Planning for a Ground-Air Robotic System with Collaborative Localization

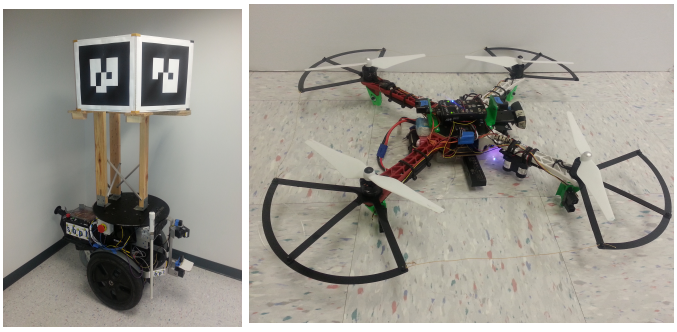
Jonathan Butzke[‡], Kalin Gochev[†], Benjamin Holden[‡], Eui-Jung Jung[⋆], Maxim Likhachev[‡]

***Abstract*— Robots are increasingly being used in situations such as search and rescue that require robust navigation capabilities, potentially in areas with little or no GPS or other high-quality localization information. As more robots are used in these scenarios, it becomes viable to collaborate between heterogeneous types of robots to leverage their individual strengths while minimizing their weaknesses. More specifically, in a scenario involving unmanned ground and aerial vehicles (UGV, UAV), the ground robot can contribute its high payload capacity to provide computational resources and high accuracy sensors while the aerial robot can bring its high mobility and capability to traverse obstacles to the team. However, in order for the team to benefit from these capabilities, it must be capable of generating a plan for both robots that allows them to collaboratively localize when necessary. Our approach to this problem is to combine a recently developed state lattice planner using controller-based motion primitives (SLC) with planning using adaptive dimensionality (PAD). The SLC planner allows for robust navigation using a wide variety of sensors including in areas with no or limited high-quality localization information while the PAD planner allows us to expand beyond a single robot and generate plans for a team of robots operating in a high dimensional space. We present our results to this combined approach for a UGV/UAV team operating indoors in areas with limited visual features.**

I. Introduction

Robots are being turned to in an increasing number and variety of situations. With this proliferation come many opportunities to collaborate between robots in different ways. Robots can provide computational or sensing resources for each other, they can act as transports, provide communication relays, or provide other support. For these types of tasks, the robots need to be able to generate plans that take into account the differences in movement, sensing, and localization abilities of the team members in order to take full advantage of the teams capabilities.

In some scenarios, these differences within the team can be significant. Teams composed of a ground vehicle and an aerial vehicle differ in many important ways. They have drastically different on-station endurance times, different payload capacities (which impact the number and types of sensors), and can traverse different types of terrain. However, these differences can be used to make the team more capable than they are individually. An example is tasks such as search and rescue where both the high endurance of the unmanned ground vehicle (UGV) and the capability to traverse debris strewn environments typical of unmanned aerial vehicles (UAV) are important. In this scenario, the UAV's limited payload places limits on the sensors it can carry while the environment places external limitations on the availability of common localization methods such as GPS. It is important that the planner is capable of generating trajectories that use all of the capability of both vehicles including the ability to gain information from each other. In this paper we propose a novel method of generating trajectories for ground-air teams of robots that allow the team to collaboratively localize.



(a) (b)

Fig. 1: (a) Segway-based UGV. 2 scanning laser range finders, high gain antenna, webcam, general purpose server. (b) Pixhawk-based quadcopter UAV. Laser Altimeter, 6 IR range sensors, standard webcam.

Our approach is based on the recently developed state lattice planner using controller-based motion primitives (SLC)[1] to allow plans to incorporate multiple different modes of localization that a robot has available along with the associated collaboration constraints into a unified planning framework. Additionally, our planner is based on the Planning with Adaptive Dimensionality (PAD) framework[2] to ensure that planning in the six-dimensional[1] combined state space of a ground-air robot team can be performed in reasonable times.

A state lattice-based planner uses a regular lattice constructed from motion primitives [3] to form the search graph, $\mathcal{G} = (\mathcal{S}, \mathcal{E})$. In a typical planner, the edges, $\mathcal{E}$, are formed by applying fixed motion primitives at each state, $s \in \mathcal{S}$. These metric motion primitives carry the implicit assumption that the robot has sufficient localization ability to

Thank you to Conor Donohue for his assistance designing and manufacturing several of the components on the UAV.

This work was supported by ONR grant N00014-15-1-2129.

[†] GRASP Laboratory, University of Pennsylvania, Philadelphia, PA, USA kgochev@seas.upenn.edu

[‡] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA {jbutzke, bholden1}@andrew.cmu.edu, maxim@cs.cmu.edu

[⋆] Manufacturing Robotics R&D Department, Korea Institute of Robots and Convergence, Gyeongsangbuk-do, Korea ejjung@kiro.re.kr

[1]The combined state space is nine-dimensional, however, as discussed in Section IV-D, we can independently determine three of the states, leaving only six for the planning problem.

be able to execute the motion and to determine the stopping point. However, in cases that this does not hold true, we can instead turn to controller-based motion primitives. By adding additional directed edges to the search graph based on forward simulating different types of controllers, the planner is capable of finding trajectories through areas that were impassable using only metric-based motion primitives. These controller-based motions rely solely on the capabilities of the controller independent of the robots ability to localize. For example, a wall following controller may not at any point during its trajectory know where in the environment it is with any degree of precision, however, by executing this controller to its natural stopping point (i.e. the end of the wall) the robot ends up in a known (and repeatable) position. Our implementation of SLC is detailed in Section III.

