Simulating and Training Autonomous Rover Navigation in Unity Engine Using Local Sensor Data

Christopher Pace

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2024

Abstract

Autonomous navigation is essential to remotely operating mobile vehicles on Mars, as communication takes up to 20 minutes to travel between the Earth and Mars. Several autonomous navigation methods have been implemented in Mars rovers and other mobile robots, such as odometry or simultaneous localization and mapping (SLAM) until the past few years when deep reinforcement learning (DRL) emerged as a viable alternative. In this thesis, a simulation model for end-to-end DRL Mars rover autonomous navigation training was created using Unity Engine, using local inputs such as GNSS, LiDAR, and gyro. This model was then trained in navigation in a flat environment using the proximal policy optimization (PPO) algorithm. The results of the training and future work are discussed.

Simulating and Training Autonomous Rover Navigation in Unity Engine Using Local Sensor Data

Introduction

Mobile robots are becoming more important and integrated into society. They are delivering food, shipping packages, and even driving cars. At the pinnacle of mobile robots sit the Mars rovers, with six successful missions from both the U.S. and China to the Red Planet, two of which are still active as of March 2024 (National Aeronautics and Space Administration [NASA], 2023). Each rover is a remarkable feat of engineering. This is exemplified best by the most recent mission, Perseverance, which took nine years to complete from conception to landing on Mars (Wall, 2012). One of the many engineering challenges that contribute to this long development time is creating an optimized system capable of autonomous navigation. Depending on the orbital position, communication may take between 5 and 20 minutes to travel between Earth and Mars, which can lead to a 40-minute delay between an instruction being sent from Earth and the response received from the rover. It becomes abundantly clear that mobile vehicles on Mars need to be able to navigate autonomously to some degree if they hope to drive further than a few feet every 30 minutes.

Successful Mars Rover Missions and Navigation Strategies

Mars Pathfinder: Sojourner

The first successful Mars rover mission, Mars Pathfinder, landed in 1997 with the rover Sojourner on board. This rover used a simple form of autonomous navigation and mapping (Matijevic, 1998). A set of coordinates were sent from Earth defining a waypoint to be reached autonomously. Sojourner then followed a pre-defined algorithm to reach the waypoint by first pointing straight towards the waypoint and driving until an obstacle was detected by its

photographic and laser sensors. The rover would then turn until the obstacle was no longer visible ahead, then drive forward again, periodically turning back towards the waypoint to check if the obstacle had been cleared. Once the obstacle was cleared, Sojourner would continue driving straight towards the waypoint until it either reached the goal or encountered another hazard. It is important to note that this system did not retain any memory of past hazards for future pathfinding.

In addition, Sojourner's mapping was based on a coordinate system concerning its base communication lander, Pathfinder. It estimated its distance traveled through odometry, by simply multiplying the wheel revolutions by the circumference. In a short-range rover, wheel odometry is a fairly consistent method for position estimating, but does not function well over longer distances. As the distance increases, small errors due to wheel slipping, floating-point rounding, and other measurement errors will add up until the assumed position of the rover is completely incorrect.

Mars Exploration Rover: Spirit and Opportunity

NASA's next rover mission, Mars Exploration Rover (MER), used twin rovers, Spirit and Opportunity, which landed on Mars in 2003. (NASA, 2023) These rovers were designed to drive much longer and further than Sojourner had. To accomplish this, the rovers needed to use a different localization and mapping strategy, not based on an estimate for their landers. In addition to wheel odometry, Spirit and Opportunity were equipped with intelligent cameras capable of visual odometry, allowing errors in the wheel odometry system to be corrected in real-time by tracking nearby terrain features (Guan, 2014). Images are taken on the lander's descent to estimate each feature's location with respect to the landing site. After 10 seconds of each rover's navigation, they pause for 20 seconds to image the surrounding terrain and match it to the

landing images. However, this visual odometry is prone to error if the locations of these terrain features are not accurate. This approach also has a range restriction, as path correcting is only possible in the local range where terrain points were identified on landing. While this range is much further than wheel odometry alone could allow, more could be wanted.

Mars Science Laboratory: Curiosity

The Mars Science Laboratory (MSL) mission landed in 2011 and deployed the Curiosity rover, which is still in operation as of March 2024 (National Aeronautics and Space Administration [NASA], 2024a). MSL's goal was to explore Mars in search of previous or existing life, analyzing samples using a variety of onboard instruments. Curiosity is a huge upgrade from the MER mission rovers in terms of size, durability, and, importantly, navigation capability. It can travel up to 90 meters per hour using a combination of long-range navigation cameras, and close-range hazard detection cameras (National Aeronautics and Space Administration [NASA], 2022). Operators on Earth can specify goal waypoints in Curiosity's field of view. It will then navigate to those waypoints autonomously without requiring prior knowledge of the in-between terrain. Curiosity's navigation system was programmed using a custom language developed by NASA engineers specifically for the rover: Plan Execution Interchange Language (PLEXIL) (Estlin et al., 2006). Using PLEXIL, Curiosity was programmed with specific algorithms to navigate around obstacles and reach waypoints. It is important to note that these algorithms were pre-programmed by the engineers to make the rover behave in a pre-conceived fashion.

Mars 2020: Perseverance

The Mars 2020 mission launched in July of 2020, landing on Mars in February of 2021 (NASA, 2023). The mission landed in the Jezero Crater with two vehicles aboard, the

Perseverance rover and the Ingenuity helicopter. Perseverance has a similar task to Curiosity, searching for previously existing life and potential habitability, but is uniquely focused on collecting soil and rock samples for an eventual retrieval mission back to Earth.

In addition to the now standard visual odometry, Perseverance uses an updated navigation protocol, dubbed "AutoNav" by NASA engineers (Verma, 2021). This program can perform visual odometry continuously while in motion better than the rovers before it, allowing it to cover distance more quickly. So far, Perseverance has crossed land at over three times Curiosity's travel rate per sol (Martian day), over 23 m/sol to under 8 m/sol (NASA, 2024a; NASA, 2024b). This may be circumstantial as Perseverance has been on Mars for about one-fourth of the time, but the rate difference is not insignificant.

