Online, interactive user guidance for high-dimensional, constrained motion planning

Abstract

We consider the problem of planning a collision-free path for a high-dimensional robot. Specifically, we suggest a planning framework where a motion-planning algorithm can obtain guidance from a user. In contrast to existing approaches that try to speed up planning by incorporating experiences or demonstrations ahead of planning, we suggest to seek user guidance only when the planner identifies that it ceases to make significant progress towards the goal. Guidance is provided in the form of an intermediate configuration $\hat{q}$, which is used to bias the planner to go through $\hat{q}$. We demonstrate our approach for the case where the planning algorithm is Multi-Heuristic A* (MHA*) and the robot is a 34-DOF humanoid. We show that our approach allows to compute highly-constrained paths with little domain knowledge. Without our approach, solving such problems requires carefully-crafting domain-dependent heuristics.

1 Introduction

Motion-planning is a fundamental problem in robotics that has been studied for over four decades [LaValle, 2006]. However, efficiently planning paths in high-dimensional, constrained spaces remains an ongoing challenge. One approach to address this challenge is to incorporate user input to guide the motion-planning algorithm. While there has been much work on planning using human demonstration (see, e.g., [Argall *et al.*, 2009; Ye and Alterovitz, 2017]), there has been far less research on incorporating guidance as an interactive part of the planning loop.

Broadly speaking, interactive planning has been typically used in the context of sampling-based motion-planning algorithms [LaValle, 2006]. User guidance was employed for biasing the sampling scheme of the planner by having the user mark regions in the *workspace* that should be avoided or explored [Denny *et al.*, 2014; Yan *et al.*, 2015]. Alternatively, interactive devices such as a 3D mouse or a haptic arm have been used to generate paths in a (low-dimensional) configuration space. Such paths were then used by a planner to bias its sampling domain [Taïx *et al.*, 2012].

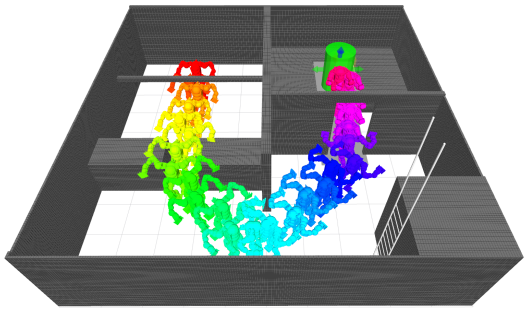


Figure 1: Planning domain—Humanoid robot needs to locomote in a challenging environment that involves circumventing obstacles and passing through narrow passages. Shown in multi-color are sparsed states from the path generated by our proposed approach.

We are interested in planning in high-dimensional, constrained spaces such as those encountered by a humanoid robot (see Fig. 1 and Sec. 2.1). In such settings, workspace regions often give little guidance to the planner due to the high dimension of the configuration space as well as the physical constraints of the robot. Additionally, providing the user guidance in the configuration space is extremely time consuming, even for expert users. Thus, though beneficial, user guidance should be employed scarcely.

Our key insight is that carefully chosen individual configurations suggested by a user can be used to effectively guide the planner when in need. Transforming this insight into a planning framework requires addressing three fundamental questions:

Q1. When should the planner ask the user for guidance?

Q2. What form should the user's guidance take?

Q3. How should the guidance be used by the planner?

Identifying *when* to obtain guidance (Q1, Sec 4.1) comes in stark contrast to existing approaches—we suggest to only employ user guidance when the planner *identifies* that it ceases to make significant progress towards the goal. Once obtained (Q2, Sec 4.2), guidance is used to *bias* the search algorithm towards regions that are likely to be beneficial (Q3, Sec 4.2). The search algorithm now has the additional task of deducing how much to rely on the user-provided guidance. It is worth emphasizing that obtaining this guidance does not require the user to understand the underlying search algorithm.

We demonstrate our approach using search-based planning algorithms (see, e.g., [Cohen *et al.*, 2014]) that perform a systematic search guided by heuristic functions. Specifically, we use multi-heuristic A* (MHA*) [Aine *et al.*, 2016; Narayanan *et al.*, 2015] which we detail in Sec. 2.2. MHA* provides a principled framework to address the aforementioned questions. Our conjecture is that the formulation can, in general, be incorporated with any motion-planning algorithm, however showing its effectiveness on other algorithms is out of the scope of the paper.

