Robotica: (2023) volume 41, pp. 3–15. © The Author(s), 2020. Published by Cambridge University Press.
doi:10.1017/S0263574720000594

Speeding Up On-Line Route Scheduling for an Autonomous Robot Through Pre-Built Paths

Raul Alves*, Josué Silva de Morais and Keiji Yamanaka

Faculty of Electrical Engineering, Federal University of Uberlândia, Uberlândia, Brazil
E-mails: josue@ufu.br, keiji@ufu.br

(Accepted June 19, 2020. First published online: 14 July 2020)

SUMMARY
Today, robots can be found helping humans with their daily tasks. Some tasks require the robot to visit a set of locations in the environment efficiently, like in the Traveling Salesman Problem. As indoor environments are maze-like areas, feasible paths connecting locations must be computed beforehand, so they can be combined during the scheduling, which can be impracticable for real-time applications. This work presents an on-line Route Scheduling supported by a Fast Path Planning Method able to adjust pre-built paths. Experiments were carried out with virtual and real robots to evaluate time and quality of tours.

KEYWORDS: Route scheduling; Autonomous robots; TSP; Path planning; Evolutionary algorithm.

1. Introduction

The Autonomous Robots market was reported to have grossed $158 million in 2016 and is projected to reach at least $390 million by 2022, according to QYResearch's report.[1] These numbers reflect the growing interest in autonomous robots, especially in service robots. Such robots aim to perform tasks considered dirty, dull, distant, dangerous, or repetitive for humans. They are categorized as: professional service providers (working in offices, hospitals, and warehouses), domestic service robots (for entertainment, education, and personal works at home), security robots (defense, safety, and rescue), and space robots (planet exploration).[1]

This paper addresses a problem in which a service robot has to visit a set of target locations within an indoor environment to perform some task, such as delivering mail, picking up trash in offices, finding objects, patrolling, etc. In order to perform any of these tasks efficiently, the robot must be guided by a tour that covers all the locations and brings it back to the origin or to a specific spot, such as a charging station, using the minimum distance. In addition, locations can be added or removed during the execution of the visits; therefore, the process of finding the proper tour should be performed on-line.

This problem can be cast as an instance of the Traveling Salesman Problem (TSP),[2] where the target locations are cities and the robot is a salesman. In general, most of the work found in the literature regarding the TSP applies evolutionary methods to generate tours and heuristic functions to estimate the distance between locations rather than the length of actual paths. However, indoor environments are maze-like structures comprising many rooms, corridors, and corners with different sizes

* Corresponding author. E-mail: raul@ufu.br

[1] https://www.researchnreports.com/service-industries/Global-Autonomous-Mobile-Robots-Market-Research-Report-2017-162973

and shapes. Thus, the distance acquired from feasible paths, provided by path planning algorithms, can yield more accurate tours.

Considering this scenario, there are two critical phases in the route scheduling process that must be performed quickly, otherwise it may be impractical for a real-time application. First, compute a path for each pair of locations. Second, evaluate different ways to arrange these paths into a tour that reduces the total distance.

In this paper, we introduce an on-line Route Scheduling System that uses pre-built optimal paths to accelerate the path planning phase. Such paths connect strategic points of the environment, such as the entrance of rooms and hallways. Given a map of the environment, the user can mark the target locations and the return spot anywhere on the map. Then, a path matrix is created, where each cell corresponds to the path from one location to another. For each pair of origin and destination of the matrix, our Fast Path Planning Method (FPPM) builds a path as follows: (1) check whether there is a straight line between origin and destination; or (2) select and adapt some pre-built path from origin to destination; or (3) compute a new optimal path from origin to destination as the other options have failed. Once the path matrix is filled, a genetic algorithm (GA) evaluates several tours, by combining paths from the matrix, and returns the shortest one.

Experiments were carried out in order to evaluate time and quality of tours using three algorithms to populate the path matrix: A*,[3] Rapidly exploring Random Tree (RRT),[4] and our FPPM. Preliminary tests show that even for a few target locations in a large environment, it is impossible to have a Route Scheduling working on-line using Path Planners that compute all the paths from scratch.

The remainder of the document is organized as follows. Section 2 lists some related works on robotics and TSP applications. The proposed Route Scheduling system along with our FPPM are described in Section 3. The description of the experiments and their results using simulation and a real robot are provided in Section 4. Finally, the conclusion and future work are described in Section 5.

2. Related Work

According to the survey of Macharet and Campos,[5] the TSP has been widely studied and frequently considered in the field of mobile robotics. We review some researches here.

In ref. [6], a path planning method is proposed for LiDAR-equipped Unmanned Aerial Vehicles for bridge inspection using a GA and A* to solve the TSP considering the potential locations of surface defects such as cracks. First, a GA selects inspection points using a heuristic, and then A* finds a collision-free path between every two points. The objective is minimizing time-of-flight to achieve acceptable visibility.

Autonomous Underwater Vehicles (AUV) are also a topic of interest in robotics. The work in ref. [7] introduces a mission planner for AUV, based on the TSP and the Dynamic Knapsack Problem, that satisfies the following objectives: (1) accommodating an effective route planning for AUV to take a maximum use of time that battery capacity allows; (2) accurate task ordering according to importance, riskiness, and time span; and (3) guaranteeing on-time completion of a mission while effectively managing the available time for a series of deployments in a long mission.

Another application, in a water environment, is presented in ref. [8]. This time, an Autonomous Surface Vehicle (ASV) must follow a path, generated by a GA, in a lake for environmental monitoring purposes. The path planning problem has been modeled as a particular case of the TSP, in which the ASV should visit a ring of beacons deployed at the shore of the lake for delivering the collected data. For achieving a complete representation of the lake, the distance traveled should be maximized instead of minimized as in the classic TSP.

