Automating the Operation of a 3D-Printed Unmanned Ground Vehicle in Indoor Environments

Utkarsha Bhave, Grant D Showalter, Dalton J Anderson, Cesar Roucco, Andrew C Hensley, and Gregory C Lewin
University of Virginia, ub4zy, gds2wz, dja3vf, cr3fd, ach4uq, gcl8a@virginia.edu

Abstract - **The United States Department of Defense anticipates unmanned systems will be integrated into most defense operations by 2030 to reduce risk to human life, enhance reliability, and ensure operation consistency and efficiency. However, current technology requires human operation for ethical decision-making, leaving an opportunity to automate some tasks to assist operators. Previously, a University of Virginia capstone team designed an unmanned ground vehicle ("the rover") to aid intelligence, surveillance, and reconnaissance missions in adversarial environments. However, a lack of GPS connectivity indoors and system latency limited the rover's performance and created a lag in the operator's view compared to the rover's true position, occasionally causing the operator to inadvertently crash the rover into obstacles. The objectives of this project are to mitigate operational risks by equipping the rover with functionalities to autonomously avoid obstacles, map an unknown indoor space, and navigate itself back to a predetermined location ("base"). Obstacle avoidance is accomplished through an algorithm that stops the rover a safe distance away from a detected obstacle, but still allows the human operator to navigate the rover away from the obstacle prior to continuing the mission. Algorithms are implemented to perform Simultaneous Localization And Mapping and to determine best-route navigation to the base. Laser rangefinder data, an improved processor, and, potentially, visual odometry sensors are used to aid in the navigation algorithms. Testing has confirmed that the rover successfully stops in front of laser-detected obstacles, builds digital maps of an unknown indoor space, and can navigate back to a base, though the performance has room for improvement. It is anticipated that incorporating visual odometry can enhance the rover's mapping implementation and obstacle avoidance performance.**

Index Terms – Adversarial environment, autonomous, reconnaissance, unmanned

INTRODUCTION

Semi-autonomous and, more recently, autonomous designs for military vehicles have become common on modern battlefields and show signs that they may phase out fully human-operated systems in the future. Semi-autonomous systems perform some tasks independently and rely on a human operator for high-level decision-making, while fully autonomous systems operate without real-time human control [1]. There are numerous advantages to machines with some degree of automation, such as increased operational flexibility and functionality [1]. For land operations, particularly in urban areas, the Department of Defense has shown increased interest in semi-autonomous wheeled vehicles, as they reduce the risk to humans in high-risk situations and can perform intelligence, surveillance, and reconnaissance (ISR) missions [2]. The purpose of ISR missions is often to assist humans in familiarizing themselves with an adversarial environment prior to entering the space themselves. Therefore, autonomous elements of vehicle control must work in conjunction with human direction during a mission and any autonomy that supersedes human intervention must be justified and thoroughly tested.

Recent Unmanned Ground Vehicle (UGV) advancements have improved the handoff of control between the operator and the machine [2]. However, to facilitate a seamless exchange of operational control, the military has sought designs with a high degree of communication, which allows operators to better monitor the machine [3]. Therefore, future designs will have to be able to capture and pass large amounts of sensor feedback to human operators. In addition, these designs will need onboard processing power to use that data to make decisions in real-time. The sensory and computational demands on unmanned vehicles is heightened during reconnaissance missions, where collecting data on a vehicle's surroundings is paramount [1]. As such, the ability of machines to autonomously conduct some tasks alongside human control not only can make those machines more reliable and effective, but it can also keep humans out of harm and focused on other operational objectives.

To produce a vehicle with semi-autonomous capabilities, a previous University of Virginia capstone team designed an unmanned ground vehicle ("the rover"), built using 3D-printed and commercial off-the-shelf ("COTS") parts. This design allowed non-trained personnel to easily build, repair, replicate, and deploy one on their own. This vehicle needed to be able to operate indoors or outdoors while providing reconnaissance information to the operator. Outdoors, the rover used GPS data to determine its location. However, a lack of GPS connectivity indoors and system latency limited the rover's performance and created a lag in the operator's view compared to the rover's true position, occasionally causing the operator to inadvertently crash the rover into

obstacles. These control issues caused by indoor operation will be solved in this project by adding new capabilities that mitigate the risks posed to the rover's operation.

