Path Planning under Interface-Based Constraints for Assistive Robotics

Alexander Broad and **Brenna Argall**
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208
alex.broad@u.northwestern.edu, brenna.argall@northwestern.edu

Abstract

We present a heuristic-based search method for path planning in shared human-robot control scenarios in which the robot should adhere to specific motion constraints imposed by the human's control interface. This approach to path planning gives special consideration to kinematic and dynamic constraints introduced to reconcile discrepancies between the control space of the user and the control space of the robot. The resulting paths more closely mirror paths produced by users of the same interface; which is helpful, for example, when inferring human intent or for control sharing. Our first insight is to develop a hierarchical finite state machine describing the constrained state space, state transitions and associated costs. We then use this definition to embed the constraints of the interface into our heuristic planning algorithm, named C*, with simple modifications to the A*/D* family of graph search algorithms. This approach allows us to maintain powerful theoretical guarantees such as complexity and completeness. In this paper, we ground our augmented path planning algorithm with an implementation on a robotic wheelchair system and a Sip-and-Puff interface. We demonstrate that the new approach produces paths and control signals that more closely resemble user-generated data and can easily be incorporated into real hardware systems.

1 Introduction

Robot path planning involves developing algorithms that are designed to optimize specific aspects of a robot's motion which allow it to safely and efficiently navigate through static and dynamic environments. Modern approaches are predominantly developed with respect to fully autonomous systems that must navigate through unknown environments on their own. However, as we transition to a world of greater collaboration between humans and robots, it will be necessary to develop path planning approaches that are easily understood by, and even collaborative with, human users.

The concept of shared human-robot control is already an integral facet of many robotics applications including manufacturing (Parker Owan 2015), assistance and rehabilitation (Argall 2014), prostheses (Cipriani et al. 2008), search and rescue, and extraterrestrial robotics (Trautman 2015). Collaboration in these domains spans from *task-level* shared control (Sa and Corke 2014), where the user must provide a set of reference points for the autonomous system to track, to *signal-level* shared control (Trautman 2015), where the user and autonomy each provide a low-level signal which are then synthesized into a final command that is sent to the robotic system. The selection of *which* level of shared control is based on how directly the user wants to control the robotic system and how well the user's control signal matches the state space of the robotic system.

Consider a differential drive robot. This system can be controlled with a two dimensional signal consisting of linear and angular velocities. As common user interfaces, such as a 2-axis joystick, are able to span the full control space of this robot, it is possible to use signal-level control synthesis techniques, such as linear blending (Argall 2014), to produce a final command. That is because the control signal produced by the user and the autonomy both exist in the same state space (linear and angular velocities), and presumably use similar optimization metrics (e.g. safely reach the desired position via the shortest route).

However, signal-level control synthesis techniques like linear blending are only valid when the user and autonomy are planning in the *same state space* with the same high level considerations, a premise that is often invalidated by a mismatch between the user's control interface and the state space of the robotic system. This inconsistency can be a result of either a restriction in dimensionality imposed by the user interface, or due to the inherently high-dimensional state space of the robotic system.

As an example of interface restrictions, consider a power-wheelchair user with limited motor capabilities who is unable to operate a standard 2-axis joystick and is limited to options like a Sip-and-Puff interface. Such an interface can only provide a one dimensional control signal, which means that the user must provide separate signals to control the translational and angular movement. As a result of these restrictions, the paths that user's generate with a limited control interfaces tend to exhibit distinct properties when compared to users of higher-dimensional control interfaces as the interface alters how difficult it is to perform certain actions (e.g. making tight turns while moving forward).

As an example of an inherently high-dimensional task

space, consider a user trying to control a six degree-of-freedom (DoF) robotic manipulator using a 2-axis joystick. Regardless of whether the user is operating within the task space (position of the end-effector) or joint space (position of each joint), the system is highly underactuated. The joystick can only produce a 2D signal, while the task space of the robot is the 6D position and orientation of the end-effector. A user would need to switch between control modes to select which subset of position and/or orientation dimensions they would like to move in at a given time step. It should be clear that the paths generated would likely differ greatly from those of a standard autonomous planning approach—a planar optimizing path length would generate the shortest path through all six spatial dimensions, while the 2-axis joystick user is only able to generate a sequence of path segments that are shortest within subsets of the control dimensions (of the active mode).

In both examples above, we run into the same issue – namely, the user interface is not able to produce a control signal in the same space as the autonomous system, and as such, we expect to see paths planned with characteristically different properties. Importantly, this also means that signal-level control sharing techniques like linear blending are non-intuitive and can lead to dangerous behaviors. Using the robotic manipulator as an example, we can see that blending signal-level commands from a user of a 2-axis joystick with those from an autonomous system that does not consider the constraints of a user interface may result in the robot moving through a part of space that is far away from where either the user or autonomy desired. Therefore, if we want to use autonomous planning algorithms in a shared control paradigm, the interplay between human and robot *must* be considered in the planning process.

