Development of Fully Autonomous and Cooperative Robotic System for Interplanetary Explorations

Undergraduate Honor Thesis

Presented in Partial Fulfillment of the Requirements for the Graduation with Honor Research Distinction in the Mechanical and Aerospace Engineering (MAE) Department of The Ohio State University

By

Kai Chuen Tan

Undergraduate Program in Mechanical Engineering

The Ohio State University

2019

Thesis Committee

Dr. Ran Dai, Advisor

Dr. Carlos Castro, Committee Member

Abstract

The next frontier of interplanetary exploration missions would encounter countless unpredictable geographical challenges including uninhabitable caves, icy craters of the Moon and Mars, unsustainable mountain cliffs, high radiation areas, and extreme temperature environments. This research will design a fully autonomous and cooperative robotics team composed of unmanned ground vehicles (UGVs) with hybrid operational modes to tackle the multiple traveling salesman problem (mTSP) and to overcome environmental obstacles, to accomplish the challenging interplanetary exploration missions. The hybrid operational modes allow every UGV in the team to not only travel on a ground surface but also jump over obstacles, and these UGVs were named jumping rovers. The jumping capability provides a flexible form of locomotion by leaping and landing on top of obstacles instead of navigating around obstacles. Through the cooperation of heterogeneous robots, the goal is to explore unknown areas subject to extreme environmental conditions. To solve the mTSP, an optimal path between any two objective points in an mTSP is determined by the optimized rapidly-exploring random tree method, named RRT*, and is further improved through a refined RRT* algorithm to find a smoother path between targets. Then, the mTSP is formulated as a mixed-integer linear programming (MILP) problem to search for the most cost-effective combination of paths for multiple UGVs that can allocate tasks like visiting target points. The effectiveness of the hybrid operational modes and optimized motion with assigned tasks is verified in an indoor, physical experimental environment using customized jumping rovers.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor, Dr. Ran Dai for the continuous support of my Honors Undergraduate Research Program, for her patience, motivation, and immense knowledge of robotics, automation, and optimization. Her guidance helped me in all the time of research and writing of this thesis. I could not have imaged having a better advisor and mentor for my Honors Undergraduate Research Program. Because of Dr. Dai, I managed to have my first conference paper submitted to the 2020 International Conference on Robotics and Automation (ICRA) as a first author. Without her strong support, it would not be possible to conduct this research.

My sincere thanks also go to Dr. Carlos Castro, who provided me insightful comments and encouragement on being an outstanding presenter as a Buckeye engineering student to prepare me for my Oral Defense. Furthermore, Dr. Castro always willing to listen patiently and to provide advice when I faced unpredictable challenges on my research like a closed labmate of mine quitted the research team out of a sudden.

Besides my advisor and Dr. Castro, I would like to thank Raymond Brooks, who is my laboratory supervisor at Department of Engineering Education, and Ali Rahimiardestani, who is my lab partner for giving me an access to use laser-cutting machine and 3D-printers to manufacture my research jumping rovers for free, teaching me the proper methods of operating the laser-cutting machine and 3D-printers and teaching me efficient techniques of soldering my rovers' electronic components. Without my lab supervisor and lab partner guidance and support, it would take me more time and cost me more money to build a team of jumping rovers.

I thank my fellow Automation and Optimization lab mates, MyungJin Jung, Changhuang Wan, and Isaac Shyu in for the stimulating discussion, for the sleepless night we were working together before deadlines, and for all the fun we had in the last four semesters. Because of their advice, suggestions, encouragement, and constructive criticism, I learned several ingenious heuristic techniques to solve and optimize a path planning problem with MATLAB and run an indoor simulation efficiently with robots and the VICON system used to track the position of objects.

Last but not least, I would like to thank my family: my parents, grandparents, my only younger brother and my relatives for supporting me spiritually throughout writing this thesis and my life in general.

Vita

June 2015 ....Cambridge International General Certificate of Education Advanced Level, Taylor's College Subang Jaya, Malaysia

July 2018 ....Operation and Maintenance Engineering Intern, Edra Power Holdings Sdn. Bhd., Kuala Langat, Malaysia

January 2018 to present ....Undergraduate Teaching Assistant, Department of Engineering Education, The Ohio State University

August 2018 to present ....Undergraduate Research Assistant, Mechanical and Aerospace Engineering Department, The Ohio State University

Fields of Study

Major Field: Mechanical Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1 : Introduction

1.1: Background and Motivation

The remarkable launch of the world's first artificial satellite, Sputnik 1 had started a new era of scientific, technological, and political achievements, which is also known as the Space Age, revolutionary technologies like computers, space launch vehicles, nanomaterials, robots, and artificial intelligence have continued to advance rapidly after the dawn of the Space Age. On February 3rd, 1966, Luna 9 was the first successful unmanned spacecraft that achieved a soft landing on the Moon and transmitted a series of lunar surface photographs to the planet Earth [1].

Figure 1.1: Luna 9

https://lunarexploration.esa.int/explore/missions/239?ia=334

The first in a series of robot lunar rovers, Lunokhod 1 was carried to the Moon by the Luna 17 on the November 17, 1970; Lunokhod 1 with an estimated lifetime of three lunar days managed to travel 10.54 kilometers, returned more than 20,000 TV images and 206 high resolution panoramas, and used its built-in RIFMA X-ray fluorescence spectrometer to conduct 25 soil analysis in 11 lunar days [2]. Later in the year 1976, National Aeronautics and Space Administration's (NASA) Viking 1 was the first rover that landed on Mars; the lander took high-resolution images and performed the first rover that landed on Mars; the lander took high-resolution images and performed the first Martian soil sample successfully with its special biological laboratory and a robotic arm to study the Martian soil sample successfully with its special biological laboratory and a robotic arm to study the Martian uninhabitable environment and to search life on Mars [3].

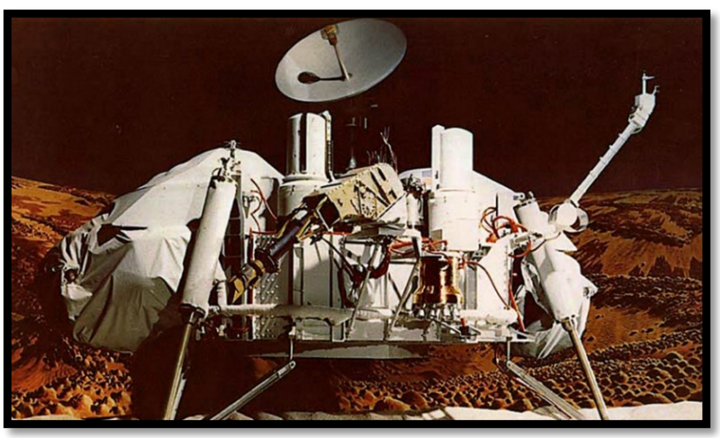

Figure 1.2: Viking 1

https://www.jpl.nasa.gov/missions/viking-1/

The development and applications of unmanned ground vehicles (UGVs) have significant contributions to today's science and technology advancement by performing planetary experiments and providing crucial information for potential, future manned missions. Therefore, robotics missions have become a part of the beginning and vital stage of the interplanetary space explorations without risking human lives.

Although current generation rovers including Mars Science Laboratory (MSL) Curiosity Rover have proven their distinctions, they are large and heavy, for example, Curiosity Rover has a similar size of a golf cart and weight of 1,982 pounds [4]. Huge size and large weight of a rover will increase the landing impact damage of a space capsule along with a rover; weight and size problems will consume more power to explore in a rough and extreme environment and reduce rovers' operation time. For instance, the first comet lander, Philae landed on the Comet 67P/Churyumov–Gerasimenko successfully in 2014, but it lost communication with the comet orbiter, Rosetta due to the inability of climbing the rocky surface of the comet and the low battery power [5]. Therefore, the upcoming rover design will need to be re-engineered to achieve increasingly ambitious interplanetary missions' goals and objectives.

1.2: Research Significance

As humanity discovers more about the universe we live in, the variety of technologies available to explore unfamiliar terrain has dramatically increased. The recent exploding development of robotic vehicle systems has been one of these technologies which include unmanned aerial and ground vehicle systems has been one of these technologies which include unmanned ground and aerial vehicles (UGVs and UAVs), respectively. UGVs and UAVs both have their advantages and disadvantages. For instance, while UGVs are able to explore tight spaces without the hindrance of rotating propellers, UGVs are unable to overcome major changes in increased elevation. On the other hand, although UAVs can generally travel at a higher speed than UGVs, UAVs will require more control efforts in unfavorable atmospheric conditions. As a result, a noble solution is required to tackle the challenges that both UGVs and UAVs are facing for interplanetary exploration purposes.

One of the possible solutions is the wheeled jumping miniature rovers. The development of the robotic jumping mechanism has been extensively investigated. These methods include the deformation of wheels [6] and shifting of internal masses [7]. Given that a rover can both attain forward movement by using both rotating wheels and a jumping mechanism, there is a major flexibility improvement when traveling in environments with obstacles. For motion planning of a jumping rover, the path over or on top of an obstacle can be treated as one part of a planned path given that rovers have the jumping capabilities to hop onto or over an obstacle below certain heights. Much of the existing studies on the control of a jumping rover entail the precise operation of motion. This includes legged motion with calculations into speed and torque [8] and motion control of a jumping rover that uses inertial force with a tail as its jumping mechanism [9]. The existing studies for motion planning of a jumping robot focus on finding an optimal position for a jumping robot by prioritizing safety and minimizing the jumping cost. Although obstacles have been considered when planning the jumping motion, each obstacle is treated as a point that cannot be used as a suitable surface if they were landed upon.

Another approach to improve the flexibility of a robotic system is the involvement of a cooperative robot team, especially for missions with multiple tasks that can be jointly accomplished by multiple robots [10], [11]. It can be predicted that the flexibility and mobility of a robotic system can be further improved by a cooperative jumping rover team where each team member has the hybrid operational modes while being able to perform assigned subtasks toward the overall mission goal. This paper focuses on motion planning and task allocation for a jumping rover team to visit several target locations in an optimal manner, known as the multiple traveling salesman problem (mTSP) [12].

Approaches to solving the traveling salesman problem (TSP) have been investigated over the years ranging from the use of genetic algorithms in an iterative approach [13] to the optimization of multiple simultaneous TSPs [14]. The work in [15] discusses the use of multiple robots to cover a 3D searching area, which is formulated as a single TSP. The use of greedy and optimal solutions in [15] directed the four robots to efficiently cover areas for a search-and-rescue type scenario. While extending the TSP to three dimensions, the robots in [15] were restricted to one form of locomotion, which indicates obstacles are treated as traversable only if a viable path, such as a ramp, was available. In contrast, a jumping rover removes the critical constraint of obstacle, which enables a traversable terrain for the entire operating environment. As a result, an optimal path for each jumping rover can be planned in the mTSP by taking into account traditional avoidance procedures, as well as creating other possibilities for each obstacle encountered. Although many algorithms for motion planning with avoidance zones have been developed, e.g., particle swarming optimization [16] and variations of the genetic algorithm [17], none of these works consider an obstacle in the operating environments as a possible pathway using both the jumping and rolling mechanisms.

1.3: Thesis Focus

To assign visiting tasks to a jumping rover team and simultaneously plan paths between any two targets, a refined and optimized rapidly-exploring random tree (RRT*) method, specifically designed for motion planning of a jumping rover, is combined with the mixed-integer linear programming (MILP) to allocate the visiting tasks. This method finds the optimal route so that each target is visited only once while minimizing a designated performance index. Targets or visiting points may be located on top of an obstacle. Adjacent obstacles of varying elevations are considered in the problem where a jumping rover may jump onto an obstacle when presented on top of another. The purpose of this research is to create a cost-effective and mission-capable robotic system that can roll on wheels or jump to explore an area of varying elevations.

1.4: Thesis Overview

This undergraduate honors thesis consists of six different chapters, and it is organized as follows. ***Chapter 2*** presents the problem statement of the research. ***Chapter 3*** describes the motion planning and task allocation algorithm, which includes the RRT*, refined RRT*, and MILP algorithms. ***Chapter 4*** displays the algorithm implementations, indoor simulation environments, and design and construction of jumping rovers. The simulation and experimental results are presented in ***Chapter 5***. Last but not least, ***Chapter 6***, the conclusion, addresses the thesis' main contributions and future work of the research.

Chapter 2 : Problem Statement

The objective for a cooperative team of p UGVs with rolling and jumping capabilities is to travel the most cost-efficient route to visit a set of targets, denoted as $T = T_{\{1,2,3,\ldots,m\}}$, where m is the total number of targets within an operating area; the UGVs are indexed as $z = 1,2,3, \ldots, p$. UGVs in the team can reach to different jumping heights, denoted by h_z. The cost considered here could be time, energy consumption, or distance. In addition to the specified visiting points or targets, the obstacles that are randomly scattered in the area are considered and denoted by $O = O_{\{1,2,3,\ldots,n\}}$, where n is the total number of obstacles. Partial of the targets are assigned on top of the obstacles. Each obstacle is a rectangle prism with a known length, width, and height dimensions. Obstacles are treated as solid and rigid objects that cannot be passed through and cannot be moved by UGVs. Although obstacle borders can reside in an adjacent, collinear orientation, such as O_3 and O_4 as shown in *Figure 2.1*, obstacles' borders cannot intersect or be inside each other. In addition, obstacles are placed so their borders do not extend outside a buffer zone that is inside the test area as illustrated in *Figure 2.1*. As a result, this provides an opportunity for jumping rovers to avoid obstacles near the edge of the operating area if necessary.

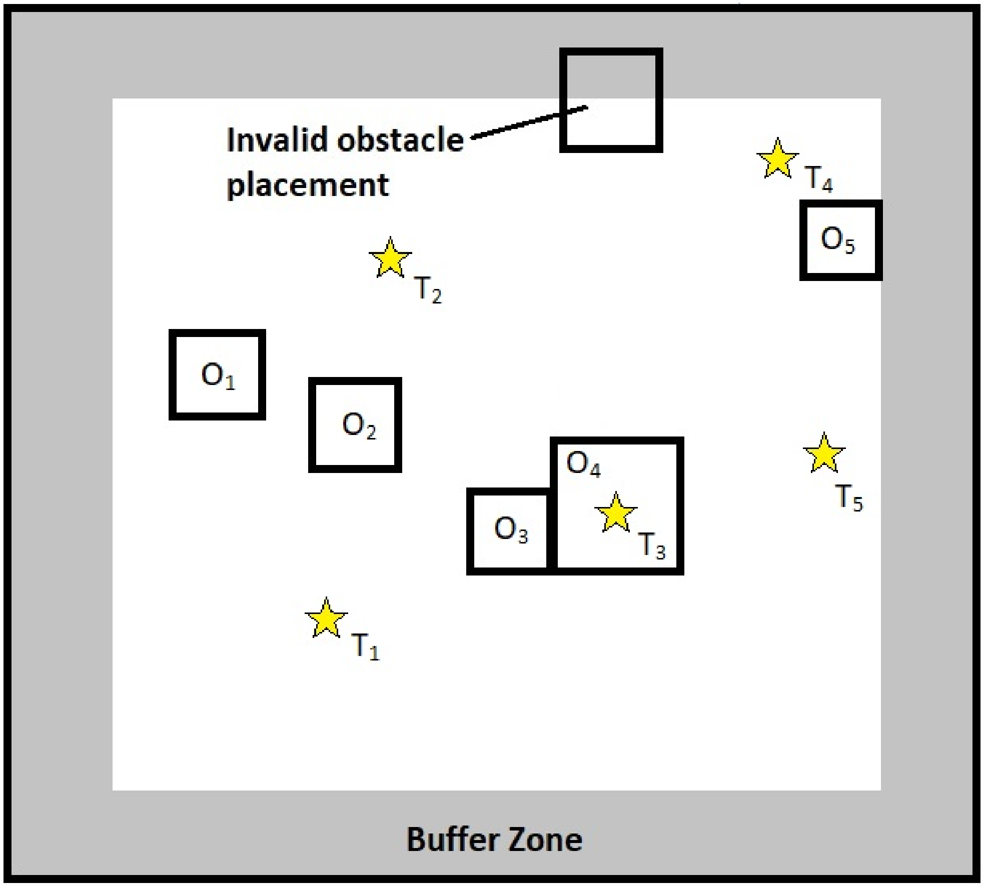

Figure 2.1: An Example Operating Area from Top View

The yellow stars are the target that jumping rovers need to visit, and the square boxes with bolded black outlines are the obstacle that jumping rovers can decide whether to avoid or jump onto it.

The overhead view of the test area as shown in *Figure 2.1* is regarded as a grid with a coordinate system based upon an X-Y plane. Within this view, the elevations for all grid points besides those occupied by obstacles are treated as constant, and all the obstacles can be jumped on top by jumping rovers.

Chapter 3 : Motion Planning And Task Allocation

3.1: Energy Consumption Model

The cost to be optimized in the mTSP of a jumping rover team can be a single performance index, for instance, time, energy, and distance, or a combination of weighted performance indices. As an example, energy consumption is considered as the cost to minimize in the mTSP. Consider a single UGV that operates with two different operational modes, rolling of wheels attached to the chassis and jumping motion through the actuation of spring. With a constant velocity during straight forward rolling and constant angular speed during zero turning radius for UGV $z, z = 1,2,3, \dots, p$, the power consumption rate of the jumping rover in straight forward and rotational motions are denoted by P_z^l and P_z^r, respectively. In addition, the passive power drawn from the vehicle's electronic components, such as the micro-controller, is a fixed value and denoted by P_z^a. The height of the jumping motion for each UGV is held constant, denoted by h_z, with a fixed energy expenditure associated with the corresponding jumping rover E_z^j. Then, the energy usage for each jumping rover traveling from target i to target j, $i, j = 1, 2, 3, \dots, m$, $i \neq j$, is determined by the equation as shown in the following:

$$c_{ij,z} = P_z^l t_{ij,z}^l + P_z^r t_{ij,z}^r + P_z^a \left(t_{ij,z}^l + t_{ij,z}^r\right) + E_z^j n j_{ij,z} \qquad \textit{(Equation 3.1)}$$

where $t_{ij,z}^l$ and $t_{ij,z}^r$ are the time duration of the straight forward and rotational motions between targets i and j, respectively, $nj_{ij,z}$ is the overall number of jumps for UGV $z, z = 1, 2, 3, \dots, p$, between targets i and j. By summarizing the energy consumption of all jumping rovers of all paths between any two targets, the overall UGV team energy usage can be found during the mTSP mission.

3.2: Introduction to Rapidly-Exploring Random Tree (RRT) and Rapidly-Exploring Random Star (RRT)*

Rapidly-exploring random tree (RRT) is a path planning tool, initially developed by S. LaValle and J. Jr. James [18], [19]. RRT is a sampling-based heuristic search algorithm that simulates an incremental space-filling tree exploring design space. The random nature of the RRT is essential to the algorithm's speed as opposed to methodically searching a space. Each tree begins at an initial point, x_{init}, and attempts to make a connection between the origin and a random point, x_{rand}, in the area. The length of the connection is dictated by an established unit length, Δx. The connection in the direction of the random point is made with the nearest point in the tree, x_{near}, to a new point, x_{new}, which can be reached. Basically, a unit vector multiplied by a scalar, Δx, in the direction of the random point. This configuration is added to the result data and a new connection made. The process is repeated for the number of desired iterations, K. The RRT algorithm can be simply illustrated by the RRT function, T pseudocode as shown in the following:

```
RRT ( x_init, K, Δx )
T.init(x_init)
for k ← 1 to K do
        x_rand ← RANDOM_STATE( )
        x_near ← NEAREST_POINT(x_rand, T)
        x_new  ← NEW_CONFI  (x_rand, T, Δx)
        T.add_vertex(x_new)
        T.add_edge(x_near, x_new)
end for
return T
```

The value of x_{rand} can be replaced by a final destination point, x_{dest} if the path-finding problem provides a value of x_{dest}. Once the destination point is reached, the function T will be terminated. Hence, the RRT function with a final destination point, T_{dest} can be presented by the modified pseudocode as shown below:

```
RRT ( x_init, x_dest, Δx )
T_dest.init(x_init)
while x_new ≠ x_dest, do
        x_rand ← RANDOM_STATE( )
        x_near ← NEAREST_POINT(x_rand, T)
        x_new  ← NEW_CONFI  (x_rand, T, Δx)
        T_dest.add_vertex(x_new)
        T_dest.add_edge(x_near, x_new)
end while
return T_dest
```

x_{near} must be remained outside of the obstacles, and any edges added cannot intrude into obstacles when the obstacle avoidance feature is implemented.

While effective in finding a solution with great speed, RRT can provide solutions that are inefficient in terms of the length of the tree path from x_{init} to x_{dest}. This is especially obvious when the function is constrained from avoidance zones. Thus, Rapidly-exploring Random Tree

Star (RRT*) is a method to find an optimal solution between the points in a tree. RRT* takes each point in a tree, finds the points within a radius (or neighborhood) of each point and replaces existing edges to that point with the most efficient path without violating constraints like intruding into the obstacles.

3.3: Modification of Rapidly-Exploring Random Star (RRT) for Jumping Rovers*

Developing the path planning algorithm for the jumping rover needs to consider multiple pathway options including avoiding and jumping onto obstacles. For each obstacle encountered on an otherwise straight-line path via RRT* to the target, the algorithm splits a new tree off the original tree starting from the point where the obstacle was encountered; the original tree avoids the obstacle and the new tree jumps onto the obstacles. This process is repeated for each obstacle encountered by the original tree. Once the original tree has reached its designated target, each new tree created is analyzed with the possibility that it will also split upon each obstacle encountered. The tree that has the minimum cost between two targets is chosen as a candidate for the path segment of mTSP. This process is repeated for each combination of targets. The Jumping Rovers' RRT* flow chart is summarized in *Figure 3.1*.

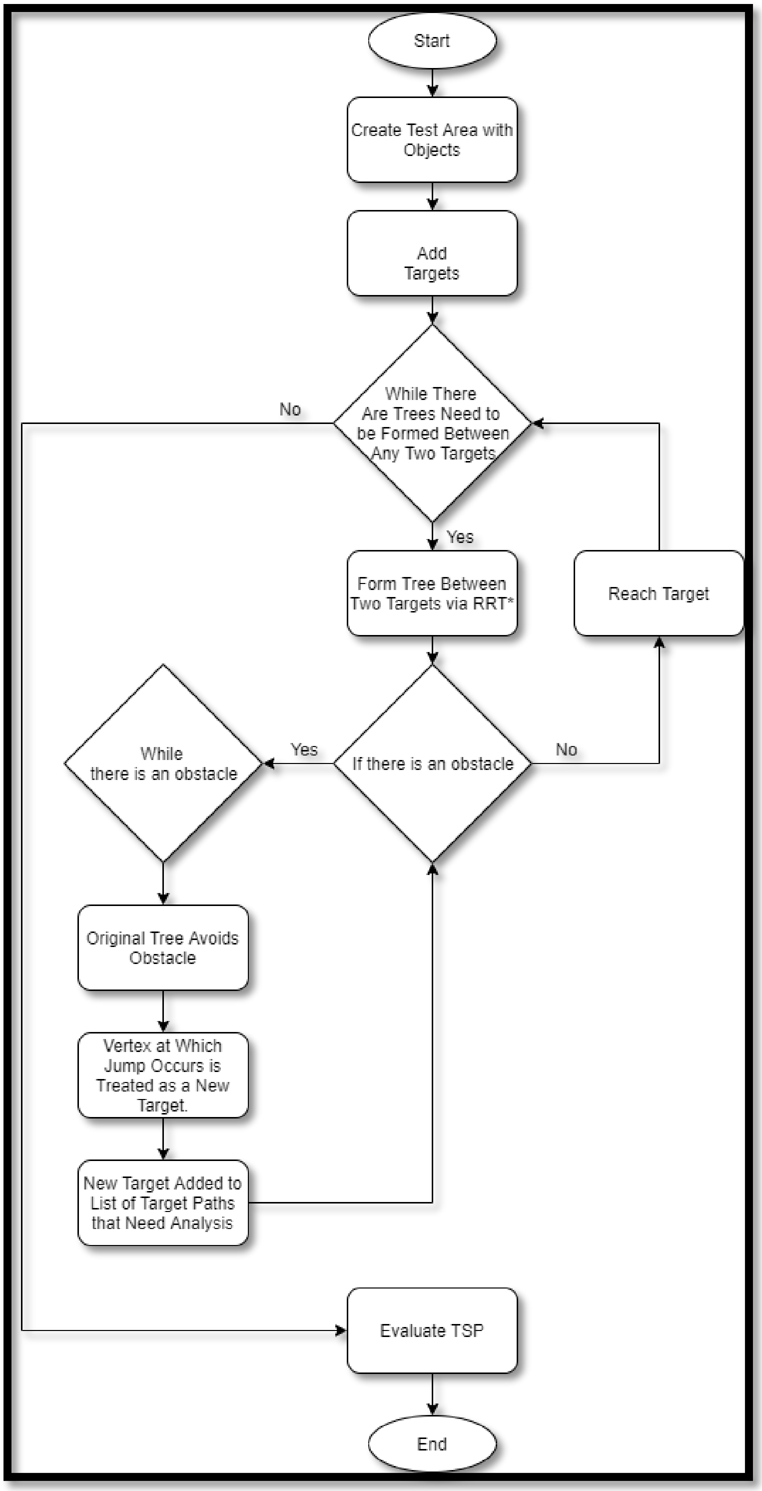

Figure 3.1: Jumping Rovers' RRT* Flow Chart

After all the possible trees between any two targets were found, the best trees with the least cost are selected and placed in a pool to be evaluated by the Energy Consumption Model via the mixed-integer linear programming (MILP). The MILP can find the global optimum solution to the mTSP by determining the best combination of tree paths between targets to form an overall route for each jumping rover.

3.4: Jumping Rovers' Refined (RRT)*

RRT* is restricted to optimization within a radius around a vertex in question or within a "neighborhood." Due to this limitation, RRT* may not provide a smooth solution that is traversable between target locations. To compensate for the limitation, the refined RRT* method was proposed with the process shown by the refined RRT* flow chart in *Figure 3.2* and a direct illustration in *Figure 3.3*. Detailed implementation of the refined RRT* for the jumping rover is further explained in ***Chapter 4.2***.

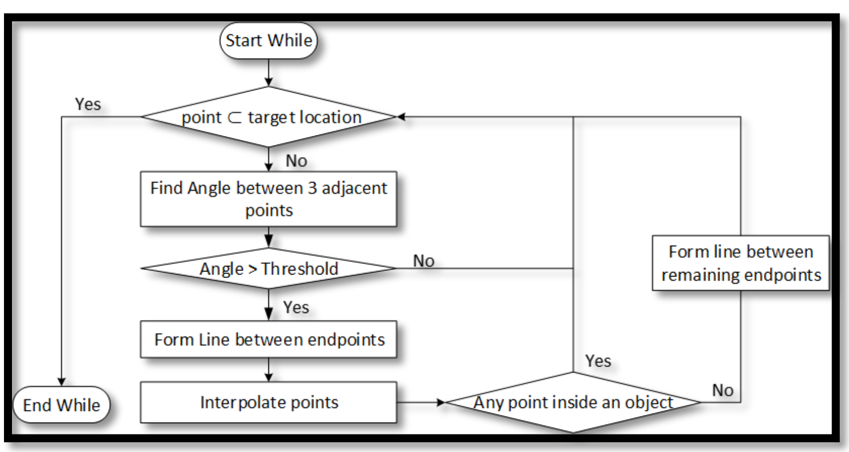

Figure 3.2: Refined RRT* Flow Chart

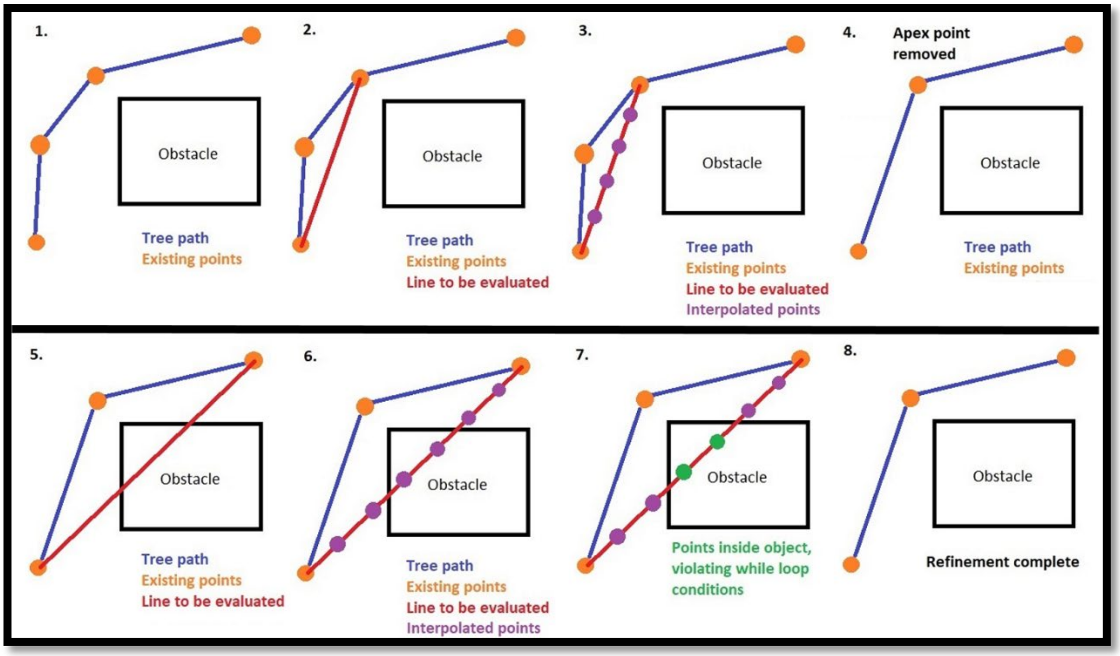

Figure 3.3: Refined RRT* with an Obstacle Illustration

3.5: Task Allocation via MILP

From the refined RRT*, the energy-efficient paths between any two visiting targets can be found. Next, the mTSP is formulated as a MILP to find the best combination of all treelike paths generated from the refined RRT* and simultaneously assign a jumping rover to every selected path segment. Each target in the mission is to be visited once by only one UGV, and all of the routes must begin and end at the same depot. This problem is classified as an mTSP which is a generalization of the TSP. The mTSP problem will be the same as the TSP when the value of z is equal to 1.