A multi-goal motion planning problem is addressed in ref. [9]. The TSP formalism is applied to find a policy in belief state space for the robot to traverse through a number of goal points by means of a Partially Observable Markov Decision Process, which takes into account motion and sensor uncertainties. The underlying motivation of this work is planning for a service or a rescue robot in order to perform tasks like search and exploration in an environment.

In ref. [10], a multi-objective GA (NSGA-II) is applied to trajectory planning in a mobile robot. A safe path is obtained by optimizing certain objectives (travel time and actuators effort) taking into account the limitations of mobile robot's geometric, kinematic, and dynamic parameters.

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

Abu-Dakka et al.[11] present a methodology to obtain smooth trajectories for industrial robots in complex environments using a GA that creates a collision-free trajectory as the robot moves. Trajectories are modeled in cubic polynomials, and discretized to solve the inverse dynamic problem and to validate the dynamics restrictions. Random search algorithms with new crossover and mutation operators have been introduced.

Robotics has also been widely applied in agriculture. Azimi et al.[12] compare a binary Particle Swarm Optimization (PSO) and a GA to find the shortest routing path for spraying operations in a greenhouse. The routing problem has been expressed in terms of the TSP, and to solve it, an objective of a total path length was measured based on the path computed using a probabilistic roadmap path planner.

In a Wireless Sensor Networks (WSN), when there is a need to transmit large volumes of data, the RF-communication process exhibits significant packet losses and demands certain energy resources. Therefore, Tsilomitrou et al.[13] address the utilization of a mobile robot as data mule for collecting and transferring data. Each static node within the WSN has its data generation rate resulting in an imposed inter-visit duration due to its hardware limitations. So the robot approaches the nodes, collects their stored data, and transfers these to a depot station. The trajectories to cover the nodes are extracted by solving a combinatorial optimization problem that resembles that of the Traveling Salesman Subset-tour Problem (TSSP).

Cheng et al.[14] introduce an algorithm based on Ant Colony Optimization (ACO) to build a patrol path for a mobile robot in an environment with an obstacle. Using the TSP model, this algorithm makes improvements in four aspects including initial position of ants, distribution of global pheromone, pheromone evaporation coefficient, and introduction of simulated annealing.

There are other applications in patrolling. For instance, Ghadiry et al.[15] show an approach in which sets of viewpoints and robots are used to monitor an environment, whose number of robots is less than the number of the viewpoints. The robots are supposed to visit the viewpoints by traveling between them continuously. This problem is considered to be a variant of the TSP, namely, the Minimum-Time Multidepot multiple Traveling Salesmen Problem. The distinguishing features of the proposed problem statement is that first of all the initial depots and their corresponding assigned number of robots are not pre-specified. Second, the cost function to be minimized is total travel time of robots as opposed to the travel distance or waiting time of the salesmen at the visited nodes for a given TSP with time windows.

Le et al.[16] present a TSP-based path planning algorithm, also for multiple robots, to create a path planning for each robot in the team in order to clean industrial environments. Usually, industrial environments are much too large for one robot to clean by itself in addition to the fact that the amount of dirt will exceed the dirt stored capacity of one robot. The proposed algorithm allows the robot team to clean efficiently by spending more time on the areas with higher dirt levels. It also outperforms the single robot approach in time efficiency while having almost the same total battery usage and cleaning efficiency result.

The TSP belongs to the class of NP-complete problems, because the time increases exponentially as more locations to visit are added to the problem. Therefore, approximation methods – such as GA, PSO, and ACO listed above – are more likely to be used when it comes to finding a satisfactory tour in a reasonable time.

The other time-consuming task addressed in this paper is the computation of paths that form a tour. Path planning algorithms aim to compute paths from origin to destination positions that minimize the distance. Typically, path planning strategies are based on search or sampling.

Search-based algorithms have dominated the field of path planning in robotics for decades. Probably the most popular algorithm of this family is A*,[3] which uses a heuristic to narrow the search. If the heuristic function is admissible, meaning that it never overestimates the actual cost to reach the destination, A* is guaranteed to return the optimal path. However, this approach is subject to certain limitations. As A* evaluates several path possibilities, the algorithm keeps all states in memory during the search until it finds the best one. Therefore, depending on the scale of the problem, this algorithm may require a lot of time and space.

On the other hand, sampling-based methods are relatively new to robotics and are the subject of a rush of recent research. For instance, the RRT[4] algorithm was designed to efficiently search on high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from states created randomly from the search space and is inherently biased to grow towards large

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

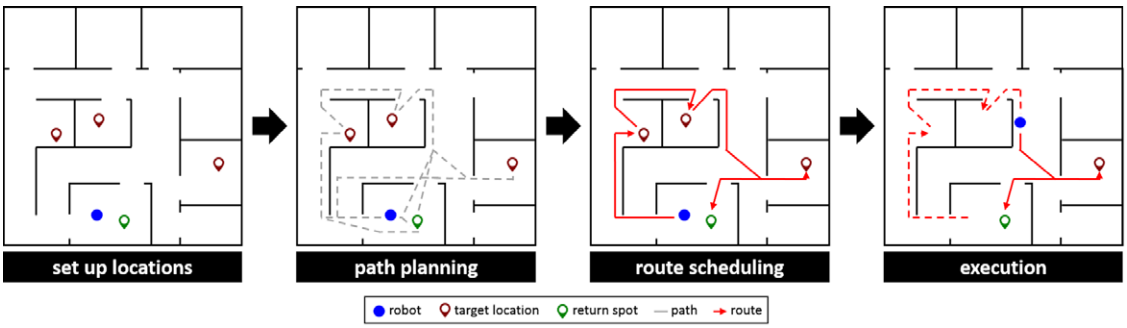

Fig. 1. Route scheduling flow. This hypothetical scenario illustrates the process of Route Scheduling, which works in four steps: set up target locations on the map, creation of a path matrix using path planning algorithms, tour generation with GA, and real-time monitoring.

unsearched areas of the problem. The algorithm ends when a state reaches the destination. Unlike A*, RRT is fast and light, but the path will not necessarily be optimal or near-optimal.

