A Complete and Scalable Strategy for Coordinating Multiple Robots Within Roadmaps

Mike Peasgood, *Associate Member, IEEE*, Christopher Michael Clark, and John McPhee

***Abstract*—This paper addresses the challenging problem of finding collision-free trajectories for many robots moving toward individual goals within a common environment. Most popular algorithms for multirobot planning manage the complexity of the problem by planning trajectories for robots individually; such decoupled methods are not guaranteed to find a solution if one exists. In contrast, this paper describes a multiphase approach to the planning problem that uses a graph and spanning tree representation to create and maintain obstacle-free paths through the environment for each robot to reach its goal. The resulting algorithm guarantees a solution for a well-defined number of robots in a common environment. The computational cost is shown to be scalable with complexity linear in the number of the robots, and demonstrated by solving the planning problem for 100 robots, simulated in an underground mine environment, in less than 1.5 s with a 1.5 GHz processor. The practicality of the algorithm is demonstrated in a real-world application requiring coordinated motion planning of multiple physical robots.**

***Index Terms*—Complexity, mobile robots, path planning, trajectory planning.**

I. Introduction

THE USE of multiple mobile robots in a common environment is valuable for the automation of many operations, such as underground mining and warehouse management. In such applications, multiple vehicles are required to drive autonomously between different locations, preferably taking the shortest possible route while avoiding collisions with static objects and other vehicles. This paper presents an algorithm for efficiently determining collision-free paths for many vehicles in environments composed of tunnels or corridors, as may be found in these applications. The problem addressed by this research is demonstrated by the multirobot planning task pictured in Fig. 1(a).

In this scenario, the environment is constructed of corridors or tunnels that are wide enough for only a single robot to travel, and we assume differential drive robots that can rotate in place. The objective in this example is to shift the positions of each robot, such that robot R_1 moves to the initial position of R_3, R_3 to the position of R_2, and R_2 to the position of R_1. Our goal is to find an algorithm that is scalable to a large number of robots (>100) densely situated in a large environment, and can solve problems such as that shown in Fig. 1(a) that require specific coordinated planning.

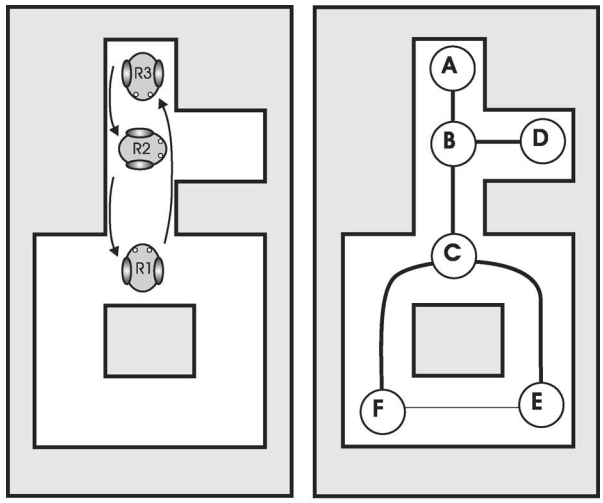


Fig. 1. Multirobot planning problem requiring coordination of three robots, and a graph-based representation of the environment. (a) Planning problem. (b) Graph-based map.

Many methods have been proposed for planning the motion of one or more robots; refer to [15] and [16] for broad reviews of the subject. Planning algorithms can be evaluated in terms of completeness (whether they are guaranteed to find a solution if one exists), complexity, and optimality.

Most multirobot planning algorithms fall into one of two categories, *coupled* or *decoupled*. *Coupled* algorithms, such as [22], plan the trajectories of all robots in the environment concurrently. By combining the states (poses) of the individual robots together into a system state representation, a sequence of state transitions can be found that will move all robots to their respective goals. Using complete search methods, such as A* [11], coupled algorithms can achieve completeness and optimality, and can solve the problem shown in Fig. 1(a). Coupled algorithms depend on a centralized architecture, where all of the state information is available to a single processor. Their limitation is in searching the large configuration space that grows in dimension as each additional robot is added to the environment. A direct application of the A* search would guarantee

Manuscript received February 17, 2007; revised October 8, 2007. This paper was recommended for publication by Associate Editor F. Lamiraux and Editor L. Parker upon evaluation of the reviewers' comments. This work was supported in part by the National Sciences and Engineering Research Council of Canada (NSERC) and Ontario Centres of Excellence (OCE).

M. Peasgood was with the Department of Mechanical and Mechatronics Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada. He is now with Aeryon Labs Inc., Waterloo, ON N2V 1A2, Canada (e-mail: peasgood@alumni.uwaterloo.ca).

C. M. Clark is with the Department of Computer Science, California Polytechnic State University, San Luis Obispo, CA 93407 USA (e-mail: cmclark@calpoly.edu).

J. McPhee is with the Department of Systems Design Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: mcphee@real.uwaterloo.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2008.918056

a resolution-complete solution. However, since the size of the configuration space (the number of possible states of the system) grows exponentially with the number of robots $[O(k^r)$ for r robots], the computational complexity of the A* search also increases exponentially and quickly becomes intractable. Hopcroft *et al*. have shown the general motion planning problem for multiple moving objects to be PSPACE-hard [12]. One approach to reduce the size of the search space is to create probabilistic roadmaps (PRMs) through the environment; this method was shown in [22] to be probabilistically complete and demonstrated in simulation for up to five robots. Another approach is to decompose a large map into subgraphs, and plan paths between subgraph segments before coordinating motion within each subgraph [20].

Decoupled methods plan for the motion of individual robots, rather than planning the motion of all robots simultaneously. One approach is to decouple path planning from mutual collision avoidance, by first finding obstacle-free paths, and then, adjusting velocities of individual robots to avoid collisions [10], [13], [19]. Alternatively, a *coordination-diagram* [18] approach can be used to independently combine generated paths of many robots while avoiding collisions [21].

Decoupled methods may use a decentralized architecture, allowing independent planning-based methods such as maze searching [17] or potential fields [3], [8], or they may use a centralized architecture planning for all robots with a single processor. Centralized decoupled planners typically determine individual trajectories sequentially and combine the plans of all robots to avoid collisions. Plans may be combined by iteratively adding new plans as obstacles into the configuration space–time [7]; however, this inherently involves assigning priorities to robots to determine the order in which plans are added, which affects the quality of the resulting plan. This can be addressed by considering all different combinations of priorities (for up to three robots, demonstrated in [2]), or running an optimization process on the priority assignment [4]. In a more dynamic paradigm, the plans of individual robots can be merged into the global coordination plan as new goals are assigned [1].

