2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Decentralized and Complete Multi-Robot Motion Planning in Confined Spaces

Adam Wiktor*, Dexter Scobee*, Sean Messenger†, and Christopher Clark†
*Princeton University, Princeton, NJ 08544
email: awiktor@alumni.princeton.edu, dscobee@alumni.princeton.edu
†Engineering Dept., Harvey Mudd College, Claremont, CA 91711
email: smessenger@hmc.edu, clark@hmc.edu

***Abstract*—This paper presents the Push-Swap-Wait (PSW) algorithm, a scalable, decentralized and complete approach for multi-robot motion planning in confined spaces. The algorithm builds upon a "push and swap" paradigm that has been used effectively in centralized navigation. This push and swap approach was expanded to apply to decentralized planning by adding a waiting mode to handle situations in which communication between robots is lost. The completeness of the PSW algorithm can be guaranteed in cases where the environment can be modeled as a tree T for which the number of leaf nodes is greater than the number of robots navigating through it. The algorithm has a time complexity that is linear with the number of robots currently within communication, indicating that this algorithm is well suited for scaling to large systems of robots. To validate the PSW algorithm it was implemented successfully in multi-robot simulations and on hardware with four Dr. Robot Jaguar Lite Robots.**

Fig. 1: DrRobot Jaguar Lite robots at Harvey Mudd College.

I. Introduction

Multi-robot systems have demonstrated the potential to increase performance over single robot systems in tasks requiring decreased mission times, spatio-temporal sampling, robustness to mission failure, and force multiplication [2], [5]. This paper addresses the problem of coordinating the motion of multiple robots attempting to reach their independent goal destinations in single lane tunnel environments.

In general, Multi Robot Motion Planning (MRMP) can be accomplished with a *centralized* control architecture, in which one robot or agent dictates the motion of all robots, or using a *decentralized* architecture in which each robot calculates its own motions. Additionally, the algorithms implemented within these architectures can either be *coupled*, in which the algorithm searches the composite configuration space of all robots to construct paths for all robots, or *decoupled*, in which the algorithms search each individual robot's configuration space for individual paths that are later coordinated. Implementation of both coupled and decoupled planning can be centralized, but decentralized architectures are more amenable to decoupled planning.

Several centralized approaches for robot navigation in confined spaces already exist [14], [10], [11], [3]. Some of these algorithms have the desirable property of being complete - that is, they guarantee that a solution will be found if it exists. Such algorithms can also be classified as either optimal or non-optimal. Optimal algorithms, such as search algorithms like A*, are capable of computing the shortest set of paths that solve the problem (if a solution exists), but the computation is NP-complete [9], [13].

Decentralized approaches offer several advantages over centralized algorithms [4] including the ability to scale to large systems by distributing computation. Most importantly, many real systems are decentralized due to limitations in communication. In these cases, global and complete information is not available to all robots, making it difficult to guarantee that a solution is always found. For this reason, decentralized algorithms to date are not complete and suffer from the possibility of deadlocks [9], [17]. Others make use of decentralized computation, but rely on globally broadcast information for completeness guarantees [6], [1].

Recently, several centralized approaches have been presented that aim for increased optimality while striving for completeness. The MAPP algorithm [19] demonstrated completeness when paths satisfy three properties termed blank availability, alternate connectivity, and target isolation. In [18], MAPP was extended to increase the range of problems for which completeness was guaranteed. In [16], the state space searched by A* is optimized using Operator Decomposition (OD) and Independence Detection (ID). By adding a check for maximum group size, they produced an anytime algorithm which finds an initial solution and then refines it for optimality.

978-1-4799-6934-0/14/$31.00 ©2014 IEEE

Several other recent approches have leveraged the linear-time feasibility test for multi-agent path finding [12]. In [7], the Tree-Based Agent Swapping Strategy (TASS) was presented which is complete within a defined domain of solvable trees. Most related to this work is the Push and Swap (PaS) algorithm presented in [11] which is complete for n-2 agents operating in a graph of size n. The algorithm uses two primitives, Push - agents move towards their goals by forcing other agents to clear a path - and Swap - two agents swap positions without altering configurations of other agents, after which all agents must return to their positions. In [8], the authors' work was improved for efficient run time and path quality.

The work proposed here builds from the PaS algorithm, and combines it with the priority approach to completeness demonstrated in [14], to develop a new MRMP algorithm for robots navigating in connected graph-based environments called Push-Swap-Wait (PSW). Unlike all the previous approaches mentioned above this approach is not only *complete*, but is *decentralized* to allow implementation in scenarios that are practical, run in real-time, and have limited inter-robot communication.

II. Problem Formulation

Consider a set of nodes N and a set of bi-directional connecting edges between them E which form a graph $G(N, E)$. Occupying G is a set of autonomous robotic agents R. At each timestep t, there is an assignment A that maps each robot $r \in R$ to its location in G, such that $A(r, t) \in N$. All agents have knowledge of $G(N, E)$ and each has a unique assigned goal $g(r) \in N$ such that $g(r_i) \neq g(r_j)$ if $i \neq j$. Each node can contain only one robot at a time, meaning that $\forall r_i, r_j \in R$, if $i \neq j$, then $A(r_i, t) \neq A(r_j, t)$. Between timesteps, robots may move from node n_o to node n_p provided that $\exists\, e \in E : e = (n_o, n_p)$.