The second component to our collaborative planner is the use of the planning with adaptive dimensionality framework (PAD). With a free flying aerial robot the state space has six degrees of freedom, $\langle x, y, z, \phi, \theta, \psi \rangle$. When the aerial robot is combined with the ground robot the overall state space increases to 9 states - position and orientation of the UGV, $\langle x, y, \psi \rangle$, plus the six UAV states. Some of these states can be determined independently from the planning problem (roll, pitch, and yaw of the UAV) leaving a six-dimensional state space. It is well known that this high dimensionality coupled with a large environment greatly increases planning times to the point that they become infeasible for on-the-fly computation. By planning with adaptive dimensionality, we can plan in only those dimensions that are critical at a given point. For example, consider moving a piano across a driveway and into a large room in a house. While the piano and the movers are transiting the driveway, the orientation of the piano is of no consequence - all orientations are equally valid. The same holds true once the piano is inside the large room. However, while the piano is moving through the narrow doorway, its orientation matters immensely - most orientations are incapable of getting through and in fact, a carefully orchestrated sequence of orientations is often required for success. In this example, the planner can plan using just the $\langle x, y \rangle$ translational states for the first and last portions of the plan, and only plan in the full six-dimensional state space in the vicinity of the doorway. In Section IV we describe our implementation of PAD.

We will discuss the history and alternative approaches to solving these problems in Section II, and present our experimental results in Section V.

II. Related Work

Collaborative localization has been a goal of robotics research for many years[4]. A lot of this work has been directed at making the detection of the other robots of a team more reliable and accurate[5] even for chains of robots where the farthest ones have no direct knowledge or sensor measurements regarding any known landmarks and instead must rely entirely on their neighboring robots[6]. Other approaches have focused on the sensor integration from the data fusion side ensuring that the data is used more effectively[7]. Our approach keeps the localization scheme simple, we use only fiducial markers and a simple camera to determine the estimated pose of the UGV from the UAV and then, knowing the UGV position, we can estimate the position of the UAV. While we do not use these advanced techniques in this work, our algorithm is capable of incorporating this improved data into its planning framework.

With the recent increase in the availability of small, low cost UAV's, in particular easy to use quadcopters, more research effort has been directed at teams of air-ground robots[8] including work on exploration[9], and collaborative localization between the team members[10]. Communications in a variety of forms has been the focus of several works in this area[11], [12], although frequently these include high-quality localization of all robots, including the use of GPS on both the UGV and UAV's even with vision augmentation[13]. However, some approaches rely purely on well-localized UGVs[14], forcing the UAV to update its position estimate only by visually extracting the pose of the UGV. Our work differs from all of these by incorporating the collaborative localization element into a larger planning framework.

An in-depth look at multi-robot localization and planning for air-ground teams of robots is found in [15]. This planner uses a simplified topological approach to planning and is not sufficient to be used directly by the UGV and UAV for navigation through a complex 3-dimensional environment.

The state lattice with controller-based motion primitive planner is based on standard graph search algorithms such as A* [16] and ARA* [17] but allow the execution of controllers similar to the sequential composition of controller approaches[18], [19], [20]. The SLC planner also includes the functionality of switching between controllers based on external perceptual triggers similar to the Linear Temporal Logics[21].

Planning in a high-dimensional configuration space, such as a team of robots, is a challenging task. A variety of techniques have been developed in order to improve planning times in high-dimensional configuration spaces. Many approaches involve a two layer planning scheme with a low-dimensional global planner that provides input to a high-dimensional local planner [22], [23], [24]. However, these approaches can result in highly sub-optimal trajectories and even trajectories that are infeasible due to mismatches in the assumptions made by the global and local planners. Another way to reduce the dimensionality of planning problems is through the use of hierarchical planners and state abstraction techniques, such as the approaches taken in [25], [26]. These approaches aim to compute very accurate heuristics in order to improve the performance of searches through the high-dimensional state-space [27]. The heuristics are derived by solving a relaxed lower-dimensional representation of the original planning problem. The PAD planning algorithm used in this work differs from the approaches above in several aspects. It does not explicitly split the planning in two levels, but rather mixes low- and high-dimensional states within a single planning process. PAD can also make effective use of a very accurate heuristic, but it does not

rely on the heuristic alone to improve performance. Thus, it is more robust to handling local minima in the heuristic function. Additionally, the PAD algorithm provides strong theoretical guarantees, such as completeness with respect to the underlying graph, and bounds on solution cost sub-optimality.

The key features that distinguish our work from the prior work in the field is that we include the collaborative localization element directly in our planning process. This allows the robots to go on separate trajectories and only meet up when required rather than travel in a fixed formation or conversely, operate completely independently. In addition, our planner provides guarantees on path quality and resolution completeness.

