# State Lattice with Controllers: Augmenting Lattice-Based Path Planning with Controller-Based Motion Primitives

Jonathan Butzke[‡], Krishna Sapkota[†], Kush Prasad[‡], Brian MacAllister[‡], Maxim Likhachev[‡]

***Abstract*— State lattice-based planning has been used in navigation for ground, water, aerial and space robots. State lattices are typically constructed of simple motion primitives connecting one state to another. There are situations where these metric motions may not be available, such as in GPS-denied areas. In many of these cases, however, the robot may have some additional sensing capability that is not being fully utilized by the planner. For example, if the robot has a camera it may be able to use simple visual servoing techniques to navigate through a GPS-denied region. Likewise, a LIDAR may allow the robot to skirt along an environmental feature even if there is not enough information to generate an accurate pose estimate. In this paper we present an expansion of the state lattice framework that allows us to incorporate controller-based motion primitives and external perceptual triggers directly into the planning process. We provide a formal description of our method of constructing the search graph in these cases as well as presenting real-world and simulated testing data showing the practical application of this approach.**

## I. Introduction

Lattice-based graphs are commonly used for robotic path planning. For example, aerial vehicles [1], automobiles [2], boats [3], and all-terrain vehicles [4] have all used state lattices for navigation. These graphs are constructed by applying a set of motion primitives to each state expanded during the search in order to generate valid successor states. By doing this, they generate edges in the search graph between the (possibly non-adjacent) discretized states which serve as the graph nodes. A graph search algorithm, such as $A^\star$ can act on this lattice to generate a trajectory consisting of a sequence of motion primitives between the start and goal state. The advantage of using a motion primitive vice simple 4- or 8-connected grids is that the motion primitives can more accurately model the kinematic constraints of the robot. For example, a car-like robot is unable to move directly sideways, but it can curve to the left and right as well as travel forward and in reverse. In this case a small set of motion primitives encoding those maneuvers is typically sufficient for planning motions in a plane.

In some environments, however, there may exist regions where the robot is unable to execute the typical motion primitives or they provide poor performance. For example,

Jonathan Butzke was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. This research was sponsored by the DARPA Computer Science Study Group (CSSG) grant D11AP00275.

[†] Jacobs University, Bremen, DE k.sapkota@jacobs-university.de

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

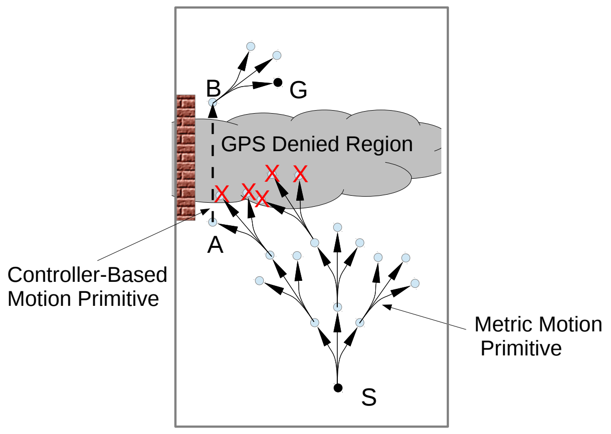


Fig. 1: Example of controller based motion primitive. From start state, S, we construct the motion primitive lattice. Metric motions that result in states inside the denied region are considered invalid, as the robot would become hopelessly lost if it entered this area. However, state A is a valid state and is sufficiently close to the wall on the left to allow a controller-based motion primitive (using a wall-following motion) to generate state B on the far side of the denied region. From B the standard metric motion primitives can generate a path to the goal, G.

Fig. 1 depicts an environment with a large GPS-denied region between the start and the goal. If the robot is relying purely on GPS data in order to navigate, then there is not a valid path to the goal state as it would become hopelessly lost when attempting to transit the GPS-denied region. On the other hand, if, in addition to our normal suite of sensors, we had the ability to follow a wall the robot could then use that ability to successfully cross the GPS-denied region. Similarly, as shown in Fig. 2, while motion primitives are well suited for the use in a parking lot, on the roads it is more common to use lane-following controllers.

More generally, a typical robot is usually equipped with a suite of controllers that utilize on-board sensors. For example, common controllers include visual servoing towards a landmark using a camera [5], direct-to-goal navigation using a GPS sensor [6], and range maintaining actions using a radar [7]. Controllers such as these operate well in real-world conditions due in part to their tight coupling of sensor information and actuation, specifically due to their ability to utilize the strengths of a given sensor system. For example, visual servoing works well in part due to the relatively large field-of-view and high angular accuracy of cameras.

