayscale image showing, from left to right, the original image, a *threshold below* operation on that image with t=93, and a *threshold above* operation with the same t value. [31]

PTFs can also be conducted with two thresholding values, and will produce similar outputs to the results of *threshold above* and *threshold below* in Figure 4.1, but are more robust as they can be used to filter for intensity ranges between the maximum (255) and minimum (0) intensity values. The most common PTFs of this type for two value thresholds are *threshold inside*, which makes all pixels with intensities between the two threshold values maximum intensity and all other pixels minimum intensity, and *threshold outside*, which performs the inverse operation [31]. The PTF that is employed during the PIF's operation is a *threshold inside* which

executes the following logic:

$$I_{i,j} = \begin{cases} [255, 0, 0], & thresh_{low} <= I_{i,j} <= thresh_{high} \\ [X, Y, Z] \end{cases} \tag{4.2}$$

with $I_{i,j}$ equal to the intensity of the pixel being analyzed, $[255, 0, 0]$ representing the RGB color, red, that pixels within the threshold values are set to, and $[X, Y, Z]$ representing the original RGB color of the pixel being analyzed, as pixels are not modified if they are outside the searched threshold. The PIF applies the thresholding algorithm to a provided grayscale "map" image, but then applies the color change to a "target" RGB image. Because of this, the values used in thresholding are $0 - 255$ pixel intensity values, while the actual pixel color changes are three-channel RGB values. The user can assign the values of the upper and lower thresholds when launching GStreamer.

The VideoStream process in the existing project code base streams video by running GStreamer with a bin string, which encapsulates all of the details for the complete pipeline on either the rover or mission control side. This bin string is then used to create a GStreamer pipeline as if GStreamer was being launched from the launch line. This eliminates the need for any extra C++ code to manage the behavior of the video pipeline, but also makes it difficult to alter data in the pipeline once it has been created. As such, creating the PIF as a GStreamer plugin allows it to be incorporated directly into the bin string the program is creating, minimizing the impact of its added functionality on the operation of the rover program as a whole. An example launch line including a PIF is shown below in Listing, with its result shown in Figure 4.2.

Listing 4.1: Example GStreamer launch line including a PixelInfluenceFilter element. High and Low dictate the upper and lower bounds of the thresholding operation while Paint dictates the color of the transformed pixels. Thresholded pixels in this case are turned red.

```
gst-launch-1.0 --gst-debug=4 videotestsrc ! capsfilter
caps=video/x-raw,width=640,height=480 ! videoconvert
! filter.sink_target videotestsrc ! capsfilter
caps=video/x-raw,format=RGB,width=640,height=480 !
videoconvert ! filter.sink_map pixelinfluencefilter name=
filter low=1 high=254 paint=R ! videoconvert ! ximagesink
```

The PIF has two source inputs, which take in a map image which will be transformed into a binary image map by the thresholding operation, and a target image which will have its pixel data altered based on the results of the thresholding operation; these images must be the same size. It has one sink output, which sends the altered target image to the next element in the pipeline. On the launch line, users can pass the PIF values for the upper and lower bounds on the PTF, as well as the color the transformed pixels on the target image will become - either red, green, or blue. If given no input, the PIF will default to filtering all black pixels (*intensity=0*) on the map image, and changing the color of the corresponding pixels on the target image to red. Before being passed to the thresholding filter, the map image collected by the PIF is stored in an OpenCV Mat object, used to contain images or video frames as matrices of data, and changed to a single-channel grayscale image using OpenCV's cvtColor() method discussed earlier, allowing the PIF to filter the image based on the desired pixel intensity range being highlighted. The PIF thresholding filter is executed using OpenCV's inRange() method, which

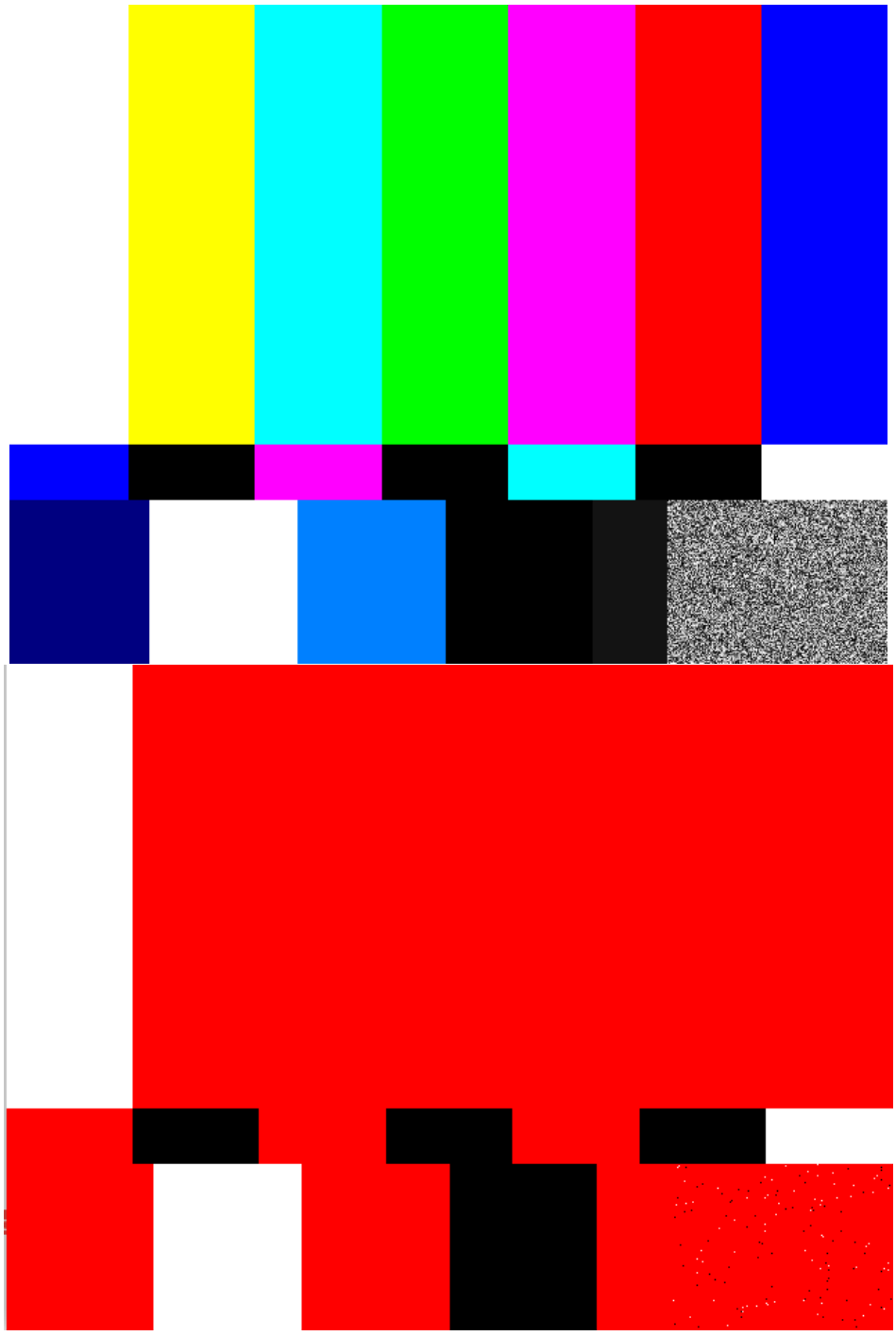

Figure 4.2: (Top) Pre-transformed SMPTE test image; (Bottom) SMPTE test image after applying PixelInfluenceFilter to transform all pixels in frame with an intensity I between 1 and 254, corresponding to all colors between pure white and black, to red.

takes a Mat object, along with two values, and checks to see if the values at each point in the Mat object are between the two values. This method then writes the resulting image to a new Mat object, which in the case of the PIF is just written back to the initial Mat object passed to inRange() [32]. The color transform is then executed using OpenCV's setTo() method, which takes the RGB scalar array representing the user's desired transformed color and the binary image generated by the thresholding operation and changes all corresponding pixels in the target image to the user's chosen color [33]. This code is shown below in Listing 4.2 for changing pixels in the target image to red.

Listing 4.2: PIF transform code

```
  if (PAINT_R == filter->paint) {
    filter->paintValue[0] = 255;
    filter->paintValue[1] = 0;
    filter->paintValue[2] = 0;
    cvtColor (filter->cvMapRGB, filter->cvMapGray,
    COLOR_RGB2GRAY);
    run_pixel_transform (filter);
  }


  int
run_pixel_transform(GstPixelInfluenceFilter * filter) {


  inRange(filter->cvMapGray, filter->searchValueLo,
  filter->searchValueHi, filter->cvMapGray);
  (filter->cvTarget).setTo(Scalar(filter->paintValue[0],
```

```
    filter->paintValue[1], filter->paintValue[2]),
    filter->cvMapGray);

 return 0;
}
```

Performing these operations using existing OpenCV methods reduces lag when compared to performing them without utilizing OpenCV, which diminishes the effect that including the PIF in a VideoStream pipeline has on the streaming rate of the rover video feed. The portions of the PIF code not related to the transform being conducted is heavily based on the existing GStreamer plugin GstDisparity by Miguel Casas-Sanchez, as both plugins are filters with exactly two inputs and one output that perform OpenCV transformations, so much of their functionality is translatable.

4.2 Depth Map Generation

The key component necessary for transmitting depth data to the navigating user in this application requires using a stereo image stream to produce a depth map of the scene in front of the rover to calculate the depth of the objects in the image. This was initially attempted with the existing rover hardware to minimize cost by incorporating GStreamer filters designed to produce disparity maps of the captured scenes, but was successfully achieved using an Intel D435 Depth Camera to produce depth images onboard the camera itself, reducing the processing stress of the depth video streaming to a level manageable by the rover and mission control computer while maintaining a resolution and framerate that is understandable to the user.

4.2.1 Disparity Map Generation with Existing Cameras

In an effort to minimize impact on the physical subsystems of the rover and its code base, the first solution explored involved using the existing stereo camera array to generate a disparity map which could then be sent as its own video feed or manipulated by mission control to overlay the depth information over a mono video feed. The ideal situation for incorporating a depth feed into the existing video streaming architecture would be for all of the processing to be conducted onboard the rover, as the pipeline for generating the stereo image feed discussed in Chapter 3 already involved rover-side video processing. As such, incorporating the disparity map generation and PIF rover-side would minimize the changes necessary to the mission control code base and involve simply repurposing the existing code for stereo image feeds for the newly created depth feed. A diagram of this pipeline is shown in Figure 4.3.

