# Path Planning of a Differentially Steered Robot with Jumping Capabilities

Thomas Garnier
Dept. of Mechanical Engineering
ERAU
Daytona Beach, Florida
United States of America
garniert@my.erau.edu

Daniel Lane
Dept. of Mechanical Engineering
ERAU
Daytona Beach, Florida
United States of America
laned7@my.erau.edu

***Abstract*—In this paper, the effectiveness of two path planning algorithms for a differentially steered robot capable of jumping is being examined. The two methods being compared are: grid-based search with A* and RRT* search. Each method was modified to include jumping over or onto obstacles when possible. Also, the cost functions were adapted to adjust to an extra cost in time for the additional jumping condition (time to load the spring) and orientation for the grid. The A* algorithm gives the optimized solution based on the fastest time. Time is added based on the Euclidean distance, the orientation change, and if a jump was accomplished or not. Furthermore, random scenarios were performed and the time difference between non-jumping and jumping robots was analyzed as well. Showing that the jumping capabilities does improve the time gradually as more obstacles are present. The RRT* performed as expected, approaching but not finding an optimal path. Ten runs were performed for each scenario with 3000 nodes per run. Jumping was only found to be included in the optimal path over certain circumstances, as the additional heuristic cost was marginal.**

***Keywords—path planning, robotics, obstacle jumping, Parrot Jumping SUMO, A*, RRT****

## I. INTRODUCTION

Automation is being expanded in most industries. Companies such as IudusLabs specialize in helping companies incorporating more autonomous processes [1]. Automation can be applied in simple repeatable tasks such as manufactory the same part over and over to more complex and sophisticated robot automation as seen by self-driving cars. For independent vehicle automation, path planning algorithms are required to reduce or completely remove the need for human interference. These automations often result in having more efficient, less error, and assisting humans in the department it is used [1].

Collision avoidance is a major factor for path planning, to make sure the drones can accomplish their mission in a safe manner without collision [2]. Usually, path planning involves avoiding obstacles, but it came to our attention that several robotic platforms are capable of jumping [3]. It is intriguing to see if the capability of jumping over or on top of certain obstacles can be achieved and observe the potential advantages of doing so. One potential advantage of jumping could be applied to the autonomous vacuum cleaner that one might have home which are limited to one floor. Being able to jump at least one step could really improve the utility of such a product.

The commercially readily available Parrot jumping SUMO was examined to find optimal path techniques using its unique dynamics. SUMO is a differential driven platform manually driven by a phone or tablet show in Figure 1. The robot contains several onboard sensors, including a camera, which could lead to autonomous pathfinding possibilities with continued research. The robot itself has a 20 cm wheelbase capable of a top speed of 1.9 meters per second and can jump up to a height of 0.8 meters. With this data alone, we have been able to simulate an environment in MATLAB to examine the optimal path using A* on a grid algorithm having the roadmap allowed to go over certain obstacles and an RRT* algorithm to obtain a solution quickly and then self-optimize.

Fig 1. Display of the Jumping Sumo robot used for research

## II. METHOD

The environment was modeled to be used by two methods: grid search optimized with A* and RRT*. To compare the two models, common scenarios have been created as displayed in Figure 2. The different simulations are going to be compared on the solution or best path found and, on the time, to find a viable solution. Then, each of the models are going to be compared individually by generating random scenarios to establish benchmarks having the start and goals being constant while the number and/or size of the obstacles are randomly being assigned.

Both models were tested in the same environment size, measuring 20 x 20 meters. These dimensions were chosen to represent a living room where the robot would have to navigate. The starting position was placed at (2,2,0) and the goal placed at (18,18,0) in x-y-z, with the origin in the lower left-hand corner of the grid. The unobstructed grid was taken to be of height zero, and the obstacles consist of axis-aligned boxes of varying dimensions and heights. Please note that the obstacle heights were discretized for ease of coding, and in figures shown, the number in the center of the obstacle represents a multiple of 0.2 meters. The first environment is empty with no obstacles, the second had one 8x8 meter obstacle placed in the middle of the grid with height of 0.8 meters, the third had 3 4x4 meter obstacles directly along the vehicle path of heights 0.6, 1.2, and 1.6 meters in height, respectively. The fourth environment was designed with the starting position at (2,10,0) and the goal at (18,10,0) with three 4x4 meter obstacles between with respective heights of 0.6, 1.2, and 1.6 meters. This last environment was designed with the idea that each obstacle height is only reachable from on top of the obstacle prior to it.

