Optimal Multi-Agent Coverage and Flight Time with Genetic Path Planning

Jacob M. Olson[1], Craig C. Bidstrup, Brady K Anderson, Alan R. Parkinson, Timothy W. McLain[2]

Abstract— **When generating 3D maps with unmanned aerial vehicles (UAVs), it is important for the mapping algorithm to have good coverage of the environment. It is also important, especially when planning paths for multiple agents, to have loop closures along each flight path and with other agents. Because multirotor UAVs are limited in flight time, the flight paths must be limited in length. Generating a good flight path to map a new environment can be difficult and tedious because of the free-form nature of a flight path. To solve this problem, we propose using a genetic algorithm designed to maximize total area coverage while minimizing flight time and enforcing sufficient loop closures. The natural ability of genetic algorithms to rapidly explore a design space is advantageous when solving complex free-form problems like path planning.**

I. Introduction

Autonomously generating an internal 3D map of a building requires intelligent control of the mapping vehicle. This ranges from manually planning a flight path with simple obstacle avoidance to full-stack exploration and path-planning algorithms.

Generating a flight path is a nontrivial task, especially when multiple objectives are considered. For example, the algorithm may want to maximize observed coverage of the space while minimizing flight time. This becomes tedious with each new space and more difficult as the complexity of the space increases with more rooms and inter-connecting hallways. When planning paths to achieve good coverage of the area rather than obstacle avoidance or traversability, it becomes much more complex. To plan effective flight paths in more complex flight spaces, using path-planning algorithms becomes a necessity.

When generating paths for the purpose of building a 3D map of the environment in a GPS-denied area such as indoors, enforcing loop closures becomes critical. Loop closures both between paths of different agents and between different segments of the same path are critical to successfully generate a usable map. Loop closures between different segments of the same path significantly reduce drift that occurs in visual odometry algorithms necessary for navigation and estimation with UAVs. Loop closures between paths of different agents ensure that the individual maps of each agent will combine into a single map.

There are several different approaches to coverage path planning that have been explored over recent years. These approaches range from 2D coverage where the robot must pass over all points in its known environment, common for uses such as cleaning robots or mine searching [1] to using coverage planning to generate 3D maps of outdoor terrain using viewpoint information from the robot's camera [2]. These approaches differ from the coverage planning desired for generating 3D maps of indoor environments with multiple UAVs in that they do not enforce loop closures in the generated paths.

The complexity of the design space renders many simple optimization routines improper for this problem. Further, by the structure of the problem, derivatives are not available for each design variable. Thus, a derivative-free approach is required. We chose to use a genetic algorithm because they are well suited to this type of problem. As outlined by Gen et al. [3], genetic algorithms are a powerful tool for solving complex, multi-objective optimization problems such as this. Genetic algorithms have been used in coverage path planning before, Yakoubi et al. details an approach to 2D coverage planning of a cleaning robot using a multi-objective genetic algorithm [4] and Hameed discussed using them for 3D terrain [5]. Similarly to [1], and [2], these approaches do not enforce loop closures and the objective is to visit all areas on the ground. Since the goal of this optimization is to plan a path that when flown will generate a high-fidelity 3D map of an indoor area with a forward-facing camera, viewing all the walls and obstacles is more important than covering all floor space. The genetic algorithm proposed in this paper takes these differences into consideration. We found our genetic algorithm was able to effectively plan single agent and multi-agent paths in an arbitrary environment and satisfy the constraints of the problem.

The remainder of the paper is organized as follows: Section II describes the method for simplifying the design space to create a reasonable problem for the genetic algorithm to solve. Section III describes the approach and architecture used to generate optimal paths for a single UAV, then the method is extended to multi-agent path planning. Results showing and evaluating the generated paths are presented in Section IV. Finally, conclusions are presented in Section V.

II. Optimization Setup

Modeling a problem for optimization requires a balance between fidelity and tractability. Model accuracy is especially important because any inaccuracies in the model can lead to solutions that are not truly optimal in the physical system. However, if the design space is too large, or ill formed, optimization algorithms can fail to converge. This is especially problematic when derivatives are not available, or there are multiple local optima.

A. Problem Statement

The goal of our optimization is to generate paths through a simple floor plan with the intent of building a 3D map using a UAV and a forward facing RGB-D (color and depth) camera. We start with a simple 2D floor plan of the area of interest and a known scale of the map. Then, with as little human interaction as possible, generate a flight path for the UAV to map the building, maximizing coverage while minimizing flight time so that the path is feasible with real hardware. We do this first with a single agent, then extend it to work for several agents collaborating to build a single map. A network diagram with the layout of the proposed optimization is show in Fig. 1. Each component of the network will be explained in detail in later sections.

B. Discretization

The open-ended nature of path planning can make optimization difficult. To simplify the design space, we discretize it by shrinking the map and constraining the path to only fly through predetermined waypoints before applying a genetic algorithm.

1) Map Generation: The first step to discretizing the design space is to simplify the 2D floor plan. The only part of discretization that must be done manually is segmenting explorable areas and occluded (inaccessible or uninteresting) areas on the 2D map. Fig. 2 illustrates how this segmentation works. The white space represents explorable areas and black space represents occluded areas.

Once this map is segmented, it is scaled to represent the minimum resolution desired in the optimization. This parameter is tunable to balance run time and fidelity. Once the map is properly scaled, we begin generating waypoints.

2) Waypoints: To further discretize and simplify the design space of the optimization, we place waypoints on the map and constrain the flight path to fly only through these waypoints. First, a grid is overlaid onto the map with a resolution of one half the width of the narrowest accessible hallway. This ensures that there are waypoints in every hallway and room to make the space fully traversable. Next, all waypoints overlapping occluded space are removed.

After removing occluded waypoints, the remaining waypoints are further pruned and adjusted to better cover the space. To remove waypoints that would cause the UAV to collide with obstacles, we generate a safety buffer around all occluded space greater than half the width of the UAV. Any waypoint that falls within this safety buffer is moved perpendicularly away from the wall so that it is outside the safety buffer. The waypoints are then pruned to remove redundancy. The distance between adjacent waypoints is checked and any that are closer than the initial grid resolution are replaced with a single waypoint at the centroid of the cluster. This prevents over-crowding of waypoints in hallways and corners to make it simpler to generate flight paths in these areas. We create another waypoint grid that also removes waypoints that are not near walls to simplify the design space even more. Fig. 3 shows an example of this waypoint pruning process. We chose to do this simplified discretization approach rather than computing Voronoi graphs of the map as done by Sipahioglu et al. [6] which tend to collapse rooms down into few waypoints. We wanted to allow more thorough exploration in open areas such as large rooms.