However, two robots cannot traverse the same edge between the same timesteps, so $\forall r_i, r_j \in R$, if $A(r_i, t + 1) = A(r_j, t)$, then $A(r_j, t + 1) \neq A(r_i, t)$. The change from one assignment $A(R, t)$ to another $A(R, t + 1)$ is determined by the individual position change made by each robot, $\pi(r, t)$. At each timestep t, every robot r computes which move $\pi(r, t)$ to make, which may take the robot along an edge e to a new node n (provided the conditions given above hold) or keep the robot at its current node. The goal is to rearrange the robots from an initial assignment $A(R, 0)$ to a final assignment $A(R, t_{final}) = g(R)$.

All robots r_i within ρ nodes of r are considered to be in direct communication with r (it is assumed that the robots' communication range is always greater than the maximum distance spanned by ρ nodes on the graph). Robots are assumed to broadcast information about themselves and any other robots of which they are aware. This information includes the positions, goals, and planned paths of all robots with which the broadcaster is in communication, including itself. In this way, robots that cannot communicate directly may do so indirectly through the formation of an ad-hoc communication network. The set $C(r)$ includes all robots in communication with r, whether direct or indirect.

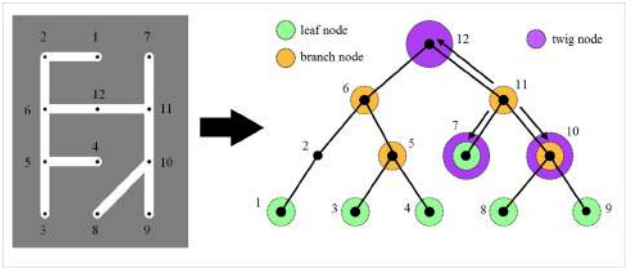


Fig. 2: *Example Tree*. Leaf and branch nodes are highlighted, while node priority is indicated by numbers. Twig nodes of the top right branch node are highlighted by arrows.

For the algorithm presented here, robotic motion is restricted to a spanning tree T of G, such that $T = T(N, \varepsilon)$, where $\varepsilon \subseteq E$. Robots are assumed to be able to plan their next move on a timescale much shorter than the time taken to travel between nodes. Furthermore, it is assumed that no robot motion failures will occur.

With this tree framework in mind, the following definitions and Figure 2 are used to describe the PSW algorithm:

Definition. Leaf Node: A *leaf is defined as a node l such that $\exists!\, n : (l, n) \in \varepsilon$, or in other words, a node connected to only one other node, i.e. a dead-end. The set of nodes L contains the leaf nodes of T, such that $L \subseteq N$.*

Definition. Branch Node: *A branch node b is a node that has 3 or more neighbors. Branch nodes are the intersections of T and are contained in a set $B : B \subset N$.*

Definition. Twig Node: *A node γ is considered to be a twig of branch node b if node γ is adjacent to b such that $\exists e \in \varepsilon : e = (b, \gamma)$.*

Definition. Available Branch Node: *A branch node b is considered to be $available$ if 3 or more of b's twig nodes are free for swapping, i.e. not blocked by other robots.*

Definition. Unavailable Branch Node: *A branch node b is considered to be $unavailable$ if 2 or fewer of b's twig nodes are free for swapping, i.e. all other twig nodes are blocked by other robots.*

Definition. Ancestors: *The set of ancestors of a node $n \in N$ is the set of nodes $P(n) \in N$ such that $P(n) = parent(n) + P(parent(n))$ and is empty for $n = root(T)$.*

Definition. Priority: *The priority of a node n belonging to tree T is denoted by $\Phi(n)$. Priorities are unique integer values, and are assigned according to a post-order traversal of T. A robot's priority is set to be that of it's goal node, i.e. $\Phi(r) = \Phi(g(r))$.*

Definition. Solved: *A robot r is solved at time t if:*

1) $\exists t_1 < t,\ A(r, t_1) = g(r)$
2) $\forall r_L \in R$ *such that* $\Phi(r_L) < \Phi(r)$ *and* $\forall t' : t_1 \leq t' \leq t$ *it holds that* $A(r, t') \notin P(A(r_L, t'))$
3) *and* $\forall r_H \in R$ *such that* $\Phi(r_H) > \Phi(r)$, *robot* r_H *is also solved.*

Note that the term "solved" refers to the global state of the problem. Therefore, an individual robot may consider itself solved based on the other robots in its communication network, but the robot is not truly solved unless the conditions listed above apply globally.

III. PUSH-SWAP-WAIT ALGORITHM

The Push-Swap-Wait (PSW) algorithm presented here draws inspiration from the centralized push-swap algorithm [11]. A third mode, waiting, is added to guarantee completeness for the decentralized problem.

Upon initialization, each robot constructs an identical spanning tree T from G. Every node $n \in N$ will be assigned a unique priority value $\Phi(n)$ based on a postorder traversal of the tree. Figure 2 illustrates a tree and the assignment of priority to nodes on that tree, with lower numbers denoting higher priorities. Each robot $r \in R$ is given a priority equal to the priority of its goal $g(r)$, such that $\Phi(r) = \Phi(g(r))$. The algorithm dictates that robots become solved in order of their priority. *The ordering of robots by priority is central to the guarantee of completeness for cases where* $|R| \leq |L| - 1$.