SUMO can jump up to 0.8 meters but requires some time to charge the spring-loaded piston it utilizes. Therefore, we incorporated this time into our heuristic calculation, adding a 0.4 second penalty for jumping. The speed of the robot was assumed to be constant of 1.9 meters per second, and dividing the distance by the speed, we determined the heuristic for time between nodes. Each model then used the environment and heuristics to calculate the fastest path to the goal.

$$\frac{distance\ traveled\ (m)}{vehicle\ speed\ (m/s)} + (jumps) * 0.4 = time\ (s) \qquad (1)$$

The obstacles are inflated by the diameter of SUMOs wheelbase, and our robot position is assumed as a point value. As such, we had to define an inner boundary on our obstacles to be valid positions. When performing a jump, SUMO must be able to clear the top edge of the obstacle, and land within the un-inflated obstacle bounds. Additionally, we assumed that every jump moved the robot a set

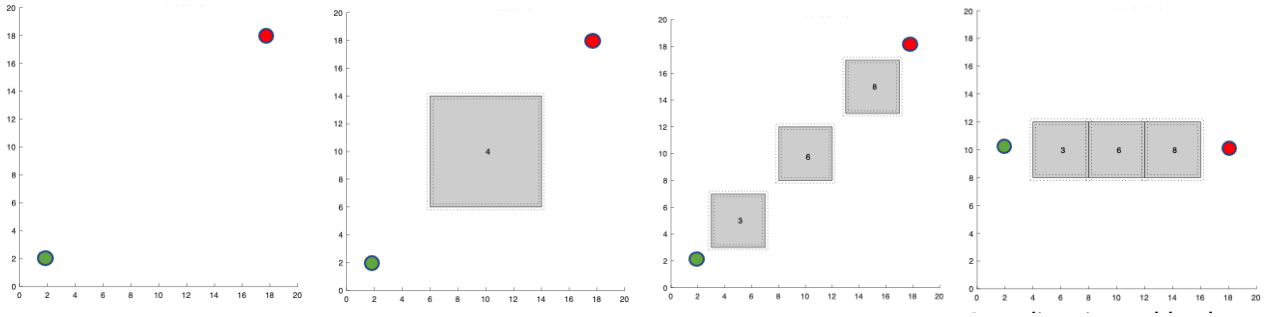

Fig. 2 standardized environments used for comparison of two methods. Green and red dots mark the start and goal respectively (enlarged for readability). The first graph is an empty environment starting at (2,2,0) and finishing at (18,18,0). The second graph has one large obstacle of height 0.8, starting at (2,2,0) and finishing at (18,18,0). The third graph has 3 smaller obstacles along the optimal path, with heights of 0.6,1.2 and 1.6, starting at (2,2,0) and finishing at (18,18,0). The last graph is a stair-step configuration of obstacles of heights 0.6,1.2 and 1.6 starting at (2,10,0) and finishing at (18,10,0).

distance, independent of the change in height achieved.

It is important to note the limitations imposed by the environment. Each obstacle is assumed to be flat, and that the robot would have a perfect

### *A. Grid Search with A**

The environment of the grid was set to be 100X100 cells, making each cell to be 20X20 cm which is the dimension of the wheelbase. Therefore, each of the cells represents the potential location of the jumping Sumo. Furthermore, for A* to be applied, it was established that the full environment was known for the simulation to run. The obstacles are inflated to automatically account for the space required to reach the top of the jump without colliding with the edge of any obstacles. Therefore, the robot is not allowed to finish a jump in the edges of the obstacles. Many conditions were imposed to know the potential connection each node can have as follows:

- The drone can have a normal 8-point connectivity as long as no obstacles are around the current state. Due to the uncertainty of where the obstacles are, no diagonal movement is allowed next to an obstacle.
  - The robot can jump only if necessary (go over or on top of obstacles when height can be reached)
  - If the Sumo is on top of an obstacle, the same 8-point connectivity applies unless different height cells are around.
  - If the drone is on the edge of an obstacle, the only roads are going down in a 4-point connectivity way

These limitations have been made to ensure that collision is not going to occur as long as the conditions are followed. Figure 3 illustrates most of the movement possibilities.

