2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Motion Planning and Task Allocation for a Jumping Rover Team

Kai Chuen Tan, Myungjin Jung, Isaac Shyu, Changhuang Wan, and Ran Dai

***Abstract*—This paper presents a cooperative robotic team composed of unmanned ground vehicles (UGVs) with hybrid operational modes to tackle the multiple traveling salesman problem (mTSP) with obstacles. The hybrid operational modes allow every UGV in the team to not only travel on a ground surface but also jump over obstacles. We name these UGVs jumping rovers. The jumping capability provides a flexible form of locomotion by leaping and landing on top of obstacles instead of navigating around obstacles. 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. We then formulate the mTSP as a mixed-integer linear programming (MILP) problem to search for the most cost-effective combination of paths for multiple UGVs. The effectiveness of the hybrid operational modes and optimized motion with assigned tasks is verified in an indoor, physical experimental environment using the customized jumping rovers.**

***Index Terms*—Jumping Robots, Multiple Traveling Salesman Problem, Path Planning, Rapidly-exploring Random Tree, Mixed-Integer Linear Programming**

I. Introduction

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 vehicles (UAVs and UGVs), respectively. UAVs and UGVs both have their benefits and drawbacks. For example, while UAVs can generally move more quickly than UGVs, they require more control efforts in unfavorable atmospheric conditions and regulations from Federal Aviation Administration make them infeasible to be operated in civil environments without permissions. In contrast, while UGVs are able to explore tight spaces without the hindrance of rotating propellers, they cannot overcome major changes in increased elevation. As a result, there requires a solution to tackle the issues that both UAVs and UGVs are facing.

The research was supported by NSF grant ECCS-1453637. Kai Chuen Tan, Myungjin Jung, Isaac Shyu, Changhuang Wan, and Ran Dai are with the Mechanical and Aerospace Engineering Department, The Ohio State University, Columbus, Ohio, 43210. Emails: tan.783, jung.661, shyu.21, wan.326 and dai.490@osu.edu

One possible solution is the wheeled jumping robots. The development of robotic jumping mechanisms has been extensively investigated. These methods include the deformation of wheels [1] and the shifting of internal masses [2]. Given that a robot can both attain forward movement using both rotating wheels and 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 capabilities of jumping onto or over an obstacle below certain elevations. Much of the existing studies on the control of a jumping robot entail the precise operation of motion. This includes legged motion with calculations into speed and torque [3] and motion control of a jumping robot that uses inertial force with a tail as its jumping mechanism [4]. The existing studies for motion planning of a jumping robot focus on finding an optimal landing position for a jumping robot by prioritizing safety and minimizing the cost of jumping [5]. Although obstacles have been considered in [5] when planning the jumping motion, each obstacle is treated as a point that cannot be used as a suitable landing surface.

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 [6]. 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 the 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) [7].

Approaches to solving the TSP have been investigated over the years ranging from the use of genetic algorithms in an iterative approach [8] to the optimization of multiple simultaneous TSPs [9]. The work in [10] 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 [10] 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 [10] 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 obstacles, which enables a traversable terrain for the entire operating environment. As a result, an optimal path for each jumping rover can

978-1-7281-7395-5/20/$31.00 ©2020 IEEE

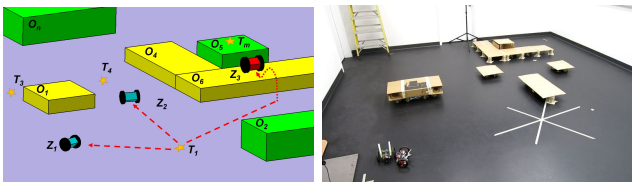
be planned in the mTSP by taking into account traditional avoidance procedures, as well as creating other possibilities for each obstacle encountered. Although many algorithms for motion planning with avoidance zones have been developed, e.g., particle swarming optimization and variations of genetic algorithm, none of these works consider an obstacle in the operating environments as a possible pathway using both the jumping and rolling mechanisms.

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. Targets may be located on top of an obstacle. 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.

The paper is organized as follows. §II presents the problem statement, §III describes the motion planning and task allocation algorithm. §IV displays the indoor experimental environments and construction of jumping rovers. The simulation and experimental results are presented in §V. Lastly, we address the conclusions and future work in §VI.