In this paper, we extend the definition of state lattice to

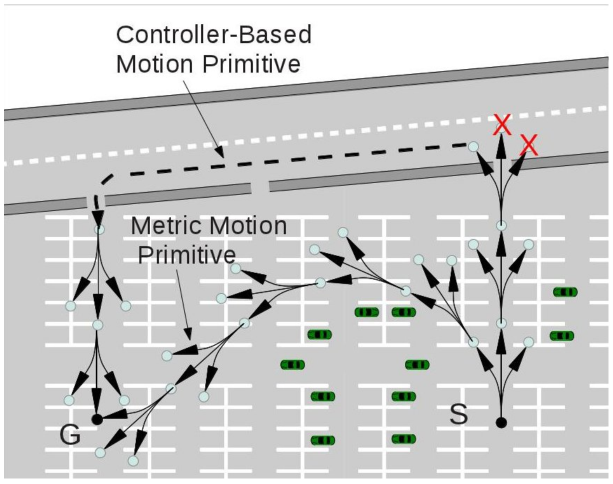


Fig. 2: The state lattice is sufficient to navigate in the parking lot area, however, once on the roadway, the lane-following controller is used.

support the availability of such controllers. Specifically, we introduce the notion of controller-based motion primitives that correspond to running available controllers terminated by a perceptual trigger. We refer to this data structure as a State Lattice with Controllers (SLC). Planning with SLC allows us to reason within a single planning process about the use of controllers and purely metric motions. In the following sections we present relevant related works, details of how we integrate controller-based motion primitives into the state lattice, and our real-world and simulation testing results.

## II. Related Work

Most of the modern approaches to global path planning for navigation represent the environment as a 2-D or 3-D geometric map and compute feasible paths on it. Typically these planners are either sampling-based approaches such as Probabilistic Road-maps (PRM) [8] and Rapidly-exploring Random Trees (RRT) [9][10][11], or approaches based on heuristic search (search-based approaches) such as A$^\star$ [12], ARA$^\star$ [13], and Field D$^*$ [14]. Our approach lies in the second class but differs from either class in that it adds reasoning over available controllers.

There also exist navigation strategies that directly encode sensing capabilities, such as the wall following assumption found in the simplistic, but effective, Bug2 and its related derivatives [15], and the "go to goal" sensing ability inherent in potential field approaches [16] and navigation functions [17]. Another example of planning integrated with control is the corner-localization scheme proposed by [18]. This approach uses only a minimal set of poor accuracy sensors but can still provably reach a desired goal location under certain conditions. However, these approaches typically do not provide path cost minimization, may get stuck in local minima, or are difficult to construct for arbitrary environments being traversed by arbitrary-shaped robots [19][20]. In contrast, lattice-based planning provides good cost minimization and deals well with arbitrarily complex environments.

One alternative to the navigation functions and potential field approaches is the sequential composition of controllers. By covering the valid states with a series of controllers that each moves the robot along to the next controller, this approach can alleviate the need to find a single globally attractive control law. Specific controllers are activated by a planner in order to approximate an arbitrary navigation function [21][22][23]. While these functions are easier to construct than a single navigation function covering the entire domain, there is still some computational overhead. More importantly, they cannot react to arbitrary perceptual triggers. In other words, they typically cannot come up with plans that say "follow the wall, until a doorway is detected on the left". By using the controller-based motion primitives in combination with perceptual triggers, we are able to construct these types of strategies.

Hybrid approaches seek to overcome the limitations that pure behavior-based systems exhibit. Sensor-based planning schemes using temporal logics to create an automaton or finite-state machine are capable of integrating sensor functionality into the planning scheme [24]. These planners also allow the robot to change its high-level behavior based on sensor inputs. However, these systems do not use any cost-minimization techniques to choose between controllers during the planning cycle. Instead they generate fixed controllers and select between them at run-time, typically shifting controllers when a higher level process detects a lack of progress towards the goal [25], and so do not incorporate the controllers into the planning process.

There have been approaches that considered the uncertainty of the robot as part of the planning problem. Of these, POMDP based approaches, even with modern approximate solvers, are slower than A$^\star$ and do not scale as well to large environments [26]. An alternative to the full POMDP method of handling uncertainty is by augmenting the state space with an uncertainty metric [27]. This approach uses detected landmarks to reduce uncertainty as a part of the planning process. In this way the planner can prune actions that raise the uncertainty above a threshold without incurring the overhead of solving a POMDP. Another example is the coastal navigation algorithm [28] which models the positional probability using a Gaussian. Our approach does not explicitly model the uncertainty in localization unlike these methods making our approach less general but significantly faster due to not increasing the problem dimensionality.

In summary, our approach combines the strengths of the search-based planning using state lattices with the advantages of pure-controller based approaches augmented by the ability to switch controllers based on perceptual triggers. Our approach retains the guarantees on path cost and consistency inherent in the search-based approaches as well as allowing, in a formal system, the ability to short-cut portions of the search graph based on available sensor-based controllers. Unlike the pure controller-based approaches, we are able to switch between controllers (including a metric motion controller) at any point based on the reception of a perceptual trigger.