The contribution of this paper is to enforce the same constraints that control interfaces place on users' control signals on our planning algorithm. The resulting paths can be used to generate autonomous control signals that are appropriate to use in a signal-level shared control paradigm. We demonstrate that with simple modifications to standard heuristic-based search algorithms (e.g. A* (Hart, Nilsson, and Raphael 1968) and D* (Stentz 1994)) we are able to embed the specific constraints of a limited control interface into a planning algorithm that we name C*. We then ground this work with an implementation in the domain of assistive and rehabilitation robotics, where shared control is particularly important as continuing to use any residual motor capabilities can be paramount in the patient's rehabilitative process.

We begin by discussing some related work in Section 2, followed by the definition of our problem statement and an explanation of our algorithm in Section 3. We ground our work in an implementation in Section 4 and analyze the resulting paths as they compare to standard grid search algorithms, user generated paths and user generated signals in Section 5. Lastly, we conclude in Section 6.

2 Related Work

Our work builds on a variety of related research areas including grid– and sampling–based path planning methods, hierarchical state machines and shared human-robot control.

Grid-based search algorithms have progressed from the now standard A* (Hart, Nilsson, and Raphael 1968) to be able to handle dynamic environments through incremental search methods such as D* (Stentz 1994) and D* Lite (Koenig and Likhachev 2005). Additionally, researchers have developed methods for planning paths that can traverse grids at any-angle such as Field D* (Ferguson and Stentz 2007) and Theta* (Nash et al. 2010). These approaches are extremely successful, however they do not consider the constraints of a user interface, as they are mainly developed for use with fully autonomous systems.

Sampling-based path planning methods, such as Rapidly Exploring Random Trees (RRTs) (LaValle 1998) and Probabilistic Roadmaps (PRMs) (Hsu et al. 1998), are commonly used in both large and kinematically constrained state spaces. These methods are empirically faster than grid-based search methods as they randomly subsample the state, or configuration, space and intelligently remove invalid paths. However, sampling-based approaches do not guarantee optimality nor do they deterministically produce the same path from a given start position. As path comprehension is one of the main aspects of this work, we view the stochasticity of sampling-based planners to be undesirable.

There is also related work in hierarchical graph-based planning. These approaches tend to focus on programatically developing a hierarchical abstraction of the two-dimensional planning space (Holte et al. 1996), (Botea, Müller, and Schaeffer 2004). In our work, we use apriori knowledge of a control interface to build a hierarchical definition of the user's state and action space, thereby embedding the constraints into the planning problem.

Lastly, there is related work in the shared-control and intent prediction domains. Namely there have been previous studies in task-level shared control where the user provides high-level commands and the system provides low-level behaviors to achieve those commands (Yanco 2000), (Philips et al. 2007) and (Wang and Liu 2014). (Demeester et al. 2008) also uses a task-level shared control approach and develops a POMDP to perform intent prediction. There is also previous research in signal-level shared control such as in (Parikh et al. 2004), (Argall 2014). In the first instance, the author's allow the system to deliberate between a hierarchical set of commands, while in the second, the author's use machine learning to blend user and robot commands. However, as stated earlier, to our knowledge, all previous work is agnostic to constraints of a user interface.

3 Problem Statement

We consider the problem of robot path planning in shared control scenarios. We formalize the problem to explicitly account for the kinematic constraints and associated state-action costs that control interfaces place on a joint human-robot system. Our approach ensures that the user and autonomous agent develop plans in the same state space. This allows us to more appropriately analyze the resulting paths by comparing both macro-level characteristics of the path, such as the number of required turns, and micro-level characteristics, such as how well the resulting control signals align with user inputs. By developing a planning algorithm

that accounts for the constraints of a user interface, we can be confident that the resulting control signal can be used in a shared control paradigm.

3.1 Algorithm Overview

We introduce a constrained grid-based search algorithm in the A*/D* framework that we name C*. At a high-level, our approach is twofold. First, we explicitly enforce the same kinematic constraints of the user interface in our search algorithm. Second, we directly model the ease and/or difficulty of each viable action given the user interface. This approach requires altering the system's state space to embed the currently active actions, and altering the costs associated with particular actions in our search method. To account for state-action costs without significantly increasing the search space or number of parameters, we develop a hierarchical model to better capture the important aspects of an interface-robot system. The desire is to produce paths that display attributes similar to those of paths generated by human users, whose commands are filtered through the control interface.

3.2 Hierarchical Statecharts

To incorporate the constraints enforced by a limited control interface into our path planning algorithm, we must first codify the constraints in a way that both restricts the possible state transitions and specifies the modified set of path characteristics that we would like to optimize.

We begin by defining a finite state machine (FSM) that models the set of states and transitions available to the user given their chosen control interface. The goal is to then incorporate the FSM into our search algorithm to exactly represent the user's control options given the limited interface.

However, for a given control interface, there may be many ways in which one could define the state machine. In fact, to help account for the inherent restrictions of a limited interface, it is often necessary to develop a control logic that produces different results from the same input conditioned on the current state of the system. This can lead to the state-explosion problem (Harel 1987), which in turn can significantly expand the size of the state space that we must search over in our planning problem. As our goal is to incorporate the constraints of the interface into our planning algorithm, it is particularly important that we design a representation that is both complete and economical in its description. To this end, we describe our state machine in the language of Harel statecharts (Harel 1987).