II. 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 $T = T_{\{1,2,\ldots,\, m\}}$, where m is the total number of targets within an operating area; the UGVs are indexed as $z = 1, 2, ..., p$. 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 targets, we consider obstacles randomly scattered in the area, denoted by $O = O_{\{1,2,\ldots,\, n\}}$, where n is the total number of obstacles. Partial of the targets are assigned on top of the obstacles. Each obstacle is a rectangular prism with leveled surface, and its length, width and height dimension are given. They are treated as solid objects that cannot be passed through and cannot be moved. They cannot overlap each other, however, their boundaries can intersect adjacent one such as O_4, O_5, and O_6 shown in Fig. 1a. Targets can be located anywhere in the mission area and may appear on top of obstacles, such as T_m in Fig. 1.



(a) Mission overview of mTSP (b) Experimental environments

Fig. 1: Scenario and experimental environments of mTSP mission

The cost to be optimized in the mTSP of a jumping rover team can be a single performance index, e.g., time, energy, and distance, or a combination of weighted performance indices. As an example, we consider energy consumption as the cost to be minimized in the mTSP. Consider a single UGV that operates with two operational modes, rolling of wheels attached to the chassis and jumping motion through the actuation of a spring. With a constant velocity during straight forward rolling and constant angular speed during zero radius rotating for UGV z, $z = 1, \ldots, p$, the power consumption rate of the jumping rover in straight forward motions is denoted by P_z^l. 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, $i, j = 1, \ldots, m$, $i \neq j$, is determined by

$$c_{ij,z} = (P_z^l + P_z^a) t_{ij,z}^l + E_z^j n j_{ij,z}, \tag{1}$$

where $t_{ij,z}^l$ is the time duration along the rolling motion between targets i and j, $nj_{ij,z}$ is the overall number of jumps for UGV z, $z = 1, \ldots, p$, between targets i and j. By summarizing the energy consumption of all jumping rovers of all paths between any two targets, we can find the overall UGV team energy usage during the mTSP mission.

III. Motion Planning and Task Allocation

A. Refined RRT*

Developing the path planning algorithm for rovers that have both rolling and jumping capabilities needs to consider multiple pathway options including avoiding and jumping onto obstacles. Unlike any other traditional UGVs, obstacles can be considered as a feasible path unless the obstacles' height is beyond the jumping capability of jumping rovers. There are two options to consider when determining the most cost-efficient pathway from one target to another, which are the avoidance option and jumping option. The pathways of both options are generated separately with an improved version of a sampling-based heuristics search algorithm, named refined RRT*.

RRT algorithm is a path-planning tool, initially developed by S. LaValle [11], to quickly cover a given area using the algorithm that grows into a tree-like shape. 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 without violating the collision constraint. The process is repeated for the number of desired iterations, N. If the path-finding problem provides

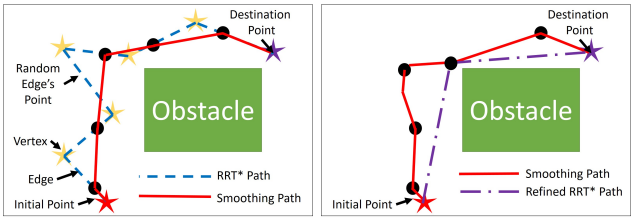
a final destination point, x_{dest}, then x_{dest} connects to their nearest random point of a generated tree without violating the collision constraint before the function terminates. While effective in finding a solution with great speed, RRT cannot guarantee that solutions are efficient in terms of the length of the tree path from x_{init} to x_{dest}. Thus, RRT* has been generated to find an optimal solution between the points in a tree. RRT* takes each point in a tree, finds the points within a radius of each point, and replaces existing edges to that point with the most efficient path without violating constraints.