Recent Approaches to Mobile Robot Autonomous Navigation

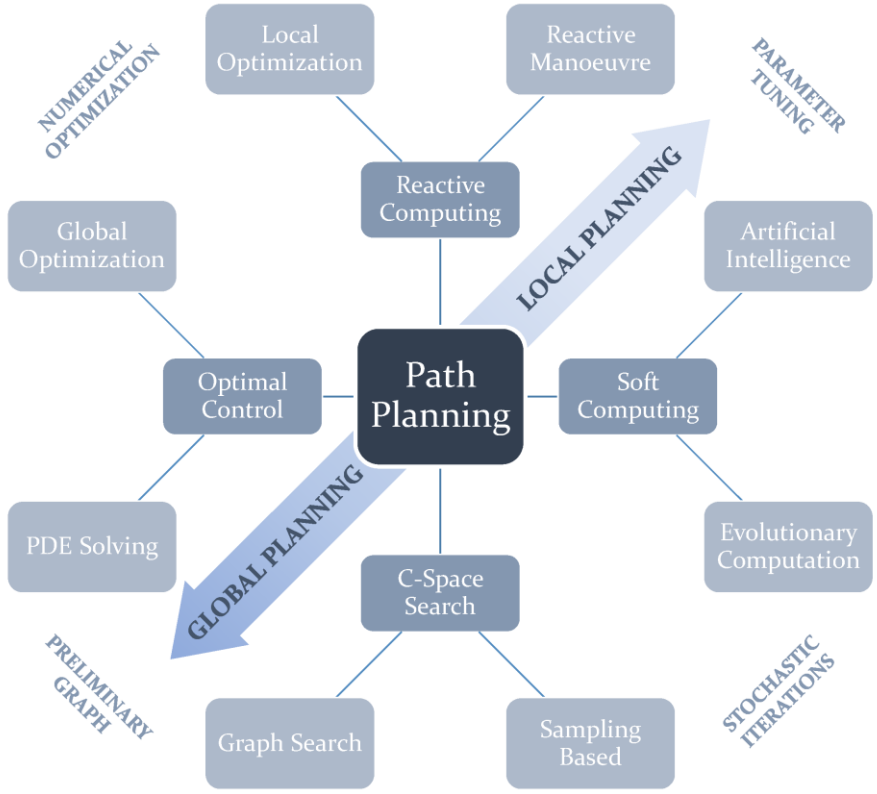
Due to the proprietary nature of the current Mars rover designs, it is difficult to know the precise methods and algorithms used in them unless one is in NASA themselves. Fortunately, many solutions have been proposed for optimal autonomous navigation of mobile robots in the scholarly arena. While a significant amount of the research in the area of autonomous navigation is focused on self-driving vehicles, there is still a host of approaches that can be applied to the improvement of autonomous navigation in the context of a Mars rover.

Global Planning

Most approaches to vehicular autonomous navigation can be split into two large categories: *Global Planning* and *Local Planning* (Sánchez-Ibáñez et al., 2021). See Figure 1. Global Planning as a navigation strategy assumes existing knowledge of the environment in which a vehicle or robot is operating. It often uses grid maps that must be developed beforehand to give the robot knowledge of its surroundings. While it has useful applications on Earth, Global

Figure 1

Contrasting Global Planning vs Local Planning



Note. From "Path Planning for Autonomous Mobile Robots: A Review," by J. R. Sánchez-Ibáñez, C. J. Pérez-del-Pulgar, and A. L. Garcia-Cerezo, 2021, *Sensors, 21*(23), p. 7898 (https://doi.org/10.3390/s21237898)

Planning is extremely difficult to implement for a rover navigating on Mars, as there is very little that can be mapped out before the rover encounters itself. Satellite images from the few Mars orbiters have often helped rover operators decide on a general path on a large scale. This allows them to choose waypoints strategically that will avoid large, obvious obstacles that can be seen from space. However, these images do nothing to assist the rovers in real time to avoid the smaller obstacles: small boulders, drops, and other rough terrains. Devices like the Ingenuity

helicopter which landed with the Perseverance can help to some degree, assisting by scouting the terrain ahead so the rover operators can plan an ideal route quickly before moving the rover (Verma, 2021). While this scouting is helpful, it will not be possible for many years to acquire global knowledge of the Martian terrain for rovers to navigate exclusively using Global Planning.

Local Planning

As a concept, Local Planning uses local sensor input from a vehicle or robot to achieve an amount of knowledge of its local surroundings, then uses that local knowledge to navigate towards its goal. In a very simple example, a small, wheeled robot could have bumper sensors that could detect when it collides with an object and on which side. An algorithm could then be used to drive the robot away from that obstacle and find another path. For use on Mars, Local Planning is a more practical option than Global Planning, as sensors can be easily implemented to give a rover information about its surroundings. All the successful Mars rover missions had some form of Local Planning on board. The laser and photo sensors on Sojourner gave it awareness to discern obstacles when they were directly in front. Visual odometry assisted all future rovers to discern their position among visual landmarks.

Simultaneous Localization and Mapping

Other autonomous vehicles have used light detection and ranging (LiDAR) sensors, sound navigation and ranging (Sonar) sensors, and cameras to allow local detection of their environments (Azar et al., 2023). Where, in Global Planning, a map must be pre-built, using such sensors, a robot can generate a map of its surroundings and place itself within that map. This technique, simultaneous localization and mapping (SLAM), is utilized extensively across the field of autonomous navigation. SLAM is a powerful tool for mapping 2D and 3D environments,

allowing pathfinding algorithms to then find optimal paths. Curiosity itself uses SLAM with its various cameras to map the surrounding terrain. (National Aeronautics and Space Administration [NASA], 2013)

However, as the amount of data increases in navigational maps generated using SLAM, it becomes increasingly difficult for algorithms designed by humans to optimally explore and reach waypoints. This becomes especially apparent when a map has several local minima, bringing the robot close to a target, but with no obvious way to reach it by moving forwards.

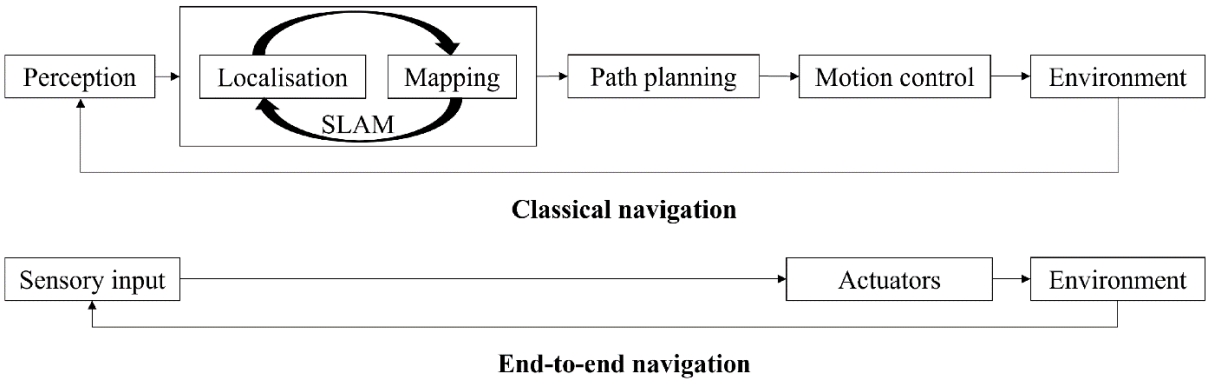
End-to-End Deep Reinforcement Learning

In the above-mentioned cases, it has recently become popular to utilize Machine Learning (ML) to develop optimized solutions to real-world robot navigation instead of using SLAM. Reinforcement learning (RL) is a subset of ML that uses model performance feedback to optimize a system's behavior. Specifically, Deep Reinforcement Learning (DRL), a subclass of ML, is a key tool for robot navigation in unknown environments (Azar et al., 2023; Lee & Yusuf, 2022). DRL utilizes a neural network (NN) to directly map inputs to outputs, or sensors to actuators in the case of robots. The inputs are summed and weighted through layers of the NN to produce the outputs. Most NNs have at least one *hidden layer* between the inputs and outputs. DRL is also characterized by its use of deep neural networks (DNN), which are neural networks with more than one hidden layer. DNNs are capable of more advanced, complex behavior than one-deep NNs.