Objectives and Metrics

The objectives of this project are to mitigate operational risks and improve rover performance in adversarial, GPS-denied environments, such that the human operator may explore and monitor an unknown environment prior to humans entering the space themselves. These objectives were determined in discussions with the client for the rover's use in ISR missions. In indoor spaces, there is a significant delay between the rover's true position and the operator's video feed, emphasizing the need for autonomous object avoidance, as the rover may crash into surroundings before the operator can react. In addition, when the space has been explored, it could be more efficient for the rover to return to the operator autonomously than to have the operator navigate the rover manually. As such, autonomous functionalities to avoid obstacles, map an unknown indoor space, and navigate to a predetermined location ("base") will be added to the system.

Through client discussions, it was determined that the goal of testing was not to meet specific user criteria, but rather to determine what is feasible and to identify the rover's limitations. Thus, metrics were identified in association with each of the three objectives to determine the rover's level of success in the completion of each objective.

For obstacle avoidance, it was determined that the objective would be satisfactorily met if the rover stopped a safe distance away from a detected obstacle, namely at least half of the length of the rover's longest diagonal, which would allow the human operator to navigate the rover away from the obstacle prior to continuing the mission.

Since the objective to map an unknown indoor space may be to assist humans in familiarizing themselves with an adversarial environment prior to entering the space themselves, the map should closely resemble what obstacles a human would encounter. In robotics, the ability for a rover to know its real-time position and orientation ("pose") within a frame of reference is known as localization. One solution is to implement Simultaneous Localization And Mapping (SLAM), a method of mapping surroundings while in motion by concurrently localizing and processing sensor data, for example from a laser rangefinder ("LiDAR"), which measures distances to its surroundings by projecting laser signals and sensing their reflections. Although humans would perceive obstacles in a 3D manner, it was determined that a 3D LiDAR would be too costly to fit the use case needs of primarily using COTS parts, limiting mapping to a 2D plane at the height of a 2D LiDAR mounted on the rover. Thus, it was determined that the metric would be a qualitative assessment comparing the rover-produced map to the 2D cross-section of the space at the height of the LiDAR.

The next objective was to add functionality for the rover to navigate back to a user-defined base, which is measured as the percent of trials where the rover returns to within a meter of the user-defined base.

Current Design

The rover's current design is shown in Figure I, which also identifies the system's main components. The On-Board Computer (OBC), autopilot, LiDAR, and antennas are all mounted on the rover's 3D-printed chassis, which houses most of the electronics. The antennas are used solely for Wi-Fi connectivity during development and debugging phases, while the final system will rely on a form of radio telemetry which is beyond the scope of this project. Additionally, not pictured is a camera used to provide the operator a first-person view video feed of the environment.

The OBC of choice is a NVIDIA Jetson TX2 module, mounted on an Orbity Carrier Board, running the Robot Operating System (ROS). ROS is a middleware, or collection of software packages, that serves as a framework for integrating tools and tasks commonly needed in autonomous systems. As an open-source framework, ROS allows developers to integrate software and hardware as needed to perform specific tasks.

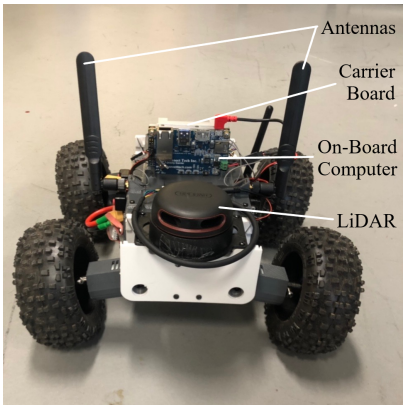

FIGURE I
FRONT VIEW OF CURRENT ROVER SYSTEM PHYSICAL STRUCTURE

In order to better understand how ROS interacts with the rest of the system, it is important to define certain terms that will be used throughout this paper, namely packages, nodes, topics, and messages. A ROS message is a data structure that encapsulates information being used by the system. These messages are exchanged between nodes over multiple, shared communication channels called ROS topics. A ROS node is a piece of code that is able to read ("subscribe to") and modify ("publish data to") topics, in order to perform tasks. Finally, a ROS package refers to a bundle of code and files, typically consisting of ROS nodes and parameters.