Algorithm 1 : Refined-RRT* Algorithm

1 **function** RRRTS($x_{\text{init}}, x_{\text{dest}}, \Delta x, K$)
2 R.initialize($x_{\text{init}}, x_{\text{dest}}$)
3 **for** $k \leftarrow 1, K$ **do**
4 $x_{\text{rand}} \leftarrow$ Random_State()
5 $x_{\text{near}} \leftarrow$ Nearest_Point(x_{rand}, R)
6 $x_{\text{new}} \leftarrow$ New_Config($x_{\text{rand}}, R, \Delta x$)
7 **if** CollisionFree **then**
8 $x_{\text{near}} \leftarrow$ Near($R_t, x_{\text{new}}, |vertices|$)
9 $x_{\text{min}} \leftarrow$ Parent($x_{\text{near}}, x_{\text{nearest}}, x_{\text{new}}, x_{\text{dest}}$)
10 $R \leftarrow$ Rewire($R, x_{\text{near}}, x_{\text{nearest}}, x_{\text{new}}, x_{\text{dest}}$)
11 **end if**
12 **end for**
13 **for** $w \leftarrow 1, S$ **do**
14 $p_1, p_2 \leftarrow$ Random_Edges_Vertex(R)
15 **if** CollisionFree **then**
16 $R \leftarrow$ Path_Smoothing(p_1, p_2, R)
17 **end if**
18 **end for**
19 **while** CollisionFree **do**
20 $R \leftarrow$ Vertices_Elimination(R)
21 **end while**
22 **return** R
23 **end function**

However, 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, we propose the refined RRT* method with the process shown in algorithm 1 from line 13 to line 22. Through the refinement, we aim to shorten the final path and make it more energy efficient by excluding unnecessary tree segments through smoothing and elimination process. The smoothing process picks two random points on two distinct edges that links RRT* vertices and creates a smoother path that does not violate the collision constraint, then a smoother path is generated as illustrated on Fig. 2a. The smoothing process continues until it reaches the desired number of smoothing iterations, S.

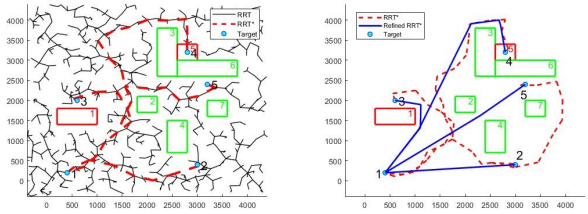
In the elimination process, a new edge that connects the first and third vertices is generated and evaluated to determine whether the newly generated edge violates the collision constraint. If the edge does not collide with any obstacles, the second vertex together with the edges that links to it will be eliminated, and the third vertex will become the second one. The elimination process continues until there are no more than three vertices left. On the other hand, if the edge collides with an obstacle, another newly generated edge that links the last vertex and the third vertex from the last will be evaluated with the collision constraint; if the edge does not collide with any obstacles, the second vertex from the last together with the edges that links to it will be eliminated, and the third vertex from the last will become the second vertex from the last. The elimination process repeats until either there are no more than three vertices left in a tree or the evaluated edge violates the collision constraint. After the smoothing and elimination processes are completed, the refined RRT* path is produced as shown on Fig. 2b.



(a) Path Smoothing (b) Vertices Elimination

Fig. 2: Refining Process for RRT*

For the example in Fig. 3, we have 1000 samples for RRT and 500 mm neighbor radius for RRT*. The series of red dash lines indicate the path-planning formed by RRT* to reach other targets from Target 1. The trees are optimized by the rewiring process to eliminate unnecessary or higher cost nodes from the RRT process. Then, our refining process continues with other trees formed between original targets by smoothing and eliminating. The algorithm produces better trajectories by considering the gap between the rover and the obstacle to avoid the collision. These gaps are determined by the geometry and maneuverability of the rover. In light of this, the refined RRT* is applied to make all paths segments traversable.

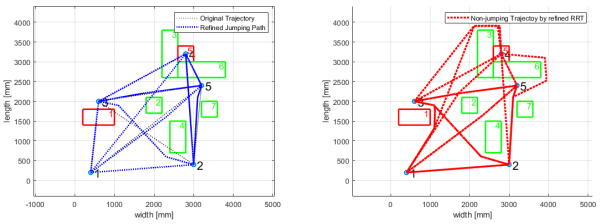


(a) RRT / RRT* from target 1 (b) Refined RRT* from target 1

Fig. 3: Refining all possible avoidance trajectories

Both avoidance and jumping options use the refined RRT* to generate all possible paths separately. For the avoidance option, an elevated path is treated as a traversable path only when a target like T_4 is set on top of an obstacle as shown in the right side of Fig. 4b. Otherwise, an elevated path is treated as an infeasible path.