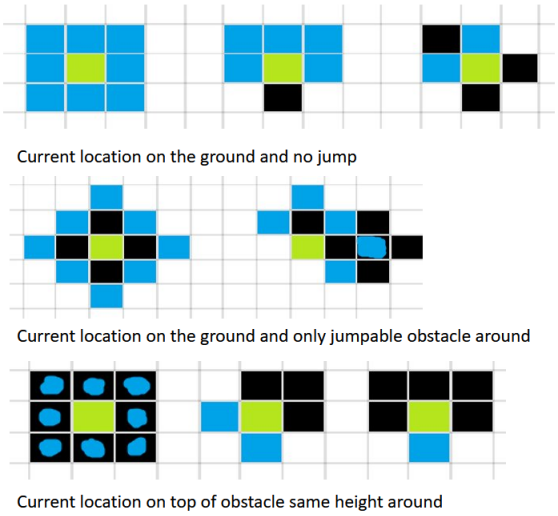


Fig. 3 *Potential movements of the Jumping Sumo. The green boxes are showing the current location, while the blue boxes are displaying the roadmaps being generated for each case with the obstacles being represented by the black boxes.*

For A* to be successful, all roadmaps are generated based on the corresponding environment created and A* algorithm finds the best solution based on the Euclidian distance between the points (height included) with additional costs whenever the robot jumps and/or turns. Any jumps have a fixed cost of 0.4 s while changing orientation is based on the change of orientation in radian times 0.026. The time cost of changing orientation was calculated based on the max speed and the perimeter obtained for a full rotation with the diameter being the wheel based. The cost function for the A* algorithms is defined as equation 2:

$$\frac{distance\ (m)}{Speed\ (m/s)} + 0.4 * jumps + 0.026\Delta\theta = time(s) \qquad (2)$$

### *B. RRT**

A standard RRT* model [4] was used for reaching the goal. The algorithm exploited towards the goal at 15% rate until the goal was found, and then switched to 100% exploration. This allowed a very quick time to find a solution to the goal and used the remaining new nodes to optimize the path. Many modifications had to be made to capture the complexities of the jumping aspect of our system dynamics.

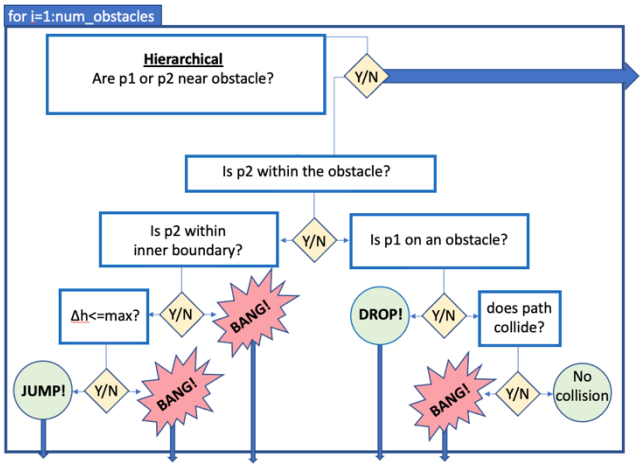


Fig. 4 *Flow chart for the obstacle avoidance loop. The outer box represents a for-loop over the number of obstacles. Arrows represent decision paths that flow out of the loop, and arrow thickness is an approximation for the volume of decisions that are made in each path.*

Hierarchical collision detection was taken first to quickly resolve any points that were not near obstacles. If this failed, then a more computationally intensive check was performed. Since the start was always at ground height, we first look to see if the next point is within an obstacle's inner boundary. If so, it is deemed to be at height = object.height. We then examined if the starting point was also on an obstacle. The relative heights (Δh) of the points were used to determine a change in height, and if this was less than the permissible jump height, no collision was detected.