This approach to autonomous navigation revolutionized the efficiency of robotic navigation systems by skipping the need to create a map and run a path planning algorithm, both of which require a significant portion of time. See Figure 2.

Figure 2

Contrasting Classical vs End-to-End Navigation Approaches



Note. From "Challenges and Solutions for Autonomous Ground Robot Scene Understanding and Navigation in Unstructured Outdoor Environments: A Review," by L. Wijayathunga, A. Rassau, and D. Chai, 2023, *Applied Sciences, 13*(17), p. 9877 (https://doi.org/10.3390/app13179877)

RL begins with randomized weights for each connection of inputs through the layers to the outputs, causing random and sporadic behavior. In RL, it is necessary to set up a reward function, which serves to guide the NN towards an optimal point, which, in the case of Mars rovers, is efficient autonomous navigation. The reward function uses an external measure to give a value of success to each iteration of the NN. In the case of land rovers with a specific waypoint as the goal, a common strategy is to increase the reward as the robot moves towards the goal.

After many randomized NNs have been tested in a system, the RL algorithm chooses the iterations with the highest reward functions to include in further iterations. DRL distinguishes itself from normal RL as it uses deep neural networks (DNN), which are NNs with more than one layer between the inputs and outputs. While this is a seemingly simple change, the performance of DNNs over standard one-deep NNs is significant. This would be a painstaking process to implement on a physical rover, as the time required would be astronomical.

For this reason, the training of DNNs using DRL is performed almost exclusively in computer simulations before being implemented and tuned on physical models. Computers can run multiple physics simulations simultaneously to accomplish remarkable training times.

In addition, DNNS can use inputs from previous time cycles as present inputs. This allows the system to have a kind of memory, which decreases the likelihood of getting stuck in local minima.

Using DRL and a DNN, it is possible to completely bypass the SLAM process requiring creating a map and then deciding on a path. As long as a goal state is defined with an appropriately shaped reward function, a DNN can easily outperform SLAM methods in terms of speed and computational efficiency once trained. While some Artificial Intelligence (AI) and Computer Vision (CV) methods are present in the camera detection systems of Curiosity and Perseverance, they still use SLAM for their regular operation. An end-to-end DNN trained with DRL could have a great effect on future Mars rovers. However, there are multiple challenges in setting up such a system.

Challenges

Sensors

An important aspect of developing such a DNN is selecting proper sensors. Multiple types of sensors are necessary to work together for navigation to be successful. First, a robot needs to have a method of detecting obstacles in the near vicinity. As mentioned above, sonar and LiDAR are fantastic options, both of which are principally used to detect obstacles. LiDAR is more often chosen for robotic applications due to its long range and accuracy compared to sonar.

Cameras are also fantastic options for obstacle detection and localization as they can give much finer details and data to the NN. Their main potential drawback is the capacity and speed to handle the greatly increased inputs of many pixels compared to other simple methods like LiDAR. It is difficult to use single cameras for navigation, as depth cannot be perceived by a single image (Wijayathunga et al., 2023). A single camera can attempt to determine its orientation by referencing previous positions, but small errors persist through this process, making it unfeasible. Fortunately, stereo cameras utilize two adjacent images, allowing for depth and reference to be perceived without drifting error. An additional question can also be asked concerning camera sensors: whether to use RGB or grayscale. In most cases, the answer here is straightforward, as computers do not need the RGB color to distinguish the depth in an image, so grayscale is almost always the lower power, cheaper option.

In addition to detecting local obstacles, rovers must be able to localize themselves within their environment. For many years, visual odometry has been the norm for localizing Mars rovers. However, while not yet implemented, it will be possible in the future to create a GNSS satellite net around Mars to enable localization anywhere on the planet. GNSS coordinate capability would give a rover global knowledge of its position, eliminating the inaccuracies that can arise through visual odometry. While this is still far from full Global Planning, GNSS localizing is a powerful tool. In addition, GNSS would allow for waypoints to be selected in a much broader range than just in the field of view of the rover, which is a current limitation of Curiosity and Perseverance.

While GNSS provides reliable positional information, navigating rough terrain requires the robot to have a sensor to detect pitch and roll (Weerakoon et al., 2022). This functionality is realized by using a 6-axis gyroscope and accelerometer. The gyroscope provides precise 3-axis

angular velocity values, while the accelerometer reads force on the sensor, including gravity. To obtain the rover's pitch and roll from this information, the gyroscope integrates the angular velocity values over time to obtain a precise value for angular position. However, due to the summing that occurs by integrating the angular velocity, floating point rounding errors stack up in the calculation, causing the final position value to drift slowly over time. This can be corrected by adding a complementary filter using the accelerometer readings as a consistent gravity reference. The resulting pitch and roll values retain the precision of the gyroscope with the consistency of the accelerometer.

Environment

When training a DNN using DRL it is important to create a training environment that will closely resemble the real environment in which the physical model will operate. Lee and Yusuf (2022) designed an indoor environment with a flat surface. In addition, they included both fixed and dynamically moving obstacles for the robot to navigate around. They combined simulated LiDAR and RGB cameras to create a true end-to-end DNN solution. However, Mars rovers do not operate indoors on flat surfaces, so this research is less applicable to the problem of Martian autonomous navigation.

Weerakoon et al. (2022) developed a system for navigation of uneven outdoor environments, Terrain Elevation-based Reliable Path Planning (TERP). This system utilizes a DRL network to identify terrain regions of low stability that would cause the vehicle to tip. This 3D terrain map is then cast to two dimensions at which point a mapping algorithm finds the optimal path to avoid instability. While TERP was successfully used to optimize uneven terrain navigation, it is not an end-to-end DNN solution. TERP separates the mapping and pathfinding algorithms, a step that is not necessary for an end-to-end DNN.

Software

When developing a DNN-controlled robot simulation, a defining characteristic is the software used for simulation. Many concepts have been developed and used over the past several years. Lee and Yusuf (2022) utilized the popular Robot Operating System (ROS) as their main framework for controlling the simulation. The simulated environment was created in the Gazebo training environment, popular for robot and physics simulations.