After describing our approach at high-level (Sec. 3) we demonstrate how it can be applied to the case of MHA* (Sec. 4). We continue by showing the effectiveness of our planner (Sec. 5). Specifically, we show that this general approach allows to compute highly-constrained motion plans for a spectrum of tasks including climbing stairs, walking under a bar, squeezing through a door and others, all with the same planner and heuristics. Without our approach, solving such problems requires the design of task-specific planners [Kaneko *et al.*, 2004] and carefully hand-designed domain-dependent heuristics. We conclude this paper with a discussion (Sec. 6).

2 Related work and algorithmic background

2.1 Motion planning for humanoid robots

Humanoid robots, for which we demonstrate our approach, often have several dozens of degrees of freedom making planning a challenging task, especially when taking additional constraints into account such as stability and contact constraints. One approach to plan the motion for such systems, is to use predefined, carefully chosen fixed gaits [Kaneko *et al.*, 2004]. However, when the terrain is uneven, such planners are inadequate at computing stable motions [Hauser *et al.*, 2008]. Another approach is to reduce the size of the search space by decomposing the degrees of freedom into functional groups such as locomotion and manipulation. Then functional-specific algorithms such as footstep planning are applied to the low-dimensional space (see, e.g., [Perrin *et al.*, 2012; Xia *et al.*, 2009] for a partial list). A high-dimensional planner is then used to "track" the plan generated in the low-dimensional space.

User guidance has been intensively incorporated in controlling the motion of humanoid robots, especially in complex tasks as those presented in the Darpa Robotics Challenge (DRC) [Spenko *et al.*, 2018]. In such settings, guidance ranged from teleoperating the robot to high-level task guidance (see e.g., [McGill *et al.*, 2017; Gray *et al.*, 2017]). However, as far as we are aware of, in all the aforementioned cases, *identification* of when to use guidance was done by a human operator and not by the system (except for relatively simple metrics where the system asks for help when it fails to complete some given task). Furthermore, the human guidance was used as a "hard" constraint forcing the system to make use of the guidance. In contrast, in our work the system automatically and independently identifies when it is in need of guidance. This guidance is then seamlessly incorporated as a soft constraint— the system biases the search towards the guidance while continuing to explore the search space as if the guidance was not given.

2.2 Multi Heuristic A* (MHA*)

Multi Heuristic A* (MHA*) [Narayanan *et al.*, 2015; Aine *et al.*, 2016] is a search-based planning algorithm that takes in multiple, possibly inadmissible heuristic functions in addition to a single consistent heuristic termed the *anchor* heuristic. It then uses the heuristic functions to simultaneously perform a set of weighted-A* [Pohl, 1970]-like searches. Using multiple searches allows the algorithm to efficiently combine the guiding powers of the different heuristic functions.

Specifically, for each search, MHA* uses a separate priority queue associated with each heuristic. The algorithm iterates between the searches in a structured manner that ensures bounds on sub-optimality. This can be done in a round-robin fashion, or using more sophisticated approaches that allow to automatically calibrate the weight given to each heuristic [Phillips *et al.*, 2015].

Part of the efficiency of MHA* is due to the fact that the value of the cost-to-come (the g-value) computed for each state is shared between all the different searches[1]. Sharing cost-to-come values between searches implies that if a better path to a state is discovered by any of the searches, the information is updated in all the priority queues. Moreover, if one search ceases to make progress towards the goal, it can use "promising" states visited by other searches to escape the local minimum.

2.3 Identifying progress in search-based planners

A key component in our work is automatically detecting when our planner requires user guidance. This requires *characterizing* regions where the planner ceases to make progress and algorithmic tools to *detect* such regions. These two requirements are closely related to the notion of *heuristic depressions* [Ishida, 1992] and *search vacillation* [Dionne *et al.*, 2011; Burns *et al.*, 2013].