The LiDAR, as seen in Figure I near the front of the rover, is a RPlidarA2, for which ROS provides a package to do basic data processing. Figure II shows the relationships and information flow between the major subsystems. ROS-Kinetic middleware on an Ubuntu 16.04 operating system is installed on the OBC, which runs several ROS packages. RPLidar is a package that reads the data coming from the LiDAR and publishes it to other nodes that rely on the data stream. Google Cartographer SLAM ("Cartographer") is a

package that subscribes to the incoming data from the LiDAR, generates a map based on the readings, and determines the rover's pose in the map. The PixRacer is a lower-level microcontroller running ArduPilot software, which is responsible for controlling and monitoring low-level hardware, such as motors. It is also used to navigate outdoors, where GPS is available. ArduPilot software and compatible hardware use the MAVLink communication protocol often used for small-scale autonomous vehicles. For the OBC to establish bidirectional communication with the PixRacer and then read and modify its state and parameters, ROS uses the MAVROS package, which converts ROS topics to MAVLink messages that the autopilot can understand.

A UGV for ISR missions should be able to adapt to any indoor and outdoor adversarial environment without getting lost. However, indoor and outdoor navigation rely on different localization techniques. In outdoor environments, the rover can use GPS data to determine its location by interpreting signals from satellites orbiting the Earth. However, system latency and GPS signal interference limit the rover's localization and navigation performance in indoor environments. Since Cartographer subscribes to incoming MAVLink-format LiDAR data and publishes it to the PixRacer through the MAVROS node, the PixRacer can shift from using a combination of GPS, magnetometer, and compass measurements to instead, in indoor spaces, using LiDAR data with its internal accelerometer and gyroscope as guidance inputs.

While the operator is remotely controlling the rover, there can be significant lag in the video feed, which may result in the operator crashing the rover into an object before the operator can react. To implement obstacle avoidance, a custom node subscribes to the LiDAR data and publishes MAVLink messages through the RCOverride topic in MAVROS. As the name would imply, RCOverride makes the autopilot ignore remote control commands and replace them with incoming MAVLink messages. Implementing obstacle avoidance required solving two main problems: determining how to detect an object and planning the rover's reaction to detected obstacles. LiDAR data consists of distance measurements at many discrete angles in a 2D plane; the RPlidarA2 collects 100 readings over 360 degrees. However, LiDAR readings often contain noise, or aberrant readings, visible in the list of distances scanned. Therefore, only looking at the smallest distance reading would lead to many false positives, in that random noise could indicate a nearby object. To avoid this, the custom node algorithm averages the 7 closest distance readings from a 60-degree cone directly in front of the rover. Once the average decreases to less than a certain distance threshold, the rover registers it as an obstacle. Because the client expressed that the operator should have the highest degree of control for seamless simultaneous human- and autonomous-control, it was decided that the system must stop in front of, rather than autonomously navigate around, an obstacle to allow the user to maintain control of the rover. So, when an object is registered, the OBC halts forward and backward linear motion using RCOverride. However, since turning motion is still possible, the operator may turn the rover in place. Once the rover's path forward is no longer obstructed, the OBC releases the RCOverride, allowing the rover to move freely again.