Research Question

From the above research, I decided to create a simulation for rover autonomous navigation using DRL and train it to navigate autonomously over increasingly rough terrain.

First, I considered the challenges listed above. For obstacle detection, I realized that sonar could be quite difficult to simulate accurately from scratch. Given the opposing simplicity of simulating LiDAR, which is as simple as drawing a line, and the overall advantages of LiDAR practically, I decided that LiDAR is easily the best choice for my project.

As the computing power I had available to me was not very large, I decided against simulating mono or stereo cameras, as they would slow the simulation and training progress significantly.

The final sensors necessary for localization were easy to select. GNSS can be easily simulated using simple X-Y coordinates, making it an easy choice for position localization. In addition, tuned gyroscope/accelerometer readings could be simply added to a simulation by accessing the angular position of the object. In addition, differential GNSS is a real-world application to access yaw, as it is not possible to obtain reliably using an accelerometer.

As all the above-chosen sensors should be simple to incorporate in a simulation and are all standardly used in similar autonomous vehicle applications as outlined previously, I was

confident in my choices. The inputs to the rover DNN would be as follows: 32 LiDAR inputs in a 2D disc around the rover, GNSS coordinates of the rover, GNSS coordinates of the goal, as well as the rover's pitch, roll, and yaw.

Next, I considered what type of environments to simulate. As cameras were not going to be utilized, the photorealism of the environments was irrelevant. However, the shaping of the terrain would be a vital component of the simulation, as that would be the primary interaction with the LiDAR. I decided it would be best to plan for a progression of difficulty. First, I would create a simple environment with a flat floor, walls, and the goal close in front of the rover. Once the rover learned to navigate forward toward the goal, I would create another scenario which would randomize the position of the rover and the goal within the environment at each reset. Then the rover would have to learn to navigate with respect to its position difference and angle from the goal. Once that was achieved, the terrain could be roughened and made more extreme in increments as the rover model became more accustomed to navigating the terrain.

The final challenge was to decide on what program to use to simulate the rover environment and training. Many ML programs utilize Linux applications heavily. Unfortunately, I have very little Linux experience, though I do have some experience coding with C-based languages. The Unity Engine is a powerful 3D simulator used often in video game development. However, it is also often used in architectural, automotive, manufacturing, and many other contexts to provide a testing ground for new ideas. It is also primarily coded in C#, making it more friendly than other alternatives. There is an open-source package Unity package, ml-agents, which can connect Unity assets to powerful TensorFlow ML algorithms such as Proximal Policy Optimization (PPO). From the research I performed, I found no study that attempted the autonomous navigation of a robotic vehicle using the Unity Engine.

I decided to create a simulation for autonomous Mars rover navigation with onboard GNSS, lidar, gyroscope, and accelerometer using the Unity 3D engine and optimize its capabilities using the PPO algorithm in the Unity ml-agents AI package for navigation across varied terrain and obstacles using only local sensor data.

Development of Autonomous Navigation Simulation

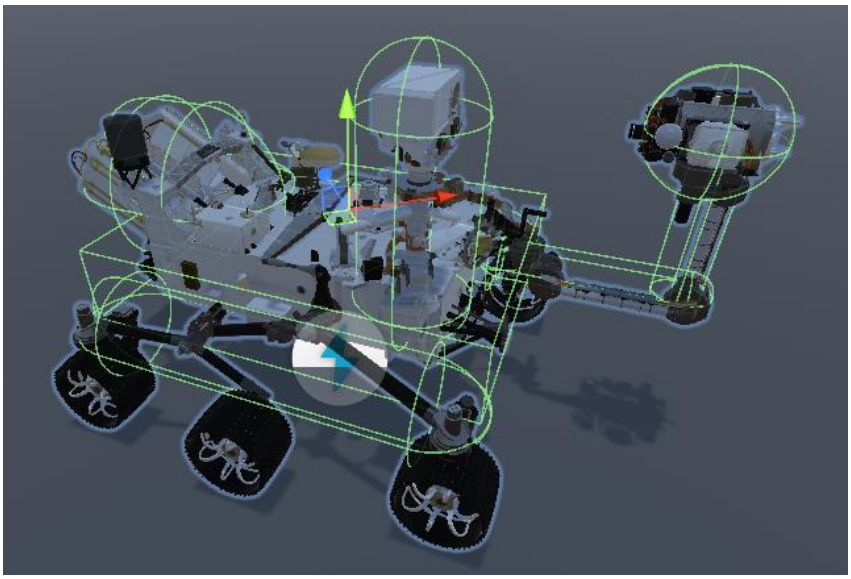
Creating the Simulation Environment

I began by creating the initial virtual environment in Unity Engine. This included establishing the rover's kinematics and physics. I created a simulation of a Mars rover with three main characteristics: visual model, hitbox, and wheel physics. For the visual model, I imported an open-source 3D model of the Perseverance rover from NASA's website. The visual model has no physical interaction with the environment and exists purely as a visual aid to the simulation. The hitbox is a collection of 3D collideable shapes that follow the shape of the rover. They provide collision for the entire rover with the environment. See the visual model and outlined hitbox from Unity in Figure 3.

The wheel physics was accomplished by establishing a Unity Transform (relative position and orientation) for six "virtual wheels" in the exact positions of the visual model's wheels. The current design accounts for a fixed-wheel, skid-steer drive. Practically, this means that none of the wheels can turn about its vertical axis and steering is performed by applying forward power on either the right or left wheels and backward power on the opposite side. Each wheel's physics is divided into three vectors: suspension along the vertical (y) axis, drive/brake along the forward (z) axis, and grip along the left/right (x) axis. The forces along each of these vectors are calculated separately at each physics update interval during runtime. The suspension force is calculated by emulating spring physics with offset and damping with strength variables

Figure 3

Visual Model and Hitbox in Unity



for both. The grip force is calculated by counteracting horizontal (x-axis) velocity. The grip force is stronger at lower x-axis velocities, simulating full grip. Once the grip is broken at higher x-axis velocities, the wheel is allowed to slide further to each side. The driving force is calculated based on an input value from -255 to +255. This input value is passed through a magnitude curve which caps the input force value at high speeds, simulating a motor force curve. The suspension, grip, and drive vectors are then summed and applied to each wheel at each physics increment of the simulation.

The physics parameters have been tuned to give a reasonable estimate of speeds, grip, and gravity that would be encountered on Mars. These parameters could be further perfected in future research, though the focus of this research is on the navigation capabilities of the developed AI, not the perfect accuracy of the physics. In a real-world application, the AI would be trained hundreds if not thousands of times in the simulation before being deployed on a physical rover to interact and fine-tune the AI model in the real world. As mentioned previously,

the power of the software simulation is its ability to run training orders of magnitude faster than training on hardware.