Algorithm 1: $\pi(r)$=***PSW***(r)

```
1  //Check to see if robot r should wait for a swapping robot to return
2  IF there exists a waiting robot r_i in set C(r)
3      and the robot that r_i is waiting for is not in C(r)
4        return π(r) ← A(r) //Wait
5
6  //Check if r should be pushed, swap, or move normally to its goal
7  ELSE
8        [swapFlag, r̄*, r̲*] ← CheckSwap(r)
9
10       IF [ swapFlag = SWAP or SOLO ] and [ r ∉ [r̄*, r̲*] ]
11             return π(r) ← Push(r, r̄*, r̲*)
12       ELSEIF swapFlag = SWAP
13             return π(r) ← Swap(r, r̄*, r̲*)
14       ELSE
15             IF there exists r_j in C(r) such that Φ(A(r_j)) > Φ(A(r))
16                 and nextNode(A(r), g(r)) belongs to path(r_j)
17                  and swapFlag = SUPPRESS
18                return π(r) ← A(r) //Yield to rj on higher priority b
19             ELSE
20                   return π(r) ← nextNode(A(r), g(r)) //Go to goal
```

At each time step t, each robot $r \in R$ calls the *PSW*() function (Alg. 1) to decide on its next move based on its knowledge of other robots in the local communication network $C(r)$. The $PSW()$ function first checks if r or any robot $r_i \in C(r)$ is waiting for a swapping robot $r^* \notin C(r)$ to complete its swap and return to the communication network. This is necessary to avoid interrupting an ongoing swap between robots that have temporarily left the communication network. The *Push*() function (Alg. 4) determines when a robot begins and ends waiting.

The $PSW()$ function next checks on line 6 if robot r itself should be swapping. The *CheckSwap*() function (Alg. 2) is called, which returns either the two robots that should be swapping, just the highest priority unsolved robot if it does not need to swap, or NULLs if no swaps should occur. Additionally, *CheckSwap*() returns a value for *swapFlag* that can have values of SWAP (two robots $\overline{r}^*$ and $\underline{r}^*$ are swapping), SOLO ($\overline{r}^*$ is moving straight towards its goal and other robots are pushed out of the way), or SUPPRESS (no swaps occur, and no robots get pushed).

If a swap is occurring, and r is not a swapping robot, *Push*() will be called to ensure that r does not interfere with the swap.

Similarly, if $swapFlag =$ SOLO, indicating that $\overline{r}^*$ is driving to its goal, and $r \neq \overline{r}^*$, *Push*() will be called to ensure that r does not block its path. If r *is* one of the swapping robots, then the algorithm will call the *Swap*() function (Alg. 3) to ensure that r carries out the swap.

Beginning on line 14, *PSW*() handles the motion of robots when no swaps are taking place. This is the case when $r = \overline{r}^*$ (driving straight to its goal), when $\overline{r}^* =$ NULL (all robots are solved), or when swapping is being suppressed in $C(r)$ (details of swap suppression are handled in the *CheckSwap*() function). If swaps are being suppressed or if all robots are solved, the algorithm checks if there is a robot $r_j \in C(r)$ on a higher priority branch than r, and r *pauses if the next node between* r *and its goal is on the path of* r_j. The $nextNode(n_1, n_2)$ function, which returns the next node on the shortest path from node n_1 to node n_2 is used here. Since robots always choose the lowest priority branch available when getting pushed (see details of *Push*(), below), this behavior ensures that a robot that got pushed down a higher priority branch moves back up first, preserving the order of solved robots.

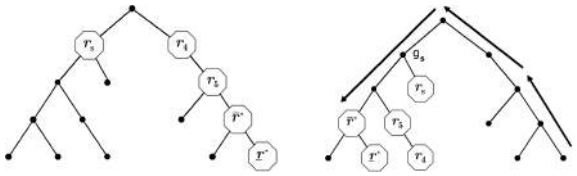
Finally, if all other checks fail, the *PSW*() function reaches the case where $r = \overline{r}^*$, no swap needs to occur, and r can simply maneuver straight for its goal.

The *CheckSwap*() function called by $PSW()$ determines which robots in a communication network should be swapping, if any. The function first finds the highest priority unsolved robot $\overline{r}^*$ in $C(r)$. If $\overline{r}^*$ is already swapping with a robot $\underline{r}^*$, the function returns the robot pair $[\overline{r}^*, \underline{r}^*]$ and $swapFlag =$ SWAP to allow the swap to finish. If no swap was previously taking place, line 9 of $CheckSwap()$ determines if swaps should be suppressed because $\overline{r}^*$ is at a child node of a solved robot's goal. This swap suppression ensures that any solved robots that were disturbed by the previous swap are able to regain communication before a new swap occurs (see Fig. 3). Note that $CheckSwap()$ will also return SUPPRESS if all robots are solved to allow all robots to return to their goals.

$CheckSwap()$ next determines if there exists another robot with which $\overline{r}^*$ needs to swap. Specifically, two robots $\overline{r}^*$ and $\underline{r}^*$ need to swap if any of the following are true:

1) $\underline{r}^*$ is on the path from $\overline{r}^*$ to $g(\overline{r}^*)$ and $\overline{r}^*$ is on the path from $\underline{r}^*$ to $g(\underline{r}^*)$
2) $\underline{r}^*$ and $g(\underline{r}^*)$ are on the path from $\overline{r}^*$ to $g(\overline{r}^*)$
3) $\overline{r}^*$ and $g(\overline{r}^*)$ are on the path from $\underline{r}^*$ to $g(\underline{r}^*)$
4) $\overline{r}^*$ is heading to its goal without swapping and $\underline{r}^*$ is unable to move out of the way

If $CheckSwap()$ finds a robot $\underline{r}^*$ that is adjacent to and needs to swap with $\overline{r}^*$, the pair of robots is returned along with $swapFlag =$ SWAP. Otherwise, $\overline{r}^*$ is returned with $swapFlag =$ SOLO so that it can move straight towards its goal, pushing other robots out of its way.