For the jumping option, all elevated paths are treated as a traversible path as long as the obstacle' height is not beyond the rover' jumping capability. After generating paths from the



(a) Paths for jumping option (b) Paths for avoidance option

Fig. 4: All possible trajectories generated from refined RRT*

refined RRT*, a further paths' refinement is required for the jumping option since obstacles cannot become circumvention area when the path lies on the obstacles without sufficient width for the rolling motion of the certain rover, then the detoured trajectory for the obstacle can be developed. Due to different power consumption rate, the cost of the same trajectory may be distinct for different rovers. With all the possible paths including avoidance and jumping trajectories, the most cost-efficient path between two targets is selected for each pair of target combinations by comparing the trajectories from both avoidance and jumping options.

B. Task Allocation via MILP

From the refined RRT*, we find the energy-efficient paths between any two visiting targets. Next, we formulate the mTSP as a MILP to find the best combination of all 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 there is only one jumping rover, i.e., $p = 1$.

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 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

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

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

$$\min \sum_{i=1}^{m} \sum_{j=1, j\neq i}^{m} \sum_{z=1}^{p} c_{ij,z} x_{ij,z} \tag{2}$$

$$s.t. \sum_{i=1, i\neq j}^{m} \sum_{z=1}^{p} x_{ij,z} = p, \ j = 1 \tag{3}$$

$$\sum_{j=1, j\neq i}^{m} \sum_{z=1}^{p} x_{ij,z} = p, \ i = 1 \tag{4}$$

$$\sum_{i=1, i\neq j}^{m} \sum_{z=1}^{p} x_{ij,z} = 1, \ j = 2, ..., m \tag{5}$$

$$\sum_{j=1, j\neq i}^{m} \sum_{z=1}^{p} x_{ij,z} = 1, \ i = 2, ..., m \tag{6}$$

$$\sum_{i=1, i\neq s}^{m} \sum_{z=1}^{p} x_{is,z} - \sum_{j=1, j\neq s}^{m} \sum_{z=1}^{p} x_{sj,z} = 0, \ s = 1, ..., m \tag{7}$$

$$u_i - u_j + q \sum_{z=1}^{p} x_{ij,z} \leq q - 1, \ 2 \leq i \neq j \leq m \tag{8}$$

$$x_{ij,z} \in \{0, 1\}, \ \forall z, i, j \tag{9}$$

$$x_{ij,z} h(i, j) \leq h_z \ \forall z, i, j \tag{10}$$

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. Constraints (3) and (4) ensure exactly p UGVs return to the depot T_1 and p UGVs depart from the depot T_1, respectively. Constraints (5) and (6) ensure only one UGV enters each target and only one UGV leaves each target. Constraints (7) ensures the same rover visits and departs from a target. Constraints (8) are the extensions of Miller-Tucker-Zemlin-based subtour elimination constraints that ensure there are no sub-routes among the non-starting targets [12]. As a result, the solution returned is a single tour instead of the union of smaller tours for each UGV. The inequality in (9) represents the jumping height constraints such that the maximum elevation along edge $(i, j) \in E$ can be achieved by UGV z.

IV. Experimental Environments and Customized Jumping Rovers

A. Experimental Environments

The scenario developed for the experimental verification of the jumping rover team uses several obstacles in different shapes and sizes. 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 black font. Objects are represented as colored rectangles and numbered with corresponding colored 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 arbitrary height of 1, and space surrounded by the lines of the orange rectangle indicates an arbitrary height of 2. Experimental tests were performed indoors. Obstacles used were cardboard structures of various dimensions while targets are indicated by taped markers throughout the test area, shown in Fig. 1b.

B. Jumping Rover Design and Construction

For a jumping rover to be used in this simulation, there are multiple requirements. First and foremost, it must be capable of both rolling motion 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 a Parrot Jumping Sumo robot chassis [13]. Built from acrylic, nylon, aluminum, and 3D printed parts, the rover is ensured to be lightweight in order to achieve an acceptable jumping height. Gearboxes are used for the wheel motors to reduce the RPMs of the output. Markers for the motion tracking system, in the form of gray spheres, were arranged in an elevated, asymmetrical pattern so the location and attitude of the rover could be accurately determined. The center of mass sits on the force vector of the jumping mechanism to ensure the 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. Each jumping rover has a fixed jumping height which is determined by the stiffness of the compressed spring. Two jumping rovers with different power consumption rates and jumping heights are demonstrated in Fig. 5. For each jump, the one with higher jumping height consumes four times more power than the other one.