Plenty of papers propose efficient approaches to solve path planning and TSP separately; however, we have not found much research addressing both aspects into a single solution, for real-time use, in the field of robotics.

3. On-Line Route Scheduling

Computational time is crucial in robotics, especially for applications that must run on-line. This work introduces a Route Scheduling system to find a tour in real-time for an autonomous robot that has to visit target locations within an indoor environment. In order to yield feasible tours in reasonable time, two tasks must be optimized: (1) *path planning* to compute path and distance from one location to another; (2) *route scheduling* to evaluate several combinations of paths to find the shortest tour.

3.1. Route scheduling process

We propose a system to handle the Route Scheduling process, as depicted in Fig. 1. This system enables the user to set target locations and the return spot on a map of the desired environment. It also tracks and displays the robot's current position. Once the locations are set, a path matrix P that represents a complete graph is created with feasible paths connecting all locations, including target locations, robot's position, and return spot. Paths are computed for all pairs of locations, with a path planning algorithm, so they can be used in the route scheduling to display the route and compute the distance. Thus, the dimension of $P = |L| + 2 \times |L| + 2$, where L is the list of target locations. Figure 2 shows an example.

To optimize the computation, elements of the main diagonal in P are set as being empty, which correspond to paths where the origin and destination are the same location. Also, when the element P_{ij} is computed, which corresponds to the path from i to j, the inverse path is automatically stored at P_{ji}.

After the path planning phase, the system initiates the route scheduling. It aims to generate the shortest tour that starts from the robot's current position and ends at the return spot while passing through all target locations. We modeled this problem as an instance of TSP, where approximate solutions are quickly generated by means of a GA. An individual has a chromosome to allocate all target locations, which must be visited only once. The sort of target locations within the chromosome is also important as it determines the sequence in which the robot will perform its visits, as shown in Fig. 2. Therefore, the individual (tour) is represented as a vector of locations, where the first one is the robot's position, followed by the chromosome of target locations, and then the return spot. During the process only locations within the chromosome are permuted, while the first and the last stay static.

Given the path matrix P, the fitness function F which evaluates an individual is calculated by the sum of all path lengths, starting by the path from the robot's position to the first target location, then

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

Algorithm 1 GA – Route Scheduling

```
Create T_p random individuals for the initial population
Evaluate the initial population
for generation t ← 1..N_iter do
    Select P_rec individuals
    Crossover
    Mutation – with probability Pmut
    Evaluate the new individuals
    Select the best T_p individuals
end for
Return the best individuals
```

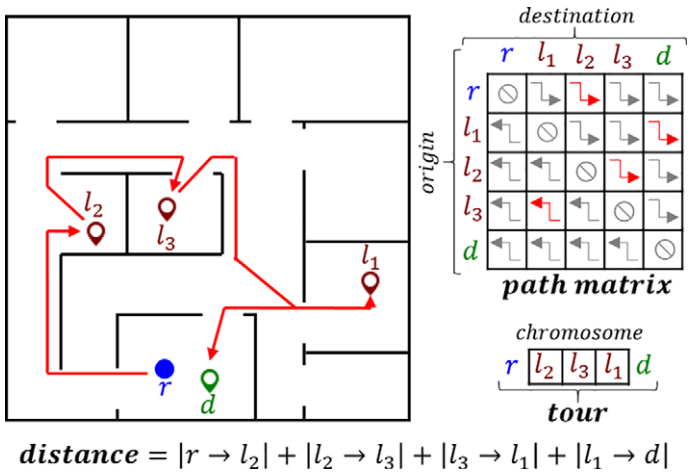

Fig. 2. Tour example. This image depicts the hypothetical tour of Fig. 1. On the left, a map shows three target locations (l_1, l_2, and l_3), the robot's position r, and the return spot (depot) d. Red arrows represent paths, stored in the path matrix, that compose the tour. The distance is calculated by summing the paths from r to d.

paths linking the sequence of target locations in the chromosome, and finally the path from the last target location to the return spot.

$$F = |P_{0,1}| + \sum_{i=1}^{n} |P_{i,i+1}|, \forall i \in chromosome + |P_{n,n+1}|.$$

Our algorithm 1 was inspired by the traditional GA.[17] First, an initial population is generated randomly with T_p individuals and submitted to the evaluation process. Each individual represents a route scheduling for the robot passing through n target locations without repetition. The main loop runs for N_{iter} iterations, where genetic operations for selection, crossover, and mutation are made. For the next step, *Prec* individuals must be selected from the current population using Roulette Wheel method. In the Roulette Wheel, each individual of the current population is represented by a value proportional to its fitness. It means that individuals with high values have more chances of being selected.

After the selection, the crossover process begins. TSP is a permutation problem as the cities should not be repeated within the chromosome. Thus, methods for crossover and mutation must consider this constraint. We implemented a crossover method, called Partially Matched Crossover (PMX), that prevents the generation of invalid individuals as it performs safe permutations between the chromosomes. In PMX, a section of the chromosomes (through two cut points) is selected, where the genetic material of both parents will be fully exchanged. Then, some changes are made in the offspring to eliminate repeated genes. PMX makes some changes to avoid repetition, which can generate different genetic structures when compared to the parents.

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

Next, some of the new individuals are submitted to the mutation process with probability *Pmut*. The mutation method applies a simple permutation between two random genes of the individual, which also avoids the generation of invalid individuals. Finally, the new individuals are evaluated and inserted into the current population. This population is sorted according to the fitness values. The best T_p individuals, in each iteration, are selected to form the new population that will perform again the processes of selection, crossover, and mutation.