C. Objective Functions

We split the optimization into two competing objectives: The first maximizes coverage of the given map, and the second minimizes flight time. The maximin fitness function, which will be described in Section III-A.5, is used to determine the relative weighting between the two objectives.

1) Maximize Coverage: The first objective is to maximize coverage of the environment. To simplify the problem, we assume the UAV only flies forward. We use the camera's minimum and maximum viewable depths (d_{min} and d_{max}) and horizontal field of view (θ_{fov}) to construct the camera viewable area. We control the direction of the camera at each waypoint to face directly towards the next waypoint in the path. Then at each camera pose, we subtract all explorable area occluded by obstacles from the viewable area by dividing the viewable area into m sections and setting the distance of each section $d_{view,i}$ to

$$d_{view,i} = \max(\min(d_{max}, \min(d_{obs,i})), d_{min}) \text{ with } i = 0...m, \tag{1}$$

where $d_{obs,i}$ is an array composed of the distances to all obstacles in section i. As m increases, the coverage calculation becomes more accurate, but also more more computationally costly. An example of how the coverage is evaluated is shown in Fig. 4.

Next, we set the modified viewable area as explored at a waypoint. After repeating this process for every waypoint in the path, we calculate the percent of the explorable space in the map that is seen in that path. Mapping and visual odometry algorithms work better when the color and depth information from the camera is feature rich. Because of this, flying near walls is more beneficial than flying in open space like the middle of large rooms. To capture this, we compute a second coverage value, which is the percent of safety buffer (computed when placing waypoints) seen by the path. We then perform an alpha blend (shown in equation 2) on the two coverage types to get a single value that is used as the first objective value

$$\mathcal{C} = \alpha_{cov}\mathcal{C}_{open} + \left(1 - \alpha_{cov}\right)\mathcal{C}_{walls}, \tag{2}$$

where $\mathcal{C}$ is the value used for the coverage constraint, $\mathcal{C}_{open}$ is the coverage of the open explorable space, $\mathcal{C}_{walls}$ is the coverage of the safety buffer, and α_{cov} is a tunable parameter to weight the importance of walls versus empty space in the coverage calculation. See Fig. 5 for an example path and corresponding coverage from this step.

2) Minimize Flight Time: The second objective function minimizes flight time. To achieve this, we assume a constant-velocity flight and compute the total distance flown in each path. To avoid excessive meandering and incentivize flying

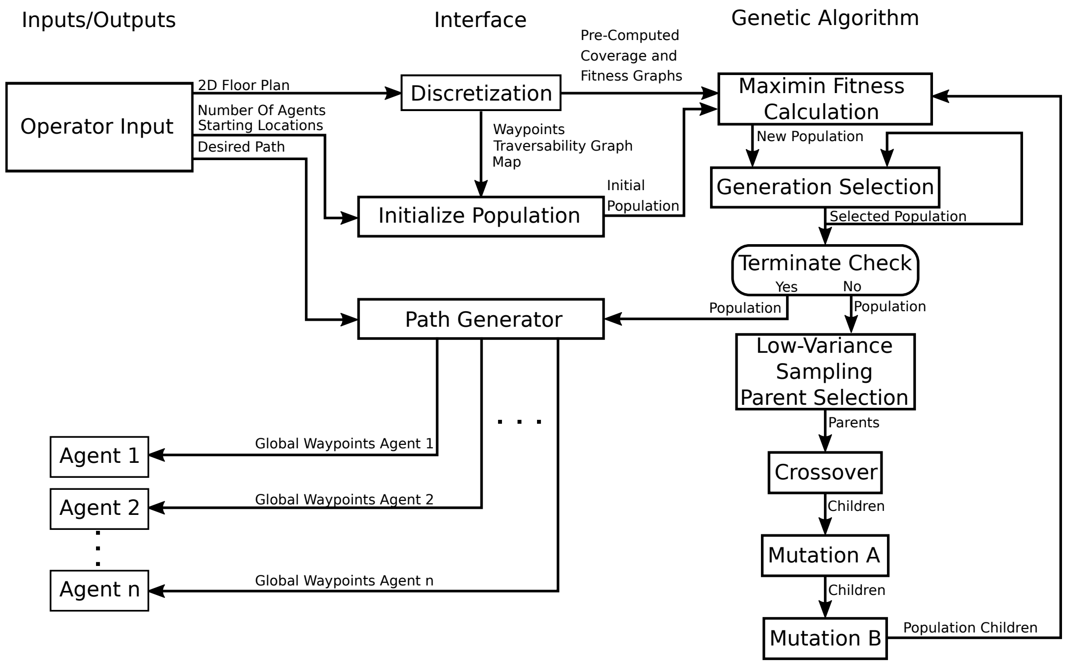


Fig. 1. The network diagram for the proposed coverage planner showing the inputs and outputs of each component.

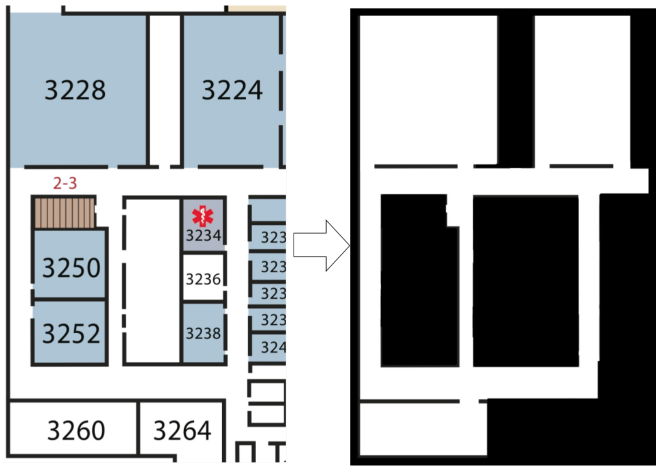


Fig. 2. The initial 2D floor map of the area of interest and the segmented floor map reflecting explorable and occluded spaces.

straight lines, we also added a turning cost proportional to the change in heading angle between waypoints. We apply the cost as an exponent of the change in heading angle by $\frac{\pi}{2}$ to heavily penalize turning more than 90 degrees at any time as given by

$$\mathit{flight\ time} = \sum_{i=0}^{l} \left(d_{w,i} + \rho \left| \frac{\Delta\psi_i}{\pi/2} \right|^{\rho} \right), \tag{3}$$

where l is the total number of waypoints in a given path, $d_{w,i}$ is the distance between the current waypoint and the next waypoint, ρ is a tunable parameter to weight the cost of turning, and $\Delta\psi_i$ is the difference in heading angle between the current and next waypoint. This favors paths with fewer waypoints and less turning.