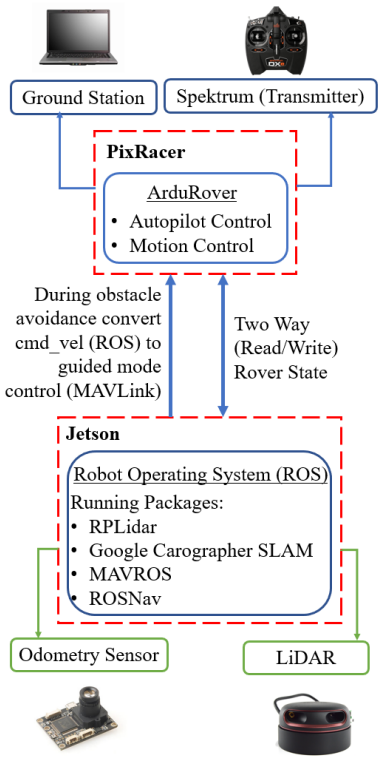

FIGURE II
TECHNICAL DIAGRAM OF RELATIONSHIPS BETWEEN ROVER SUBSYSTEMS

Finally, to allow the rover to autonomously navigate to a user-defined base in GPS-denied environments, the ROSNav autonomous navigation package was implemented. This package overrides the GPS-based guided mode that the PixRacer traditionally supports. Rather than comparing the rover's GPS location to the desired GPS coordinates, the rover instead localizes using the map made with SLAM. This map has its own coordinate system as well as "no-go" zones, or solid objects read by the LiDAR as obstacles. The navigation package then implements a shortest-path algorithm (A-Star) to determine what route the rover should take while concurrently avoiding obstacles. ROS-based velocity messages are then translated into a format which the PixRacer can use via MAVROS. In its current state, the rover remembers its activation location and can navigate back to that location when a RC controller switch is toggled.

DESIGN DECISIONS

Physical Structure

The majority of the rover's current physical design had been laid out by previous work on the system, though minor changes were made to accommodate upgraded hardware components, detailed later in this paper. The design employs

skid steering and all-wheel drive to reduce complexity in construction. In addition, skid steering provides the ability to turn in place, which is ideal for constrained indoor locations. The wheels extend beyond the front of the rover, allowing it to climb over small obstacles. Each wheel has spring-based suspension, increasing the rover's motion smoothness and reducing LiDAR data noise. However, skid steering hinders the SLAM algorithms' ability to localize the rover. Particularly, during fast turns, algorithms are unable to compare incoming LiDAR data to the previously established map, often causing the map to become skewed. Additionally, wheel slippage common to skid steering limits the reliability of some odometry sensors, such as motor encoders.

Hardware

The system, with architecture as seen in Figure III, has two distinct subsystems, one centered around the autopilot and the other around the OBC. The rover uses an ArduPilot PixRacer as its autopilot because the client uses other products that rely on the PixRacer and because ArduPilot has readily available, compatible hardware. The PixRacer was also retained for its effectiveness in outdoor operation, as ArduPilot uses GPS for many of its localization algorithms, velocity calculations, and GPS waypoint-based autonomous features. This autopilot also removes the need for ROS to handle any motor control task or chassis-specific requirements. So, if the physical rover is ever changed, ROS and software solutions would need little to no updates. The rover must be capable of indoor operation, where GPS is often unavailable. Distance sensor options include 3D LiDARs and ultrasonic sensors, but the RPlidarA2, a 2D LiDAR, was preferred because it fit use case cost constraints while serving the same purpose as a tool to aid in mapping and obstacle avoidance algorithms. In addition, documentation also suggested it would interface well with ArduPilot, and the chosen LiDAR provides system designers with flexibility to choose the angle in front of the vehicle from which to collect data for obstacle avoidance.

Adding the desired higher-level functionalities to the existing system design requires more processing power than using the ArduPilot PixRacer. As the PixRacer is the rover's autopilot, it directly communicates with the lower-level hardware, such as the electronic speed controllers (ESCs) and the motors. To meet the higher-level processing capacity for autonomous obstacle avoidance, navigation, and SLAM in GPS-denied environments, choosing an OBC to add to the system was an important decision. The Raspberry Pi ("RPi") would fit the cost constraint due to its low price as well as its widespread use with ArduPilot and similar rover applications. However, the RPi proved to have insufficient processing power for required tasks, causing it to shut down during use due to high load. As such, the RPi was replaced with the NVIDIA Jetson TX2 ("Jetson"), as the Jetson provides the needed higher processing capacity despite a higher cost. The Jetson also enables scalability, as its higher processing capacity would allow for additional automation to work concurrently alongside current functionality.

Software

Although other platforms, such as MAVProxy and DroneCode, interact with the autopilot in a similar manner, ROS was chosen for its ease of SLAM implementation. In addition, ROS was the best option for autonomous navigation and for easy expansion beyond the scope of this project, as there were several packages already modified for work in conjunction with ArduPilot hardware.