## III. Planning with the State Lattice with Controllers

Typically, graph search-based planners decompose the environment into an $n$-dimensional grid where each cell becomes a node on the graph. They then construct edges based on proximity or available motion primitives. Given a graph, one can use any graph search algorithm, such as A$^\star$, to find a least-cost or good quality path in the graph. One method of constructing this graph is the use of the state lattice which is formed from applying, to each state, metric motion primitives; short kinematically (and dynamically, if the state space includes velocities) feasible motions going from a center of one cell to the center of another. Our approach builds on the state lattice construct by also adding edges that correspond to executing different controllers available on the robot. We refer to these as controller-based motion primitives. These primitives are terminated using perceptual triggers generated by the robot sensing system. The resulting construct, a state lattice augmented with controller-based motion primitives and perceptual triggers, we refer to as the State Lattice with Controllers (SLC).

### A. State Lattice

As a type of graph, a state lattice consists of a set of states, $\mathcal{S}$, connected by edges, $\mathcal{E}$ (see Fig. 1). To construct the set of states, each dimension in the planning domain is discretized into cells of finite size. Typically, metric dimensions are divided into small grids, for example, 10cm square regions for a small ground robot, while other dimensions, such as angular dimensions, are discretized into a small number of distinct values, e.g. $\{0°, 45°, \ldots, 315°\}$. The discretization size of the cells must balance two competing objectives. Having larger cells, and therefore fewer states, improves planning speed, while smaller cells can more accurately represent the environment. State lattices can have any number of dimensions and can include dimensions representing curvature, velocity, or other state variables as well. For navigation a typical choice is $(x, y, yaw)$ or $(x, y, yaw, vel)$ [2].

The edges of the state lattice are constructed by applying a set of pre-computed motion primitives to each state $s \in \mathcal{S}$ and then adding a directed edge from $s$ to the state $s' \in \mathcal{S}$ that the motion primitive ends at. The motion primitives can be generated in a number of different ways. For example, they can be generated through an offline optimization process [29] or by applying a feasible control signal for a short period of time [4]. The trajectory resulting from this control input is the motion primitive. Typically, motion primitives are picked to span the space well and are symmetric in the space of controls. For example, for a car-like robot, there are usually motion primitives for both left and right hand turns, and for forward and reverse directions (see Fig. 3). They also need to be generated in a way that their start and end points land on the center of cells. Typically, they are generated for every possible orientation of a vehicle and then during planning they only need to be translated to $(x, y)$ coordinates of a state for which the planner is getting successors. Since the individual motion primitives are feasible trajectories between states, the composition of multiple motion primitives between a series of states creates a feasible path for the robot. See [4] for an additional discussion on motion primitive construction.

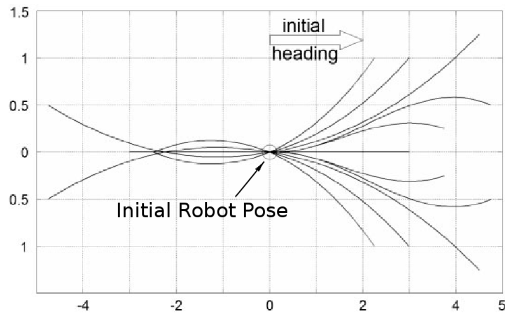


Fig. 3: An example of motion primitives for a car-like robot

### B. Adding Controller-Based Motion Primitives

Our contribution is in adding additional motion primitives that, instead of being generated a priori, are generated during the planning process based on controllers available to the robot. We begin with the introduction of three new functions. The first function,

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

provides us with a set of available controllers, $\mathscr{C}$, from the powerset of all controllers, $\mathcal{P}(\mathcal{C})$, available at a state $s \in \mathcal{S}$. $\mathcal{C}$ is the set of all controllers available to the robot. In other words, for any given state $s$, $\mathrm{C}(s)$ returns all of the controllers which can be executed at that state. Similarly, the second function,

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

provides 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, such as the ability to detect doors, intersections, etc. While metric motion primitives are of fixed length, controller-based motion primitives need to have a stopping condition. To this end we introduce a notion of a perceptual trigger. A controller-based motion primitive is a pair $\langle c, \tau \rangle$ representing the execution of a controller $c \in \mathcal{C}$ until either a perceptual trigger $\tau \in \mathcal{T}$ or an intrinsic trigger (explained below) is detected. Different controllers may have different triggers. For example, a controller for FollowLeftWall may have a trigger OpeningOnLeft, whereas that trigger may not be valid for a controller performing VisualServoToLandmark.[1]

[1]For the purposes of this paper we use relatively simple controllers in order to more clearly demonstrate the method of constructing the search graph. As such, the controllers and triggers listed are only a small subset of controllers and triggers that one can construct. For example, a wide collection of behavior based controllers are described in [30].

We classify triggers into two categories: *intrinsic* or *extrinsic*. Intrinsic triggers result from the natural completion of a controller. Intrinsic triggers are not selectable - they are always in effect. When following a wall, the robot must stop when the wall stops - there is no option to continue following a now non-existent wall. To account for these types of intrinsic triggers algorithmically, we do not individually include them in $\mathcal{T}$ and thus they are never returned as an element from $\mathrm{T}(c)$. Instead, a universal trigger, COMPLETION, is used to signify execution of a controller until its natural conclusion.[2] Compared to intrinsic triggers, extrinsic triggers are not directly related to the current controller, but instead result from some independent external signal. When following a wall, an extrinsic trigger could be sighting a door on the left or an opening on the right.