D. Traversable Graph

To save computation when generating or modifying paths, we pre-compute the traversability graph. The UAV is allowed to move to any waypoint defined in the traversability graph for its current position. For any given waypoint, we want to constrain movement only to adjacent waypoints that are not obscured by obstacles. To achieve this, we split up the space around each waypoint into octants. Then, we find the nearest waypoint in each octant and add an edge between it and the current waypoint. To prevent flying into obstacles, we prune the graph to remove any traversals that collide with obstacles.

Using this traversability graph, we generate feasible paths for the UAV. Given a desired path length and starting location, a random waypoint is selected from the traversability graph. The probability of choosing the next waypoint favors forward motion, with probability decreasing as turning angle increases to reduce meandering. The probability drops off according to a logarithmic scale with a tunable weight to control how much the path will favor forward movement. To avoid backtracking, we also encode a short-term memory so that the UAV will not return to a recent waypoint unless there is no other choice.

Because of the computational cost of computing the modified viewable area at each waypoint, we use the information in the traversable graph to precompute the modified viewable area for every possible waypoint traversal. We also do this for the distances and angles between traversable waypoints used in computing flight time. By pre-computing this information and storing it, the optimization is able to run significantly

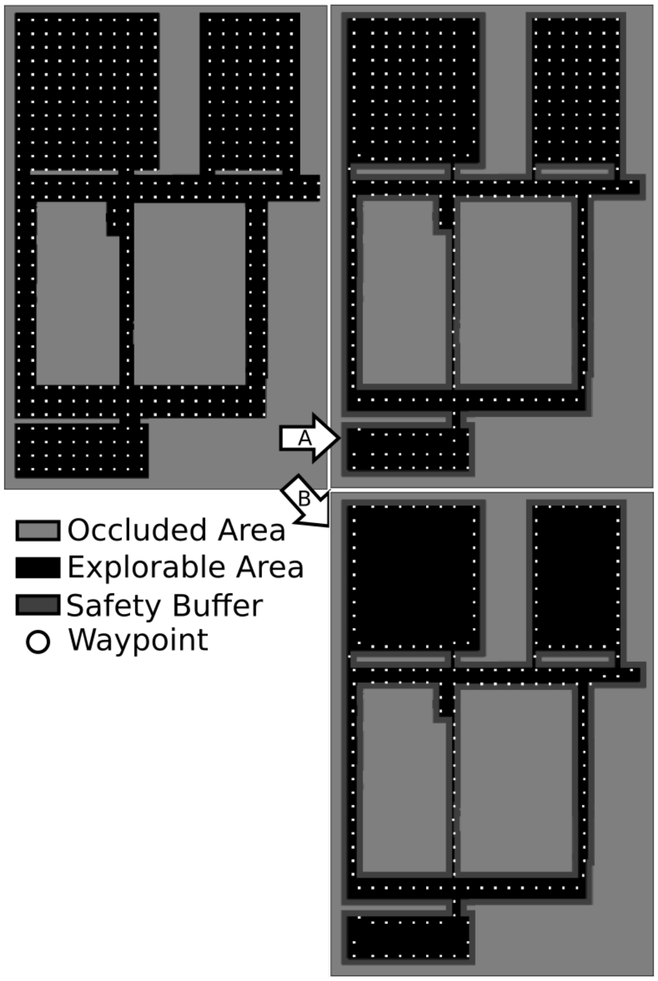


Fig. 3. The initial waypoint grid with only occluded waypoints removed (upper-left), the modified waypoint grid after applying the safety buffer and anti-crowding (upper-right), and the modified waypoint grid after applying safety buffer, anti-crowding, and pruning waypoints not near walls (lower-right).

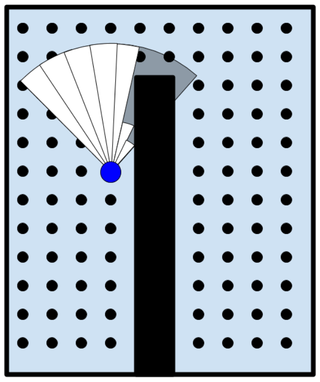


Fig. 4. An example of how the coverage would be evaluated at a single waypoint with m=7. The blue dot is the agent at a waypoint, the white area is what would be counted as viewed and the grey area represents the maximum viewable area.

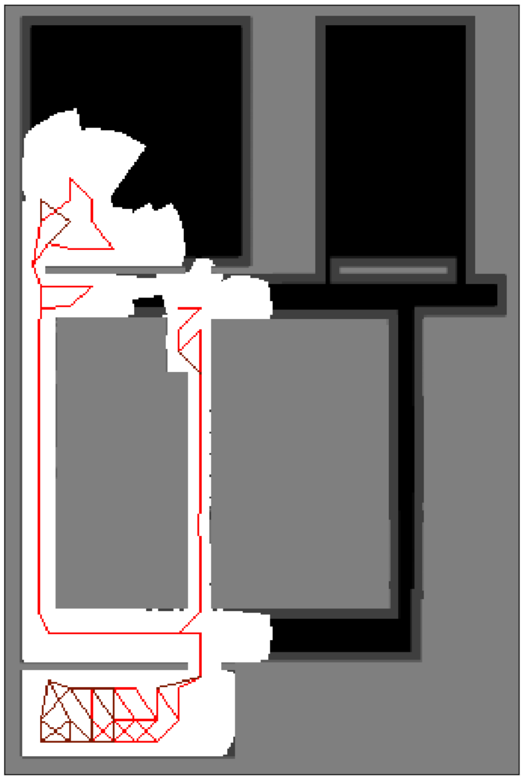


Fig. 5. An example path and its corresponding coverage. The red line is the path flown, light grey represents obstacles, white represents explored area, black represents unexplored open space, and dark grey represents unexplored walls

faster.