Harel statecharts, or more generally, hierarchically nested finite state machines, provide a level of abstraction necessary for maintaining a manageable set of states by allowing one to define *superstates*, or *composite states*, comprised of sets of low-level states. Using this technique, we are able to reduce the search space of our planning algorithm.

Our approach is to develop a high-level representation by defining a set of *orthogonal regions*, Q, that represent high-level states, and actions, T, that represent transition between high-level states. Using this definition, we group low-level states, S, together into a composite state based on their respective low-level transition functions, A, as defined by the user interface. The low-level states and actions are the traditional definitions of states and actions in the configuration or task space of the robot. The high-level states and actions are hierarchical abstractions of those same properties. The purpose of this new definition is to marginalize internal transitions between low-level states while developing a language by which we can explicitly interact with external transitions.

With this new model, we can then modify our planning algorithm's cost structure to place special importance on the external transitions by applying different costs to high-level actions, T, as compared to low-level actions, A.

While it is possible to achieve similar results with a higher-dimensional A* variant, it would require both searching over a *significantly larger* state space as well as developing a *complex cost structure* that appropriately weights each state-transition. Finding the appropriate weighting parameters can be a difficult and time consuming task. By contrast, with our approach, it is possible to design a very compact description of the system by using a hierarchical modeling technique and restricting state transitions. To analyze the complexity of creating a cost structure with a higher-dimensional A* variant, consider that N actions requires $N \cdot N$ cost assignments (since self-transitions between low-level states are possible). By contrast under C*, what needs to be specified are the transition costs between M Harel states ($(M-1) \cdot (M-1)$ assignments) and between low-level states within a given Harel state (at fewest 1 and at most $N - M + 1$ assignments, for each of N low-level states). By observing that $M \leq N - 1$ and $2 \leq M$, we can prove that the number of elements to specify for a C* cost function (at fewest $N \cdot 1 + (M-1) \cdot (M-1)$ and at most $N \cdot (N - M + 1) + (M-1) \cdot (M-1)$) is always less than what needs to be specified without the HFSM structure.

To incorporate the statechart description, we expand the state space to include the composite state, Q. This expanded state space enables us to both appropriately constrain the possible set of motions as well as develop plans in the same space as the user. Additionally, the magnitude of this extra dimension is relatively small as it is based on the number of high-level composite states. This state space definition can be used with any control interface as the description only relies on the inherent task space of the robot and a generic definition of the composite states. Using our examples from the introduction, this would mean that we expand the state space of a differential drive robot from (x, y, θ) to (x, y, θ, q) and the state space of a robotic manipulator from (x, y, z, r, p, y) to (x, y, z, r, p, y, q). That is, in both cases we expand the initial state space (two transitional and one rotational dimensions, and three transitional and three rotational dimensions, respectively) to include the system's current composite state.

3.3 Constrained Path Planning

In addition to altering the state space, we need to incorporate the constrained state transitions and update the associated costs in our algorithm. We achieve this by augmenting the standard A* family of algorithms in two main areas. These are the functions CONSTRAINEDNEIGHBORS (lines 18-21) and UPDATEG (lines 22-29) in the C* pseudocode.

First, we alter how unexplored nodes are generated and

added to the *open set*. We limit these states based on kinematic– and interface–based constraints (line 18-21). Specifically, we constrain the set of all possible neighbors to the subset that are valid under the constraints of the composite state q.

The second way in which we modify the standard A* algorithm is that we augment the associated state-action costs to account for internal and external state transitions. The standard A* approach expands vertices in the *open set* in an order defined by a vertex's f-value:

$$f(s) = g(s) + h(s, s_{goal})$$

where $g(s)$ defines the true cost of reaching vertex s from the initial position and $h(s, s_{goal})$ is a chosen heuristic that estimates the cost from the current vertex to the goal.

We augment the standard g-value to include an extra cost when transitioning composite states. This concept is illustrated in the function defined at line 22 in the C* pseudocode. In this definition $\alpha_I < \alpha_E$ encodes the fact that it is preferential to perform transitions *within* composite states instead of transitions *between* composite states. Note that it is only necessary to define *one* value for α_I and *one* value for α_E (s.t. $\alpha_I < \alpha_E$) which, with the Harel states definition, now encodes the difficulty and ease of transitioning between *all* combinations of high– and low–level states.

Algorithm C*

```
1: Given: s, s_goal, open set, closed set
2: while s ≠ s_goal do
3:     s ← arg min_{s_o ∈ openset} f(s_o)
4:     open set ← open set \ s
5:     closed set ← closed set ∪ s
6:     for s' ∈ ConstrainedNeighbors(s) do
7:         if s' ∉ closed set then
8:             g_tentative ← UpdateG(s, s')
9:             if s' ∈ open set || g_tentative < g(s') then
10:                assign g(s') ← g_tentative
11:                if s' ∉ open set then
12:                    open set ← open set ∪ s'
13:                end if
14:            end if
15:        end if
16:    end for
17: end while
18: function ConstrainedNeighbors(s)
19:    q ← extract q from s
20:    Return {AllNeighbors(s) | q}
21: end function
22: function UpdateG(s, s')
23:    q, q' ← extract q, q' from s, s'
24:    if q = q' then
25:        Return g(s') + α_I
26:    else
27:        Return g(s') + α_E
28:    end if
29: end function
```

Any heuristic-based search algorithm in the A*/D* family can easily be extended to incorporate these modifications–it requires only altering the state space, how vertices are added to the open set and how the g-value is updated.