When the algorithm reaches the stop criterion, it sends the best tour to the robot. During the execution, the robot may not follow exactly the same path computed by the route scheduling as it may deviate from dynamic obstacles, such as people and moving objects, that were not presented on the map before. At any time, the user can add new target locations. When this happens, the system reschedules the route from the robot's current position considering the new and remaining locations to be visited.

3.2. FPPM with pre-built paths

In our system we use a path matrix to store paths that connect all desired locations. Such paths can be computed by any path planning algorithm based on the map of the environment. The problem is that path planning is also costly in terms of computation, especially when it comes to compute $n \times n$ paths to populate a path matrix in real-time.

To solve this problem we introduce a FPPM using pre-built paths to speed up the population of the path matrix. First, given a map of the environment, we pick some strategic points, such as the entrance of rooms and hallways, and compute optimal paths connecting all these points with A* algorithm. Depending on the size of the environment and the amount of points, this task can take a lot of time, but it is done only once when a new map is available. At the end, four files are generated:

- *environment*.png: image file of the map.
- *environment*.points: list of strategic points with their coordinates.
- *environment*.paths: list of paths connecting each pair of points. In this case, a path is a sequence of coordinates, from the origin point to the destination point. These paths are mainly used to be plotted on the system.
- *environment*.distance: list of path lengths. Instead of computing the length of each path every time during the route scheduling, we provide this value on this file to avoid this computation.

When the user launches our system, an environment must be selected. Then, the system opens the image file of the map and loads points, paths, and distances to the memory so they can be easily accessed. After that, as depicted in Fig. 1, the user marks all target locations and the return spot on the map. In addition to the robot's current position, the structure of an empty path matrix is created. So, for each pair of origin and destination, the Algorithm 2 is called to compute the corresponding path and store it on the matrix.

The pseudo-code of Algorithm 2 describes our FPPM with pre-built paths. The list of parameters are:

- *s*: Start location.
- *g*: Goal location.
- *P*: List of strategic points saved in memory from the *environment*.points file.
- *SP*: Hash Table structure saved in memory from the *environment*.paths file.

A hash table, or a hash map, is a data structure that associates keys with values. Each element is assigned a key. By using that key we can access the element in $O(1)$ time. Using the key, a hash function computes an index that suggests where an entry can be found or inserted. In this case, we combine *s* and *g* to create a unique key where we can find its respective static path. This is important because static paths are frequently used in this algorithm, and they must be retrieved quickly.

The algorithm starts by setting the *shortestPath* variable as empty (line 1). This variable aims to keep the shortest path, which will be returned in the end (line 17). Then, **Line** function checks whether there is a straight line connecting *s* to *g* without hitting any wall (line 2). This scenario is depicted by the first frame of Fig. 3, where the line l_1 represents a path from *start* to *goal*. In such

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

Algorithm 2 Fast Path Planning (s, g, P, SP)

```
1:  shortestPath ← ∅
2:  if (Line(s,g)) then
3:      shortestPath ← Line(s,g)
4:  else
5:      for all p′ ∈ P and Line(s,p′) do
6:          for all p″ ∈ P and Line(g,p″) do
7:              path ← Line(s,p′) ∪ SP[p′,p″] ∪ Line(p″,g)
8:              if (|path| < |shortestPath|) then
9:                  shortestPath ← path
10:             end if
11:         end for
12:     end for
13: end if
14: if (shortestPath = ∅) then
15:     shortestPath ← A*(s,g)
16: end if
17: return shortestPath
```

a case, this line from s to g is stored at *shortestPath* (line 3). Otherwise, the algorithm tries to find a path based on the static paths of SP (lines 4–13).

The loop of line 5 lists all strategic points p' of P that are accessible via a straight line from s, while the inner loop of line 6 lists the strategic points p'' for g. Thus, for each strategic point p' reachable from s, and for each strategic point p'' reachable from g, the algorithm evaluates new hypothetical paths by concatenating the static path from p' to p'' of SP along with the lines from s to p' and from p'' to g (line 7). For instance, on the second frame of Fig. 3, two paths are available from *start* to *goal*. The list of reachable points of *start* is composed by p_1 and p_2, while p_3 is the only point for *goal*. The following straight lines are used as sub-paths:

- l_1: line from *start* to p_1
- l_2: line from *start* to p_2
- l_3: line from p_3 to *goal*

Moreover, the static paths of SP that connect the points of *start* to the points of *goal* are:

- sp_1: static path from p_1 to p_3
- sp_2: static path from p_2 to p_3

The resulting paths are given by:

- $path_1 = l_1 \cup sp_1 \cup l_3$
- $path_2 = l_2 \cup sp_2 \cup l_3$

In this scenario, $path_1$ is slightly smaller and will be considered as the shortest path from *start* to *goal*. Although these paths are not optimal anymore, they are always composed by one static optimal path along two lines, which are the shortest way of linking two points. Thus, we can provide sub-optimal paths in constant time.

As stated, every time a new path is computed, the algorithm compares its length against to the *shortestPath* (line 8), and if it is smaller, this path is set to be the shortest one (line 9). If the map did not have enough strategic points, these previous approaches may fail or yield not very reasonable paths. So before the end of algorithm, if the *shortestPath* is still empty, a dynamic path is created from scratch with **A*** (lines 14–16) to guarantee that a path will be returned no matter where s and g are on the map. This scenario is illustrated by the last frame of Fig. 3, where *start* and *goal* are connected by a dynamic path dp_1.