A heuristic depression region is a region in the search space where the correlation between the heuristic values and the actual cost-to-go is weak. It is defined as a maximal connected component of states $\mathcal{D}$ such that all states in the boundary of $\mathcal{D}$ have a heuristic value that is greater than or equal to the heuristic value of any state in $\mathcal{D}$.

Such regions often occur in real-time search algorithms such as LRTA* [Korf, 1990] and LSS-LRTA* [Koenig and Sun, 2009] where the heuristic function is updated as the search progresses. Subsequently, current state-of-the-art algorithms guide the search to avoid states that have been marked as part of a heuristic depression [Hernández and Baier, 2012]. As we will see, we will use a slightly different characterization of when the planner ceases to make progress which we call *stagnation regions*. For details, see Sec. 4.1.

Search vacillation is a phenonemon when the search explores multiple small paths due to error in the heuristic. We

[1]To be precise, there are two variants of MHA* described in [Aine *et al.*, 2016]: Independent and Shared MHA* where the queues do not share and do share the g-values of states, respectively. In this paper when we use the term MHA*, it refers to the latter (shared) variant.

Algorithm 1 User-guided planning ($\mathcal{A}$)

```
1: while ¬A.solution_found() do
2:   while ¬A.in_stagnation_region() do
3:     A.run()                                   ▷ no user guidance
4:   g ← get_user_guidance()                    ▷ A in a stagnation region
5:   A.update_user_guidance(g)                  ▷ account for guidance
6:   while A.in_stagnation_region() and
     ¬A.in_stagnation_region(g) do
7:     A.run()                                   ▷ A uses guidance to escape stagnation region
8:   A.update_user_guidance(¬g)                 ▷ remove guidance
```

will use the notion of search vacillation to identify when the planner is in a stagnation region (i.e., when it ceases to make progress towards the goal). For details, see Sec. 4.1.

3 Algorithmic approach—user-guided planning

To employ our planning framework, we assume that we are given a motion-planning algorithm $\mathcal{A}$ that is endowed with two non-standard procedures which are planner dependent. The first, `in_stagnation_region()`, identifies when it is in a *stagnation region* (with or without the guidance), namely when $\mathcal{A}$'s search does not progress towards the goal. The second, `update_user_guidance()`, incorporates (or removes) the user guidance provided to $\mathcal{A}$.

Equipped with these functions, we can describe our planning framework, detailed in Alg. 1. The framework runs as long as no solution is found (line 1). It runs the planner $\mathcal{A}$ (lines 2-3) as long as it continuously makes progress towards the goal (namely, it is not in a stagnation region). Once a stagnation region is identified, user guidance is invoked (line 4) and $\mathcal{A}$ is updated to make use of this guidance (line 5). The planner continues to run while using the guidance as long as it is still in the stagnation region, and the guidance is useful (lines 6-7). Once it escapes the stagnation region (or if the guidance appears appears to be unhelpful), $\mathcal{A}$ is updated to remove the guidance that was provided by the user (line 8).

4 User-guided planning via MHA*

We demonstrate our general planning framework described in Sec. 3 for the case where the motion-planning algorithm $\mathcal{A}$ is MHA*. We assume that MHA* has a set of possibly inadmissible heuristics in addition to the anchor heuristic. We will refer to these heuristics as *baseline heuristics*. The user guidance will be used to generate *dynamic heuristics*. In Sec. 4.1-4.3 we describe how we answer the three questions that were posed in Sec. 1.

4.1 Invoking user guidance (Q1)

The heuristic functions of search-based planning algorithms, such as MHA*, can be used to estimate in a principled manner when the planner is in a stagnation region (Alg 1, lines 2 and 6). We suggest two alternative methods to identify when the planner ceases to make progress towards the goal, each resulting in a slightly different definition of a stagnation region. The first, which we call *vacillation-based* detection uses the notion of expansion delays [Dionne *et al.*, 2011]: Let $\mathcal{Q}$ be a priority queue and let e_{curr} be a counter tracking the total number of node expansions performed when using $\mathcal{Q}$. When a node is expanded, for each child node s we store the current expansion number using a counter $e(s)$. Now, the expansion delay for a state s is defined as

$$\Delta e = e_{\text{curr}} - e(s).$$

A moving average of the expansion delay of the nodes being expanded can be used as a proxy to measure the average progress that a planner makes along any single path. When the heuristic used by a planner is perfect, the expansion delay equals one, while on the other extreme, when performing uniform-cost search, the expansion delay can grow exponentially.