The third function, $\Phi$, generates a state where a controller-based motion primitive ends, given a start state, a controller, and a trigger.

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

So for a given state $s$, an allowable controller $c$ for that state, and an allowable trigger $\tau$ for that controller, function $\Phi$ simulates the execution of the controller $c$ starting at state $s$ until either trigger $\tau$ or an intrinsic trigger is detected (whichever comes first). The resulting state $s'$ is returned by the function. The function $\Phi$ accounts for feasibility in execution as part of the generation process. Unlike a typical state lattice graph, there is no guarantee that the final position of a robot after executing a controller will be in the center of a cell aligned with one of the discretized orientations. Given a maximum deviation from the cell center, $\vec{\delta}$, that the low-level path following controller (used for following metric motion primitives) can correct for, the function $\Phi$ can determine whether the final position and orientation of the robot is within these bounds as part of its feasibility verification.[3]

With this construction of the function $\Phi$ we can now modify how the state lattice is being constructed. Typically, when constructing a lattice, there needs to be a function that returns all of the successors of any state $s$ together with the corresponding edge costs. Let us call this function GETSUCC($s$).[4] During planning, a planner repeatedly calls this function to construct whatever portion of the graph it needs. Typically, the planner only calls this function for states that it expands. In this way, the full lattice is never explicitly constructed beforehand, but rather each edge and node is constructed as needed during the planning instance.

Algorithm 1 shows the GETSUCC($s$) function for the state lattice with controllers. This function first determines the successors by applying the set of metric motion primitives to

**Algorithm 1** $[S' \;\; Cost] = \mathrm{GetSucc}(\text{state } s)$

```
1: vector⟨state, cost⟩ $[S' \;\; Cost] = \bigcup\{[s(m) \;\; cost(s,m)]\}$
2:                                   $\forall m \in \{\text{Motion Primitives}\}$
3: $\mathscr{C} = \mathrm{C}(s)$
4: **for all** $c$ in $\mathscr{C}$ **do**
5:     $\mathscr{T} = \mathrm{T}(c)$
6:     **for all** $\tau$ in $\mathscr{T}$ **do**
7:         $[s' \;\; cost] = \Phi(s, c, \tau)$
8:         $[S' \;\; Cost] = [S' \;\; Cost] \cup [s' \;\; cost]$
9:     **end for**
10: **end for**
11: **return** $[S' \;\; Cost]$
```

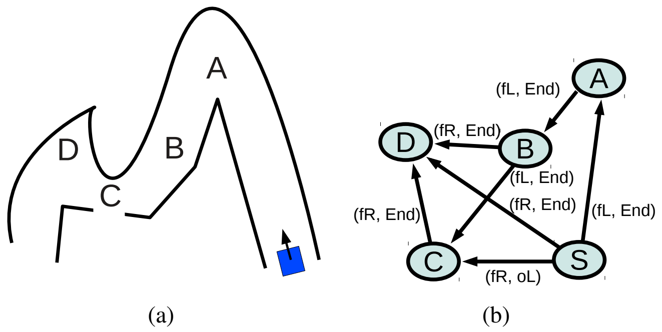


Fig. 4: (a) Environment and (b) segment of graph $\mathcal{G}$ based on the SLC with $\mathcal{C} = \{\text{FOLLOWLEFTWALL}(fL), \text{FOLLOWRIGHTWALL}(fR)\}$ and triggers $\mathcal{T} = \{\text{COMPLETION}(End), \text{OPENINGLEFT}(oL), \text{OPENINGRIGHT}(oR)\}$.

the current point (line 1). $s(m)$ is meant to represent a state resulting from applying metric motion primitive $m$ at state $s$. $cost(s, m)$ is meant to represent the cost of executing motion primitive $m$ at state $s$. It then iterates through the available controllers (line 3) and triggers (line 5) to generate the successor states and edge costs (line 7). For each controller-trigger pair we determine a valid successor state, if one exists, for that pairing applied at the start state.

To see an example of a lattice incorporating controller-based motion primitives, consider the environment shown in Fig. 4a. For the sake of illustration, it does not include metric motion primitives. Suppose we are given a set of controllers $\mathcal{C} = \{\text{FOLLOWLEFTWALL}, \text{FOLLOWRIGHTWALL}\}$ with an intrinsic trigger of COMPLETION corresponding to the end of the wall, and a set of extrinsic triggers $\mathcal{T} = \{\text{OPENINGLEFT}, \text{OPENINGRIGHT}\}$. 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 $\mathcal{G}$ shown in Fig. 4b. Likewise, with controller FOLLOWRIGHTWALL, $fR$, and trigger $End$, the edge goes from $S$ to $D$. However, if the trigger were OPENINGLEFT, $oL$, then the edge would have been from $S$ to $C$. Note, it is possible for multiple controller/trigger

[2]Some controllers have more than one intrinsic trigger.