III. Approach

A. Single Agent Architecture

Performing coverage planning for a single agent is simpler than multi-agent planning. We begin by detailing the architecture for a single agent as follows. The methodology of genetic algorithms is to structure the optimization problem in a way that mimics the genetics and evolution process observable in nature. The variables of the optimization are stored in a chromosome data structure similarly to how genes are stored in nature. To enforce survival of the fittest, fitness pressure is applied through mate selection. Then to move from generation to generation, reproduction mechanisms such as gene crossover and mutation are implemented on the new organisms introduced to the population from mating. Fig. 1 shows the flow of the genetic algorithm used in this research.

1) Chromosome: The chromosome defining each member organism in a generation has a minimum, initial, and maximum number of genes. Each gene represents one waypoint in a path and its value encodes a location in the grid of possible waypoints, as seen in Fig. 3. The first generation of parents is spawned by generating a random path within the 2D map. The first step to create this path is to seed the same initial waypoint to each organism in the generation's population, representing the idea that the doorway to enter the building

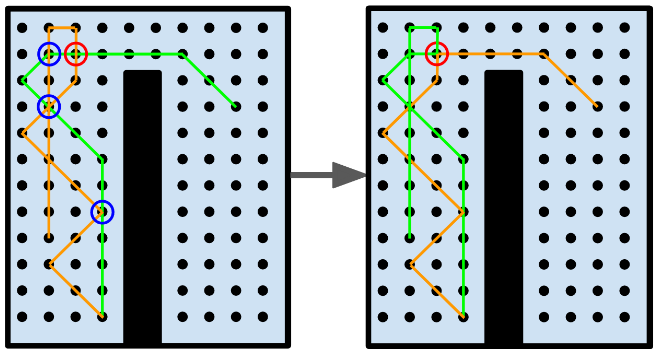


Fig. 6. An example of the crossover mechanism. The orange and green paths represent the paths of the selected parents. First, common waypoints are found (marked in blue and red). Next, one of the common waypoints is chosen at random (red). Finally, the tails of the two paths are swapped, as shown on the right.

is always in the same place. After the initial waypoint is seeded, the traversability graph is used to populate the rest of the path. All organisms in the first generation start with the same initial number of waypoints, however, the chromosome length is allowed to vary in crossover and mutation between the minimum and maximum number of genes. Once the chromosome is defined, both the coverage and flight-time objective values are computed for that organism. To start the first generation with only feasible organisms, any chromosomes that do not satisfy the starting constraints are discarded and a new organism is generated in its place. This process is repeated until the desired number of organisms has been added to the first generation.

2) Crossover: Given a set of two parents, crossover creates two children. Crossover occurs with a defined probability. If crossover does not occur, the children are clones of the parents. Otherwise, a search is done along the length of both chromosomes to determine where, if at all, both paths cross through the same waypoint. Next, one of the common waypoints is randomly chosen, and the tails of the chromosomes are swapped. See Fig. 6 for an example. If at any point a chromosome becomes longer than the maximum allowable length, it is truncated to the maximum length. Similarly, if the chromosome becomes shorter than the minimum allowable length, the chromosome is lengthened with a random feasible path up to the minimum chromosome length. To reduce the frequency of truncation and regeneration of paths, a parameter is set to only select points for crossover if they are within a set range of each other.

3) Mutation: After a new chromosome is generated, either by cloning or crossover, it is subjected to two different types of mutation ($\mathcal{A}$ and $\mathcal{B}$), each with a defined probability of occurring. The first type of mutation ($\mathcal{A}$) randomly selects both a random gene in the chromosome and a random length between the current and maximum chromosome length. Using the traversable graph, a new path is randomly generated from that gene onward, up to the randomly chosen length. See Fig. 7 for an example of this style of mutation.

The second mutation ($\mathcal{B}$) is intended to straighten and

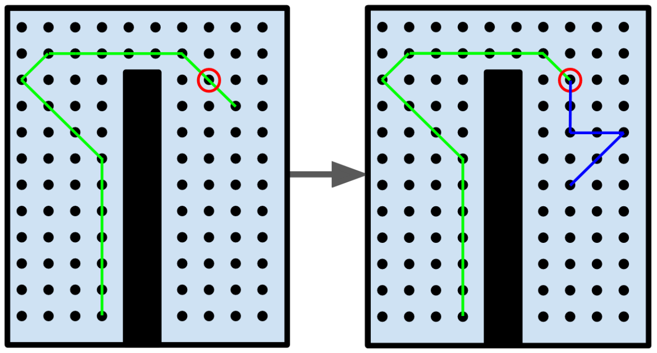


Fig. 7. An example of the Mutation $\mathcal{A}$ mechanism. A random waypoint in the path is chosen (marked in red) and the tail is replaced with a random path of a random length.

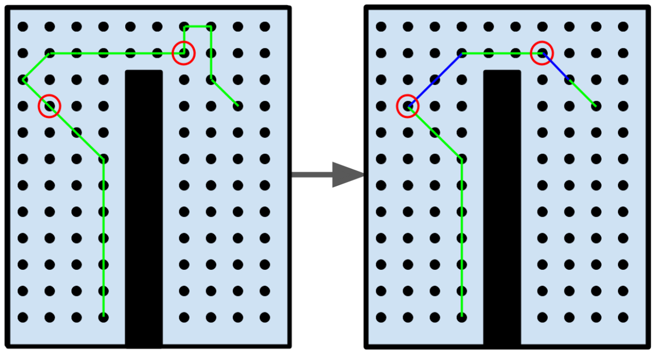


Fig. 8. An example of the Mutation $\mathcal{B}$ mechanism. Multiple random points are chosen (marked in red). From each point, the path is searched up to a certain length to find places where shortcuts or alternate paths can be taken (shown as blue segments).

shorten the paths. A random segment (random in position, static in length) of the chromosome is selected. The Hamming distance from the first waypoint in the segment to all other waypoints in the segment, starting with the last point, is computed. If a point is found to be within two waypoints of the first point in the segment, all waypoints in the segment between those two are replaced with a shortcut, as illustrated in Fig. 8. This is only completed according to a defined sub-probability of mutation, which is typically much higher than the overall mutation probability. The segment is left as is if no points are within a Hamming distance of two. This type of mutation is applied to the chromosome a specified number of times with the intent of finding multiple useful shortcuts in the path.