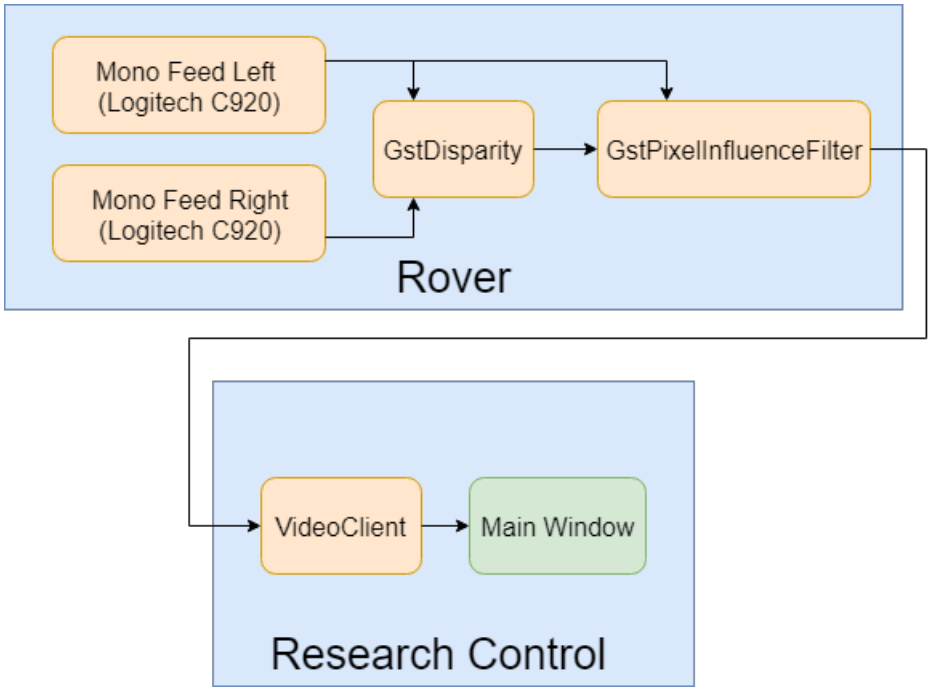

Figure 4.3: Initial pipeline concept for depth data streaming

The GStreamer plugin GstDisparity was initially considered for use in generating the disparity map which would be sent to the PIF as the map image, as it could be simply integrated into the existing GStreamer bin strings being used by the rover's VideoStream processes. GstDisparity is a filter plugin packaged with GStreamer's in-development plugin package (gst-plugins-bad) that, much like GstPixelInfluenceFilter, offers OpenCV functionality without needing to break the GStreamer pipeline. GstDisparity collects two source inputs from identical, aligned, rectified stereo cameras and applies a stereo matching algorithm of the user's choice to generate a disparity map with the two images, which it then outputs to the next element in the pipeline using its sink output. Users can specify the stereo matching algorithm on the GStreamer launch line to be either OpenCV's Stereo Block Matching algorithm (SBM) or the Semi-Global Matching algorithm (SGBM). SBM is best for strong, varied textures like rugs and is unable to accurately handle soft textures like a single color wall, while SGBM is less accurate at handling stronger textures, but is more capable of handling soft textures and fast despite its large memory footprint [34]. An example of a GStreamer launch line including a GstDisparity element and pulling video feeds from two camera devices is shown below in Listing 4.3, and the application of GstDisparity to image inputs is shown in Figure 4.4.

Listing 4.3: Example GStreamer launch line including a GstDisparity element. Method dictates the algorithm which will be used by the element. The algorithm used in this example is Stereo Block Matching (SBM).

```
gst-launch-1.0 v4l2src device=/dev/video1 !
video/x-raw, width=320,height=240 ! videoconvert
! disp0.sink_right v4l2src device=/dev/video0
! video/x-raw,width=320,height=240 ! videoconvert
```

```
! disp0.sink_left disparity name=disp0 method=sbm
disp0.src ! videoconvert ! ximagesink
```

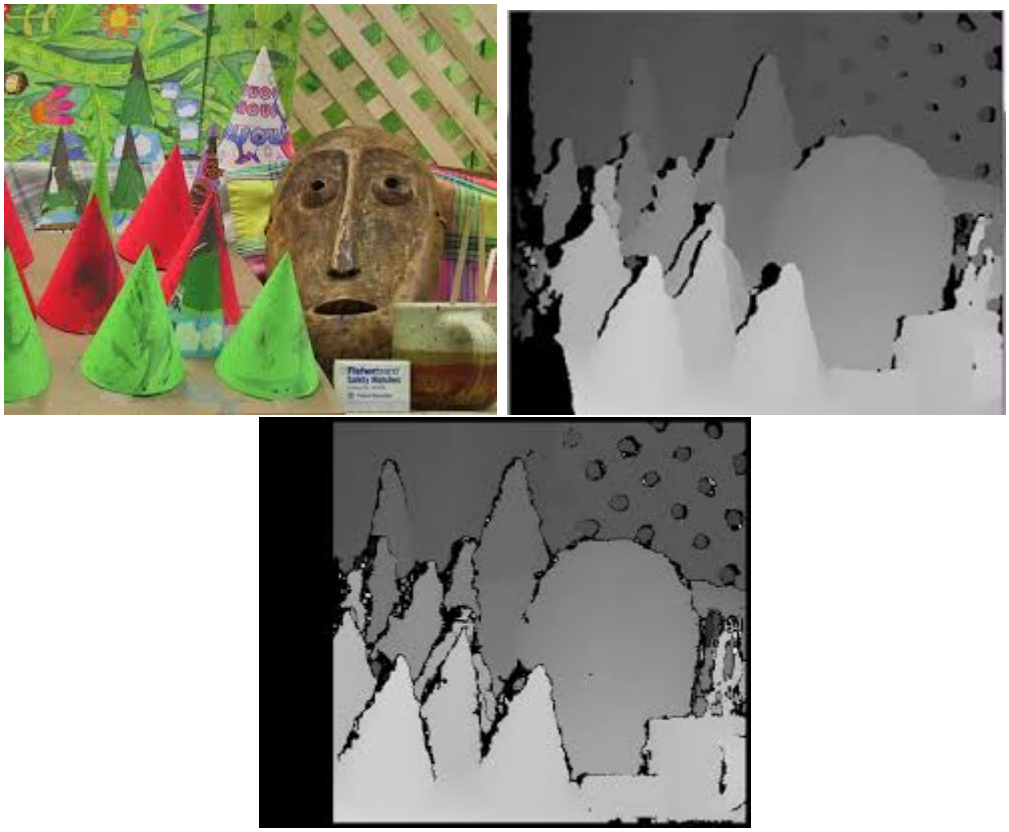

Figure 4.4: Example of OpenCV disparity algorithms used by GstDisparity applied to classic stereo matching image Cones, shown unmodified (Top Left), after applying SGBM (Top Right), and after applying SBM (Bottom)[35]. In both images, objects nearer to the camera are more white, and objects farther away are more black.

The pipeline shown in Figure 4.3 was implemented, but upon testing it was discovered that the operations performed by GstDisparity were too computationally intensive for the Up Board computer onboard the rover, even when run outside of the rover control program. As such, the pipeline needed to be restructured to accomodate for offloading the computational stress of the depth map creation to the mission control computer. The pipeline adjustments made to handle this are shown in Figure 4.5, which adapts to the issues found in Figure 4.3's pipeline by

creating an additional channel to pass a video feed from the rover to mission control.

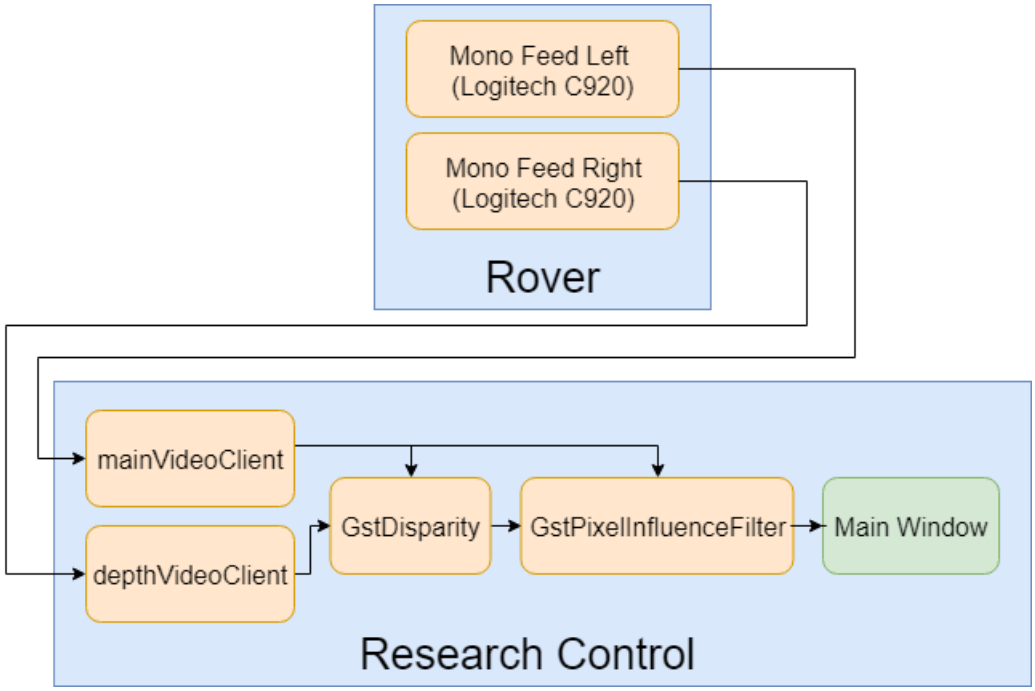

Figure 4.5: Second pipeline concept for depth data streaming, involving two video channels to stream video back to mission control for pre-processing.

This channel, which creates a new pair of VideoClient and VideoServer objects, named depthVideoClient and depthCameraServer, are bound on a new available TCP port and are only active during the streaming of depth data. This new channel, along with the main video channel which handles mono video streaming, simultaneously stream mono views from the two cameras on the rover to mission control, where simultaneous frames are grabbed from the video channels and fed to GstDisparity and GstPixelInfluenceFilter to generate the desired output of depth information. GstDisparity and GstPixelInfluenceFilter are incorporated into the mission control-side bin string in this pipeline.