Obstacle interaction also became a challenging task to handle when re-wiring the tree. Node heights had to be considered in addition to just x-y Euclidean distance. Traditional RRT* uses a search radius typically larger than its steer step to re-wire the tree, looking for nodes that can be reconnected to achieve the same pose at a lower cost. This meant that when searching for nodes on top of obstacles, the maximum distance, direction, and Δh had to be considered.

Once the goal was found, the algorithm re-calculated the total time to reach the goal after every 50 nodes were added. This provided a measure of how quickly the method converged toward optimal. A fixed number of nodes are used here so that the results can be compared. Maximizing optimality and computational efficiency are outside the scope of this paper.

In Fig. 6 we can see how the RRT* works when adding nodes. Black paths are nodes joined by the original RRT algorithm. Blue paths indicate when a lower cost node is chosen to connect new nodes to instead of the closest. Green paths are nodes that get re-wired through the RRT*. Red paths are jumps, and cyan paths are drops.

## III. Results

### A. *Constant Scenarios – Grid Search A**

For the constant scenarios, Figure 5 shows the corresponding optimum path obtained by the A* algorithms in blue. The path shows the cells in which the SUMO is in contact with the ground or obstacles. Then, Table I displays the times for the robot to reach the goal, and the time it took to run each simulation to compare it to the RRT* method.

TABLE I. – Grid Search A* Scenario Results

| Scenario | Time for path | Time to process solution |
|---|---|---|
| 1 | 11.91 s | 122.94 s |
| 2 | 12.52 s | 103.56 s |
| 3 | 13.20 s | 89.53 s |
| 4 | 9.35 s | 84.58 s |

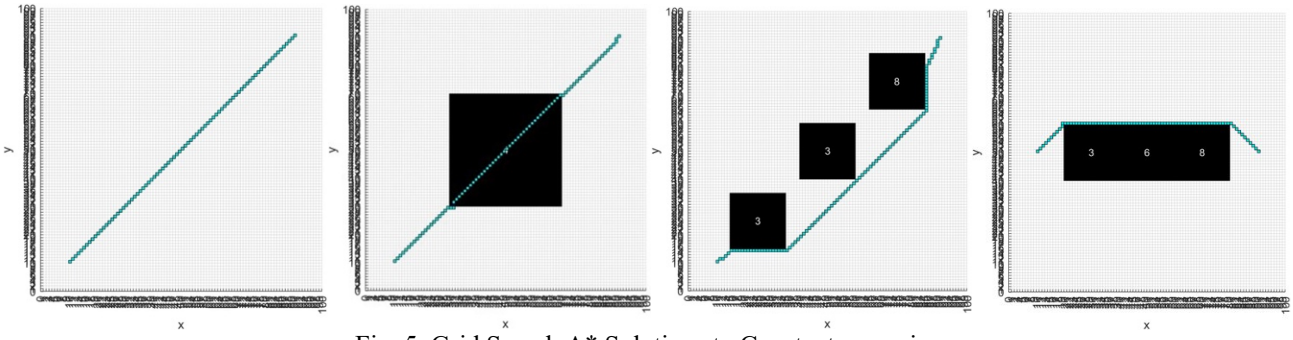


Fig. 5. Grid Search A* Solutions to Constant scenarios

TABLE II. – RRT* Results

| | Scenario 1 | Scenario 2 | Scenario 3 | | Scenario 4 | |
|---|---|---|---|---|---|---|
| Run # | Solved path time | Solved path time | Solved path time | Jump? | Solved path time | Jump? |
| 1 | 12.3278 | 13.1391 | 13.8862 | No | 10.1943 | No |
| 2 | 13.008 | 12.8793 | 13.6258 | Yes | 10.4441 | Yes |
| 3 | 12.4807 | 13.0872 | 13.8285 | No | 10.0596 | No |
| 4 | 12.8724 | 13.1249 | 13.8119 | No | 10.2432 | Yes |
| 5 | 12.3708 | 12.9495 | 13.9706 | Yes | 10.3615 | No |
| 6 | 12.4068 | 13.2028 | 13.8292 | Yes | 10.3566 | No |
| 7 | 12.3313 | 13.3074 | 14.0682 | No | 10.0861 | No |
| 8 | 12.4137 | 13.3884 | 13.8295 | No | 10.3223 | Yes |
| 9 | 12.419 | 12.7923 | 13.9239 | No | 10.4211 | No |
| 10 | 12.5255 | 13.0403 | 13.7617 | No | 10.5557 | Yes |