4) Constraints: Most of the constraints of the system are modeled into the discretization, traversability graph, and upper and lower bounds of the chromosome length as described in Section II. However, constraining the minimum coverage and loop closures could not be applied directly to the model.

The coverage constraint was applied by adding to the cost to make it worse than the worst feasible design. Since we are using two objectives, we implemented this constraint by adding a large cost to the flight-time objective to guarantee that all organisms violating the constraint will be dominated

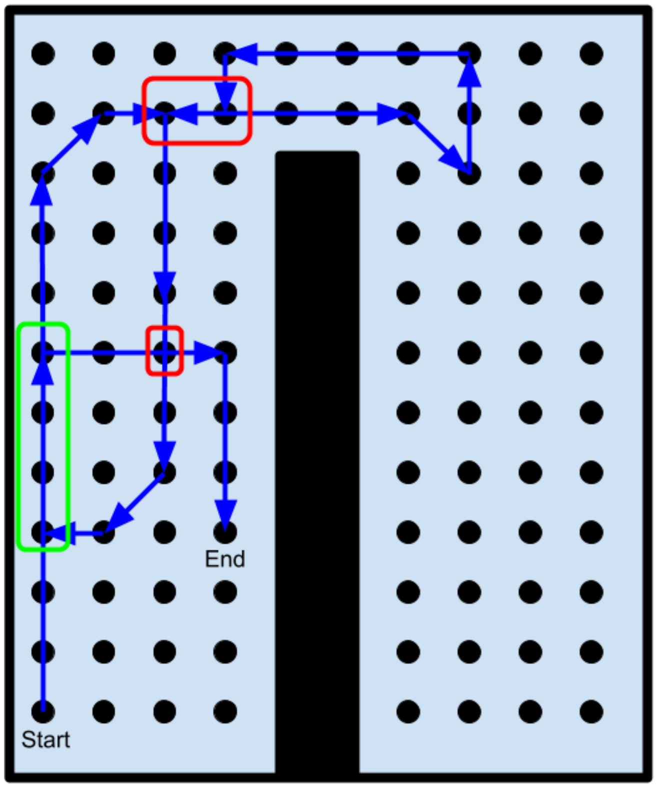


Fig. 9. An example of single agent loop closure. A valid loop closure is marked in green where the path passes through the same waypoints in the same direction. Examples of times when the path crosses over its self but not loop closures are marked in red.

by other designs (according to the maximin fitness).

Rather than start with a high constraint that would eliminate many of the original designs, we initialized the constraint with a low value. Over several generations we increase the constraint slowly to push the designs into the desired region of objective space. This also inadvertently increases the average flight time, but as the system continues to evolve, more efficient flight paths are found even with the new, strict coverage constraint and the average flight time decreases again.

We defined loop closures as anywhere that the path passes through the same waypoints in the same direction more than once as shown in Fig. 9. For a path to satisfy the loop closure constraint, it has to have a set number of loop closures. To avoid useless loop closures, a loop closure is only counted towards the constraint if it is separated by at least a defined number of waypoints. Similarly to the coverage constraint, we enforce this constraint by adding a large cost to the coverage, ensuring that all paths that violate the constraint are dominated by other designs.

As described in [7], constraints are applied to the fitness value. Because we have two objectives, and our non-modeled constraints directly relate to only one of the objectives, we implement these constraints on the objective values prior to computing the fitness value for the organism.

5) Maximin and Elitism: After all of a generation's children have been created, the fitness of the parents and children is computed by using the maximin fitness function as described by Balling [8]. In brief, the maximin scheme encourages a spread of designs across the Pareto front between competing objective values, with a target of minimizing the objectives. It does this by calculating the distance between each of the objective values of all the organisms, and finds the minimum difference of both objectives to each other. It then computes the maximum of all of these minima as the fitness value. Following the notation in [7], if f_k^i is the computed k^{th} objective value for organism i, the maximin fitness for organism i is computed by,

$$\mathit{fitness} = \max_{i \neq j} \left(\min_k \left(f_k^i - f_k^j \right) \right). \tag{4}$$

Once all parents and children have fitness values computed with respect to each other, the best N, comprising of the most negative fitness values, is selected as the next generation, where N is the size of a generation.

6) Low-Variance Sampling: We use a Roulette-style sampling method to pick parents in a way that gives organisms with better designs (determined by the maximin fitness) a higher probability of being selected as a parent for the crossover process. We employ a low-variance sampling technique as described in [9]. This method normalizes the population's fitnesses and treats them as a set of probabilities. A naive approach randomly samples N times from the pool of parents according to their respective probabilities. To reduce the possibility of a parent being sampled disproportionately to its probability, we draw a single uniform random number r between 0 and $\frac{1}{N}$. The probabilities are then stacked up and compared to r. Whichever organism corresponds to the bin that r falls into gets sampled and $\frac{1}{N}$ is added to r and again compared to the stack of probabilities. This process is repeated N times. We also apply fitness pressure with the low-variance sampling with a tunable parameter γ with a value between 0 and 1. If $\gamma = 0$, no fitness pressure is applied, and the sample is random. If $\gamma = 1$, it will only select the most fit organism for the next generation.

B. Multi-Agent Architecture

When modifying the algorithm to generate multi-agent paths, the crossover and mutation mechanisms remained largely unchanged. Each agent was treated separately and crossover and mutation were applied independently on each path. The maximin fitness function, elitism, and low-variance sampling all remained unchanged; however the chromosome and constraints did need to be modified to accommodate the multi-agent architecture. The following section details these changes and additions to the architecture to generate multi-agent paths.

1) Chromosome: The chromosome was expanded to be of size $n \times l_{max}$ where n is the number of agents and l_{max} is the maximum path length where each row represents the path of its respective agent. The genes each row have the same representation as in the single agent case. To accommodate for paths that are shorter than the maximum length, any path shorter than l_{max} is padded with -1 at the end to populate any unused genes in the chromosome. We also allow for the agents to either start at the same waypoint as each other (if there were only one entry to the map), or different waypoints (if there are different entrances into the map).

2) Constraints: The majority of the architecture change between the single-agent and multi-agent case was in modeling the constraints for the multi-agent case. The constraint on coverage was applied to the combined coverage of the full chromosome, not the individual paths. The loop closure constraint had to be expanded to handle loop closures between agents. For a single 3D map to be generated when dealing with more than one agent, the agents must have loop closures between them. For only two or three agents, this is simple to enforce. For four or more agents, graph theory is necessary to efficiently evaluate the connectivity of the paths.