Upon testing this pipeline, it was discovered that, while streaming two video feeds

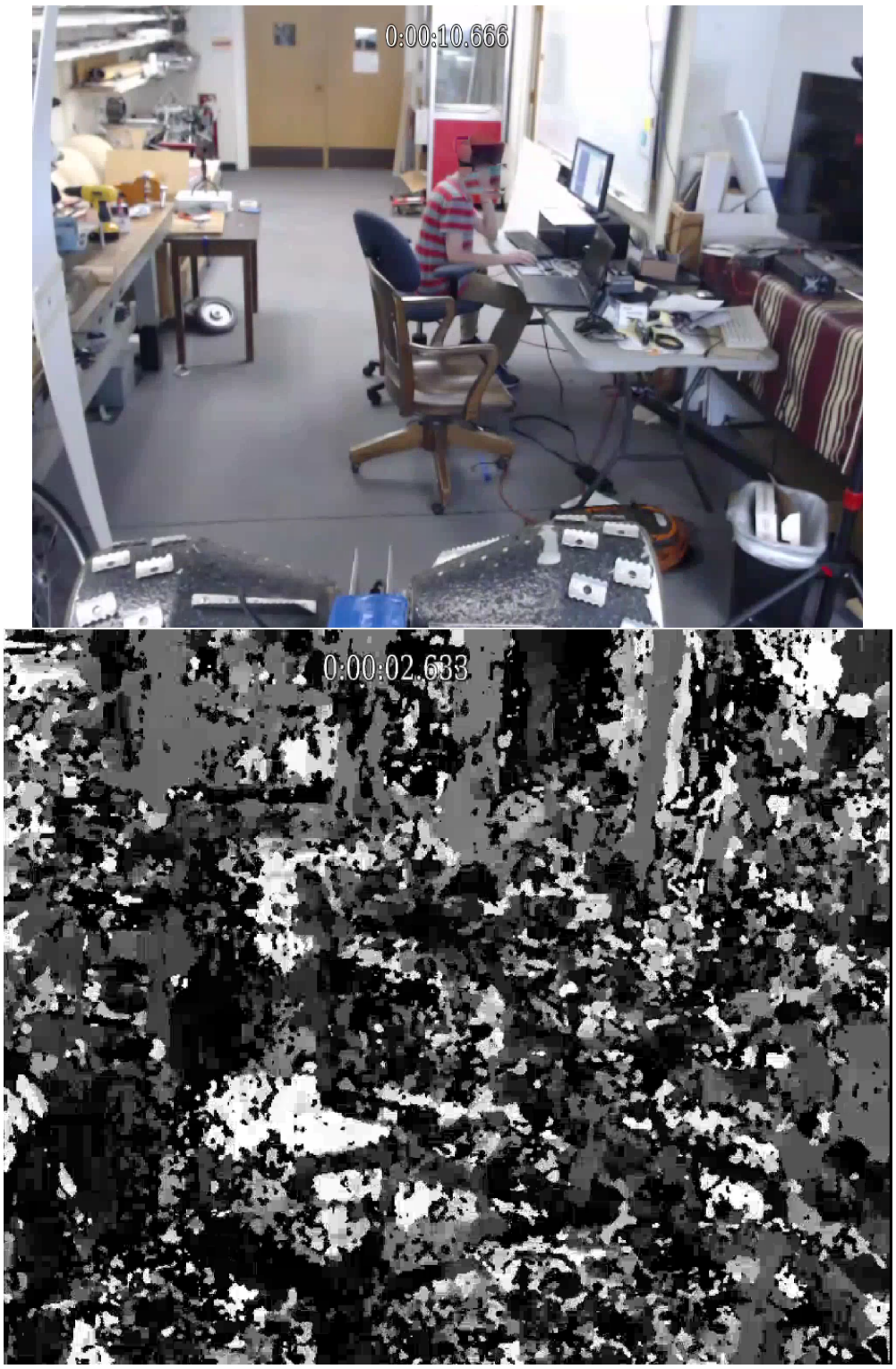

Figure 4.6: (Left) Input image to GstDisparity depth pipeline; (Right) Disparity map generated by GstDisparity.

was possible without significant computationally-imposed latency in the streams from the rover, running GstDisparity on mission control at any resolution above 320x240 pixels resulted in high computational latency of about 1-2 seconds. This computational latency would have made it impossible to test the depth pipeline in any capacity in future tests with the current hardware to see if it improved navigator times. Additionally, as shown in Figure 4.6, GstDisparity was incapable of matching a significant number of pixels in its algorithm, resulting in a mostly incomprehensible image. While this could have been corrected by changing the orientation of the mounted cameras to point inward so as to allow the cameras vision of each other, testing with the rover cameras indicated that the cameras would need to be tilted inward each at about a 45° angle to allow for reasonable pixel matching. This 45° angle maximized the distance in front of the rover that could be perceived by the cameras while still generating a useful depth map, about 3.5 meters. This would have drastically limited the rover's field of view, as well as the distance in front of the rover that the cameras were able to view. This would would have additionally rendered the other camera view options unhelpful to navigation, as they would be showing camera feeds facing off to the side of the rover.

Because GstDisparity seemed to be too computationally intensive to include in any portion of the actual video streaming process, disparity map generation using the existing camera setup on the rover became an infeasible option for passing depth data to the navigating user. As such, a new camera setup for the rover that would allow for the current video feed functionality while also adding depth data functionality needed to be developed.

Figure 4.7: D435 Depth Camera.

4.2.2 Depth Video with D435 Camera

In order to pass depth data to the navigating user, the pair of Logitech C920 cameras capturing video for the rover were replaced with an Intel RealSense D435 camera as the primary rover camera, though they were still retained to transmit the image parallax stereo feed. The D435 is a combination RGB/Stereo Depth camera which performs all depth processing onboard the camera, allowing depth video to be read like a standard webcam by GStreamer using Linux's innate video streaming API, Video4Linux. This takes a tremendous amount of stress off of the rover and mission control computers by making the inclusion of GstDisparity unnecessary to provide high-quality depth data to the navigating user. The D435 depth camera also possesses a higher FOV (91.2°x65.5°) [36] compared to the Logitech C920(70.4°x 43.3°) [25]. Increased FOV in teleoperation scenarios like the ones this camera setup are designed for is critical in improving navigator performance, allowing for time im-

provements of over 50% in some instances [11], making it a natural upgrade to the rover vision systems to maximize their effectiveness for guiding and aiding users.

Testing the D435 with the current rover electronics also revealed that the USB 2.0 ports provided by the rover's onboard UP board did not allow for fast enough data streaming for the camera to communicate successfully with GStreamer. Additionally, the USB 3.0 OTG port, which would allow for fast enough data streaming, was unusable due to a known, currently unfixable bug in the D400 camera series firmware that causes issues with its USB enumeration and prevents reliable use of the port. As such, it was necessary to upgrade the board to an Up Squared which possesses both the USB 3.0 A ports necessary for a fast, stable connection with the camera, as well as a more powerful Intel Celeron N3350 Processor capable of a maximum processing frequency of 2.4GHz [37].

4.2.3 Additional D435 Pipeline Considerations

While switching to the D435 dramatically reduced the amount of processing stress placed on both the rover and mission control computers managing the video streaming, the D435 output its depth images as a four-channel image, which made it unreadable by GStreamer and therefore impossible to incorporate directly into the existing GStreamer video pipeline. In order to bridge the disconnect between the two technologies, a final pipeline was developed, shown below in Figure 4.8.

The D435, now being the primary source of video feeds for the rover, sends mono and depth streams to be read separately or in tandem by GStreamer, which then sends the streams to the appropriate client at mission control. The depth feed from the D435 camera must be initially read in through the D435's RealSense SDK and OpenCV, as the D435 depth data is sent in a four-channel video format

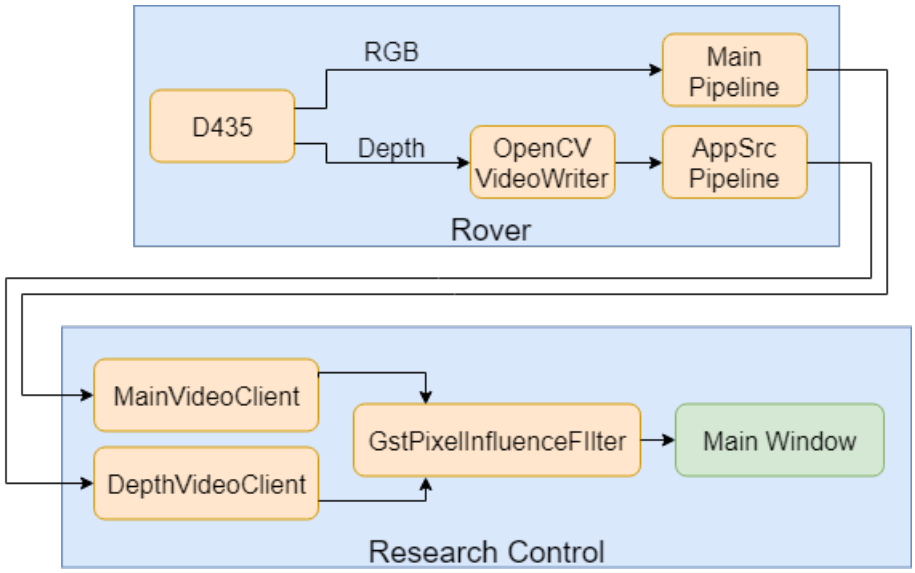

Figure 4.8: Final pipeline for depth streaming using the D435 Depth Camera.

which includes distance measurements for each pixel.Typical video streaming using GStreamer makes use of the Video4Linux API to directly capture video from cameras registered to the Linux system through the use of a source object called v4l2Src [38]. While this is usually effective, v4l2Src does not allow for modification the four-channel video format produced by the D435, which must be pre-processed to become a three-channel format understandable by GStreamer. In order for the image to be successfully streamed back to mission control, the RealSense image must be changed to a three-channel RGB image using a RealSense colorizer object, which applies a filter to incoming depth frames to render them as an RGB image with a specific color scheme [39]. The data from the camera is packaged as an OpenCV Mat object and pushed into a buffer to be delivered to the GStreamer pipeline using an AppSrc object. AppSrc is an abstract source element used by Gstreamer to allow external applications to insert data into a GStreamer pipeline [40]. Replacing the standard v4l2Src currently used by rover-side GStreamer bin strings in the project with an AppSrc element allows the program to perform the necessary pre-processing

on the depth data it is pulling from the D435 camera, and subsequently pass it to the GStreamer video streaming pipeline when it is in an understandable format.

Figure 4.9: (Top) RGB image of outdoor scene captured by D435 camera, FOV = (69.4°x42.5°); (Bottom) depth map of same scene generated by D435 camera, FOV = (91.2°x65.5°). [36]

Figure 4.9 above shows a still taken from the depth stream generated by the D435 compared to a mono image of the same scene, with redder objects being closer and bluer objects being farther away. This image clearly shows the scene in front of the

rover to the user, and allows them to distinguish potential obstacles in the distance, such as the tree and bush in Figure 4.9. This image was also comprehensible enough to be used as a source image for the AR overlay developed as a user option on the rover. The current pipeline is capable of streaming this footage at 30 frames per second in up to 640x480p resolution, though it can also be reduced down to the 2 frames per second and 176x144p resolution necessary to properly replicate the conditions a navigating user will experience if attempting to control the rover at a lunar distance. Note that the D435 depth camera does not actually allow for capturing video on a gradient of framerate and resolution values, and instead has a large library of set pairs of resolutions, framerates, video modes, and encoding protocols that can be selected to allow for streaming of all of the D435's video modes at common resolutions and framerates. Thus, all video captured using the D435 camera is initially captured at 30 frames per second and in 640x480p resolution, and has its resolution and framerate lowered or changed in the GStreamer pipeline to match the resolution and framerate selections on the tester interface before it is displayed to the navigating user. 640x480p resolution was selected as the initial capture resolution to maximize depth data quality, as higher resolutions allow for higher depth data fidelity, while keeping video streaming latency as low as possible. 30 frames per second was chosen as the initial capture framerate to ensure that the high framerate necessary for initially familiarizing navigating users with the rover control scheme would be achievable in the depth and RGB pipelines.