[3]Another possibility is to set the discretization step size for each dimension smaller than the maximum allowable error for that dimension so that any point in the final cell is a valid start condition for the following motion primitive.

[4]Besides the GETSUCC($s$) function, the remainder of the planning process remains identical to the typical search-based planning algorithm implementation such as in [31].

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).

## *C. Cost Function*

As with any graph search-based planning, the edges in graph $\mathcal{G}$ need to have costs associated with them. These costs can represent an arbitrary cost function that includes such factors as distance, risk, closeness to obstacles and other factors. For controller-based motion primitives the cost can also incorporate the reliability of different controllers and the risk of relying on the detection of perceptual triggers.

## *D. The Output of the* SLC*-based Planner*

When a planner runs on a state lattice, the typical output is an ordered list of poses for the path-follower to execute. With the introduction of controller-based motion primitives the output of the planner becomes an ordered list of controllers together with the associated triggers. For the metric motion primitives, this will consist of the poses comprising the motion primitive for the path following controller to execute, just as in the state lattice. For the other primitives, it will specify which controller is to direct the actuators as well as the perceptual signal to relinquish control.

## *E. Theoretical Properties*

The State Lattice with Controllers is just a graph. Therefore, by running an optimal search such as A$^\star$ on it we can guarantee completeness and optimality with respect to the discretization and given controllers:

*If there exists a sequence of metric motion primitives intermixed with controllers $c_i \in \mathcal{C}$ terminated by triggers $\tau_i \in \mathcal{T}$ that move the robot from its starting position to the goal, then the planner will return a solution and it will be an optimal sequence of metric motion primitives and controllers/triggers with respect to the cost function used.*

## IV. Experiments

## *A. Vision-Based Micro Aerial Vehicle (MAV) Navigation*

To demonstrate the State Lattice with Controllers, we conducted flight testing using a low-cost AR.Drone 2.0 by Parrot. This quadcopter comes with 2 cameras, one forward and one downward looking, an ultrasonic and barometric altitude capability, a compass, on-board WiFi for interfacing and an ARM A8 micro-controller to perform low level controls. An off-board laptop running ROS-Fuerte performed the planning and control calculations as well as acting as the user interface to the robot. We added a 4.5mW laser line projector to provide depth information using the forward looking camera (Fig. 5).

The platform was incapable of path following due to its highly imprecise localization system. As a result, translational metric motion primitives had to be disabled. Instead, using the laser line and the forward camera, we implemented both a FollowThe[Left|Right]Wall controller and a

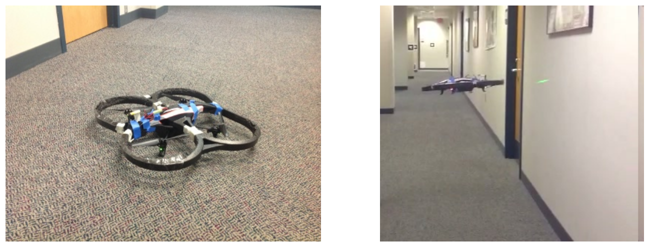


Fig. 5: The AR-Drone 2.0 with a laser line generator attached above its camera (a) and during flight (b).

GoToLandmark[5] visual servo controller in addition to metric turn-in-place motion primitives. For the Landmarks we implemented a detector to pick out the door jambs from each image. This detector provided the controller with the angular position, height and width of the door jambs located in the image. The controller then determined which jamb was the most appropriate to follow based on angle and range; turning as appropriate to consistently proceed to the desired jamb. Since the doors were indistinguishable from each other, the planner had the capability to execute the turn-in-place motion primitives in order to re-orient from its current heading by $\pm30°$ using the on-board compass, when searching for a door. This effectively allowed the controller to reliably transition to any adjacent door on the wall in front of it, or, by executing four turns ($\pm120°$), a door to the right or left on the opposite wall. These four motions worked for all tested door goal locations in our facility.

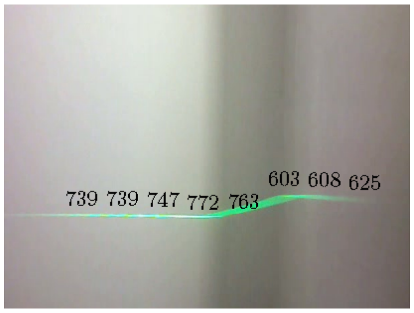


Fig. 6: Example of Laser Line. Range in millimeters to wall with pillar on right hand side.

The laser line allowed the AR.Drone to calculate the distance to nearby objects within the field of view of the camera via triangulation (Fig. 6). Due to the drop-off in horizontal brightness associated with using a cylindrical lens, the distance information was only reliable over the central two-thirds of the image, or approximately $30° - 40°$. Once calibrated, it proved fairly accurate and very consistent out to 1.5m.