Given some parameter $\omega > 0$ we compute $\Delta e(\omega)$, the average expansion delay over a moving window of size ω.

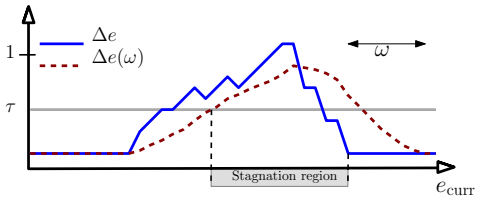


Figure 2: Visualization of the way vacillation-based stagnation regions are detected—expansion delay as a function of the number of expansions. Notice that the stagnation regions ends when $\Delta e = 1$. However, this is detected after ω additional steps.

Definition 1 *Let $\omega > 0$ be some window size and τ be some threshold value. A heuristic h associated with a queue $\mathcal{Q}$ is defined to*

1. *enter a stagnation region if $\Delta e(\omega) \geq \tau$,*
2. *exit a stagnation region if $\Delta e(\omega) = 1$*

For a visualization of vacillation-based stagnation regions and how they are detected (Def. 1), see Fig. 2.

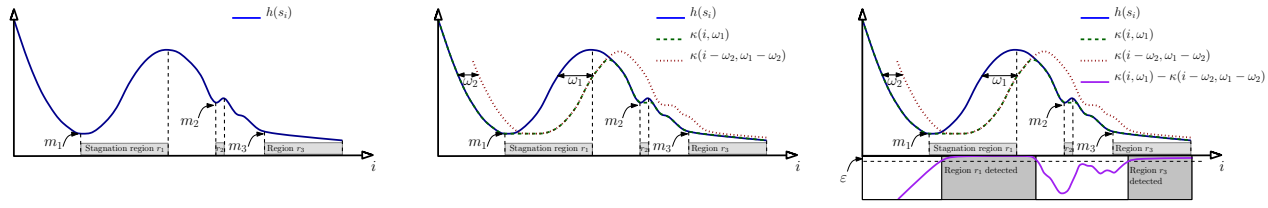
As we will see in our experimental section, due to the nature of the domain or the heuristic function, vacillation-based detection could be deceptive and not detect regions where the planner is not making significant progress. Thus, we suggest an alternative method, which we call *heuristic-based* detection. Let $\mathcal{Q}$ be a priority queue ordered according to some heuristic function $h(\cdot)$, s_i be the node expanded from $\mathcal{Q}$ at the i'th iteration and ω_1, ω_2 and ε be parameters such that $\omega_1 > \omega_2$ and ε is some threshold. We define $\kappa_{\mathcal{Q}}(i, \omega) = \min_{i-\omega \leq j \leq i}\{h(s_j)\}$. Namely, $\kappa_{\mathcal{Q}}(i, \omega)$ denotes the minimal value attained by h over the past ω expansions.

Definition 2 *A heuristic h associated with a queue $\mathcal{Q}$ is defined to be in a (heuristic-based) stagnation region if $\kappa_{\mathcal{Q}}(i, \omega_1) \geq \kappa_{\mathcal{Q}}(i - \omega_2, \omega_1 - \omega_2) - \varepsilon$.*

Namely, $\mathcal{Q}$ is in a heuristic-based stagnation region if looking at the previous ω_1 iterations, there was no reduction (by more than ε) in the minimum value of h in the last ω_2 states expanded from $\mathcal{Q}$. For a visualization of heuristic-based stagnation regions and how they are detected (Def. 2), see Fig. 3.