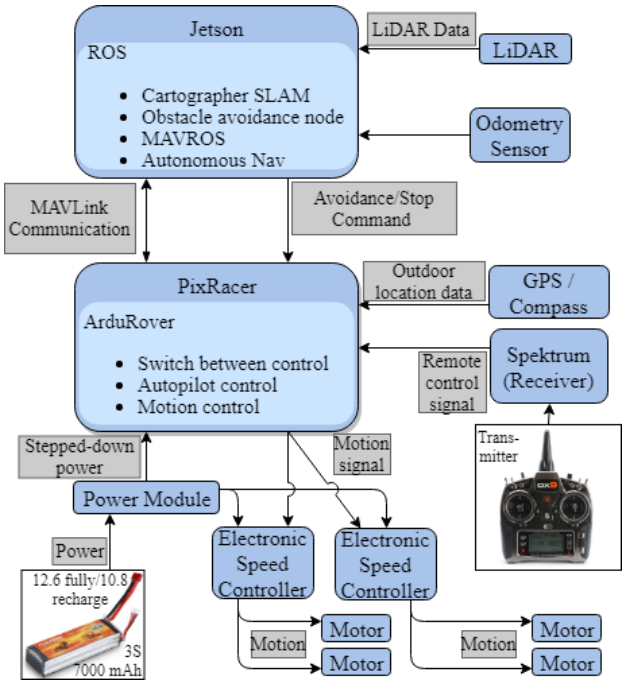

FIGURE III
ROVER SYSTEM ARCHITECTURE

The autopilot is connected to the RC controller and to a pair of ESCs, allowing the autopilot to convert signals from the controller to motor operation, which is the standard configuration for ArduPilot based vehicles. The OBC, running ROS is connected to the LiDAR and other sensors required by ROS packages. A key project challenge was to determine how to allow the OBC to control the power outputs from the ESCs when needed. Initial architectures included the implementation of a multiplexer with two inputs – one from the autopilot and the other from the OBC – with the output to the ESCs. In default, the autopilot's signal would be output, but when the LiDAR registered an object, the mux's input would be triggered and an alternate signal would be output from the OBC. Since additional physical hardware would complicate the system and increase costs, a software-based solution was deemed preferable. This solution came in the form of the RCOverride topic. In order to streamline the communication process, the LiDAR was connected to the OBC, as transmitting the data from the LiDAR to ROS would be easily achieved in this manner. Although some LiDARs may be attached to ArduPilot-based autopilots, accessing the LiDAR data through MAVROS would be difficult, and most ROS packages work under the assumption that the LiDAR is connected directly to the OBC. In this setup, the LiDAR

scans the environment for obstacles, software interprets how far and where those obstacles are, and then the software sends RCOverride commands to the autopilot to halt the rover.

To ensure that the system is able to effectively map new environments, a robust SLAM algorithm must be employed. For the purposes of this project, three SLAM algorithms were examined: Hector SLAM, Gmapping, and Cartographer. Hector SLAM is capable of localizing the rover with LiDAR and an IMU alone, while Gmapping requires other external odometry inputs. Cartographer is adaptable, in that external odometry inputs are optional. To date, a decision on the preferred algorithm has yet to be finalized, as only Hector SLAM and Cartographer – without odometry – have been tested. However, Cartographer proved to be more robust after initial tests, resulting in its inclusion in the current system.

TESTING AND RESULTS

Testing was performed to evaluate how well the rover fulfilled key project objectives: autonomous obstacle avoidance, mapping a GPS-denied adversarial environment, and navigation to a user-defined base. The metrics developed are measured on a pass/fail basis, with the purpose of these results being to guide future design improvements and prepare the rover for real-world scenarios it might encounter.

Obstacle Avoidance

The obstacle avoidance algorithm was tested under different scenarios evaluated as the percent of trials in which the rover passed. If the rover stopped more than 20 cm – half of the distance of its longest diagonal – away from an obstacle, then it passed the test; otherwise, it failed. During obstacle avoidance tests, the rover, running the LiDAR and obstacle avoidance ROS nodes, was remotely controlled. The operator drove the rover towards a set of predetermined obstacles to test system response to obstacles of different widths – "thin" if less than 4 cm in width and "wide" otherwise – and reflectivity. Results of testing can be seen in Table I.