To keep track of loop closures, we create a symmetric $n \times n$ loop closure matrix C with each element defined by

$$c_{ij} = \text{loop closures between agent } i \text{ and agent } j \qquad (5)$$
$$i, j = 0...n,$$

where n is the number of agents. The values along the diagonal of C are the number of loop closures the agent has with its own path. The off-diagonal terms store the total number of loop closures between agent i and agent j. The loop closure constraint from the single agent architecture is applied here using these values to ensure that paths have loop closure with themselves. We also added a new constraint to ensure the agents paths are connected enough to build a single map.

As described by [10], we create the $n \times n$ symmetric adjacency matrix A from the loop closure matrix C with each element defined by

$$a_{ij} = \begin{cases} 1, & \text{if } (c_{ij} \geq 1 \text{ and } i \neq j) \\ 0, & \text{otherwise} \end{cases} \qquad (6)$$
$$i, j = 0...n$$

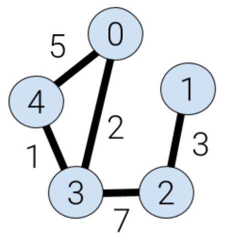
which encodes which agents are directly connected to other agents Then we create the $n \times n$ diagonal degree matrix D defined by

$$d_{ii} = \sum_{j=0}^{n} a_{ij}, \quad i = 0...n$$

so each element along the diagonal represents the number of agents that each agent is directly connected to through loop closure. Next, we define the graph Laplacian matrix L as

$$L = D - A \qquad (7)$$

which encodes the connectivity of the agents. We can determine whether there are enough loop closures between agents to connect all paths into a single map by looking at the eigenvalues of L. The number of zero eigenvalues represents the number of separate connected components in the graph. So, if all agents are connected to each other, there will be only one zero eigenvalue. If there are not sufficient loop closures to combine into a single map, there will be more than one zero eigenvalue. We use this to set the last constraint used for the multi-agent architecture. There must be exactly one zero eigenvalue of L to satisfy the constraint. An example connectivity graph for a system with five agents is shown in Fig. 10



$$C = \begin{bmatrix} 7 & 0 & 0 & 2 & 5 \\ 0 & 10 & 3 & 0 & 0 \\ 0 & 3 & 3 & 7 & 0 \\ 2 & 0 & 7 & 2 & 1 \\ 5 & 0 & 0 & 1 & 5 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Fig. 10. An example connectivity graph with five agents and its respective loop closure matrix C, adjacency matrix A, and degree matrix D. The agent number is labeled on each blue node and the number of loop closures between agents is shown with the black edges between nodes.

TABLE I
PARAMETERS USED TO OBTAIN OPTIMIZATION RESULTS.

Parameter	Value
Generation Size (Organisms)	100
Crossover Probability	0.7
Crossover Time Threshold (Waypoints)	70
Mutation $\mathcal{A}$ Probability	0.3
Mutation $\mathcal{B}$ Probability	0.3
Number of Mutation $\mathcal{B}$ Instances	20
Mutation $\mathcal{B}$ Acceptance Probability	0.8
Mutation $\mathcal{B}$ Search Distance (Waypoints)	5
Objective Scaling Factor: Flight Time	0.0001
Objective Scaling Factor: Coverage	-1.0
Coverage Blending Factor α_{cov}	1.0
Coverage Sections	15
Minimum Chromosome Length (Waypoints)	75
Maximum Chromosome Length (Waypoints)	250
Pixel Scale (meters/pixel)	0.15
Minimum Coverage Constraint Start	0.3
Minimum Coverage Constraint End	0.8
Minimum Coverage Constraint Aging (Generations)	60
Loop Closure Separation Threshold (Waypoints)	30
Minimum Loop Closures	2
Path Memory (Waypoints)	10
Turning Cost Factor ρ	8.0
Log Scale Turning Weight	0.7
Fitness Pressure Factor	0.5

IV. RESULTS

The designer is required to set many parameters for the genetic algorithm to work as derived in [7]. The parameters we used, along with parameters specific to our implementation, are outlined in Table I.

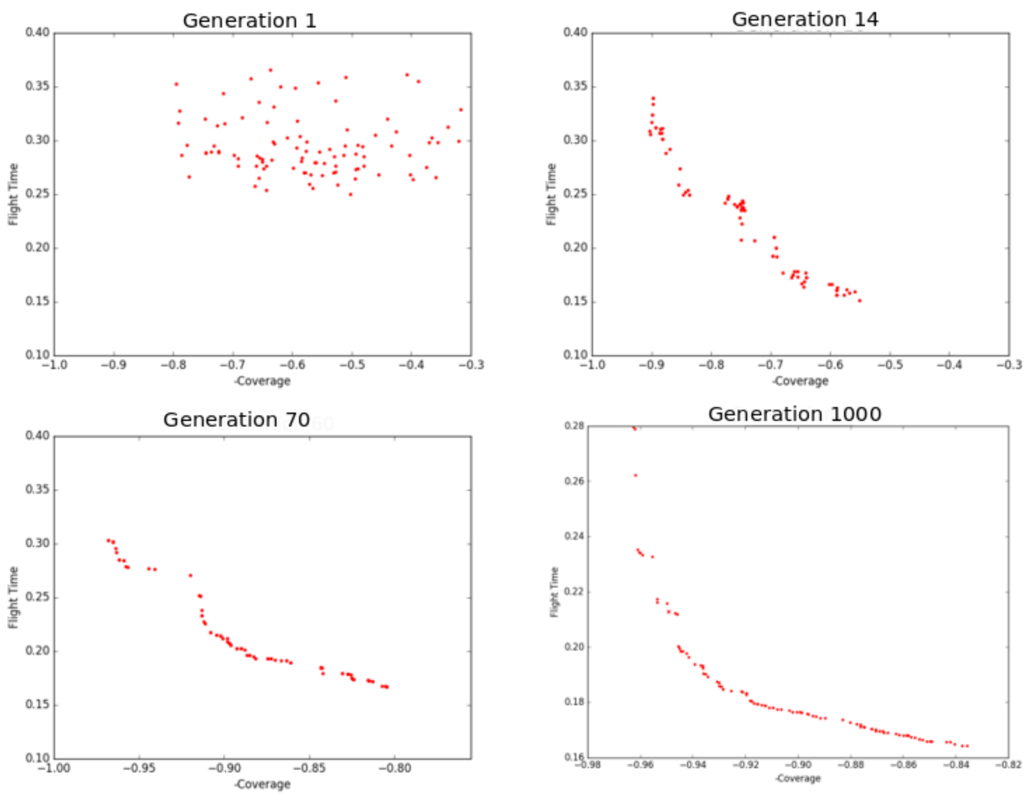


Fig. 11. History of the Pareto front across several generations. The dynamic constraint placed on coverage prioritizes high coverage but also induces high flight time for the first 1000 generations. Afterward, the algorithm can work with high coverage organisms, optimizing them for lower flight times, while maintaining their high coverage properties.