The mTSP can be represented by a complete graph G which consists of a set of targets T, denoted as vertices of the graph, and a set of edges, E (connections between any two target points); associated with each edge $(i, j) \in E$ is the edge cost for UGV z travels along that edge, denoted by $c_{ij,z}$. Since a jumping rover may land on the top of an obstacle, the maximum elevation along the edge $(i,j) \in E$, denoted by $h(i,j)$, should be less equal than the jumping height of the jumping rover. Furthermore, a binary three-index variable, $x_{ij,z}$ for edge $(i,j) \in E$ is defined as:

$$x_{ij,z} = \begin{cases} 1 & edge\ (i,j)\ \in E\ will\ be\ visited\ by\ UGV\ z \\ 0 & edge(i,j)\ \in E\ will\ not\ be\ visited\ by\ UGV\ z \end{cases}$$

Then, the mTSP with energy consumption model is formulated as:

$$min\, G(T,E) \qquad \textit{(Equation 3.2)}$$

$$min \sum_{i=1}^{m} \sum_{j=1,j\neq i}^{m} \sum_{z=1}^{p} c_{ij,z} x_{ij,z} \qquad \textit{(Equation 3.3)}$$

$$min \sum_{i=1}^{m} \sum_{j=1,j\neq i}^{m} \sum_{z=1}^{p} (P_z^l t_{ij,z}^l + P_z^r t_{ij,z}^r + P_z^a \left(t_{ij,z}^l + t_{ij,z}^r\right) + E_z^j n j_{ij,z}) x_{ij,z} \qquad \textit{(Equation 3.4)}$$

$$s.t. \quad \sum_{i=1,i\neq j}^{m} \sum_{z=1}^{p} x_{ij,z} = p, j = 1 \qquad \textit{(Equation 3.5)}$$

$$\sum_{j=1,j\neq i}^{m} \sum_{z=1}^{p} x_{ij,z} = p, i = 1 \qquad \textit{(Equation 3.6)}$$

$$\sum_{i=1,i\neq j}^{m} \sum_{z=1}^{p} x_{ij,z} = 1, j = 2, 3, \dots, m \qquad \textit{(Equation 3.7)}$$

$$\sum_{j=1,j\neq i}^{m} \sum_{z=1}^{p} x_{ij,z} = 1, i = 2, 3, \dots, m \qquad \textit{(Equation 3.8)}$$

$$u_i - u_j + q \sum_{z=1}^{p} x_{ij,z} \leq q - 1, 2 \leq i \neq j \leq m \qquad \textit{(Equation 3.9)}$$

$$x_{ij,z} \in \{0,1\}, \forall z, i, j \qquad \textit{(Equation 3.10)}$$

$$x_{ij,z} h(i,j) \leq h_z, \forall z, i, j \qquad \textit{(Equation 3.11)}$$

where u_i and u_j are the positions of the targets i and j, respectively, and q is the maximum number of targets that can be visited by any rovers. The depot is assumed to be located at T_1. *Equation 3.5* and *Equation 3.6* constraints ensure exactly p UGVs return to the depot T_1 and p UGVs depart from the depot T_1, respectively. *Equation 3.7* and *Equation 3.8* ensure only one UGV enters each target and only one UGV leaves each target. *Equation 3.9* is the extensions of Miller-Tucker-Zemlin-based sub-tour elimination constraints that ensure there are no sub-routes among the non-starting targets [20]. As a result, the solution returned is a single tour instead of the union of smaller tours for each UGV. The last constraints are the jumping height constraints such that the maximum elevation along the edge $(i, j) \in E$ can be achieved by UGV z.

Chapter 4 : Experimental Environments And Customized Jumping Rovers

4.1: Experimental Environments

The scenario developed for the experimental verification of the jumping rover team uses several obstacles in different shapes and sizes, illustrated in *Figure 4.1*.

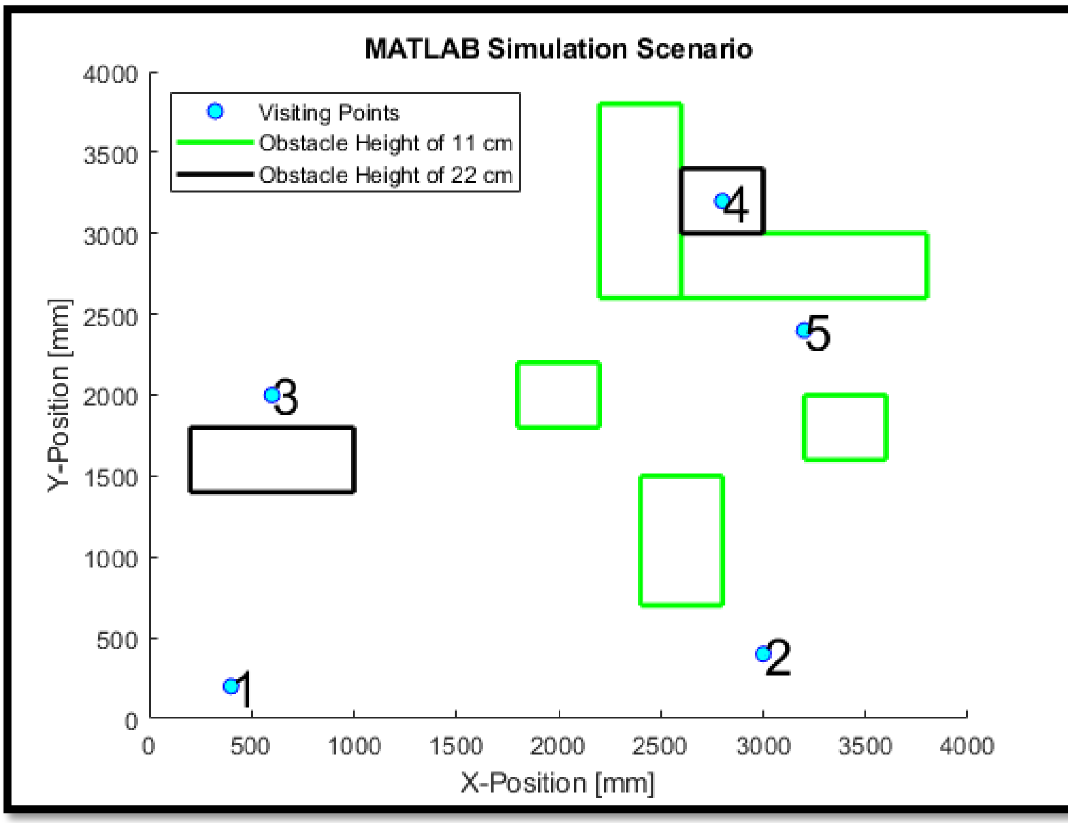

Figure 4.1: MATLAB Simulation's mTSP Scenario

Obstacles and targets are distributed in a manner to provide different challenges for the developed algorithm to overcome. This includes targets directly behind obstacles, targets on top of obstacles, and targets with multiple obstacles obstructing a direct line-of-sight path. Targets are represented by filled, blue circles and numbered with a black font as shown in *Figure 4.1*. Objects are represented as colored rectangles and numbered with a corresponding color font. Note that all spaces not encompassed by a colored rectangle indicate ground level, space surrounded by the lines of a green rectangle indicates an approximate height of 11 cm and space surrounded by the lines of the black rectangle indicates an approximate height of 22 cm as illustrated in *Figure 4.1*. Experimental tests were performed indoors in the Automation and Optimization Laboratory in Scott Laboratory (i.e., Mechanical and Aerospace Engineering Department) at the Ohio State University. Obstacles used were cardboard structures of various dimensions while targets are indicated by taped markers throughout the test area as shown in *Figure 4.2*.

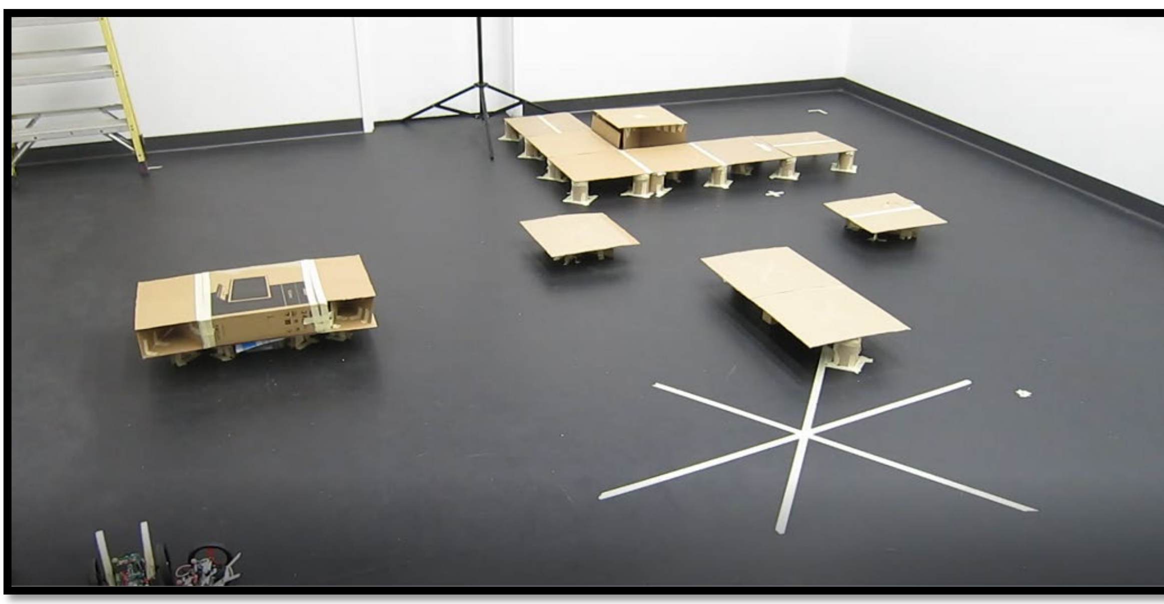

Figure 4.2: The mTSP Experimental Environment

4.2: Algorithm Implementation

Trees are formed between targets via RRT as shown in *Figure 4.3* and are optimized by RRT* to form a cost-efficient path between a query point and an existing point in the tree within a given radius. If an obstacle is encountered, the default algorithm reaction is to avoid the obstacle, create a new branch from the obstacle encountered, and continue till reaching the target, as outlined in ***Chapter 3.4*** and shown in *Figure 4.4*. The series of green thin lines indicate the path-planning formed by RRT* to reach target 3 from target 1. The tree is optimized multiple times to minimize its cost giving it a "splintered" appearance while also creating a connected path from its initial point to the destination point.

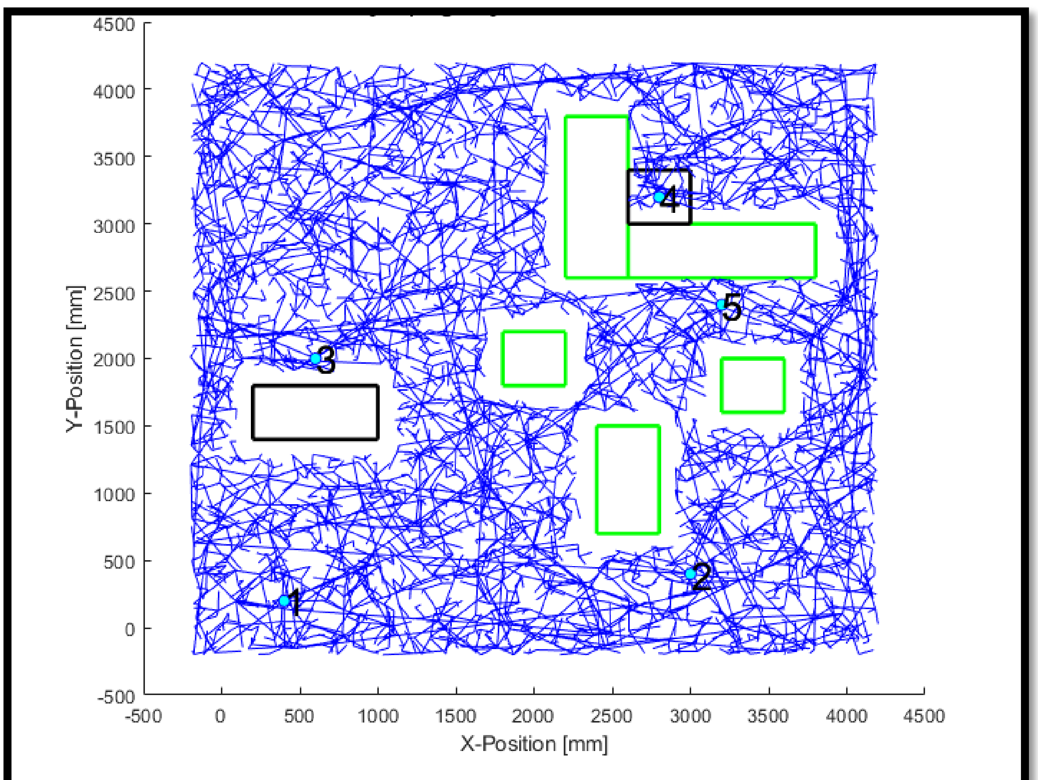

Figure 4.3: Completed RRT Paths Generation

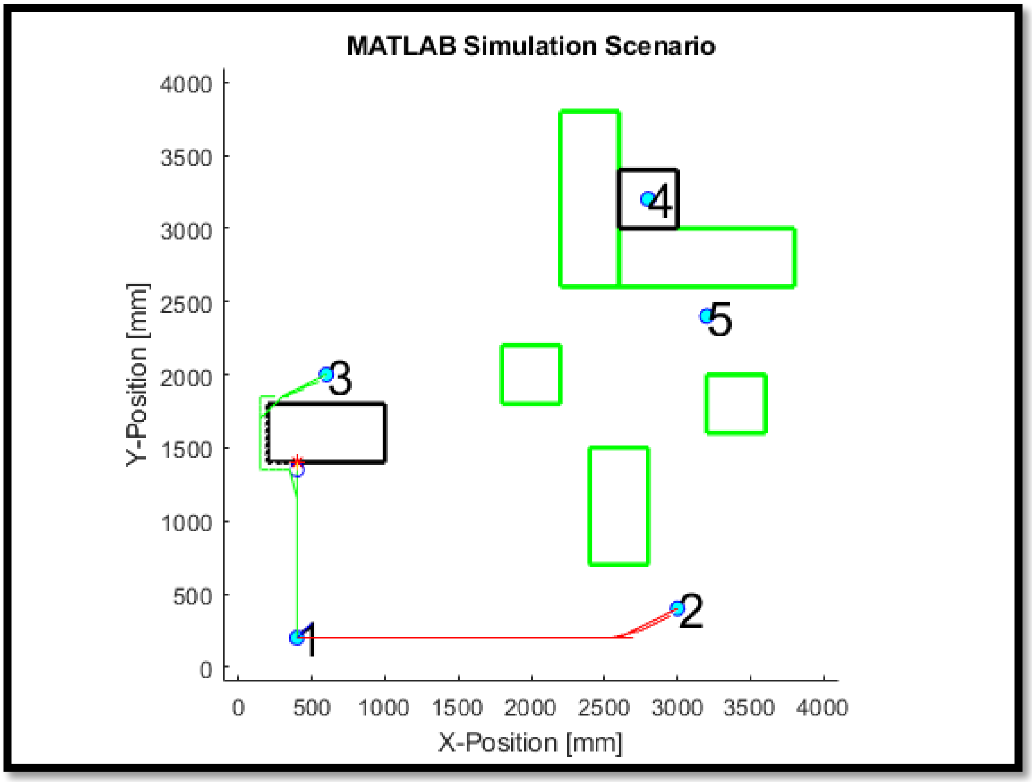

Figure 4.4: Trees Between Target 1 and Target 2 and Between Target 1 and 3.

As illustrated in *Figure 4.4*, the empty blue circle indicates the point at which an obstacle is first encountered and requires jumping. The red star indicates the theoretical landing location of the jump and is also the beginning of a new tree formed with target 3. In other words, the red star is treated as a new target location but is strictly a tree to be formed with target 3 and not with any other targets. The process continues with other trees formed between original targets. These trees formed also produce their own additional targets due to jumping and avoiding obstacles. The algorithm continues to form each tree between targets and new ones that were formed to account for the need to jump until a complete graph is constructed, shown in *Figure 4.5*.

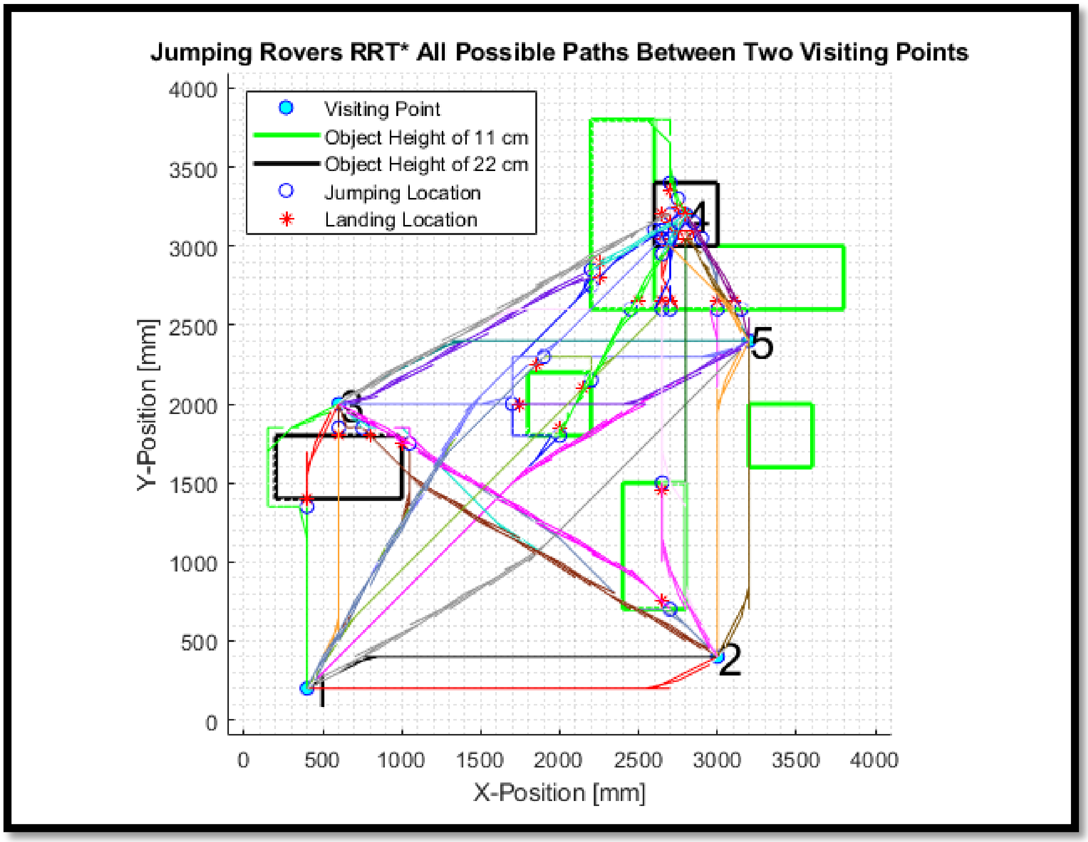

*Figure 4.5: All Possible Paths or Trees Generated by RRT**

The solution from RRT* without refinement is a coarse path that is not continuous, nor traversable. In light of this, the refined RRT* is applied to make all paths segments traversable, Next, using an open-looped single TSP as an example, the MILP formulated in ***Chapter 3.5*** is applied to produce a solution with minimum energy usage. The energy usage includes the rolling motion of the wheels, rotation of the rover to change its heading, and jumping, as shown in *Figure 4.6*. The single TSP results are shown in *Figure 4.6*.

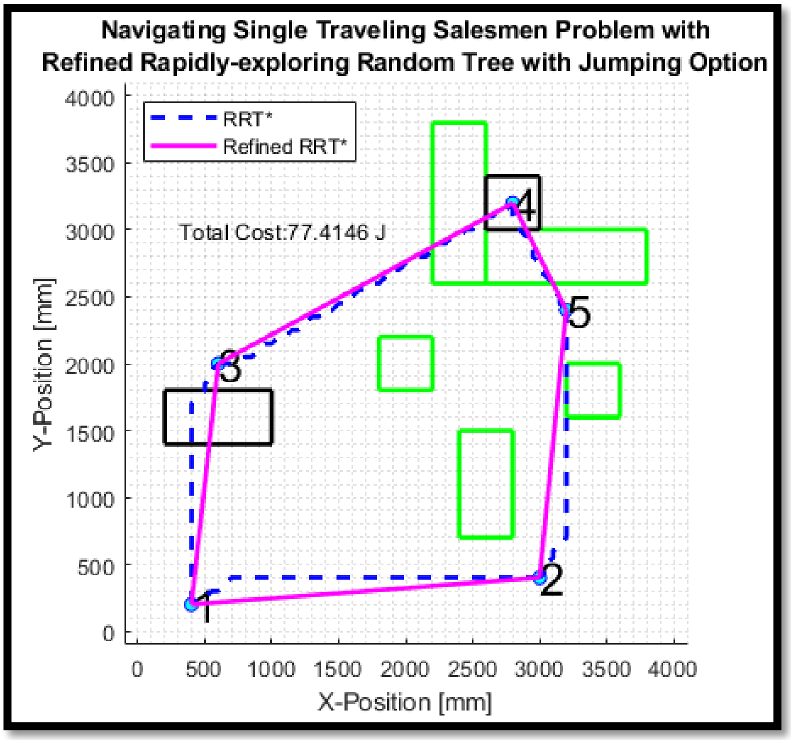

Figure 4.6: Single TSP's Solution Using Refined RRT* and RRT* Paths

4.3: Jumping Rovers' Design and Construction

For a rover to be used in this simulation, there are multiple requirements. First and foremost, it must be capable of both rolling motions to produce ground-based locomotion and jumping motion to produce vertical displacement. Second, it must be able to record power consumption from both types of motion to properly validate the proposed algorithm. The wheeled, jumping rovers are based upon the Parrot Jumping Sumo robot chassis [21]. There are two jumping rovers with a similar design, but both jumping rovers have a different power consumption rate and jumping heights. One of the rovers was built for a longer range of operation time, and it is named Jumping Rover 2 as shown in *Figure 4.7*; it consumed lesser energy to operate but will only be able to jump onto an obstacle with a maximum height of 110 mm. On the other hand, the other jumping rover was built for a better and more powerful jumping capability, but it has a shorter operation duration because it requires more power to run; it can jump twice the height of the longer-range rover, which is approximately 220 mm, and it is named as Jumping Rover 1 as shown in *Figure 4.8*.

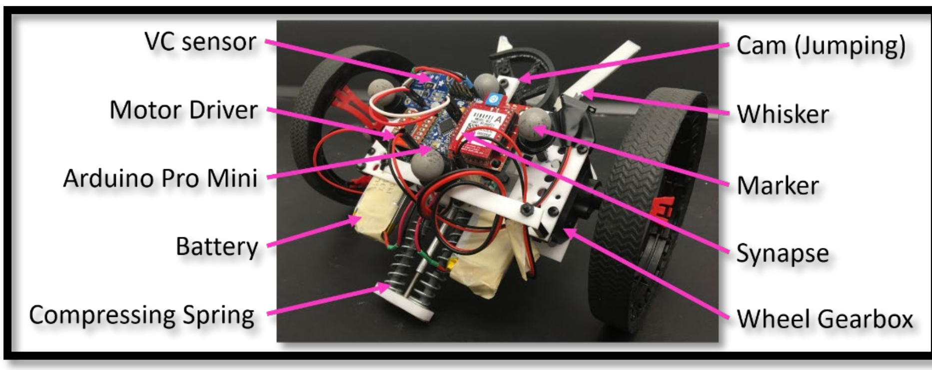

Figure 4.7: Jumping Rover 2 with Lesser Power Consumption and Lower Jumping Height

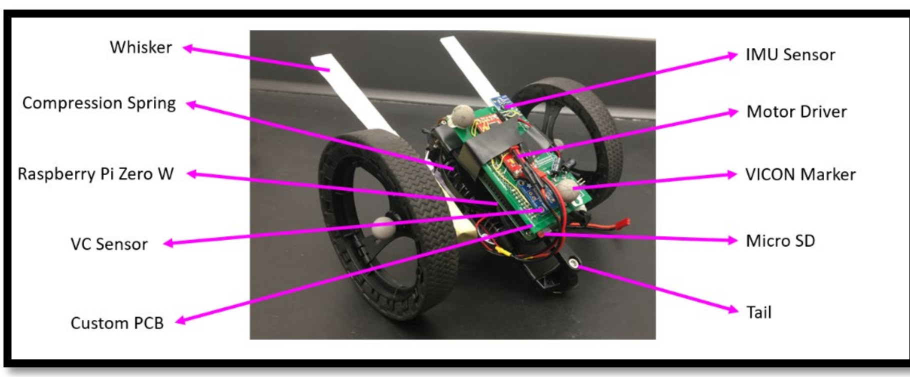

Figure 4.8: Jumping Rover 1 with More Power Consumption and Higher Jumping Height

Jumping Rover 2 and Jumping Rover 1 chassis was mainly built from acrylic, nylon, aluminum, acrylonitrile butadiene styrene (ABS) and 3D printed parts to ensure it to be lightweight in order to achieve an acceptable height. Gearboxes are used for the wheel motors to reduce the RPMs of the output. All circuits are placed on top of the jumping rover to allow ease of access for debugging. Markers for the motion tracking system, in the form of gray spheres, were arranged in an elevated, asymmetrical pattern so the location and the altitude of the jumping rover could be accurately determined. The center of mass sits on the force vector of the jumping mechanism to ensure the jumping rover does not flip after actuation. “Whiskers” in the front enable the rover to descend from the top of obstacles without flipping over since the spring mechanism can catch the edge of the obstacle in some cases.

Although both jumping rovers use a closed cam wheel to engage and pull the compression springs mechanism that enables the rover to jump as shown in *Figure 4.9* and *Figure 4.10*, Jumping Rover 1 has a cam profile with a shorter prime circle radius as compared to Jumping Rover 2’s cam profile, which allows Jumping Rover 1 to compress the springs further and jump higher after the jumping mechanism released the springs. As a result, each jumping rover has a fixed jumping height which is determined by the closed cam’s prime circle radius and the stiffness of the two compression springs. For each jump, the one with higher jumping height consumes two and a half times more energy than the other one.

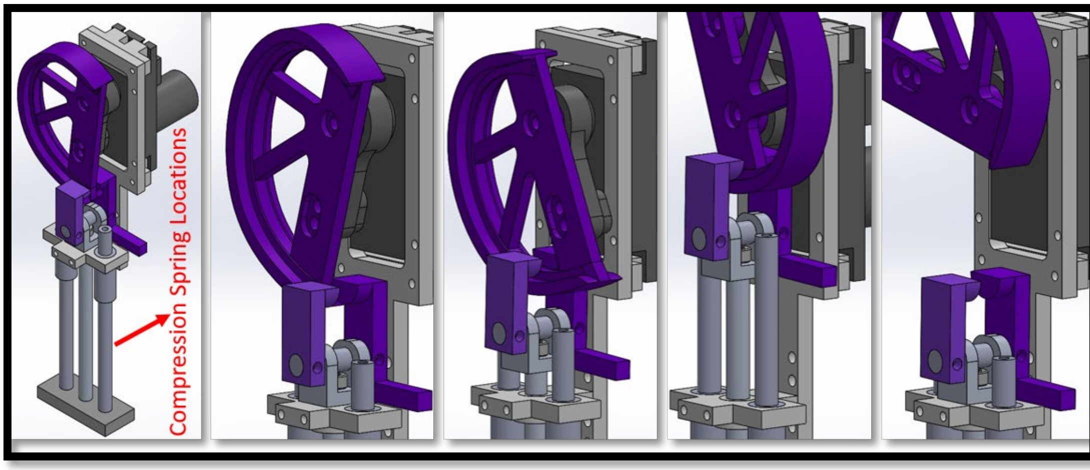

Figure 4.9: Jumping Rover 2's Jumping Mechanism Isometric View

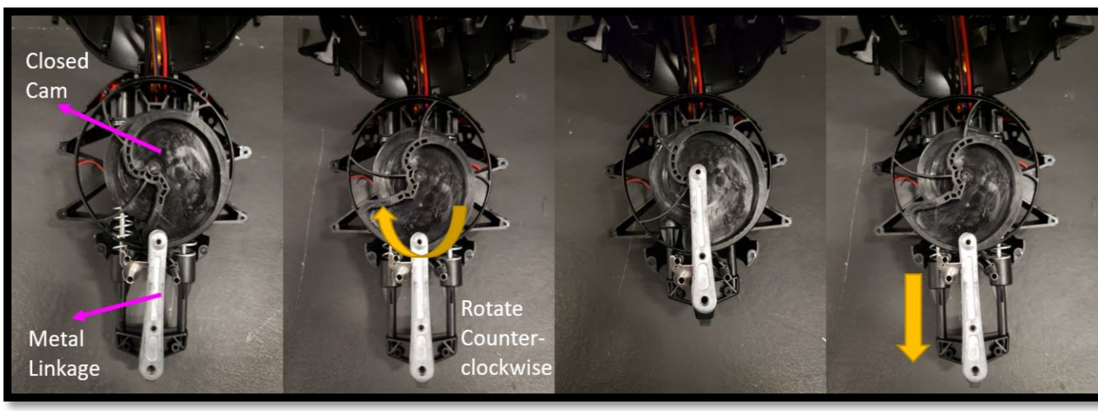

Figure 4.10: Jumping Rover 1's Jumping Mechanism Top View

Besides different cam profiles, Jumping Rover 2's jumping mechanism uses a servo motor to rotate the cam wheel instead of a DC motor because the servo motor is able to produce enough torque to engage the spring pullers and compress a high stiffness spring before launching as illustrated in *Figure 4.9*. On the other hand, Jumping Rover 1 uses the entire Parrot Jumping Sumo's unique jumping mechanism with a tiny 3.3 V DC gear motor that can produce sufficient torque to rotate the cam wheel. The rotating closed cam causes the metal linkage to compress the two springs until the metal linkage's pin fits into the cam wheel's dip as shown in *Figure 4.10*'s third sub-image. To launch the Jumping Rover 1, the cam will continue to rotate past the dip where the closed-cam slides downward sharply and release the compression springs causing the Jumping Rover 1 to jump.

The Parrot Jumping Sumo robot that can a off the shelf is manually operated by a user. In addition to the design of a more robust jumping mechanism described above, a micro-controller like Arduino or a mini-computer like Raspberry Pi Zero W and a communication system were integrated into the jumping rovers' design to achieve autonomous operation. Jumping Rover 2 is controlled wirelessly through serial commands with an Arduino Pro Mini via Synapse communication protocols with MATLAB and Arduino IDE. Since Jumping Rover 1 is built together with Raspberry Pi Zero W, it is controlled wirelessly via Wi-Fi with MATLAB and Python 3 IDE. A series of commands are then relayed to a motor controller that supplies power to the wheel motors. Signals for the Jumping Rover 2's jumping mechanism are applied directly from the Arduino to a servo motor; nevertheless, Jumping Rover 1's jumping mechanism is actuated by the motor controller and Raspberry Pi Zero W. Instructions for the optimal path are output from the MATLAB simulation and followed by the physical rover through an XY-coordinate system. The information flow chart for both jumping rovers is demonstrated in *Figure 4.11*.