By planning the motion of robots sequentially, decoupled methods have lower complexity and greater scalability than a coupled planner; however, this comes at the cost of completeness and optimality. The problem in Fig. 1(a), for example, cannot be solved by a sequential planner. By selecting the optimal plan for any robot independently, an obstacle is created in the space–time map that cannot be avoided by the other two robots.

This paper presents an alternative *multiphase* planning method that can solve these coordinated planning problems, and is scalable to a large number of robots in a large environment. A graph representation of the environment is first created, and a spanning tree through the graph is selected. A multiphase planning approach then takes advantage of the properties of the graph and spanning tree to create and maintain obstacle-free paths while robots move to their respective goals.

A topological graph of the environment (a *roadmap*) is a prerequisite for this planning algorithm, in which the current and goal positions of robots are identified by nodes in the graph. For the derivation and simulation of the method in this paper, a graph is generated for two-dimensional environments, assuming circular robots capable of translation and in-place rotation. For more general problems, the method presented here can be applied directly once a suitable roadmap has been constructed.

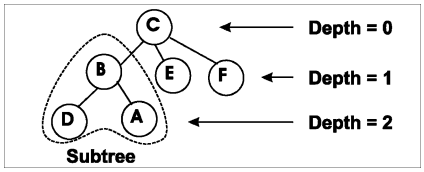


Fig. 2. Spanning tree T^* for the graph representation of the environment rooted at node C, and a subtree T_B rooted at node B.

II. Map Representation and Tree Selection

Map representation is a significant factor in the efficiency of motion planning algorithms. By abstracting the structure of the environment to a set of open spaces (nodes) connected by corridors or tunnels (edges), a graph representation of the environment reduces the number of possible states of the system, and therefore, reduces the complexity of the search for collision-free paths.

Several methods of generating roadmaps have been developed for different applications, such as PRMs [14] and Voronoi graph planners [5]. The planning algorithm presented in this paper requires such a roadmap, but is independent of the particular method used to generate it.

For the example of Fig. 1(a), assuming circular robots that have omnidirectional or differential drive kinematics (so that they can turn in place), a graph G can be constructed by hand as shown in Fig. 1(b), consisting of $N = 6$ nodes and $E = 6$ edges. We assume that the initial and goal positions of all robots lie on the nodes of the graph; in this representation, the goal positions of robots R_1, R_2, and R_3 are nodes A, C, and B, respectively.

Given the graph representation, we can also select a spanning tree T^* in the graph, that is, a subset of edges connecting all nodes without forming any loops. A given spanning tree has L *leaf nodes* (nodes with only one incident edge) and $N - L$ *interior nodes*. A suitable spanning tree for the example is shown in Fig. 2 where node C, closest to the geographic center of the map, is selected as the root. Selecting all edges except for $E - F$ into the spanning tree as shown gives $L = 4$ leaf nodes, A, D, E, and F, and two interior nodes, B and C.

In general, the spanning tree is not unique, and a heuristic approach for tree selection is used that tends to maximize the number of leaves and minimize the distance between leaves. We have found that an effective approach is to iteratively add edges to the tree that lead to the nodes with the maximum number of incident edges, starting from the root node. Again, the planning algorithm requires the selection of a spanning tree, but is independent of the tree selection method used.

III. Multiphase Planning Algorithm

The multiphase algorithm finds a feasible solution to the multirobot trajectory planning problem by breaking the problem into a sequence of four subproblems. Each phase can be solved in

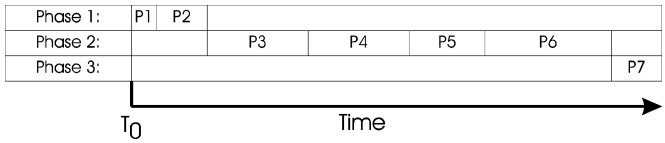


Fig. 3. Each path segment P_i indicates the motion of one robot. In Phases 1–3, individual collision-free segments are planned and concatenated in time.

time proportional to the number of robots by taking advantage of the graph and spanning tree structures developed earlier.

A plan is first found that moves the robots to the leaves of the spanning tree (Phase 1 of the algorithm). We then use the following observations to plan a sequence of paths to drive each robot to its goal. For a system with $r < L$ robots, we have the following.

Lemma 1: When all robots occupy leaf nodes, any robot can move to any interior node in the graph G.

Lemma 2: When all robots occupy leaf nodes, any two robots can swap positions.

Lemma 1 is clear since an obstacle-free path can be found between any two nodes through the spanning tree T^*, and no robots remain as obstacles on the interior nodes of the tree. Lemma 2 follows, since with $r < L$ robots, there is always one unoccupied leaf N_{tmp} in the spanning tree. Robots R_i and R_j at nodes N_i and N_j can swap positions by moving R_i to N_{tmp}, R_j to N_i, and R_i to N_j.

Note that these lemmas guarantee that there exists at least one path through the spanning tree. However, a shorter path may exist using graph edges that are not in the tree [e.g., moving from E to F in Fig. 1(b)]. Where an A* search is used in the following steps, the entire graph is searched, and the shortest paths will be selected.

As described in detail later, a plan is constructed by first building a sequence of individual paths, or *segments*, in which one robot moves between two nodes (as shown in Fig. 3). Once all robots have been moved to the leaves of the tree in Phase 1, the aforementioned lemmas guarantee that the robots can be arranged in the graph such that every robot will have an obstacle-free path to its goal. This is accomplished in Phase 2 by moving each robot to a node within a subtree of its goal. In Phase 3, we can then move each robot in sequence to its goal. Finally, in Phase 4, the time and distance required to complete the sequence of individual robot movements can be reduced by removing redundant motions and moving robots concurrently whenever possible.