III. State Lattice-based Planning with Controller-based Motion Primitives for Ground-Air Teams

A. State Lattice Planners with Controller-based Motion Primitives

A state lattice planner uses predefined motion primitives to generate, as required, a graph, $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ spanning the environment. Motion primitives are short kinematically feasible motions that are designed to connect one state with another nearby state and form the set of edges, $\mathcal{E}$, of the search graph [3]. This graph is traversed by a graph search algorithm such as A* in order to find the minimal cost path from the start state s_S to the goal state s_G, $s_S, s_G \in \mathcal{S}$. During the planning cycle, the graph generation and planning process are interleaved so that only those elements of the graph that are required for the search algorithm are explicitly constructed. The SLC planner[1] modifies this construct by adding additional directed edges to $\mathcal{G}$ that correspond to executing a controller c from a given set of controllers $\mathcal{C}$, at a given start state. These new edges are formed by forward simulating the desired controller from a given state, s_i, in order to determine the end state, s_j, and thus forming a new edge, $e(s_i, s_j)$, which is added to the set of states in the search graph, $\mathcal{E}' = \mathcal{E} \bigcup^n e_n$, then $\mathcal{G} = (\mathcal{S}, \mathcal{E}')$.

Formally, SLC requires three functions to be defined to generate the graph $\mathcal{G}$, $\mathrm{C}(s)$, $\mathrm{T}(c)$, and $\Phi(s, c, \tau)$. The first function, $\mathrm{C}(s)$ defines the available controllers at a given state, $s \in \mathcal{S}$:

$$\mathrm{C}(s) : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{C})$$

The result is a set of available controllers, $\mathscr{C}$, from the powerset of all controllers, $\mathcal{P}(\mathcal{C})$, i.e. $\mathrm{C}(s)$ provides all of the controllers which can be executed at state s. These controllers can be simple, such as a wall following controller using two range measurements, or complex, such as a full visual odometry system to navigate to a particular key-frame.

The SLC algorithm also allows controllers to be stopped in the middle of execution through the use of perceptual triggers. A trigger can be setup to halt a controller based on any perceptual signal, such as sighting a new landmark. In addition, each controller has an intrinsic trigger that is the default stopping point for that controller. For example, a wall following controller has an intrinsic trigger that stops execution when the robot reaches the end of the wall. An example of various controllers and triggers is shown in Fig. 2.

The second required function maps the controllers onto available triggers:

$$\mathrm{T}(c) : \mathcal{C} \rightarrow \mathcal{P}(\mathcal{T})$$

returning a set of available triggers, $\mathscr{T}$, based on the given controller, $c \in \mathcal{C}$. $\mathcal{T}$ is the set of all triggers available to the robot.

The last required function is the actual controller logic defined as:

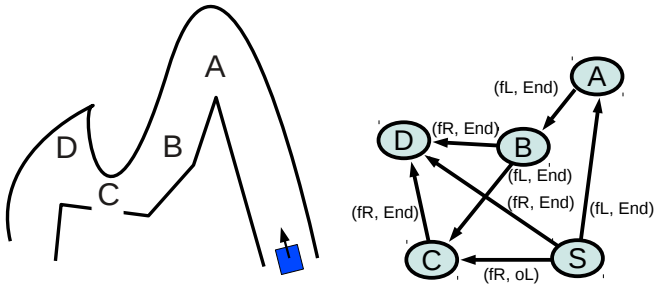
$$\Phi(s, c, \tau) : \mathcal{S} \times \mathrm{C}(s) \times \mathrm{T}(c) \rightarrow \mathcal{S}$$

For a given state s, an allowable controller c for that state, and an allowable trigger τ for that controller, function Φ simulates the execution of the controller c starting at state s until either trigger τ or an intrinsic trigger is detected (whichever comes first). The resulting state s' is returned by the function.

The problem is thus formally a 6-tuple,

$$\mathcal{G} = \{\mathcal{S}, \mathcal{C}, \mathcal{T}, \mathrm{C}(\cdot), \mathrm{T}(\cdot), \Phi(\cdot, \cdot, \cdot)\}$$

used to produce the graph, $\mathcal{G}$. Further details on the algorithm can be found in [1].



(a) (b)

Fig. 2: (a) Environment and (b) segment of graph $\mathcal{G}$ based on controllers $\mathcal{C} = \{\textsc{FollowLeftWall}(fL), \textsc{FollowRightWall}(fR)\}$ and triggers $\mathcal{T} = \{\textsc{Completion}(End), \textsc{OpeningLeft}(oL), \textsc{OpeningRight}(oR)\}$.

As an example, suppose we are given a set of controllers[2] $\mathcal{C} = \{\textsc{FollowLeftWall}, \textsc{FollowRightWall}\}$ with an intrinsic trigger of Completion corresponding to the end of the wall, and a set of extrinsic triggers $\mathcal{T} = \{\textsc{OpeningLeft}, \textsc{OpeningRight}\}$. Given an example environment as shown in Fig. 2a we can see how the graph, $\mathcal{G}$, is constructed in Fig. 2b. Consider a state S, indicated by the square in the lower right corner and suppose both controllers are available at S. From state S there is an edge to A corresponding to the controller FollowLeftWall, fL, and trigger Completion, End, as shown in the portion of $\mathcal{G}$. Likewise, with controller FollowRightWall, fR, and trigger End, the edge goes from S to D. However, if

[2]For the sake of illustration, it does not include metric motion primitives.

the trigger were OPENINGLEFT, oL, then the edge would have been from S to C. Note, it is possible for multiple controller/trigger combinations to connect two nodes. For example, $B \rightarrow C$ is formed by the (fL, End) pair in the graph, however $B \rightarrow C$ is also connected by the pair (fR, oL) (which is not depicted).