4.3 Image Transform Algorithm

While the depth image created by the D435 was clear enough to be used as a pure depth information stream to the user, it was not immediately useful as a source for

the AR overlay. This was due to differences in the images produced by the D435's RGB and depth cameras at a given resolution, and the D435's position relative to the world frame the rover was traveling through, which made it impossible to filter out unhelpful depth data about the ground the rover was traveling along instead of potential obstacles. In order to convey meaningful information to the navigating user, the depth image needed to be aligned to the RGB image's camera, and the perspective of the camera needed to be transformed to world coordinates to filter out unhelpful depth information being sent to the user.

4.3.1 Depth Image Alignment

Because the depth and RGB images captured by the D435 camera are captured through different lenses and through distinct positions on the camera, they have different intrinsic and extrinsic properties. The intrinsic properties of the depth and RGB cameras represent the relation between camera coordinates and the pixels in each image, and are primarily affected by the differences in the camera FOVs (91.2°x65.5°for the depth camera, and 69.4°x42.5°for the RGB camera) [36], and differences in the cameras' internal characteristics. The extrinsic properties of the cameras represent the differences between the two cameras' positions with respect to the external fixed world frame, as they are located in two physically different places [21]. In order to compensate for these differences, we can apply a perspective transform to the depth image which will warp the image in three dimensions to align with a new size. This is accomplished by first using the getPerspectiveTransform() method, which calculates a 3x3 transformation matrix using sets of four selected points that map to the same objects on both images such that

$$\begin{bmatrix} t_i x_i^{'} \\ t_i x_i^{'} \\ t_i \end{bmatrix} = map\_matrix * \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{4.3}$$

where

$$dst(i) = (x_i^{'}, y_i^{'}), src(i) = (x_i, y_i), i = 0, 1, 2, 3 \tag{4.4}$$

After this matrix is generated, the warpPerspective() method is used, which applies this matrix to a source image and generates an output image of a specified size [41].

In order to determine these sets of points, the Intel RealSense Viewer plugin was used to view images of a scene captured using the D435 RGB and depth cameras simultaneously, and pinpoint the positions of significant geometry within the images that could be used to create the perspective transform matrix. Figure 4.10 shows the two images below, with the rough locations of the points used for the perspective transform matrix generation circled in white.

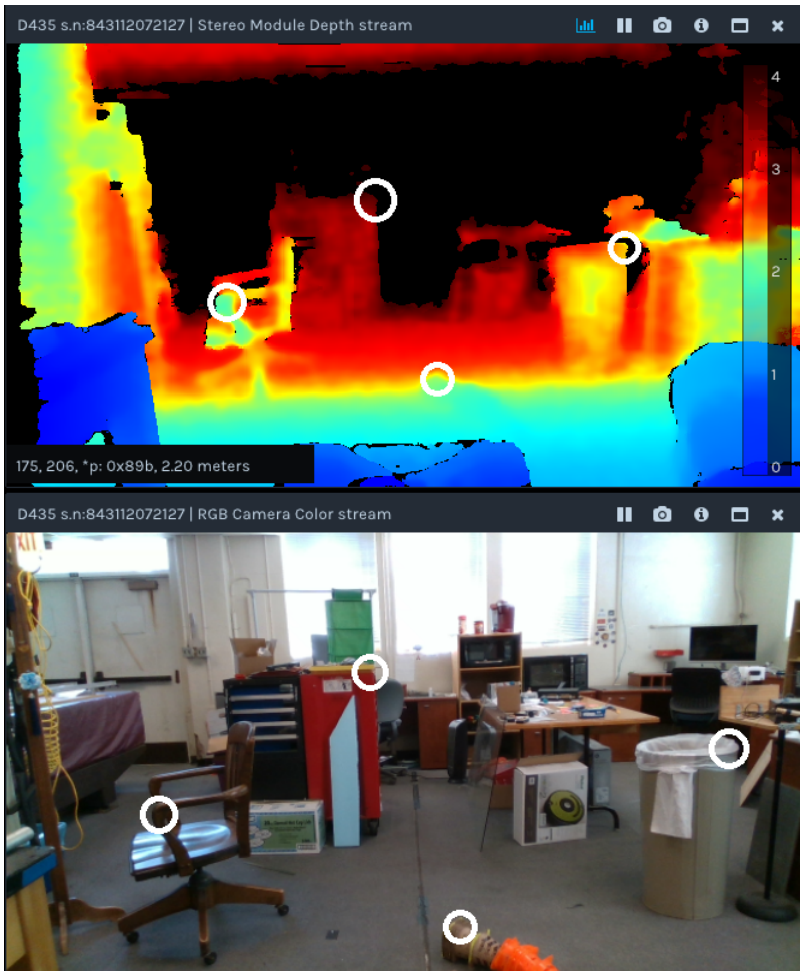

Figure 4.10: Images of the same scene captured by the D435 RGB and depth cameras for determining points to be used in Perspective Transform matrix generation. White circles denote significant objects whose rough coordinates in each image were used as points for matrix generation.

Even though this manual mapping is unique for a given image resolution, it only needs to be conducted once, as the rover-side image capture loop for the D435 captures images at a fixed resolution of 640x480p for both the RGB and depth cameras. The images are adjusted to the desired resolution of the navigating user before displaying them at mission control, after the perspective transform has been conducted. Figure 4.11 below shows images from the depth and RGB feeds after the depth image has been aligned to the RGB image, giving both images an FOV

of (69.4°x42.4°). Note that, while the depth FOV is smaller than that shown in Figure 4.9, the components of both images are much more closely aligned, allowing for accurate mapping of depth data from the depth image to the RGB image.

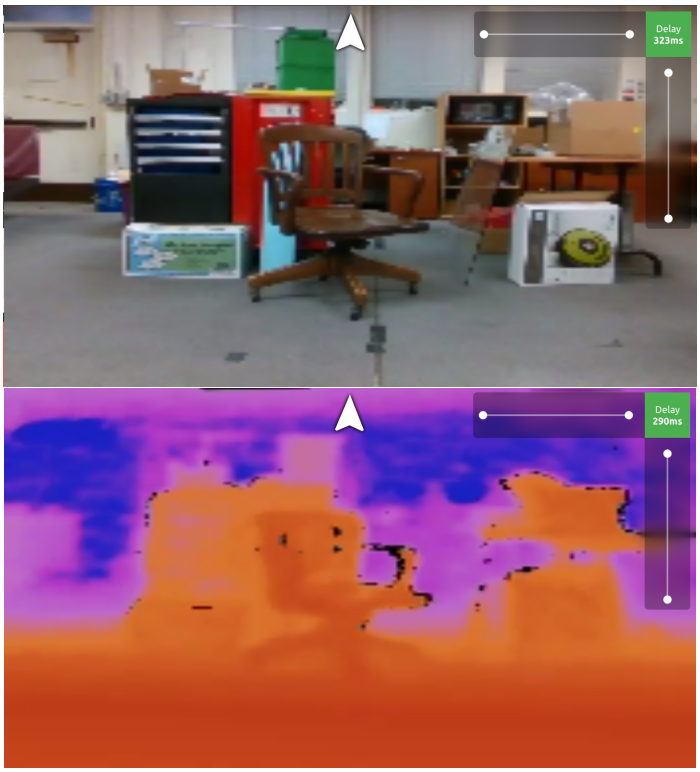

Figure 4.11: Images of the same scene captured by the D435 RGB (top) and depth (bottom) cameras after being aligned using perspectiveTransform(). The resulting FOV of the images is the FOV of the D435 RGB camera, (69.4°x42.5°)

4.3.2 Camera Coordinate Transform and Ground Filtering

The Pixel Influence Filter used to transfer depth data to a target image simply edits the color of pixels on the target image which correspond to pixels on the source image within a given intensity range. This means that the intensity range must be large in order to highlight obstacles at a range of distances in front which would allow a user to benefit from the additional information. However, this large highlighted

intensity range causes a significant portion of the ground on the target image to be highlighted as an obstacle as it falls within the targeted depth range, cluttering the image and making it impossible to effectively use it for navigation. In order to make an AR overlay of depth data beneficial to a navigating user, pixels associated with the ground in the image must be filtered out before being sent to the Pixel Influence Filter to ensure that they will not be highlighted on the target image. This filtering can be done utilizing the real-world coordinate information derived from the depth images captured using the D435 depth camera.

The presence of depth data in the four-channel images captured using the D435 depth camera provides a method for filtering out ground pixels in the depth images, as it gives a 3D world coordinate to associate with the 2D camera coordinates of the pixels in the image. Using a method discussed by Yang et. al. [42], the corresponding 3D X and Y coordinates for a pixel can be calculated from the pixel's camera coordinates if the camera's focal length and principal point are known. This relationship is written below as

$$X = Z * \frac{u - u_0}{f} \tag{4.5}$$

$$Y = Z * \frac{v - v_0}{f} \tag{4.6}$$

where (u, v) are the coordinates of a pixel in the image in camera coordinates, (u_0, v_0) are the coordinates of the camera's principal point, f is the calibrated focal length of the camera, and (X, Y, Z) is the 3D coordinates of the pixel in the image. The principal point of an image is the point where the principal ray, which intersects with the camera pinhole, intersects with the image plane [43]. Knowing the principal point allows the image plane's intersection with the principal ray to be set as the image ori-

gin, referencing the generated 3D camera coordinates for each pixel to the center of the camera's perspective rather than the upper left hand corner. The obtained value for focal length was converted from its real-world distance, 1.93 mm., based on the resolution of the initially captured video stream from the D435 camera, 640x480p, so that it could be represented as a multiple of the pixel width of the captured image [36]. This allowed the focal length of the D435 to be used for converting the 2D coordinates of each pixel to 3D coordinates with the correct units, namely, length [44]. The calibrated focal length and principal point of the D435 depth camera in pixel units were determined for the specific video format and resolution that will be used in depth video generation to be 381.454 and $(321.783, 230.656)$, respectively. These values were obtained using a RealSense API function called rs-sensor-control which allows for the generation of device information, camera intrinsic and extrinsic information, and granular sensor manipulation [45]. As this video format and resolution will not change regardless of the output resolution of the video at mission control, these values can be assumed to be constants for the coordinate conversion.

Once the 3D coordinates of a pixel are obtained, they must be transformed further, as they are currently accurate with respect to the camera's frame of reference in the world instead of the external world frame the rover is in. This transformation can be expressed as a combination of rotations and translations performed on the 3D world coordinates of pixel to obtain its 3D coordinates with respect to the world frame, written as

$$P_W = (R^{-1}P_C) + T \tag{4.7}$$

where P_C is the 3x1 matrix representing the pixel's 3D coordinates in the camera frame of reference, T is the 3x1 matrix representing the total translations necessary

to align the camera frame with the world frame, R is the 3x3 matrix representing the total rotations necessary to align the camera frame with the world frame, and P_W is the resulting 3x1 matrix representing the pixel's 3D coordinates in the fixed world frame. Once this world frame coordinate matrix has been obtained for a pixel, it's Y coordinate can be checked to see if it is a low enough height to be considered part of the ground and filtered out.