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

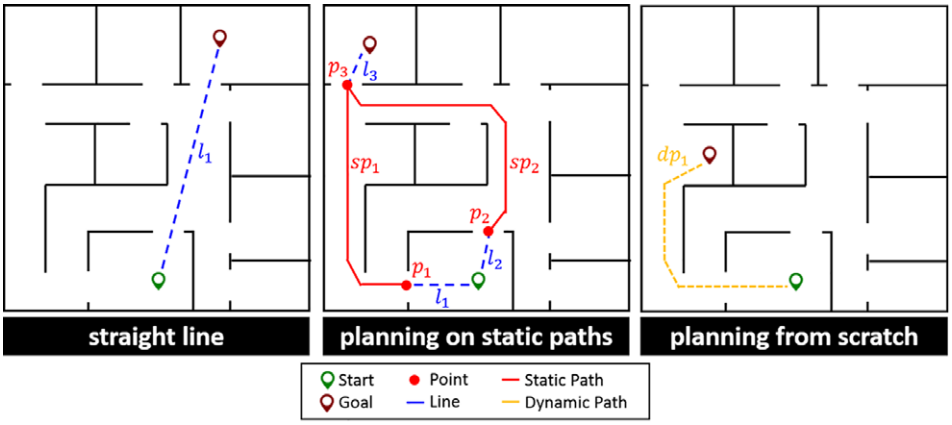

Fig. 3. Path planning strategy. This picture illustrates three possible scenarios handled by our algorithm. First, a straight line can be used as a path from the start location to the goal location. Second, different static paths combined with straight lines are evaluated to find the shortest path between start and goal locations. Third, if all the previous approaches have failed, a path is computed with A*.

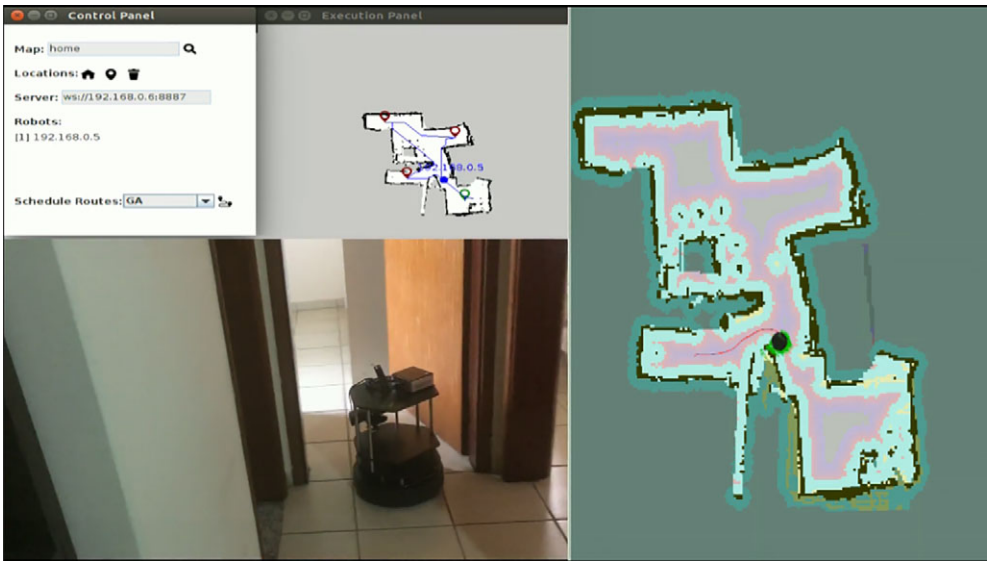

Fig. 4. Real environment. The system is shown on top, with a map of the house, during the execution of a tour. At the bottom, a real turtlebot-2 is navigating while its current path is plotted on RViz.

4. Experiments and Results

Experiments were carried out to evaluate our on-line Route Scheduling system in terms of processing time and travel distance. Tours were generated by the proposed GA with paths created using different path planners: A*, RRT, and our FPPM.

A robotic platform, called turtlebot-2, was used for these experiments. This robot consists of an Yujin Kobuki base, wheel encoders, wheel drop sensors, an integrated gyroscope, bump sensors, cliff sensors, a 2200 mAh battery pack, a Microsoft Kinect sensor, an Intel NUC i7, a fast charger, a WiFi dongle, and a hardware mounting kit attaching everything together.

Our system is able to connect to this robot in order to track its position on the map and send the generated tour. Two environments were set for the tests. The first one is a real house with some rooms and hallways (Fig. 4). Its map was acquired by running *GMapping* algorithm on the turtlebot-2. As it is difficult to find a large and complex real area available for testing, our second environment is a 3D building widely used in robotics community, called WillowGarage Inc. It is the worst case scenario for indoor environments available on *Gazebo* simulator. Figure 5 illustrates a trial on this environment with a virtual model of turtlebot-2. Given these two environments, some scenarios were set with 3, 5, 10, and 15 target locations to be visited, as illustrated in Figs. 6 and 8. Red markers are