B. Controllers and Triggers Implemented for Ground-Air Teams

In order to use the SLC planner, a set of available controllers and triggers were constructed. For the UAV, we implemented WALLFOLLOWING, GOTOLANDMARK, METRICTURN, and METRICSTEP[3] controllers. The UGV had high-quality localization data from its two scanning laser range finders and so was only given a METRICMOTION controller.

The WALLFOLLOWING controller on the UAV used two IR range sensors mounted on each side of the UAV in order to maintain a flight path parallel to, and a specified distance from, any given wall in the environment. It was given the ability to trigger when the wall ended (COMPLETION) and when an obstacle was within a certain distance of the front or back of the UAV (OBSTACLE). This sensing system was noisy and the associated controller was very simplistic resulting in only marginally stable flight. However, it should be noted that our overall architecture works in spite of the poor implementation of the controllers - a better controller would only improve the overall performance.

We used two different instantiations of the GOTOLANDMARK controller. The first ($\text{GOTOLANDMARK}_{\text{STAT}}$) used static landmarks in the environment that the UAV could detect with its onboard camera system and knowing the position and orientation of the landmark, could determine its own position. The second controller ($\text{GOTOLANDMARK}_{\text{DYN}}$) used fiducial markers on the ground robot for the same purpose. However, this then requires that during the planning cycle the UAV and UGV positions are both considered simultaneously in order for the collaborative localization to occur (see Section IV-D). In other words, $\text{GOTOLANDMARK}_{\text{DYN}}$ could only be exercised while the UGV was close enough and within line-of-sight of the UAV.

Since the UAV does have an IMU and optical flow system there are locations within the environment that it is capable of generating short range metric motions. We used two such motions, an ability to yaw to a desired heading, and the ability to move a set distance forward. The accuracy of the IMU and optical flow system did not allow for continuous metric motion without receiving some external sensor information so the planner limited the allowable locations during planning time by imposing a high cost on these motions.

IV. PLANNING WITH ADAPTIVE DIMENSIONALITY

The search-based planning framework for Planning with Adaptive Dimensionality builds on the fact that many high-dimensional path planning problems have lower-dimensional projections that represent the problem very well in most areas. For example, path planning for a non-holonomic vehicle needs to consider the planar position of the vehicle $\langle x, y \rangle$, but also the heading angle, ψ, to ensure that system constraints, such as minimum turning radius, are obeyed. However, a two-dimensional representation of the problem, only considering the planar position of the vehicle $\langle x, y \rangle$, can work well in many areas of the state-space (Fig. 3). For our work, the high-dimensional states will be the full planning state space $\langle (x, y, z)_{\text{uav}}, (x, y, \psi)_{\text{ugv}} \rangle$ while the low-dimensional states will be just the UAV position, $\langle x, y, z \rangle_{\text{uav}}$.

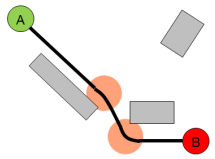


Fig. 3: Example trajectory for a non-holonomic vehicle with minimum turning radius constraints. Planning for the heading of the vehicle is needed in areas that require turning in order to ensure constraints are satisfied (light red circles). Planning for the heading of the vehicle is unnecessary for areas that can be traversed in a straight line. A: start location; B: goal location; gray boxes: obstacles.

In this section we will provide a brief overview of the PAD algorithm. For a more detailed explanation of the algorithm, we refer the reader to the original work [2], [28]. We will use the notation $\pi_{\mathcal{G}}(s_i, s_j)$ to denote a path from state s_i to state s_j in a graph[4] $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ with a vertex set $\mathcal{S}$ and edge set $\mathcal{E}$. The cost of a path π will be denoted by $c(\pi)$. We will use $\pi^*_{\mathcal{G}}(s_i, s_j)$ to denote a least-cost path and $\pi^{\epsilon}_{\mathcal{G}}(s_i, s_j)$ for $\epsilon \geq 1$ to denote a path of bounded cost sub-optimality: $c(\pi^{\epsilon}_{\mathcal{G}}(s_i, s_j)) \leq \epsilon \cdot c(\pi^*_{\mathcal{G}}(s_i, s_j))$.

A. Overview

The PAD algorithm considers two graphs as defined by their corresponding state-spaces and transition sets—a high-dimensional $\mathcal{G}^{hd} = (\mathcal{S}^{hd}, \mathcal{E}^{hd})$ with dimensionality h, and a low-dimensional $\mathcal{G}^{ld} = (\mathcal{S}^{ld}, \mathcal{E}^{ld})$ with dimensionality l, where $\mathcal{S}^{ld}$ is a projection of $\mathcal{S}^{hd}$ onto a lower dimensional manifold ($h > l, |\mathcal{S}^{hd}| > |\mathcal{S}^{ld}|$) through a projection function λ.

$$\lambda : \mathcal{S}^{hd} \rightarrow \mathcal{S}^{ld}$$

The projection function λ^{-1} maps low-dimensional states to their high-dimensional pre-images:

$$\lambda^{-1} : \mathcal{S}^{ld} \rightarrow \mathcal{P}(\mathcal{S}^{hd})$$

and is defined as

$$\lambda^{-1}(X^{ld}) = \{X \in \mathcal{S}^{hd} | \lambda(X) = X^{ld}\}$$

where $\mathcal{P}(\mathcal{S}^{hd})$ denotes the power set of $\mathcal{S}^{hd}$.