I also prepared a basic training environment for the rover. It consists of a simple flat square with short walls. See Figure 4.

Preparing Machine Learning

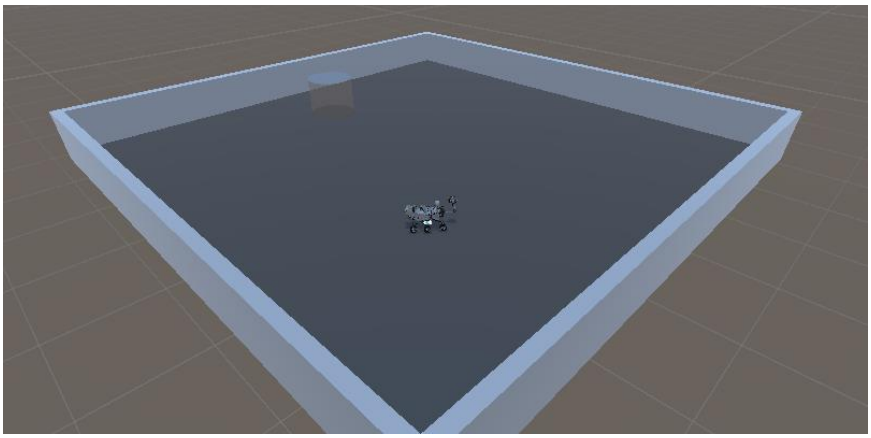
Coding with Unity's ml-agents package begins by defining an *agent* which, in this case, is the rover. I set up the inputs and outputs of the DNN by linking the inputs to the rover's position (GNSS), rotation (gyroscope/accelerometer), and the goal's position (GNSS) and the outputs to the forces acting on each of the 6 wheels. I started the simulation without implementing the LiDAR as I wanted the rover to initially simply learn how to operate using GNSS.

Each loop of the DRL program takes place as an *episode*. For my first arrangement, at the beginning of each episode, the rover is returned to its neutral position facing towards the goal. Once the episode starts, the rover will begin to act by *inference* to the DNN values it is receiving until it reaches one of the following four end conditions: It reaches the goal, hits a wall, flips over, or runs out of time. If any one of these conditions is met, the Episode will end and ml-agents' PPO algorithm will calculate the reward function for the episode. Based on the scores over several episodes, PPO will choose the most successful episode NN weights, mutate them slightly, and then reuse them again.

The most important aspect of this training was in the crafting of the reward function. In all iterations, I included a constant positive reward for reaching the goal and a negative reward for hitting walls or flipping. In addition, I experimented with several reward function components to see what would work best to train the rover to navigate properly. My most

Figure 4

Basic Training Environment in Unity



successful functions used a form of the dot product to give increasing rewards the closer the rover was pointed towards the goal and for moving towards the goal, which will be detailed below.

To speed up training, I duplicated the training environment to enable up to 100 agents to train at a time, each individually contributing to the training. This would allow the training time to only be limited by the graphics render speed of my computer.

Model Training and Results

First Session: Rover1

In the first NN training session, labeled *rover1,* the rover was set pointing straight at the goal at the start of each episode. In addition, the LiDAR functionality had not been added to the simulation, so the rover's NN inputs consisted of the rover's x-z position (GNSS), the rover's orientation (Accelerometer/Gyroscope), and the goal's x-z position (GNSS), with the outputs being the force, negative or positive, applied to each of the six wheels. In the neural network, I included 3 hidden layers on 120 nodes each, which should be more than enough to accommodate

the number of inputs and outputs and the complex behavior between them. This approach was used to initially test the functionality of the physics, training program, and reward function working together. As the initial positions of the rover and the goal were not randomized, the training would end up only training the rover to drive *forward*, not specifically *towards the goal*.

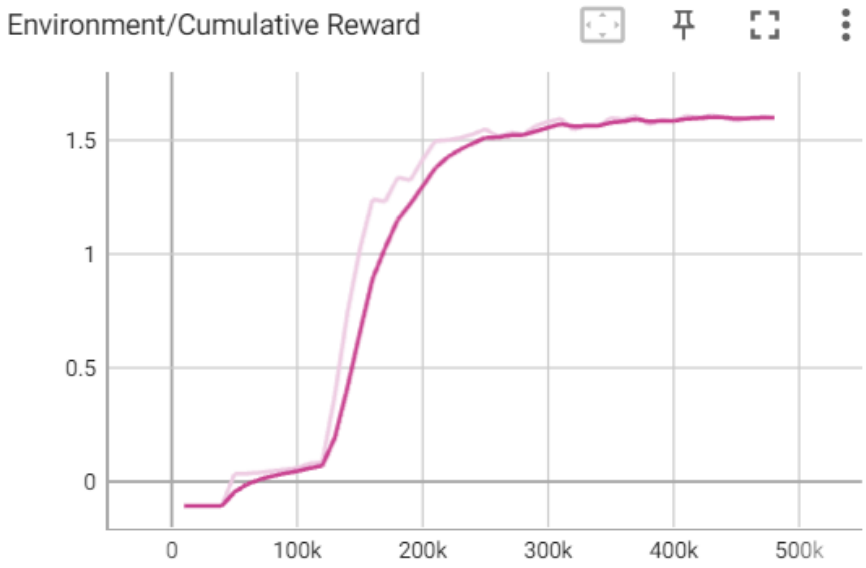
For rover1, I created a reward function including four parameters: finish, time, angle, and velocity. The finish parameters include a large positive reward for reaching the goal and large negative rewards for either hitting the barriers or flipping the rover over. The time parameter gives a small negative reward for each step in which the rover has not reached the goal. Its purpose is to encourage the model to achieve a time-efficient solution. To train the rover to point at the goal, the angle parameter adds a *set* positive reward whenever the rover is pointed within 10^{o} of the goal. Finally, the velocity parameter gave a positive reward proportional to the magnitude of the rover's velocity, regardless of the direction.

The ml-agents package runs the training in Python and records the agent's average reward values at certain update intervals throughout training. The average *Cumulative Reward*, or total reward earned in an individual episode, is graphed versus the *Step Count*, or number of total physics frames processed by all the agents. See Figure 5. The standard physics framerate is 50 fps, but ml-agents boost that rate to function as fast as the allocated memory, graphics, and computing power will allow. Most training sessions use between 500 thousand and 20 million steps.

The rover quickly learned to point at the goal within the first 100,000 steps as can be seen in the slight slope upwards in that range. Shortly thereafter, the model learned to drive forward, gaining both the velocity award and positive goal award when it was pointed properly. This resulted in a drastic increase in the cumulative reward, leveling out at around 250,000 steps. This

Figure 5

Rover1 Simulation Cumulative Reward Over Time



the training was very short compared to future training as noted above, the rover simply needed to drive forward each time as opposed to learning to drive toward a randomized goal.