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

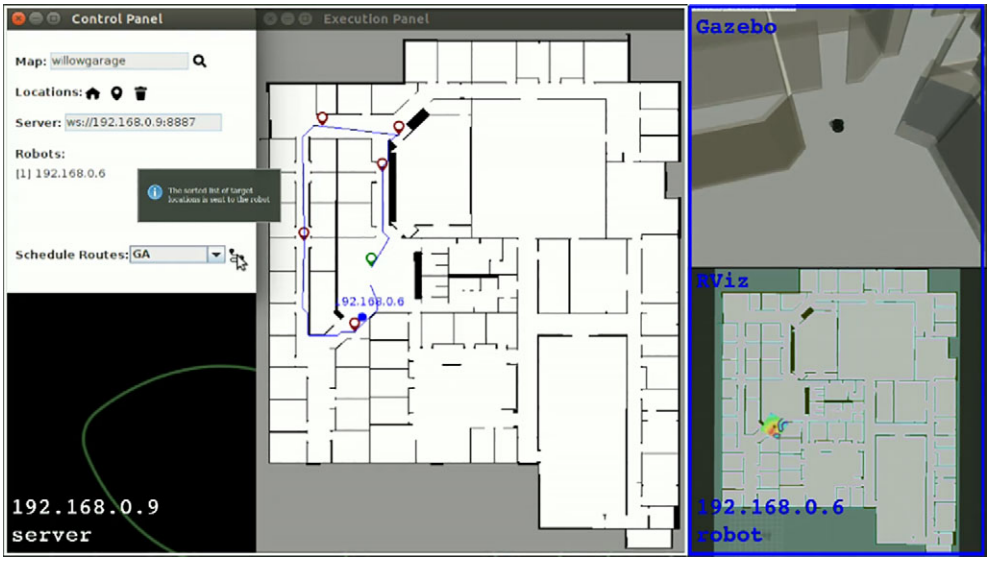

Fig. 5. Simulation environment. The system is shown on the left, with a map of WillowGarage, during the execution of a tour. On the right, a virtual turtlebot-2 is navigating on Gazebo simulator while its current path is plotted on RViz.

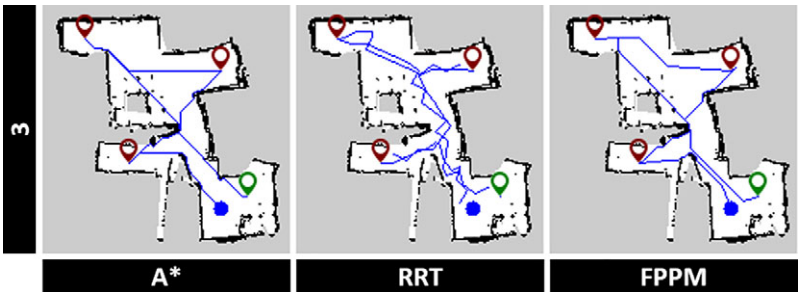

Fig. 6. Route scheduling – House. Tours generated to cover three target locations from paths computed by A*, RRT, and FPPM with pre-built paths.

target locations selected by the user on the system, green marker is the return spot (depot), and the blue dot is the robot's current position in the environment.

We start by showing the results in terms of tour distance. Figure 6 presents the tours generated by our system using each path planning approach, on the house environment, with three target locations. The tour starts at the blue dot, passes through all red markers, and ends at the green marker. The number 3 on the left of the image identifies the amount of target locations for this test. The blue lines represent paths computed from one place to another in the tour. The first frame of Fig. 6 shows paths computed with A*, the second with RRT, and last using our approach FPPM.

The chart of Fig. 7 shows the total distance (in meters) given by the sum of the lengths of its inner paths. A* was slightly better than our FPPM, as it seeks for the optimal path. On the other hand, RRT aims at finding a path quickly by connecting tree nodes randomly, as a result, its paths and consequently the tours are longer. The tour distance with A*, RRT, and our method were respectively 17.95, 29.20, and 19.30 m.

For the WillowGarage environment, three scenarios were set with 5, 10, and 15 target locations each, where the initial position of the robot and return spot were the same to all, as shown in Fig. 8. The first row on top is the scenario with 5 target locations, the second scenario with 10 target locations is on the middle, and the last scenario shown at the bottom has 15 target locations. The columns represent each path planner: A*, RRT, and FPPM, respectively. Following the test illustrated in Fig. 6, blue lines are paths from one place to another that together form a tour. It is visible, for both Figs. 6 and 8, that A* paths are straighter while RRT paths are more sinuous. Paths created by A* are almost

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

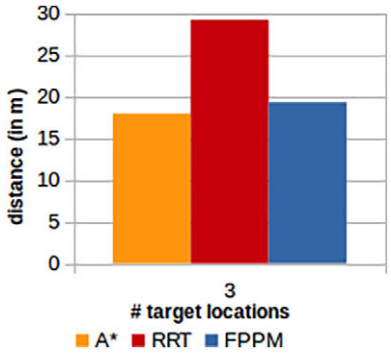

Fig. 7. Tour distance – House. Distances of each tour depicted in Fig. 6.