The cost function used for these experiments was distance traveled plus a value proportional to the time to perform each maneuver. By making the cost proportional to time, turn-in-place motions that do not have any translational component

[5]referred to as GoToDoor in the remainder of this section

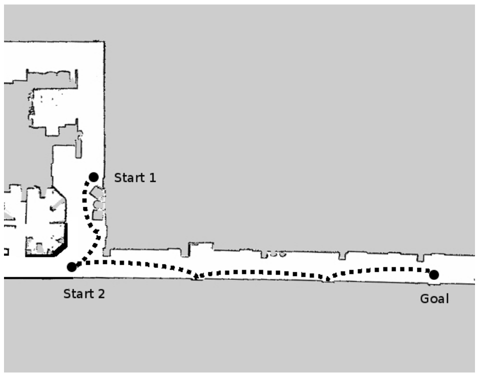


Fig. 7: Floor plan of map traversed with start and goal positions. The plan as a sequence of controllers is shown by a dashed line.

still had a positive cost.

In our experiments, we had the AR.Drone fly in a typical indoor environment consisting of a hallway with multiple doors along it (Fig. 7). The accompanying video shows one of the autonomous flights of the AR.Drone executing the calculated plan. The MAV navigates from one area of an indoor environment to a specific door at the far end without running SLAM or generating any form of metric map. Fig. 7 also shows the associated plan encoding the sequence of controllers terminated by the perceptual triggers, namely the detected doors.

Altogether there were 20 flights conducted which consisted of a series of GoToDoor controller phases intermixed with long FollowTheLeftWall controller execution phases, each terminated by the door detection trigger.

This experiment demonstrates the ability to use the SLC-based planner in situations where conventional planning is not feasible. Since the test MAV does not have the localization capability to translate reliably, typical search or sampling-based planners would not be able to construct a feasible trajectory. Pure controller-based approaches would be able to use the same controllers that were used here, but would not be able to shift based on the perceptual triggers received to terminate the execution of a controller prior to reaching its goal-set.

### B. Experimental Analysis of Runtime and Quality

*1) Setup:* While the first experiment showed that the SLC-based planner could successfully generate executable plans in domains where conventional planning could not, this series of tests was performed to demonstrate that for those domains where conventional methods do work SLC-based planning was also competitive in terms of planning time and path length.[6] The three planners we compared ours against were the lattice-based weighted A$^\star$ search from the SBPL package [31], and RRT-Connect and RRT$^*$ from the OMPL package [32]. We ran the four planners on seven maps; four indoor and three outdoor. The indoor maps, shown in Fig. 8, were generated from building floor-plans[†], converted laser scan data[†], and a custom drawn map. The outdoor maps, shown in Fig. 9 were a park[†], a set of buildings in a city, and a randomly generated map. None of the maps contained any regions that disallowed metric motions in order to not provide any advantage solely to the State Lattice with Controllers. On each map, each planner was tasked with finding solutions between 100 randomly selected pairs of points (each planner was given the same set of (start, goal) pairs). The planners planned in $\mathbb{R}^3 = \{x, y, \theta\}$. In addition, each planner was given approximately 10 seconds of total processing time, including any post-generation smoothing (smoothing was performed on all trajectories generated by the RRT$^*$, and RRT-Connect planners). Each pair of points was planned from scratch with the planner being restarted each time. While several planners can efficiently reuse data between plans, we only compared initial planning times since we did not allow for mid-execution replanning and we were interested in measuring the time before the robot could initially begin to execute a plan. For the SLC-based planning we used weighted-A$^\star$ to search the graph constructed from the State Lattice with Controllers. For the weighted A$^\star$ search running on both the normal lattice and the State Lattice with Controllers the heuristic inflation ($\epsilon$) weighting was fixed at 2.0. The heuristic was computed as the 2-D distance to the goal while accounting for obstacles. It was computed by running a single Dijkstra's search backwards from the goal, which was included in the planning times.

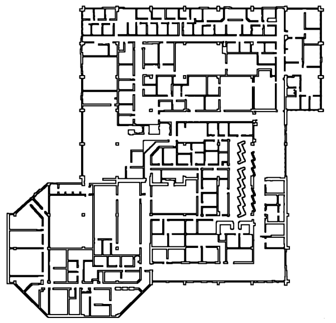
*2) Controller-based Motion Primitives:* For these experiments, in addition to WallFollowing and GoToLandmark controllers, we have also introduced a GoToGoal controller and a perceptual trigger based on the distance traveled. For the metric motion primitives, we used a left arc motion, a right arc motion and a straight motion, all in both forward and reverse direction, and a turn-in-place motion. The GoToGoal controller was only used on the outdoor maps as an example of a GPS capable sensor. This controller operated by having an attractive force directed towards the goal position, and repulsive forces from local obstacles in a manner similar to the potential fields approach.