(a) (b)

Fig. 3: *Swap suppression at child of* $g(r_s)$. In (a), two robots $\overline{r}^*$ and $\underline{r}^*$ initiate a swap that takes them to the other side of the tree, where they push two unsolved robots, r_4 and r_5, and one solved robot r_s. In (b), r_s is waiting for the return of the swappers before moving back to its goal g_s, preventing it from becoming unsolved.

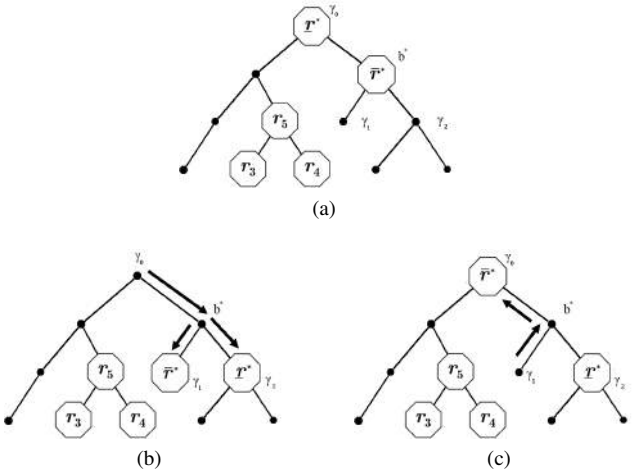
Algorithm 2: $[swapFlag, \overline{r}^*, \underline{r}^*] = CheckSwap(r)$

```
1  r̄* ← robot with the highest priority in C(r) that is not yet solved
2  //If all robots are solved, r̄* ← NULL
3
4  //Complete swaps before starting new swaps
5  if r̄* is swapping
6        r̲* ← swapping partner of r̄*
7        return [SWAP, r̄*, r̲*]
8
9  if [ there exists a solved robot r_s such that g(r_s) is P(A(r̄*)) ] or
10    [ r̄* = NULL ]
11       return [SUPPRESS, NULL, NULL]
12
13 // Check if a robot r̲* exists that is beside r̄* and must swap with r̄*
14 for r_i in C(r)
15       if [r_i and r̄* are adjacent ] and [ r_i and r̄* should swap ]
16             r̲* ← r_i
17             return [SWAP, r̄*, r̲*]
18
19 return [SOLO, r̄*, NULL]
```

If *CheckSwap()* determines that two robot should be swapping, $PSW()$ will call the *Swap*() function to plan the motion of r as it carries out a swap. In the first call to *Swap*(), as well as in subsequent calls whenever a new branch node must be selected, the algorithm chooses a branch node b^* at which the two robots may be able to perform a swap. Node b^* is added to the $visited$ array which stores the list of branch nodes that have already been checked. The function also removes any ancestor nodes of b^* from the $visited$ array to ensure that the swapping robots will check those nodes again on their way back up the tree.

On all other calls to *Swap*(), r will be directed along a path to branch node b^*. If at any point b^* is determined to be unavailable, the next call to *Swap*() will select a new branch node. Otherwise, once the two robots have reached b^*, *Swap*() uses logic to exchange the positions of the two robots, at which point the swap is complete (see Fig. 4).

The $PSW()$ function also calls *Push*() to govern the behavior of robot r in the case where r is neither swapping nor the highest priority robot in $C(r)$ *and* r is not waiting to regain communication with r^*. If r is on the path of either one of the swapping robots or another pushed robot, it sets its destination to the lowest-priority adjacent node available, $\delta_L(A(r))$. If no such node exists, r is stuck, the swapping robots will determine that the branch node they are heading



(a) (b) (c)

Fig. 4: *Process of finishing a swap*. In (a), the robots $\overline{r}^*$ and $\underline{r}^*$ have just arrived at an available branch node b^*. In (b), $\overline{r}^*$ and $\underline{r}^*$ arrive at their respective twigs γ_1 and γ_2 (c) shows $\overline{r}^*$ and $\underline{r}^*$ after the swap with positions reversed.

Algorithm 3: $\pi(r)$ = ***Swap***$(r, \overline{r}^*, \underline{r}^*)$

```
1  if first call to swap for the pair of robots [r̄*, r̲*] or b* is unavailable
2     b* = highest priority available branch node that is not in set visited
3
4     // Add b* to list of visited branches and remove the ancestors of b*
5     visited += b*
6     visited −= P(b*)
7
8  //Set destination to b*
9  π(r) ← nextNode(A(r), b*)
10
11 if r̄* and r̲* have reached b*
12    π(r) ← swap at b*
13
14 return π(r)
```

towards is unavailable, and the *Swap*() algorithm will select a new branch node.

Next, r checks if it must wait in case it loses communication with $\overline{r}^*$. If $\overline{r}^*$ is in direct communication with r and moving away from its goal, r will know to wait if $\overline{r}^*$ is not in $C(r)$ on the next iteration of $PSW()$. Otherwise, r

Algorithm 4: $\pi(r)$ = ***Push***$(r, \overline{r}^*, \underline{r}^*)$

```
1  //Check if r will get pushed by another robot in C(r) that is
2  //swapping or being pushed
3  if [ there exists a robot r_i in C(r) such that A(r) in path(r_i) ] and
4     [ r_i is being pushed or belongs to [r̄*, r̲*] ] and
5     [ there exists a node n available to be pushed to ]
6        return π(r) ← δ_L(A(r))
7
8  //If r̄* is in direct communication with r
9  if r̄* is in c(r)
10       if r̄* is moving towards g(r̄*)
11             r is not waiting
12       else
13             r will wait for r̄*
14
15 return π(r) ← A(r)
```

will not wait and will proceed through the rest of $PSW()$ on the next iteration. In either case, r sets its goal to its current position and remains stationary.