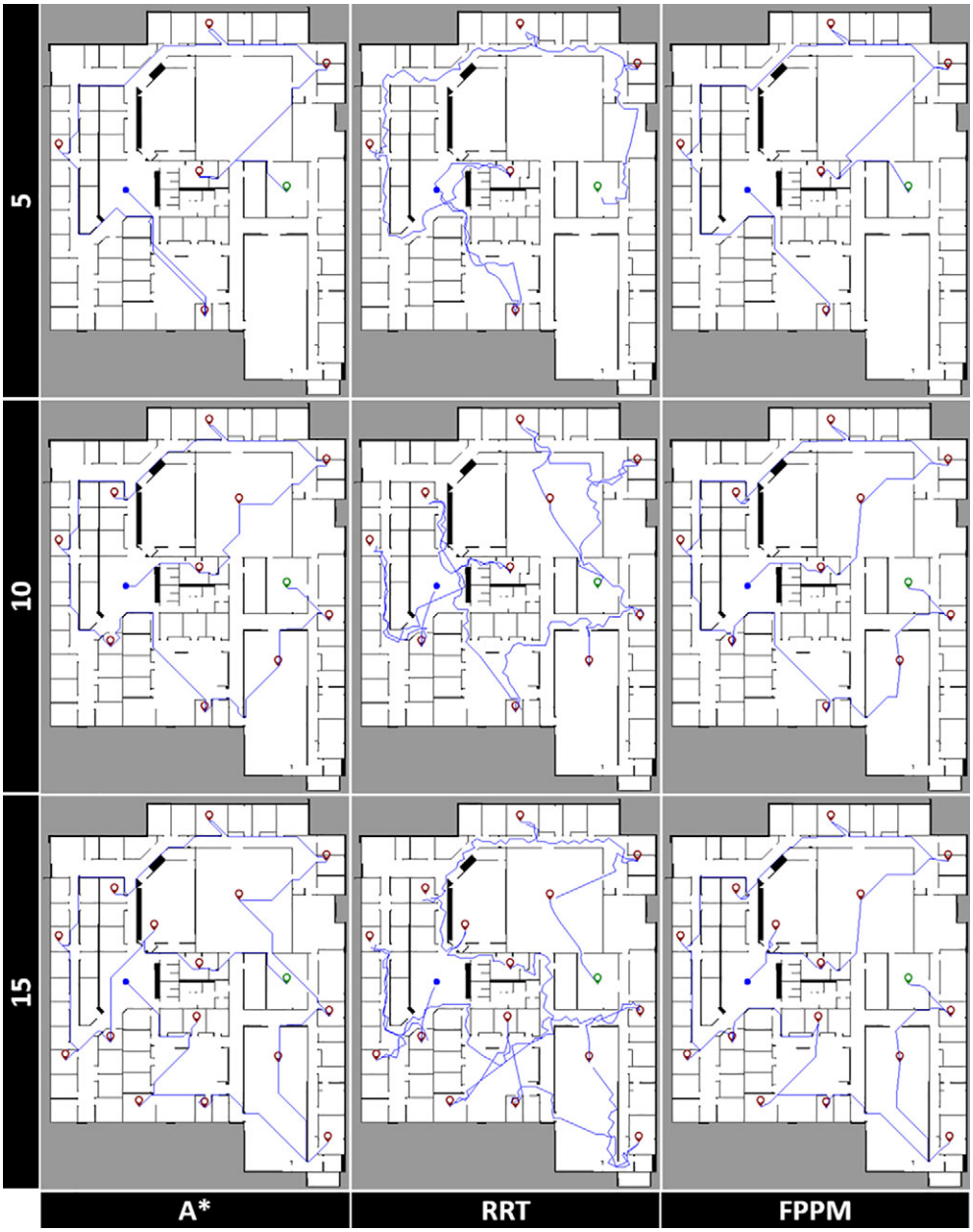

Fig. 8. Route scheduling – WillowGarage. Tours generated to cover 5, 10, and 15 target locations from paths computed by A*, RRT, and FPPM with pre-built paths.

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

Table I. Time to populate the Path Matrix for the scenarios of Figs. 6 and 8.

#Target locations	A*	RRT	FPPM
House (in s)			
3	0:18	3:21	**0.002**
WillowGarage (in min)			
5	2:31	0:38	**0:002**
10	5:20	2:06	**0:004**
15	11:55	3:39	**0:006**

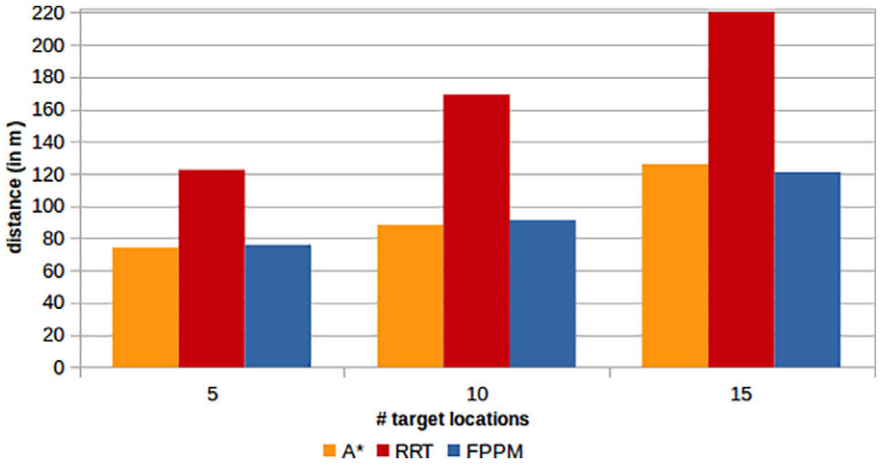

Fig. 9. Tour distance – WillowGarage. Distances of each tour depicted in Fig. 8.

the same of those created by our FPPM as they are derived from optimal paths computed off-line, also by A*. However, RRT is still losing quality in detriment of time.

The charts of Fig. 9 present the distance of each tour. The average difference between our method and A* was less than 1 m, and 74 m when compared to RRT.

Finally, we show the results in terms of time, which is crucial for on-line applications. Table I shows the time required to find all paths linking target locations, return spot, and the robot's position for each scenario, so they can be used to compute the tours presented in the experiments.

As the house environment is small and has only three target locations, the path planning is done quickly by all algorithms. After the path planning, the GA, described in Section 3.1, takes 1 s to find the best tour. However, for the scenarios of WillowGarage environment, the time to populate the path matrix using A* and RRT are not acceptable for real-time applications. While they take minutes, our approach can compute all paths in less than 1 s. In addition, the GA takes 1:50 s to find the best tour in this environment.

Consider a real situation in which the robot is executing some tour, and suddenly the user adds 10 new target locations. With A*, the robot will be stopped in idle for more than 5 min wasting battery and retarding the visiting, while our approach can reschedule the tour in less than 2 s.

The main reason why we chose A* and RRT to compare with our algorithm is because A* is a well-known path planner capable of finding optimal paths, that is, it guarantees to find the best solution among all possibilities. On the other hand, RRT is the most popular path planner in robotics today, as it is designed to efficiently explore paths in a high-dimensional space as fast as possible.