TABLE I
RESULTS OF OBSTACLE AVOIDANCE TESTING

Scenario	Metric	Results
Thin Obstacle	Stops more than half the diagonal away	20%
Wide Obstacle	Stops more than half the diagonal away	90%
Reflectivity	Stops more than half the diagonal away	100%

The rover stopped in about 20% of thin obstacle trials and succeeded in about 90% of wide obstacle trials. Since the rover stopped in 100% of the trials involving obstacles of different reflectivity, it was determined that reflectivity did not significantly affect obstacle detection.

Mapping

The SLAM algorithms were tested under a pass/fail criterion. If the implementation's generated map was not skewed after turning the rover in place, it passed; otherwise, it failed. The four implementations to test are Hector SLAM (no odometry), Cartographer (no odometry), Cartographer (with odometry), and Gmapping (with odometry). As the latter two implementations require a source of odometry, they cannot be tested until an adequate form of odometry is added to the system. Testing results for implementations not involving odometry can be seen in Table II. Figure IV shows the space that was mapped (left) and the results of Cartographer SLAM (center) and Hector SLAM (right). While the Cartographer map looks better, it still has some issues, such as the large bump protruding from the island.

TABLE II
RESULTS OF NON-ODOMETRY SLAM IMPLEMENTATIONS TESTING

Implementation	Metric	Results
Hector SLAM	Map quality maintained in operation	Fail
Cartographer	Map quality maintained in operation	Fail

FIGURE IV
MAPPING RESULTS

ONGOING DEVELOPMENT

SLAM algorithms require hardware, usually a LiDAR, which is occasionally supplemented with an Inertial Measurement Unit (IMU). However, some algorithms require external sources of odometry – the use of motion sensors to determine the robot's movement relative to some known position – such as motor encoders or visual odometry sensors [4]. Therefore, the system will need a reliable source of odometry to enhance ROS SLAM package capabilities. Motor encoder odometry works by estimating the rover's movement based on motor (wheel) motion. Due to the high-slip nature of skid steering, this method was discarded.

Visual odometry, a form of odometry in which data from a camera in real time using sequential images is used to estimate motion, has proven useful for similar systems [5]. The particular odometry sensor of interest is the PX4 Flow ("Flow"), an optical flow sensor, which, using a camera, tracks the relative positions of a pattern of pixels frame-to-frame in order to determine its motion relative to the surface it is pointing towards, typically the ground. The Flow was chosen to test due to its ease of integration and claimed robustness to low-light situations. If determined to calculate odometry accurately, the Flow could be used to improve the SLAM algorithms' performance. As testing has not yet been finalized, the system does not currently include the Flow, but, if implemented, the sensor will potentially be placed at the back of the rover facing towards the floor. This sensor would be integrated with the navigation algorithms in ROS.

A commercial pre-made robot, the TurtleBot 3 Burger ("Burger"), will serve as a testing platform for the Flow. The Flow will be attached to the Burger via an adjustable bracket and the system will be driven around under ideal conditions – standard indoor lighting and highly reflective surfaces, for

example – to limit sources of error when tuning sensor parameters. The sensor's odometry readings will then be compared to readings from the Burger's built-in motor encoders, and, if sufficiently similar, will undergo further testing. Otherwise, adjustments to the sensor's firmware or position on the Burger will be made prior to additional tests.

In a second test, the Flow will be mounted to the rover using an adjustable bracket. Under predetermined test cases, it will be determined how fast the rover can accelerate, cruise, and turn while the Flow maintains accurate odometry. To measure the odometry accuracy, the rover will drive from one predetermined point to another, and the rover's calculated distance travelled will have to be within 5% error of the true distance travelled to pass the test. To ensure that odometry is reliable, the rover will be driven on surfaces of various roughness and reflectivity as well as in different lighting conditions. Through this testing, situations in which visual odometry is feasible can be determined.

Returning to Base

The ability for the rover to return to a base will be tested under a pass/fail criterion. If the rover returns to within 1 meter of the base while avoiding obstacles, it will pass; otherwise, it will fail. The return-to-base algorithm's reliability will be measured as the percentage of trials in which the rover passed.