Second Sessions: Rover2 & Rover3

To simulate a more realistic situation, I updated the environment to place the rover and goal at randomized locations within the environment at the beginning of each training episode. The instantaneous position of both the goal and the rover is still given as NN inputs, requiring the agent to learn to drive using the two in the context of each other. I used the same reward function from rover1, increasing the reward both for driving forward and for pointing towards the goal.

I started rover2 from scratch using a new randomized NN. This training ran for 500,000 steps and experienced little progress. Figure 6 shows the distribution of the cumulative reward over time.

Figure 6

Rover2 Simulation Cumulative Reward Over Time Histogram

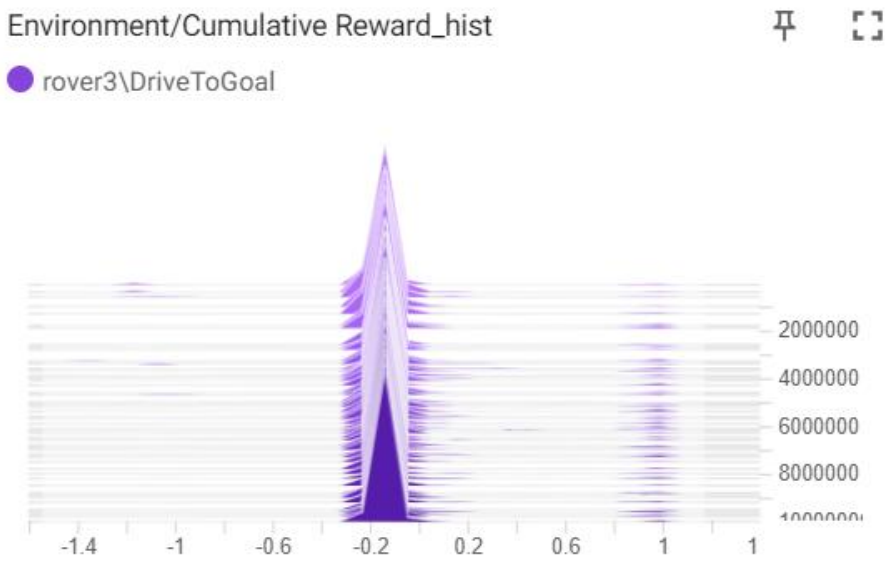
The above histogram displays three clear categories of rewards in the ranges -1.2, -0.2, and +1.0. These rewards can be easily correlated with situations in which the rover hits a wall (-1 reward), reaches the goal (+1 reward), or fails to do either within the maximum step count, in which case the time factor subtracts a small reward over time, resulting in the slightly negative reward. As made clear from the histogram, no real progress was made.

The next session, rover3, initiated from the results and reward function parameters of rover2 and ran for an additional 10 million steps. This test was completed to determine if the model needed time to find the path to a higher reward. Unfortunately, this test also gained very little ground as can be seen in Figure 7.

This simulation had one very minor improvement over rover2 as it was able to filter out hitting the walls, removing the -1 reward in general. While this brought the average reward up slightly, no progress was made towards consistently reaching the goal.

Figure 7

Rover3 Simulation Cumulative Reward Over Time Histogram



Fourth Session: Roverl1

After the sub-par results of the initial training sessions, I overhauled the reward function and added the LiDAR input functionality to the rover simulation. To simulate LiDAR, I used Unity's raycast function, which measures line of sight and distance along a vector from an origin. I created an array of 32 raycasts, equally distributed pointing out around the vertical axis of the rover, with each returning the distance from the rover to the nearest wall in their respective direction, up to about 30 meters. This is very similar to the capabilities of most LiDAR, though they often have even tighter sensor density. Each of the raycast distances was specified as inputs to the NN.

To fine-tune the reward function, I updated the angle parameter reward to give a proportional scaling reward for pointing towards the goal using the dot product.

$$angleReward = (posToGoal \cdot roverForward) * angleRewardFactor \quad (1)$$

PosToGoal is the vector from the rover's position to the goal. RoverForward is the unit vector pointed straight forward from the rover body. I controlled the magnitude of the reward by altering the angleRewardFactor. As the dot product scales relative to the angle, with the maximum value given when the two vectors are parallel, this would allow the agent to learn more easily without needing to find the specific binary reward for pointing within 10 degrees. In addition, the dot product will have a negative value when the two vectors are pointed in opposite directions, which should discourage the rover from pointing away from the goal.

In addition, I updated the velocity parameter to include the dot product functionality, using the rover's forward velocity as the input parameter this time.

$$speedReward = (posToGoal \cdot roverForwardVelocity) * speedRewardFactor \quad (2)$$

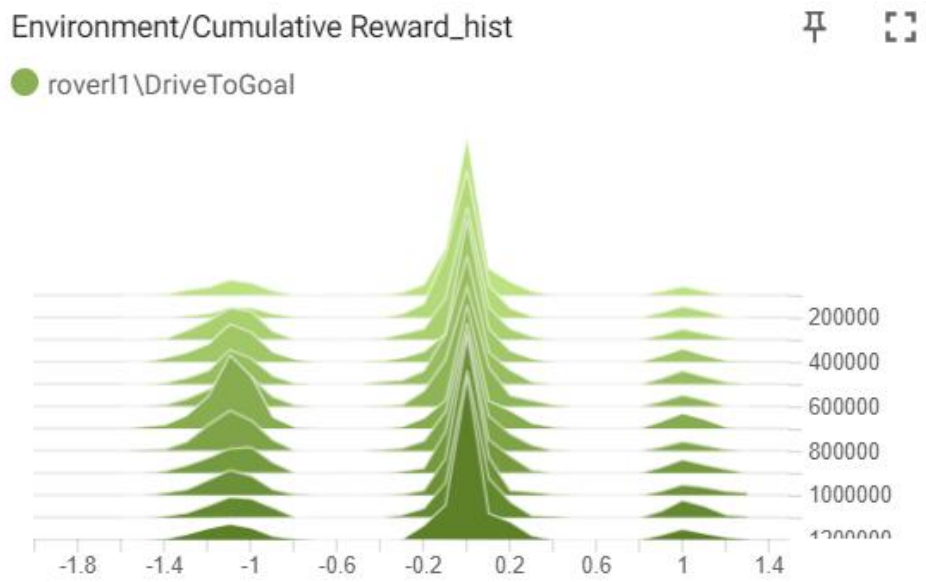
This function encourages the rover to drive with a velocity towards the goal, but also drive quickly, as the dot product scales the angle result by the magnitudes of both vectors.

Incorporating these updated angle and velocity parameters into the existing time and goal parameters, I ran the training again for 1.2 million steps, yielding disappointing results similar to rover2. See Figure 8.

Note the three separate reward distributions corresponding to hitting the wall, reaching the goal, or neither.