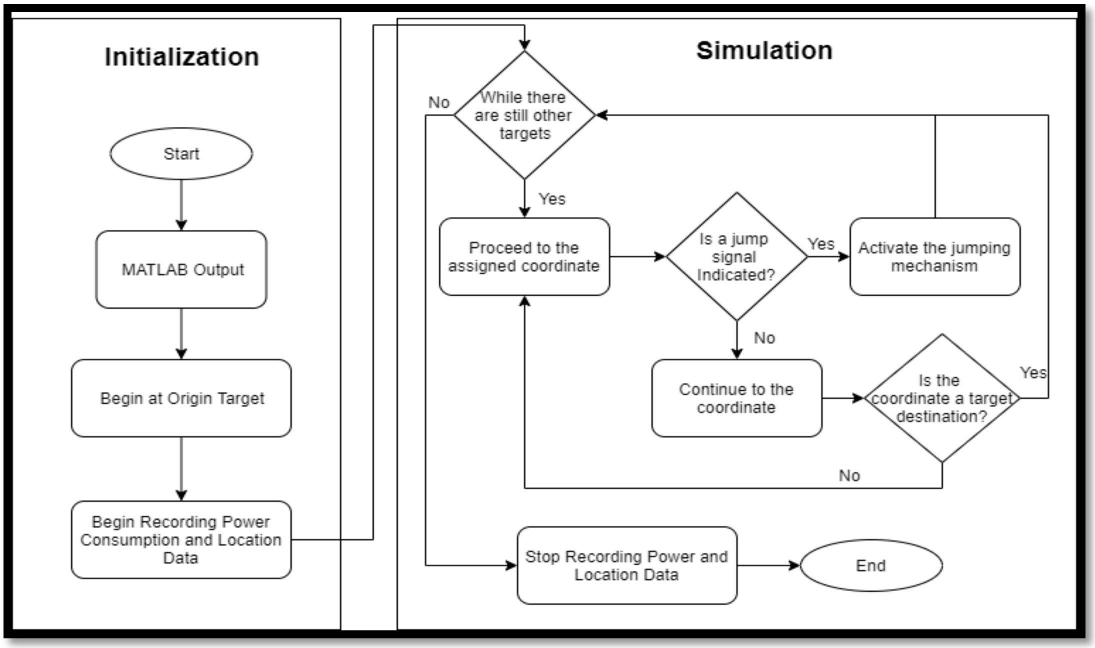

Figure 4.11: Information Flow Chart for Both Jumping Rovers

Power throughout both jumping rovers' electrical system is supplied by two Lithium Polymer (LiPo) batteries. Jumping Rover 2 and Jumping Rover 1's general system hookups are presented in *Figure 4.12* and *Figure 4.13*, respectively.

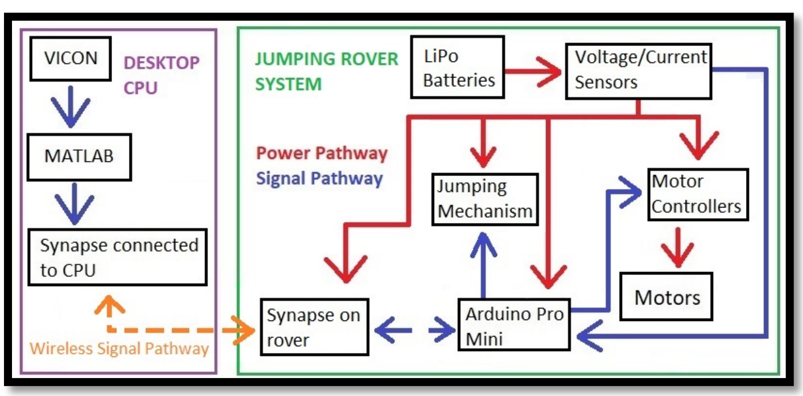

Figure 4.12: Jumping Rover 2's Power and Signal Pathways

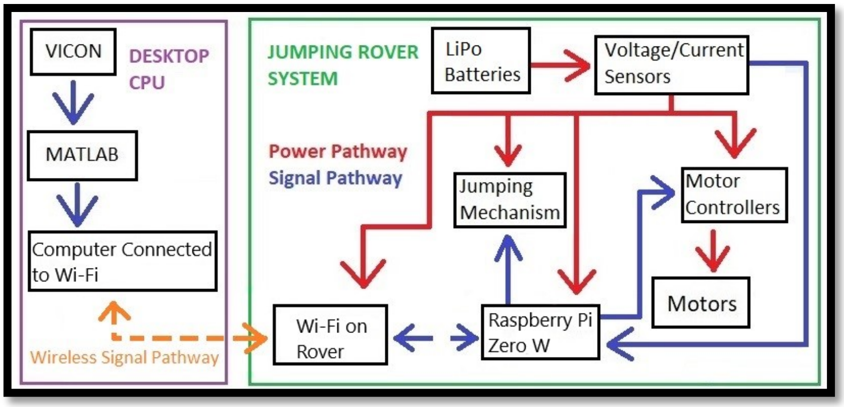

Figure 4.13: Jumping Rover 1's Power and Signal Pathway

As illustrated in *Figure 4.12* and *Figure 4.13*, power data is recorded at a fixed frequency using the voltage/current sensor (V/C sensor). Both jumping rovers are programmed to follow a set of coordinates put forth by the results of the motion planning and task allocation algorithm presented in ***Chapter 3***. Using the VICON motion tracking system to determine the location and attitude, MATLAB provides a series of commands to the Arduino via Synapse and Raspberry Pi Zero W via Wi-Fi simultaneously to dictate the motor rotation, allowing the jumping rover to reach each target within 65 mm tolerance.

Chapter 5 : Simulation and Experimental Results

Simulation and experimental results using the two constructed jumping rovers and the scenario described in ***Chapter 4*** are presented here. The trajectories of the simulation and experimental results of the mTSP are shown in *Figure 5.1*. The blue solid lines represent the rolling and jumping paths of Jumping Rover 1, which is known as Route 1. The red lines represent the rolling and jumping paths of Jumping Rover 2, which is known as Route 2. The planned paths indicate that when visiting target 3, the Jumping Rover 1 elects to jump on top of the obstacle, the Jumping Rover 2 elects to reach target 4 by jumping on top of a neighboring obstacle first. The time history of power consumption of two jumping rovers in the experimental test is shown in *Figure 5.2*. A sequential time-lapse of the jumping rover team performing the physical experiment of the mTSP mission images are provided in *Appendix C*.

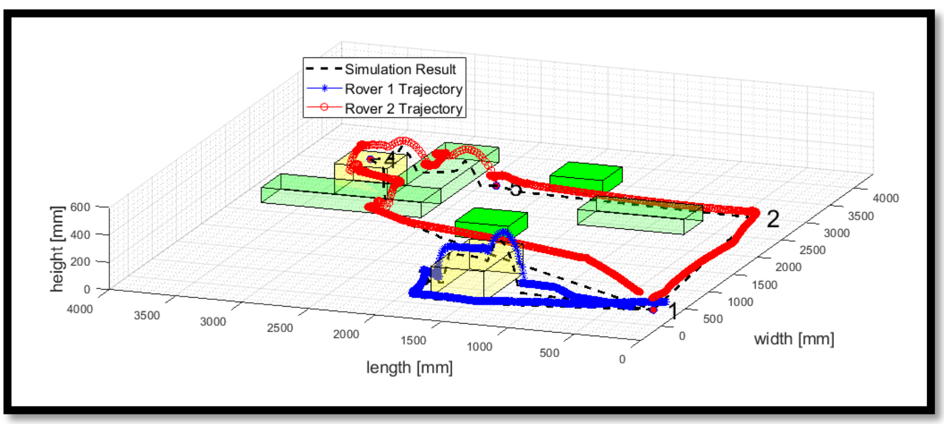

Figure 5.1: mTSP 3D Trajectories with Two Jumping Rovers

Compared to the simulation result in *Figure 5.1*, the experimental result shows slight differences. The major discrepancies are due to variations in location, attitude, jumping accuracy, and recovery time from jumping. As a result, this requires the rover to compensate and reach a coordinate using a different amount of power and time than originally planned. In addition, when a jumping rover is located next to an obstacle or is at a certain range of pitch angle while jumping at mid-air, the VICON motion tracking system may not be able to identify all of the markers on the rover due to the blocked view, which generates navigation errors.

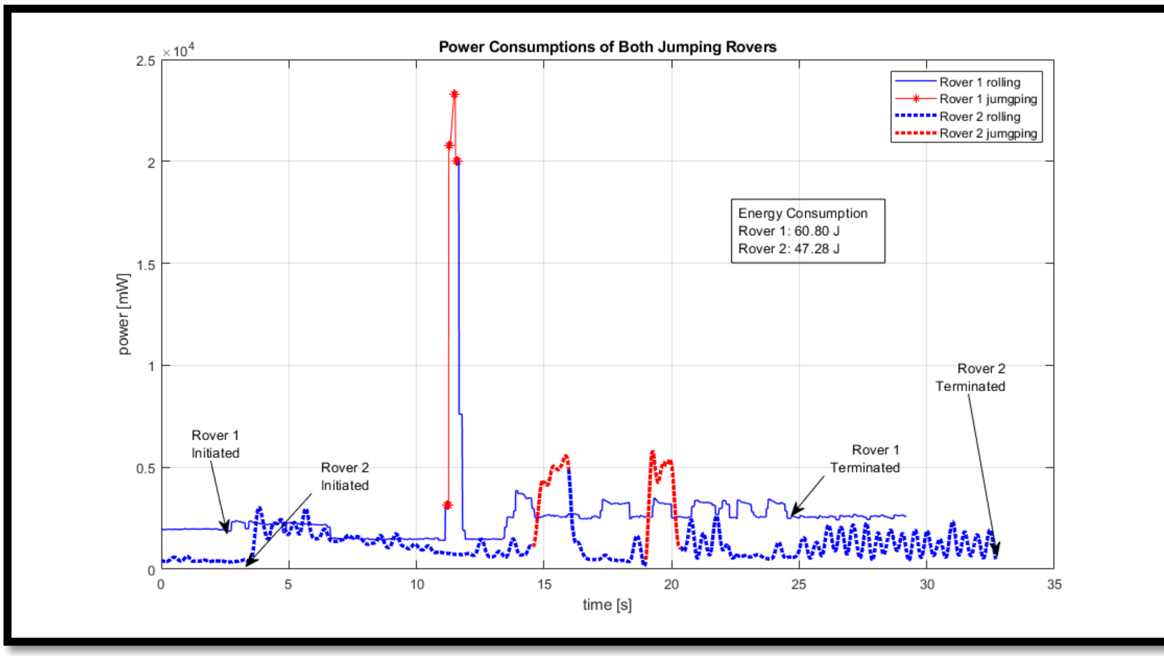

Figure 5.2: Time History of the Power Consumption of Both Jumping Rovers in the Experimental Test.

Table 5.1: Energy Consumed by Each Rover while Running at its Assigned Route.

Name of the Jumping Rover	Route Number	Experimental Energy Consumption [Joules, J]	Completion Time [seconds, s]
Jumping Rover 1	1	60.80	15.0
Jumping Rover 2	2	47.28	24.0

Based on *Figure 5.2* and *Table 5.1,* Jumping Rover 1 consumed 60.80 J of energy and took 15 seconds to complete Route 1; moreover, the energy consumption of Jumping Rover 2 is 13.52 J higher than the Jumping Rover 1's energy consumption, and it took 9 seconds longer to complete Route 2. *Figure 5.2* also shown that Jumping Rover 1 has a higher passive power drawn by the Raspberry Pi Zero W and other electronic components than the Jumping Rover 2's passive power drawn by the Arduino Pro Mini and its electronic components at the first 2.5 s of *Figure 5.2*. Furthermore, the estimated jumping energy consumed by Jumping Rover 1 is determined to be approximately 18.376 J, and the Jumping Rover 2's estimated jumping energy consumption is calculated to be approximately 7.280 J based on *Figure 5.2*. Both rovers' jumping energy consumption calculations are shown as following:

$$Jumping\ Energy, E_j\ [J] = Power\ Change, P\ [W] \times Jumping\ Period\ [s]$$

(Equation 5.1)

Jumping Rover 1's Jumping Energy Consumption Calculation:

$$E_{j,1} = (23290\ mW - 1414\ mW) \times (11.96\ s - 11.12\ s) = 18.376\ J$$

Jumping Rover 2's Jumping Energy Consumption Calculation:

$$E_{j,2} = (5671\ mW - 471\ mW) \times (20.39\ s - 18.99\ s) = 7.280\ J$$

Jumping Rover 1's overall rolling energy consumption is higher than Jumping Rover 2's overall rolling energy consumption as well based on *Figure 5.2*.

For the comparison purpose, an alternative result for the mTSP where jumping is avoided unless the jumping rover needs to reach a target assigned on top of an obstacle. The top view of the result in *Figure 5.1* and the alternative result without jumping options is presented in *Figure 5.3* and *Figure 5.4*, respectively. From the alternative result, it indicates that when jumping on top of an obstacle is not an option when avoiding an obstacle, Jumping Rover 2 elects to take a longer path to reach target 3 and Jumping Rover 1 elects to take a longer path to reach target 4. Compared to the solution in *Figure 5.3* with jumping option which consumes 105.74 Joules, the solution in *Figure 5.4* consumes 120.79 Joules. The comparative results indicate the energy reduction of 14.23 % with the jumping option for obstacle avoidance. Although time consumption is not considered in the performance index, the mission duration in both results is compared. It indicates that using the jumping option in *Figure 5.3*, it takes 15 seconds and 24 seconds, respectively, for Jumping Rover 1 and Jumping Rover 2 to finish their corresponding tasks. While for the results without jumping option, it takes 15.76 s and 29.2 seconds, respectively, for Jumping Rover 1 and Jumping Rover 2 to complete their corresponding tasks. The time reduction for the result with jumping option verifies the byproduct of time efficiency.

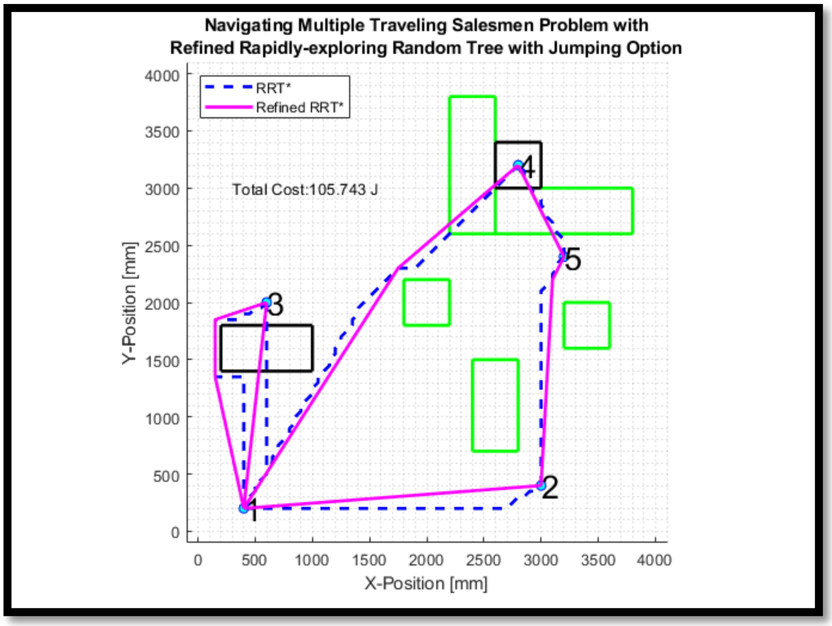

Figure 5.3: 2D Trajectories with Jumping Options

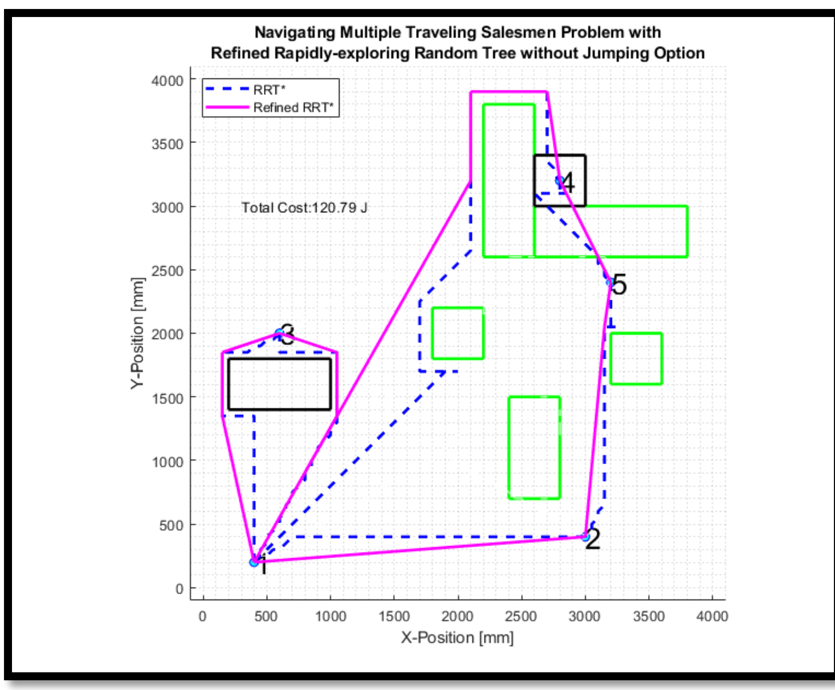

Figure 5.4: 2D Trajectories to Avoid Jump if Possible

Table 5.2: Comparative Results for the mTSP without Jumping Option

Type of Trajectories	**Simulation Total Energy Consumption [Joules, J]**	**Total Completion Time [seconds, s]**
2D Trajectories with Jumping Options	105.743	24.0
2D Trajectories to Avoid Jumping Options if Possible	120.790	29.2

The calculation of the total energy consumption percentage change between 2D trajectories with jumping options and without jumping options if possible is shown as following:

$$Total\ Energy\ Consumption\ Percentage\ Change\ [\%] = \frac{|E_{T,WJ} - E_{T,WOJ}|}{E_{T,WJ}} \times 100\ \%$$

(Equation 5.2)

$$Total\ Energy\ Consumption\ Percentage\ Change\ [\%] = \frac{|105.743 - 120.790|}{105.743} \times 100\ \%$$

$$Total\ Energy\ Consumption\ Percentage\ Change\ [\%] = 14.23\ \%$$

where, $E_{T,WJ}$ is the total energy consumption for the 2D trajectories with jumping options [Joules, J], and $E_{T,WOJ}$ is the total energy consumption for the 2D trajectories without jumping options if possible [Joules, J]. Based on *Table 5.2*, the total completion time for the 2D trajectories with jumping options is 5.2 seconds faster than the total completion time for the 2D trajectories without jumping options if possible.

Table 5.3: Experimental and Simulation Results Comparison for the 2D Trajectories with Jumping Options

	Experimental Results	Simulation Results
Total Energy Consumption [Joules, J]	108.080	105.743

According to *Table 5.3*, the experimental total energy consumption and the simulation total energy consumption for the 2D trajectories with jumping options are roughly the same, but the experimental total energy consumption result is 2.337 J higher than the simulation total energy consumption result due to variations in location, attitude, and jumping accuracy. The percentage error is calculated as the following:

$$Percentage\ Error\ [\%] = \frac{|E_{T,experimental} - E_{T,simulation}|}{E_{T,simulation}} \times 100\ \% \qquad \textit{(Equation 5.3)}$$

$$Percentage\ Error\ [\%] = \frac{|108.08\ J - 105.743\ J|}{105.743\ J} \times 100\ \%$$

$$Percentage\ Error\ [\%] = 2.21\%$$

Since the percentage error of the simulation result is 2.21%, which is less than 5.0 % as shown in the calculation above, the simulation total energy consumption result is considered valid.

Chapter 6 : Conclusion

The purpose of this research is to develop a cost-efficient and mission-capable robotic system with a fully autonomous cooperative team of miniature unmanned ground vehicles (UGVs) that can jump over obstacles to navigate a multiple traveling salesman problem (mTSP) and explore an area of varying elevations.

6.1: Contributions

Since the dawn of space-age in 1957 when the world's first artificial satellite, Sputnik 1 was successfully launched by the Soviet Union, new technological and scientific developments were emerged and advanced rapidly till today's modern society [22]; that one small step of the historical achievement in space exploration inspired many generations of people and motivated humans to push beyond boundaries to achieve further like making the first manned moon landing, Apollo 11 possible [23] and sending Mars Curiosity Rover to planet Mars to explore an area where it is risky for humans to explore due to unknown environmental factors [24]. However, as the space technology advances and humanity discover more about the universe we live in, the cost of robotics interplanetary exploration increases as well, for example, the cost of the Mars Curiosity mission is estimated to be USD 2.1 billion [24].

Developing a cost-efficient and smart robotic system with a team of fully autonomous and miniature rovers that have a jumping capability not only helps to reduce the risk of a mission failure, explore a rug environment like cliffs, canyons, and small caves but also allow humans to reduce the cost of a mission by 14.23 % as shown in ***Chapter 5***; for instance, if the cost of the Mars Curiosity Mission is estimated to be USD 2.0 billion, the implementation of the robotic system will help to save the expenditure of the interplanetary exploration by USD 284.6 million, which is not a small amount of money. Besides that, if the next generation of rovers is in a cooperative robotic system, it will help to mitigate the risk of failure of a space mission that worth billions of dollars because when one of the rovers fails to operate, the other rovers are expected to help on completing the remaining interplanetary exploration mission. Furthermore, miniature rovers with a jumping capability will be able to explore the rug environment easily without consuming a lot of energy due to lightweight. Last but not least, the cooperative robotic system that this research developed will also increase the time-efficiency of completing assigned tasks.

6.2: Additional Applications

Creating a cost-efficient and mission capable cooperative robotic system with an algorithm that is developed based on heuristic optimization methods integrating refined Rapidly-exploring Random Tree Star (RRT*) and Mixed-Integer Linear Programming (MILP) not only applies for interplanetary exploration mission, but it can also be very beneficial for search-and-rescue missions, surveillance missions like unmanned traffic monitoring, and autonomous driving vehicle application. The ability for the cooperative robotic system's heuristic optimization algorithm with a formulated main objective function and several mathematical constraints to plan and perform path planning, motion planning, and task allocations is an intelligent behavior's critical component, and it can be applied for most of automated machines to optimize their performance on a certain tasks or missions. For instance, the cooperative robotic system that is implemented on this research can be used for a search-and-rescue mission for a team of UGVs to maximize the area coverage and information gathering with a given set of mathematical constraints like energy constraints and time constraints to optimize and navigate mTSP effectively; the results can also be heavily investigated and compared with other heuristic optimization methods like particle swarm optimization and genetic algorithm to verify whether both approaches will give similar or distinct path planning results and to determine the advantages and disadvantageous of both heuristics optimization methods.

6.3: Future Work

In addition to creating a cost-efficient and mission-capable cooperative robotic system, the future research work will further improve and enhance the cooperative robotic system by developing a self-sustainable system with a solar-charging station rover. The new cooperative robotic system will still be applied to similar heuristic optimization methods, which is the RRT* and MILP to navigate mTSP. Furthermore, the use of multiple jumping rovers will be investigated in a more challenging outdoor environment like gaps and stairs. Besides that, future work will redesign the jumping mechanism of the jumping rover to allow the jumping rovers to control its jumping height, which helps to minimize jumping energy consumption of a rover.

6.4: Summary

This research develops a motion planning and task allocation method to find the energy-efficient solution of a multiple traveling salesman problem (mTSP) with obstacles using a jumping rover team. Each jumping rover has the capability to jump over obstacles under certain elevations and then treat the jumping route as a feasible path. The optimized rapidly-exploring random tree (RRT*) is improved by implementing a refined RRT* method to smooth paths between targets. The established path from the refined RRT* allows the formulation of the mTSP as a mixed-integer linear programming (MILP) problem to find the visiting sequence and simultaneously assign a jumping rover to each selected path segment. The results from virtual simulation and physical experiments demonstrate the improved performance using the jumping capability to solve the mTSP with obstacles and effectiveness of the proposed motion planning and task allocation methods.

Bibliography

[1] European Space Agency, "Robotic Landers," 2 November 2019. [Online]. Available: https://lunarexploration.esa.int/explore/missions/239?ha=334.

[2] National Aeronautics and Space Administration, "Luna 17," National Aeronautics and Space Administration, 26 January 2018. [Online]. Available: https://solarsystem.nasa.gov/missions/luna-17/in-depth/. [Accessed 2 November 2019].

[3] NASA Jet Propulsion Lab, "Mission to Mars Viking 1," National Aeronautics and Space Administration, [Online]. Available: https://www.jpl.nasa.gov/missions/viking-1/. [Accessed 2 November 2019].

[4] NASA Jet Propulsion Lab, "Discovery Guide: Mars Rover Curiosity," National Aeronautics and Space Administration, [Online]. Available: https://www.jpl.nasa.gov/education/marsrover.cfm. [Accessed 2 November 2019].

[5] European Space Agency, "Philae Found!," European Space Agency, 5 September 2016. [Online]. Available: https://www.esa.int/Science_Exploration/Space_Science/Rosetta/Philae_found. [Accessed 2 November 2019].

[6] C. Ye, B. Wang, B. Wei and B. Tang, "Modeling and Analysis of a Jumping Robot with Deforming Wheeled Mechanism," in *IEEE International Conference on Mechatronics and Automation (ICMA)*, 2018.

[7] Y. Mizumura, K. Ishibashi, S. Yamada, A. Takanishi and H. Ishii, "Mechanical Design of a Jumping and Rolling Spherical Robot for Children with Developmental Disorders," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2017.

[8] Y. Ding and H.-W. Park, "Design and Experimental Implementation of a Quasi-direct-drive Leg for Optimized Jumping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[9] N. Iwamoto and M. Yamamoto, "Jumping Motion Control Planning for 4-wheeled Robot with a Tail," in *IEEE/SICE International Symposium on System Integration (SII)*, 2015.

[10] A. Yamashita, T. Arai, J. Ota and H. Asama, "Motion Planning of Multiple Mobile Robots for Cooperative Manipulation and Transportation," in *IEEE Transactions on Robotics and Automation*, 2003.

[11] M. J. B. Krieger, J.-B. Billeter and L. Keller, "Ant-like Task Allocation and Recruitment in Cooperative Robots," *Nature,* vol. 406, no. 6799, pp. 992-995, 2000.

[12] T. Bektas, "The Multiple Traveling Salesman Problem: an Overview of Formulations and Solution Procedures," *Omega,* vol. 34, no. 3, pp. 209-219, 2006.

[13] O. Matei and P. Pop, "An Efficient Genetic Algorithm for Solving the Generalized Traveling Salesman Problem," in *Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing*, 2010.

[14] P. Kitjacharoenchai, M. Ventresca, M. Moshref-Javadi, S. Lee, J. M. A. Tranchoco and P. A. Brunese, "Multiple Traveling Salesman Problem with Drones: Mathematical Model and Heuristic Approach," *ScienceDirect,* vol. 129, pp. 14-30, 2019.

[15] C. Dornhege, A. Kleiner, A. Hertle and A. Kolling, "Multirobot Coverage Search in Three Dimensions," *Journal of Field Robotics,* vol. 33, no. 4, pp. 537-558, 2016.

[16] M. Nazarahari, E. Khanmirza and S. Doostie, "Multi-objective Multi-robot Path Planning in Continuous Environment Using an Enhanced Genetic Algorithm," *ScienceDirect,* vol. 115, pp. 106-120, 2019.

[17] M. Saska, M. Macas, L. Preucil and L. Lhotska, "Robot Path Planning using Particle Swarm Optimization of Ferguson Splines," in *IEEE Conference on Emerging Technologies and Factory Automation*, 2006.

[18] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

[19] S. M. LaValle and J. J. Kuffner Jr., "Randomized Kinodynamic Planning," *The International Journal of Robotics Research,* vol. 20, no. 5, pp. 378-400, 2001.

[20] C. E. Miller, A. W. Tucker and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM,* vol. 7, no. 4, pp. 326-329, 1960.

[21] Parrot, "Parrot jumping sumo-races, slaloms, acrobatics, you can do it all," Parrot, [Online]. Available: https://www.parrot.com/us/minidrones/parrot-jumping-sumo. [Accessed 6 November 2019].

[22] S. Garber, "Sputnik and The Dawn of the Space Age," NASA History Web Curator, 10 October 2007. [Online]. Available: https://history.nasa.gov/sputnik/. [Accessed 7 November 2019].

[23] B. Dunbar, "July 20, 1969: One Giant Leap For Mankind," National Aeronautics and Space Administration, 15 July 2019. [Online]. Available: https://www.nasa.gov/mission_pages/apollo/apollo11.html. [Accessed 7 November 2019].

[24] National Aeronautics and Space Administration, "Mars Science Laboratory Landing," July 2012. [Online]. Available: https://mars.nasa.gov/internal_resources/823/. [Accessed 7 November 2019].

[25] K. C. Tan, I. Shyu, M. Jung, C. Wan and R. Dai, "Motion Planning and Task Allocation for a Jumping Rover Team," in *International Conference on Robotics and Automation*, 2020.