In order to appropriately detect the ground in a generated depth image, this transformation needs to be conducted hundreds of times for each depth frame, which introduces high computational delays to the AR video stream that lower the framerate to unusable levels of less than 0.2 Hz. This framerate does not provide information quickly enough to navigating users to be useful, and also frequently results in connection timeouts between the rover and mission control. In order to address this, several assumptions are made about the geometry of the transformation and necessary image analysis to limit processing time. First, the ground the rover is traveling along is assumed to be a flat plane - this makes detecting the ground easy, as it simply needs to lie within a certain height or depth independent of slope, making any feature outside of that height an obstacle. Second, the ground is assumed to only lie within the bottom 60% of each frame, which allows the ground filtering algorithm to skip nearly half of each image it analyzes. Finally, the D435 camera's view of the scene in front of the rover is assumed to be exactly centered. This means that only a translation along the Z-axis equal to the height of the rover off of the ground, measured to be 0.85 meters, and a rotation about the X-axis equal to the downward angle of the camera measured to be 6.0°, are necessary to transform each pixel's 3D coordinates from the camera frame to the world frame, reducing computation time. The rotation and translation matrices applied to each pixel's set of 3D coordinates as a result of this simplification are as follows:

$$T = \begin{bmatrix} 0m. \\ -0.85m. \\ 0m. \end{bmatrix} \tag{4.8}$$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.994 & -0.104 \\ 0 & 0.104 & 0.994 \end{bmatrix} \tag{4.9}$$

4.3.3 Algorithm and Results

The algorithm applied to each depth frame that is a part of an AR stream, resulting from combining the perspective transform and ground filtering steps, is as follows:

Algorithm 1 Depth Image Transform Algorithm

Require: Perspective transform matrix, camera principal point, camera focal length, RGB depth image and associated depth data

1: **for all** Pixels in depth frame s. t. v in (u, v) $\geq$ 192 **do**
2: Get depth (3D Z coordinate) of pixel
3: Calculate 3D X and Y coordinates
4: Apply inverse rotation matrix R^{-1} to 3D pixel coordinates
5: Add translation matrix T to 3D pixel coordinates
6: **if** $|Y| < groundHeight || vin(u, v) > 405$ **then**
7: $pixelIntensity = 0$
8: **end if**
9: Apply perspective transform
10: **end for**

The algorithm above filters the ground out of the image, which is captured at a fixed 640x480p resolution, by setting the intensity of any pixel which qualifies as "ground," i. e. has a Y coordinate in the 3D world frame determined to be lower than the absolute value of the ground height tolerance, by setting its color to black, therefore setting its intensity to 0. When this depth image is read by the Pixel Influence Filter, these pixels are not recolored on the target mono image because

the Pixel Influence Filter is programmed to recolor pixels with an intensity of 65 or higher, up to 255. The intensity range for recolored pixels corresponds to nearby objects being colored in white by the depth image when it is adjusted to grayscale. This cutoff value was chosen based on the effective distance of the D435 Depth camera, 10 meters [46], and the average speed of the rover reported during previous course testing, 0.42 m/s. Since the user will be experiencing three seconds of input delay on their commands, objects will need to be highlighted at least 1.26 meters in advance on average to be useful to a user. In order to give the user a buffer for responding to obstacles and allow for route planning in advance, this buffer was doubled to 2.52 meters, or 25.2% of the D435's effective range. Since pixel intensity scales linearly with the depth of an object detected by the D435, it was possible to determine the minimum highlighted pixel intensity to be approximately 65. Closer objects, which would have higher intensity, are also highlighted to ensure users stay aware of potential hazards while navigating.

In order to speed the execution of this algorithm and reduce unnecessary computation time, the algorithm begins to search for ground in the 192^{nd} row of the image, cutting out the upper 40% of each image as it is unlikely for the ground in the image to lie in this area. In addition, the bottom sixth of the image, starting at the 405^{th} row, is also filtered out. A navigating user would not be able to respond to any object that appears below this row in the image given the delay they are navigating under as it is too close, so providing that information only risks cluttering the video feed and searching it for ground is unnecessary. The approximate area that the algorithm searches for ground is shown below in Figure 4.12.

Figures 4.13 and 4.14 below showcase one indoor and one outdoor scene, the result of applying the image transform algorithm to a depth frame, and the resulting AR image after being run through the Pixel Influence Filter. Both images are at

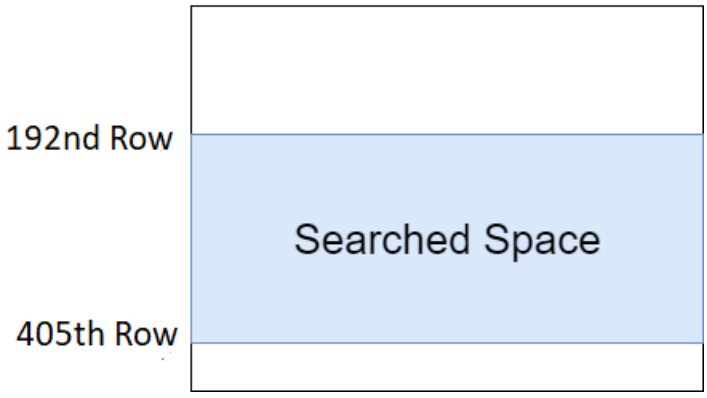

Figure 4.12: Space searched for ground by ground detection - algorithm searches from 192nd row of image to 405th, and cuts off all other rows in image.

the resolution that a navigating user will see to simulate the compression which would occur for lunar navigation. The obstacles shown in Figures 4.13 and 4.14 are not totally highlighted by the AR video feed, as the approximate cutoff value for the ground needed to be raised to ± 7.5 centimeters. The assumptions currently made about the ground to simplify the image analysis means that the detection isn't robust to significant movements of the camera during operation, as this moves the ground plane from the perspective of the camera. The raised ground tolerance helps to counteract this current lack of robustness, as if this value was set any lower, a large portion of the ground would be highlighted in addition to the lower portions of the obstacles, rendering the AR highlighting completely unhelpful to navigation. While this isn't ideal, significant obstacles are still highlighted by the algorithm, and the AR pipeline is still able to show the ability for the rover to transmit depth data to a navigating user with acceptable computational latency to be used for testing.

The combination of the Pixel Influence Filter, D435 depth camera, and Image Transform Algorithm allows for the production of both a high-quality depth video stream, as shown in Figure 4.9, and an AR filter which highlights potential obstacles several meters away from the rover by filtering those obstacles from the ground with

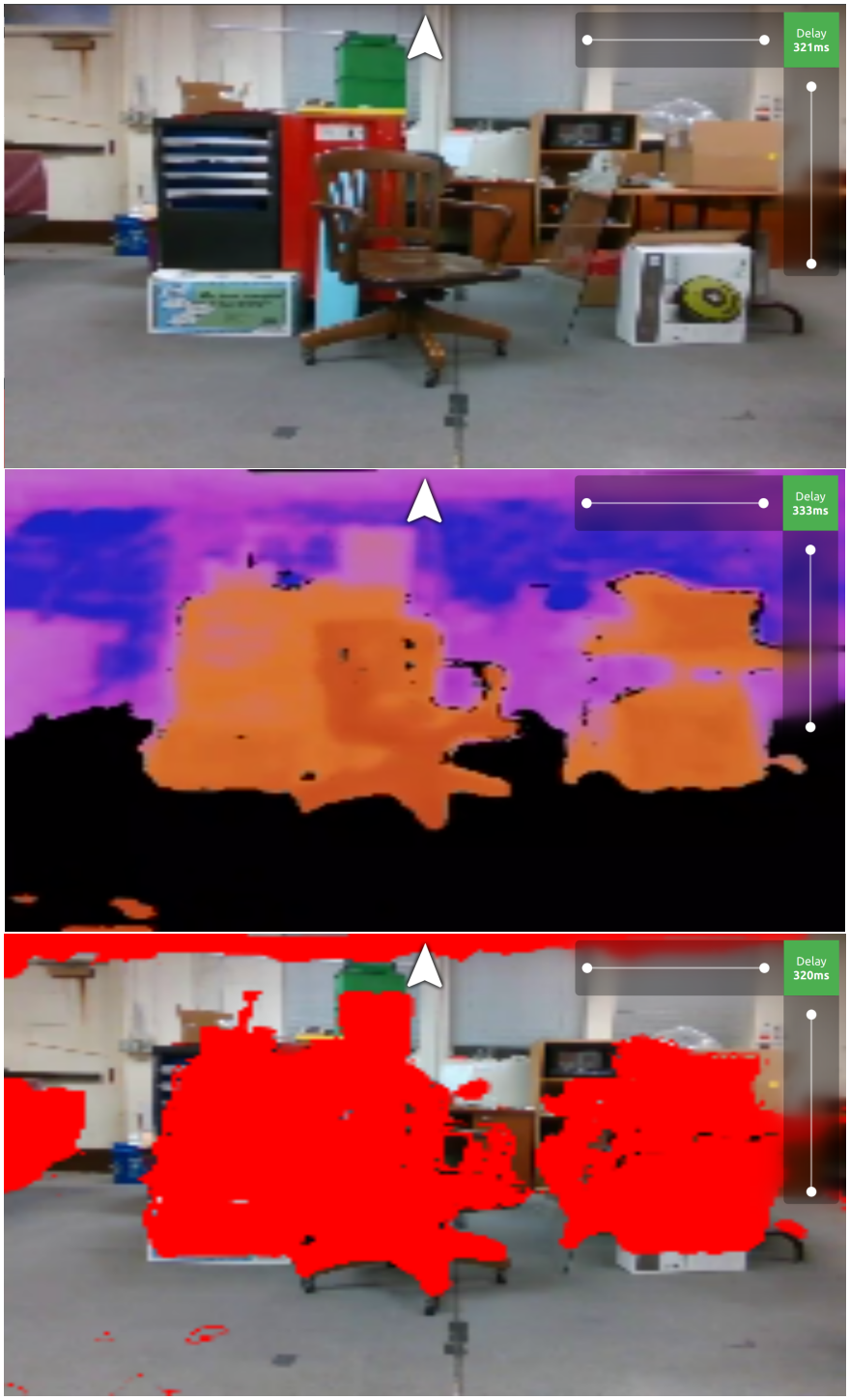

Figure 4.13: (Top) Input image to AR pipeline; (Middle) depth image of scene after ground filtering; (Bottom) final image delivered to user with AR overlay.

Figure 4.14: (Top) Input image to AR pipeline; (Middle) depth image of scene after ground filtering; (Bottom) final image delivered to user with AR overlay.

a tolerance of ±7.5 centimeters and tracking them so that navigating users can detect and avoid them, as shown in Figures 4.13 and 4.14. The AR overlay can currently be run at 0.5 frames per second at up to 640x480p resolution. Because the video is captured These additions allow for clear transmission of two new depth data streams to a navigator of the rover that are compatible with lunar conditions and delay, and can be tested against the existing stereo and mono streams to determine if they improve navigating user performance.