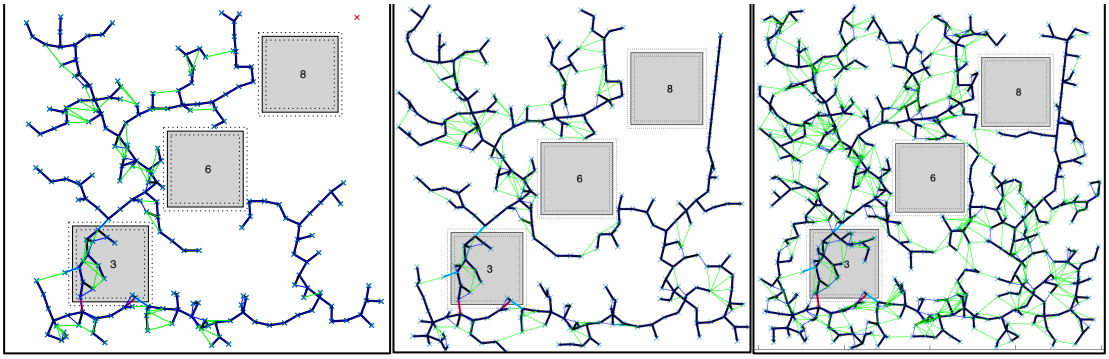


Fig. 6 RRT* search algorithm at 200, 400 and 600 nodes. Here we can see the path re-wiring itself as well as avoiding tall obstacles.

### *B. Constant Scenarios – RRT**

As the RRT* model is a statistically optimal solution, many nodes are required to guarantee optimality. As such we do not expect to get an optimal solution every run, unlike the grid search with A* method. Instead, we see that the RRT* approaches optimal with more runs. It is shown that all the runs improving the scores were added as new nodes. Also, RRT* is very fast to obtain a solution, a solution was always found in under 0.5 seconds. Then, it tooks approximately 20 seconds to generate 3000 nodes based on the computer used. Ten runs were performed for each of the 4 scenarios. Each run consisted of 3000 nodes. The first scenario was an un-obstructed environment with a best time achievable of 11.9092 seconds. The best run from this scenario completed with a calculated run time of 12.3278 seconds, and the average of all 10 runs was 12.5156.

On the second scenario, every optimal path found that jumping over the obstacle improved the total run time. The ideal calculated run time for this scenario is 12.3092 (the empty set run plus cost of one jump). SUMO however must make near close to perpendicular jumps to the edge of the obstacles, and therefore we see slightly reduced performance from this ideal calculation. The ideal score was 12.7923 seconds, and the fastest path average was 13.09112 seconds.

The third scenario with 3 obstacles produced some varied results. Three runs included jumps over the lowest of the 3 obstacles, while others did not. The best run of this scenario included a jump and scored 13.6258 seconds. The best run without a jump scores 13.7617 seconds. The average of the three runs that included jumps was 13.8085 seconds, and the average of runs without jumps was 13.8728 seconds. The average of all ten runs was 13.8535 seconds. It appears that a single jump in this scenario did not provide a large benefit to the fitness of the algorithm, and as a result it did not find many solutions that included it. Figures 7-10 show the best and worst solutions to each respective scenario.