4.2 Form of user guidance (Q2)

We chose to obtain user guidance (Alg 1, line 4) in the form of an intermediate configuration $\hat{q}$ that is used to guide the planner. The framework includes a graphical user interface (GUI)



(a) (b) (c)

Figure 3: Visualization of heuristic-based stagnation-region detected. (a) Heuristic $h(s_i)$ (solid blue) has three local minima, m_1, m_2 and m_3 followed by three stagnation regions (light grey). Local minima m_2 is very small while m_3 is not a local minimum per se, yet the progress made between consecutive steps is smaller than the predefined threshold ε. (b) The function $\kappa(i, \omega_1)$ (dashed green) returns the minimal value $h(s_i)$ attained over the past ω_1 iterations. The function $\kappa(i - \omega_2, \omega_1 - \omega_2)$ (dotted red) returns the minimal value $h(s_i)$ attained over the past ω_1 iterations excluding the last ω_2 iterations. (c) The difference between the two functions (solid purple) indicates if there was significant progress (i.e., more than ε) made over the last ω_2 iterations. If not, then the planner detects a stagnation region (dark grey). Notice that the hysteresis parameters ω_1 and ω_2 (i) induce a lag from the time that the stagnation region starts until it is detected, (ii) allow to avoid detecting stagnation region r_2.

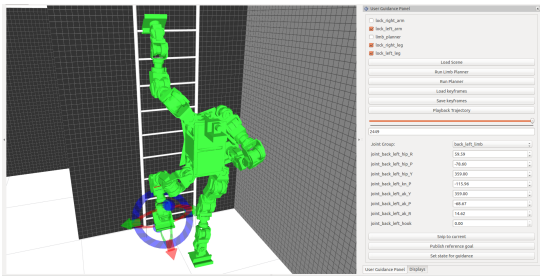


Figure 4: GUI used to provide guidance to the planner. The panel on the right hand side allows to select different joint groups, move the joints and pass the guidance to the planner. The interactive marker shown on the left hand side allows to move the robot's end effectors and its torso. The user can move the torso while having specified end-effectors locked in place. The tool runs an inverse-kinematics solver to enable such movements.

(Fig. 4) capable of depicting the robot and the workspace. Once user guidance is invoked, a configuration in the stagnation region is obtained and the robot is placed in that configuration in the GUI. This allows the user to intuitively try and understand where the planner faces difficulty and how to guide it out of the stagnation region. The user then suggests the guidance $\hat{q}$ by moving the robot's joints, end effectors or torso. The tool runs a state validity checker in the background which restricts the user from providing invalid configurations, e.g. with respect to collision, joint limits and stability. The same validity checker is used by the planner.

4.3 Using user guidance (Q3)

We assume that MHA* has at least one baseline (possibly inadmissible) heuristic h_{goal} (in addition to the anchor heuristic) which approximates the cost to reach the goal from every state. Furthermore, we assume that for every configuration q, there exists a heuristic h_q where $h_q(s)$ estimates the cost to reach q from state s.

Given user guidance in the form of a configuration $\hat{q}$, we dynamically generate a new heuristic

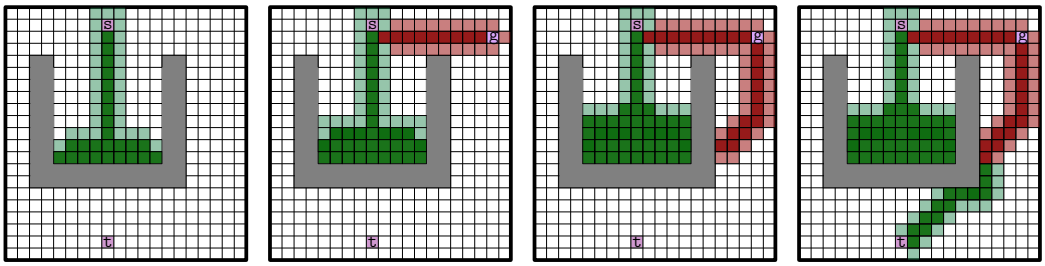
$$\hat{h}(s) = \begin{cases} h_{\hat{q}}(s) + h_{\text{goal}}(\hat{q}), & \text{if } \hat{q} \text{ is not an ancestor of } s, \\ h_{goal}(s), & \text{if } \hat{q} \text{ is an ancestor of } s. \end{cases}$$

Namely, $\hat{h}$ estimates the cost to reach the goal via $\hat{q}$ (see also [Chakrabarti *et al.*, 1986] for intuition regarding such a heuristic). If the state was reached by passing through $\hat{q}$, then the value of $\hat{h}$ is simply the estimation of the cost to reach the goal.