Chapter 5

Other Usability Improvements

In addition to the improvements made to the video subsystem by modifying the stereo-based depth visualization to involve the generation and use of a disparity map, additional usability improvements were made to other areas of the rover control program to improve quality of life and ease of understanding for testers and users. This chapter details the largest of these improvements, including changes made to the modifications made to the user HUD based on qualitative feedback from previous testing discussed in Chapter 3, and improvements to the data logging system which allows for easy data visualization by parsing the gathered data and generating graphs of rover speed and wheel power over the course of all trials in a test.

5.1 User HUD

Based on the user feedback obtained during the testing discussed in Chapter 3, it was clear that many parts of the current HUD design were not helpful to users attempting to teleoperate the rover. Figure 5.1 shows the HUD configuration used in previous testing with the rover; Widgets 1 and 3 were found to be helpful, but

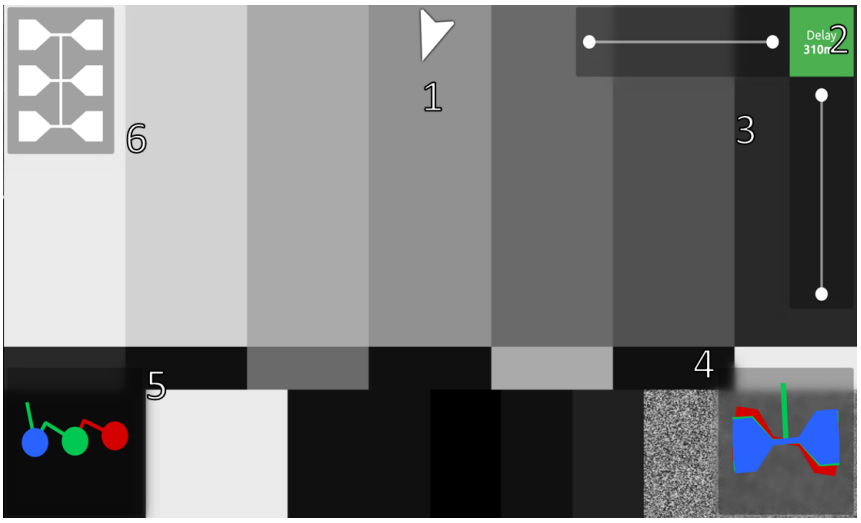

Figure 5.1: Main video window showcasing current rover HUD.

Widgets 2, 4, 5, and 6 were all found to be unnecessary, with some test participants rating them as actively unhelpful to navigating the rover. Thus, Widgets 2, 4, 5, and 6 were focused on when considering how to improve the user HUD.

Qualitative Average Scores, N=14, Maximum=3	
Mono Average	2.2, STDEV=0.6
Stereo Average	1.9 STDEV=0.7
Widget 1 (Compass)	2.8 STDEV=0.4
Widget 2 (Latency)	2.2 STDEV=0.4
Widget 3 (Directional Inputs)	3.0 STDEV=0.0
Widgets 4 and 5 (Rover Roll and Pitch)	2.1 STDEV=0.7
Widget 6 (Current Draw)	2.2 STDEV=0.4

Table 5.1: Average qualitative scores for both video options and all widgets used in testing.

Figure 5.2 shows the standard new HUD design, which was modified by leveraging user feedback with the functionality the widgets being altered provided to the user. Widget 2, which presents the current delay being applied/experienced by the video feed, was ultimately left unchanged, as the widget was considered mostly just vestigial by users, but it allows the tester to quickly see the delay being experi-

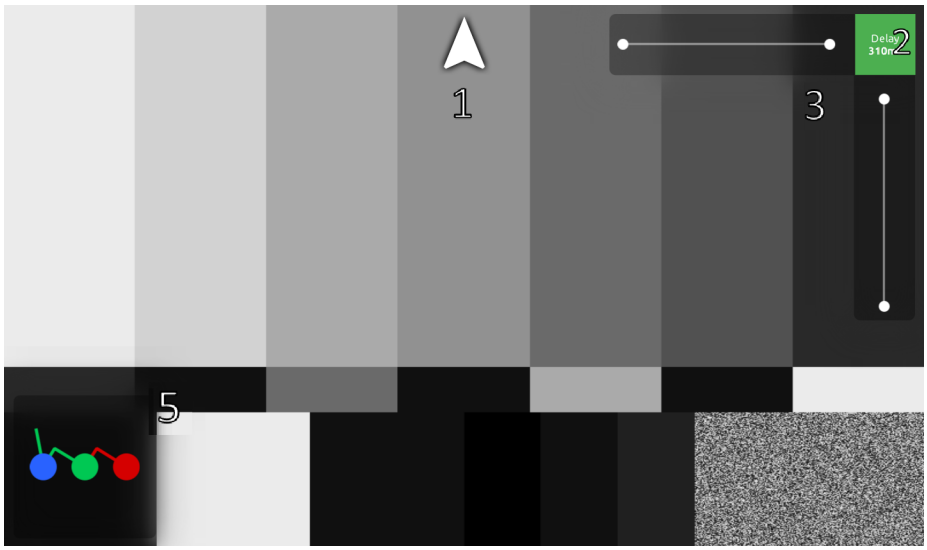

Figure 5.2: Main video window showcasing modified user HUD.

enced by the user in real-time while still watching the user's performance in the test. Widgets 4 and 5, which present the roll and pitch of the rover body, respectively, were the most harshly critiqued by users for their distracting and unhelpful nature. While it was clear that this set of widgets was not considered helpful by users, it was difficult to say if that was due to the nature of the widgets themselves or the state of the rover during the tests, as a malfunctioning IMU which was providing data to the widgets made them especially distracting. In order to further investigate this in future tests while still reducing the distraction caused by these widgets, Widget 4 has been disabled in the new design, as it was noted to be the more distracting of the two in short-answer feedback from test participants, and Widget 5 has been left unchanged. This design allows us to continue testing if these widgets have any positive impact on user navigation performance, while also cutting down on visual clutter in the HUD, increasing the area of the video feed easily visible by the user, and giving the design a more symmetrical appearance.

Widget 6, which visualizes the current draw of all wheel motors to indicate

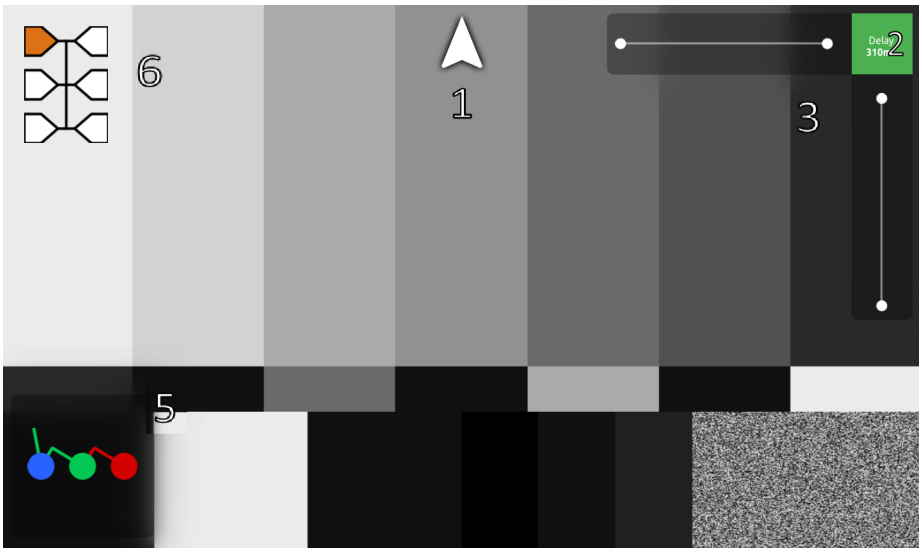

Figure 5.3: Main video window showcasing modified user HUD with wheel stall warning.

stalls, was also considered largely unnecessary by users, but its role in warning of high wheel current draw and troubleshooting connection issues meant that it still needed to be present in the HUD design in some fashion. Still, the information provided by this widget is not relevant to the user for most of their time spent navigating, so minimizing its impact would reduce clutter on the HUD and increase the portion of the video feed easily visible to the user. In order to achieve both of these requirements, Widget 6 was changed from a persistent indicator to a warning which flashes in the upper right hand corner when any wheel motors draw more than 65% of the pre-established critical current draw used in the previous HUD implementation, displaying the current draws of all other wheels in the process; an example of this behavior is shown in Figure 5.3. The widget activation level can be edited in the HUDPowerImpl.cpp file in the rover codebase if it is decided that the widget should activate and warn the testers and navigating user at a lower or higher current draw level. The current draw is measured by a current sensor within the wheel and sent back to mission control to be compared against the critical value

of 500mA, determined based on the 7600mAh current capacity of the wheel motors and the 180mA typical draw on the batteries from the wheels. Converting Widget 6 to a warning maximizes its effectiveness by ensuring it is out of the way when not necessary during testing, but noticeable if the rover is stalling and the user needs to cease commands to the rover so that it can be safely moved.

The new HUD design which will be utilized in future testing effectively combines the user feedback on HUD design obtained from previous testing while ensuring that the functionality of the HUD, both in terms of the user and the tester, remains intact. As further testing is performed, more user feedback will be obtained related to HUD design, which will allow consistent iterative improvement to maximize the HUD's effectiveness at aiding users in the teleoperation of the rover under high time delay, high latency, and low bandwidth conditions.

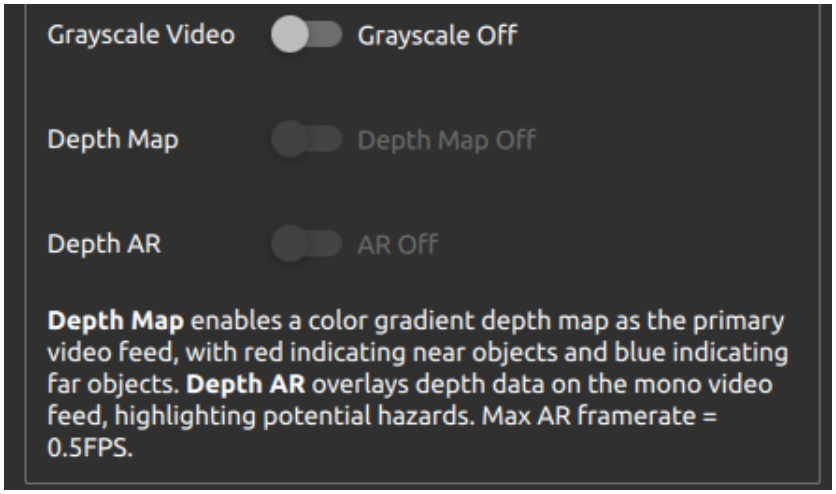

Figure 5.4: The new tester interface has two new buttons for enabling or disabling the Depth and AR video feeds - traditional stereo is now a camera option, instead of a feed option.