The pseudocode given next assumes the following functions are available.

current Node (*robot*): returns the node occupied by *robot* at the current time step of the plan.

freeLeaf Node(): returns an unoccupied leaf node of the spanning tree. The *freeLeaf InSubtree*(*node*) and *freeLeaf NotInSubtree*(*node*) functions perform the same search, restricted to the subtree of *node*, or the subset of the graph not in the subtree of *node*, respectively.

astarPath(*start, end*): returns the shortest connected sequence of nodes between nodes *start* and *end*, assuming no obstacles in the graph.

```
1  function plan_phase1()
2    for each robot
3      start = currentNode(robot)
4      if isLeafNode(start)
5        continue
6      end if
7      leaf = freeLeafNode()
8      path = astarPath(start, leaf)
9      obstacle = findObstacleRobot(path)
10     if (obstacle not found)
11       addPath(path, robot)
12     else
13       start = currentNode(obstacle)
14       path = astarPlan(start, leaf)
15       addPath(path, obstacle)
16     end if
17   end for
18 end function
```

Fig. 4. Pseudocode for Phase 1.

freeAstarPath(*start, end*): returns the shortest connected sequence of nodes between nodes *start* and *end*, avoiding any already occupied nodes.

findObstacleRobot(*path*): searches for an occupied node in the *path* sequence, in reverse order from the end to start. A reference to the *first* robot found occupying a node (if any) is returned.

addPath(*path, robot*): adds the sequence of nodes in *path* as a new sequence for *robot* in the plan, and updates the current position of *robot* to the last node in *path*.

planRobotToNode(*robot, goal*): uses *freeAstarPath* to find the shortest obstacle-free path from the robot's current position to the goal, and adds this new trajectory segment using *addPath*.

subTreeContains(*root, node*): returns true if *node* is in the subtree of *root* within the spanning tree.

getBlockedRobot(*node*): searches for robots currently within the subtree of *node*, whose goal is outside of the subtree of *node*.

sortRobotsByDepthOfGoal(): orders the robots according to the depth of their goal nodes, from deepest to shallowest, in the spanning tree. This order is applied in the *for each robot* ... loop of Fig. 4.

A. Phase 1: Reaching Leaf Nodes

In Phase 1, we develop a plan that will move all robots to leaf nodes of the spanning tree. This is accomplished by repeatedly selecting a candidate robot R_i that is not currently on a leaf node (lines 2–6 of the pseudocode shown in Fig. 4), and selecting an unoccupied leaf node L_i (line 7). This is guaranteed to succeed, since there are L leaf nodes and $r < L$ robots to occupy them. A heuristic may be used to select a leaf node close to the robot or its goal. In the example in Fig. 1(a), node D may be selected as the leaf node for robot R_1.

An A* search is then used to find a path (sequence of nodes) P_i, from the initial position of robot R_i to the target leaf node L_i, *ignoring all other robots in the system* (line 8). The path P_i is then examined for robots occupying any nodes of the path (line 9). If the path is clear, the path moving R_i to the leaf node is added to the plan (line 11), and the iteration is complete.

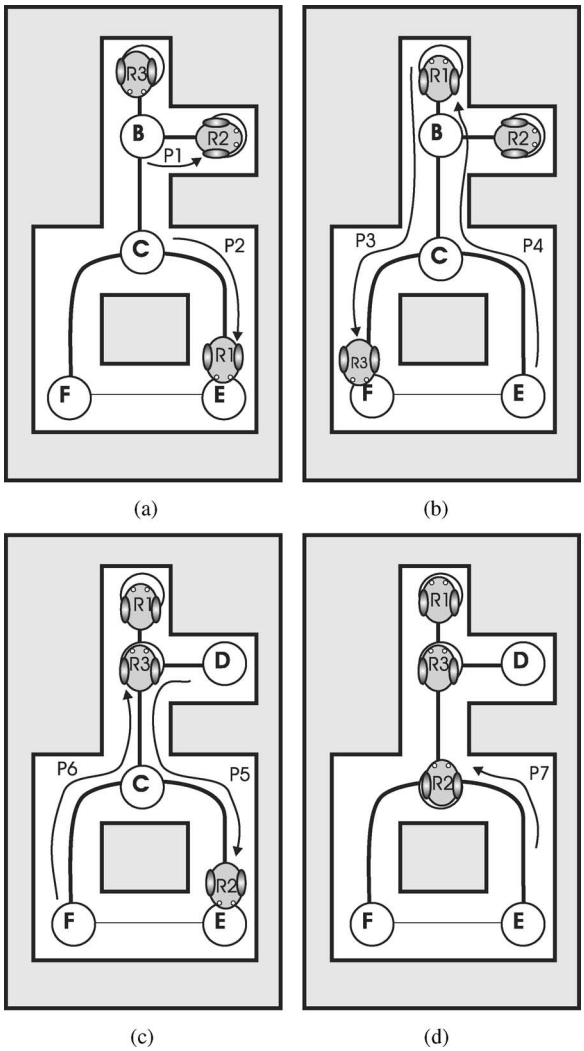


Fig. 5. Multiphase solution to the planning problem of Fig. 1(a). Refer to text for details of each step. (a) Phase 1. (b) Phase 2-a. (c) Phase 2-b. (d) Phase 3.

Otherwise, other robots are obstacles along the path—consider them R_j, $R_k, \ldots, R_n$, in that order. In that case, R_n is the final robot on the path, and has a collision-free path to the leaf node L_i. In this case, we plan for R_n to move to L_i instead, using the obstacle-free subpath of P_i that connects R_n to L_i (lines 13–15). In either case, one robot is moved to a leaf node at every iteration, as required.

In Fig. 5(a), since robot R_2 is an obstacle between the selected robot R_1 and leaf node D, a path P_1 moving R_2 from node B to D is added instead. Continuing the process, R_1 remains to be moved to a leaf node, and either node E or F may be selected, indicated by path P_2.