Figure 1: Sip-and-Puff interface.

This approach, along with a consistent (i.e. monotonically non-decreasing) heuristic, allows us to perform a sparse search over the state space resulting in an efficient algorithm (Koenig, Likhachev, and Furcy 2004). The specified modifications also do not affect the algorithm's completeness. In fact, with the simple requirement of an admissible heuristic (Pearl 1984), C* retains the guaranteed optimality of other heuristic-based search algorithms.

4 Implementation Details

We ground our algorithm with an example interface and implementation from the domain of assistive and rehabilitation robotics. The goal in this domain is to develop technologies, such as autonomous wheelchairs, that are operated by, and assist, people with varying degrees of physical disability. When the disability negatively affects a person's motor control, it is often the case that standard interfaces, such as a 2-axis joystick, are insufficient to control the platform. In this case, the user must rely on alternative interfaces that better fit his or her physical capabilities.

For example, users with high spinal cord injuries often have severely limited control of their arms and hands, but maintain control over their diaphragm and breathing. In this case, a Sip-and-Puff interface (Figure 1) is a viable option. Using a Sip-and-Puff, a user can control their wheelchair by providing one of four discrete signals: a hard puff, a soft puff, a hard sip or a soft sip. In general, these signals are mapped to the following control logic :

- Hard Puff → Move Forward (latch)
- Hard Sip → Move Backward (latch)
- Soft Puff → Rotate Counterclockwise
- Soft Sip → Rotate Clockwise

Using a Sip-and-Puff, the hard puff and hard sip commands are *latching*–this means that once they are activated, they remain so until the user explicitly unlatches. It is possible, but requires a decent level of expertise, to execute turning schemes while latched, effectively providing a 2D signal.

The latching behavior illustrates one of the control challenges when using the Sip-and-Puff interface. Because of this latching behavior, a user can only transition from moving forwards to moving backwards by first sending a signal to stop their forward motion and then sending a second signal to continue backwards. For this reason, it can be difficult to transition between forward and backward motion.

The limited input space means that actions that would generally be possible using a single joystick command may

take multiple steps to achieve using a Sip-and-Puff. In addition to transitioning between forwards and backwards movement, it can be difficult to perform actions such as turning in tight spaces as the Sip-and-Puff interface provides non-proportional control of the wheelchair's linear and angular speed. This means that the force with which a user sips or puffs does not affect the magnitude of the speed signal. For this reason, it is often necessary for a user to come to a complete stop before making a tight turn.

To model the Sip-and-Puff interface constraints within our algorithm, we begin by developing a finite state machine for the Sip-and-Puff interface, which can be seen in Figure 2.

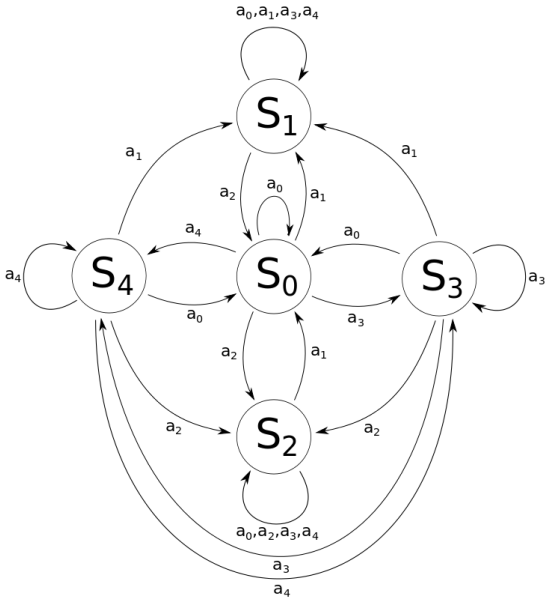

Figure 2: State diagram of the Sip-and-Puff interface.

In this state diagram, we have five low-level states and five low-level actions. They are defined in Table 1. [1]

states		actions	
s_0	not moving	a_0	null input
s_1	moving forwards	a_1	hard puff
s_2	moving backwards	a_2	hard sip
s_3	turning right	a_3	soft sip
s_4	turning left	a_4	soft puff

Table 1: Sip-and-Puff state and action definitions.

We recast the state diagram as a Harel statechart. Our statechart for the Sip-and-Puff interface is shown in Figure 3.

This representation allows us to explicitly model the difficulty of performing specific actions with a limited user interface, while minimizing the effect it has on the state space.

[1] Another valid state decomposition for this interface could include separate states for when the robot is turning left while moving forward, turning right while moving forward, turning left while moving backward, and turning right while moving backward.