Equipped with the heuristic $\hat{h}$, we add a new queue to MHA* prioritized using the $\hat{h}$. States expanded using this queue will be biased towards $\hat{q}$ (see also [Islam *et al.*, 2015] for more details on adding heuristics dynamically to MHA*). Recall that in MHA*, nodes are shared between the different queues. Thus, once a state has been found that can be used to get the planner out of the stagnation region, it will be expanded by the other queues using their heuristics. Once this is detected, the newly-added queue is removed. In general, we can add a dynamic queue for every baseline heuristic if they are more than one. However, for simplicity, in this work we use a single baseline heuristic and add one dynamically generated queue when the user provides guidance. Since the search runs under the MHA* framework, although the dynamic heuristic $\hat{q}$ is inadmissible, we still hold completeness and bounded sub-optimality guarantees.

For pseudo-code describing each specific function used in Alg. 1 under the framework of MHA*, see Alg. 2. When adding the dynamic heuristic, if the baseline heuristic escaped a stagnation region but the configuration $\hat{q}$ was *not* reached, we suspend the dynamic queue but do not discard it. This is done to first try reusing the last guidance before obtaining a new one. Thus, when the planner will detect that it is in a stagnation region, it will first resume the suspended dynamic heuristic (if one exists). If the baseline heuristic escaped a stagnation region and the configuration $\hat{q}$ was reached it will no longer be useful again and hence will be discarded. Finally, if the dynamic heuristic is in a stagnation region then it is discarded and the user will be queried for a new guidance.

For a visualization of the way the algorithm progresses, see Fig. 5. Note that although for the illustrated example, the search happens to pass through the guidance, in general it is not a constraint in the framework.



(a) (b) (c) (d)

Figure 5: Algorithm progression. States popped from a priority queue Q (still in Q) are depicted using dark (light) colors. Start, target and user-guided states are depicted in purple with the letters s, t, g, respectively. In this example MHA* alternates between queues in a round-robin fashion and heuristic values are inflated by a weight of $w = \infty$. MHA* starts with a single baseline heuristic (green) which is the Euclidean distance to goal and a stagnation region is identified. (b) User provides guidance g and an additional heuristic (red) is automatically generated and drives the search towards the guidance (notice that the baseline heuristic continues to search Q). After passing through the g, the additional heuristic (red) drives the search towards the goal. (c) After the additional heuristic found states that are placed at the top of Q, the additional heuristic is deleted and the baseline heuristic continues to drive the search towards the goal.

Algorithm 2 User-guided MHA*

```
1:  function in_stagnation_region()
2:    if baseline heuristic in stagnation then
3:      return true
4:    return false

5:  function in_stagnation_region(g)
6:    if dynamic heuristic in stagnation then
7:      return true
8:    return false

9:  function update_user_guidance(arg)
10:   if arg = g then                          ▷ account for guidance
11:     if exists suspended dynamic heuristic then
12:       add suspended dynamic heuristic
13:     else
14:       get new user guidance and add dynamic heuristic
15:   if arg = ¬g then                          ▷ remove guidance
16:     if in_stagnation_region(g) then
17:       remove dynamic heuristic            ▷ guidance is not useful
18:     else                                  ▷ dynamic heuristic is not in stagnation
19:       if states passed through guidance then
20:         discard dynamic heuristic         ▷ will not be useful in future
21:       else
22:         suspend dynamic heuristic         ▷ may be useful in future
```

5 Evaluation

We evaluated the performance of our planning framework on a 34-DOF WAREC humanoid robot [Matsuzawa *et al.*, 2015] which is constrained to maintain static stability at all times. To construct the search space, we used a set of motion primitives which are short kinematically feasible motion sequences. These are used to generate the actions, or the successors, of a state during the search. For additional details see [Dornbush *et al.*, 2018]

For each experiment we used a small set of fairly generic heuristics. This was done to demonstrate that our approach allows to significantly reduce the engineering effort required in carefully crafting domain-specific heuristics. We conducted two experiments which largely differ in the nature of mobility and thus employ different baseline heuristic functions. However, the dynamic heuristic function in both the experiments is the Euclidean distance in the joint 34-DOF space. The parameter values used in both experiments are $\omega_1(200), \omega_2(50)$ and $\varepsilon(50)$ for the heuristic-based, and $\tau(30)$ and $\omega(10)$ for the vacillation-based stagnation-region detection. We conducted the same experiments (results ommited due to lack of space) with a wide variety of parameters and obtained only slight differences in the results, demonstrating that our approach is highly robust to the choice of parameters for our domain.