The control interface used by the tester during the rover tests was not changed in any significant way as a result of the changes made to the video feeds and user HUD; the only modification was the addition of additional switches to enable and

disable the new video feed options available for use on the rover, shown in Figure 5.4. These new buttons are dependent on each other - the AR video feed can only be enabled if the Depth video feed is also enabled, and is disabled if the depth feed is turned off or switched. No changes were made to the tester comment window.

5.2 Data Log Visualizer

As discussed in Chapter 3, the rover control program possesses a data logger which collects data about characteristics of the rover at a predetermined interval throughout operation, as well as information about the video feed configuration and tester comments whenever a change in the video feed occurs or a comment is recorded. This information is then logged to CSV files for future use and analysis; a Data file for rover data, a Settings file for video feed configuration information, and a Comments file for tester comments. While the on-demand population of the Settings and Comments files makes them relatively small for each test, the Data file collects data at a frequent fixed interval, meaning that the resulting files are often tens of thousands of lines long and extremely difficult to utilize for assessing user performance. In order to address this issue, a program was written to parse the produced data files and their corresponding settings files to produce graphs representative of all of the data contained in the CSV files, making them easy to use for assessing and visualizing user performance in a test. This program, known as DataPlotter, can be run on any existing pair of Data and Settings files to produce graphs displaying rover speed and wheel power for all trials in a given test.

DataPlotter is a simple data parser and trimmer, which incorporates a third-party library, MatPlotLibCpp, to generate graphs with the processed data. The program first prompts the user to input the names of the Data and Settings files

they will be using to generate graphs, and then reads through these files, extracting speed, wheel power, and time data from the Data file given, and video feed type, dimensions, and timestamps. Once this data is acquired, it must be divided into packets representing the data for each individual trial, as the Data, Settings, and Comments files are representations of the data collected over the course of an entire hour-long test. This is done by using the timestamps recorded for each video setting change in the Settings files and each line of data in the Data files, as during testing the video settings are only changed between trials to switch between video modes. This allows the program to create discrete packets for each trial, trimming off data from other trials and the data from the initial calibration phase at the beginning of each test.

Once these packets have been created, the program then uses MatPlotLibCpp to generate presentable graphs of the testing data. MatPlotLibCpp is a wrapper library created for MatPlotLib, a Python library designed to allow for easy, professional representation of 2D and 3D data in a variety of formats and styles [47]. MatPlotLibCpp, created by Benno Evers, wraps most of the functionality of MatPlotLib, and allows it to be easily ported into C++ [48]. By using MatPlotLibCpp, it was possible to quickly and easily take the trimmed packets of data produced by DataParser to generate graphs of rover speed and wheel power for all trials in a test. Examples of these graphs generated from previously collected rover data are shown in Figure 5.4 and 5.5.

The figures produced by DataParser allow for compact visualization of the tens of thousands of data points gathered over the course of a rover teleoperation test. Use of the program will make it easy to see trends in user navigation behavior, find correlations between difficult points in the user's navigation experience and rover data, and represent the data in presentations and papers.

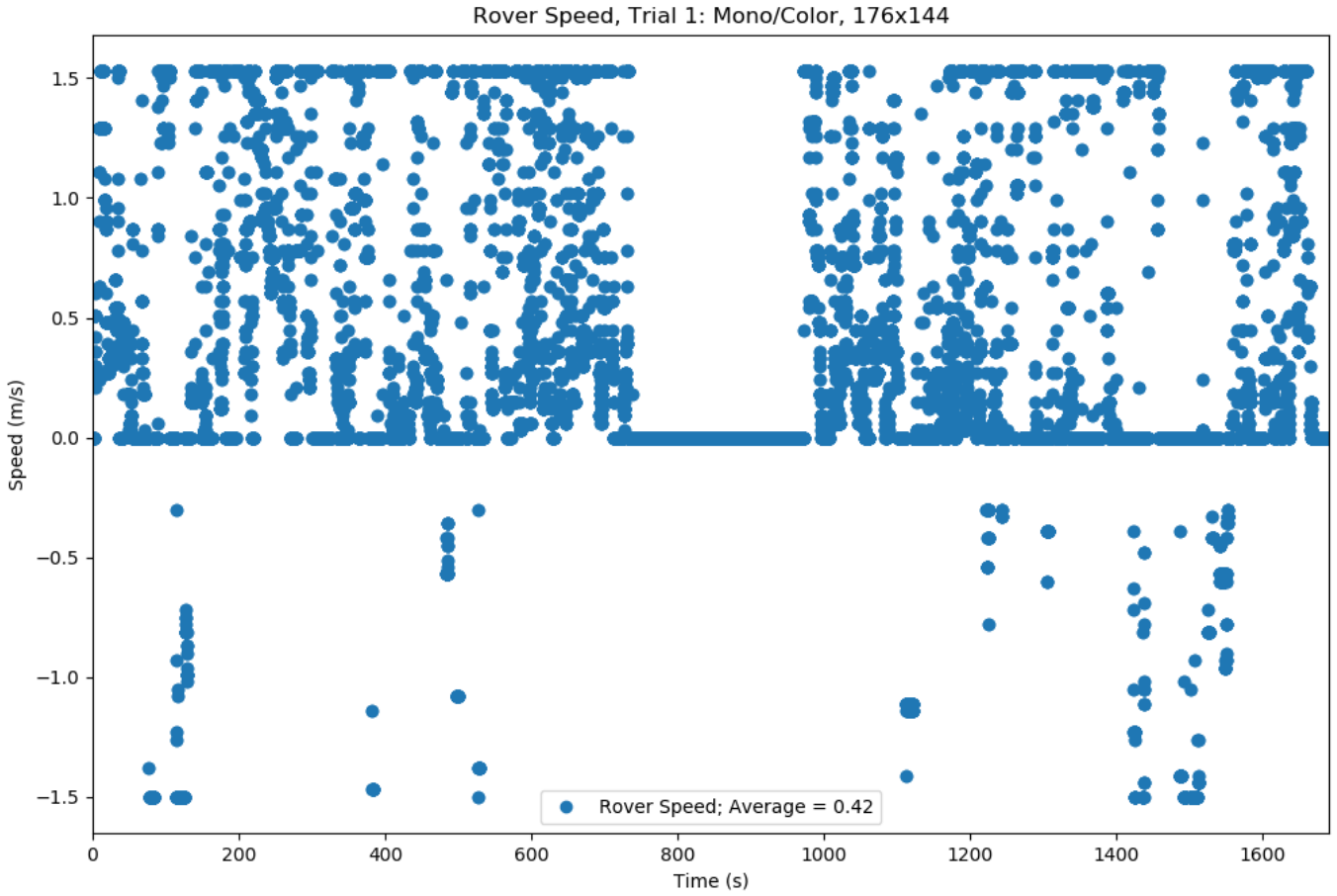

Figure 5.5: Graph produced by DataParser showing rover speed data from trial 1 of the rover teleoperation test performed on 11/19/18.

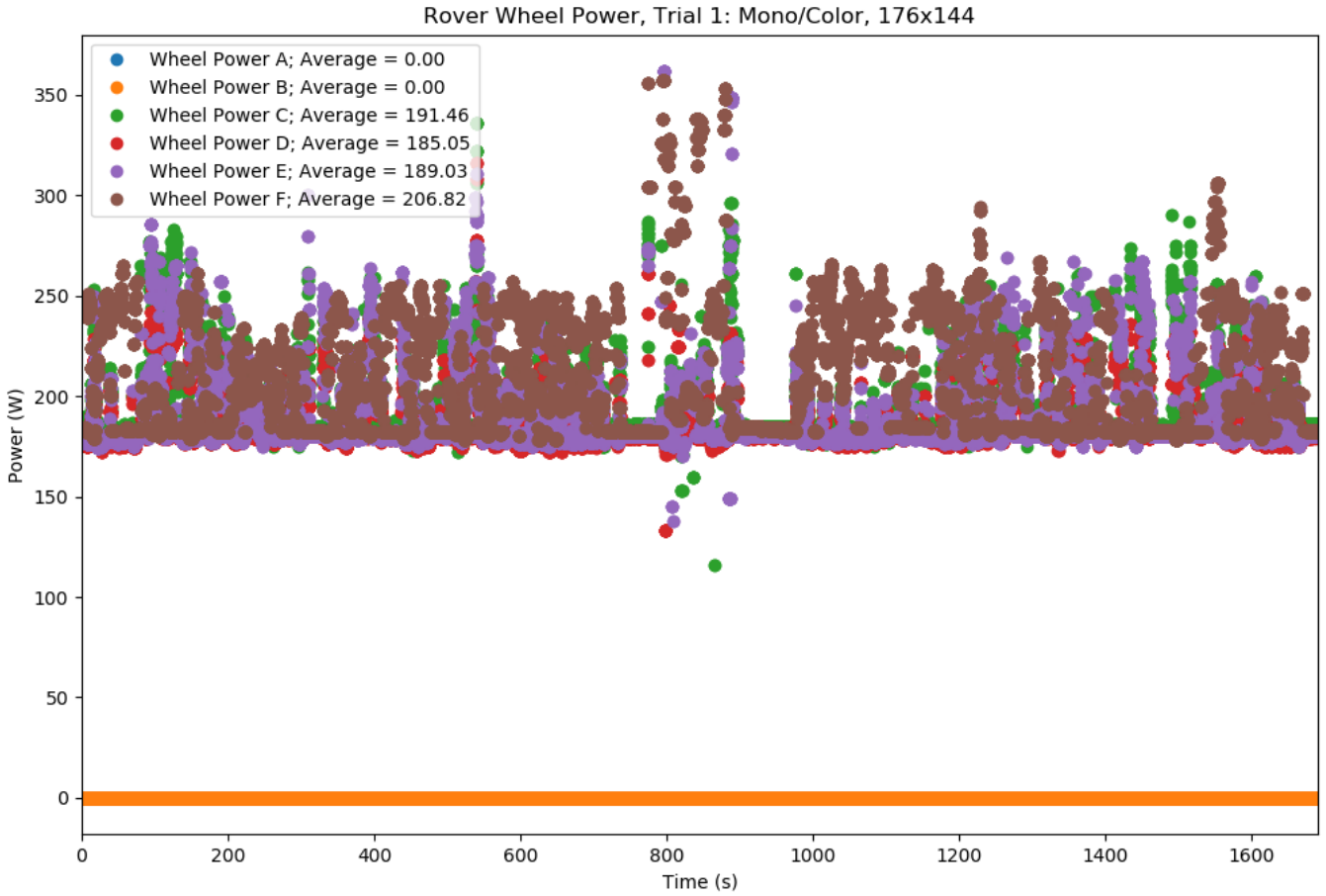

Figure 5.6: Graph produced by DataParser showing rover wheel power data from trial 1 of the rover teleoperation test performed on 11/19/18. The wheel labels correspond to the following wheel positions: A = Right Middle, B = Right Front, C = Left Front, D = Left Middle, E = Left Rear, F = Right Rear.