B. Phase 2: Sorting Robots by Depth of Goals

In Phase 2, we move all robots into positions where they can reach their goals without creating an obstruction for another robot. The need for this arrangement step can be seen in Fig. 5(a): robots R_2 and R_3 have goals on the interior nodes C and B, respectively, and if either moves directly to its goal, it will create

```
19 function plan_phase2()
20   sortRobotsByDepthOfGoal()
21   for each robot
22     start = currentNode(robot)
23     goal  = goalNode(robot)
24     if subTreeContains(goal, start)
25       continue
26     end if
27     blockedRobot = getBlockedRobot(goal)
28     if (blockedRobot found)
29       blockedNode = currentNode(blockedRobot)
30       leaf = freeLeafNotInSubtree(goal)
31       if (leaf found)
32         planRobotToNode(blockedRobot, leaf)
33         planRobotToNode(robot, blockedNode)
34       else
35         leaf = freeLeafInSubtree(goal)
36         planRobotToNode(robot, leaf)
37         planRobotToNode(blockedRobot, goal)
38         continue
39       end if
40     else
41       leaf = freeLeafInSubtree(goal)
42       planRobotToNode(robot, leaf)
43     end if
44   end for
45 end function
```

Fig. 6. Pseudocode for Phase 2.

an obstacle for the other. For a general algorithm to resolve this potential deadlock, we consider the problem in terms of robot positions relative to their goals within the spanning tree structure.

Let T_{G_i} be a subtree of the spanning tree with root at the goal node G_i of robot R_i. A deadlock condition occurs only if:

1) when G_i is occupied, another robot R_j is *inside* the subtree of T_{G_i} and is blocked from reaching its goal *outside* the subtree; or
2) when G_i is occupied, another robot R_j is *outside* the subtree of T_{G_i}, and is blocked from reaching its goal *inside* the subtree.

We can prevent these conditions by:

1) moving robots to nodes within the subtree of their goal nodes;
2) ordering the depth of the robots within the subtree based on the depth of their goals.

To accomplish this task, we process robots in the order of the *depth* of their goals, that is, the distance from the goal node to the root of the spanning tree (refer to Fig. 2 and lines 20 and 21 of the pseudocode in Fig. 6). For each robot R_i, we determine whether it is already in T_{G_i}, in which case the requirements are already satisfied (lines 24 and 25). If not, we test whether filling the goal G_i will create an obstacle for any robots in the subtree T_{G_i}, and if so, select the deepest positioned robot R_j (line 27). The blocked robot R_j can be moved out of the subtree if an unoccupied leaf is available outside of the subtree (lines 30–33). Otherwise, the free leaf must be within the subtree; the depth ordering condition can be achieved by moving R_i to the available leaf within subtree T_{G_i}, and moving R_j to the original goal node G_i (lines 34–38). This phase achieves the two conditions required earlier to avoid deadlock conditions when filling interior node goals.

```
46 function plan_phase3()
47   reverseSortRobotsByDepthOfGoal()
48   for each robot
49     goal = goalNode(robot)
50     if (robot is not at goal)
51       planRobotToNode(robot, goal);
52     end if
53   end for
54 end function
```

Fig. 7. Pseudocode for Phase 3.

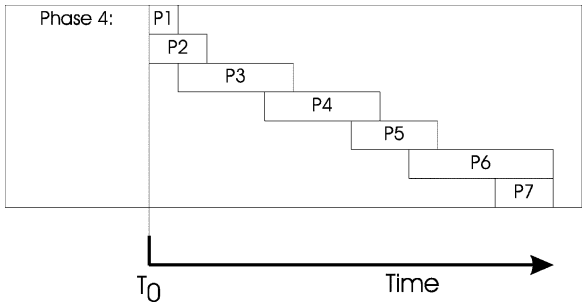


Fig. 8. Individual path segments are overlapped in time whenever possible while avoiding collisions.

The total path length can be reduced by only partially completing the swap in some cases.

1) If the temporary unoccupied leaf used for swapping is not in T_{G_i}, robot R_j may remain at that leaf rather than completing the swap to the previous position of R_i.
2) If R_j is the only robot that would be blocked into the subtree, robot R_i can fill its goal node immediately after robot R_j has been moved.

In the example, R_1 has the deepest goal node A, so is processed first. The subtree of the goal consists of only the node A, and contains the robot R_3, which must be moved to avoid the deadlock condition (line 27). Robot R_3 is therefore moved to the unoccupied leaf node F (line 32), before moving R_1 to its goal node A (line 33), shown by paths $P3$ and $P4$ in Fig. 5(b).

The goals of robots R_2 and R_3 are interior nodes C and B with C being the root of the spanning tree T^*. Robot R_3 has the deeper goal node B, so is processed first. Its goal node B is the root of the subtree containing nodes A and D, as shown in Fig. 2, so we must check for robots that would be blocked into the subtree (line 27). Referring to Fig. 5(b), R_2 at node D is such a robot. We therefore move R_2 to an unoccupied leaf node E (line 32), then plan robot R_3 to its goal node (line 33), indicated by paths P_5 and P_6 in Fig. 5(c). This leaves R_2 and R_3 in subtrees of their goal nodes, and in the same depth order as their goals, as required.

C. Phase 3: Filling Remaining Goals

In Phase 3, we move any robots to the remaining unfilled goals (Fig. 7). If we plan for robots with goals closest to the top of the tree first (line 47), an obstacle-free path for each robot is guaranteed by the arrangement determined in Phase 2, where the robots are sorted in order of the depth of their goals. For the example scenario, this requires planning robot R_2 to its goal at node C (line 51), resulting in the desired goal configuration shown in Fig. 5(d).

D. Phase 4: Building a Concurrent Plan

The plan determined in Phases 1–3 consists of a sequence of segments or paths P_i, in which only one robot moves at any time, as shown in Fig. 3 for the example problem. The sequence of paths guarantees that all robots reach their goal positions without collisions with other robots. However, the sequence of paths is generally very suboptimal in terms of time and total distance required to reach the goal positions, compared to a decoupled planning solution (if one is possible). Since travel time and distance are often significant evaluation criteria in practical applications, a number of methods may be applied to generate a more optimal solution from the sequence of segments. This stage introduces a tradeoff between solution optimality and computational complexity; the ideal method will depend on the scale of the application (number of robots and size of the map), the computational resources available, the requirements for real-time performance, and the relative importance of optimality in the trajectory solution.

Because the algorithm first moves robots to leaf nodes of the spanning tree (a process that is required to guarantee completeness, but is often unnecessary in the final solution), a significant reduction in total distance traveled can typically be gained by finding and removing any redundant motion. This can be found for each robot by checking all cases where the robot returns to a node it previously visited. If the node was not occupied in the intervening time, the robot may simply remain at that node for the duration.