Appendix A. Codes

Simulation MATLAB Codes

Jumping Rovers' Main Heuristic Optimization Methods including RRT, RRT, Refined RRT* and MILP Code*

```
% Jumping Rovers Code
% Develop by Isaac Shyu, Kai Chuen Tan and MyungJin Jung
% Avoid or jump onto objects to reach targets
% Rapidly-exploring Random Tree (RRT) w/ RRT* Option
% Main Loop

clear, clc, close

%% Toggle program options

% Toggle RRT*
RRT_Star = 1;
% Toggle object avoidance
objAvoid = 1;
% Animation pause time
tPause = 0.0;

%% Parameters

% Maximum iterations
maxIt = 100000;
% Display area
xMin = -2;
xMax = 82;
yMin = -2;
yMax = 82;

% Cost weightage
% 1 unit length (equivalent to 50 mm) rolling = 2.5J
% Cost for jumping [Energy/jump = 11 J/jump]
jumpWeight = 4.4;
% Cost of rotation (Energy/degree = 0.0593 J/degree)
rotWeight = 0.0237;

% Max incremental distance between each point in the tree when not optimizing
distInc = 1;
% Max incremental distance between each point in the tree when optimizing
neighborhood = 5;
```

```
% Margin for the display of objects in plot and apply actual vehicles' width
% vehicles' width is about 200mm, consider markers visible distance (50mm
% expected) then, we can set the margin to 10mm(200/2)
objMarg = 2;

% Mixed-Integer Linear Programming (MILP)
%Number of Salesmen
numSalesmen = 2;
%Depot point
depotPt = 1;

%% Example Scenarios
example = 12; % mTSP Scenarios is number 12.
% Output targets, number of targets, properties of the obstacles/objects.
[targets, numTarget, objMatrix] = Senario(example);

%% Initialization

% Object variable initialization
numObj = size(objMatrix,1);
widthHalves = objMatrix(:,3)/2;
lengthHalves = objMatrix(:,4)/2;
xObjMin = objMatrix(:,1) - widthHalves;
xObjMax = objMatrix(:,1) + widthHalves;
yObjMin = objMatrix(:,2) - lengthHalves;
yObjMax = objMatrix(:,2) + lengthHalves;

% Exception for avoiding objects if the targets are on top of objects
avoidException = zeros(1,numObj);
for i = 1:numTarget

    for j = 1:numObj

        if (targets(i,1) > xObjMin(j)) && (targets(i,1) < xObjMax(j)) &&
(targets(i,2) > yObjMin(j)) && (targets(i,2) < yObjMax(j))
            avoidException(1,j) = 1;
        end

    end

end

% Initialize matrix called targetPage to keep track of target numbers,
locations, cost and completion status
% Rows are tree numbers, columns are: start (target) #, end (target) #,
xStart, yStart, xEnd, yEnd,
% jump status, completion status, linked tree
targetOrder = zeros((numTarget-1)^2, 2);
a = 0;
% Brute force method for getting all combinations of target index to target
index with repeating indices
```

```
% (ex. 1 to 2 & 2 to 1, 1 to 3 % 3 to 1)
for j = 1:numTarget

    for k = 1:numTarget

        if k == j
            continue
        else
            a = a + 1;
            % targetOrder() function holds all combinations of target indices
            targetOrder(a,1) = j;
            targetOrder(a,2) = k;
        end

    end

end

numTree = size(targetOrder,1); % Number of initial trees to reach targets
(assuming no objects in the way)

% If there are less than two targets, it will return error
if numTree < 2
    error('Need more than 1 target')
end

% Creating emptyTree properties
emptyTree.Vertices = []; % Holds [x1 y1; x2 y2;...] vertex coordinates
emptyTree.VertexConn = {}; % Holds [vertex #; vertex # vertex #;...] vertices
that it is connected to in the tree
emptyTree.Cost = 0; % Cost placeholder
emptyTree.JumpCost = []; % Holds cost of jumping
emptyTree.JumpedObj = []; % Holds object number that was jumped
emptyTree.RotCost = []; % Holds cost of rotation
emptyTree.AvoidObject = []; % Flag for each index to indicate if the point
was moved to avoid objects
emptyTree.InObject = zeros(1,numObj); % Array to indicate which object the
present vertex is on top of

% Assigning emptyTree properties to all trees
tree = repmat(emptyTree, numTree, 1);

% Assign starting point (target) to each tree
% If destination objective is active, assign each tree to a specific start
point
targetPage = zeros(numTree,9); % Matrix to keep track of all tree
combinations

% Determine all possible start-and-end targets and find out which targets
% are on top of an object/obstacle.
k = 0;
```

```
for i = 1:numTree

    k = k + 1;
    targetPage(i,1:2) = targetOrder(i,1:2); % Start and end target index
numbers
    targetPage(i,3:4) = targets(targetOrder(i,1),:); % Start xy coordinates
    targetPage(i,5:6) = targets(targetOrder(i,2),:); % End xy coordinates
    tree(i).Vertices = targets(targetOrder(i,1),:); % Assigning first xy
coordinate to trees

    for j = 1:numObj % If the tree starts on top of an object, set that
object number as a jumped target

        if (objMatrix(j,1) == targetPage(i,3)) && (objMatrix(j,2) ==
targetPage(i,4))
            tree(i).JumpedObj = [tree(i).JumpedObj j];
            break
        end

    end

end

% If the option to avoid objects is active, create variable to indicate that
path has already run
% into an object to prevent making multiple new trees
if objAvoid == 1
    avoidanceActivated = zeros(1, numObj);
end

% Plotting targets
hold on;
grid minor;
plot(targets(:,1), targets(:,2), 'ob', 'MarkerFaceColor', 'c')
for i = 1:size(targets,1)
    text(targets(i,1), targets(i,2), num2str(i), 'Color', 'k', 'FontSize',
20)
end

% Plotting objects
for i = 1:numObj

    objHeight = objMatrix(i,5);

    % Color of objects changes based upon height (110 mm = green, 220 mm =
black, anything else = red)
    if (objHeight > 3 || i == 1)
        Color = 'k';
    elseif objHeight > 1
        Color = 'g';
    else
```

```
        Color = 'r';
    end

    plot([xObjMin(i) xObjMax(i)], [yObjMin(i) yObjMin(i)], Color,
'LineWidth', 2)
    plot([xObjMax(i) xObjMax(i)], [yObjMin(i) yObjMax(i)], Color,
'LineWidth', 2)
    plot([xObjMax(i) xObjMin(i)], [yObjMax(i) yObjMax(i)], Color,
'LineWidth', 2)
    plot([xObjMin(i) xObjMin(i)], [yObjMax(i) yObjMin(i)], Color,
'LineWidth', 2)

end

% Graph's X-Y limits.
xlim([xMin xMax]);
ylim([yMin yMax]);
axis equal

% Start timer.
tic

%% Iteration Loop

superFlag = 0; % Flag to break out of all RRT* iterations

for i = 1:maxIt

    repeat = 1;
    while repeat == 1

        % DESTINATION OBJECTIVE
OPTION_______________________________________________________________________
_____________________
        % Loop through all combinations of paths
        cntflag = 0;

        for j = 1:numTree

            if targetPage(j,8) ~= 1 % If target is not yet reached, generate
next vertex for path
                xRand = targetPage(j,5); % Target tree x
                yRand = targetPage(j,6); % Target tree y
                cntflag = 1;
                break
            end

        end

        if targetPage(:,8) == 1 % If all targets reached, end program
```

```
            superFlag = 1; % Break out of all iterations
            break
        end

        % Assign tree that has not yet completed it's path to bestTree to
have another point in the tree placed
        curTree = j;
        curVertexIndex = size(tree(curTree).Vertices,1); % Assign the next
connecting vertex

        % Create unit vector point for plot
        xOrigin = tree(curTree).Vertices(curVertexIndex, 1);
        yOrigin = tree(curTree).Vertices(curVertexIndex, 2);

        % Start and End targets must not be the same.
        if (xOrigin == xRand) && (yOrigin == yRand)
            continue
        end

        % Calculate the distance magnitude.
        xVec = xRand - xOrigin;
        yVec = yRand - yOrigin;
        mag = norm([xVec yVec]);

        % If the distance to the target is less than the distance increment,
make the increment shorter so
        % it reaches the target, then mark it as a satisfying objective
        if mag < distInc
            xUnit = xVec;
            yUnit = yVec;
            targetPage(curTree,8) = 1; % Mark tree as completed
            xNew = xOrigin + xUnit; % New x vertex
            yNew = yOrigin + yUnit; % New y vertex
        else % Otherwise, calculate the unit vector normally
            xUnit = round((xVec/mag)*distInc);
            yUnit = round((yVec/mag)*distInc);
            xNew = xOrigin + xUnit; % New x vertex
            yNew = yOrigin + yUnit; % New y vertex
            % If the new point is the target point, mark as destination
reached
            if (xNew == xRand) && (yNew == yRand)
                targetPage(curTree, 8) = 1; % Mark tree as completed
            end
        end

        % If the point is not yet used, assign it to a bank of used points
        repeat = 0;
    end

    if superFlag == 1 % Flag to break out of all iterations
        break
    end
```

```
    % Assign unit vector point to best tree.
    tree(curTree).Vertices = [tree(curTree).Vertices; xNew yNew];
    costArraySize = size(tree(curTree).Vertices,1);
    tree(curTree).Cost(costArraySize,1) = tree(curTree).Cost(curVertexIndex)
+ hypot(xUnit, yUnit);
    tree(curTree).VertexConn{end+1,1} = [];

    hold on;

    % Keeps tally of which vertex is connected to each other vertex
    % Each row of the cell matrix corresponds to the row of the vertex number
    % The array in each cell matrix holds the vertex numbers that uses that
row number vertex as an origin point
    connArraySize = size(tree(curTree).VertexConn{curVertexIndex,1},2);
    tree(curTree).VertexConn{curVertexIndex,1}(connArraySize+1) =...
        size(tree(curTree).Vertices,1);

    % Plot lines to show the shortest distance to each point
    [Color] = defineColor(curTree);
    plot([xOrigin xNew], [yOrigin yNew], 'Color', Color);
    axis([xMin xMax yMin yMax]);
    pause(tPause)

    % If the option to avoid objects is active, determine if path runs into
object and move path if applicable
    if objAvoid == 1

        % AVOIDANCE
FUNCTION_______________________________________________________________
________________________
        % avoid path running into objects
        [treeVerticesNew, newTreeCost, treeAvoidObject, connIndex, insideObj,
targetPageJumpOut, objNum, treeInObj] =...
            objectAvoidance(tree(curTree), objMatrix, targetPage(curTree,:),
distInc, avoidException);
        tree(curTree).AvoidObject = treeAvoidObject;

        % If a point is inside an object, move it outside, and animate
        if insideObj == 1

            if (avoidanceActivated(1, objNum) == 0)

                newTree = numTree + 1; % New tree index
                targetPage(newTree,1:2) = targetPage(curTree,1:2); % New
tree's start and end index (same as original tree)
                targetPage(newTree,3) = xOrigin; % New tree's start x
coordinate (before jump happens)
                targetPage(newTree,4) = yOrigin; % New tree's start y
coordinate (before jump happens)
```

```
                targetPage(newTree,5:6) = targetPage(curTree,5:6); % New
tree's target coordinate (same as original tree)
                targetPage(newTree,7) = 1; % Indicate that newly created tree
was the result of a jump
                targetPage(newTree,9) = curTree; % Tree that this new tree
originated from

                % Transfer vertices from old tree to new tree (including the
vertices inside object)
                tree(newTree) = tree(curTree);

                % Indicate which vertex is on top of an object
                % or obstacle.
                tree(newTree).InObject = treeInObj;


                % JUMP COST
FUNCTION______________________________________________________________________
_________________
                % Calculate cost of new tree jumping onto objects
                [jumpCost, jumpedObj] =
jumpObjectCost(tree(newTree).Vertices, tree(newTree).VertexConn, objMatrix,
jumpWeight);
                tree(newTree).JumpedObj = [tree(newTree).JumpedObj
jumpedObj];
                finalIndex = size(tree(curTree).Vertices,1);
                tree(newTree).JumpCost(finalIndex,1) = jumpCost;

                % Replace color that jumps onto object with newTree color
                [newTreeColor] = defineColor(newTree);
                plot(xOrigin, yOrigin, 'ob') % Blue circle indicates jump
location
                plot(xNew, yNew, 'r*') % Red start indicates landing location
                plot([xOrigin xNew], [yOrigin yNew], 'Color', 'y')

                avoidanceActivated(1, objNum) = 1; % Prevents multiple trees
from being created after initial encounter with object
                numTree = size(targetPage,1); % Increase number of trees to
be completed after creation of new tree

            else

                % If there was already an encounter with an object, erase
plotted line that intrudes into object again
                plot([xOrigin xNew], [yOrigin yNew], 'Color', 'w')

            end

            tree(curTree).Vertices = treeVerticesNew; % New vertices from
objectAvoidance function
```

```
            tree(curTree).Cost = newTreeCost; % New cost from objectAvoidance
function
            pause(tPause)
            xOrigin = tree(curTree).Vertices(connIndex,1); % New start x
coordinate for tree segment
            yOrigin = tree(curTree).Vertices(connIndex,2); % New start y
coordinate for tree segment
            xNew = tree(curTree).Vertices(end,1); % New end x coordinate for
tree segment
            yNew = tree(curTree).Vertices(end,2); % New end x coordinate for
tree segment
            plot([xOrigin xNew], [yOrigin yNew], 'Color', Color); % Tree
color for tree segment
            jumpIndicator = 0; % Indicator for jump if target is on object
            pause(tPause)

        elseif (insideObj == 2) && (jumpIndicator == 0)

            jumpIndicator = 1; % Indicator for jump if target is on object
            plot(xOrigin, yOrigin, 'ob') % Blue circle indicates jump
location
            plot(xNew, yNew, 'r*') % Red start indicates landing location

        else

            % If no encounter with an object, set avoidance to 0 to allow new
tree to be created for next encountered object
            avoidanceActivated = zeros(1, numObj);
            targetPage(curTree,7) = targetPageJumpOut; % Set output for jump
status from objectAvoidance function
            jumpIndicator = 0; % Indicator for jump if target is on object

        end
    end


    %RRT STAR
FUNCTION____________________________________________________________________
____________________________________
    if RRT_Star == 1

        % Rearrange vertex connections to optimize distance cost
        [bestTreeVertexConnNew, minVtxIdx, newTreeCost] =
jumpNeighCost2(tree(curTree), neighborhood, objAvoid);

        % If there is a change in the connections for optimization, change
vertex connections, and animate
        if minVtxIdx ~= 0

            tree(curTree).Cost = newTreeCost;
            tree(curTree).VertexConn{curVertexIndex,1}(end) = [];
```

```
            tree(curTree).VertexConn{minVtxIdx,1} =
[tree(curTree).VertexConn{minVtxIdx,1} size(tree(curTree).Vertices,1)];
            plot([xOrigin xNew], [yOrigin yNew], 'Color', 'w')
            pause(tPause)
            xOrigin = tree(curTree).Vertices(minVtxIdx,1);
            yOrigin = tree(curTree).Vertices(minVtxIdx,2);
            plot([xOrigin xNew], [yOrigin yNew], 'Color', Color);
            pause(tPause)

        end
    end

    % Display the number of iterations
    disp(['Iteration: ' num2str(i)])

end

% Display the total running time
toc

%% Results

% MINIMUM COST
FUNCTION______________________________________________________________________
___________________________________
% Find best cost path for combinations of paths so all targets are visited

% Determine cost of rotation
for i = 1:numTree

    for j = 2:(size(tree(i).Vertices)-1)

        vec1 = tree(i).Vertices(j,:) - tree(i).Vertices(j-1,:);
        vec2 = tree(i).Vertices(j+1,:) - tree(i).Vertices(j,:);
        calcRotCost =
round(acosd(dot(vec1,vec2)/(norm(vec1)*norm(vec2))))*rotWeight;
        tree(i).RotCost = [tree(i).RotCost calcRotCost];

    end

end

% Solve single traveling salesman problem with the Mixed-integer Linear
% Programming (MILP)
[pathOrderMin,x_tsp] = Multiple_TSP_V4(tree, targets,
targetPage,numSalesmen,depotPt);

% Variable for exporting
totalVertices = [];
result = 1;
```

```
if result

    % Plot each path in treeOrder so the best cost path is highlighted in the
figure
    for i = 1:(numTarget+(numSalesmen-1))

        activeTree = pathOrderMin(i);

        % Only plot points that have continuity to the targeted point
        continuousVertices =
[find(~cellfun(@isempty,tree(activeTree).VertexConn));
size(tree(activeTree).Vertices,1)] ;
        treeLength = size(continuousVertices); % Length of continuous points
        tree(activeTree).Vertices =
tree(activeTree).Vertices(continuousVertices, :);
        tree(activeTree).VertexConn =
tree(activeTree).VertexConn(continuousVertices(1:(end-1), :));
        tree(activeTree).Cost = tree(activeTree).Cost(continuousVertices, :);
        tree(activeTree).JumpCost =
tree(activeTree).JumpCost(continuousVertices(continuousVertices <=
size(tree(activeTree).JumpCost,1), :));
        tree(activeTree).AvoidObject =
tree(activeTree).AvoidObject(continuousVertices, :);

        % Add zeros to the end of jump cost to match the number of elements
in vertices
        if size(tree(activeTree).Vertices, 1) >
size(tree(activeTree).JumpCost, 1)

            tree(activeTree).JumpCost(numel(tree(activeTree).Vertices(:,1)))
= 0;

        end

        % Total Vertices of Each Tree.
        totalVertices = [totalVertices; tree(activeTree).Vertices
tree(activeTree).JumpCost];

        for j = 1:(treeLength-1)

            xOrigin = tree(activeTree).Vertices(j,1);
            yOrigin = tree(activeTree).Vertices(j,2);
            xNew = tree(activeTree).Vertices(j+1,1);
            yNew = tree(activeTree).Vertices(j+1,2);
            % Plot each segment
            pb = plot([xOrigin xNew], [yOrigin yNew], 'Color', 'b',
'LineWidth', 2);
            pause(tPause)

        end
```

```
    end

end

% Auto-refine RRT* Toggle
refine = 1;
% Manual refine RRT* Toggle
manualrefine = 1;

% Refine RRT*
if refine

    objDispMarg = 0;
    totalRefVertices = [];

    % Refine best path
    for i = 1:(numTarget+(numSalesmen-1))

        activeTree = pathOrderMin(i);
        treeEval = tree(activeTree);
        [refVert, refCost] = refinedRRT(treeEval, objMatrix, distInc,
objDispMarg, rotWeight);
        totalRefVertices = [totalRefVertices; refVert];

        for k = 1:(size(refVert,1)-1)

            hold on;
            xRefVertOrigin = refVert(k,1); yRefVertOrigin = refVert(k,2);
            xRefVertNew = refVert(k+1,1); yRefVertNew = refVert(k+1,2);
            plot([xRefVertOrigin xRefVertNew], [yRefVertOrigin, yRefVertNew],
'Color', 'm', 'LineWidth', 2)
            pause(tPause)

        end

    end

end

hold off;

%% Manual Refine RRT* Check
if manualrefine

    hold on
    numseltree = length(pathOrderMin);
    straight = [1 1 1 0 0 1];
    jumptree = [0 0 0 1 2 0];
    jumpdist = norm([534.3934 1491.7923] - [556.3348 1221.9086])/50;
```

```
    rollcost = zeros(numseltree,1);
    jumpcost = zeros(numseltree,1);
    rotcost = zeros(numseltree,1);

    for i = 1:numseltree

        rftree(i).Vertices = tree(pathOrderMin(i)).Vertices;
        pb = plot(rftree(i).Vertices(:, 1), rftree(i).Vertices(:,2), 'b--',
'LineWidth', 2);
        hold on

    end


    for i = 1:numseltree

      if (i == 1 || i == 4 || i==6)
          nu = size(rftree(i).Vertices(:,1),1);
          rftree(i).Vertices(:,1) = linspace(rftree(i).Vertices(1,1),
rftree(i).Vertices(end,1), nu);
          rftree(i).Vertices(:,2) = linspace(rftree(i).Vertices(1,2),
rftree(i).Vertices(end,2), nu);
          pp = plot(rftree(i).Vertices(:, 1), rftree(i).Vertices(:,2), 'm',
'LineWidth', 2);
          treelength = norm(rftree(i).Vertices(end, :) -
rftree(i).Vertices(1, :));
          rollcost(i) = treelength - jumptree(i) * jumpdist;
          jumpcost(i) = jumptree(i) * jumpWeight;
      elseif i == 2
          nu = size(rftree(i).Vertices(:,1),1);
          extrpt1 = 29;
          extrpt2 = 39;
          rftree(i).Vertices(1:extrpt1,1) = linspace(rftree(i).Vertices(1,1),
rftree(i).Vertices(extrpt1,1), extrpt1);
          rftree(i).Vertices(1:extrpt1,2) = linspace(rftree(i).Vertices(1,2),
rftree(i).Vertices(extrpt1,2), extrpt1);
          rftree(i).Vertices(extrpt2:end,1) =
linspace(rftree(i).Vertices(extrpt2,1), rftree(i).Vertices(end,1), nu-
extrpt2+1);
          rftree(i).Vertices(extrpt2:end,2) =
linspace(rftree(i).Vertices(extrpt2,2), rftree(i).Vertices(end,2), nu-
extrpt2+1);
          plot(rftree(i).Vertices(:, 1), rftree(i).Vertices(:,2), 'm',
'LineWidth', 2)
          length1 = norm(rftree(i).Vertices(end, :) -
rftree(i).Vertices(extrpt2, :));
          length2 = norm(rftree(i).Vertices(extrpt2, :) -
rftree(i).Vertices(extrpt1, :));
          length3 = norm(rftree(i).Vertices(extrpt1, :) -
rftree(i).Vertices(1, :));
          rollcost(i) = length1 + length2 + length3  - jumptree(i) *
jumpdist;
```

```
            jumpcost(i) = jumptree(i) * jumpWeight;
        elseif (i == 3 )
            nu = size(rftree(i).Vertices(:,1),1);
            extrpt = 37;
            rftree(i).Vertices(1:extrpt,1) = linspace(rftree(i).Vertices(1,1),
rftree(i).Vertices(extrpt,1), extrpt);
            rftree(i).Vertices(1:extrpt,2) = linspace(rftree(i).Vertices(1,2),
rftree(i).Vertices(extrpt,2), extrpt);
            rftree(i).Vertices(extrpt:end,1) =
linspace(rftree(i).Vertices(extrpt,1), rftree(i).Vertices(end,1), nu-
extrpt+1);
            rftree(i).Vertices(extrpt:end,2) =
linspace(rftree(i).Vertices(extrpt,2), rftree(i).Vertices(end,2), nu-
extrpt+1);
            plot(rftree(i).Vertices(:, 1), rftree(i).Vertices(:,2), 'm',
'LineWidth', 2)
            length1 = norm(rftree(i).Vertices(end, :) -
rftree(i).Vertices(extrpt, :));
            length2 = norm(rftree(i).Vertices(extrpt, :) -
rftree(i).Vertices(1, :));
            rollcost(i) = length1 + length2  - jumptree(i) * jumpdist;
            jumpcost(i) = jumptree(i) * jumpWeight;
        elseif i == 5
            nu = size(rftree(i).Vertices(:,1),1);
            extrpt = 22;
            rftree(i).Vertices(1:extrpt,1) = linspace(rftree(i).Vertices(1,1),
rftree(i).Vertices(extrpt,1), extrpt);
            rftree(i).Vertices(1:extrpt,2) = linspace(rftree(i).Vertices(1,2),
rftree(i).Vertices(extrpt,2), extrpt);
            rftree(i).Vertices(extrpt:end,1) =
linspace(rftree(i).Vertices(extrpt,1), rftree(i).Vertices(end,1), nu-
extrpt+1);
            rftree(i).Vertices(extrpt:end,2) =
linspace(rftree(i).Vertices(extrpt,2), rftree(i).Vertices(end,2), nu-
extrpt+1);
            plot(rftree(i).Vertices(:, 1), rftree(i).Vertices(:,2), 'm',
'LineWidth', 2)
            length1 = norm(rftree(i).Vertices(end, :) -
rftree(i).Vertices(extrpt, :));
            length2 = norm(rftree(i).Vertices(extrpt, :) -
rftree(i).Vertices(1, :));
            rollcost(i) = length1 + length2  - jumptree(i) * jumpdist;
            jumpcost(i) = jumptree(i) * jumpWeight;
        end

    end

end

% Calculate Total Cost manually.
totalcost = sum(rollcost+jumpcost)/2.5;
```

```
% Plot Final Results
xticklabels({'0', '500', '1000','1500','2000','2500', '3000','3500','4000'})
yticklabels({'0', '500', '1000','1500','2000','2500', '3000','3500','4000'})
ylabel('Y-Position [mm]')
xlabel('X-Position [mm]')
title({'Navigating Multiple Traveling Salesmen Problem with','Refined
Rapidly-exploring Random Tree with Jumping Option'})
legend([pb pp], {'RRT*','Refined RRT*'}, 'Location', 'NW')
text(6, 60, ['Total Cost:', num2str(totalcost)])
```

Jumping Rover RRT's Object Avoidance Criteria Function*

```
% Develop by Isaac Shyu, Kai Chuen Tan, and MyungJin Jung
% Rapidly-exploring Random Tree* (RRT*) object avoidance criteria function
% Object Avoidance Criteria

function [xNewVec, yNewVec] = avoidCriteria(distInc, treeVertices, objPoints,
xObjMin, xObjMax, yObjMin, yObjMax, objNum, xConnVtx, yConnVtx, xPrevConnVtx,
yPrevConnVtx)

    xVtx = treeVertices(end,1);
    yVtx = treeVertices(end,2);

    % APPROACH FROM
LEFT_________________________________________________________________________
_________________________
    % If approaching from left, divert around in clockwise:
    if (yConnVtx > yObjMin(objNum)) && (yConnVtx < yObjMax(objNum)) &&
(xConnVtx <= xObjMin(objNum))

        xVec = xConnVtx - xPrevConnVtx;
        yVec = yConnVtx - yPrevConnVtx;

        if (yVec > 0) && (yVtx > objPoints(objNum,2))
            xNewVec = xObjMin(objNum) - xConnVtx;
            yNewVec = distInc;
        elseif (yVec < 0) && (yVtx < objPoints(objNum,2))
            xNewVec = xObjMin(objNum) - xConnVtx;
            yNewVec = -distInc;
        elseif (yVec > 0) && (yVtx < objPoints(objNum,2))
            if xVec > yVec
                xNewVec = xObjMin(objNum) - xConnVtx;
                yNewVec = -distInc;
            else
                xNewVec = xObjMin(objNum) - xConnVtx;
                yNewVec = distInc;
            end
        elseif (yVec < 0) && (yVtx > objPoints(objNum,2))
            if xVec > abs(yVec)
                xNewVec = xObjMin(objNum) - xConnVtx;
                yNewVec = distInc;
            else
                xNewVec = xObjMin(objNum) - xConnVtx;
                yNewVec = -distInc;
            end
        else
            xNewVec = xObjMin(objNum) - xConnVtx;
            yNewVec = distInc;
        end
```

```
    % APPROACH FROM
RIGHT_____________________________________________________________________________
__________________________
    % If approaching from right, divert around in clockwise:
    elseif (yConnVtx > yObjMin(objNum)) && (yConnVtx < yObjMax(objNum)) &&
(xConnVtx >= xObjMax(objNum))

        xVec = xConnVtx - xPrevConnVtx;
        yVec = yConnVtx - yPrevConnVtx;

        if (yVec > 0) && (yVtx > objPoints(objNum,2))
            xNewVec = xObjMax(objNum) - xConnVtx;
            yNewVec = distInc;
        elseif (yVec < 0) && (yVtx < objPoints(objNum,2))
            xNewVec = xObjMax(objNum) - xConnVtx;
            yNewVec = -distInc;
        elseif (yVec > 0) && (yVtx < objPoints(objNum,2))
            if abs(xVec) > yVec
                xNewVec = xObjMax(objNum) - xConnVtx;
                yNewVec = -distInc;
            else
                xNewVec = xObjMax(objNum) - xConnVtx;
                yNewVec = distInc;
            end
        elseif (yVec < 0) && (yVtx > objPoints(objNum,2))
            if abs(xVec) > yVec
                xNewVec = xObjMax(objNum) - xConnVtx;
                yNewVec = distInc;
            else
                xNewVec = xObjMax(objNum) - xConnVtx;
                yNewVec = -distInc;
            end
        else
            xNewVec = xObjMax(objNum) - xConnVtx;
            yNewVec = -distInc;
        end

    % APPROACH FROM
BOTTOM____________________________________________________________________________
__________________________
    % If approaching from bottom, divert around in clockwise:
    elseif (xConnVtx > xObjMin(objNum)) && (xConnVtx < xObjMax(objNum)) &&
(yConnVtx <= yObjMin(objNum))

        xVec = xConnVtx - xPrevConnVtx;
        yVec = yConnVtx - yPrevConnVtx;

        if (xVec > 0) && (xVtx > objPoints(objNum,1))
            xNewVec = distInc;
            yNewVec = yObjMin(objNum) - yConnVtx;
        elseif (xVec < 0) && (xVtx < objPoints(objNum,1))
```

```
            xNewVec = -distInc;
            yNewVec = yObjMin(objNum) - yConnVtx;
        elseif (xVec > 0) && (xVtx < objPoints(objNum,1))
            if yVec > xVec
                xNewVec = -distInc;
                yNewVec = yObjMin(objNum) - yConnVtx;
            else
                xNewVec = distInc;
                yNewVec = yObjMin(objNum) - yConnVtx;
            end
        elseif (xVec < 0) && (xVtx > objPoints(objNum,1))
            if yVec > abs(xVec)
                xNewVec = distInc;
                yNewVec = yObjMin(objNum) - yConnVtx;
            else
                xNewVec = -distInc;
                yNewVec = yObjMin(objNum) - yConnVtx;
            end
        else
            xNewVec = -distInc;
            yNewVec = yObjMin(objNum) - yConnVtx;
        end

    % APPROACH FROM
TOP____________________________________________________________________________
________________________
    % If approaching from top, divert around in clockwise:
    elseif (xConnVtx > xObjMin(objNum)) && (xConnVtx < xObjMax(objNum)) &&
(yConnVtx >= yObjMax(objNum))

        xVec = xConnVtx - xPrevConnVtx;
        yVec = yConnVtx - yPrevConnVtx;

        if (xVec > 0) && (xVtx >= objPoints(objNum,1))
            xNewVec = distInc;
            yNewVec = yObjMax(objNum) - yConnVtx;
        elseif (xVec < 0) && (xVtx <= objPoints(objNum,1))
            xNewVec = -distInc;
            yNewVec = yObjMax(objNum) - yConnVtx;
        elseif (xVec > 0) && (xVtx <= objPoints(objNum,1))
            if abs(yVec) > xVec
                xNewVec = -distInc;
                yNewVec = yObjMax(objNum) - yConnVtx;
            else
                xNewVec = distInc;
                yNewVec = yObjMax(objNum) - yConnVtx;
            end
        elseif (xVec < 0) && (xVtx >= objPoints(objNum,1))
            if abs(yVec) > abs(xVec)
                xNewVec = distInc;
                yNewVec = yObjMax(objNum) - yConnVtx;
            else
```

```
                xNewVec = -distInc;
                yNewVec = yObjMax(objNum) - yConnVtx;
            end
        else
            xNewVec = distInc;
            yNewVec = yObjMax(objNum) - yConnVtx;
        end

    % APPROACH FROM BOTTOM
LEFT_________________________________________________________________________
_________________________
    % If approaching from bottom left, divert around in clockwise:
    elseif (xConnVtx <= xObjMin(objNum)) && (yConnVtx <= yObjMin(objNum))

        xVec = xVtx - xConnVtx;
        yVec = yVtx - yConnVtx;
        xPrevVec = xConnVtx - xPrevConnVtx;
        yPrevVec = yConnVtx - yPrevConnVtx;

        if (xPrevVec >= 0) && (yPrevVec >= 0)
            if (xVec > yVec)
                xNewVec = distInc;
                yNewVec = yObjMin(objNum) - yConnVtx;
            elseif (xVec < yVec)
                xNewVec = xObjMin(objNum) - xConnVtx;
                yNewVec = distInc;
            else
                xNewVec = xObjMin(objNum) - xConnVtx;
                yNewVec = distInc;
            end
        elseif (xPrevVec < 0)
            xNewVec = xObjMin(objNum) - xConnVtx;
            yNewVec = distInc;
        elseif (yPrevVec < 0)
            xNewVec = distInc;
            yNewVec = yObjMin(objNum) - yConnVtx;
        else
            xNewVec = distInc;
            yNewVec = yObjMin(objNum) - yConnVtx;
        end

    % APPROACH FROM TOP
LEFT_________________________________________________________________________
_________________________
    % If approaching from top left, divert around in clockwise:
    elseif (xConnVtx <= xObjMin(objNum)) && (yConnVtx >= yObjMax(objNum))

        xVec = xVtx - xConnVtx;
        yVec = yVtx - yConnVtx;
        xPrevVec = xConnVtx - xPrevConnVtx;
        yPrevVec = yConnVtx - yPrevConnVtx;
```

```
        if (xPrevVec >= 0) && (yPrevVec <= 0)
            if (xVec > abs(yVec))
                xNewVec = distInc;
                yNewVec = yObjMax(objNum) - yConnVtx;
            elseif (xVec < abs(yVec))
                xNewVec = xObjMin(objNum) - xConnVtx;
                yNewVec = -distInc;
            else
                xNewVec = distInc;
                yNewVec = yObjMax(objNum) - yConnVtx;
            end
        elseif (xPrevVec < 0)
            xNewVec = xObjMin(objNum) - xConnVtx;
            yNewVec = -distInc;
        elseif (yPrevVec > 0)
            xNewVec = distInc;
            yNewVec = yObjMax(objNum) - yConnVtx;
        else
            xNewVec = distInc;
            yNewVec = yObjMax(objNum) - yConnVtx;
        end

    % APPROACH FROM TOP
RIGHT__________________________________________________________________________
__________________________
    % If approaching from top right, divert around in clockwise:
    elseif (xConnVtx >= xObjMax(objNum)) && (yConnVtx >= yObjMax(objNum))

        xVec = xVtx - xConnVtx;
        yVec = yVtx - yConnVtx;
        xPrevVec = xConnVtx - xPrevConnVtx;
        yPrevVec = yConnVtx - yPrevConnVtx;

        if (xPrevVec <= 0) && (yPrevVec <= 0)
            if (abs(xVec) > abs(yVec))
                xNewVec = -distInc;
                yNewVec = yObjMax(objNum) - yConnVtx;
            elseif (abs(xVec) < abs(yVec))
                xNewVec = xObjMax(objNum) - xConnVtx;
                yNewVec = -distInc;
            else
                xNewVec = xObjMax(objNum) - xConnVtx;
                yNewVec = -distInc;
            end
        elseif (xPrevVec > 0)
            xNewVec = xObjMax(objNum) - xConnVtx;
            yNewVec = -distInc;
        elseif (yPrevVec > 0)
            xNewVec = -distInc;
            yNewVec = yObjMax(objNum) - yConnVtx;
        else
            xNewVec = xObjMax(objNum) - xConnVtx;
```

```
            yNewVec = -distInc;
        end

    % APPROACH FROM BOTTOM
RIGHT___________________________________________________________________________
__________________________
    % If approaching from bottom right, divert around in clockwise:
    elseif (xConnVtx >= xObjMax(objNum)) && (yConnVtx <= yObjMin(objNum))

        xVec = xVtx - xConnVtx;
        yVec = yVtx - yConnVtx;
        xPrevVec = xConnVtx - xPrevConnVtx;
        yPrevVec = yConnVtx - yPrevConnVtx;

        if (xPrevVec <= 0) && (yPrevVec >= 0)
            if (abs(xVec) > yVec)
                xNewVec = -distInc;
                yNewVec = yConnVtx - yObjMin(objNum);
            elseif (abs(xVec) < yVec)
                xNewVec = xObjMax(objNum) - xConnVtx;
                yNewVec = distInc;
            else
                xNewVec = -distInc;
                yNewVec = yObjMin(objNum) - yConnVtx;
            end
        elseif (xPrevVec > 0)
            xNewVec = xObjMax(objNum) - xConnVtx;
            yNewVec = distInc;
        elseif (yPrevVec < 0)
            xNewVec = -distInc;
            yNewVec = yObjMin(objNum) - yConnVtx;
        else
            xNewVec = -distInc;
            yNewVec = yObjMin(objNum) - yConnVtx;
        end

    % IF ALL ELSE
FAILS___________________________________________________________________________
____________
    % If fails, continue to penetrate objects to show errors and bugs.
    else

        xVec = xVtx - xConnVtx;
        yVec = yVtx - yConnVtx;
        xNewVec = xVec;
        yNewVec = yVec;

    end
 end
```