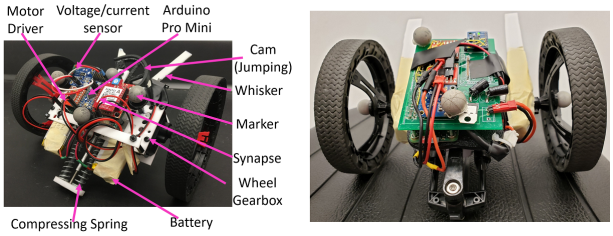


Fig. 5: Jumping rover 2 (left) with less power consumption and lower jumping height and jumping rover 1 (right) with more power consumption and higher jumping height.

The Parrot Jumping Sumo robot that can be purchased off the shelf is a manually operated robot. In addition to the design of a more robust jumping mechanism described above, we also integrate the micro-controller and communication system to achieve autonomous operation. The jumping rovers are controlled wirelessly through serial instructions with an Arduino Pro Mini via Synapse communication protocols with MATLAB and Arduino IDE. Instructions are then relayed to a motor controller that supplies power to the wheel motors. Signals for the jumping mechanism are applied directly from the Arduino to a servo motor. 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 is demonstrated. Power throughout the jumping rover electrical system is supplied by two Lithium Polymer (LiPo) batteries. Throughout the experimental tests, power data is recorded at a set frequency using the current/voltage sensor.

The Jumping Rover is instructed to follow a set of coordinates put forth by the results of the motion planning and task allocation algorithm presented in §III. Using the Vicon motion tracking system to determine location and attitude, MATLAB provides instructions to the Arduino via Synapse to dictate motor rotation.

V. SIMULATION AND EXPERIMENTAL RESULTS

To verify efficiency with the jumping option, we provide an alternative result for the mTSP where jumping is avoided unless the rover needs to reach a target assigned on top of an obstacle. The projected view of the result is shown in Fig. 7, and the alternative results without jumping options are presented in Fig. 6b, respectively, for Scenario 1. From the alternative result, it indicates that when jumping on top of an obstacle is not an option when avoiding an obstacle, rover 1 elects to take a longer path to reach target 3 and rover 2 elects to take a longer path to reach target 4. Compared to the solution in Fig. 6a with the jumping options which consumes 133.03 Joules, the solution in Fig. 6b consumes 256.59 Joules. The comparative results indicate the energy reduction of 92.88% with the jumping options for obstacle avoidance.

Although time consumption is not considered in the performance index, we compared the mission duration in both results. It indicates that using the jumping options in Fig. 6a, it takes 27 and 31 seconds, respectively, for jumping rovers 1 and 2 to finish their corresponding tasks in Scenario 1. While for the results without jumping options, it takes 16.0 and 36.0 seconds, respectively, for jumping rovers 1 and 2 to finish their corresponding tasks. Based on Table I and Fig. 6, Scenario 2 also shows that having the jumping options help to save rovers' power consumption by 178.1 % and shorten the rovers' travel completion time by 41.49 %. The time reduction for the result with jumping options verifies the byproduct of time efficiency. All simulation cases are executed on a desktop PC with an Intel Xeon CPU E5-1620v4 @ 3.50 GHz and 32GB RAM.

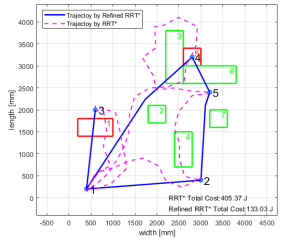
	Scenario 1		Scenario 2	
Jumping [jmp]/ Avoidance [avd]	jmp	avd	jmp	avd
Cost of RRT* [J]	405.37	353.05	479.95	818.74
Cost of Refined RRT* [J]	133.03	256.59	110.14	306.33
Time for Travel [s]	31.00	36.04	24.46	34.61
Time for computation [s]	88.05	91.58	101.59	99.82

TABLE I: Simulation Result

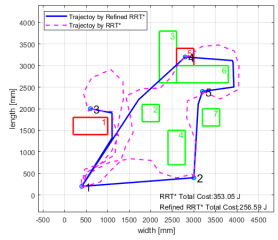
Simulation and experimental results using the two constructed jumping rovers and the scenario (i.e., Scenario 1) described in §IV are presented here. The trajectories of the simulation and experimental results of the mTSP are shown in Fig. 7. The planned paths indicate that when visiting target 3, the jumping rover 1 elects to jump on top of the obstacle in front of target 3 to reduce cost. Target 4 is located on top of an 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 Fig. 8. A video of the jumping rover team performing the physical experiment of the mTSP mission is provided.