The result of this first optimization is that there may be steps of the trajectory where no robots are moving; these can be simply removed to reduce the total trajectory execution time.

1) Concurrency by Overlapping Segments: An additional step is then to allow multiple robots to move concurrently by overlapping the individual segments in time as much as possible without introducing any collisions, as shown in Fig. 8. Each successive segment of the original plan is added to a concurrent plan by first considering it appended to the end of the plan. The start position of the segment is then moved earlier in time until the motion in the new segment would create a collision between robots in the concurrent plan. The motion of the robot in the new segment is then incorporated into the concurrent plan. This approach was used in generating the simulation results of Section IV, involving up to 40 robots operating in a map of several hundred nodes, with subsecond computation times.

2) Concurrency by a Space–Time Search: An alternative approach in generating a concurrent plan is to generate a concurrent plan for all robots using a sequential A* search in time and space, based on the method described for general multibody motion planning by Erdmann and Lozano-Perez [7].

1) For each segment of the plan, considerer the initial and final positions of the moving robot in each segment.

2) Perform an A* search, in a space–time map. This map is based on the topological-node-based map used in Phases 1–3, but extended in the time dimension with resolution corresponding to the movement of a robot between two adjacent nodes. The initial and final states for the A* search are the initial and final states of the robots in the trajectory segment.
3) Add the A* solution trajectory for the moving robot to the space–time map as an obstacle to be avoided in future searches.

Note that each A* search is guaranteed to find a solution, due to the conditions and ordering of the sequences established in Phases 1–3. In the worst case, for each segment, the initial state will correspond to the final state of the space–time map generated so far, and the A* search will append the same motion as found in the original trajectory segment. Typically, however, the space–time search will find a solution where the motion of one moving robot can be at least partially concurrent with the motion of previously added segments.

This approach of a full space–time search for each trajectory segment is very effective, as the shortest possible paths are found for each required robot motion, and the maximum concurrency of motion is obtained. However, this comes at a substantial computational cost, since the space–time map adds an additional dimension to the A* search space, and the length of the time dimension grows with the number and length of individual segments of the original plan. The method was found to be practical for up to 20 robots in simulation, and was used in the real-world implementation described in Section V.

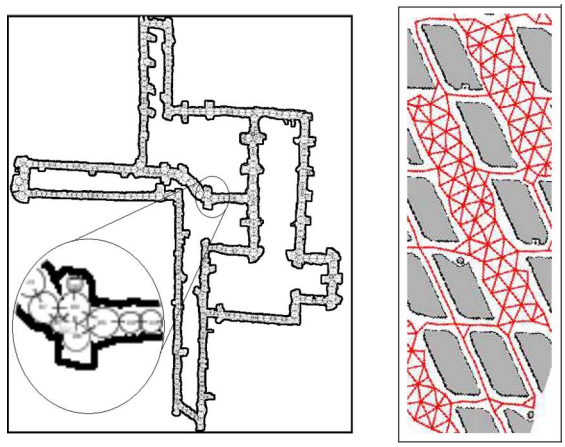
E. Complexity Analysis

The plan completed at the end of Phase 3 will move all robots to their respective goals, as required for a *complete* planner. In each of the three phases, we iterate once over the set of r robots, and require at most three (in the case of swapping) A* plans for each. Each A* search has a fixed complexity C that depends on the size of the graph and the heuristic used, but remains independent of the number of robots in the environment. The total computational complexity of the first three phases is therefore $O(r \cdot C)$ for r robots.

As discussed earlier, the complexity of Phase 4 depends on the method used and the degree of optimization required. In the method of overlapping segments, for example, as each segment overlaps the concurrent plan by one additional step in time, a "collision check" is required for the moving robot at each state in the segment. The worst case complexity of the operation, given a trajectory of s states, is $O(s^2)$.

F. Hybrid Planning

The multiphase planner is fast and complete; it will quickly generate a solution to the planning problem for a large number of robots in a complex graph. However, the resulting plans are typically suboptimal, in terms of path length for each robot.



(a) (b)

Fig. 9. Simulation environment maps. (a) Tunnel map. (b) Open map.

A decoupled planning approach, such as that proposed by Bennewitz [4], can use priority scheduling to consider many different possible plans. Unfortunately, the generation of many plans using different sequences of priorities is CPU intensive, and may fail to find a solution for complex planning problems. When successful, the resulting plans from the decoupled approach are typically shorter than those found by the multiphase planner.

To take advantage of the properties of each approach, a hybrid planner was implemented and evaluated. One valid plan is first quickly generated using the multiphase planner. The decoupled planner is then invoked in an attempt to find a shorter path solution. The decoupled planner may then be terminated at any time, and the most optimal plan selected.

IV. Simulation Results

The four-phase planner described earlier was implemented and evaluated in Monte Carlo simulations in the underground ("tunnel") mine map shown in Fig. 9(a), using between 3 and 40 robots. The planner was also evaluated on a map with more open space, shown in Fig. 9(b), using between 3 and 150 robots.

For each map, a topological representation was generated from an occupancy grid by finding adjacent circular regions of open space (nodes) and connecting all adjacent nodes by edges. The spanning tree selected for the tunnel map contains 43 leaf nodes, allowing for motion planning of up to 42 robots in the environment. Random initial and goal positions are selected for each robot. For the environment with open spaces, a mesh-like topological structure results, allowing robots to pass each other in the open areas. The resulting spanning tree has 142 leaves, allowing for planning of up to 141 robots.

As expected from the analysis earlier, the multiphase planner finds a collision-free plan for every configuration in both maps.

For comparison, a *decoupled planner* using a sequential A* planning approach for each robot was also implemented, which randomly selects a priority sequence of robots. This sequential planner finds the shortest collision-free path for each robot through the space–time map, avoiding obstacles including the

trajectories of all previously planned robots. The results of such a planner are dependent on the priority sequence used, so up to 100 randomly selected priority sequences were applied for each case in an attempt to find a sequence for which a plan could be found. Finally, the hybrid planner approach described in Section III-F was used to evaluate the benefit of using a combination of multiphase and decoupled planning.

The plots in the following sections show the results of applying the algorithms to the same randomly-generated problems in the two different environments.