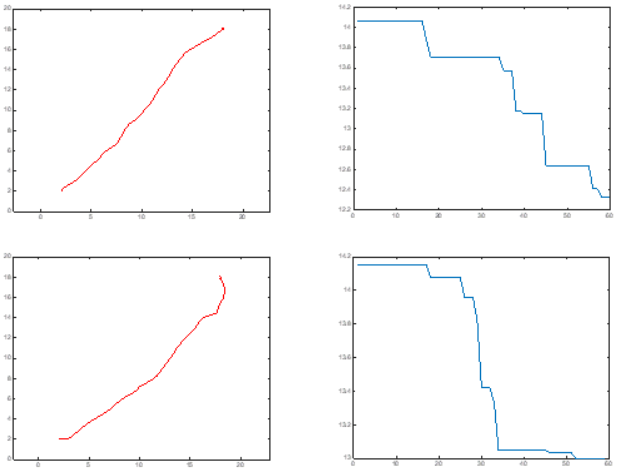

*Fig. 7. Best (top row) and worst (bottom row) solution paths for scenario 1, and fitness improvement with added nodes (right)*

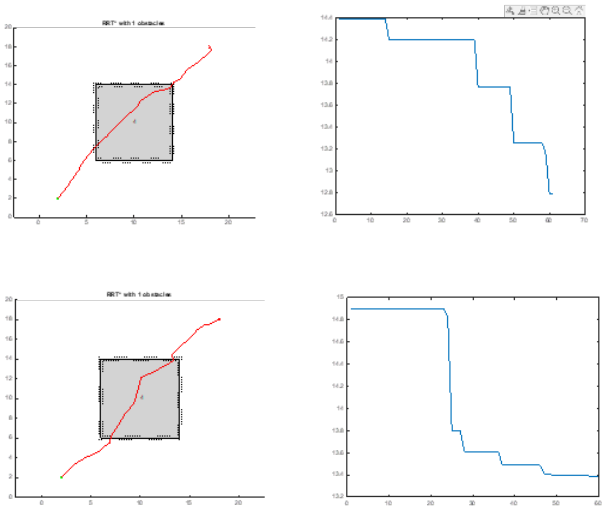

*Fig. 8. Best (top row) and worst (bottom row) solution paths for scenario 2, and fitness improvement with added nodes (right)*

The fourth and final scenario was a simulated stair-step environment. In this scenario, again we saw 3 runs with jumps over all 3 obstacles, 1 run with a jump on only the first obstacle, and the rest went around all three. The seventh run was the quickest at 10.0861 seconds and went around all obstacles. The runs with jumps performed marginally worse than runs with solutions not including jumps. The jumping runs averaged 10.3913 seconds, and the non-jumping runs averaged 10.2653 seconds.

### C. *Constant Scenarios – Comparison*

Comparing the two methods shows that A* achieved the best time to reach the goal. Since it is the optimum solution, it is not surprising, but the differences are small. However, the time required for the A* algorithms to produce the answer is significantly at least four times higher than RRT* producing an improved answer, and even far greater time difference between RRT* just finding a solution. A* and RRT* both have advantages that could be used in different circumstances. A* requires a full knowledge of the map and always provides the best answer within the given constraints while RRT* favors the execution speed

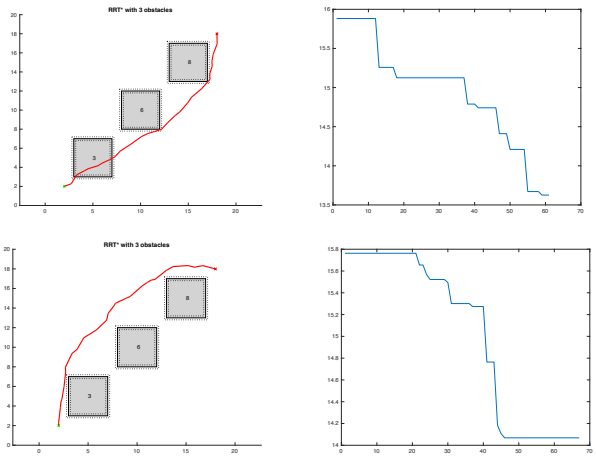

*Fig. 9. Best (top row) and worst (bottom row) solution paths for scenario 3 and fitness improvement with added nodes (right)*