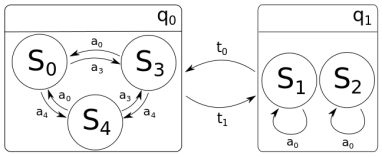

Figure 3: Harel statechart for Sip-and-Puff interface.

We develop this representation by defining a set of *orthogonal regions*, q_0 and q_1, that represent high-level states, and actions, t_0 and t_1, that transition between them. In this definition we group states s_0, s_3 and s_4 together into composite state q_0, and states s_1 and s_2 together into composite state q_1. We choose this grouping to separate the low-level latching switch states from their momentary switch counterparts. This allows us to marginalize internal transitions such as a_0, a_3, and a_4 while focusing on external transitions such as t_0 and t_1. Note that there are no direct transitions between low-level states s_1 and s_2 in either the FSM or Harel statechart definition as the Sip-and-Puff interface restricts that behavior. Using this hierarchical modeling technique and restricting state transitions, we are able to design a very compact description of the system.

We then embed this state definition in our algorithm by augmenting the state space of the grid-based search algorithm to include the composite state as such:

$$S = (x;\ \ y;\ \ \theta;\ \ q)$$

In the standard 2D A* approach, up to 8 vertices are expanded at each iteration. In our approach, this number is reduced based on kinematic– and interface–based constraints. Using the Sip-and-Puff we expand three vertices if the system is in composite state q_0 and four if it is in composite state q_1. That is, from state q_0 the system can rotate clockwise, counterclockwise, or transition to state q_1, and from state q_1, the system can translate linearly along its current heading, linearly while rotating clockwise or counterclockwise, or transition to state q_0.

Lastly, we set $\alpha_E = 2 * \alpha_I$ to impart a preference for the easier transitions. This parameter can be tuned to better align with user generated data in a given use case, however, we found this was a good value for our experiments.

As a final point, it is worthwhile to note that the HFSM formulation generalizes to other control interfaces where constraints can be encoded as between-harel-state transitions. While defining an HFSM that effectively captures these constraints will be interface-specific, we can identify a few broad categories, namely: (1) When there is a mismatch in the dimensionality of the interface (lower) and the control space of the robot (higher). These interfaces require methods (e.g. buttons) of switching between control mappings (i.e. to different subspaces of the robot control space). Actions that require a switch are inherently more expensive than actions that do not require a switch. The Harel statechart can be built along these lines, grouping actions that do not require a switch into a single harel state. (2) Where taking the same action from different low-level states incurs different costs. For example, some actions are inherently riskier to perform

at high speeds (e.g. robotic systems with complex dynamics like quadcopters).

5 Analysis

We validate our approach with a number of different quantitative and qualitative techniques. We perform a multi-fold analysis comparing the results of C* to multiple A* implementations as well as user-generated paths.

Our entire software suite runs in the Robot Operating System (ROS) framework (Quigley et al. 2009), including a simulated version of our semi-autonomous power wheelchair that is faithful in dimension and mechanics to the real system. The simulated system is missing only the onboard computational system that sits atop the base of the wheelchair (Figure 4). We plan all paths in the costmap provided by the ROS Navigation Stack, as such, it is trivial to transition between running the system on the real hardware and in simulation. We therefore demonstrate the efficacy of our approach using a range of simulated environments developed in Gazebo, which allows us to exactly reproduce the conditions of a particular experiment for multiple users. It also allows us to design environments with varying degrees of maneuverability to best examine different classes of realistic scenarios.

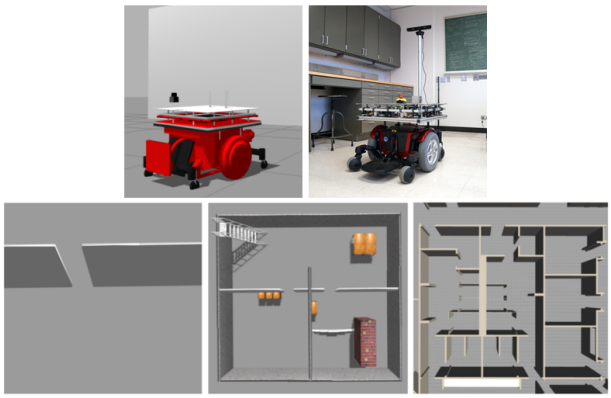

Figure 4: Top: Power wheelchair robot. Simulation (left), real hardware (right). Bottom: Simulation environments.

All simulated environments can be seen in Figure 4. The first environment contains only a single wall and doorway. The second environment is slightly more complex, has multiple rooms, two doorways and a tight corridor. The third, and last, environment is the most intricate. It is a simulated replica of the floor plan of the Willow Garage office space. It contains numerous rooms, hallways and doorways.