Jumping Rover RRT's Color Options for Each Tree*

```matlab
% Develop by Isaac Shyu
% Rapidly-exploring Random Tree* (RRT*)
% Color Option

% Define color of line plot

% Input:
% -bestTree- a single-digit number indicating tree assignment

% Output:
% -Color- color designation of plot, letter or RGB ratio

function [Color] = defineColor(bestTree)
    if bestTree == 1
        Color = 'r'; %red
    elseif bestTree == 2
        Color = 'g'; %green
    elseif bestTree == 3
        Color = 'b'; %blue
    elseif bestTree == 4
        Color = 'm'; %magenta
    elseif bestTree == 5
        Color = 'k'; %black
    elseif bestTree == 6
        Color = [0.4 0.5 0.7]; %grey/blue
    elseif bestTree == 7
        Color = [1 0.19 1]; %pink
    elseif bestTree == 8
        Color = [1 0.64 0.19]; %orange
    elseif bestTree == 9
        Color = [0.54 0.17 0.06]; %dark red
    elseif bestTree == 10
        Color = [0.09 0.9 0.8]; %aqua
    elseif bestTree == 11
        Color = [0.45 0.1 0.9]; %purple
    elseif bestTree == 12
        Color = [0.45 0.45 1]; %light blue
    elseif bestTree == 13
        Color = [0.48 0.7 0.1]; %yellow green
    elseif bestTree == 14
        Color = [0.04 0.43 0.05]; %forest green
    elseif bestTree == 15
        Color = [1 0.85 1]; %light pink
    elseif bestTree == 16
        Color = [1 0.8 1]; %light purple
    elseif bestTree == 17
        Color = [0.59 0.59 0.59]; %gray
```

```
elseif bestTree == 18
    Color = [0.49 0.33 0.04]; %brown
elseif bestTree == 19
    Color = [0 0.51 0.52]; %dark cyan
elseif bestTree == 20
    Color = [0.51 0 0.53]; %dark pink
elseif bestTree == 21
    Color = 'r'; %red
elseif bestTree == 22
    Color = 'g'; %green
elseif bestTree == 23
    Color = 'b'; %blue
elseif bestTree == 24
    Color = 'm'; %magenta
elseif bestTree == 25
    Color = 'k'; %black
elseif bestTree == 26
    Color = [0.4 0.5 0.7]; %grey/blue
elseif bestTree == 27
    Color = [1 0.19 1]; %pink
elseif bestTree == 28
    Color = [1 0.64 0.19]; %orange
elseif bestTree == 29
    Color = [0.54 0.17 0.06]; %dark red
elseif bestTree == 30
    Color = [0.09 0.9 0.8]; %aqua
elseif bestTree == 31
    Color = [0.45 0.1 0.9]; %purple
elseif bestTree == 32
    Color = [0.45 0.45 1]; %light blue
elseif bestTree == 33
    Color = [0.48 0.7 0.1]; %yellow green
elseif bestTree == 34
    Color = [0.04 0.43 0.05]; %forest green
elseif bestTree == 35
    Color = [1 0.85 1]; %light pink
elseif bestTree == 36
    Color = [1 0.8 1]; %light purple
elseif bestTree == 37
    Color = [0.59 0.59 0.59]; %gray
elseif bestTree == 38
    Color = [0.49 0.33 0.04]; %brown
elseif bestTree == 39
    Color = [0 0.51 0.52]; %dark cyan
elseif bestTree == 40
    Color = [0.51 0 0.53]; %dark pink
elseif bestTree == 41
    Color = 'r'; %red
elseif bestTree == 42
    Color = 'g'; %green
elseif bestTree == 43
    Color = 'b'; %blue
elseif bestTree == 44
```

```
    Color = 'm'; %magenta
elseif bestTree == 45
    Color = 'k'; %black
elseif bestTree == 46
    Color = [0.4 0.5 0.7]; %grey/blue
elseif bestTree == 47
    Color = [1 0.19 1]; %pink
elseif bestTree == 48
    Color = [1 0.64 0.19]; %orange
elseif bestTree == 49
    Color = [0.54 0.17 0.06]; %dark red
elseif bestTree == 50
    Color = [0.09 0.9 0.8]; %aqua
elseif bestTree == 51
    Color = [0.45 0.1 0.9]; %purple
elseif bestTree == 52
    Color = [0.45 0.45 1]; %light blue
elseif bestTree == 53
    Color = [0.48 0.7 0.1]; %yellow green
elseif bestTree == 54
    Color = [0.04 0.43 0.05]; %forest green
elseif bestTree == 55
    Color = [1 0.85 1]; %light pink
elseif bestTree == 56
    Color = [1 0.8 1]; %light purple
elseif bestTree == 57
    Color = [0.59 0.59 0.59]; %gray
elseif bestTree == 58
    Color = [0.49 0.33 0.04]; %brown
elseif bestTree == 59
    Color = [0 0.51 0.52]; %dark cyan
elseif bestTree == 60
    Color = [0.51 0 0.53]; %dark pink
elseif bestTree == 61
    Color = 'r'; %red
elseif bestTree == 62
    Color = 'g'; %green
elseif bestTree == 63
    Color = 'b'; %blue
elseif bestTree == 64
    Color = 'm'; %magenta
elseif bestTree == 65
    Color = 'k'; %black
elseif bestTree == 66
    Color = [0.4 0.5 0.7]; %grey/blue
elseif bestTree == 67
    Color = [1 0.19 1]; %pink
elseif bestTree == 68
    Color = [1 0.64 0.19]; %orange
elseif bestTree == 69
    Color = [0.54 0.17 0.06]; %dark red
elseif bestTree == 70
    Color = [0.09 0.9 0.8]; %aqua
```

```
    elseif bestTree == 71
        Color = [0.45 0.1 0.9]; %purple
    elseif bestTree == 72
        Color = [0.45 0.45 1]; %light blue
    elseif bestTree == 73
        Color = [0.48 0.7 0.1]; %yellow green
    elseif bestTree == 74
        Color = [0.04 0.43 0.05]; %forest green
    elseif bestTree == 75
        Color = [1 0.85 1]; %light pink
    elseif bestTree == 76
        Color = [1 0.8 1]; %light purple
    elseif bestTree == 77
        Color = [0.59 0.59 0.59]; %gray
    elseif bestTree == 78
        Color = [0.49 0.33 0.04]; %brown
    elseif bestTree == 79
        Color = [0 0.51 0.52]; %dark cyan
    elseif bestTree == 80
        Color = [0.51 0 0.53]; %dark pink
    end
end
```

Jumping Rover RRT's Neighborhood Cost Function*

```
% Develop by Isaac Shyu
% Rapidly-exploring Random Tree* (RRT*)
% Neighborhood Cost Function (Version 2)

% Finds the least cost connection for a new vertex in a neighborhood of
vertices
% This version looks at cost potential in chains throughout the defined
neighborhood radius

% Inputs are:
% -tree- the tree of interest
% -neighborhood- radius around the new point that creates a circle where
vertices inside the circle can connect to the new vertex



% Output is:
% -vertexConnNew- new cell array for connection indices
% -minVtxIdx- the vertex index that the new point is connected to (will
return 0 if no change in vertex connections)

function [vertexConnNew, minVtxIdx, newTreeCost] = jumpNeighCost2(tree,
neighborhood, objAvoid)

    %% Initializing Neighborhood Points
    treeVertices = tree.Vertices; % Vertices of the tree in which it was
placed
    treeCost = tree.Cost; % Array of each vertex for the cumulative cost to
travel to each vertex from the origin
    treeVertexConn = tree.VertexConn; % Cell array of vertices that a vertex
is an origin point for
    treeAvoidObject = tree.AvoidObject; % Indicator for the object avoidance
option

    newVertex = treeVertices(end,:); % Coordinates of the newest vertex
placed
    newIdx = size(treeVertices,1); % Index of the newest vertex placed

    % If the object was moved to prevent running into something and has more
than 2 vertices find connecting vertices
    if (objAvoid == 1) && (size(treeVertices,1) > 2)

        % Find the connecting vertex index to the new index vertex
        flag = 0;
        for i = 1:size(treeVertexConn,1)
            for j = 1:size(treeVertexConn{i},2)
                if treeVertexConn{i}(j) == newIdx
```

```
                    flag = 1;
                    connIdx = i;
                    break
                end
            end
            if flag == 1
                break
            end
        end
        connAvoidObject = treeAvoidObject(connIdx);

        % Find the previous connecting vertex index to the connecting index
vertex
        flag = 0;
        for i = 1:size(treeVertexConn,1)
            for j = 1:size(treeVertexConn{i},2)
                if treeVertexConn{i}(j) == connIdx
                    flag = 1;
                    prevConnIdx = i;
                    break
                end
            end
            if flag == 1
                break
            end
        end
        prevConnAvoidObject = treeAvoidObject(prevConnIdx);
    end

    % Distance for every point in the tree to the newest vertex placed
    vertexDistances = sqrt((treeVertices(:,1)-newVertex(1,1)).^2 +
(treeVertices(:,2)-newVertex(1,2)).^2);
    vertexDistances(end,:) = []; %get rid of the newest vertex distance
(since it's 0)

    % Find the index of the vertices are in the neighborhood radius
    indicesInNeighborhood = find(vertexDistances(:,1) < neighborhood);
    indicesInNeighborhood(:,2) = 0;

    % If there is only one other vertex in the vicinity (or none cause the
neighborhood is too small or the
    % one the new vertex is already connected to) skip rest of function
    if size(indicesInNeighborhood,1) <= 1
        vertexConnNew = treeVertexConn;
        minVtxIdx = 0;
        newTreeCost = treeCost;

    % If one of the two previous connected points were moved to avoid a
object, don't change the
    % connection to prevent creating a path that passes through an object
    elseif (connAvoidObject == 1) || (prevConnAvoidObject == 1)
        vertexConnNew = treeVertexConn;
```

```
        minVtxIdx = 0;
        newTreeCost = treeCost;
    else

        %% Main Loop to Find Neighborhood Connections

        % For each index in "indicesInNeighborhood", find the total cost from
the origin point
        possibleCost = zeros(size(indicesInNeighborhood,1),2);

        for i = 1:size(indicesInNeighborhood,1)
            refIndex = indicesInNeighborhood(i);
            possibleCost(i,:) = [refIndex
(treeCost(refIndex)+vertexDistances(refIndex))];
        end


        %% Assigning New Vertex Connections
        if range(possibleCost(:,2)) == 0 % If all the costs are the same,
keep the connections the same
            vertexConnNew = treeVertexConn;
            minVtxIdx = 0;
            newTreeCost = treeCost;
        else
            [~, minNeighVtxIdx] = min(possibleCost(:,2)); % Find minimum
distance from all unlinked vertices in the neighborhood
            minVtxIdx = indicesInNeighborhood(minNeighVtxIdx,1); % Pull
minimum vertex index

            % Remove new vertex index from existing vertex connection
            vertexConnNew = cellfun(@(x) x(x~=newIdx), treeVertexConn,
'UniformOutput', 0);
            vertexConnNew{minVtxIdx} = [vertexConnNew{minVtxIdx} newIdx]; %
Add new vertex index to the minimum distance vertex
            newTreeCost = treeCost;
            newTreeCost(newIdx,1) = possibleCost(minNeighVtxIdx,2);
        end
    end
end
```

Jumping Rover RRT's Object Jump Cost Function*

```
% Develop by Isaac Shyu
% Rapidly-exploring Random Tree* (RRT*)
% Object Jump Cost

% Determines if next vertex is on top of an object and whether it had jumped
there

% Inputs:
% -treeVertices- n x 2 matrix that contains the coordinates for each vertex
indicated by the index number
% -vertexConn- cell array that holds arrays containing the vertex numbers
that the cell index number is connected to
% -objectMatrix- matrix that contains the object dimensions (center
coordinates, length, width, height)
% -jumpWeight- weight for the cost of jumping

% Output:
% -jumpCost- the cost of jumping onto an object
% -jumpedObj- the array containing which objects were jumped over

function [jumpCost, jumpedObj] = jumpObjectCost(treeVertices, vertexConn,
objMatrix, jumpWeight)

    %% Initialize objects

    numObject = size(objMatrix,1);
    jumpedObj = [];

    % Object limits extracted
    objHeight = objMatrix(:,5);
    widthHalves = objMatrix(:,3)/2;
    lengthHalves = objMatrix(:,4)/2;
    xObjMin = objMatrix(:,1) - widthHalves;
    xObjMax = objMatrix(:,1) + widthHalves;
    yObjMin = objMatrix(:,2) - lengthHalves;
    yObjMax = objMatrix(:,2) + lengthHalves;
    xObjMax(3) = xObjMax(3) + 2;
    xObjMin(6) = xObjMin(6) - 2;
    yObjMax(5) = yObjMax(5) + 2;
    yObjMin(6) = yObjMin(6) - 2;
    xObjMin(2) = xObjMin(2) -1.5;
    yObjMax(2) = yObjMax(2) +1.5;
    xObjMin(1) = xObjMin(1) -1;
    xObjMax(1) = xObjMax(1) +1;
    yObjMax(1) = yObjMax(1) +1;
    yObjMin(1) = yObjMin(1) -1;
```

```
    %% Query Point Analysis

    queryIndex  = size(treeVertices,1); % Index of the last vertex plotted
    xQueryVtx = treeVertices(queryIndex,1); % x value of last vertex plotted
    yQueryVtx = treeVertices(queryIndex,2); % y value of last vertex plotted

    % Determine if last plotted vertex is inside object limits
    queryInsideX_min = (xQueryVtx > xObjMin);
    queryInsideX_max = (xQueryVtx < xObjMax);
    queryInsideY_min = (yQueryVtx > yObjMin);
    queryInsideY_max = (yQueryVtx < yObjMax);

    % Determine if last plotted vertex is inside an object
    for i = 1:numObject
        if (queryInsideX_min(i) == 1) && (queryInsideX_max(i) == 1) &&
(queryInsideY_min(i) == 1) && (queryInsideY_max(i) == 1)
            queryVtxHeight = objHeight(i);
            jumpedObj = [jumpedObj, i];
            break
        else
            queryVtxHeight = 0;
        end
    end

    %% Connecting Vertex Analysis

    flag = 0;

    % Finding vertex that connects to last vertex plotted
    for i = 1:size(vertexConn,1)
        for j = 1:size(vertexConn{i},2)
            if vertexConn{i}(j) == queryIndex
                flag = 1;
                connIdx = i;
                break
            end
        end
        if flag == 1
            break
        end
    end

    xConnVtx = treeVertices(connIdx,1); % x value of vertex connecting to
last vertex plotted
    yConnVtx = treeVertices(connIdx,2); % y value of vertex connecting to
last vertex plotted

    % Determine if vertex connecting to last plotted vertex is inside object
limits
    connInsideX_min = (xConnVtx > xObjMin);
    connInsideX_max = (xConnVtx < xObjMax);
```

```
    connInsideY_min = (yConnVtx > yObjMin);
    connInsideY_max = (yConnVtx < yObjMax);

    % Determine if vertex connecting to last plotted vertex is inside an
object
    for i = 1:numObject
        if (connInsideX_min(i) == 1) && (connInsideX_max(i) == 1) &&
(connInsideY_min(i) == 1) && (connInsideY_max(i) == 1)
            connVtxHeight = objHeight(i);
            break
        else
            connVtxHeight = 0;
        end
    end

    %% Determine Jump Costs
    % If there is a net gain in height, calculate the resulting cost from the
jump weight; otherwise, the cost is 0
    if queryVtxHeight > connVtxHeight
        heightDiff = queryVtxHeight - connVtxHeight;
        jumpCost = heightDiff * jumpWeight;
    else
        jumpCost = 0;
    end

end
```

Jumping Rovers' mTSP with MILP Function

```
% Develop by Kai Chuen Tan
% Contributed by Changhuang Wan
% Mixed-integer Linear Programming (MILP)
% Multiple Traveling Salesman Problem (mTSP)

% Determines the route to execute for each rover.

function [pathOrderMin, x_tsp] = Multiple_TSP_V4(tree, targets, targetPage,
numSalesmen, depotPt)

% Applied Mixed Linear Integer Programming (MILP)
% to solve multiple Traveling Salesman Problem (mTSP)
% by using "intlinprog."

% Initialization.
%----------------
% Initial Checkpoint (CP) for the "targetPage" input column number.
initialCP = 1;
% Final Checkpoint (CP) for the "targetPage" input column number.
finalCP = 2;
% Size of the Row for the "targetPage."
sizeRow = 1;
% Minimum Cost ID Column for the "minCostIDnPath" array.
ID_column = 1;
% Minimum Amount of Cost Column for the "minCostIDnPath" array.
cost_column = 2;



% Extract data from the inputs.
%-------------------------------
% Number of checkpoints.
numCPs = size(targets,sizeRow);
% Number of paths/trees between two checkpoints.
numPaths = size(tree,sizeRow);
% Different pairs of checkpoints with replacement.
rowNum = 1;
for i = 1:numCPs
    for j = 1:numCPs
        if i == j
            continue
        else
            PairsCPs(rowNum,1:2) = [i j];
            rowNum = rowNum + 1;
        end
    end
end
% Number of different pairs of checkpoints without replacement (nCk).
lenPairCPs = length(PairsCPs);
```

```matlab
% A list of different combinations of a pair of checkpoints.
combinationCPs = targetPage(:,initialCP:finalCP);
% An array that stores the cost for each path.
cost = zeros(numPaths,sizeRow);
% An array that determines and stores
% the minimum cost for each pair of checkpoints with its ID.
minCostIDnPath = zeros(lenPairCPs,2);


% Multiple Traveling Salesman Problem (mTSP) Solving Process.
%----------------------------------------------------------

% Stage 1 Process: Re-order the combination of checkpoints order
%                  to be ascending for all possible paths/trees.
%-----------------------------------------------------------------
for i = 1:size(targetPage,sizeRow)

    % If there is a jump cost and the jump cost is not included in
    % the jump cost tree property
    if ~isempty(tree(i).JumpCost)

        % Add the jump cost to the total cost for the path/tree.
        cost(i) = tree(i).Cost(end) + sum(tree(i).JumpCost);

    else

        % or else, let the total cost of each path/tree
        % to be just the total travel cost of each path/tree.
        cost(i) = tree(i).Cost(end);

    end

end

% Stage 2 Process: Determine the minimum cost for
%                  each checkpoints combination.
%-------------------------------------------------

% For instance, there are a combination of Checkpoint 1 and Checkpoint 2
% and a combination of Checkpoint 2 and 1 (Different CP 1 to CP 2 path);
% the combination with the least cost will be selected.

for i = 1:size(PairsCPs,sizeRow)

    % Identity of the checkpoint the path/tree will start from.
    begin = PairsCPs(i,initialCP);
    % Identity of the checkpoint the path/tree will finish at.
    finish = PairsCPs(i,finalCP);

    % Find paths/trees that contain the same initial checkpoint.
```

```
    % Index (Column by column; from top to bottom).
    pathStart = find(combinationCPs(:,initialCP) == begin);

    % Find paths/trees within pathStart that contain the end checkpoint.
    pathStartFinish = find(combinationCPs(pathStart,finalCP) == finish);

    % Determine paths/trees that contain
    % both the start and finish checkpoints.
    idPath = pathStart(pathStartFinish);

    % Determine the path/tree that has
    % the minimum cost within idPath (index).
    [~,minCostIdPath] = min(cost(idPath));

    % Determine the path/tree that has the minimum cost for the given start
    % and finish checkpoint combo.
    % Store tree index for the given start and finish combo and
    % respective path/tree index cost.
    minCostIDnPath(i,ID_column) = idPath(minCostIdPath);
    minCostIDnPath(i,cost_column) = cost(minCostIDnPath(i,ID_column));

end

minCostPath = minCostIDnPath(:,cost_column);
minCostPath = [minCostPath; zeros(numCPs-1,1)]'; % minCostPath  size should
be PairsCPs+numCPs-1 by 1

% Stage 3 Process: MATLAB TSP: Solver-Based Methods
%---------------------------------------------------
% Sub-stage 1: Equality Constraints
%*********************************
% First type of equality: [Aeq] * [x_tsp] = [beq].
% The first constraint enforces that all checkpoints must be visited once.
% Aeq Properties:
% 1.) Size: 1 x number of different combinations of checkpoints(c).
% 2.) [1_1, 1_2, 1_3,..., 1_c-2, 1_c-1, 1_c]
%Aeq = spones(1:length(PairsCPs));
% beq Properties:
% 1.) Size: 1 x 1 (Scalar)
% 2.) [Total Number of Checkpoints]
%beq = numCPs+(numSalesmen-1);

% Second type of equality:
% The second constraint enforces that there must be two paths/trees
% are attached to a checkpoint.
% Illustration Example:
%           (Path 1)    (Checkpoint 1)    (Path 2)
% ----------------------------X----------------------------

% Initialize how many equalities equation and how many x_tsp together with
% u_i
```

```
% Row
% 2 equalities for deport and return
% 2 * number of targets for non starting points (entry and exit)
% Column
% number of combinations with replacements and number of u_i's.
%%
Aeq = zeros(2*numCPs,length(PairsCPs)+numCPs-1);
beq = zeros(2*numCPs,1);

%%
for CPnum = 1: numCPs

    % Find the paths/trees that include a specific checkpoint.
    % "whichPath" is a logical array ( 1 or 0 only).
    whichPath = (PairsCPs(:,1) == CPnum);
    % Include paths/trees where a specific checkpoint is at either end.
    % Sum of column 1 and column 2 in logical term (1 or 0)
    % Include in the constraint matrix.
    Aeq(CPnum,1:length(PairsCPs)) = whichPath';
    Aeq(CPnum, length(PairsCPs)+1:length(PairsCPs)+numCPs-1) =
zeros(1,numCPs-1); % For u_i

end

% "beq" is a (number of checkpoints+1) x 1 matrix
% because there are 2 equality constraints.
% beq vector is the number of paths/trees.
for CPnum = 1 : numCPs

    if CPnum == depotPt

        beq(CPnum) = numSalesmen;

    else

        beq(CPnum) = 1;

    end
end

for CPnum = 1: numCPs

    % Find the paths/trees that include a specific checkpoint.
    % "whichPath" is a logical array ( 1 or 0 only).
    whichPath = (PairsCPs(:,2) == CPnum);
    % Sum of column 1 and column 2 in logical term (1 or 0)
    % Include in the constraint matrix.
    Aeq(numCPs+CPnum,1:length(PairsCPs)) = whichPath';
    Aeq(numCPs+CPnum, length(PairsCPs)+1:length(PairsCPs)+numCPs-1) =
zeros(1,numCPs-1); % For u_i
```

```
end

% "beq" is a (number of checkpoints+1) x 1 matrix
% because there are 2 equality constraints.
% beq vector is the number of paths/trees.
for CPnum = 1 : numCPs

    if CPnum == depotPt

        %beq(CPnum+1) = 2*numSalesmen;
        beq(numCPs+CPnum) = numSalesmen;

    else

        %beq(CPnum+1) = 2;
        beq(numCPs+CPnum) = 1;

    end
end

% Third type of equality:
% The third constraint enforces that all rovers must start and end at
% a specific visiting point.

A = zeros((numCPs - 1)*(numCPs-2),lenPairCPs+numCPs-1);
b = zeros((numCPs-1)*(numCPs-2),1);
Ineqnum =0;

for i=2:numCPs
    for j=2:numCPs
       if i~= j
          Ineqnum =  Ineqnum+1;
          A(Ineqnum,1:lenPairCPs+numCPs-1) = zeros(1,lenPairCPs+numCPs-1);
          A(Ineqnum,lenPairCPs+i-1) = 1; % ui
          A(Ineqnum,lenPairCPs+j-1) = -1; % uj
          if i<j
              A(Ineqnum,(numCPs-1)*(i-1)+j-1) = numCPs-numSalesmen+1; % xij
          else
              A(Ineqnum,(numCPs-1)*(i-1)+j) = numCPs-numSalesmen+1; % xij
          end
          b(Ineqnum) = numCPs-numSalesmen;
       end

    end
end


% Sub-stage 2: Binary Bounds for the x_tsp
%*******************************************
% Number of decision variables (x_ij) or Index of x_tsp.
xtspIndex = 1:(lenPairCPs+numCPs-1);
```

```
% Lower bound for the x_ij is zero.
lower_Bound = zeros(lenPairCPs+numCPs-1,1);
% Lower bound for the u_i and u_j is 1.
lower_Bound(lenPairCPs+1:end) = ones(1,numCPs-1);
% Upper bound for the x_ij is one.
upper_Bound = ones(lenPairCPs+numCPs-1,1);
% Upper bound for the u_i and u_j is the maximum number of targets
upper_Bound(lenPairCPs+1:end) = numCPs*ones(1,numCPs-1);

% Sub-stage 3: Optimizing the solution of x_tsp
%              with MILP using "intlinprog"
%**********************************************
MILP_Settings = ...
    optimoptions('intlinprog','Display','iter','Heuristics','advanced');
[x_tsp] = ...
    intlinprog(minCostPath, xtspIndex, A, b, Aeq, beq,...
               lower_Bound, upper_Bound, MILP_Settings);

% Sub-stage 4: Constraint the number of subtours
%              by adding an inequality constraint
%              and optimize the solution of x_tsp again
%              with MILP using "intlinprog" and
%*****************************************************
% Determine which index of the x_tsp vector has elements of one.
x_tsp = round(x_tsp);

minCostIDnPathIndex = find(x_tsp == 1);

for i = 1:length(minCostIDnPathIndex)

    if minCostIDnPathIndex(i) > lenPairCPs
        break
    end
    realMinCostIDnPathIndex(i) = minCostIDnPathIndex(i);
end

% Determine each path order.
pathOrderMin = minCostIDnPath(realMinCostIDnPathIndex', ID_column);

end
```