A. *Single Agent*

1) Fitness: Fig. 11 shows how the objective values of the organisms evolved from generation to generation. Four different snapshots are shown: the first, randomly generated set of parents, the state of the generation at about a third and two-thirds of the way through the selected number of generations, and last, the final result of the progression of 1000 generations. Fig. 12 shows how the average negative coverage and average flight time progresses over the course of the 600 generations. The coverage flattens out temporarily, but then continues to drop after 400 generations. The interesting result displayed is in the flight-time objective value history. After about 30 generations, the flight time starts to rise again. This is due to the dynamic constraint that invalidates all organisms with coverage below the defined threshold at the defined generation number. Once the coverage constraint settles to a constant value, the organisms with good coverage dominate the generation population and the algorithm successfully lowers the flight time again to represent a useful Pareto front that has a good spread of options from which to choose.

2) Paths Generated: Our algorithm was successful in generating useful paths, as seen in Fig. 15. The single agent path covers over 95% of the walls and wastes little time returning to parts of the map that have already been visited. The only exception is generating a useful loop closure in the center hallway that would help in generating a 3D map.

B. *Multiple Agents*

1) Fitness: Similar to the single agent case, our algorithm was successful in exploring the design space to generate paths that had both good coverage and low flight times. Fig. 13 shows the progression of the average objective values over 500 generations for a case with six agents on a much larger map. The six agents began distributed throughout the map, so the initial average coverage is much higher than the single agent case, so the average flight time does not have the same spikes as with the single agent case.

2) Paths Generated: Our algorithm was also successful in generating paths for two and three agents on the same map as shown in Fig. 15 and for six agents on a larger, more complicated map as shown in Fig 14. In the multi-agent case, the paths were not only able to create loop closures with themselves, they were able to connect with each other enough to create a single connected map between all agents.

V. Conclusions

We were able to successfully generate paths with sufficient coverage to generate a high-fidelity map of the environment both for the single-agent case and the multi-agent case. These paths also covered the environment efficiently to decrease the flight time necessary to map the environment. The results show that with a properly formulated genetic algorithm,

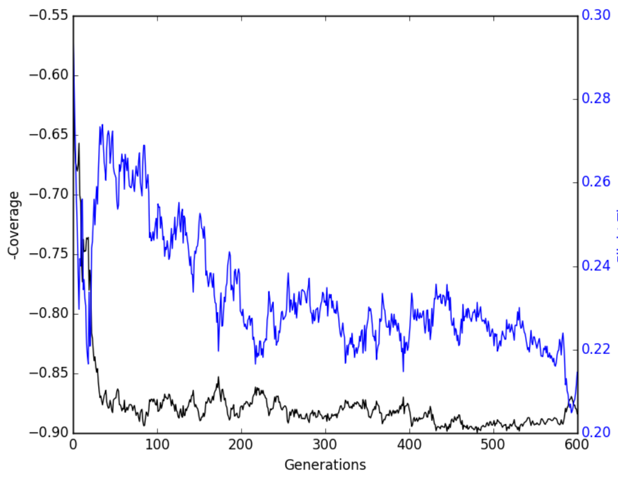


Fig. 12. A plot of the average objective values over 600 generations. Note how the average flight time initially decreases then sharply increases again, as the rolling constraint on coverage is applied. As the generations are then allowed to evolve, the flight time settles back down below the previous minimum.

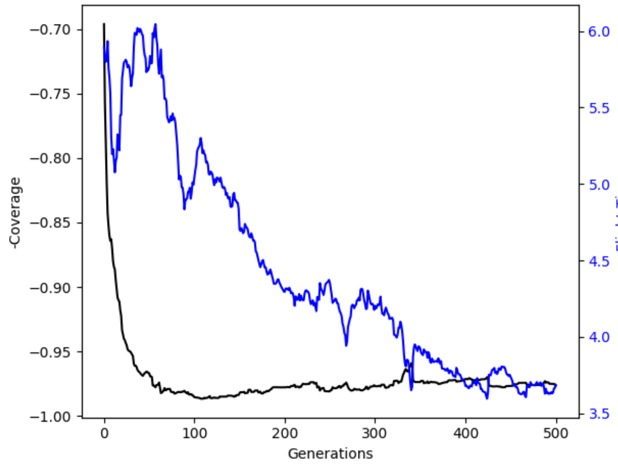


Fig. 13. A plot of the average objective values over 500 generations for six agents on a large map. Note how the average flight time initially decreases then sharply increases again, as the rolling constraint on coverage is applied. As the generations are then allowed to evolve, the flight time settles back down below the previous minimum.

efficient area-coverage paths can be generated to assist in building 3D maps in complex environments.