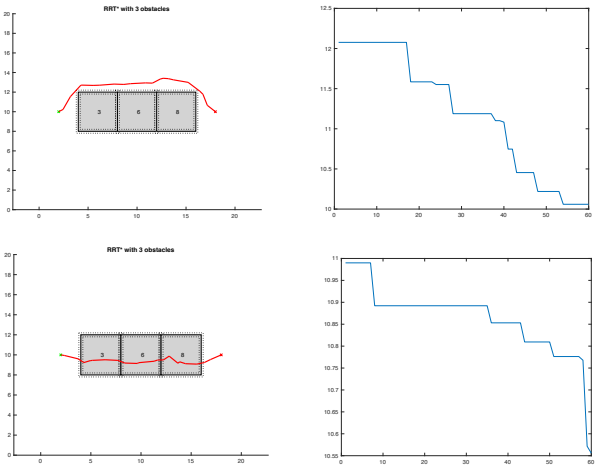

*Fig. 10. Best (top row) and worst (bottom row) solution paths for scenario 4, and fitness improvement with added nodes (right)*

TABLE III. – Time Difference Between Jump and no Jump on Random Scenarios using Grid Search A*

| | *Sparce (5 obstacles)* | *Dense (15 obstacles)* | *Very Dense (30 obstacles)* |
|---|---|---|---|
| Run 1 | *0 s* | *2.55 s* | *1.45 s* |
| Run 2 | *0 s* | *0 s* | *4.35 s* |
| Run 3 | *0.63 s* | *0.33 s* | *3.98 s* |
| Run 4 | *0 s* | *1.33 s* | *1.26 s* |
| Run 5 | *0.94 s* | *0.52 s* | *1.36 s* |
| Run 6 | *0.2 s* | *0 s* | *0.67 s* |
| Run 7 | *0 s* | *2.55 s* | *1.98 s* |
| Run 8 | *0 s* | *0 s* | *1.95 s* |
| Run 9 | *0 s* | *0 s* | *1.04 s* |
| Run 10 | *0 s* | *0.13 s* | *0.27 s* |
| *Average Time Difference* | ***0.18 s*** | ***0.72 s*** | ***1.83 s*** |

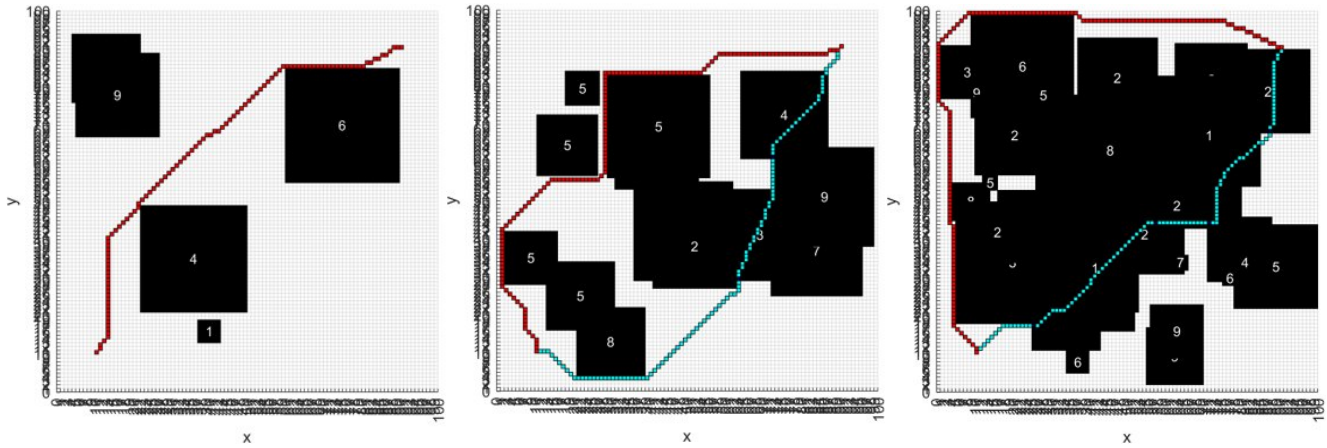

Fig. 11. Grid Search with A* for randomly generated scenarios. Left: Sparce (5 obstacles). Center: Dense (15 obstacles). Right: Very Dense (30 obstacles).

to provide an answer without prior knowledge of the potential paths to explore. If the jumping Sumo needs to act or react near real time, RRT* would be the most suitable algorithm to use. However, if the map is fully known and the robot has no time constraints, the grid method using A* would guarantee an optimal solution.