The experimental results show that our FPPM with pre-built paths can fill the path matrix almost instantly, faster than RRT. Moreover, optimal paths from A* are only slightly shorter than those generated by our approach. Therefore, our approach can be benefitted from both time and quality.

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press

5. Conclusion

Service Robotics represents a branch of the subject that aims to develop autonomous robots to help humans with their daily tasks. Many of these tasks require the robots to visit target locations within an indoor environment in order to perform some action in real-time. This problem can be modeled such as an instance of the TSP, in which target locations are cities and a robot is the salesman. The goal is to find the shortest tour that starts from the robot's current position towards a return spot, as a charging station, while visiting all target locations.

Our work is focused on finding paths that connect all target locations, robot positions, and return spot, as fast as possible, so they can be used on-line by a GA that combines them to find a tour. We introduce a FPPM that adjusts pre-built paths to speed up this process. This is done by storing in memory some optimal paths connecting strategic points of the map, such as the entrance of rooms and hallways, before the operation.

Experiments were carried out on virtual and real environments to evaluate time and quality of tours generated with A*, RRT, and our proposed approach FPPM. Both A* and RRT are often applied in robotics, especially in path planning problems. The first guarantees to find the optimal path but requires a lot of time, the other aims at finding a path quickly but not necessarily the shortest one.

The results shows that our method yields short tours almost immediately as it searches and links optimal paths, stored in memory, to the desired locations and combines them to form a tour. Therefore, we avoid to compute complex paths in real-time. For future work, we are intended to analyze other path planners and apply this technique on the multi-robot scenario.

References

1. T. Zielinska, "History of Service Robots," *Robotics: Concepts, Methodologies, Tools, and Applications* (IGI Global, 2014) pp. 1–14, doi: 10.4018/978-1-4666-4607-0.ch001.
2. R. Matai, S. Singh and M. L. Mittal, "Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches," **In:** *Traveling Salesman Problem: Theory and Applications.* (D. Davendra, ed.) (InTech, 2010) pp. 1–26.
3. P. E. Hart, N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths" *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968).
4. S. M. LaValle and J. J. Kuffner, "Rapidly-Exploring Random Trees: Progress and Prospects," **In:** *Algorithmic and Computational Robotics* (B. R. Donald, K. M. Lynch and D. Rus, eds.) (AK Peters/CRC Press, 2001) pp. 303–307.
5. D. G. Macharet, and M. F. M. Campos, "A survey on routing problems and robotic systems", *Robotica.* **36**(12), 1781–1803 (2018).
6. N. Bolourian and A. Hammad, "Path Planning of LiDAR-Equipped UAV for Bridge Inspection Considering Potential Locations of Defects," **In:** *Advances in Informatics and Computing in Civil and Construction Engineering (Proceedings of the 35th CIB W78 2018 Conference: IT in Design, Construction, and Management)* (I. Mutis and T. Hartmann, eds.) (Springer, Cham, 2019) pp. 545–552.
7. S. MahmoudZadeh, D. W. M. Powers and R. B. Zadeh, "Mission Planning in Terms of Task-Time Management and Routing," **In:** *Autonomy and Unmanned Vehicles: Augmented Reactive Mission and Motion Planning Architecture (Cognitive Science and Technology)* (Springer, Singapore, 2019) pp. 55–71.
8. M. Arzamendia, D. Gregor, D. G. Reina and S. L. Toral, "A Path Planning Approach of an Autonomous Surface Vehicle for Water Quality Monitoring Using Evolutionary Computation," **In:** *Technology for Smart Futures* (M. Dastbaz, H. Arabnia and B. Akhgar, eds.) (Springer, Cham, 2018) pp. 55–73.
9. A. Noormohammadi-Asl and H. D. Taghirad, "Multi-goal motion planning using traveling salesman problem in belief space" *Inf. Sci.* **471**(1), 164–184 (2019)
10. V. Sathiya and M. Chinnadurai, "Evolutionary algorithms-based multi-objective optimal mobile robot trajectory planning", *Robotica.* **37**(8), 1363–1382 (2019).
11. F. J. Abu-Dakka, F. J. Valero, J. L. Suñer and V. Mata, "A direct approach to solving trajectory planning problems using genetic algorithms with dynamics considerations in complex environments", *Robotica.* **33**(3), 669–683 (2015).
12. S. Azimi, M. S. Zainal Abidin and Z. Mohamed, "Solving an agricultural robot routing problem with binary particle swarm optimization and a genetic algorithm," *Int. J. Mech. Eng. Rob. Res.* **7**(5), 521–527 (2018), doi: 10.18178/ijmerr.7.5.521-527.
13. O. Tsilomitrou, N. Evangeliou and A. Tzes, "Mobile Robot Tour Scheduling Acting as Data Mule in a Wireless Sensor Network," *Proceedings of the 5th International Conference on Control, Decision and Information Technologies (IEEE-CoDIT)*, Thessaloniki, Greece (2018) pp. 327–332.
14. J. Cheng, Z. Miao, B. Li and W. Xu, "An Improved ACO Algorithm for Mobile Robot Path Planning," *Proceedings of the International Conference on Information and Automation (IEEE-ICIA)*, Ningbo, China (2016) pp. 963–968.

15. W. Ghadiry, J. Habibi, A. G. Aghdam and Y. Zhang, "Time-Efficient Trajectory Optimization in Patrolling Problems with Non-Prespecified Depots and Robots," *Proceedings of the 24th Mediterranean Conference on Control and Automation (IEEE-MED)*, Athens, Greece (2016) pp. 1047–1052.
16. C. Le, H. X. Pham and H. M. La, "A Multi-Robotic System for Environmental Cleaning," *CoRR* available at: http://arxiv.org/abs/1811.00690 (2018).
17. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st edition, (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989).

https://doi.org/10.1017/S0263574720000594 Published online by Cambridge University Press