[3]The two metric controllers were always used in groups to turn corners: METRICSTEP(0.5m) $\rightarrow$ METRICTURN($\pm 90°$) $\rightarrow$ METRICSTEP(1.0m)

[4]The SLC output is a graph, however, since the graph construction and planning are interleaved, $\mathcal{E}$ is generated online from $\{\mathcal{C}, \mathcal{T}, \mathrm{C}(\cdot), \mathrm{T}(\cdot), \Phi(\cdot, \cdot, \cdot)\}$ as defined in Section III-A

Each of the two state-spaces may have its own transition set. However, in order to provide path cost sub-optimality guarantees, the algorithm requires that the costs of the transitions be such that for every pair of states s_i and s_j in $\mathcal{S}^{hd}$,

$$c\left(\pi^*_{G^{hd}}\left(s_i, s_j\right)\right) \geq c\left(\pi^*_{G^{ld}}\left(\lambda(s_i), \lambda(s_j)\right)\right) \tag{1}$$

In other words, it is required that the costs of least-cost paths in the low-dimensional state-space always underestimate the costs of the least-cost paths between the corresponding states in the high-dimensional state-space.

B. Algorithm

The PAD algorithm iteratively constructs and searches a hybrid graph $\mathcal{G}^{ad} = (\mathcal{S}^{ad}, \mathcal{E}^{ad})$ consisting mainly of low-dimensional states and transitions. The algorithm only introduces regions of high-dimensional states and transitions into the hybrid graph where it is necessary in order to ensure the feasibility of the resulting path and maintain path cost sub-optimality guarantees. Each iteration of the algorithm consists of two phases: planning phase and tracking phase.

In the planning phase, the current instance of the hybrid graph $\mathcal{G}^{ad}$ is searched for a path $\pi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{ad}}(s_S, s_G)$. Any graph search algorithm that provides a bound on path cost sub-optimality can be used to compute $\pi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{ad}}$. Similar to the original implementation of the algorithm, we used the weighted A* graph-search algorithm [2], [28].

In the tracking phase, a high-dimensional tunnel τ (a subgraph of $\mathcal{G}^{hd}$) is constructed around the path found in the planning phase. Then τ is searched for a path $\pi_\tau(s_S, s_G)$ from start to goal. If $c(\pi_\tau) \leq \epsilon_{\text{track}} \cdot c(\pi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{ad}})$, then π_τ is returned as the path computed by the algorithm. If no path through τ is found or $c(\pi_\tau)$ does not satisfy the above constraint, the algorithm identifies locations in $\mathcal{G}^{ad}$, where the search through τ got stuck or where large cost discrepancies between $\pi^{\epsilon_{\text{plan}}}_{\mathcal{G}^{ad}}$ and π_τ are observed. The algorithm then introduces new high-dimensional regions in $\mathcal{G}^{ad}$ centered at the identified locations. For more details on how the locations of new high-dimensional regions are computed and how high-dimensional regions are introduced in $\mathcal{G}^{ad}$, please refer to [2], [28]. The algorithm then proceeds to the next iteration.

C. Theoretical Properties

If the high-dimensional state-space $\mathcal{S}^{hd}$ is finite, the PAD algorithm is complete with respect to the underlying graph $\mathcal{G}^{hd}$ encoding the search problem and is guaranteed to terminate. If a path π is found by the algorithm, then π satisfies

$$c(\pi) \leq \epsilon_{\text{plan}} \cdot \epsilon_{\text{track}} \cdot \pi^*_{\mathcal{G}^{hd}}(s_S, s_G)$$

In other words, the cost of a path returned by the algorithm is bounded by $\epsilon_{\text{plan}} \cdot \epsilon_{\text{track}}$ times the cost of an optimal path from s_S to s_G in the high-dimensional graph $\mathcal{G}^{hd}$, where ϵ_{plan} and ϵ_{track} are user-specified parameters. These theoretical properties are proven in [2].

D. Application to Multi-Robot State-Lattice Planning

In the particular application considered in this work, the task is to navigate a UAV with limited self-localization capabilities to a desired target location with the assistance of a UGV with good localization capability. The aerial vehicle is able to localize itself relative to the ground vehicle, when the ground vehicle is visible from the UAV's position. The full-dimensional system state is represented by 6-dimensional state-vectors: $\langle (x, y, z)_{\text{uav}}, (x, y, \psi)_{\text{ugv}} \rangle$ The transitions available for each state consist of pre-computed motion primitives (metric motions) for both vehicles, and state-lattice controller actions for the aerial vehicle. The cost of each transition is proportional to the cumulative distance traveled by each vehicle during the transition. The roll and pitch of the UAV, ϕ and θ, are derived variables from the desired velocity and error and are calculated by the controllers to meet the desired trajectory points. The heading (yaw), ψ, of the aerial vehicle is also not a free variable and is determined by the specific controller used in a transition. For example, when executing a WALLFOLLOWING transition, the heading is kept parallel to the direction of the wall, and for transitions using the ground vehicle for localization (GOTOLANDMARK$_{\text{DYN}}$), the heading is kept facing the ground vehicle. We assume that in many areas of the environment the aerial vehicle is capable of autonomous navigation by using the state-lattice controllers (following walls or going around corners, for example), and the localization assistance of the ground vehicle is needed only in rare occasions, when no state-lattice controllers are available to the UAV and metric motions need to be performed. Thus, the low-dimensional representation of the system used for Planning with Adaptive Dimensionality is a 3-dimensional state-vector $\langle x, y, z \rangle_{\text{uav}}$, only considering the position of the aerial vehicle. The costs of transitions in the low-dimensional space satisfies (1) as only the cost of moving the aerial vehicle is considered. High-dimensional regions are introduced in the hybrid graph only in areas of the environment where ground vehicle localization assistance is needed.