IV. Algorithm Completeness

The completeness of this algorithm requires that the tree T is known by all robots ahead of time and that there are more leaf nodes than robots. Completeness can be proven with several lemmas.

Lemma 1 - For any given tree T and set of robots R such that $|R| \leq |L|-1$, the highest priority pair of swapping robots, $\overline{r}^*, \underline{r}^* \in R$, will find a branch with which to swap.

Proof - Let B be the set of all branch nodes in T. The swapping robots will attempt to exhaustively search all branch nodes in B for one that is available for swapping, and do so in order of their priority $\Phi()$.

Each time the robots visit a branch node b_i and determine it is unavailable, they will depart b_i and move to the next potentially available branch node b_i' such that $\Phi(b_i) > \Phi(b_i')$ as determined by the order in which branches are searched. All other non-swapping robots encountered will wait or be pushed according to Algorithm 1.

The Lemma can now be proved indirectly. Assume that the swapping pair completely searches all branch nodes in T and does not find one available. Define c_T as the sum of the number of extra twigs beyond 3 connected to each branch node. An equation relating c_T to number of leaf and branch nodes can be derived as follows: when adding a node to a tree, it can be connected to an existing node with 1 edge (preserving $|L|$, $|B|$, and c_T), an existing node with 2 edges (increasing $|L|$ and $|B|$ by one while preserving c_T), or an existing node with 3 or more edges (increasing $|L|$ and c_T by one while preserving $|B|$). That is, $\Delta|L| = \Delta c_T + \Delta|B|$. In the most basic trees (containing no branches), $|B| = 0$, $c_T = 0$, and $|L| = 2$. Given this initial condition, this yields $|L| = c_T + |B| + 2$.

Now, for each individual branch node b_i that is found to be unavailable, all but 2 of the twig nodes of b_i must be occupied by non-swapping robots that are now stuck (unable to leave those twig nodes). That is, b_i will have c_i extra twigs (i.e. $c_i + 3$ total twigs), of which at least $c_i + 1$ will be occupied by non-swapping robots. For all branch nodes in a tree to be unavailable, the total number of non-swapping robots r_{ns} must satisfy $r_{ns} \geq (c_1 + 1) + (c_2 + 1) + ... + (c_B + 1) = c_T + |B|$. Since, $|R| = r_{ns} + 2$, it can be determined that $|R| \geq c_T + |B| + 2$.

Substituting $|L|$ for $c_T + |B| + 2$ in this inequality, we see that if the swapping robots find no available branch nodes, then $|R| \geq |L|$. But this contradicts our property of the workspace that $|R| < |L|$. Hence Lemma 1 must hold.

Once the swapping pair has found an available branch node, it can conduct a swap as proven below.

Lemma 2 - For a pair of adjacent robots $\overline{r}^*, \underline{r}^* \in R$ such that $\overline{r}^*$ resides on an available branch node $b* \in T$, there exists a sequence of moves Π over some time period t_1 to t_2 such that $A(\overline{r}^*, t_2) = A(\underline{r}^*, t_1)$ and $A(\underline{r}^*, t_2) = A(\overline{r}^*, t_1)$.

Proof - At t_1, $\overline{r}^*$ occupies b^* and, because the robots are adjacent, $\underline{r}^*$ occupies one of the available twig nodes of b^*, γ_0. Because b^* is an available branch node, there exists at least two other available twig nodes, γ_1 and γ_2. If $\overline{r}^*$ moves to γ_1, then $\underline{r}^*$ is free to move from γ_0 through b^* to γ_2, which then allows $\overline{r}^*$ to move from γ_1 through b^* to γ_0. Finally, at t_2, $\underline{r}^*$ can move from γ_2 to b^*, such that $b^* = A(\underline{r}^*, t_2) = A(\overline{r}^*, t_1)$ and $\gamma_0 = A(\overline{r}^*, t_2) = A(\underline{r}^*, t_1)$, meaning that the two robots have swapped positions (see Fig. 4).

The algorithm guarantees that once the highest priority robot has swapped with any robot meeting one of the four swap conditions described in Section III, those two robots will never need to swap again.

Lemma 3 - Once the highest priority unsolved robot $\overline{r}^*$ has swapped with another robot r, the two robots will not need to swap again for as long as $\overline{r}^*$ is unsolved.

Proof - When the swap between robots $\overline{r}^*$ and r is complete, any of the four conditions listed in Section III that caused the swap to occur will no longer be true. At the completion of the swap, then, $\overline{r}^*$ and r will not immediately need to swap again. As time goes on, no motion by either robot will allow r to come between $\overline{r}^*$ and $g(\overline{r}^*)$, and vice versa. To prove this, consider three cases: r and $\overline{r}^*$ maintain communication; r and $\overline{r}^*$ lose communication while $\overline{r}^*$ is moving away from $g(\overline{r}^*)$; and r and $\overline{r}^*$ lose communication while $\overline{r}^*$ is moving towards $g(\overline{r}^*)$.

Case 1: Whenever the two robots are in communication, r will not move at all unless pushed by $\overline{r}^*$, and therefore cannot come between $\overline{r}^*$ and $g(\overline{r}^*)$.

Case 2: If communication is lost and $\overline{r}^*$ is heading away from its goal, r will wait without moving for $\overline{r}^*$ to return. Therefore, the two robots cannot exchange positions.