Jumping Rover RRT's Object Avoidance Function*

```
% Develop by Isaac Shyu
% Rapidly-exploring Random Tree* (RRT*)
% Object Avoidance

function [treeVerticesNew, newTreeCost, treeAvoidObject, connIdx, insideObj,
targetPageJumpOut, objNum, treeInObj] =...
    objectAvoidance(tree, objMatrix, targetPage, distInc, avoidException)
    %% Initialize objects
    treeVertices = tree.Vertices;
    treeCost = tree.Cost;
    treeVertexConn = tree.VertexConn;
    treeAvoidObject = tree.AvoidObject;
    treeInObj = tree.InObject;

    %Number of objects in test area
    numObj = size(objMatrix,1);

    %object limits extracted
    widthHalves = objMatrix(:,3)/2;
    lengthHalves = objMatrix(:,4)/2;

    %% Consider vehicles's width and marker distance
    xObjMin = objMatrix(:,1) - widthHalves;
    xObjMax = objMatrix(:,1) + widthHalves;
    yObjMin = objMatrix(:,2) - lengthHalves;
    yObjMax = objMatrix(:,2) + lengthHalves;
    xObjMax(3) = xObjMax(3) + 2;
    xObjMin(6) = xObjMin(6) - 2;
    yObjMax(5) = yObjMax(5) + 2;
    yObjMin(6) = yObjMin(6) - 2;
    xObjMin(2) = xObjMin(2) -1.5;
    yObjMax(2) = yObjMax(2) +1.5;
    xObjMin(1) = xObjMin(1) -1;
    xObjMax(1) = xObjMax(1) +1;
    yObjMax(1) = yObjMax(1) +1;
    yObjMin(1) = yObjMin(1) -1;
    %object points extracted
    objPoints = [objMatrix(:,1) objMatrix(:,2)];

    %% Query Point Analysis

    queryIndex  = size(treeVertices,1); %index of the last vertex plotted
    xQueryVtx = treeVertices(queryIndex,1); %x value of last vertex plotted
    yQueryVtx = treeVertices(queryIndex,2); %y value of last vertex plotted
    treeAvoidObject(queryIndex,1) = 0; %indicator for if point needs to be
moved to avoid object
    xStart = targetPage(3);
```

```
    yStart = targetPage(4);
    xEnd = targetPage(5);
    yEnd = targetPage(6);

    %determine if last plotted vertex is inside object limits
    queryVtxInsideX_min = xQueryVtx > xObjMin;
    queryVtxInsideX_max = xQueryVtx < xObjMax;
    queryVtxInsideY_min = yQueryVtx > yObjMin;
    queryVtxInsideY_max = yQueryVtx < yObjMax;

    %determine if tree start is inside object limits
    queryStartInsideX_min = xStart > xObjMin;
    queryStartInsideX_max = xStart < xObjMax;
    queryStartInsideY_min = yStart > yObjMin;
    queryStartInsideY_max = yStart < yObjMax;

    %determine if tree target is inside object limits
    queryTargetInsideX_min = xEnd > xObjMin;
    queryTargetInsideX_max = xEnd < xObjMax;
    queryTargetInsideY_min = yEnd > yObjMin;
    queryTargetInsideY_max = yEnd < yObjMax;

    %determine if last plotted vertex is inside an object
    insideObj =  0;
    objNum = NaN;
    for i = 1:numObj
        if (queryVtxInsideX_min(i) == 1) && (queryVtxInsideX_max(i) == 1) &&
(queryVtxInsideY_min(i) == 1) && (queryVtxInsideY_max(i) == 1)
            insideObj = 1;
            objNum = i;
            break
        end
    end
    %     objNum = i;
    if(~isnan(objNum))
        %determine if the tree start is on top of the object
        if (queryStartInsideX_min(objNum) == 1) &&
(queryStartInsideX_max(objNum) == 1) &&...
                (queryStartInsideY_min(objNum) == 1) &&
(queryStartInsideY_max(objNum) == 1)
            startOnObj = 1;
        else
            startOnObj = 0;
        end

        %determine if the tree target is on top of the object
        if (queryTargetInsideX_min(objNum) == 1) &&
(queryTargetInsideX_max(objNum) == 1) &&...
                (queryTargetInsideY_min(objNum) == 1) &&
(queryTargetInsideY_max(objNum) == 1)
            targetOnObj = 1;
        else
```

```
            targetOnObj = 0;
        end

        %determine if target is on top of object
        if (avoidException(1,objNum) == 1) && (targetOnObj == 1)
            targetObjException = 1;
        elseif (avoidException(1,objNum) == 1) && (startOnObj == 1)
            targetObjException = 1;
        else
            targetObjException = 0;
        end
    end
    %% Move Newest Vertex Outside Object, if Applicable

    if (insideObj == 0) %if not inside the object, the vertex coordinates
remain the same
        treeVerticesNew = treeVertices;
        newTreeCost = treeCost;
        connIdx = 0;
        targetPageJumpOut = 0;
    %if inside an object, but the vertex is already on top of the object or ,
output not inside object
    elseif (insideObj == 1) && (treeInObj(1,objNum) == 1)
        treeVerticesNew = treeVertices;
        newTreeCost = treeCost;
        connIdx = 0;
        insideObj = 0;
        targetPageJumpOut = 1;
    %if the target is on top of the object, output inside object = 2
    elseif (targetObjException == 1) && (targetOnObj == 1)
        treeVerticesNew = treeVertices;
        newTreeCost = treeCost;
        connIdx = 0;
        insideObj = 2;
        targetPageJumpOut = 0;
    %if the start vertex is on top of the object, output inside object = 3
    elseif (targetObjException == 1) && (startOnObj == 1)
        treeVerticesNew = treeVertices;
        newTreeCost = treeCost;
        connIdx = 0;
        insideObj = 3;
        targetPageJumpOut = 0;
    else %avoidance by tree of object needed
        treeInObj = zeros(1, numObj);
        treeInObj(1,objNum) = 1;
        %identify the vertex that the newest vertex is connected to
        flag = 0;
        for j = 1:size(treeVertexConn,1)
            for k = 1:size(treeVertexConn{j},2)
                if treeVertexConn{j}(k) == queryIndex
                    flag = 1;
                    connIdx = j;
                    break
```

```
                end
            end
            if flag == 1
                break
            end
        end
        xConnVtx = treeVertices(connIdx,1);
        yConnVtx = treeVertices(connIdx,2);

        %identify the vertex that the connecting vertex is connected to
        flag = 0;

        %if the number of vertices is less than 2, make the previous
connecting vector 0
        if size(treeVertexConn,1) < 2
            xPrevConnVtx = 0;
            yPrevConnVtx = 0;
        else %otherwise, find the next  previous connecting vertex connecting
to the connecting vertex
            for j = 1:size(treeVertexConn,1)
                for k = 1:size(treeVertexConn{j},2)
                    if treeVertexConn{j}(k) == connIdx
                        flag = 1;
                        prevConnIndex = j;
                        break
                    end
                end
                if flag == 1
                    break
                end
            end
            xPrevConnVtx = treeVertices(prevConnIndex,1);
            yPrevConnVtx = treeVertices(prevConnIndex,2);
        end

        %function to determine new vector direction
        [xNewVec, yNewVec] = avoidCriteria(distInc, treeVertices, objPoints,
xObjMin, xObjMax, yObjMin, yObjMax, objNum,...
            xConnVtx, yConnVtx, xPrevConnVtx, yPrevConnVtx);

        %unit vector calculation
        mag = norm([xNewVec yNewVec]);
        xUnit = round((xNewVec/mag)*distInc);
        yUnit = round((yNewVec/mag)*distInc);
        xNewVtx = xConnVtx + xUnit;
        yNewVtx = yConnVtx + yUnit;

        %new vertex coordinates for query vertex
        treeVerticesNew = treeVertices;
        treeVerticesNew(queryIndex,1) = xNewVtx;
        treeVerticesNew(queryIndex,2) = yNewVtx;
```

```
        %flag to indicate that the point was moved to avoid object (1 =
needed to be moved)
        treeAvoidObject(queryIndex,1) = 1;

        %new cost value for changing coordinates
        newTreeCost = treeCost;
        newTreeCost(queryIndex,1) = newTreeCost(connIdx,1) + hypot(xConnVtx-
xNewVtx, yConnVtx-yNewVtx);
        targetPageJumpOut = 0; %indicate point is not a result of jumping
    end
end
```

Jumping Rover's Refined RRT Function*

```
% Develop by Isaac Shyu
% Rapidly-exploring Random Tree* (RRT*)
% Refined RRT*

% Refines tree path to make it smoother and optimized

% Inputs:
% -tree- path to be refined
% -objectMatrix- matrix that contains the object dimensions (center
coordinates, length, width, height)
% -distInc- the incremental distance in which the vertices in each path are
evaluated
% -objDispMarg- margin for object display (discrepancy between actual border
of object and displayed border on plot)

% Output:
% -refVert- refined vertices for given tree
% -refCost- corresponding cost for given refined vertices


function [refVert, refCost] = refinedRRT(tree, objMatrix, distInc,
objDispMarg, rotWeight)

jumpedObj = tree.JumpedObj;
numObj = size(objMatrix,1);
widthHalves = objMatrix(:,3)/2;
lengthHalves = objMatrix(:,4)/2;
xObjMin = objMatrix(:,1) - widthHalves - objDispMarg;
xObjMax = objMatrix(:,1) + widthHalves + objDispMarg;
yObjMin = objMatrix(:,2) - lengthHalves - objDispMarg;
yObjMax = objMatrix(:,2) + lengthHalves + objDispMarg;
    xObjMax(3) = xObjMax(3) + 2;
    xObjMin(6) = xObjMin(6) - 2;
    yObjMax(5) = yObjMax(5) + 2;
    yObjMin(6) = yObjMin(6) - 2;
    xObjMin(2) = xObjMin(2) -1.5;
    yObjMax(2) = yObjMax(2) +1.5;
    xObjMin(1) = xObjMin(1) -1;
    xObjMax(1) = xObjMax(1) +1;
    yObjMax(1) = yObjMax(1) +1;
    yObjMin(1) = yObjMin(1) -1;
refVert = tree.Vertices;
refVertOld = [];
refCost = tree.Cost;
prevAngle = 180;
n = 3; flag = 0;
```

```
while n < 10 %will keep iterating to find an angle between three points
greater than angleMax
    numPoint = size(refVert,1);
    %find where the angle between three points is greater than a threshold
    for i = n:(numPoint-2)
        %start vertex
        x0 = refVert(i-1,1); y0 = refVert(i-1,2)
        %middle vertex
        x1 = refVert(i,1); y1 = refVert(i,2)
        %end vertex
        x2 = refVert(i+1,1); y2 = refVert(i+1,2)
        v1 = [x1-x0; y1-y0] %start vector
        v2 = [x2-x1; y2-y1] %end vector
        x0_old = x0; x2_old = x2; y0_old = y0; y2_old = y2;
        angle = round(acosd(dot(v1,v2)/(norm(v1)*norm(v2)))) %angle between
vectors
        if (angle > 0) && abs(prevAngle-angle) > 1 %if the angle is greater
than the max allowable angle, record the vertex index
            prevAngle = angle;
            apexVtx = i;
            break
        end
    end
    if i == (numPoint-2) %if no apex was found, break out of while loop
        break
    end

    trigger = 0;
    k = 0;

    while trigger ~= 1 %will keep iterating a shorter tree path until path
intersects an object

        %find interpolated points on either side of the middle vertex
        slope = (y2 - y0)/(x2 - x0 + 0.001);
        b = y2 - slope*x2;
        if x2 > x0
%             xq = x0:0.05:x2;
                xq = linspace(x0, x2, 10);
        elseif x2 < x0
%             xq = x0:-0.05:x2;
xq = linspace(x2, x0, 10);
        elseif (x2 == x0) && (x0 > 0)
            x2 = x2 + 0.1;
            xq = x0:0.05:x2;
        elseif (x2 == x0) && (x0 < 0)
            x2 = x2 - 0.1;
            xq = x0:-0.05:x2;
        end
        x = [x0, x2];
        v = slope*x + b;
        vq1 = interp1(x,v,xq);
```

```
        %find if any of the interpolated points are inside an object
        for j = 1:numObj
            %if the object was jumped over, ignore the intersection of tree
with object
            if ismember(j, jumpedObj) == 1
                continue
            end
            for q = 1:size(xq,2)
                if (xObjMin(j) <= xq(q)) && (xObjMax(j) >= xq(q)) &&
(yObjMin(j) <= vq1(q)) && (yObjMax(j) >= vq1(q))
                    trigger = 1;
                    flag = 1;
                    break
                end
            end
            if flag == 1
                flag = 0;
                break
            end
        end

        if (size(refVert,1) < (apexVtx+2)) || (apexVtx-2 < 0) %prevents error
out from not enough points to connect to
            trigger = 1;
        elseif isequal(refVert, refVertOld) == 1
            trigger = 1;
            n = n + 1;
        elseif trigger ~= 1 %removal of apex vertex
            x0_old = x0; x2_old = x2; y0_old = y0; y2_old = y2; refVertOld =
refVert;
            x0 = refVert(apexVtx-1,1); y0 = refVert(apexVtx-1,2);
%establishing new initial vertex for segment before apex
            x2 = refVert(apexVtx+1,1); y2 = refVert(apexVtx+1,2);
%establishing new end vertex for segment before apex
            refVert(apexVtx,:) = []; %removal of apex vertex
        end
        if trigger ~= 1
            apexVtx = apexVtx - 1; %iterating apexVtx to prevent error out
from few vertices at the end of path
        else
            x0 = x0_old; x2 = x2_old; y0 = y0_old; y2 = y2_old; %restore old
vertices and coordinates
            refVert = refVertOld ;
            apexVtx = apexVtx + 1;
            xEndVtx = refVert(apexVtx+1, 1); %identify the end vertex between
removed vertices
            yEndVtx = refVert(apexVtx+1, 2);
            xVec = xEndVtx - refVert(apexVtx, 1); %define vector from apex to
end vertex
            yVec = yEndVtx  - refVert(apexVtx, 2);
            xUnitVec = xVec/norm([xVec yVec])*distInc; %define unit vector
scaled to distance increment
```

```
            yUnitVec = yVec/norm([xVec yVec])*distInc;
            m = 1;
            while hypot(xUnitVec*m, yUnitVec*m) < hypot(xVec, yVec) %plot
interpolated points between apex and end vertices
                refVert = [refVert((1:(apexVtx+m-1)),:); refVert(((apexVtx+m-
1):end),:)];
                refVert(apexVtx+m,:) = [xUnitVec*m + refVert(apexVtx,1),
yUnitVec*m + refVert(apexVtx,2)];
                m = m + 1;
            end
        end
    end
end

%editting tree vertices (eliminating points on straight lines) and cost based
on refined vertices
i = 1;
finish = 0;

%removing redundant points
while finish == 0
    a = refVert(i+1, 1:2) - refVert(i, 1:2);
    b = refVert(i+2, 1:2) - refVert(i+1, 1:2);
    angle = acosd(dot(a,b)/(norm(a)*norm(b))); %find if angle between 3
points is greater than 0
    if angle == 0 %if angle equals 0, remove repetitive coordinate for a
straight line
        refVert(i+1,:) = [];
    else
        i = i + 1;
    end
    if (i+2) > size(refVert,1)
        finish = 1;
    end
end

%evaluating cost
for j = 1:(size(refVert,1)-1)
    if j > 1
        vec1 = refVert(j,:) - refVert(j-1,:);
        vec2 = refVert(j+1,:) - refVert(j,:);
        rotCost =
round(acosd(dot(vec1,vec2)/(norm(vec1)*norm(vec2))))*rotWeight;
    else
        rotCost = 0;
    end
    refCost(j+1) = norm(refVert(j+1, 1:2) - refVert(j, 1:2)) + refCost(j) +
rotCost;
end

%eliminating any extra costs at end
refCost((size(refVert,1)+1):end) = [];
```

```
end
```

Jumping Rovers' Scenario Function

```
function[targets, numTarget, objMatrix] = Senario(num)

switch num
    case 1 %basic 3 targets, 1 in each corner, object off-screen
        %Target locations
        targets = [5 5; 45 5; 45 45];
        numTarget = size(targets,1);

        %Objects [x, y, width(x), length(y), height]
        objMatrix = [75 75 8 8 2];

    case 2 %2 targets with 1 object in between
        %Target locations
        targets = [5 25; 45 25];
        numTarget = size(targets,1);

        %Objects [x, y, width(x), length(y), height]
        objMatrix = [25 25 8 8 2];

    case 3 %2 targets with 1 long object in between
        %Target locations
        targets = [5 25; 45 25];
        numTarget = size(targets,1);

        %Objects [x, y, width(x), length(y), height]
        objMatrix = [25 25 8 20 2];

    case 4 %3 targets diagonal, 1 target on top of the object in center
        %Target locations
        targets = [5 5; 25 25; 45 25];
        numTarget = size(targets,1);

        %Objects [x, y, width(x), length(y), height]
        objMatrix = [25 25 8 8 2; 35 25 8 10 4];

    case 5  %2 targets, 2 objects adjacent to different heights
        %Target locations
        targets = [25 5; 25 45];
        numTarget = size(targets,1);

        %Objects [x, y, width(x), length(y), height]
        objMatrix = [25 30 20 8 2; 25 35 8 8 4];

    case 6  %2 targets, 2 long objects of different heights
        %Target locations
        targets = [5 25; 45 25];
```

```
    numTarget = size(targets,1);

    %Objects [x, y, width(x), length(y), height]
    objMatrix = [20 25 8 24 2; 30 25 8 24 6];

case 7 %4 targets, 4 random objects of different heights
    %Target locations
    targets = [5 5; 35 25; 25 35; 25 45];
    numTarget = size(targets,1);

    %Objects [x, y, width(x), length(y), height]
    objMatrix = [20 20 10 10 2; 35 25 8 25 4; 20 35 8 8 4; 25 35 8 8 6];

case 8  %4 targets, 5 adjacent objects of different heights
    %Target locations
    targets = [5 5; 25 25; 25 45; 45 45];
    numTarget = size(targets,1);

    %Objects [x, y, width(x), length(y), height]
    objMatrix = [15 25 8 25 2; 25 25 8 25 4; 35 25 8 25 6];

case 9 %basic 3 targets, 1 in each corner, No Object.
    %Target locations
    targets = [5 5; 40 75 ; 75 5];
    numTarget = size(targets,1);

    %Object [x, y, width(x), length(y), height]
    objMatrix = [0, 0, 0, 0, 0];

case 10
    %Target locations
    % Initialize Starting Coordinate.
    x_CP_Start = 500; % Lattitude
    y_CP_Start = 500; % Longitude

    % Set equidistant distance between checkpoints.
    for row = 1 : 16
        for column = 1 : 2
            if column == 1
                targets(row,column) = x_CP_Start;
                x_CP_Start = x_CP_Start + 1000;
            else
                targets(row,column) = y_CP_Start;
                if x_CP_Start > 4000
                    x_CP_Start = 500;
                    y_CP_Start = y_CP_Start + 1000;
                end
            end
        end
    end
```

```
        targets = targets/50;
        numTarget = size(targets,1);

        %Object [x, y, width(x), length(y), height]
        objMatrix = [ 10, 70, 23, 23, 0.5;...
            30, 70, 23, 23, 0.5;...
            30, 50, 23, 23, 0.5;...
            30, 30, 23, 23, 0.5;...
            50, 30, 23, 23, 0.5 ];

    case 11
        %Target locations
        targets = [3 10; 15 2; 2 1; 14 16; 16 12; 5 15; 8 12; 10 10]*4;
        numTarget = size(targets,1);
        objMatrix = [0, 0, 0, 0, 0];

    case 12 %manual
        %Target locations
        targets = [2 1; 15 2;3 10; 14 16; 16 12]*4;
        numTarget = size(targets,1);

        %Objects [x, y, width(x), length(y), height]
        objMatrix = [3 8 4 2 0.5; 10 10 2 2 0.5; 12 16 2 6 0.5; 13 5.5 2 4
0.5;...
            16 14 6 2 0.5; 14 16 2 2 1.1; 17 9 2 2 0.5]*4;
    otherwise
end
end
```

Experimental MATLAB Codes

Parallel Programming Initiation Code

```
% Develop by Kai Chuen Tan.
% Initiate parallel programming mode to control both rovers
% simultaneously
% Clear History.
clear all; clc; close all; % Clear all workspace, clear command window, and
close all figures.
delete(instrfind); % Arduino Clear Ports.

Advanium = raspi ('192.168.0.7','pi','Caltechtkc12345@'); % Connect the
MATLAB to the Raspberry Pi Zero W.
% Power Data Acquisition (DAQ)
openShell(Advanium); % Type 'cd Desktop/', then 'python Power_DAQ.py &'

% Clear History.
clear; clc; close all; % Clear all workspace, clear command window, and close
all figures.
delete(instrfind); % Arduino Clear Ports.
%delete(gcp('nocreate'));

Imap = 'I90'; % Jumping Rover 2's map.
Kmap = 'K0'; % Jumping Rover 1's map.


JRteam = {@IJR, @KJR}; % Jumping Rover Functions
settings = {Imap; Kmap}; % Jumping Rovers' map selections

% Run Jumping Rover 1 and Jumping Rover 2 functions at the same time.
parfor rover_Num = 1:2

    JRteam{rover_Num}(settings{rover_Num,:});

end
```

Jumping Rover 1's Function

```
function [] = KJR(Kmap)

% Raspberry Pi Jumping Rover - Coordinate-based command.
% Original code by MyungJin for Charging Rover and Blimp.
% Edited by Kai for Raspberry Pi Jumping Rover.

% Attempt a connection between the Raspberry Pi and Computer.
%Advanium = raspi ('192.168.2.5','pi','Caltechtkc12345@'); % Connect the
MATLAB to the Raspberry Pi Zero W.
Advanium = raspi ('192.168.0.7','pi','Caltechtkc12345@'); % Connect the
MATLAB to the Raspberry Pi Zero W.
%Advanium = raspi ('192.168.43.42','pi','Caltechtkc12345@'); % Connect the
MATLAB to the Raspberry Pi Zero W.

%% VICON Setup
HostName = 'localhost:801'; % Initialize the host name.
addpath('C:\Program Files\Vicon\DataStream SDK\Win64\MATLAB'); % Adds the
MATLAB folder to the top of the search path for the current MATLAB session.
% Load the Software Development Kit (SDK).
fprintf('Hello, creators. SDK is loading...\n');
Client.LoadViconDataStreamSDK();
fprintf('Loading Complete, sir!\n');

Kai_Client = Client(); % Create a new client.

% Connect to the server.
fprintf('VICON System is connecting to %s ...', HostName);

% Waiting for a connection.
while ~Kai_Client.IsConnected().Connected

    %Establish direct connection.
    Kai_Client.Connect(HostName);
    fprintf('.');

end

% Enable several different data types. (RasPi)
Kai_Client.EnableSegmentData();
Kai_Client.EnableMarkerData();
Kai_Client.EnableUnlabeledMarkerData();
Kai_Client.EnableDeviceData();

% Set the streaming mode.
Kai_Client.SetStreamMode(StreamMode.ClientPull);

% Set the global up axis.
```

```
Kai_Client.SetAxisMapping( Direction.Forward, Direction.Left, Direction.Up );
% Define positive X, Y, Z directions.

% Obtain the axis mapping.
Kai_Client.GetAxisMapping();

% Discover the VICON Version Number. (RasPi)
Output_GetVersion_RasPi = Kai_Client.GetVersion();
fprintf( 'Version: %d.%d.%d\n', Output_GetVersion_RasPi.Major,
Output_GetVersion_RasPi.Minor, Output_GetVersion_RasPi.Point );


%% Position Data Recorder Log Files
current_moment = clock; % Store current clock time, [year month day hour
minute seconds].

% Store year, month, day, hour, minute.
dateNtime = [num2str(current_moment(1), '%02d'),...
             num2str(current_moment(2), '%02d'),...
             num2str(current_moment(3), '%02d'),...
             '_',...
             num2str(current_moment(4), '%02d'),...
             num2str(current_moment(5), '%02d')];

existance = 7; % Existance is 7.

% If the file does not exist.
if exist('CoOp_Mission_Data', 'dir') ~= existance && exist('mission_data',
'dir') ~= existance

    mkdir('CoOp_Mission_Data'); % Create the CoOp_Mission_Data folder.

end

% Create Positioning Data File in csv file format (RasPi).
File_Name_Pos = [ 'CoOp_Mission_Data/Data_Pos_', dateNtime, '.csv' ];
FID_Pos = fopen(File_Name_Pos, 'w'); % Open the file ID with write access.
disp('Opening Cooperational Mission Data File...') % Report the status
message.

% Start
missionFile = xlsread([Kmap,'.xlsx']);

% Set matrix for coordinates with scaling
next_RP(:,1) = missionFile(:,1);
next_RP(:,2) = missionFile(:,2);
next_RP(:,3) = missionFile(:,3);

% GPIO Initializations
% GPIO Pins Initialization
```

```
Jump_STBY = 20; % Motor Controller Switch Pin.
Drive_Jump_BIN1 = 4; % Motor Controller Clockwise Input Pin.
Drive_Jump_BIN2 = 23; % Motor Controller Direction Counter-clockwise Input
Pin.
Drive_Jump_Speed_PWMB = 19; % Motor Controller Speed Control Pin.
Drive_STBY = 17; % Motor Controller Switch Pin.
Drive_Left_Forward_AIN1 = 27; % Motor Controller Direction Forward Input Pin.
Drive_Left_Backward_AIN2 = 22; % Motor Controller Direction Reverse Input
Pin.
Drive_Left_Speed_PWMA = 18; % Motor COntroller Speed Control Pin.
Drive_Right_Forward_BIN1 = 5; % Motor Controller Direction Forward Input Pin.
Drive_Right_Backward_BIN2 = 6; % Motor Controller Direction Reverse Input
Pin.
Drive_Right_Speed_PWMB = 12; % Motor COntroller Speed Control Pin.

% Motor Pins Configuration
configurePin(Advanium, Jump_STBY, 'DigitalOutput'); % Set the Stanby Pin /
Motor Switch as the Digital Output Pin.
configurePin(Advanium, Drive_Jump_BIN1, 'DigitalOutput'); % Set the Clockwise
Pin as the Digital Output Pin.
configurePin(Advanium, Drive_Jump_BIN2, 'DigitalOutput'); % Set the Counter-
clockwise Pin as the Digital Output Pin.
configurePin(Advanium, Drive_Jump_Speed_PWMB, 'PWM'); % Set the Speed Control
Pin as the Pulse Width Modulation Pin.
configurePin(Advanium, Drive_STBY, 'DigitalOutput'); % Set the Stanby Pin /
Motor Switch as the Digital Output Pin.
configurePin(Advanium, Drive_Left_Forward_AIN1, 'DigitalOutput'); % Set the
Forward Pin as the Digital Output Pin.
configurePin(Advanium, Drive_Left_Backward_AIN2, 'DigitalOutput'); % Set the
Reversed Pin as the Digital Output Pin.
configurePin(Advanium, Drive_Left_Speed_PWMA, 'PWM'); % Set the Speed Control
Pin as the Pulse Width Modulation Pin.
configurePin(Advanium, Drive_Right_Forward_BIN1, 'DigitalOutput'); % Set the
Forward Pin as the Digital Output Pin.
configurePin(Advanium, Drive_Right_Backward_BIN2, 'DigitalOutput'); % Set the
Reversed Pin as the Digital Output Pin.
configurePin(Advanium, Drive_Right_Speed_PWMB, 'PWM'); % Set the Speed
Control Pin as the Pulse Width Modulation Pin.
writeDigitalPin(Advanium, Drive_STBY, 1); % Turn on the Motor Controller.
writeDigitalPin(Advanium, Jump_STBY, 1); % Turn on the Motor Controller.
writePWMFrequency(Advanium, Drive_Left_Speed_PWMA, 2000); % Set the PWM
Frequency to be 2 kHz.
writePWMFrequency(Advanium, Drive_Right_Speed_PWMB, 2000); % Set the PWM
Frequency to be 2 kHz.


% PWM Duty Cycle (%) / Velocity.
MinVel = 0.07;
MedVel = 0.12;
%MaxVel = 0.14;
turnComp = 1; % Compensation value for turning to stay on a straight line
path (overcome difference in left/right motor)
```

```
coordNum_RP = 1; % Coordinate index number (RPi)
%scale = 1; % Scaleable value for coordinate system to actual test area for
dimensional purposes
%xOffset = 0; yOffset = 0; zOffset = 0; angOffset = 0; % Offsets to match
with actual test area for dimensional purposes
thetaDegTol_RasPi = 20; % (RaspPi)Tolerance for degree difference in current
heading and desired heading


distTol_RasPi = 100; % Tolerance for distance difference between current
position and desired position (mm).
%linJumpDist_RasPi = 800; % Distance forward that the rover jumps (mm).
linJumpDist_RasPi = 920; % Distance forward that the rover jumps (mm).

% Raspberry Pi Settings
jump_RP = 0; % Jump servo command
PWML_RP = 0; % PWM Left motor (Range: 0.00-1.00)
PWMR_RP = 0; % PWM Right motor (Range: 0.00-1.00)
DIRL_RP = 0; % Direction Left motor (2 = forward, 1 = backward 0 = idle)
DIRR_RP = 0; % Direction Right motor (2 = forward, 1 = backward, 0 = idle)
PWM_adj_RP = 0.018; % Adjustment for mismatched motors
jumpTrigger_RP = 0; % Jump triggered from xlsx files.

%shellOpen = 1; messageOpen = 0; % Shell Status, Message Status, and Request
Status Initialization

disp('\nJumping Rover Power Data is Recording in Real-time...\n') % Report
the status message.

% Iterating through all coordinates

tic % Begin time recording

while coordNum_RP <= size(missionFile,1)


    Pos_time = toc;

    [~, dTheta_RP, distError_RP, RoverPos_RP] = posHeadRasPi(Kai_Client,
next_RP(coordNum_RP,:)); % Evaluate next vector and error in heading

    % (RPi) write data (time, position, heading, power) of rover to mission
data file
    recordDataLocRasPi(FID_Pos, Pos_time, RoverPos_RP)

    % Raspberry Pi Error Adjustment
    if distError_RP < distTol_RasPi

        coordNum_RP = coordNum_RP + 1; %iterate coordinate index by 1
```

```
        jumpTrigger_RP = 0; %trigger toggle jump command on and off to avoid
multiple jumps

    % Change heading
    elseif abs(dTheta_RP) > thetaDegTol_RasPi

        %Rotation direction
        if dTheta_RP > 0 % Reverse left and forward right to turn left

            DIRL_RP = 1;
            DIRR_RP = 2;
            PWML_RP = MinVel;
            PWMR_RP = MinVel + 1.8*PWM_adj_RP;

        elseif dTheta_RP < 0 % Forward left and reverse right to turn right

            DIRL_RP = 2;
            DIRR_RP = 1;
            PWML_RP = MinVel;
            PWMR_RP = MinVel + 1.8*PWM_adj_RP;

        end

        PWML_RP = max(MinVel, PWML_RP);
        PWMR_RP = max(MinVel + PWM_adj_RP, PWMR_RP);
        jump_RP = 0;

    % Minimize distance error
    elseif distError_RP > distTol_RasPi

        PWML_RP = 0;
        PWMR_RP = 0;
        DIRL_RP = 0;
        DIRR_RP = 0;

        if next_RP(coordNum_RP,3) == 1 % If there is a 1 in the z-coordinate
position, jumping is needed

            %if distance between the current position and the desired
position is less than the forward distance the robot can jump, jump
            if (distError_RP <= linJumpDist_RasPi) && (jumpTrigger_RP == 0)

                jump_RP = 1;
                jumpTrigger_RP = 1;

            else

                % Run forward motion
                jump_RP = 0;
```

```
                [DIRL_RP, DIRR_RP, PWML_RP, PWMR_RP] =
fwdMotionRasPi(dTheta_RP, thetaDegTol_RasPi, turnComp, MedVel, PWM_adj_RP);

            end

        else

            % Run forward motion
            jump_RP = 0;
            [DIRL_RP, DIRR_RP, PWML_RP, PWMR_RP] = fwdMotionRasPi(dTheta_RP,
thetaDegTol_RasPi, turnComp, MedVel, PWM_adj_RP);

        end
    end


    [jump_RP] = normMtrCtrl(Advanium,...
                 Drive_Left_Forward_AIN1, Drive_Left_Backward_AIN2,
Drive_Left_Speed_PWMA,...
                 Drive_Right_Forward_BIN1, Drive_Right_Backward_BIN2,
Drive_Right_Speed_PWMB,...
                 jump_RP, PWML_RP, PWMR_RP, DIRL_RP, DIRR_RP);


    if jump_RP == 1

        % GPIO Initializations
        % GPIO Pins Initialization
        Jump_STBY = 20; % Motor Controller Switch Pin.
        Drive_Jump_BIN1 = 4; % Motor Controller Clockwise Input Pin.
        Drive_Jump_BIN2 = 23; % Motor Controller Direction Counter-clockwise
Input Pin.
        Drive_Jump_Speed_PWMB = 19; % Motor Controller Speed Control Pin.
        Drive_STBY = 17; % Motor Controller Switch Pin.
        Drive_Left_Forward_AIN1 = 27; % Motor Controller Direction Forward
Input Pin.
        Drive_Left_Backward_AIN2 = 22; % Motor Controller Direction Reverse
Input Pin.
        Drive_Left_Speed_PWMA = 18; % Motor COntroller Speed Control Pin.
        Drive_Right_Forward_BIN1 = 5; % Motor Controller Direction Forward
Input Pin.
        Drive_Right_Backward_BIN2 = 6; % Motor Controller Direction Reverse
Input Pin.
        Drive_Right_Speed_PWMB = 12; % Motor COntroller Speed Control Pin.

        % Motor Pins Configuration
        configurePin(Advanium, Jump_STBY, 'DigitalOutput'); % Set the Stanby
Pin / Motor Switch as the Digital Output Pin.
        configurePin(Advanium, Drive_Jump_BIN1, 'DigitalOutput'); % Set the
Clockwise Pin as the Digital Output Pin.
```

```
        configurePin(Advanium, Drive_Jump_BIN2, 'DigitalOutput'); % Set the
Counter-clockwise Pin as the Digital Output Pin.
        configurePin(Advanium, Drive_Jump_Speed_PWMB, 'PWM'); % Set the Speed
Control Pin as the Pulse Width Modulation Pin.
        configurePin(Advanium, Drive_STBY, 'DigitalOutput'); % Set the Stanby
Pin / Motor Switch as the Digital Output Pin.
        configurePin(Advanium, Drive_Left_Forward_AIN1, 'DigitalOutput'); %
Set the Forward Pin as the Digital Output Pin.
        configurePin(Advanium, Drive_Left_Backward_AIN2, 'DigitalOutput'); %
Set the Reversed Pin as the Digital Output Pin.
        configurePin(Advanium, Drive_Left_Speed_PWMA, 'PWM'); % Set the Speed
Control Pin as the Pulse Width Modulation Pin.
        configurePin(Advanium, Drive_Right_Forward_BIN1, 'DigitalOutput'); %
Set the Forward Pin as the Digital Output Pin.
        configurePin(Advanium, Drive_Right_Backward_BIN2, 'DigitalOutput'); %
Set the Reversed Pin as the Digital Output Pin.
        configurePin(Advanium, Drive_Right_Speed_PWMB, 'PWM'); % Set the
Speed Control Pin as the Pulse Width Modulation Pin.
        writeDigitalPin(Advanium, Drive_STBY, 1); % Turn on the Motor
Controller.
        writeDigitalPin(Advanium, Jump_STBY, 1); % Turn on the Motor
Controller.

    end

    if coordNum_RP > size(missionFile,1)

        % (RasPi) Switch off the Driving Motor Controller
        writeDigitalPin(Advanium, Drive_Left_Forward_AIN1, 0); % Turn off the
forward pin.
        writeDigitalPin(Advanium, Drive_Left_Backward_AIN2, 0); % Turn off
the reverse pin.
        writeDigitalPin(Advanium, Drive_Right_Forward_BIN1, 0); % Turn off
the forward pin.
        writeDigitalPin(Advanium, Drive_Right_Backward_BIN2, 0); % Turn off
the reverse pin.
        writeDigitalPin(Advanium, Drive_Jump_BIN1, 0); % Turn off the
clockwise pin.
        writeDigitalPin(Advanium, Drive_Jump_BIN2, 0); % Turn off the
counter-clockwise pin.
        writeDigitalPin(Advanium, Drive_STBY, 0); % Turn on the Motor
Controller.
        writeDigitalPin(Advanium, Jump_STBY, 0); % Turn on the Motor
Controller.

    end
end

% Disable Raspberry Pi Functions
% fclose(fid_loc); % (Arduino) Close Location File
fclose(FID_Pos); % (RPi) Close Location File
% fclose(fid_pwr); % (Arduino) Close Power File
```

```
system(Advanium, 'pkill -f Power_DAQ.py'); % Terminate the pyhton script.
%getFile(Advanium, '/home/pi/Desktop/*.csv', 'C:\Users\User\Desktop'); % Save
data.
getFile(Advanium, '/home/pi/Desktop/PowerDataPackage/*.csv',
'C:\Users\tan.783\Desktop\Multiple Jumping Rovers MATLAB
Code_V2\MATLAB\CoOp_Mission_Data'); % Save data.
disp('Closing mission data files.')

end
```