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

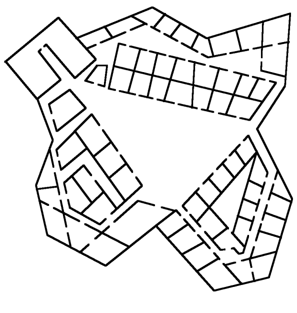
*3) Simulation Results:* The results from the 3500 planning attempts are shown in Table I. Since the results across all of the indoor maps and all of the outdoor maps were similar, they were each combined into a single entry. No one map was significantly better or worse than the others for any planner within each group. Planning time was the wall time for the planner to run from when it is given the $(start, goal)$ pair until a solution is returned. The average planning time ratio takes the planning time for a single $(start, goal)$ pair and divides it by the time it takes the SLC-based planner for the same $(start, goal)$ pair, then averages it across all runs from that map. Average Path Length Ratio is calculated in the

[6]All tests were performed on an 2.6 GHz i7 processor with 8 GB RAM running ROS-Electric on 64-bit Kubuntu 11.04.

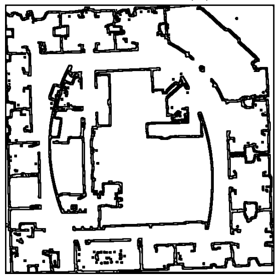
[†]Some of the maps were obtained from the Robotics Data Set Repository (Radish) [33]. Thanks go to Ashley Tews, Richard Vaughan, and Dieter Fox for providing this data.



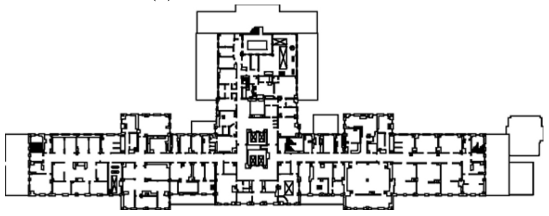
(a) Library floorplan $1500 \times 1600$



(b) Custom $1000 \times 1000$

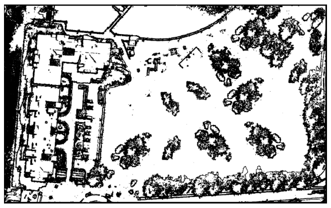


(c) Laser scan $600 \times 600$

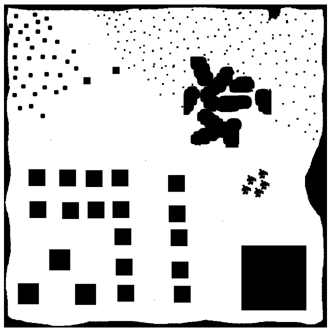


(d) Hospital floorplan $500 \times 1200$

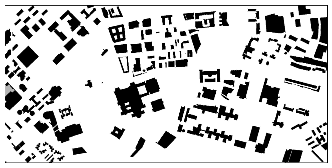
Fig. 8: Indoor Maps



(a) Park $1400 \times 2300$



(b) Custom $1000 \times 1000$



(c) City $900 \times 1800$

Fig. 9: Outdoor Maps

same way comparing the path length from each individual run to the path length returned by the SLC-based planner.

For some of the $(start, goal)$ pairs, not all planners were able to find a solution. This percentage is reported under the Plan Failure percentage column. The weighted A$^\star$ row refers to running weighted A$^\star$ on a normal lattice while SLC refers to the SLC-based planner.

As can be seen from this table, the planning time for the State Lattice with Controllers is comparable to the other planners and produces paths that are similar in length. The small increase in average path length compared to the A$^\star$ results is principally due to the State Lattice with Controllers making wider turns around corners and both planners having $\epsilon > 1$. While the A$^\star$ heuristic drives expansions right against the obstacles, the State Lattice with Controllers's frequent use of the wall following and goal directed controllers resulted in many paths maintaining a larger standoff from the wall on one or both sides of a corner. The results of both tests indicate that incorporating controller-based motion primitives has only a minimal impact on planning time.

## V. Conclusion

In this paper we introduced the concept of State Lattice with Controllers-based planning. The State Lattice with Controllers has the important feature that, by taking into account the specific sensing capabilities of the robot and any controllers already instantiated on the robot, it can use the corresponding additional motion primitives in instances when metric motion primitives are not available with only a minimal impact to planning times. Our primary contribution is this ability to seamlessly integrate controller-based motion primitives along with perceptual triggers into the existing state lattice planning framework due to our method of constructing the search graph. One of the main directions for our future work is to determine automatically which controllers or types of controllers are likely to be beneficial for a given environment. For example, FollowTheWall is of little use in a forest, and GoToGoal requires additional sensing hardware in a typical indoor environment, however as newer more complex controllers are developed simple intuition for which ones will work successfully may fail. A second direction for future work is to determine if there is a method to efficiently handle changes to the map in a similar way as D$^\star$ (and related) or other incremental planners. Testing on a wider variety of robots, such as those with Ackermann steering or other non-holonomic constraints, is the third direction for future work.

## References

[1] D. Thakur, M. Likhachev, J. Keller, V. Kumar, V. Dobrokhodov, K. Jones, J. Wurz, and I. Kaminer, "Planning for opportunistic surveillance with multiple robots," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov 2013, pp. 5750–5757.