	Bipedal locomotion		Ladder mounting	
	(a)	(b)	(a)	(b)
Planning time(s)	205.2 ± 22.7	183.4 ± 25.0	149.4 ± 30.0	101.4 ± 10.3
Total time(s)	293.5 ± 33.1	256.9 ± 38.0	176.3 ± 29.6	128.4 ± 10.3
Expansions	1800.1 ± 88.3	1770.5 ± 125.9	980.8 ± 240.6	644 ± 69.3
Num. of guidances	5.4 ± 1.17	5.4 ± 1.1	4.7 ± 1.2	5 ± 0.8
Avg. guidance time	16.4 ± 2.8	13.6 ± 3.6	6.0 ± 1.5	5.5 ± 0.9

Table 1: Experimental results for the bipedal locomotion and ladder mounting tasks for (a) vacillation-based detection (b) heuristic-based detection averaged over 10 trials.

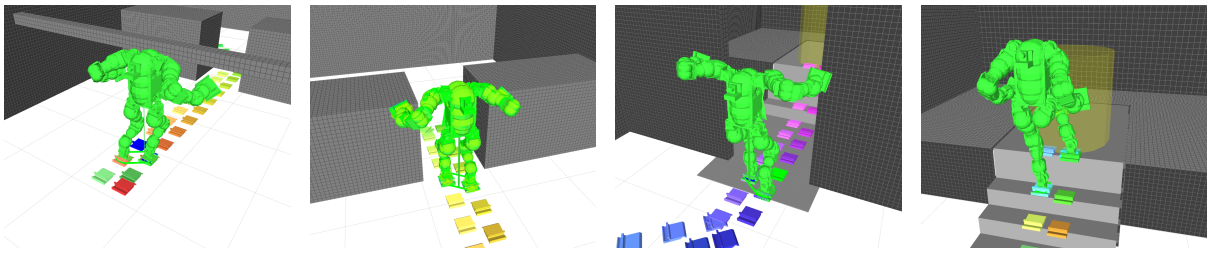
Planning statistics for all experiments are provided in Table 1. In all the experiments the planner succeeded to reach the goal with the help of a user. The deviations in the results come from the quality of the guidance provided. For videos demonstrating the approach, see [2].

5.1 Bipedal locomotion

The first task we considered is bipedal locomotion, depicted in Fig. 1. Here, we employ the adaptive dimensionality framework [Gochev *et al.*, 2011] which iterates over two stages: an adaptive-planning phase which plans in a low-dimensional space when possible and a tracking phase which plans in the high-dimensional space. Our user-guided planner is integrated within the high-dimensional planner to search for a path that tracks the footstep plan generated in the adaptive-planning phase.

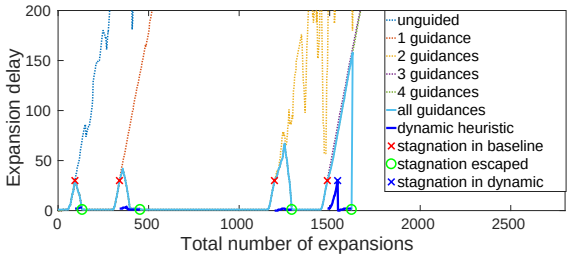
The baseline heuristic we used was designed to assist the search with a general walking or stepping capability. For each step the footprint of the target footstep is visualized for the user to be able to better understand where the planner is stuck

[2] https://drive.google.com/drive/folders/1OFJ7XwbiVVbgw1vmentz575nTCrQBtCu?usp=sharing