Future Work

The final system will still have room for improvement, with a recommended primary focus of future research being to increase the robustness of obstacle avoidance and SLAM algorithms with regards to indoor environmental conditions. Further research should focus on enabling the obstacle avoidance algorithm to account for obstacles that are small, narrow, opaque, translucent, transparent, or moving, as this would help increase the system's reliability. Regarding SLAM algorithms, mapping robustness could be improved by utilizing more than one visual odometry sensor.

As the current obstacle avoidance algorithm cannot detect cliffs or staircases, the rover is unable to avoid falling off of edges, which can be resolved with future development. In addition, a current system limitation is that the 2D LiDAR only allows mapping objects at the specific height above the ground at which it is placed on the rover. This can give the impression that a space has either fewer or more obstacles than it actually does because they are located above or below the LiDAR's height. Using a 3D LiDAR capable of mapping an entire room would help remove or reduce disparities between what the rover detects as an obstacle and what a human would perceive as an obstacle in the same space. Although the operator has video feed of the environment, in high-latency situations, the rover can become unreachable. So, further increasing the system's autonomous capabilities would ensure more reliable and efficient operation while also enabling a lower strain on the human operator.

Conclusion

Implementing and adopting autonomous features in a remote-controlled UGV has the potential to improve operational performance and system responsiveness. The implemented obstacle avoidance algorithm enabled the rover's safe and effective operation under the scenarios tested, and greatly reduced the chances of the rover crashing into obstacles. Integrating SLAM algorithms to the system enabled the rover to autonomously conduct map-making and localization in indoor spaces where GPS signal is unreliable.

Although SLAM without odometry performed well in certain scenarios, a reliable source of odometry is needed for robust map-making implementation, and it is recommended to use an odometry-based algorithm for ISR applications. Combining obstacle avoidance and SLAM algorithms enabled the rover to achieve its third and last objective to return to an indoor base while building a map and avoiding obstacles. However, since this solution has yet to be tested, the system has potential for improvement. Overall, this project was successful in mitigating operational risks and enhancing rover performance in adversarial environments.

Acknowledgment

The team would like to thank Michael Balazs, Nathan Gaul, Andy Chapman, and the rest of the MITRE Corporation for their valuable contributions to this project through their mentorship and expertise.

References

[1] Office of the Secretary of Defense. 2005. "Unmanned Aircraft Systems Roadmap, 2005-2030." apps.dtic.mil/docs/citations/ADA445081

[2] Gilbert, Gary R. and Beebe, Michael K. January 2010. "United States Department of Defense Research in Robotic Unmanned Systems for Combat Casualty Care." apps.dtic.mil/dtic/tr/fulltext/u2/a526596.pdf. Accessed: March 19, 2019.

[3] Schaefer, Kristin E., Straub, Edward R., Chen, Jessie Y.C., et. al. December 2017. "Communicating Intent to Develop Shared Situation Awareness and Engender Trust in Human-Agent Teams." Cognitive Systems Research 46, pp.26-39. doi.org/10.1016/j.cogsys.2017.02.002

[4] MIT CSAIL Distributed Robotics Laboratory. "Robo-Rats Locomation: Odometry." 2001. groups.csail.mit.edu/drl/courses/cs54-2001s/odometry.html. Accessed: March 16, 2019.

[5] McGarey, Patrick. February 2016. "Visual Odometry." http://www.cs.toronto.edu/~urtasun/courses/CSC2541/03_odometry.pdf. Accessed: March 16, 2019.

Author Information

Utkarsha Bhave, Student, Department of Engineering Systems and Environment, University of Virginia.

Grant D Showalter, Student, Department of Electrical and Computer Engineering, University of Virginia.

Cesar Roucco, Student, Department of Electrical and Computer Engineering, University of Virginia.

Dalton J Anderson, Student, Department of Mechanical and Aerospace Engineering, University of Virginia.

Andrew C Hensley, Student, Department of Engineering Systems and Environment, University of Virginia.

Gregory C Lewin, Assistant Professor, Department of Engineering Systems and Environment, University of Virginia.