References

[1] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001. [Online]. Available: https://link.springer.com/content/pdf/10.1023{%}2FA{%}3A1016639210559.pdf

[2] A. Bircher, M. Kamel, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, "Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots," *Autonomous Robots*, vol. 40, no. 6, pp. 1059–1078, 2016. [Online]. Available: https://link.springer.com/content/pdf/10.1007{%}2Fs10514-015-9517-1.pdf

[3] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*. Wiley, 2000. [Online]. Available: https://www.wiley.com/en-us/Genetic+Algorithms+and+Engineering+Optimization-p-9780471315315

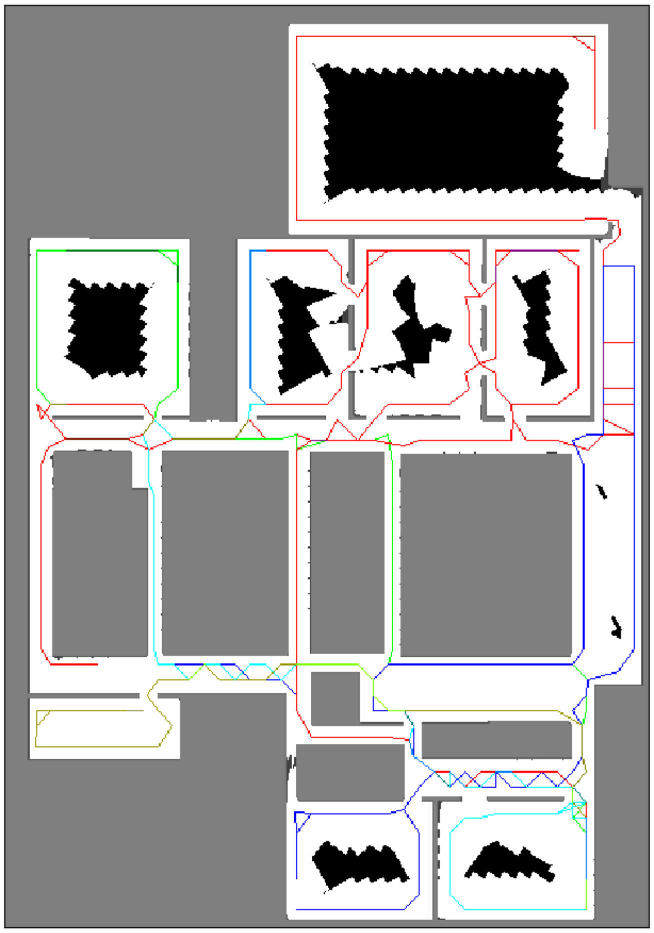


Fig. 14. An example path generated for six agents on a larger, more complex map. Each agent's path is represented by a different color. Darker colors indicate loop closure. Note the near perfect wall coverage and loop closures between all agents.

[4] M. A. Yakoubi and M. T. Laskri, "The path planning of cleaner robot for coverage region using genetic algorithms," *Journal of innovation in digital ecosystems*, vol. 3, no. 1, pp. 37–43, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352664516300050{#}br000025

[5] I. A. Hameed, "Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 3-4, pp. 965–983, 2014. [Online]. Available: http://link.springer.com/10.1007/s10846-013-9834-6

[6] A. Sipahioglu, G. Kirlik, O. Parlaktuna, and A. Yazici, "Energy constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach," *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 529–538, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889010000060

[7] A. R. Parkinson, *Optimization-based design version 3.2*. BYU Academic Publishing, 2019.

[8] R. Balling, "The maximin fitness function; multi-objective city and regional planning," in *International Conference on Evolutionary Multi-Criterion Optimization*, Springer. Springer, Berlin, Heidelberg, 2003, pp. 1–15. [Online]. Available: http://link.springer.com/10.1007/3-540-36970-8{_}1

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005. [Online]. Available: https://mitpress.mit.edu/books/probabilistic-robotics

[10] E. W. Weisstein, "Laplacian matrix." [Online]. Available: http://mathworld.wolfram.com/LaplacianMatrix.html

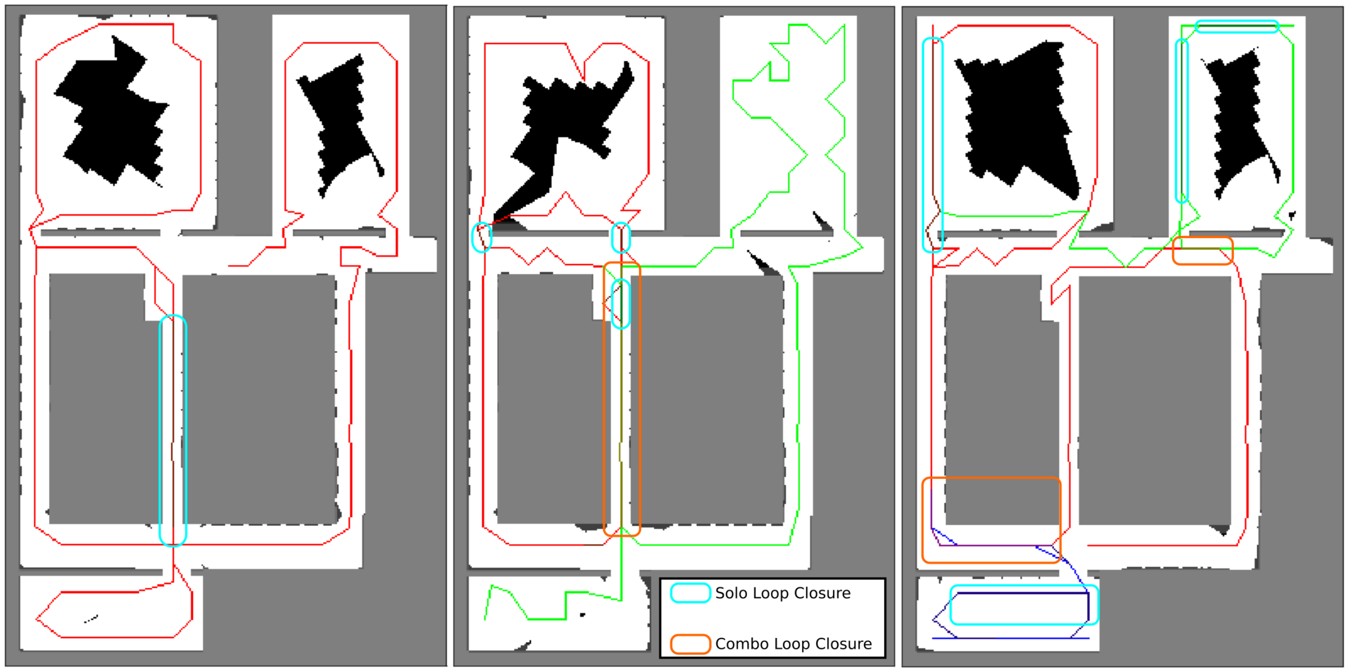


Fig. 15. Example of final paths generated by the genetic algorithm for a single agent (left), two agents (center), and three agents (right) on the same map. These paths show excellent wall coverage and loop closures both with themselves (shown in teal) and with each other (shown in orange) to generate a single well-structured map.