E. The Output of the Planner

A state lattice with controller-based motion primitive planner generates trajectories that are defined as a series of controllers to execute. For this collaborative planner we expand that to include at each time step the appropriate controller for all robots in the team. When the robots are operating independently, the trajectory execution finite state machine of each robot independently tracks where it is in the plan. When a robot reaches a planner step that requires another robot to be at a specific location, the first robot will pause and hold position until the other one finishes its controller sequences preceding that point. When the two are back in sync, both will be allowed to continue executing their controllers. In practice, the UAV rarely has to wait for the UGV with the one common occurrence being using GOTOLANDMARK$_{\text{DYN}}$ motion that went behind a pillar. The UAV would move as far as it could and still see the UGV,

which would then make a quick motion to the side to allow the UAV to continue on.

V. Experimental System

A. Robots

For our testing we used two robots: a Segway-based ground robot named Melvin (Fig. 1a), and a Pixhawk/DJI-based aerial robot (Fig. 1b). All of the computers ran Kubuntu 12.04 and ROS Groovy with the exception of the computer on the aerial vehicle which runs Xubuntu 14.04 and ROS Indigo.

Melvin the UGV is a relatively large indoor robot with a significant payload capacity and high endurance. With a normal operating load, Melvin is capable of operating for 3+ hours running two independent computer systems, and carrying all required communications infrastructure. The first computer system is used as the low level controller and consists of an i3 3.4GHz processor with 8GB of RAM. This system is used for all navigation, sensing, and interfacing with the Segway base. The second system is a general purpose computer equipped with a dual processor Xeon with 8 physical cores and 16GB of RAM. The planner and plan execution agent are both run on this computer. In addition, this computer is used to run the processor intensive tasks of the UAV such as AR marker detection/extraction and all of the mid-level controllers (the wall following controller, the metric motion controllers, and the landmark controllers). Melvin is also equipped with two Hokuyo scanning laser sensors mounted on tilt mounts for a full 3-dimensional scanning capability, and a web camera for visual sensing. To assist the UAV with collaborative localization, the UGV has six AR markers arranged in a horizontally aligned hexagon so that the UAV can detect and accurately determine orientation of the UGV even in the presence of some low obstacles.

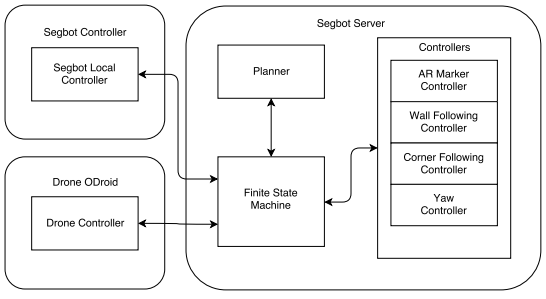
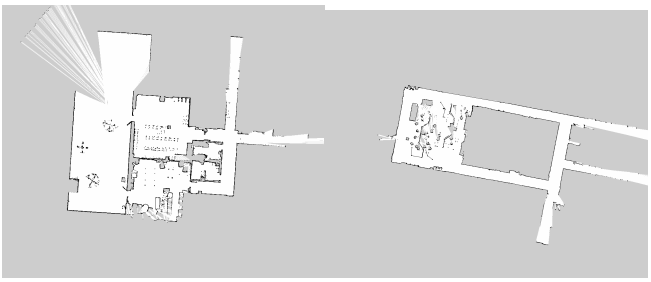


Fig. 4: Block diagram of the system. The server was physically located on the ground robot and performed most of the processing of the UAV camera data.

Unlike the UGV, the UAV is relatively spartan in terms of sensing and computing power. The airframe itself is a DJI Flamewheel 450 with a Pixhawk flight control computer and an ODROID XU3 supplemental computer. A standard web camera is used for landmark detection, while 6 Sharp IR sensors with 1.5m range are arranged around the perimeter to provide obstacle detection and wall following capabilities. The ODROID captures the images and transmits them to the UGV for processing, then receives the output from the mid-level controllers and translates them into the required format for the Pixhawk to execute.