A. Planning Success Rate

The first measure of the algorithm performance is the success rate of finding a feasible solution. As expected for a complete algorithm, the success rate of the multiphase planner is 100% for up to 42 robots given a spanning tree in the tunnel map with 43 leaves. However, the sequential planner failed to find solutions for some randomly generated problems with 14 or more robots, and failed to find solutions for all problems with 25 or more robots.

In the open-space map, the spanning tree with 142 leaves guarantees a solution for up to 141 robots using the multiphase planner. The decoupled planner began to fail for some problems with 25 robots, and failed to find a solution for any problems with 75 or more robots.

The success rate of the sequential planner will increase if more randomly selected priority sequences are tried; however, the planning cost also increases with each additional priority sequence. One hundred different sequences was a practical maximum value considering the time required for each attempted plan.

B. Average Robot Path Length

The average distance required for each robot to travel to reach its goal is plotted in Fig. 10. The results indicate that in the tunnel map, the multiphase planner typically generates longer paths for each robot, particularly as the number of robots increases. This is not unexpected, since the planner first directs robots to positions other than their goals in order to create an obstacle-free path for the final phases of the process.

In the open map, the average path lengths are very similar. This is due to the increased density of leaf nodes in the map; when the multiphase planner moves robots to leaf nodes, the average additional distance is much less than for the tunnel map.

When the sequential planner begins to fail for some of the randomly generated problems (>14 robots in the tunnel map and >25 robots in the open map), the average path length is computed only for those scenarios where a solution was found.

C. Average Total Execution Time

The average execution time (the number of time steps required for all robots to execute their plans) is plotted in Fig. 11. The plans generated by the decoupled planner can typically be executed in less time than the multiphase planner solutions.

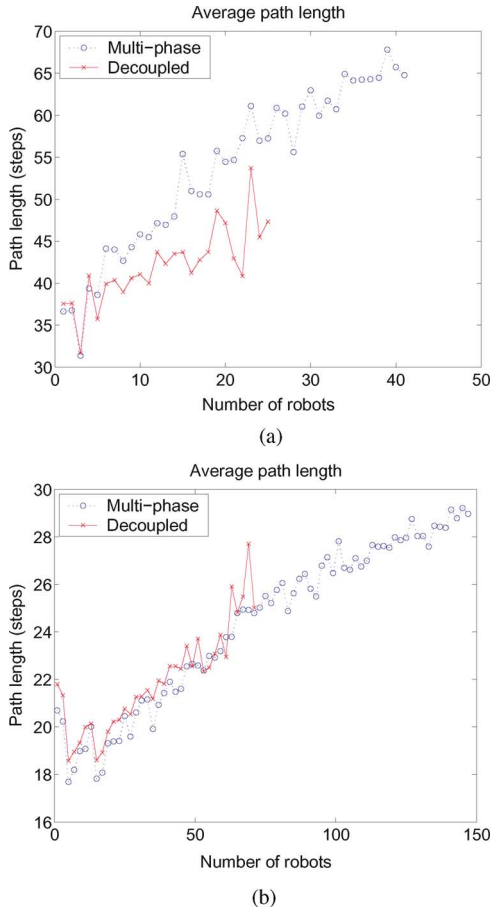


Fig. 10. Average robot path length generated by each planner. (a) Tunnel map. (b) Open map.

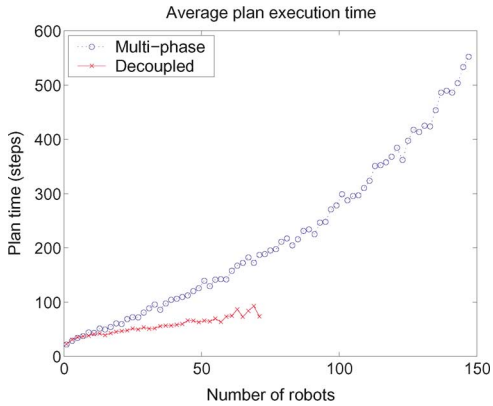


Fig. 11. Average execution time for paths generated by each planner in the open map.

This is due to the serialized nature of the multiphase planner path generation, where the plan is constructed of a sequence of individual robot movements. The execution time is reduced in Phase 4 by executing multiple segments concurrently; however, improving the concurrency involves greater computational complexity.

In contrast, the decoupled planner attempts to immediately move all robots toward their goals from the first time step. This

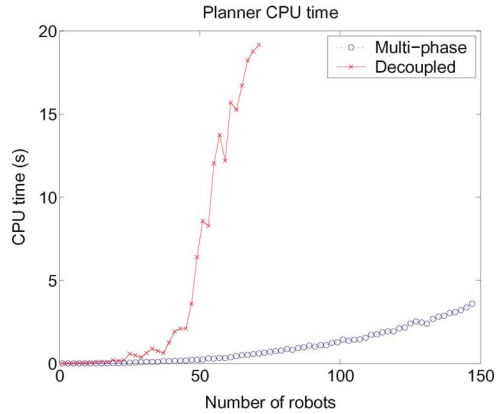


Fig. 12. Average CPU time used by each planner in the open map.

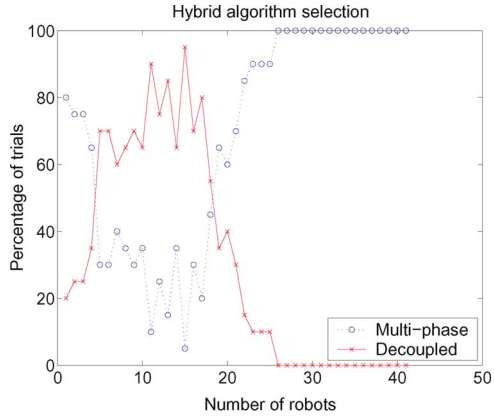


Fig. 13. Hybrid planner selection in the tunnel map.

results in greater concurrency and a shorter execution time. However, this gain comes at a cost of complexity and loss of completeness. In the open-space map, the decoupled planner failed to find solutions for some random scenarios of 25 robots. In the tunnel environment, it failed to find solutions for *any* trials with 25 or more robots.

D. Search Cost

The search cost is a measure of the complexity of the planning algorithm, or the time required to complete the search for a feasible solution. Fig. 12 shows the CPU time required by each algorithm; the processing time has been normalized by the number of robots in the plan, and shows the exponential growth in complexity of the decoupled planning method. The values indicate the time required to find a feasible solution given the graph representation, and not the (one-time) cost of generating the map.