TABLE I: Experimental Results.

| Map | Algorithm | Avg Plan Time (s) | Std Dev | Avg Plan Time Ratio | Std Dev | Avg Path Length Ratio | Std Dev | Planning Failure (%) |
|---|---|---|---|---|---|---|---|---|
| All Indoor | Weighted-A$^\star$ | 0.32 | 0.21 | 0.94 | 0.40 | 0.89 | 0.06 | 0 |
| | RRT* | 1.59 | 2.32 | 4.41 | 5.99 | 1.06 | 0.31 | 0 |
| | RRT-Connect | 0.56 | 0.72 | 1.69 | 1.23 | 1.09 | 0.29 | 0 |
| | SLC | 0.42 | 0.42 | 1 | - | 1 | - | 0 |
| All Outdoor | Weighted-A$^\star$ | 1.00 | 0.40 | 0.74 | 0.26 | 0.92 | 0.05 | 0 |
| | RRT* | 10.25 | 0.34 | 8.73 | 4.22 | 0.94 | 0.09 | 15.6 |
| | RRT-Connect | 0.33 | 0.64 | 0.22 | 0.38 | 1.14 | 0.26 | 27 |
| | SLC | 1.50 | 0.85 | 1 | - | 1 | - | 0 |

[2] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009. [Online]. Available: http://ijr.sagepub.com/content/28/8/933.abstract

[3] P. Svec, B. C. Shah, I. R. Bertaska, J. Alvarez, A. J. Sinisterra, K. von Ellenrieder, M. Dhanak, and S. K. Gupta, "Dynamics-aware target following for an autonomous surface vehicle operating under colregs in civilian traffic," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3871–3878.

[4] 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.

[5] G. J. Agin, *Real time control of a robot with a mobile camera*. SRI International, 1979.

[6] M. O'Connor, G. Elkaim, and B. Parkinson, "Kinematic gps for closed-loop control of farm and construction vehicles," in *PROCEEDINGS OF ION GPS*, vol. 8. Citeseer, 1995, pp. 1261–1268.

[7] S.-B. Choi and J. Hedrick, "Vehicle longitudinal control using an adaptive observer for automated highway systems," in *American Control Conference, Proceedings of the 1995*, vol. 5, Jun 1995, pp. 3106–3110 vol.5.

[8] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566 –580, aug 1996.

[9] J. J. Kuffner Jr. and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 2, 2000, pp. 995 –1001 vol.2.

[10] I. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *Robotics, IEEE Transactions on*, vol. 28, no. 1, pp. 116 –131, feb. 2012.

[11] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[12] 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.

[13] 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.

[14] D. Ferguson and A. T. Stentz, "Field d*: An interpolation-based path planner and replanner," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, October 2005.

[15] V. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *Automatic Control, IEEE Transactions on*, vol. 31, no. 11, pp. 1058 – 1063, nov 1986.

[16] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, mar 1985, pp. 500 – 505.

[17] C. I. Connolly, J. B. Burns, and R. Weiss, "Path planning using laplace's equation," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, may 1990, pp. 2102 –2106 vol.3.

[18] J. S. Lewis and J. M. O'Kane, "Planning for provably reliable navigation using an unreliable, nearly sensorless robot," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1339–1354, September 2013.

[19] A. A. Masoud, "Managing the dynamics of a harmonic potential field-guided robot in a cluttered environment," *Industrial Electronics, IEEE Transactions on*, vol. 56, no. 2, pp. 488 –496, feb. 2009.

[20] R. C. Arkin, "Motor schema based mobile robot navigation," *The International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989. [Online]. Available: http://ijr.sagepub.com/content/8/4/92.abstract

[21] 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

[22] 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.

[23] 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.

[24] 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.

[25] R. C. Arkin and T. Balch, "Aura: principles and practice in review," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 175–189, 1997. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/095281397147068

[26] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces," in *Proc. Robotics: Science and Systems*, vol. 62, 2008.

[27] J. P. Gonzalez and A. T. Stentz, "Planning with uncertainty in position using high-resolution maps," in *Robotics and Automation, 2007 IEEE International Conference on*, april 2007, pp. 1015 –1022.

[28] N. Roy and S. Thrun, "Coastal navigation with mobile robots," in *In Advances in Neural Processing Systems 12*, 1999, pp. 1043–1049.

[29] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *The International Journal of Robotics Research*, vol. 22, no. 7 - 8, pp. 583 – 601, July 2003.

[30] R. Arkin, *Behavior-based robotics*. MIT press, 1998.

[31] M. Likhachev, "The Search-Based Planning Library," 2012. [Online]. Available: http://sbpl.net

[32] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, 2012, to appear. [Online]. Available: http://ompl.kavrakilab.org

[33] A. Howard and N. Roy, "The robotics data set repository (radish)," 2003. [Online]. Available: http://radish.sourceforge.net/