### *D. Random Scenarios – Grid Search A**

A quick analysis of random scenarios was established with the number of obstacles being constant but with the size and location varying through the grid. Tables III show the result of running the simulations for sparce, dense, and very dense obstacles environment for 10 iteration each and compare the results. From the results, the jumping scenario is more beneficial with more obstacles present on the map compared to the non-jumping. Figure 6 shows examples of sparce, dense, and very dense scenarios and path to compare the timing between the non-jumping path in red and the jumping ability path in blue. Then, to see the effect of the object size on the jumping condition, 3 simulations were run with a different fixed size for the obstacle with 15 obstacles having the location varying. Table IV displays the results of the difference in time between the jumping and no jump condition. From the results, the jumping scenario is more beneficial with larger obstacles present on the map compared to the non-jumping.

## IV. CONCLUSION

Adding a jumping capability to a wheeled robot has a beneficial factor. For the purpose of this study, the change is not significant, but compared to the size of the map it is. For the dense scenario, an average of 1.8 seconds is roughly a 10% time reduction obtained by adding the jumping capabilities. The time benefit discovered using RRT* was less obvious due to the limited number scenarios we were able to capture,

TABLE IV. – Time Difference Between Jump and no Jump n Changing Size Scenarios using Grid Search A*

| | 1x1 m | *3x3 m* | *6x6 m* |
|---|---|---|---|
| Run 1 | *0 s* | 0.68 s | *5.17 s* |
| Run 2 | *0 s* | *0.13 s* | *1.79 s* |
| Run 3 | *0 s* | *0 s* | *0.75 s* |
| Run 4 | *0 s* | *0 s* | *0.92 s* |
| Run 5 | *0 s* | *0 s* | *2.16 s* |
| Run 6 | *0 s* | *0 s* | *2.20 s* |
| Run 7 | *0 s* | *0 s* | *2.55 s* |
| Run 8 | *0 s* | *0 s* | *2.18 s* |
| Run 9 | *0 s* | *0 s* | *0 s* |
| Run 10 | *0 s* | *0 s* | *No valid path for no jump* |
| *Average* | ***0 s*** | ***0.08 s*** | ***1.97 s*** |

and this is seen as a limitation to our research. A full comparison using the same randomly generated scenarios between methods would likely result in similar findings.

## V. Other Considerations

The two methods used a simplistic approach for the obstacles design. To make it even more realistic, upgraded methods could include the battery life of the robot as part of the cost function and jumping could imply reducing the efficiency of the robot afterward.

Future work with the RRT* method should consider implementing an additional cost to adjusting the pose angle. The scenarios examined within this paper were mostly unidirectional, but it could be considered that more complicated paths could contain harder turns. Alternatively, a path smoothing method could be used.

Additionally, optimality could be examined further by looking at the convergence rate of randomly generated maps compared to computation time. Our RRT* Fitness was measured after adding 50 nodes; however, it would be interesting to see the real-time cost of achieving higher optimality as new nodes are added.

Lastly, an additional acknowledgement to Sai Vemprala [5] who posted an RRT* script to the Mathworks website. It was used as the foundation of our RRT* script. We found that it was incomplete, and did not re-wire the networked trees, but was a very useful tool for understanding the complex

## References

[1] "Industrial Automation: A Complete Guide — Optimize Your Production Process," *IndusLabs,* Jan. 19, 2022 [Online]. Available: https://www.industlabs.com/news/industrial-automation. [Accessed: May. 1, 2022].

[2] M. Gerdts, R. Henrion, D. Hömberg, and C. Landry, "Path planning and collision avoidance for robots," *Numerical Algebra, Control and Optimization*, Germany, vol. 3, pp, 437-463, Feb. 2012.

[3] V. Klemm, A. Morra, C. Salzmann, F. Tschopp, K. Bodie, L. Gulich, and R. Siegwart, "Ascento: A two-wheeled jumping robot," In *2019 International Conference on Robotics and Automation (ICRA),* IEEE, pp, 7515-7521, May 2019.

[4] Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7), 846-894.

[5] Vemprala, Sai. "Mathworks." 05 Jan. 2017. https://www.mathworks.com/matlabcentral/fileexchange/60993-2d-3d-rrt-algorithm