Chapter 6

Conclusion

This thesis has presented video feed improvements and additions for human navigator control of a low-bandwidth prototype lunar rover by incorporating a dedicated D435 depth video camera to allow for a pure depth video feed and an AR overlay of depth data on a mono video stream as potential video options, and by using user feedback to make usability improvements to the user Heads-Up Display overlaid on the video feed. A data log visualizer was also created to make it easy for testers to analyze data collected during a user test by parsing it and converting it into a pair of graphs of rover speed and power use over the course of the test. The AR overlay created as a result of this thesis is produced by using an image transform algorithm on captured depth images from the D435 depth camera to filter out the ground from the image and map it correctly to a mono RGB image of the same scene, and then running both images through a Pixel Influence Filter to highlight potential obstacles on the mono image based on pixel intensity on the depth image.

The contribution of this thesis is the two additional video stream options now available on the prototype rover being tested, which can be used to see if different, more exaggerated forms of depth information help a navigating user improve their

ability to navigate the rover when faced with the time delay and low bandwidth inherent in a long-distance remote teleoperation scenario like one could see in a lunar application. Additionally, the usability improvements made to the user HUD will improve user experience by reducing onscreen clutter during user navigation of the rover and the improvements made to data visualization will reduce data analysis time for testers, allowing for more clear and presentable testing results.

6.1 Future Work

The additional video streams created as a result of this thesis act as a proof of concept for their functionality in a human-operated rover teleoperation application, but both new video streams must be tested with multiple users alongside the existing stereo and mono options to see if they actually improve user performance and experience when navigating the rover. These tests could either be done in the manner explained in Chapter 3, involving user navigation of a set of predetermined courses, or could involve other, more concise trials involving small obstacles to closely check the various video stream options' ability to improve obstacle avoidance. While the pure depth and AR overlay options clearly present depth data to a navigating user, it is possible that the increased screen content could reduce user performance; this would need to be tested with a set of human navigators.

Improvements could also be made at various points of the creation of the AR overlay stream. The simplest improvement would be to implement dynamic plane detection for the ground detection algorithm, which would reduce the amount of image noise generated by movement of the camera during rover operation. While this would increase computation time, it would make the detection algorithm more robust, and would reduce the amount of time necessary to calibrate the camera

in-between runs.

The AR stream could also be improved by incorporating machine learning to reduce in-application computation time and increase accuracy of the created depth overlay maps. Machine learning could be used in the Pixel Influence Filter to allow it to better detect edges or objects which would be common obstacles, such as rocks, which would allow for a more robust obstacle highlighting algorithm than the intensity matching system currently employed. Additionally, the image transform algorithm could be improved by incorporating a RANdom SAmple Consensus (RANSAC) algorithm for ground segmentation, like that utilized by Yang et. al. [42], which would allow for ground plane detection by performing a random search for ground near the perspective of the cameras, assumed to be the largest plane in the image being analyzed. The RANSAC use case proposed in [42] functions by randomly selecting a set of 3D world coordinates generated by the image transform, determining the plane the coordinates lie in, and determining how many of the other points in the image lie within this plane. The plane with the most "inlier" points discovered is considered to be the ground plane, and could then be filtered out. Incorporating RANSAC could provide a more robust ground detection method that would require less assumptions given enough training time. Incorporating machine learning tactics such as these could also help to deal with the obstacle cutoff problems present in the current design, as more robust ground and object detection would increase the ability of the algorithm to properly section ground. This would, in turn, make it easier to detect smaller obstacles whose height is closer to the height of the ground plane, which would give the navigating user more information to use for effective piloting of the rover.

Bibliography

[1] Steven J. Dick. *The Importance of Exploration*. Tech. rep. 1. NASA, 2007. URL: https://www.nasa.gov/missions/solarsystem/Why_We_01pt1.html.

[2] *Benefits Stemming from Space Exploration*. Tech. rep. International Space Exploration Coordination Group, Sept. 2013.

[3] Melanie Whiting and Laurie Abadie. *5 Hazards of Human Spaceflight*. Tech. rep. Aug. 2018. URL: https://www.nasa.gov/hrp/5-hazards-of-human-spaceflight.

[4] *Communication Delay*. URL: https://www.spaceacademy.net.au/spacelink/commdly.htm.

[5] Thomas B. Sheridan. "Supervisory Control". In: *Telerobotics, Automation, and Human Supervisory Control*. MIT Press, Aug. 1992.

[6] Reid Simmons et al. *Experience with Rover Navigation for Lunar-Like Terrains*. Tech. rep. The Robotics Institute, Carnegie Mellon University; Sandia National Laboratories.

[7] *Mars in a Minute: How Do Rovers Drive on Mars?* URL: https://www.jpl.nasa.gov/edu/learn/video/mars-in-a-minute-how-do-rovers-drive-on-mars/.

[8] Eric Krotkov and Reid Simmons. "Perception, Planning and Control for Autonomous Walking with the Ambler Planetary Rover". In: *The International Journal of Robotics Research* (1996).

[9] Terry Huntsberger. "Biologically Inspired Autonomous Rover Control". In: *Autonomous Robots* 11 (Aug. 2001).

[10] Markus Wilde, Sean C. Hannon, and Ulrich Walter. "Evaluation of Head-Up Displays for Teleoperated Rendezvous and Docking". In: *IEEE* (2012).

[11] Darwin G. Caldwell et al. "Sensory Requirements and Performance Assessment of Tele-Presence Controlled Robots". In: The Institution of Electrical Engineers, Apr. 1996.

[12] Alan M. Thompson. *The Navigation System of the JPL Robot*. NASA. 1977.

[13] Don Murray and Jim Little. "Using Real-Time Stereo Vision for Mobile Robot Navigation". In: *Autonomous Robots* 8 (2000).

[14] Nicolas Socquet, Didier Aubert, and Nicolas Hautiere. "Road Segmentation Supervised by an Extended V-Disparity Algorithm for Autonomous Navigation". In: *2007 IEEE Intelligent Vehicles Symposium*. IEEE, 2007.

[15] Nizar Fakhfakh, Dominique Gruyer, and Didier Aubert. "Weighted V-disparity Approach for Obstacles Localization in Highway Environments". In: *2013 IEEE Intelligent Vehicles Symposium*. IEEE, 2013.

[16] Steven B. Goldberg, Mark W. Maimone, and Larry Matthies. "Stereo Vision and Rover Navigation Software for Planetary Exploration". In: *2002 IEEE Aerospace Conference*. IEEE, 2002.

[17] Martin J. Schuster et al. "The LRU Rover for Autonomous Planetary Exploration and its Success in the SpaceBotCamp Challenge". In: *2016 IEEE International Conference on Autonomous Robot Systems and Competitions*. IEEE, 2016.

[18] *Depth Map from Stereo Vision*. Tech. rep. 2015. URL: https://docs.opencv.org/3.1.0/dd/d53/tutorial_py_depthmap.html.

[19] Guido Gerig. *Image Rectification (Stereo)*. Tech. rep. 2012. URL: http://www.close-range.com/docs/Image_rectification.pdf.

[20] Robyn Owens. "Epipolar Geometry". 1997. URL: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT10/node3.html#SECTION00031000000000000000.

[21] *Camera Models and Parameters*. Tech. rep. URL: http://ftp.cs.toronto.edu/pub/psala/VM/camera-parameters.pdf.

[22] D. N. Jarrett. *Cockpit Engineering*. Ashgate Pub., 2005.

[23] Karlin Bark et al. "Personal Navi: Benefits of an Augmented Reality Navigational Aid Using a See-Through 3D Volumetric HUD". In: Honda Research Institute USA. Sept. 2014.

[24] Marcus Tonnis, Gudrun Klinker, and Marina Plasvec. *Survey and Classification of Head-Up Display Presentation Principles*. Tech. rep. Technische Universitat Munchen.

[25] *HD Pro Webcam C920*. Logitech, 2019. URL: https://www.logitech.com/en-in/product/hd-pro-webcam-c920.

[26] *55" Class (54.6" Diagonal) UHD 4K Smart 3D LED TV w/ webOS*. LG, 2019. URL: https://www.lg.com/us/tvs/lg-55UB9500-led-tv.

[27] *GStreamer Features*. 2019. URL: https://gstreamer.freedesktop.org/features/.

[28] *GStreamer Qt Bindings*. 2019. URL: https://gstreamer.freedesktop.org/modules/qt-gstreamer.html.

[29] *videomixer*. GStreamer. URL: https://gstreamer.freedesktop.org/documentation/videomixer/index.html?gi-language=c.

[30] *OpenCV: Miscellaneous Image Transformations*. 2014. URL: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html.

[31] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, 2002, pp. 97–103.

[32] *OpenCV: Operations on Arrays*. Apr. 2019. URL: https://docs.opencv.org/3.4/d2/de8/group__core__array.html.

[33] *OpenCV: Basic Structures*. 2014. URL: https://docs.opencv.org/2.4/modules/core/doc/basic_structures.html?highlight=setto.

[34] Miguel Casas-Sanchez. *GstDisparity.cpp*. GStreamer, 2013. URL: https://github.com/GStreamer/gst-plugins-bad/blob/master/ext/opencv/gstdisparity.cpp.

[35] Rachna Verma and A. K. Verma. "A Comparative Evaluation of Leading Dense Stereo Vision Algorithms using OpenCV". In: *International Journal of Engineering Research and Technology* (). ISSN: 2278-0181.

[36] *Intel RealSense Depth Camera D400-Series*. Intel. URL: https://www.mouser.com/pdfdocs/Intel_D400_Series_Datasheet.pdf.

[37] *Up Squared Specifications*. Up. URL: https://up-board.org/upsquared/specifications.

[38] *v4l2src*. GStreamer. URL: https://gstreamer.freedesktop.org/documentation/video4linux2/v4l2src.html?gi-language=c.

[39] *Colorizer.cpp*. Intel, 2017. URL: https://github.com/IntelRealSense/librealsense/blob/master/src/proc/colorizer.cpp.

[40] *appsrc*. GStreamer. URL: https://gstreamer.freedesktop.org/documentation/app/appsrc.html?gi-language=c.

[41] OpenCV Dev Team. *Geometric Image Transformations*. OpenCV, 2014. URL: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html.

[42] Kailun Yang et al. "Expanding the Detection of Traversable Area with RealSense for the Visually Impaired". In: *Sensors (Basel)* 16 (2016).

[43] Kyle Simek. *Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix*. Aug. 2013. URL: http://ksimek.github.io/2013/08/13/intrinsic/.

[44] *rs2_intrinsics Struct Reference*. Intel. URL: https://intelrealsense.github.io/librealsense/doxygen/structrs2__intrinsics.html.

[45] *Sensor Control Sample*. Intel, 2017. URL: https://github.com/IntelRealSense/librealsense/tree/master/examples/sensor-control.

[46] *Intel RealSense Depth Camera D435*. Intel, 2019. URL: https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435.html.

[47] Matplotlib Development Team. *Matplotlib*. NumFocus, 2018. URL: https://matplotlib.org/gallery/index.html.

[48] Benno Evers. *matplotlib-cpp*. 2019. URL: https://github.com/lava/matplotlib-cpp.