Jumping Rover 1's Forward Motion Control Function

```
% Jumping Rover 1's Forward Motion Control Function
function [DIRL, DIRR, PWML, PWMR] = fwdMotionRasPi(dTheta, thetaDegTol,
turnComp, speed, PWM_adj)

if dTheta > thetaDegTol
    DIRL = 2;
    DIRR = 2;
    PWML = min(max(round(speed - abs(dTheta)/1000*turnComp),0), 1);
    PWMR = min(round(speed + abs(dTheta)/1000*turnComp + PWM_adj),1);
elseif dTheta < -thetaDegTol
    DIRL = 2;
    DIRR = 2;
    PWML = min(round(speed + abs(dTheta)/1000*turnComp),1);
    PWMR = min(max(round(speed - abs(dTheta)/1000*turnComp + PWM_adj),0),1);
else
    DIRL = 2;
    DIRR = 2;
    PWML = speed;
    PWMR = speed+PWM_adj;
end

end
```

Jumping Rover 1's Jump Function

```
function [] = Jump_Test(Advanium)

% Execute Jump.py Python Script.
system(Advanium, 'python /home/pi/Desktop/Jump.py');

end
```

Jumping Rover 1's Motion Control Function

```
% Jumping Rover 1's Motion Control Function
function [jump] = normMtrCtrl(Advanium,...
                          Drive_Left_Forward_AIN1,
Drive_Left_Backward_AIN2,Drive_Left_Speed_PWMA,...
                          Drive_Right_Forward_BIN1,
Drive_Right_Backward_BIN2,Drive_Right_Speed_PWMB,...
                          jump, PWML_val, PWMR_val, DIRL, DIRR)
% Master Control

% Left Motor Direction
% Left Forward.

if (DIRL == 2)

    writeDigitalPin(Advanium, Drive_Left_Forward_AIN1, 1); % Set the
direction to be forward.
    writeDigitalPin(Advanium, Drive_Left_Backward_AIN2, 0); % Turn off the
reverse pin to prevent stalling the motor.

% Left Stop.
elseif (DIRL == 0)

    writeDigitalPin(Advanium, Drive_Left_Forward_AIN1, 0); % Set the
direction to be forward.
    writeDigitalPin(Advanium, Drive_Left_Backward_AIN2, 0); % Turn off the
reverse pin to prevent stalling the motor.

% Left Reverse.
elseif (DIRL == 1)

    writeDigitalPin(Advanium, Drive_Left_Forward_AIN1, 0); % Set the
direction to be forward.
    writeDigitalPin(Advanium, Drive_Left_Backward_AIN2, 1); % Turn off the
reverse pin to prevent stalling the motor.

end

% Right Motor Direction
% Right Forward.
if (DIRR == 2)

    writeDigitalPin(Advanium, Drive_Right_Forward_BIN1, 1); % Set the
direction to be forward.
    writeDigitalPin(Advanium, Drive_Right_Backward_BIN2, 0); % Turn off the
reverse pin to prevent stalling the motor.

% Right Stop.
```

```
elseif (DIRR == 0)

    writeDigitalPin(Advanium, Drive_Right_Forward_BIN1, 0); % Set the
direction to be forward.
    writeDigitalPin(Advanium, Drive_Right_Backward_BIN2, 0); % Turn off the
reverse pin to prevent stalling the motor.

% Right Reverse.
elseif (DIRR == 1)

    writeDigitalPin(Advanium, Drive_Right_Forward_BIN1, 0); % Set the
direction to be forward.
    writeDigitalPin(Advanium, Drive_Right_Backward_BIN2, 1); % Turn off the
reverse pin to prevent stalling the motor.

end

writePWMDutyCycle(Advanium, Drive_Left_Speed_PWMA, PWML_val); % Set the PWM
Duty Cycle.
writePWMDutyCycle(Advanium, Drive_Right_Speed_PWMB, PWMR_val); % Set the PWM
Duty Cycle.


% Jump action.
if (jump == 1)

    Jump_Test(Advanium)

end

end
```

Jumping Rover 1's Location and Heading Angle Tracker Function

```
% Function that outputs the position and heading angle of the Jumping Rover 1
function [WheelCenter, JRAng, GlobalAng] = OrganizeVICON_RoverRasPi(marker)

%Offset vector TBD
offset = [0 0];

%Labeling Rover and Goal
switch marker(3,1).obj_name

%     case 'Goal'
%        Goal_Marker_1 = 1;

      case 'RaspJumpingPi'
         RaspJumpingPi = 3;
end

% Marking Jumping Rover markers
WheelRight = 1;
FrontRight = 2;
WheelLeft  = 3;
BackCenter = 4;

% Define Axis.
x_Dir = 1;
y_Dir = 2;
z_Dir = 3;

% Organizing data from Vicon for rover orientation
 WheelCenter(x_Dir) = (marker(RaspJumpingPi,WheelLeft).Translation(x_Dir) +
marker(RaspJumpingPi,WheelRight).Translation(x_Dir))/2;
 WheelCenter(y_Dir) = (marker(RaspJumpingPi,WheelLeft).Translation(y_Dir) +
marker(RaspJumpingPi,WheelRight).Translation(y_Dir))/2;
 WheelCenter(z_Dir) = (marker(RaspJumpingPi,WheelLeft).Translation(z_Dir) +
marker(RaspJumpingPi,WheelRight).Translation(z_Dir))/2;

 % Back Center Coordinates.
 RearCenter = [marker(RaspJumpingPi,BackCenter).Translation(x_Dir)
marker(RaspJumpingPi,BackCenter).Translation(y_Dir)
marker(RaspJumpingPi,BackCenter).Translation(z_Dir)];

% Heading Vector.
JRVec = [WheelCenter(x_Dir)- RearCenter(x_Dir)+offset(x_Dir);
WheelCenter(y_Dir)- RearCenter(y_Dir)+offset(y_Dir)];

% Heading Angle.
JRAng = atan2d(JRVec(y_Dir),JRVec(x_Dir));
```

```
% Global Vector.
GlobalVec = [100; 0];

% Global Angle.
GlobalAng = atan2d((det([GlobalVec JRVec])),dot(GlobalVec, JRVec));


end
```

Jumping Rover 1's VICON Identification Function

```
function [ marker ] = PingVICONRasPi( MyClient )

for i=1:2 % Numbers seem to vary during the first 2 runs

    % Get a frame if VICON is paused, do loop will wait

   while MyClient.GetFrame().Result.Value ~= Result.Success
   end% while

   fprintf( '\n' );

   % Get the frame number
   Output_GetFrameNumber = MyClient.GetFrameNumber();


   % Get the timecode
   Output_GetTimecode = MyClient.GetTimecode();


   % Count the number of subjects
   SubjectCount = MyClient.GetSubjectCount().SubjectCount;

   for SubjectIndex = 1:SubjectCount

     % Get the subject name
     SubjectName = MyClient.GetSubjectName( SubjectIndex ).SubjectName;

     % Get the root segment
     RootSegment = MyClient.GetSubjectRootSegmentName( SubjectName
).SegmentName;

     % Count the number of segments
     SegmentCount = MyClient.GetSegmentCount( SubjectName ).SegmentCount;

     % Count the number of markers
     MarkerCount = MyClient.GetMarkerCount( SubjectName ).MarkerCount;

     for MarkerIndex = 1:MarkerCount

       % Get the marker name
       MarkerName = MyClient.GetMarkerName( SubjectName, MarkerIndex
).MarkerName;

       % Get the marker parent
```

```
       MarkerParentName = MyClient.GetMarkerParentName( SubjectName,
MarkerName ).SegmentName;

       % Get the global marker translat ion
       Output_GetMarkerGlobalTranslation =
MyClient.GetMarkerGlobalTranslation( SubjectName, MarkerName );
       Output_GetMarkerGlobalTranslation.mk_name=MarkerName;
       Output_GetMarkerGlobalTranslation.obj_name=SubjectName;
       Output_GetMarkerGlobalTranslation.mkr_count=MarkerCount;
       marker(SubjectIndex,MarkerIndex)=Output_GetMarkerGlobalTranslation;


     end % MarkerIndex
   end % SubjectIndex
end % Double check

end % Function Ends.
```

Jumping Rover 1's Next Vector and Heading Error Evaluation Function

```
% Function that evaluates next vector and error in heading
function [nextVec, dTheta, distError, RoverPos] = posHeadRasPi(Kai_Client,
next)

    [Marker] = PingVICONRasPi(Kai_Client); % Collects activated marker
coordinates
    [RoverPos, ~, GlobalAng] = OrganizeVICON_RoverRasPi(Marker); % Determines
center of rover and heading; RoverPos(1 = x, 2 = y, 3 = z), heading in vector
form
    nextVec = next(1:2) - RoverPos(1:2); % Vector to reach next coordinate
    dTheta = atan2d(nextVec(2), nextVec(1)) - GlobalAng; % Heading angle
change needed

    % Adjust the range of reading from (-180 Degrees - 180 Degrees) to (0
Degree - 360 Degrees).
    if dTheta > 180

        dTheta = dTheta - 360;

    elseif dTheta < -180

        dTheta = dTheta + 360;

    end

    % Overall length between the jumping rover position and the next tartget
position.
    distError = sqrt((RoverPos(1)-next(1))^2 + (RoverPos(2)-next(2))^2);

end
```

Jumping Rover 1's Location Data Acquisition Function

```
%Record Jumping Rover 1's Location Data
function [] = recordDataLocRasPi(fid_loc, time, RoverPos)

        % Store data in a vector form.
        dataLog = [double(time) double(RoverPos(1)) double(RoverPos(2))
double(RoverPos(3))];

        % Initialize the variable as empty string.
        format_string = '';

        for i=1:1:length(dataLog)

            format_string = strcat(format_string, '%3.2f'); % Float type
instead of string.

            if i < length(dataLog)

                format_string = strcat(format_string, ','); % Different
Column

            else

                format_string = strcat(format_string, '\n'); % New line.

            end

        end

        % Write to the csv file.
        fprintf(fid_loc, format_string, dataLog);

end
```

Jumping Rover 2's Function

```
% Jumping Rover following set of coordinates
%Original code by Myungjin for Charging Rover and Blimp
%Editted by MyungJin for Jumping Rover

%% Setup
r2d = 180 / pi;

%% Serial
%com = 'com8';
% com = 'com8';
% com = 'com12';
%init_Serial;
s = serial('com8', 'Baudrate', 9600, 'DataBits', 8, 'Timeout', 1);
fopen(s);

%% Vicon
init_Vicon;
init_Object;

%% Logfile
init_Log;

%% Parameter
JR_I_ID = hex2dec('C1'); %Jumping rover id (to distinguish between other
rovers and blimp)
command = 65;
STPSPD = 111;
TRNSPD = 60;
PWML = 0; %PWM Left motor (Range: 0-255)
PWMR = 0; %PWM Right motor (Range: 0-255)
DIRL = 0; %Direction Left motor (1 = forward, 0 = backward)
DIRR = 0; %Direction Right motor (1 = forward, 0 = backward)
JUMP = STPSPD; %Jump servo command
% command packet
%[ID command PWML PWMR DIRL DIRR JUMP CHECKSUM END]
cntWP = 1;
disable = 0;
%timeold = 0;
loop = 1;
count = 0;
gooddata = 0;
volt = 0;
curr =0;
packet = zeros(1,10);
JumpDist = 255;
jump_init = 0;
t_jump = 0;
```

```
%% Start.
% coord = xlsread(strcat('I90', '.xlsx'));
coord = assignjump(Imap, JumpDist);
%coord = assignjump('I90', JumpDist);
numWP = size(coord(:,1),1);

% pause(7);

tic
while(disable < 50)

    % positoins and attitude
    [marker] = PingVICON(MyClient);
    [Goal, C_I, Vec_I, C_K] = OrganizeVICON_Rover(marker, Goal, JR_I, JR_K);
    time = toc;
    displacement = [coord(cntWP, 1) - C_I.x; coord(cntWP, 2) - C_I.y];
    distance = sqrt(displacement(1)^2 + displacement(2)^2);
    dangle = atan2((det([Vec_I, displacement])), dot(Vec_I, displacement)) *
r2d;

    % terminating condition
    if (Goal.z < 200)
        disable = disable + 10;
    end

    % Running (jumping or rolling)
    if(coord(cntWP, 3) == 1)
           JUMP = STPSPD;
        if (abs(dangle) > 10)
            if dangle > 0
                DIRL = 0;
                DIRR = 1;
                PWML = .4 * dangle;
                PWMR = .4 * dangle;
            elseif dangle < 0
                DIRL = 1;
                DIRR = 0;
                PWML = .4 * abs(dangle);
                PWMR = .4 * abs(dangle);
            end
            PWML = max(42, PWML);
            PWMR = max(42, PWMR);
        elseif dangle >= 0
            DIRL = 1;
            DIRR = 1;
            forward = distance;
            PWML = .6*forward + dangle;
            PWMR = .6*forward - dangle;
        elseif dangle < 0
            DIRL = 1;
            DIRR = 1;
            forward = distance;
```

```
        PWML = .6 *forward + abs(dangle);
        PWMR = .6 *forward - abs(dangle);
    end
    PWML = min(60, PWML);
    PWML = uint8(max(0, PWML));
    PWMR = min(60, PWMR);
    PWMR = uint8(max(0, PWMR));
    if (distance < 50 && jump_init == 0)
        jump_init = 1;
        t_jump = toc;
    end
    if(jump_init == 1 && toc-t_jump < 1.4)
        PWML = 0;
        PWMR = 0;
        DIRL = 0;
        DIRR = 0;
        JUMP = TRNSPD;
    end
    if(jump_init == 1 && toc - t_jump > 1.43)
        jump_init = 0;
        t_jump = -99999;
        cntWP = cntWP + 1;
        JUMP = STPSPD;
    end
else
    JUMP = STPSPD;
    if (abs(dangle) > 15)
        if dangle > 0
            DIRL = 0;
            DIRR = 1;
            PWML = .9 * dangle;
            PWMR = .9 * dangle;
        elseif dangle < 0
            DIRL = 1;
            DIRR = 0;
            PWML = .9 * abs(dangle);
            PWMR = .9 * abs(dangle);
        end
        PWML = max(42, PWML);
        PWMR = max(42, PWMR);
    elseif dangle >= 0
        DIRL = 1;
        DIRR = 1;
        forward = distance;
        PWML = 1.5*forward + dangle;
        PWMR = 1.5*forward - dangle;
    elseif dangle < 0
        DIRL = 1;
        DIRR = 1;
        forward = distance;
        PWML = 1.5 *forward + abs(dangle);
        PWMR = 1.5*forward - abs(dangle);
    end
```

```
        PWML = min(80, PWML);
        PWML = uint8(max(0, PWML));
        PWMR = min(80, PWMR);
        PWMR = uint8(max(0, PWMR));
        if (C_I.z > 180)
            PWML = uint8(max(80, PWML));
            PWML = uint8(max(80, PWML));
        end
        collision = [C_I.x-C_K.x, C_I.y-C_K.y, C_I.z-C_K.z];
        if norm(collision) < 270
            PWML = 0;
            PWMR = 0;
            DIRL = 0;
            DIRR = 0;
            JUMP = STPSPD;
        end
        if distance < 150
            cntWP = cntWP + 1;
        end
    end

    % command
    scommand = [JR_I_ID command PWML PWMR DIRL DIRR JUMP 0 13];
%     scommand = [ID command 8 9 0 DIRR JUMP 0 13];
    checksum = 0;
    for i = 2:1:7
        checksum = bitxor(checksum, scommand(i));
    end
    scommand(8) = checksum;
    fwrite(s, scommand);

    % vc telemetry
    if(s.BytesAvailable >= 10)
        for k = 1:10
            packet(k) = fread(s, 1, 'uint8');
        end
        checksum = 0;
        for j = 3:8
            checksum = bitxor(checksum, packet(j));
        end
        if (packet(1) == 193 && packet(2) == 0 && packet(9) == checksum &&
packet(10) == 13)
            count = double(packet(3)) * 256 + double(packet(4));
            volt = (double(packet(5)) * 256 + double(packet(6))) / 100;
            curr = (double(packet(7)) * 256 + double(packet(8))) / 10;
            gooddata = 1;
        else
            flushinput(s);
            gooddata = 0;
        end
    end
```

```
    % log
    data_to_log = [double(time), double(C_I.x), double(C_I.y), double(C_I.z),
double(PWML), double(PWMR), double(JUMP), double(count),double(volt),
double(curr), double(gooddata), double(C_K.x), double(C_K.y), double(C_K.z)];
    if loop == 1
        fprintf(fid_I, 'time [s], pos_x [mm], pos_y [mm], pos_z [mm], PWML,
PWMR, JUMP, count, volt [V], curr [mA], gooddata \n');
        format_string = '';
        for i=1:1:length(data_to_log)
            format_string = strcat(format_string, '%3.4f');
            if i < length(data_to_log)
                format_string = strcat(format_string, ',');
            else
                format_string = strcat(format_string, '\n');
            end
        end
    end
    fprintf(fid_I, format_string, data_to_log);
    loop = loop + 1;

    if cntWP > numWP
        disable = 99999;
    end
end

%% Terminate
% disable all functions
fclose(fid_I);
PWML = 0;
PWMR = 0;
DIRL = 0;
DIRR = 0;
JUMP = STPSPD;
scommand = [JR_I_ID command PWML PWMR DIRL DIRR JUMP 0 13];
checksum = 0;
for i = 2:1:7
    checksum = bitxor(checksum, scommand(i));
end
scommand(8) = checksum;
fwrite(s, scommand);
fclose(s);
disp('Terminated!');
```

Jumping Rover 2's Log File Initialization Script

```
%% Log File Initialization

thismoment = clock;
date_time = '';
for i = 1:5
    date_time = [date_time, num2str(thismoment(i), '%02d')];
    if i == 3
        date_time = [date_time, '_'];
    end
end

if exist('mission_data', 'dir') ~= 7
    mkdir('mission_data');
end

file_name = ['mission_data/JR1_',date_time,'.csv'];
fid_I = fopen(file_name, 'w');
```

Jumping Rover 2's VICON Objects Initialization Function

```
%% Objects
% Labeling Rovers and Goal
[marker] = PingVICON(MyClient);
for i = 1:size(marker,1)
    switch marker(i,1).obj_name
        case 'magic_wand'
            Goal.idx = i;

        case 'Jumping_Rover2'
            JR_I.idx = i;

        case 'RaspJumpingPi'
            JR_K.idx = i;
    end
end

% Wand
Goal.marker = 2;

% Jumping rover_Isaac
for j = 1:4
    switch marker(JR_I.idx,j).mk_name
        case 'Left'
          JR_I.L = j;

      case 'Right'
          JR_I.R = j;

      case 'Front'
          JR_I.F = j;

%       case 'BRight'
%           JR_I.BR = j;

%       case 'Back'
%           JR_I.B = j;
    end
end

% Jumping rover_Kai
for j = 1:4
    switch marker(JR_K.idx,j).mk_name
        case 'WheelRight'
          JR_K.WR = j;

      case 'FrontRight'
          JR_K.FR = j;
```

```
        case 'WheelLeft'
            JR_K.WL = j;

        case 'BackCenter'
            JR_K.BC = j;

          otherwise
    end
end
```

Jumping Rover 2's VICON Initialization Function

```
%% Vicon Initialization

% adds the specified folders to the top of the search path for the current
MATLAB® session.
addpath('C:\Program Files\Vicon\DataStream SDK\Win64\MATLAB')

% Load the SDK
fprintf('Loading SDK...');
Client.LoadViconDataStreamSDK();
fprintf('done\n');

% Program options
HostName = 'localhost:801';

% Make a new client
MyClient = Client();

% Connect to a server
fprintf('Connecting to %s ...', HostName);
while ~MyClient.IsConnected().Connected
  % Direct connection
  MyClient.Connect(HostName);

  fprintf( '.' );
end
fprintf( '\n' );

% Enable some different data types
MyClient.EnableSegmentData();
MyClient.EnableMarkerData();
MyClient.EnableUnlabeledMarkerData();
MyClient.EnableDeviceData();

% Set the streaming mode
MyClient.SetStreamMode( StreamMode.ClientPull );

% Set the global up axis
MyClient.SetAxisMapping( Direction.Forward, ...
                         Direction.Left,    ...
                         Direction.Up );    % Z-up

Output_GetAxisMapping = MyClient.GetAxisMapping();


% Discover the version number
Output_GetVersion = MyClient.GetVersion();
```

```
fprintf( 'Version: %d.%d.%d\n', Output_GetVersion.Major, ...
                                Output_GetVersion.Minor, ...
                                Output_GetVersion.Point );
```

Jumping Rover 1's Python Scripts

Jump Activation Code

```
import RPi.GPIO as GPIO
from gpiozero import PWMOutputDevice
from gpiozero import DigitalOutputDevice

import time
from time import sleep


GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)


# Rolling Motor Driver Pinouts
Roll_STBY = 17
# Left Motor
Roll_L_PWMA = 18
Roll_L_AIN1 = 27
Roll_L_AIN2 = 22
# Right Motor
Roll_R_PWMB = 12
Roll_R_BIN1 = 5
Roll_R_BIN2 = 6
# Rolling Motors Setup
GPIO.setup(Roll_STBY, GPIO.OUT)
LMotor = PWMOutputDevice(Roll_L_PWMA, True, 0, 50)
RMotor = PWMOutputDevice(Roll_R_PWMB, True, 0, 50)
GPIO.setup(Roll_L_AIN1, GPIO.OUT)
GPIO.setup(Roll_L_AIN2, GPIO.OUT)
GPIO.setup(Roll_R_BIN1, GPIO.OUT)
GPIO.setup(Roll_R_BIN2, GPIO.OUT)

# Jumping Motor Driver Pinouts
Jump_STBY = 20
Jump_PWMB = 19
Jump_BIN1 = 4
Jump_BIN2 = 23
# Jumping Motor Setup
GPIO.setup(Jump_STBY, GPIO.OUT)
```

```
JMotor = PWMOutputDevice(Jump_PWMB, True, 0, 50)
GPIO.setup(Jump_BIN1, GPIO.OUT)
GPIO.setup(Jump_BIN2, GPIO.OUT)


# Jumping Mechanism System

GPIO.output(Jump_BIN1, True)
GPIO.output(Jump_BIN2, False)
GPIO.output(Jump_STBY, True)

for i in range (5,11):

        JMotor.value = i/10
        time.sleep(0.06)

time.sleep(0.4)
JMotor.value = 0

GPIO.output(Roll_L_AIN1, True)
GPIO.output(Roll_L_AIN2, False)

GPIO.output(Roll_R_BIN1, True)
GPIO.output(Roll_R_BIN2, False)

GPIO.output(Roll_STBY, True)

##for i in range (1,9): #9
##
##        LMotor.value = i/10
##        RMotor.value = i/10
##        time.sleep(0.015) #0.015
LMotor.value = 0.7
RMotor.value = 0.7
#time.sleep(0.10)
time.sleep(0.05)

GPIO.output(Jump_BIN1, False)
GPIO.output(Jump_BIN2, True)
JMotor.value = 5/10
time.sleep(0.30)

GPIO.output(Roll_L_AIN1, False)
GPIO.output(Roll_R_BIN1, False)
```

```
#GPIO.output(Roll_STBY, False)

GPIO.output(Jump_BIN2, False)
#GPIO.output(Jump_STBY, False)
GPIO.output(Roll_L_AIN2, False)
GPIO.output(Roll_R_BIN2, False)
#GPIO.output(Roll_STBY, False)
```

Power Data Acquisition Code

```
import time
from time import sleep
from ina219 import INA219

import os, shutil
from datetime import datetime

#remove previous file.
shutil.rmtree('/home/pi/Desktop/PowerDataPackage/')
path = "/home/pi/Desktop/PowerDataPackage"
os.mkdir(path)

ina = INA219(shunt_ohms = 0.1,
          max_expected_amps = 3,
          address = 0x40)

ina.configure(voltage_range = ina.RANGE_16V,
           gain = ina.GAIN_AUTO,
           bus_adc = ina.ADC_128SAMP,
           shunt_adc = ina.ADC_128SAMP)

timestr = time.strftime("%Y%m%d_%H%M%S")
filename = 'PowerDAQ_' + timestr + '.csv'


file = open("/home/pi/Desktop/PowerDataPackage/"+filename, "a")

if
os.stat("/home/pi/Desktop/PowerDataPackage/"+filename).st_size
== 0:
    file.write("Time\tVoltage\tCurrent\tPower\n")

current_Time = time.time()

try:

    while (round(time.time()-current_Time,3) <= 60):
```

```
        file.write( str(round(time.time()-
current_Time,3))+"\t"+str(round(ina.voltage(),3))+"\t"+str(round
(ina.current(),3))+"\t"+str(round(ina.power(),3))+"\n")


except KeyboardInterrupt:
    pass

file.close()

exit (0)
```