These results demonstrate that while a decoupled approach can find shorter paths for simpler planning problems, the multiphase planner involves much less computational cost. The cost of the sequential planner grows exponentially, since it requires many attempts with different random priority sequences to find a solution. The cost of the multiphase planning algorithm, however, increases close to linearly with the increase in number of robots. For 100 robots in the open-space map, feasible plans were computed by the multiphase planner in less than 1.5 s using a 1.5 GHz Pentium M processor.

E. Hybrid Planner

The graph of the algorithm selection in the hybrid scheme, shown in Fig. 13, indicates the algorithm behavior as the number of robots in the system increases. The plots show the percentage of time the results of each planner are selected, indicating how often the multiphase planner generates a more optimal result (a shorter total travel distance) than the decoupled planner. For very small numbers of robots, the multiphase planner results are often better than the decoupled planner results. The randomly selected order used by the decoupled planner is typically suboptimal, required longer paths to be generated for some robots. As the number of robots increases, the decoupled planner can often find shorter path solutions. However, beyond a threshold, the decoupled planner fails to find any solutions, and the multiphase planner results are required.

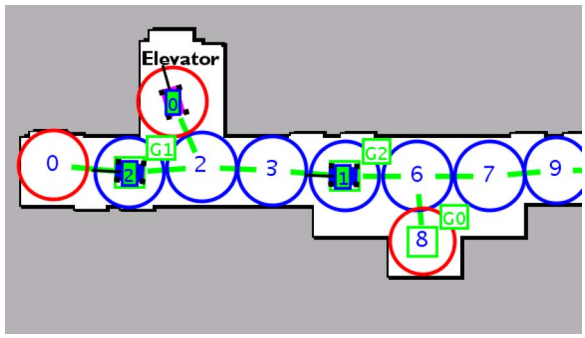


Fig. 14. Graph representation of hallway environment.

V. Real-World Implementation

To validate the algorithm in a real-world application, the planner was implemented on a system of three WBR-914 PCBot robots, provided by FrontLine Robotics, shown in Fig. 15(b).

The task selected for the robots was to cooperatively navigate in a long, narrow corridor, repeatedly traveling between an elevator and randomly selected locations, as would be required for autonomous delivery robots. We assume that robots may not pass each other within the hallway due to size and safety constraints. As a practical application, note that the trajectory planning problem in this scenario is similar to that for several autonomous mining vehicles operating in the same area of an underground mine.

The floor plan and graph representations of the environment are shown in Fig. 14. By selecting a node near the center of the map as the root of the spanning tree (node 3 for example), the tree has $L = 4$ leaves, and the planner is guaranteed to find a solution for up to $r = 3$ robots.

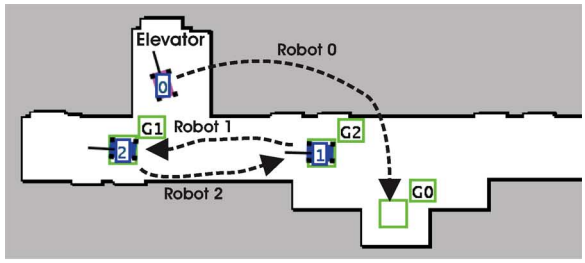
A. Planning and Control Implementation

Each robot runs the Player [9] robot server, and is equipped with a scanning laser rangefinder for localization. A *trajectory*

tracking function was added to Player as a plug-in module. This module drives the robot through a trajectory as generated by the multiphase planner; that is, a list of waypoints and the desired arrival and departure time at each waypoint.

A centralized host PC coordinates the system, communicating with the Player server on each robot via wireless ethernet. The host PC monitors the positions of the robots, and assigns a new goal position for each robot as the goals are achieved. Whenever a new goal is assigned, a new multirobot plan is generated for all of the robots, and the individual trajectories are transmitted to the trajectory followers of each robot.

The addition of a time-dependent trajectory following module to the Player robot server allows for scalable simultaneous motion control of many robots from a centralized server. A new multirobot plan is generated whenever the goals change, and the communication to each robot involves only a list of waypoints with arrival and departure times. In the real-world implementation, robots typically diverge from the planned trajectories to some degree, due to reactive avoidance of obstacles in the environment, errors in localization, wheel slip, etc. The individual robots correct for small deviations in trajectory following (due to wheel slip, for example) by accelerating or decelerating between waypoints. The server also detects deviations from the plan by monitoring the localization and status information transmitted from each robot. If a robot diverges significantly from the specified trajectory, (pausing in a corridor to avoid colliding with a person walking past, for example), the plan is invalidated. A new plan is generated from the current positions of all robots, all trajectories are updated, and the robots resume moving toward their goals.



(a)

(b)

Fig. 15. Graphical and real-world views of the sample problem. (a) User interface view. (b) WBR-914 robots.

B. Sample Problem and Results

Fig. 15(a) shows an example problem requiring a coordinated solution to demonstrate the implementation. The required transitions are indicated by dotted arrows, moving robots to the goal locations indicated by squares. Robots 1 and 2 are required to swap positions, and robot 0 is required to move from the top to the bottom of the map, crossing the direct paths of robots 1 and 2. Fig. 15(b) shows the robots in the hallway environment.

For this problem, a typical decoupled planning approach will fail; if any robot takes the most direct path to its goal, it creates an unavoidable obstacle for another robot. However, the multiphase algorithm finds a solution by first moving robot 1 to the leaf node 8, then robot 2 to node 7. From this arrangement, all robots reach their goal nodes: first robot 1 to G1; then robot 0 to G0; and finally, robot 2 to G2.

The planning algorithm and the trajectory follower have been validated in the corridor environment by randomly allocating new goals as each robot completes its current trajectory. The system runs continuously, limited only by the battery life of the robots.

VI. Discussion

In this paper, maps of tunnels and corridors were considered specifically, since they have a relatively simple topological representation and present a challenging environment for the coordination of a large number of robots. For more general cases, including arbitrary obstacles and nonholonomic motion constraints, the generation of a suitable roadmap or graph representation can be a challenging problem in itself. However, once a suitable graph is created, the multiphase algorithm can be applied directly.