5.1 Path Analysis

We begin our analyses by directly comparing paths generated with our constrained approach, C*, with a standard 2D A* (x,y) implementation and a 3-dimensional A* (x,y,θ) variant. The 3D A* cost function consists of the same costs on state transitions in (x,y) as is used in the 2D A* cost function (Euclidean distance), and the same cost on state transitions in θ as is used in the C* cost function. We choose a representative sampling of start and goal locations. In particular, we chose start and goal locations that exemplify the differences between the standard 2D A* approach and C*. The resulting paths must navigate through doorways, tight corners and otherwise restricted environments. In these cluttered areas, we find the greatest juxtaposition between algorithms. This is due to a trade-off between optimizing the specified path solely for distance versus a balance of distance and ease of achieving the path under the constraints of the user interface. Using the Sip-and-Puff, moving in a sequence of straight lines, even if they do not adhere to the shortest path, is easier to do then making tight turns. Table 2 presents pertinent path information for these three approaches on six different sets of start and goal locations.

In particular, we observe that C* produces paths that are on average the same length (only 0.72% longer) as those produced by the standard 2D A* implementation, but require an average of 61% fewer turns. In comparison to 3D A*, we find C* to again produce paths with equivalent length, but require an average of 21% fewer turns.

In less restricted environments, all approaches produce similar, if not identical results. For example, we observe in Figure 5 that the majority of differences in the 2D A* and C* generated paths are nearby changes in heading. Along straight paths, both approaches find that the optimal solution is to take the most direct route, as we would expect.

5.2 Experimental Path Analysis

We additionally compare our constrained planning approach and the 2D A* implementation to user-generated paths. To do so, we gathered pilot data from three lab members.

Each trial consisted of a user navigating the simulated wheelchair between a specific start and goal location. Prior to the trials, users were given 10 minutes of free time to explore the state space and improve their understanding of the Sip-and-Puff interface. Odometery data was recorded as the user navigated the robot through the environment. Figure 5 provides a visual comparison of the user paths, those generated using C* and the standard 2D A* implementation.

We quantify distinctions between our approach and A* by computing the difference in average density between the two algorithms and the user-generated paths. The density defined by two paths is the area of the polygon specified by joining the paths at their respective ends. Unlike the Frechet distance (Alt and Godau 1995) or dynamic time warping (Sakoe and Chiba 1978), the average density approach does not account for the location or ordering of the points along the curves. Instead, density computations consider only the area of the polygon which removes any importance placed on reaching a particular point along the path. This is a more appropriate

Environment	1	2	3
Mean Density Ratio (C*/A*)	0.90	0.81	0.78

Table 3: Mean density ratio breakdown by environment.The *density ratio* is computed as the C* density divided by the A* density. A lower ratio is correlated with a path that more closely aligns with the user generated paths.

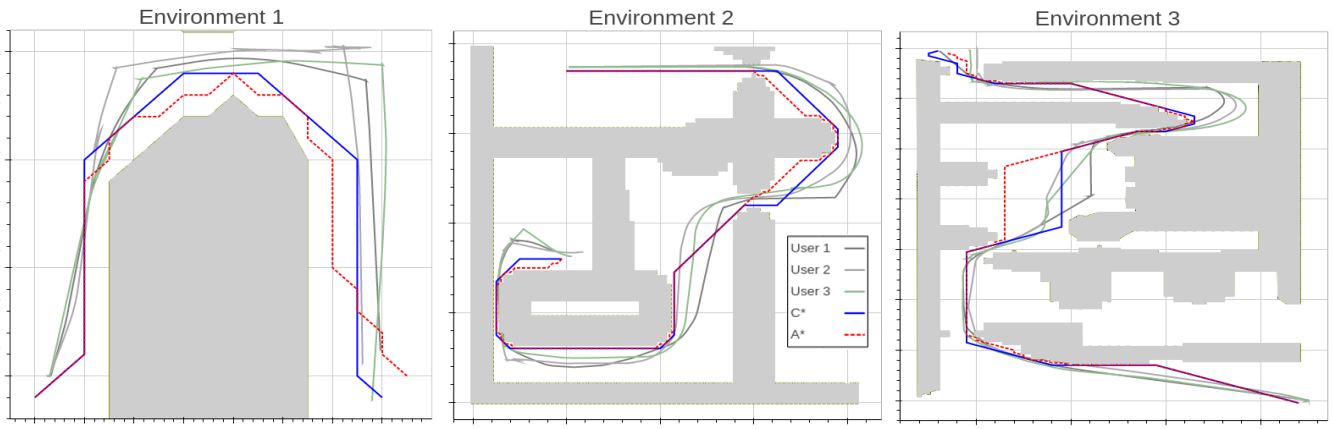

Figure 5: Visual comparison of example paths generated by users, C* and A*.

	A*		3D A*		C*	
Trial	Path Length	# Turns	Path Length	# Turns	Path Length	# Turns
1	3.89	12	3.99	6	3.95	6
2	5.14	12	5.14	7	5.14	5
3	3.70	19	3.89	7	3.76	7
4	19.65	43	19.65	19	19.65	13
5	18.71	39	18.89	24	18.93	17
6	18.44	32	18.44	16	18.44	10

Table 2: Path analysis. Path length reported in meters.

measure for our purposes as the planning algorithms do not take speed or acceleration into account. Example polygons created in this manner can be seen in Figure 6.