Figure 8

Roverl1 Simulation Cumulative Reward Over Time Histogram



Fifth Session: Roverl2

From this point, I decided to shift to a different training strategy, teaching the rover agent to make specific sequential actions, first pointing at the goal, and then driving towards it. To accomplish this, for roverl2, I multiplied the angle reward parameter by 10. This would focus on the

rover pointing at the goal. I ran this training for 10 million steps, with promising results. As seen in Figure 9, the cumulative reward increased steadily over time, reaching its plateau at about 6 million steps.

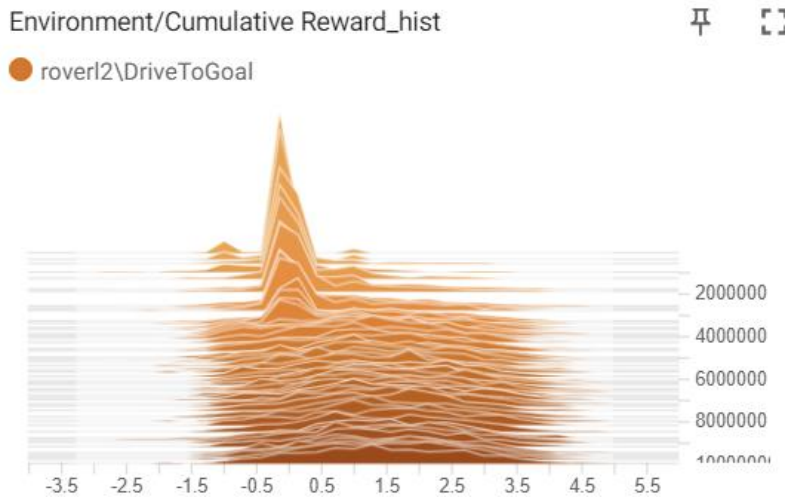
However, the histogram data tells a slightly different story. See Figure 10. The reward became widely distributed at its peak value, likely due to the smooth, sinusoidal shape of the dot product reward function. Observing the rover's behavior after training, it was fairly consistent at pointing within 15 degrees of the goal.

Figure 9

Roverl2 Simulation Cumulative Reward Over Time

Figure 10

Roverl2 Simulation Cumulative Reward Over Time Histogram



Sixth-Tenth Sessions: Roverl3-Roverl7

The next training session, roverl3, was initialized from roverl2 and focused on training the rover agent to move toward the goal. I reduced the pointing reward, increased the velocity reward, and added a new speed parameter, which gives a small reward proportional to the

magnitude of the rover's velocity, regardless of the direction. These combined with the previous angle training in roverl2 should help teach the rover to first turn towards the goal, then drive forward. With these parameters, I ran training for 5 million steps. See the histogram results in Figure 11.

At the end of training, the rover agent began to creep forward, but was by no means near its top speed potential. After this session, I experimented with various weights for the reward function parameters for each training session, totaling over 46 million steps. After the final session, roverl7, the rover moved consistently forwards, though not always towards the goal. See Figure 12. Upon inspection, the model appeared to focus more on avoiding the walls and their large negative reward than on reaching the goal for its large positive reward. The behavior of the rover consisted mostly of pointing away from the wall and driving in loops around the environment. It only occasionally locked onto and reached the goal, almost exclusively when the goal was positioned near the center. Whenever the goal was placed near the walls, the rover rarely reached it, turning to the side to avoid hitting the walls.

Results

I started several other training attempts using a few other parameters and approaches, but none of them converged to a better solution than roverl7 within a similar timeframe. Each of the above training sessions took hours, sometimes days, to complete, as I used my personal laptop computer to run these simulations, which is not extremely fast or efficient. Most successful AI training is completed using a much more powerful computer or even multiple working together. While I intended to expand the rover training to include simulations on rugged terrain, developing and troubleshooting the simulation environment, physics, and training took more time than anticipated.

Figure 11

Roverl3 Simulation Cumulative Reward Over Time Histogram

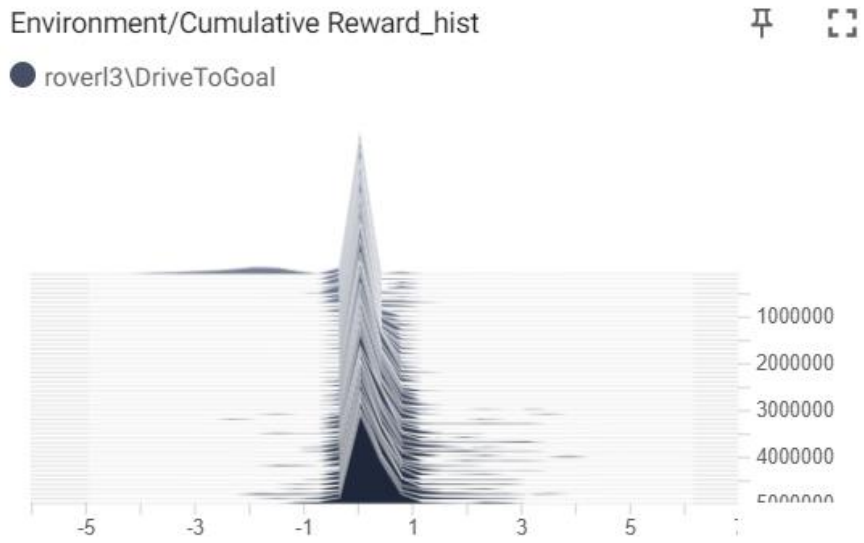
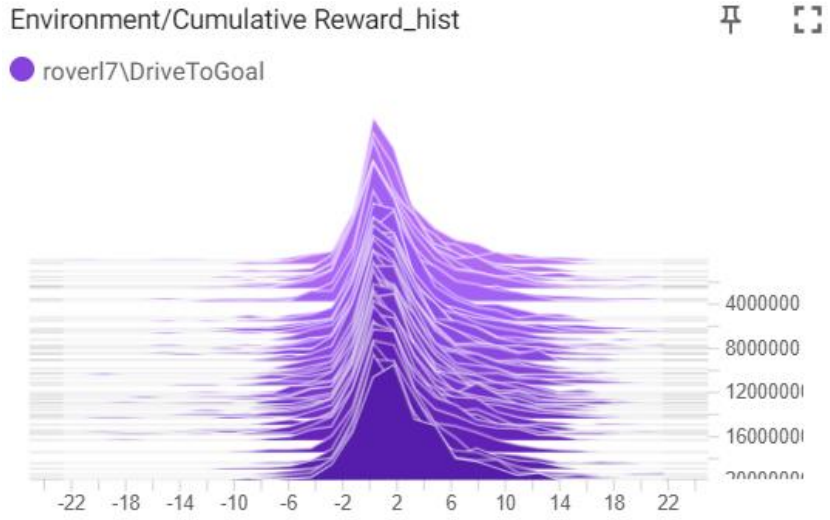


Figure 12

Roverl7 Simulation Cumulative Reward Over Time Histogram



Regardless, I successfully created a simulation environment for training Mars rover autonomous navigation and trained a DRL NN to navigate within it.