Case 3: If communication is lost and $\overline{r}^*$ is heading towards its goal, r may move and even begin other swaps. However, since robots select branch nodes for swaps in order of decreasing priority, r will always select the same branch node as $\overline{r}^*$ and will follow $\overline{r}^*$ to maintain the same relative position between the two. Hence, they will not swap again for as long as $\overline{r}^*$ is the highest priority unsolved robot.

Note that it is possible for another, higher-priority robot to enter the network and move such that $\overline{r}^*$ and r will need to swap again, but in this case $\overline{r}^*$ was not the highest priority robot in the first place, so the conditions stated in Lemma 3 were not met and the lemma is still valid. Next, it will be shown that through a series of swaps a robot becomes solved.

Lemma 4 - The highest priority unsolved robot $\overline{r}^*$ can become solved through a series of swaps and, once solved, will never enter into another swap.

Proof - As robot $\overline{r}^*$ progresses towards its goal and encounters any other robot r_i with which it must swap, the pair will find an available branch node b^* (Lemma 1), and once at b^* $\overline{r}^*$ and r_i will be able to exchange positions (Lemma 2). Once $\overline{r}^*$ and r_i have swapped, the two robots will never again need to swap for as long as $\overline{r}^*$ is the highest priority unsolved robot (Lemma 3). Therefore, in the worst

case, robot $\overline{r}^*$ will swap with every other robot, at which point it must be solved since none of the conditions listed in Section III can be true with any other robots. Once $\overline{r}^*$ has been solved, it will not enter any swaps with other robots. This follows from the definition of solved robots: all robots below $\overline{r}^*$ are solved and therefore do not need to swap with $\overline{r}^*$, and all robots above $\overline{r}^*$ are lower priority than $\overline{r}^*$ and hence have goals up the tree from $\overline{r}^*$, so none of the swap conditions listed in Sec. III are met.

Yet while a robot $\overline{r}^*$ can become solved through a series of swaps, and once solved will not enter into new swaps, it is possible for lower priority robots swapping up the tree from $\overline{r}^*$ to push $\overline{r}^*$ off $g(\overline{r}^*)$. Even in these cases the algorithm prevents $\overline{r}^*$ from becoming unsolved.

Lemma 5 - Once a robot r_s becomes solved, no future action will cause it to become unsolved.

Proof - As shown in Lemma 4, once a robot becomes solved, it will never need to initiate another swap; therefore, the only action that can cause a solved robot (r_s) to leave its goal is getting pushed by a new, lower-priority pair of swapping robots ($\overline{r}^*$ and $\underline{r}^*$). Since a solved robot has no lower priority robots below it in the tree, this swapping pair must be coming from above r_s and will be pushing r_s down the tree. Furthermore, as r_s is pushed down the tree, the *PSW()* algorithm dictates that it will be pushed down toward the lowest priority node available (line 6 of algorithm 4). If r_s maintains communication with $\overline{r}^*$ and $\underline{r}^*$, it will hold its position until the swappers have reached a lower priority node above $A(r_s)$ (because *PSW()* gives right of way to robots on higher priority nodes), at which point r_s can move up the tree towards its goal.

If communication is lost, r_s will wait until it regains communication with $\overline{r}^*$ and $\underline{r}^*$ (since any swapping robots heading down the branch of a solved robot will be heading away from their goals). Once communication is reestablished, r_s will again hold its position, due to the aforementioned right-of-way rule, and then return to its goal after the swapping pair has passed up the tree. Additionally, no new swaps can be initiated below the goal of a solved robot, due to the swap suppresion rule of *PSW()* (line 9 of algorithm 2), so r_s will always observe $\overline{r}^*$ and $\underline{r}^*$ heading up the tree past it before it observes a new swapping pair that keeps it pushed below its goal $g(r_s)$. Regardless of how many swaps may take place with r_s pushed below its goal, the right-of-way rule will ensure that it is the last pushed robot to reach $g(r_s)$, at which time there will again be no lower priority robots below it on the tree.

Proof of Completeness - By Lemma 4, the highest priority robot can be solved with a series of swaps, and, by Lemma 5, once solved will never become unsolved by future swaps. Because robots are solved in order of their priority without un-solving previously solved robots, for a finite number of robots, all robots will be solved. Therefore, for the cases where $|R| \leq |L|-1$, the algorithm is complete.

V. Algorithm Run Time Complexity

To evaluate the run-time complexity of a single call of the *PSW*() algorithm, one can observe that the greatest number of steps occurs in the *CheckSwap*() function (algorithm 2). *CheckSwap*() loops through all robots in $C(r)$ to find the highest priority unsolved robot $\overline{r}^*$ (line 1), a second time on line 9, and again to find $\underline{r}^*$ (line 14). These loops iterate $3\,|C(r)|$ times. Given that $|C(r)| \leq R$, the resulting run-time complexity is $O(R)$ for a single call of *PSW*().