Using this analysis we find that the density of the difference between user paths and those generated by C* is 84.7% as large as the density of the difference between user paths and those generated by A*. This shows that user paths more closely align with paths generated by C* than those generated by A*. Additionally, when compared with user paths, the C* approach produced smaller densities than A* in 88.9% of the trials.

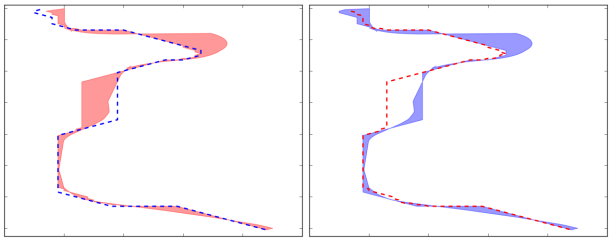

Figure 6: Example polygon comparisons between users, 2D A* and C*. Left: In red is the polygon defined by a user generated path and A*, the blue dotted line is the C* path. Right: In blue is the polygon defined by a user generated path and C*, the red dotted line is the A* path. We see that the density of the left–hand polygon is larger, meaning that there is greater difference between the A* and user paths.

An environment-based breakdown of the relative densities can be seen in Table 3. This table helps elucidate the point that C* is particularly useful in larger, more complex environments such as environments 2 and 3. In these environments, the user is more likely to need to perform very precise maneuvers around objects and tight corners. As such, users account more for ease of achievement rather than directly optimizing their path for distance.

5.3 Experimental Signal Analysis

Lastly, we compare how well the autonomous control signals match those generated in the user trials. We perform this analysis by comparing the vector of the user's control signal with that of the vector between the current position and the first point along the planned A* and C* paths. We compute these values at equally spaced intervals along the user path and analyze the results by calculating three metrics. First, we calculate how often the heading of the signal produced by the path planning algorithm exactly aligns with the signal provided by the user–we call this metric *agreement*. Second we compute the average *divergence* between the two signals, which refers to the angle (in radians) between the heading of the path planning algorithms and the user signal over the course of the entire path. Lastly, we present the average *divergence* subject to the condition that the user and

	Trial						Mean	Std. Err.
	1	2	3	4	5	6		
Avg. Agreement (A*)	52.06 %	32.00 %	41.39 %	44.74 %	47.45 %	55.52 %	45.53 %	3.40
Avg. Agreement (C*)	59.26 %	47.12 %	67.14 %	69.98 %	72.29 %	73.63 %	64.90 %	4.13
Avg. Divergence (Total A*)	0.65 rad	0.70 rad	0.51 rad	0.96 rad	0.93 rad	0.49 rad	0.71 rad	0.08
Avg. Divergence (Total C*)	0.40 rad	0.48 rad	0.35 rad	0.46 rad	0.55 rad	0.30 rad	0.43 rad	0.15
Avg. Divergence (Disagreement A*)	1.36 rad	1.04 rad	0.85 rad	1.66 rad	1.80 rad	1.10 rad	1.30 rad	0.04
Avg. Divergence (Disagreement C*)	0.99 rad	0.91 rad	1.06 rad	1.50 rad	2.00 rad	1.15 rad	1.27 rad	0.17

Table 4: Signal Analysis. Each row represents the average value of the designated metric over all users. The last two columns represents the mean and standard error of each metric over all users over all trials.

path planning algorithm are not in *agreement*. We perform this analysis for each user in our study and present the average results in Table 4.

Using these metrics, we find that our C* algorithm produces signals that are in agreement with those provided by users an average 42.5% more often than those produced by standard A*. Over the course of an entire path, we find the headings produced by C* are an average of 60% more aligned with the user produced headings. Lastly, we find, that even when the user and path planning algorithm disagree, C* is no worse than A* at predicting the user-generated signal.

5.4 Discussion

The results of our hierarchical constrained planning algorithm are paths that clearly respect the cost structure and limitations of the control interface. In particular, in our example application we see that C* tends to create paths made up of sequences of straight lines with significantly fewer turns than in the unconstrained case. This behavior is similar to what we observe in the pilot study and is validated by the post-study questionnaire, in which users reported an aversion to 'starting and stopping' and 'sharp turns'.

Additionally, our C* approach generalizes immediately to more complex domains with higher controllable degrees of freedom (e.g. robotic manipulators) where users similarly report control challenges based on interface constraints. While the control space of such problems will be larger, we are able to apply the same planning algorithm modifications by defining a Harel statechart to constrain the kinematics and assigning weights to the defined internal and external transitions. We would expect similar results, where the difficulties in operating a particular interface-robot combination result in specific path characteristics that are captured by C*.

Achieving similar results to C* with an expanded A* variant would perhaps be possible by tuning of the associated cost for each state-transition pair. However, using the C* algorithm, we only need to tune the single parameter representing how difficult external Harel state transitions are compared to internal transitions.

One limitation to our pilot study is that the participants were all new to the Sip-and-Puff interface. This interface is not trivial to master and can often take weeks for a user to become fully comfortable using. As such, we observed participants unintentionally follow paths they would otherwise not. For example, on several occasions, we noticed users struggle to unlatch from a latched state resulting in some unintended back-tracking. We expect that with experienced Sip-and-Puff users we would still see a difference between paths generated by their odometry data and the C* algorithm, but we could be more confident that all differences were due to user preference and not unfamiliarity with the interface.