Future Work

Multiple aspects of this project could be improved in future work. An immediate improvement could be made by running the training on a more powerful computer or server with more RAM and graphics memory. This would allow for more environments to run training simultaneously, making longer training sessions more efficient and feasible.

In addition, the environment could be expanded to a larger size. This would allow the rover to have more space to explore, potentially avoiding the outcome of the rover avoiding the walls instead of seeking the goal.

An additional reward factor could also be added to the reward function based on the position of the rover. My training focused more on the velocity and angle of the rover, but including a reward for the rover being closer to the goal also has potential.

References

Azar, A.T., Sardar, M.Z., Ahmed, S., Hassanien, A.E., Kamal, N.A. (2023). Autonomous Robot Navigation and Exploration Using Deep Reinforcement Learning with Gazebo and ROS. In: A. Hassanien, R.Y. Rizk, D. Pamucar, A. Darwish, & K. C. Chang, (Eds.), *AISI 2023: Proceedings of the 9th International Conference on Advanced Intelligent Systems and Informatics 2023* (pp. 287-299). Springer, Cham. https://doi.org/10.1007/978-3-031-43247-7_26

Biesiadecki, J. J., Leger, P. C., & Maimone, M. W. (2007). Tradeoffs Between Directed and Autonomous Driving on the Mars Exploration Rovers. *The International Journal of Robotics Research, 26*(1), 91-104. https://doi.org/10.1177/0278364907073777

Estlin, T., Jonsson, A., Pasareanu, C., Simmons, R., Tso, K., & Verma, V. (2006, April 1). *Plan Execution Interchange Language (PLEXIL)*. NTRS - NASA Technical Reports Server. https://ntrs.nasa.gov/citations/20060019246

Guan, X., Wang, X., Fang, J., & Feng, S. (2014). An innovative high accuracy autonomous navigation method for the Mars rovers. *Acta Astronautica, 104*(1), 266-275. https://doi.org/10.1016/j.actaastro.2014.08.001

Hewitt, R. A., Ellery, A., & de Ruiter, A. (2017). Training a terrain traversability classifier for a planetary rover through simulation. *International Journal of Advanced Robotic Systems, 14*(5). https://doi.org/10.1177/1729881417735401

Iagnemma, K., Rzepniewski, A., Dubowsky, S., & Schenker, P. (2003). Control of Robotic Vehicles with Actively Articulated Suspensions in Rough Terrain. *Autonomous Robots, 14*(1), 5-16. https://doi.org/10.1023/A:1020962718637

Kato, Y. & Morioka, K. (2019). Autonomous Robot Navigation System Without Grid Maps Based on Double Deep Q-Network and RTK-GNSS Localization in Outdoor Environments. *2019 IEEE/SICE International Symposium on System Integration (SII)* (pp. 346-351). IEEE. https://doi.org/10.1109/SII.2019.8700426

Lee, M. R. & Yusuf, S. H. (2022). Mobile Robot Navigation Using Deep Reinforcement Learning. *Processes, 10*(12), 2748. https://doi.org/10.3390/pr10122748

Matijevic, J. (1998, February 1). *Autonomous Navigation and the Sojourner Microrover*. Jet Propulsion Laboratory. https://hdl.handle.net/2014/19052

Maurette, M. (2003). Mars Rover Autonomous Navigation. *Autonomous Robots, 14*(2-3), 199-208. https://doi.org/10.1023/A:1022283719900

National Aeronautics and Space Administration. (2013, August 27). *NASA'S Mars Curiosity Debuts Autonomous Navigation* (G. Webster, Ed.). https://www.jpl.nasa.gov/news/nasas-mars-curiosity-debuts-autonomous-navigation

National Aeronautics and Space Administration. (2021, October 29). *Driving Farther and Faster With Autonomous Navigation and Helicopter Scouting* (V. Verma, Ed.). https://mars.nasa.gov/mars2020/mission/status/342/driving-farther-and-faster-with-autonomous-navigation-and-helicopter-scouting/

National Aeronautics and Space Administration. (2022, October 28). *Mars Science Laboratory (MSL)* (D. Williams, Ed.). https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=2011-070A

National Aeronautics and Space Administration. (2023, May 31). *Chronology of Mars Exploration* (D. Williams, Ed.). https://nssdc.gsfc.nasa.gov/planetary/chronology_mars.html

National Aeronautics and Space Administration. (2024a, March 4). *Where is Curiosity?* (J. Platt, Ed.). https://mars.nasa.gov/msl/mission/where-is-the-rover/

National Aeronautics and Space Administration. (2024b, March 4). *Where is Perseverance?* (J. Platt, Ed.). https://mars.nasa.gov/mars2020/mission/where-is-the-rover/

Nguyen, A., Nguyen, N., Tran, K., Tjiputra, E., & Tran, Q. D. (2020). Autonomous Navigation in Complex Environments with Deep Multimodal Fusion Network. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 5824-5830). IEEE. https://doi.org/10.1109/IROS45743.2020.9341494

Ramkumar, M. P., Suresh, P., Siddharth, S., Zair, H., Mohd, A., & Anandakumar, H. (2022). Autonomous navigation system based on a dynamic access control architecture for the internet of vehicles. *Computers and Electrical Engineering, 101,* 108037. https://doi.org/10.1016/j.compeleceng.2022.108037

Sánchez-Ibáñez, J. R., Pérez-del-Pulgar, C. J., & Garcia-Cerezo, A. (2021). Path Planning for Autonomous Mobile Robots: A Review. *Sensors, 21*(23), 7898. https://doi.org/10.3390/s21237898

Swan, R. M., Atha, D., Leopold, H. A., Gildner, M., Oij, S., Chiu, C., & Ono, M. (2021). AI4MARS: A Dataset for Terrain-Aware Autonomous Driving on Mars. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (pp. 1982-1991). IEEE. https://doi.org/10.1109/CVPRW53098.2021.00226

Wall, M. (2012, December 4). *NASA to launch new Mars Rover in 2020*. Space.com. https://www.space.com/18763-nasa-new-mars-rover-2020.html

Weerakoon, K., Sathyamoorthy, A. J., Patel, U., & Manocha, D. (2022). TERP: Reliable Planning in Uneven Outdoor Environments using Deep Reinforcement Learning. *2022 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 9447-9453). IEEE. https://doi.org/10.1109/ICRA46639.2022.9812238

Wijayathunga, L., Rassau, A., & Chai, D. (2023). Challenges and Solutions for Autonomous Ground Robot Scene Understanding and Navigation in Unstructured Outdoor Environments: A Review. *Applied Sciences, 13*(17), 9877. https://doi.org/10.3390/app13179877s