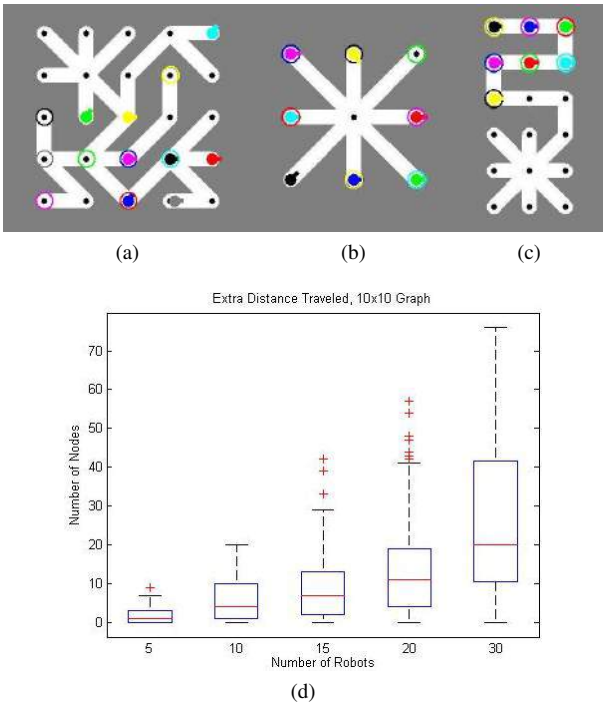


Fig. 5: PSW Performance: In (a), test cases with 10 robots in a randomly generated 5x5 grid, with more difficult cases in (b,c). In (d), extra distance is (Number of nodes traversed) - (Number of nodes on the direct path from start node to goal node). The red line indicates the median number of nodes and the blue box illustrates the interquartile range.

VI. Implementation and Experiments

The first implementation of the PSW algorithm (refer to Scobee and Wiktor thesis, [15]) was tested in MATLAB. Robots were modeled as differential drive non-holonomic robots that are restricted to following paths along the tree T. First, the algorithm successfully solved one hundred problem instances with a random graph of 5x5 nodes and ten robots with random positions and goals (see Fig. 5a). Second, the algorithm successfully solved problems in a sparsely populated map, this time solving one hundred random problem instances with a 10x10 node graph and ten robots. Third, several problem instances were specifically designed to test certain aspects of the implementation, and once again the algorithm successfully solved them all. These included a map with only one branch node and many leafs designed to test the ability of robots to choose and execute swaps, as well as one with a single long branch and a distant branch node designed to test the ability of robots to push others out of the way (see Fig. 5b, c).

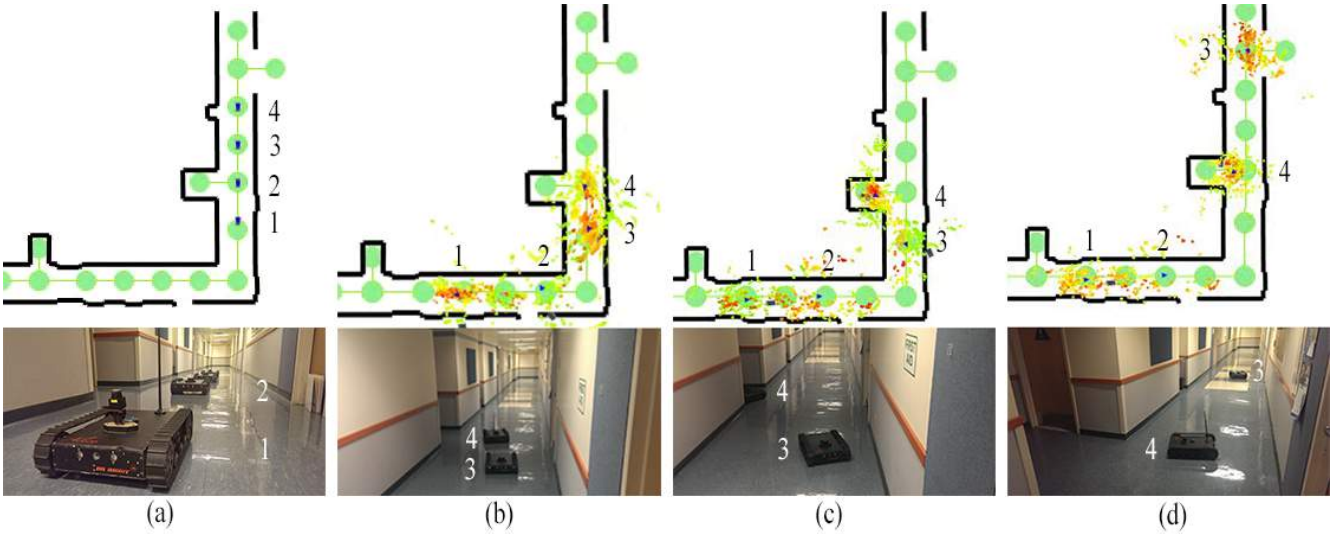


Fig. 6: Jaguars performing a swap in hardware and the graphical interface, where nodes are represented by dark circles and walls are depicted by solid lines. Robots were positioned in the hallway on distinct nodes. In (a), robots start lined up. Next, Robots 1 and 2 move down the hallway (b), while 3 and 4 begin a swap. In (c) Robot 4 moves out of the way of Robot 3. In (d) Robot 3 moves past and the swap is complete. All robots have reached their goal.

To further evaluate the performance of the PSW algorithm, it was tested by generating a set of ten random 10x10 node graphs, then running ten simulations with random robot positions and increasing the number of robots $|R| = 5, 10, 15, 20, 30$. Performance data is presented in Fig. 5(d). As expected, robots are pushed further off their path as the number of robots increases, thereby increasing path length with the number of robots. More interestingly, the worst case individual call to $PSW()$ took on the order of $0.01\,\mathrm{s}$ for cases with 30 robots. This short individual planning time demonstrates the scalability of the algorithm to large numbers of robots.