A notable difference between the paths generated by our algorithm and those generated by the users is that C* tends to produce control sequences that only transition composite states twice. That is, the general structure of paths generated by our algorithm is the following. Beginning in the first composite state, the robot rotates towards its next goal. It then transitions to the second composite state which allows it to begin moving forward towards its goal location. Then the robot continues along this path, turning while moving until arriving at the goal location. Finally, it transitions back to the first composite state, allowing the robot to rotate in place until it achieves the goal orientation. By contrast, even with the constraint on the robot's heading, users produced paths that exaggerate turns even more than our algorithm requiring a greater number of external state transitions. We believe this area could be improved upon by learning average turning rates from further user studies and introducing those rates into our planning algorithm.

Lastly, it should be reiterated that our approach produces plans in the same state space as the user interface, which allows us to directly compare output signals from a user and the algorithm. The results of our pilot study show that C* produces control signals that align with user signals significantly more often than standard grid based search approaches and also reduce the average divergence between the user and autonomous signal. This is useful for control blending and analyzing the intent of a user.

6 Conclusion

We have presented a novel addition to heuristic-based search algorithms for robot path planning in shared human-robot control scenarios. The modifications are easy to understand and implement and can be used to alter any grid-search method and with any limited control interface. The resulting algorithm has many benefits including the ability to plan in the same state space, with the same constraints, as a user with a given control interface. Additionally, this work produces paths that more closely mimic those develop by a user, and as such, could be used more readily as part of a collaborative control schema.

References

Alt, H., and Godau, M. 1995. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications* 5:75–91.

Argall, B. D. 2014. Modular and adaptive wheelchair automation. In *International Symposium on Experimental Robotics*. IEEE.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development* 1:7–28.

Cipriani, C.; Zaccone, F.; Micera, S.; and Carrozza, M. C. 2008. On the shared control of an emg-controlled prosthetic hand: analysis of user–prosthesis interaction. *IEEE Transactions on Robotics* 24(1):170–184.

Demeester, E.; Hüntemann, A.; Vanhooydonck, D.; Vanacker, G.; Van Brussel, H.; and Nuttin, M. 2008. User-adapted plan recognition and user-adapted shared control: A bayesian approach to semi-autonomous wheelchair driving. *Autonomous Robots* 24(2):193–211.

Ferguson, D., and Stentz, A. 2007. Field D*: An interpolation-based path planner and replanner. In *Robotics Research*. Springer. 239–253.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of computer programming* 8(3):231–274.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, 530–535. Citeseer.

Hsu, D.; Kavraki, L. E.; Latombe, J.-C.; Motwani, R.; Sorkin, S.; et al. 1998. On finding narrow passages with probabilistic roadmap planners. In *Robotics: The Algorithmic Perspective: Workshop on the Algorithmic Foundations of Robotics*, 141–154.

Koenig, S., and Likhachev, M. 2005. Fast replanning for navigation in unknown terrain. *Transactions on Robotics* 21(3):354–363.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong planning A*. *Artificial Intelligence* 155(1):93–146.

LaValle, S. M. 1998. Rapidly-exploring random trees a new tool for path planning.

Nash, A.; Daniel, K.; Koenig, S.; and Felner, A. 2010. Theta*: Any-angle path planning on grids. volume 39, 533–579.

Parikh, S. P.; Grassi Jr, V.; Kumar, V.; and Okamoto Jr, J. 2004. Incorporating user inputs in motion planning for a smart wheelchair. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, 2043–2048. IEEE.

Parker Owan, Joseph Garbini, S. D. 2015. Uncertainty-based arbitration of human-machine shared control. *arXiv preprint* arXiv:1511.05996.

Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc., Reading, MA.

Philips, J.; Millán, J. d. R.; Vanacker, G.; Lew, E.; Galán, F.; Ferrez, P. W.; Brussel, H. V.; and Nuttin, M. 2007. Adaptive shared control of a brain-actuated simulated wheelchair. In *IEEE 10th International Conference on Rehabilitation Robotics (ICORR)*, 408–414. IEEE.

Quigley, M.; Conley, K.; Gerkey, B. P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Sa, I., and Corke, P. 2014. Vertical infrastructure inspection using a quadcopter and shared autonomy control. In Yoshida, K., and Tadokoro, S., eds., *Field and Service Robotics*, volume 92 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg. 219–232.

Sakoe, H., and Chiba, S. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing* 26(1):43–49.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 3310–3317. IEEE.

Trautman, P. 2015. Assistive planning in complex, dynamic environments: a probabilistic approach. *arXiv preprint* arXiv:1506.06784.

Wang, H., and Liu, X. P. 2014. Adaptive shared control for a novel mobile assistive robot. *IEEE/ASME Transactions on Mechatronics* 19(6):1725–1736.

Yanco, H. A. 2000. *Shared user-computer control of a robotic wheelchair system*. Ph.D. Dissertation, Massachusetts Institute of Technology.