Considering the performance comparison between the sequential planner and the multiphase planner, it may be advantageous to consider a hybrid approach, taking advantage of the features of both algorithms. By first generating a plan using the multiphase planner, a feasible solution can be generated very efficiently. To search for a more optimal plan, a sequential planner can then be applied to the same problem, and permitted to run within the time bounds of the application.

The algorithm as presented here assumes a centralized planning architecture, where all information and resources are available at a single processing point. Incorporating this centralized planner into a distributed planning architecture, as proposed in [6], will be another subject of future work.

VII. Conclusion

This paper presented a multirobot planning algorithm that is based on a topological graph and spanning tree representation. By breaking the planning process into several phases, it was shown that the algorithm guarantees a solution to the planning problem, and is scalable with linear increase in complexity for up to $r < L$ robots given a spanning tree with L leaves. For practical systems, the number of leaves L is a reasonable upper bound on the number of robots that would be used in a shared space. In comparison to a decoupled sequential planning algorithm, the multiphase planner typically produces longer paths, but at a much lower computational cost when planning for many robots. Finally, an implementation on physical robots demonstrated the practicality of the multiphase planner in real-world applications. In future work, the multiphase planner will be incorporated into a higher level task allocation system for robots operating in corridor and tunnel environments.

Acknowledgment

We would like to thank S. Thrun of the Stanford Artificial Intelligence Laboratory for the use of the robot-generated maps of underground mines. We would like to thank Frontline Robotics for the contribution of the WBR-914 robots and technical support.

References

[1] R. Alami, F. Robert, F. Ingrand, and S. Suzuki, "Multi-robot cooperation through incremental plan-merging," in *Proc. ICRA*, 1995, pp. 2573–2579.

[2] K Azarm and G Schmidt, "Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1997, pp. 3526–3533.

[3] Jérîme Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *Int. J. Robot. Res.*, vol. 10, no. 6, pp. 628–649, 1991.

[4] M. Bennewitz, W. Burgard, and S. Thrun, "Optimizing schedules for prioritized path planning of multi-robot systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2001, pp. 271–276.

[5] H. Choset and J. Burdick, "Sensor based motion planning: The hierarchical generalized voronoi graph," in *Proc. Workshop Algorithmic Found. Robot.*, 1996.

[6] C. Clark, S. Rock, and J. C. Latombe, "Motion planning for multi-robot systems using dynamic robot networks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, pp. 4222–4227.

[7] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–521, 1987.

[8] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Auton. Robots*, vol. 13, no. 3, pp. 207–222, 1997.

[9] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proc. ICAR 2003*, Coimbra, Portugal, Jun. 2003, pp. 317–323.

[10] Y. Guo and L. Parker, "A distributed and optimal motion planning approach for multiple mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2002, pp. 2612–2619.

[11] E. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. SSC*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[12] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the 'warehouseman's problem'," *Int. J. Robot. Res.*, vol. 3, no. 4, pp. 76–88, 1984.

[13] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 72–89, 1986.

[14] L. E. Kavraki and J. C. Latombe, "Probabilistic roadmaps for robot path planning," in *Practical Motion Planning in Robotics: Current Approaches and Future Directions*. New York: Wiley, 1998, pp. 33–53.

[15] J. C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.

[16] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ Press, 2006.

[17] V. J. Lumelsky and K. R. Harinarayan, "Decentralized motion planning for multiple mobile robots: The cocktail party model," *Auton. Robots*, vol. 4, no. 1, pp. 121–135, 1997.

[18] P. A. O'Donnell and T. Lozano-Pérez, "Deadlock-free and collision-free coordination of two robot manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1989, pp. 484–489.

[19] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *Int. J. Robot. Res.*, vol. 24, no. 4, pp. 295–310, 2005.

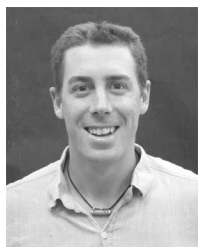
[20] M. R. K. Ryan, "Graph decomposition for efficient multi-robot path planning," in *Proc. IJCAI*, 2007, pp. 2003–2008.

[21] T. Siméon, S. Leroy, and J.-P. Laumond, "Path coordination for multiple mobile robots: A resolution complete algorithm," *IEEE Trans. Robot. Autom.*, vol. 18, no. 1, pp. 42–49, Feb. 2002.

[22] P. Svestka and M. Overmars, "Coordinated path planning for multiple robots," *Robot. Auton. Syst.*, vol. 23, pp. 125–152, 1998.

Mike Peasgood (A'05) received the M.A.Sc. degree in systems design engineering and the Ph.D. degree in mechanical engineering from the University of Waterloo, Waterloo, ON, Canada, in 2004 and 2007, respectively.

His current research interests include multirobot systems, aerial robotics, motion planning, and modeling of human walking gait. He is currently an Entrepreneur and cofounder of Aeryon Labs Inc., Waterloo.



Christopher Michael Clark received the Ph.D. degree in aeronautics and astronautics - computer science minor from Stanford University, Stanford, CA, in 2004.

From 2004 to 2007, he was an Assistant Professor at the University of Waterloo, Waterloo, ON, Canada. He is currently an Assistant Professor in the Department of Computer Science, California Polytechnic State University, San Luis Obispo. His current research interests include motion planning, multirobot systems, intelligent transportation systems, modular and reconfigurable robots, and underwater robot systems.

John McPhee received the Ph.D. degree in mechanical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1990.

He was with the Université de Liège, Belgium, and Queen's University, Canada. During 1992, he was a faculty member at the University of Waterloo, where he is currently a Professor in systems design engineering, and also the Executive Director of the Waterloo Centre for Automotive Research. He is the author or coauthor of 50 papers published in the top journals, consulted to many industries in Canada and the United States, and appeared as an expert witness in the Federal Court of Canada. He has supervised 35 graduate students and postdoctoral fellows. He is an Associate Editor for six international journals. His current research interests include multibody system dynamics, with principal application to the analysis and design of vehicles, mechatronic devices, and biomechanical systems.

Prof. McPhee was elected as a Fellow of the American Society of Mechanical Engineers during 2005. He is the recipient of the Premier's Research Excellence Award in 2000 and the I.W. Smith Award from the Canadian Society of Mechanical Engineers in 2001. He is a registered Professional Engineer in the Province of Ontario.