The algorithm was also implemented on the Dr. Robot Jaguar Lite platform (see Fig. 1). The Jaguar Lite is a rugged, treaded, outdoor robotic platform capable of speeds up to 2.4 m/s. The robot is outfitted with two encoder equipped motors, a GPS, a 9 DOF IMU, a 30 FPS color camera, and a $4\,\mathrm{m}$ range model Hokuyo URG laser scanner. A computer running C# code, including the planner, communicates with robots over an ad hoc WiFi 802.11N network. Four robots were able to successfully use the algorithm as a decentralized approach to all reach their respective goals. Communication was limited to 2 edges and messages were passed between robots in a decentralized manner. Online localization was performed with a particle filter fusing odometry, inertial measurement unit data, and laser scan data. Robots were able to navigate without collision and performed as expected as shown in Fig. 6.

These hardware experiments validate the approach is feasible on real systems where robots are running asynchronously and communication is not always possible.

VII. Conclusion

Unlike all previous work, the Push-Swap-Wait algorithm presented is a complete solution to the problem of effectively coordinating the motion of many autonomous agents navigating a graph structure G in real time without reliance on global communication. The decentralized nature of the algorithm allows each robot $r \in R$ to plan its next move without full knowledge of the other robots in the system, but with a subset of information from its ad-hoc communication network $C(r)$. Even with this limited information, the algorithm was proven to always find a solution in cases where G can be transformed into a tree T such that $|R| \leq |L| - 1$ and the radius of communication ρ is greater than or equal to two edge lengths on this tree. In contrast to other multi-robot motion planning algorithms presented to date, the Push-Swap-Wait algorithm is guaranteed to find a solution and avoid deadlocks while still offering the advantages of a decentralized control architecture.

VIII. Future Work

An important next step for this work will be to fully explore the quality of solutions found by the PSW algorithm, e.g. in terms of path length. The numerous other algorithms referenced that are complete, although not decentralized, have strived to increase optimality. These algorithms will provide a means for comparison if implemented in similar test environments. The authors believe there are several opportunities to modify the algorithm to decrease path length. Also, relaxing the constraint $|R| \leq |L|-1$ to provide completeness for a greater range of planning problems would increase the value of the approach.

REFERENCES

[1] Ayanian, N., Kumar, V.: Decentralized feedback controllers for multi-agent teams in environments with obstacles. In: Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pp. 1936 –1941 (2008). DOI 10.1109/ROBOT.2008.4543490

[2] Cao, Y., Fukunaga, A., Kahng, A.: Cooperative mobile robotics: Antecedents and directions. Autonomous Robots **4**(1), 7–27 (1997). URL http://www.ingentaconnect.com/content/klu/auro/1997/00000004/00000001/00125219

[3] Carpin, S., Pagello, E.: On parallel rrts for multi-robot systems. In: Proc. 8th Conf. Italian Association for Artificial Intelligence, pp. 834–841 (2002)

[4] Clark, C.M.: Dynamic robot networks: A coordination platform for multi-robot systems. Ph.D. thesis (2004)

[5] Dudek, G., Jenkin, M.R.M., Milios, E., Wilkes, D.: A taxonomy for multi-agent robotics. AUTONOMOUS ROBOTS **3**, 375–397 (1996)

[6] Guo, Y., Parker, L.: A distributed and optimal motion planning approach for multiple mobile robots. In: Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on, vol. 3, pp. 2612 –2619 (2002). DOI 10.1109/ROBOT.2002.1013625

[7] Khorshid, M.M., Holte, R.C., Sturtevant, N.R.: A polynomial-time algorithm for non-optimal multi-agent pathfinding. In: Fourth Annual Symposium on Combinatorial Search (2011)

[8] Krontiris, A., Luna, R., Bekris, K.E.: From feasibility tests to path planners for multi-agent pathfinding. In: Sixth Annual Symposium on Combinatorial Search (2013)

[9] Luna, R.: Efficient multi-robot path planning in discrete spaces. Master's thesis, University of Nevada (2011)

[10] Luna, R., Bekris, K.E.: Efficient and complete centralized multi-robot path planning. In: SOCS (2011)

[11] Luna, R., Bekris, K.E.: Push and swap: Fast cooperative path-finding with completeness guarantees. In: International Joint Conference on Artificial Intelligence, pp. 294–300 (2011)

[12] Masehian, E., Nejad, A.H.: Solvability of multi robot motion planning problems on trees. In: Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, pp. 5936–5941. IEEE (2009)

[13] Parker, L.E.: Path Planning and Motion Coordination in Multiple Mobile Robot Teams (2009)

[14] Peasgood, M., Clark, C.M., McPhee, J.: A complete and scalable strategy for coordinating multiple robots within roadmaps. IEEE Transactions on Robotics **24.2** (2008)

[15] Scobee, D., Wiktor, A.: Decentralized and complete multi-robot motion planning in confined spaces. Princeton University Undergraduate Thesis (2012). URL http://www.hmc.edu/lair/publications/2012/scobee_thesis_2012

[16] Standley, T., Korf, R.: Complete algorithms for cooperative pathfinding problems. In: IJCAI, pp. 668–673 (2011)

[17] Velagapudi, P., Sycara, K., Scerri, P.: Decentralized prioritized planning in large multirobot teams. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pp. 4603 –4609 (2010). DOI 10.1109/IROS.2010.5649438

[18] Wang, K.H.C., Botea, A.: Scalable multi-agent pathfinding on grid maps with tractability and completeness guarantees. In: ECAI, pp. 977–978 (2010)

[19] Wang, K.H.C., Botea, A.: Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. Journal of Artificial Intelligence Research **42**(1), 55–90 (2011)