Compared to the simulation results in Fig. 7, the experimental result shows slight differences. The major discrepancies are due to variations in location, attitude, jumping

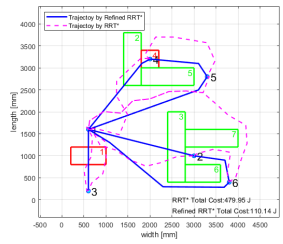
accuracy, and recovery time from jumping. This requires the rover to compensate and reach a coordinate using a different amount of power and time than originally planned. In addition, when a jumping rover is located next to an obstacle, the Vicon motion tracking system may not be able to identify all of the markers on the rover due to the blocked view, which generates navigation errors.



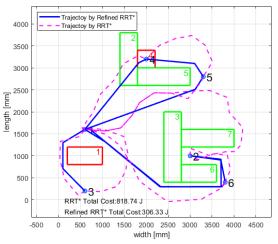
(a) Scenario 1 projected trajectories with jumping options



(b) Scenario 1 projected trajectories without jumping options



(c) Scenario 2 projected trajectories with jumping options



(d) Scenario 2 projected trajectories without jumping options

Fig. 6: Comparative results for the with and without jumping options

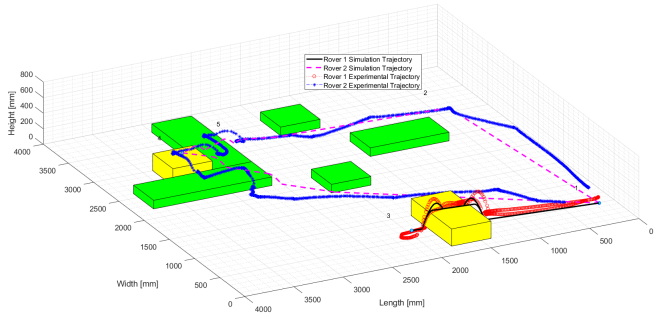


Fig. 7: Scenario 1's trajectories of the mTSP with two jumping rovers

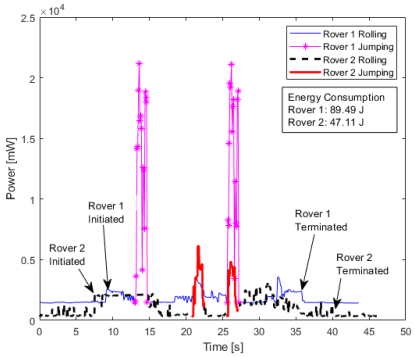


Fig. 8: Scenario 1's time history of power consumption of the two jumping rovers in the experimental test

VI. CONCLUSION

This paper 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 under certain elevations and then treats 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 paths from the refined RRT* allow 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 of using the jumping capability to solve the mTSP with obstacles and effectiveness of the proposed motion planning and task allocation method. Further work will investigate the use of multiple jumping rovers in more challenging environments, e.g., gaps and stairs, and control of jumping height.

REFERENCES

[1] 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, pp. 980–985.

[2] 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, pp. 1062–1067.

[3] 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, pp. 300–305.

[4] 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, pp. 871–876.

[5] M. Ushijima, Y. Kunii, T. Maeda, T. Yoshimitsu, and M. Otsuki, "Path planning with risk consideration on hopping mobility," in *IEEE/SICE International Symposium on System Integration (SII)*, 2017, pp. 692–697.

[6] M. J. Krieger, J.-B. Billeter, and L. Keller, "Ant-like task allocation and recruitment in cooperative robots," *Nature*, vol. 406, no. 6799, p. 992, 2000.

[7] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[8] 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, pp. 87–92.

[9] P. Kitjacharoenchai, M. Ventresca, M. Moshref-Javadi, S. Lee, J. M. Tanchoco, and P. A. Brunese, "Multiple traveling salesman problem with drones: Mathematical model and heuristic approach," *Computers & Industrial Engineering*, vol. 129, pp. 14–30, 2019.

[10] 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.

[11] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[12] C. Miller, A. Tucker, and Z. R, "Integer programming formulation of traveling salesman problems," *Journal of the ACM*, vol. 7, no. 4, pp. 326–329, 1960.

[13] "Parrot jumping sumo-races, slaloms, acrobatics, you can do it all," https://www.parrot.com/us/minidrones/parrot-jumping-sumo, 2019.