B. Environment

Our test environments are meant to replicate a standard indoor office environment (see Fig. 5). We used one area that consisted of two large conference rooms, an outdoor patio area, and a few hallways with small offices. The other test area was comprised of a cluster of cubicles, boxes, equipment, and office furniture in half the area, while the other half is a set of featureless hallways. For our experiments, we restricted the UGV to operate only in the room portions of the environments by placing obstacles at each hallway entrance. The UAV was free to operate throughout the map with different areas performing better with different controllers. For example, since the hallways had no features and the UGV was unable to enter them, the GoToLandmark controllers were not usable (for both static and dynamic landmarks). On the other hand, the crowded, erratically configured cubicle area did not feature any navigable straight walls and thus required cooperation between the UGV and UAV in order to successfully navigate to a desired location. In addition, since the UAV only had a very limited set of available controllers, it required the collaborative localization capability to get to any goal that was not located at the terminus of a wall.



(a) Conference Rooms and Patio (b) Cubicles and Hallways

Fig. 5: Maps of two testing environments.

C. Test Setup

To test our planner's performance in real-world scenarios, we randomly selected start and goal points throughout the environment for the UAV and start points only for the UGV. This allowed us to construct plans where the two robots started near each other but allowed the UAV to operate independently if required. The planner would allow the UGV to move as necessary to support the UAV motion to get to the goal.

The cost function used for these experiments was proportional to the time and distance traversed for each motion.

D. Test Results

Overall the system is able to generate plans that would not be solvable without using the controller-based motion

TABLE I: Experimental Results Conference Room Fig. 5a

Algorithm	Planning Time (s) Avg.	Std. Dev.	Min	Max	Num. Iter. Avg.	Num. Expansions Avg.	Path Cost Avg.	Final Eps. Avg.	Success Rate (%) @3min
PAD MR SLC	20.05	22.00	1.19	91.58	1.59	7348	24401	1.36	100
Full-D ARA* MR SLC	9.37	20.32	0.08	128.04	n/a	1385	23960	1.30	82

TABLE II: Experimental Results Cubicles Fig. 5b

Algorithm	Planning Time (s) Avg.	Std. Dev.	Min	Max	Num. Iter. Avg.	Num. Expansions Avg.	Path Cost Avg.	Final Eps. Avg.	Success Rate (%) @3min
PAD MR SLC	12.74	9.95	1.19	40.00	1.26	2736	36112	1.31	100
Full-D ARA* MR SLC	12.33	22.23	0.01	88.65	n/a	960	38083	1.38	38

primitives due to the lack of an adequate localization capability of the UAV operating alone. In addition, the adaptive dimensionality planner played a key role in making these plans computationally feasible given the high dimensionality of the combined state space. Planning times for 100 randomly generated start-goal pairs on two different indoor environments are shown in Table I and Table II (50 on each). The performance of our collaborative localization algorithm (labeled PAD MR SLC) is compared against a full six-dimensional ARA* algorithm running on the multi-robot SLC. The results shown in Table I and Table II are averaged over the scenarios that both planners were able to solve successfully. Each planner was timed until it reached the first solution, plus up to 5s to improve the solution quality. The full-dimensional ARA* planner was unable to solve the 40 most difficult scenarios within 3 minutes, which was considered a planning failure, whereas the maximum time that our approach took to solve a scenario was 125.3s (for a case that the full-dimensional planner was unable to find a solution). In several of the cases that the full-dimensional planner was able to solve, the heuristic allowed it to find the optimal solution in very few expansions. For environments where this is common, the overhead from our adaptive dimensionality approach may not be beneficial. However, for harder problems, the adaptive dimensionality extension allows solutions to be found when the full-dimensional planner is unable to find any solution within the alloted time. As can be seen, using adaptive dimensionality reduction we are able to produce solutions for even the hardest problems while still remaining competetive in time and solution quality for the easier problems.

An example of a generated plan is shown in Fig. 6 and in our attached video[5]. This particular plan initially has both robots near each other in the open portion of the operating area. The UAV is tasked to move to a location at the end of one of the hallways, then to return the cubicles and land (the second portion is not shown in the figure for clarity). Other testing used random start and goal locations on the two environments shown in Fig. 5.

We did discover that our existing controllers were insufficient to reliably pass through a standard doorway (which provides less than 20cm clearance total around our UAV) and this remains an area for further development.

[5]https://youtu.be/BVhR-Bv09q0

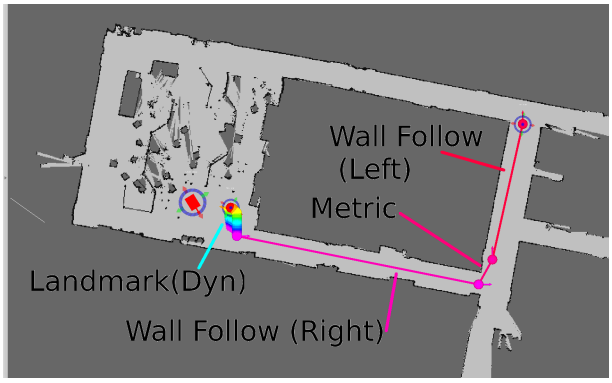


Fig. 6: Portion of plan showing the UAV starting at the lower left and using GOTOLANDMARK$_{DYN}$ motions to get into the hallway. Once in the hallway, the UAV uses WALLFOLLOWING and metric motions to reach the goal position in the upper right. The UGV is the magenta rectangle near the UAV start. The start and goal configurations have a blue circle around them.

VI. CONCLUSION

State lattice-based planning using controller-based motion primitives combined with an adaptive dimensionality planner provides a method of solving complex, high-dimensional navigation problems that cannot be solved using typical existing methods. This is due to the computation requirements of planning in a high-dimensional space and the inherently limiting assumption that most planners make on the existence of sufficient localization ability of the target robots.