Jumping Rover 2's Arduino Sketches

Jumping Rover 2 Main Code

```
///////////////////////////////////////////////////////////////
/////////
/*
    Jumping Rover Code
*/
///////////////////////////////////////////////////////////////
/////////

#include <avr/io.h>
#include <string.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <Adafruit_INA219.h>
#include <Servo.h>
#include <Wire.h>
#include "config_JR.h"
#include "serial_JR.h"


Adafruit_INA219 ina219;

//Timer2 Overflow Interrupt for periodic telemetry
ISR(TIMER2_OVF_vect){

  TCNT2 = 100; // timer set for 10 ms
  ovf_idx++;
  if(ovf_idx == 18){ // 10ms * 20 = 200ms == 0.2s, temporality
set to 1s
    telemetryready = true;
     }
  if(ovf_idx > 19){
    sendtelemetry = true;
    ovf_idx = 0;
          count++;
  }
}

void setup(){
```

```
  // serial communication
  Serial.begin(9600);
  // Timer 2 Overflow
  Timer2Init();

  Wire.begin();
  ina219.begin();

  digitalWrite(STBY, HIGH);
  pinMode(13, OUTPUT);

}

void loop(){

  // trasmit telemetry
  if(telemetryready == true){
    telemetryready = false;
    VCRead();
  }

  if(sendtelemetry == true){
    sendtelemetry = false;
    checksum = 0x00;
    for(chk_idx = 2; chk_idx < 8; chk_idx++){
      checksum ^= telemetry[chk_idx];
    }
    telemetry[8] = checksum;
    //Serial.flush();
    for (int i=0; i <= 8; i++){
      Serial.write(telemetry[i]);
    }
  }

  // receive command
  trigger = 0;
  while(Serial.available() > 0){
    temp = Serial.read();
    delay(4);

    if (temp == 0x88){
      trigger = 1;
```

```
    }

    if (trigger == 1){
      cmd[cmd_idx] = (unsigned char)(temp);
      cmd_idx++;

    }

    if(temp == 0xFF){
      receiveCOMPLETE = true;
      //Serial.flush();
      trigger = 0;
      cmd_idx = 0;
      break;

    }
     }

  if(receiveCOMPLETE == true){
    receiveCOMPLETE = false;
    header_cmd = cmd[0];
    basic_cmd = cmd[1];
    jump = cmd[2];
    PWML_val = cmd[3];
    PWMR_val = cmd[4];
    DIRL = cmd[5];
    DIRR = cmd[6];
    TBD = cmd[7];
    end_cmd = cmd[8];



  }
    if (header_cmd != 0x88){
      //if robot id does not equal 136 (88 in HEX), do nothing
    }
    else if (basic_cmd > 0x00 && basic_cmd < 0x0B ){ //basic
commands for test_serial in MATLAB
      basicMtrCtrl(basic_cmd);
    }
    else{ //normal controls for Rover_Follow in MATLAB
      normMtrCtrl(jump, PWML_val, PWMR_val, DIRL, DIRR);
    }
```

```
    header_cmd = 0x00;
    basic_cmd = 0x00;
    jump = 0x00;
    PWML_val = 0x00;
    PWMR_val = 0x00;
    DIRL = 0x00;
    DIRR = 0x00;

}
```

Jumping Rover 2's Configuration Code

```
#ifndef __config_JR_H__
#define __config_JR_H__

// define Rover IO pins
// A6, A7 are for I2C Communication
#define VOLT2 A6
#define CURR2 A7

//digital
//TB6612FNG pinouts
#define AIN1 4 //Left For.
#define AIN2 2 //Left Rev.
#define PWML 6 //Left PWM (for AIN1 and AIN2)
#define BIN1 7 //Right For.
#define BIN2 8 //Right Rev.
#define PWMR 5 //Right PWM (for BIN1 and BIN2)
#define STBY 3 //STBY

//servo pinout
Servo PWMV;

//motors
#define MINSPD 75
#define MEDSPD 110
#define MAXSPD 255

//servo speed
#define TRNSPD 1465
#define STLSPD 1514
#define STOPSPD 1515

//variables
//motor control
unsigned char PWMV_val;
unsigned char PWML_val;
unsigned char PWMR_val;
unsigned char DIRL;
unsigned char DIRR;
unsigned char jump;

#endif
```

Jumping Rover 2's Timer Interrupt Code

```
/* initialization */

void Timer2Init(void){

    // timer2 configuration for 10ms
    TCCR2B = 0x00;              // normal mode, timer2 stop
    TCNT2 = 100;                // initial value of timer2

    // timer2 interrupt configuration
    TIFR2  = 0x00;              // clear timer overflow flag
    TIMSK2 = (1<<TOIE2);        // enable timer2 overflow
interrupt
    TCCR2A = 0x00;              // normal wave gen mode
    TCCR2B |= (7<<CS20);        // set scale factor 1024, start
timer2

}
```

Jumping Rover 2's Serial Initialization Code

```
#ifndef __serial_JR_H__
#define __serial_JR_H__

#define HEADER01 0xA5                       // type of vehicle
#define HEADER02 0x05                       // vehicle number
#define NUM_DATA 0x06                       // number of data

// System Baud Rate
#define BR9600      9600
#define BR19200     19200
#define BR38400     38400
#define BR57600     57600
#define BR74880     74880
#define BR115200    115200

// initialize Timer2
void Timer2Init(void);

// periodic telemetry variables
volatile unsigned char ovf_idx;
bool telemetryready;
volatile bool sendtelemetry;
volatile unsigned char count;

//servo variables
unsigned long time1; //difference in time
unsigned long time2; //difference in time
bool jumpAct; //toggle servo jump activation
bool detachAct; //toggle detach servo pin activation

//command values
volatile unsigned char telemetry[30];                     //
telemetry packet
unsigned char nbyte;
unsigned char checksum;                           // checksum
unsigned char chk_idx;
unsigned char temp;
unsigned char trigger = 0;
volatile unsigned char cmd[9];                      // command
unsigned char cmd_idx = 0;                      // command index
```

```
unsigned char header_cmd;                                   // header of
command
unsigned char basic_cmd;                                    // basic
command
unsigned char TBD;                                         // extra
command value
unsigned char end_cmd;                                      //end
command
unsigned char val_cmd[10];                                    // value
of command
bool receiveCOMPLETE = false;

#endif
```

Jumping Rover 2's Power Data Acquisition Code

```
void VCRead(){
    uint16_t Vraw;
    uint16_t Iraw;


    float v = 5.03;
    float cur = 1023.34;
    //Vraw = (uint16_t)(v*100);
    //Iraw = (uint16_t)(cur);

    Vraw = (uint16_t)(ina219.getBusVoltage_V()*100);
    Iraw = (uint16_t)(ina219.getCurrent_mA());
    if (Vraw < 0){
      Vraw = 0;
    }
    else if (Iraw < 0){
      Iraw = 0;
    }

    telemetry[0] = HEADER01;
    telemetry[1] = (unsigned char)(Vraw >> 8);
    telemetry[2] = (unsigned char)(Vraw & 0xFF);
    telemetry[3] = (unsigned char)(Iraw >> 8);
    telemetry[4] = (unsigned char)(Iraw & 0xFF);
    telemetry[5] = 0x00;
    telemetry[6] = 0x01;
    telemetry[7] = 0x02;
}
```

Jumping Rover 2's Motion Control Code

```
void normMtrCtrl(int jump, int PWMRL_val, int PWMR_val, int
DIRL, int DIRR){
  //Left motor direction
  if (DIRL == 2){                  //Left forward
    digitalWrite(AIN1, HIGH);
    digitalWrite(AIN2, LOW);
  }
  else if (DIRL == 0){             //Left neutral
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, LOW);
  }
  else if (DIRL == 1){             //Left reverse
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
  }

  //Right motor direction
  if (DIRR == 2){                  //Right forward
    digitalWrite(BIN1, HIGH);
    digitalWrite(BIN2, LOW);
  }
  else if (DIRR == 0){             //Right neutral
    digitalWrite(BIN1, LOW);
    digitalWrite(BIN2, LOW);
  }
  else if (DIRR == 1){             //Right reverse
    digitalWrite(BIN1, LOW);
    digitalWrite(BIN2, HIGH);
  }

  analogWrite(PWML, PWML_val);
  analogWrite(PWMR, PWMR_val);

  if (jump == 0xFE){
    PWMV.attach(9);
    PWMV.writeMicroseconds(TRNSPD); //DELAY NOT YET TESTED FOR
SPRING COMPRESSION
    delay(1750);
    PWMV.writeMicroseconds(STOPSPD);
    delay(500);
  }
}
```

Appendix B. Computer-Aided Design Drawings

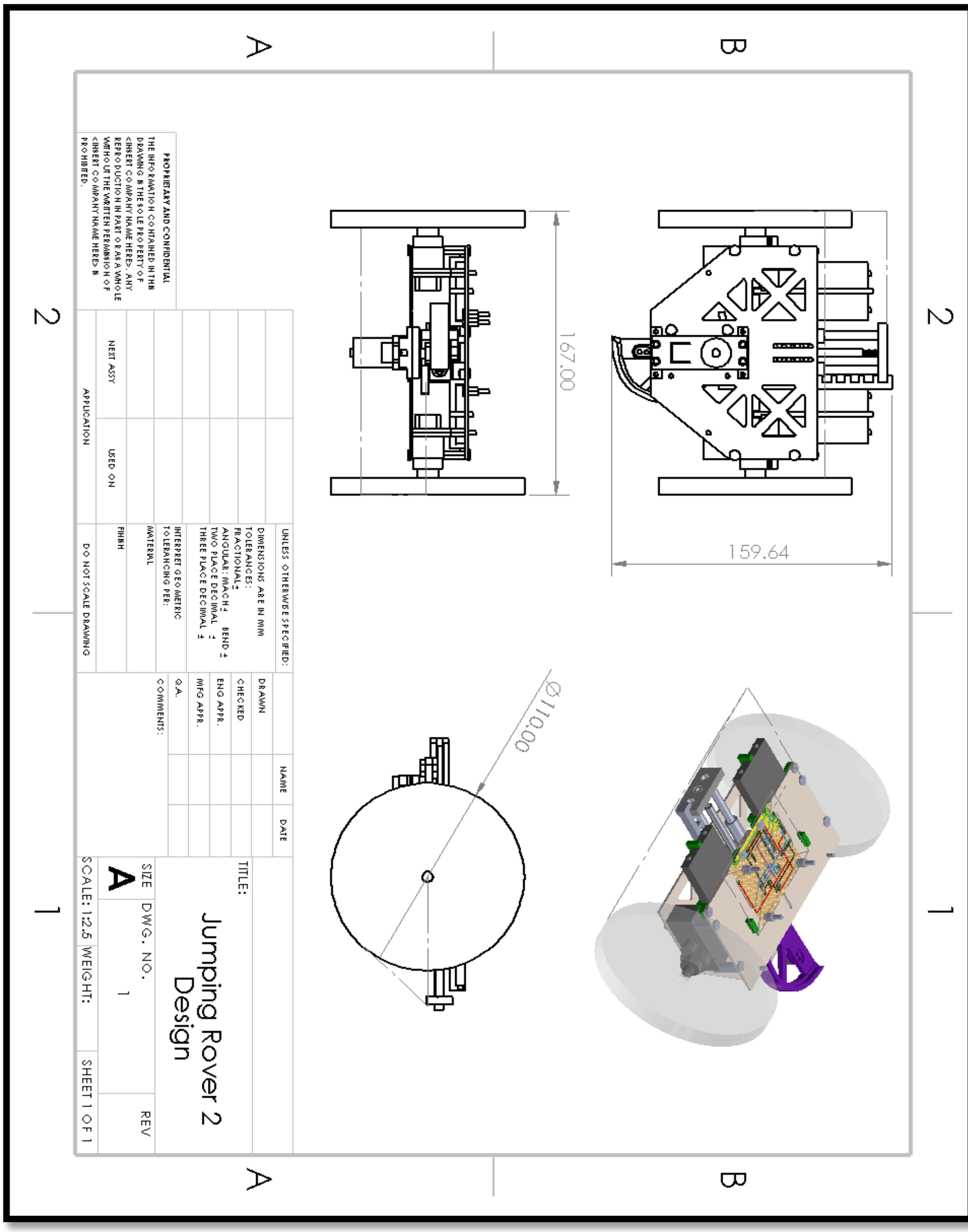

Figure B.1: Jumping Rover 2 Design CAD Drawings

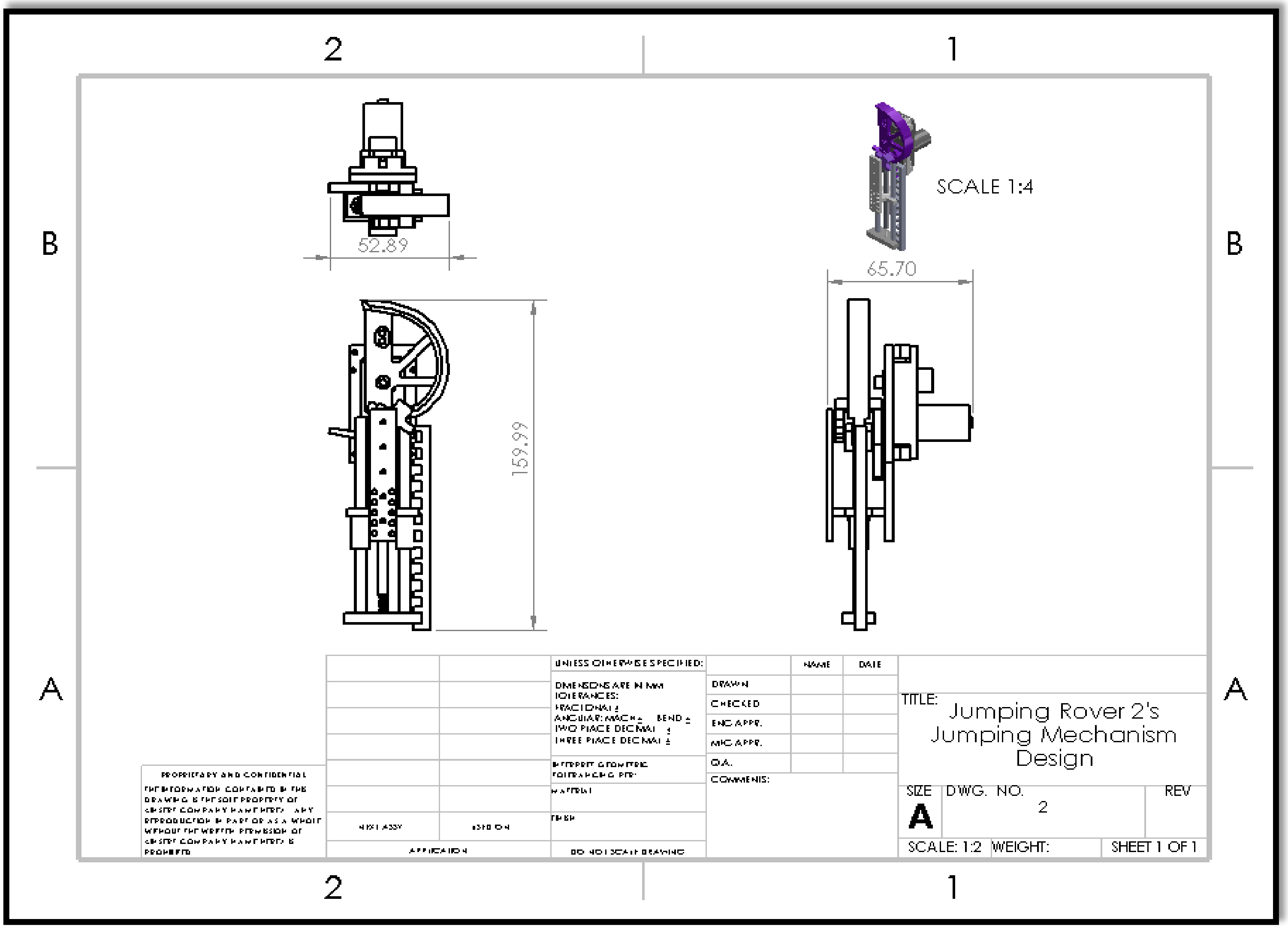

Figure B.2: Jumping Rover 2's Jumping Mechanism Design

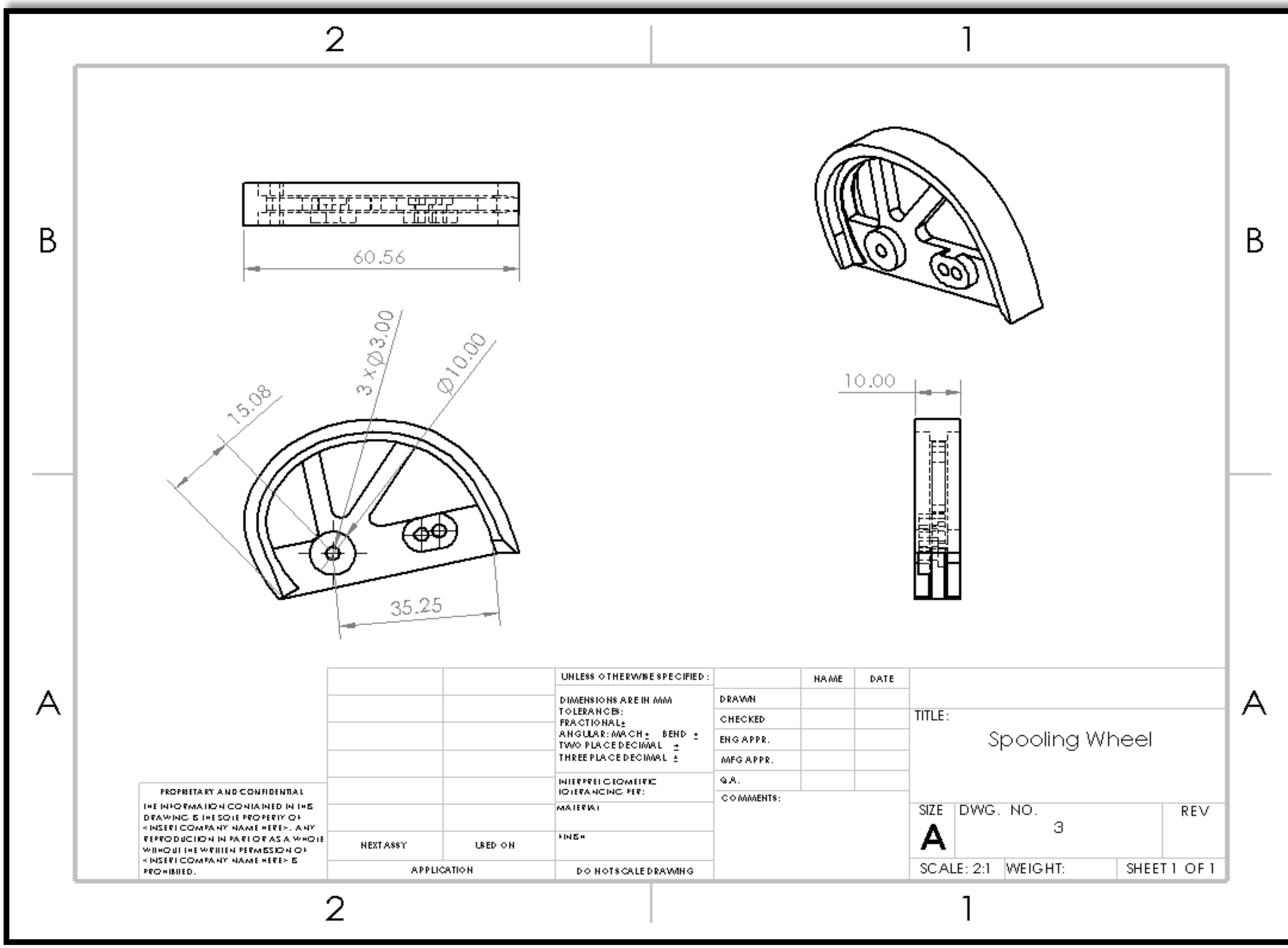

Figure B.3: Spooling Wheel Design CAD Drawing

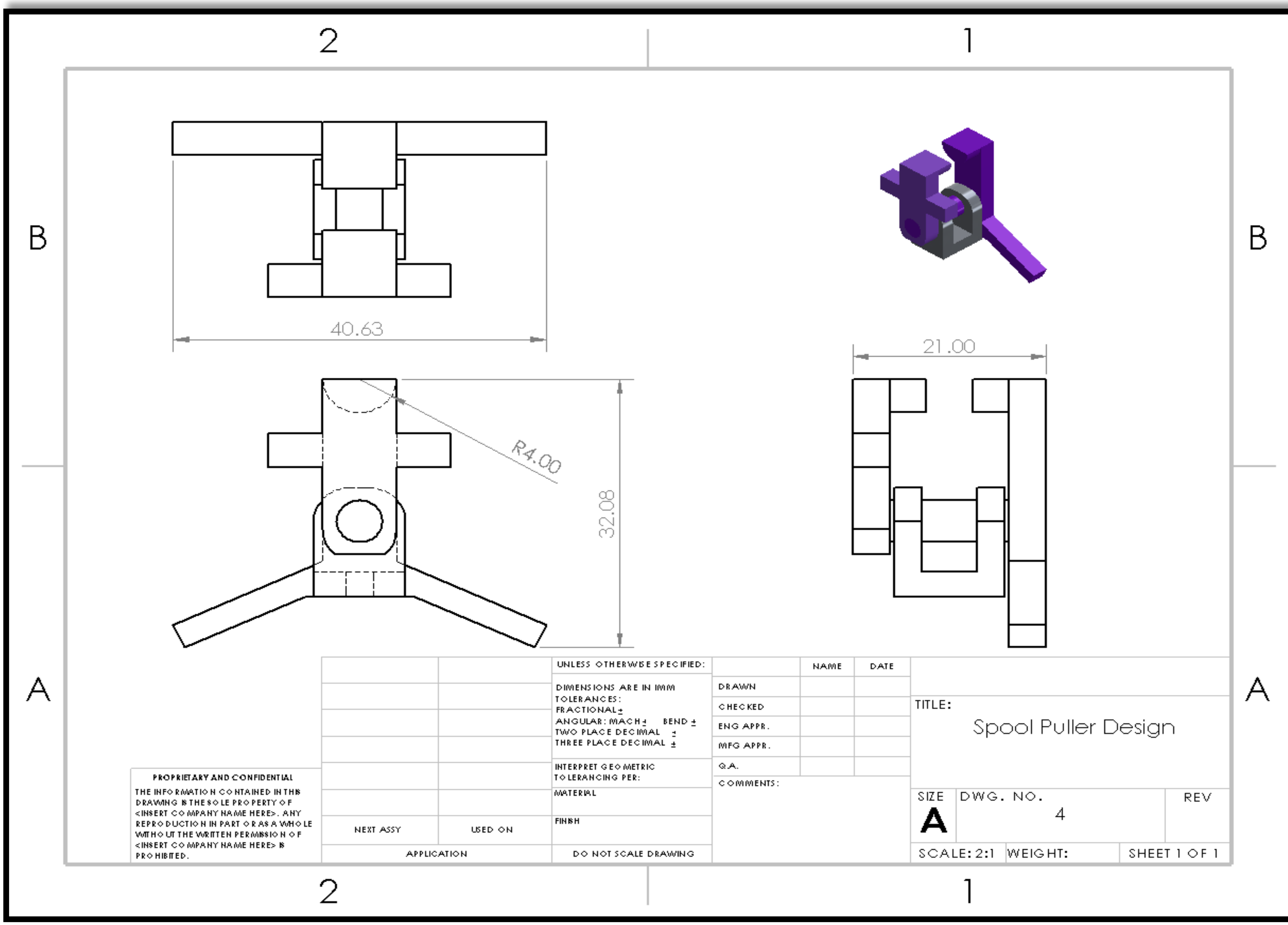

Figure B.4: Spool Puller Design CAD Drawing

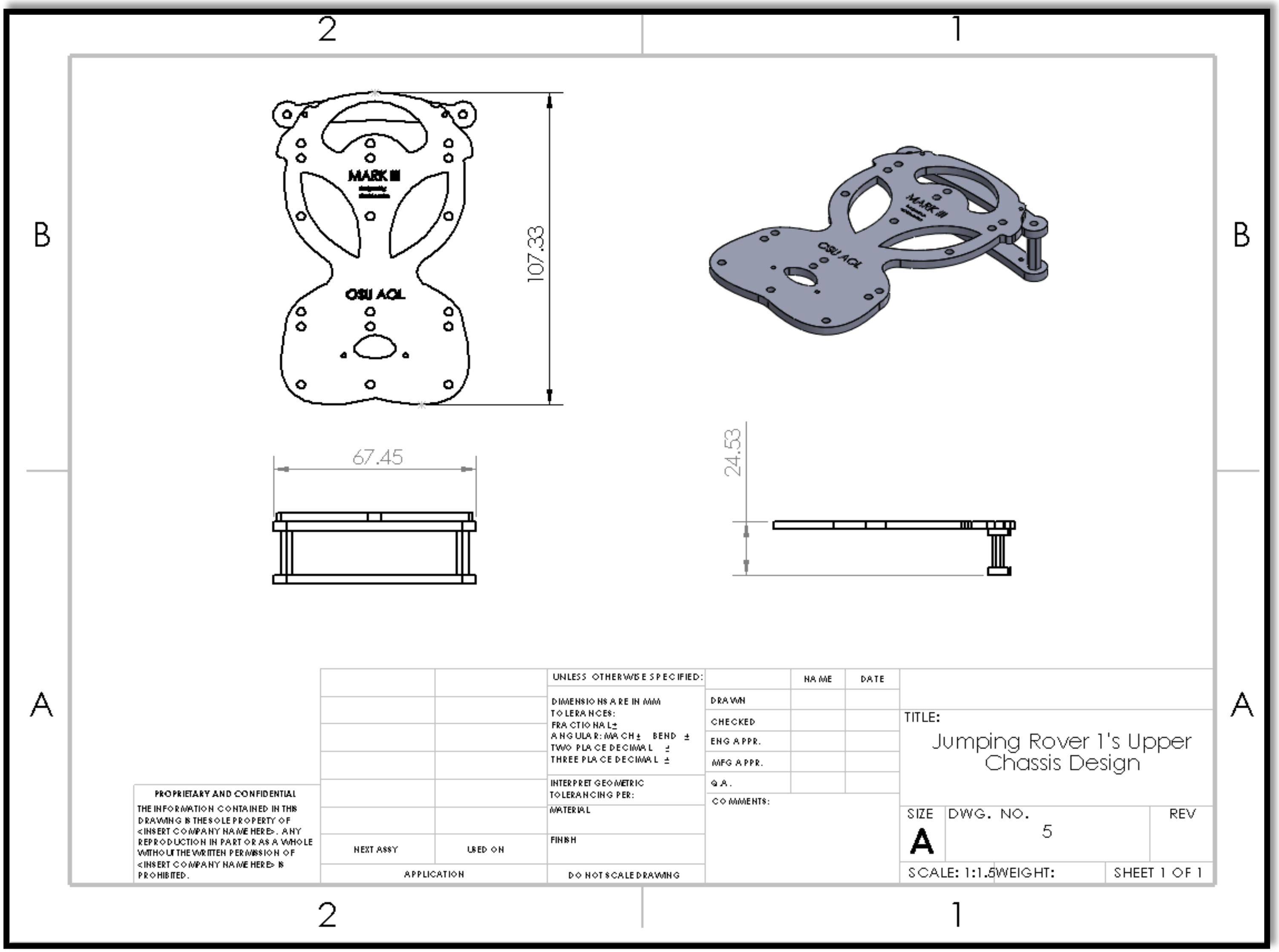

Figure B.5: Jumping Rover 1's Upper Chassis CAD Drawing

Appendix C. Additional Research Images and Tables

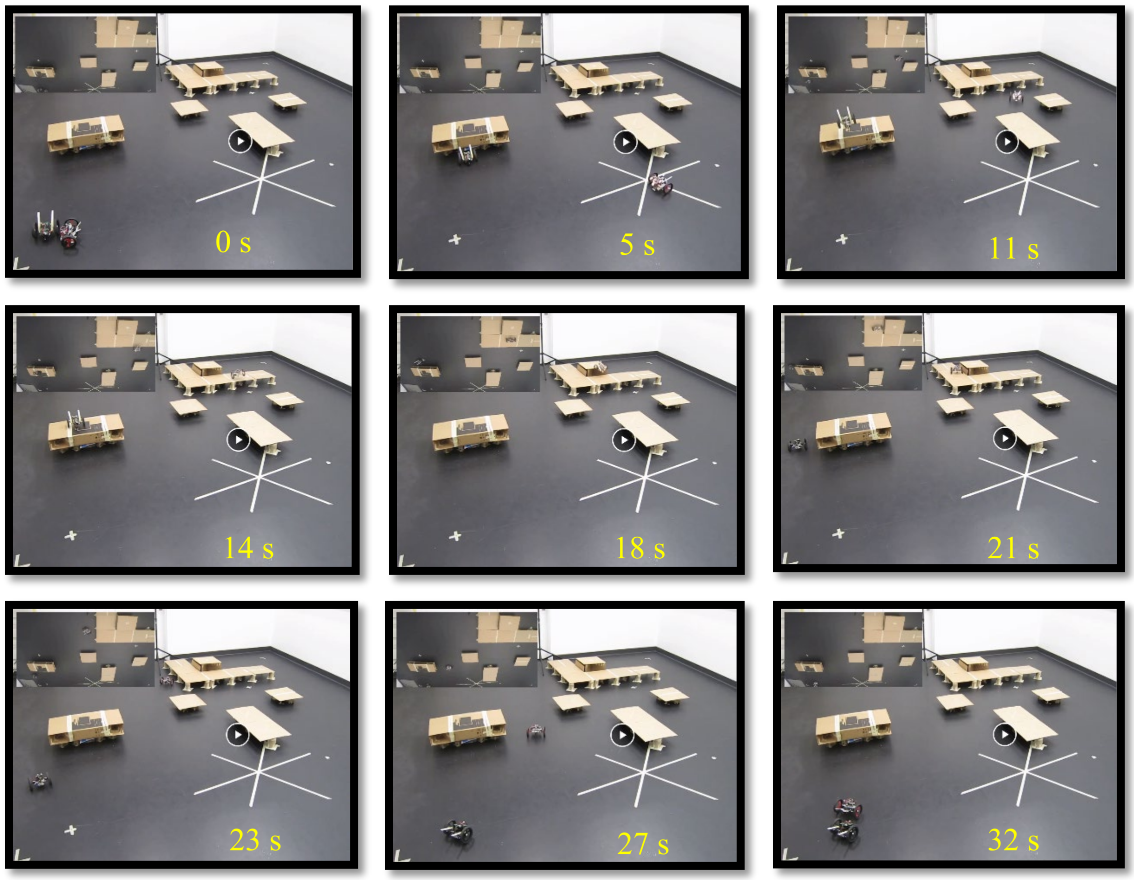

Figure C.1: Sequential Time-lapse of the Jumping Rover Team Performing the Physical Experiment of the mTSP Mission

Table C.1: Jumping Rover 1's Assigned Coordinates for the Experimental Test

x-coordinate [mm]	y-coordinate [mm]	Jumping Signal (1 - Jump, 0 – No Jump)
600	1200	0
600	2000	1
20	1900	0
20	1200	0
400	200	0
400	50	0

Table C.2: Jumping Rover 2's Assigned Coordinates for the Experimental Test

x-coordinate [mm]	y-coordinate [mm]	Jumping Signal (1 - Jump, 0 – No Jump)
400	200	0
3000	400	0
3000	1800	0
3200	2350	0
3195	2720	1
2857	2762	0
2900	2900	0
2863	3123	1
2400	3200	0
2400	2800	0
2000	2800	0
1660	2300	0
1110	890	0
400	200	0