In our future work we will explore the limits of scalability for this algorithm. While our setup works well with two robots, additional work is required to determine how well, in terms of planning times, communications constraints, etc., this approach will work for larger teams. In addition, we are also looking at relaxing the known map requirement to incorporate the ability to learn the map in an online fashion.

REFERENCES

[1] J. Butzke, K. Sapkota, K. Prasad, B. MacAllister, and M. Likhachev, "State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Sept 2014, pp. 258–265.

[2] K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev, "Path planning with adaptive dimensionality," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2011.

[3] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2005.

[4] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "Collaborative multi-robot localization," in *Mustererkennung 1999*, ser. Informatik aktuell, W. Frstner, J. Buhmann, A. Faber, and P. Faber, Eds. Springer Berlin Heidelberg, 1999, pp. 15–26. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-60243-6_2

[5] O. De Silva, G. Mann, and R. Gosine, "Development of a relative localization scheme for ground-aerial multi-robot systems," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, Oct 2012, pp. 870–875.

[6] T. R. Wanasinghe, G. K. I. Mann, and R. G. Gosine, "Distributed leader-assistive localization method for a heterogeneous multirobotic system," *IEEE T. Automation Science and Engineering*, vol. 12, no. 3, pp. 795–809, 2015. [Online]. Available: http://dx.doi.org/10.1109/TASE.2015.2433014

[7] K.-T. Song, C.-Y. Tsai, and C.-H. C. Huang, "Multi-robot cooperative sensing and localization," in *Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on*, Sept 2008, pp. 431–436.

[8] S. Lacroix and G. Le Besnerais, "Issues in cooperative air/ground robotic systems," in *Robotics Research*, ser. Springer Tracts in Advanced Robotics, M. Kaneko and Y. Nakamura, Eds. Springer Berlin Heidelberg, 2011, vol. 66, pp. 421–432. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14743-2_35

[9] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *Robotics, IEEE Transactions on*, vol. 21, no. 3, pp. 376–386, June 2005.

[10] I. Rekleitis, R. Sim, G. Dudek, and E. Milios, "Collaborative exploration for the construction of visual maps," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 3, 2001, pp. 1269–1274 vol.3.

[11] A. Viguria, I. Maza, and A. Ollero, "Distributed service-based cooperation in aerial/ground robot teams applied to fire detection and extinguishing missions," *Advanced Robotics*, vol. 24, no. 1-2, pp. 1–23, 2010. [Online]. Available: http://dx.doi.org/10.1163/016918609X12585524300339

[12] R. T. Vaughan, G. S. Sukhatme, F. J. Mesa-Martinez, and J. F. Montgomery, "Fly spy: Lightweight localization and target tracking for cooperating air and ground robots," in *Distributed autonomous robotic systems 4*. Springer, 2000, pp. 315–324.

[13] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas, "Cooperative air and ground surveillance," *Robotics Automation Magazine, IEEE*, vol. 13, no. 3, pp. 16–25, Sept 2006.

[14] W. Li, T. Zhang, and K. Kuhnlenz, "A vision-guided autonomous quadrotor in an air-ground multi-robot system," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2980–2985.

[15] M. Peasgood, "Cooperative navigation for teams of mobile robots," Ph.D. dissertation, University of Waterloo, 2007.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100 –107, july 1968.

[17] M. Likhachev, G. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," *Advances in Neural Information Processing Systems (NIPS)*, vol. 16, 2003.

[18] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *IJRR*, vol. 18, no. 6, pp. 534–555, 1999. [Online]. Available: http://ijr.sagepub.com/content/18/6/534.abstract

[19] D. C. Conner, H. Choset, and A. A. Rizzi, "Integrating planning and control for single-bodied wheeled mobile robots," *Autonomous Robots*, vol. 30, no. 3, pp. 243–264, 2011.

[20] V. Kallem, A. Komoroski, and V. Kumar, "Sequential composition for navigating a nonholonomic cart in the presence of obstacles," *Robotics, IEEE Transactions on*, vol. 27, no. 6, pp. 1152–1159, Dec 2011.

[21] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *Robotics and Automation, 2007 IEEE International Conference on*, april 2007, pp. 3116 –3121.

[22] S. Thrun *et al.*, "Map learning and high-speed navigation in RHINO," in *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, D. Kortenkamp, R. Bonasso, and R. Murphy, Eds. Cambridge, MA: MIT Press, 1998.

[23] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," in *ICRA*, 2003, pp. 446–451.

[24] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," 1999, pp. 341–346.

[25] A. Botea, M. Müller, and J. Schaeffer, "Near Optimal Hierarchical Path-Finding," *Journal of Game Development*, vol. 1, no. 1, pp. 7–28, 2004.

[26] V. Bulitko, N. Sturtevant, J. Lu, and T. Yau, "Graph abstraction in real-time heuristic search," *Journal of Artificial Intelligence Research (JAIR)*, vol. 30, pp. 51–100, 2007.

[27] R. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct. 2006, pp. 3375–3380.

[28] K. Gochev, A. Safonova, and M. Likhachev, "Planning with adaptive dimensionality for mobile manipulation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.