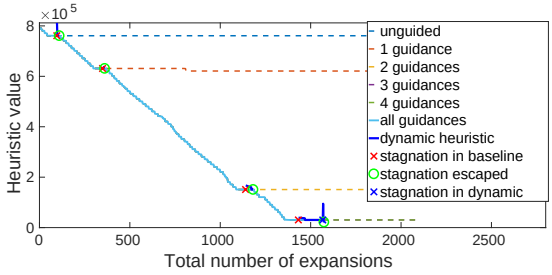


(a) (b) (c) (d)

Figure 6: Bipedal locomotion in challenging scenarios: (a) The robot has to squat or bend down to pass under a beam. (b) The robot has to pass through two tables by lifting its elbows high above the tables. (c) The robot has to squeeze through a narrow doorway by tucking in its arms. (d) The robot has to step onto a relatively high platform to reach the goal for which it has to lean a little bit more to its left side to be able to lift the right foot further up.



(a) Vacillation-based detection



(b) Heuristic-based detection

Figure 7: MHA* progress with and without guidance. (a) vacillation-based and (b) heuristic-based stagnation-region detection.

and how to provide the guidance. In relatively easier scenarios this heuristic would suffice in guiding the search. However if the search encounters harder situations such as those depicted in Fig. 6, then it is likely to get trapped into a stagnation region and would invoke user-guidance.

Results demonstrating the effectiveness of our framework are depicted in Fig. 7. Specifically, we plot the method to detect stagnation regions (average expansion delay and heuristic values for vacillation-based and heuristic-based detection, respectively) as a function of the number of queue expansions. For both methods, we ran our algorithm with and without user guidance. Namely, if a stagnation region was detected, we recorded the state of the planner and then continued to run it once without asking the user for guidance and once using our approach. This was done every time a stagnation region was detected. Results show that without guidance, the planner fails to make any significant progress. On the other hand, when guidance is given, then the algorithm escapes the stagnation region and resumes to make progress towards the goal.

Interestingly, vacillation-based detection incurs a lot of noise—notice the large oscillations in the dashed yellow curve in Fig. 7(a). A possible explanation is that even in regions where the planner cannot ultimately progress towards the goal, there may be multiple short feasible paths to explore. In such settings, the expansion delay for each such path will be small while the planner does not really progress towards the goal. Indeed, numerical results stated in Table 1 confirm that our heuristic-based method for stagnation-region detection outperforms the vacillation-based method. We observed similar trends for the second experiment (ladder climbing) where vacillation-based method falsely detected a stagnation region exiting, plots ommitted.

5.2 Mounting onto a ladder

The second task we considered was mounting onto a ladder from a standing position (see Fig. 4) to the desired contact poses. This problem is challenging because the ladder rungs and the robot grasping hooks induce narrow spaces in the configuration space which have to be traversed while maintaining stability, to establish the desired contacts. The baseline heuristic used is the summation of four six-dimensional Euclidean distances between each of the four end effectors and their respective target poses. For numerical results, see Table 1.

6 Discussion

In Sec. 5 we demonstrated how user guidance allows to solve highly-constrained motion-planning problems in high-dimensional spaces with only simple baseline heuristics. An alternative approach would be to add domain-dependent carefully-crafted heuristics [Vijayakumar, 2017] which allow to faster solve the same problems completely autonomously.

When faced with a new problem domian (say, climbing a ladder, crawling on uneven terrain etc.) we can either design additional heuristics to tackle this problem or leverage from user-provided online guidance. If our problem requires planning in multiple, diverse domains this problem is accentuated—should we use a large arsenal of heuristics that can address each domain or should we have a small set of baseline heuristics that will (roughly) address all domains and rely on user guidance when these baseline heuristics fail? There is no clear answer to this question and our approach simply offers a general alternative to existing approaches.

References

[Aine *et al.*, 2016] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic A*. *IJRR*, 35(1-3):224–243, 2016.

[Argall *et al.*, 2009] Brenna Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[Burns *et al.*, 2013] Ethan Burns, Wheeler Ruml, and Minh Binh Do. Heuristic search when time matters. *J. Artif. Intell. Res*, 47:697–740, 2013.

[Chakrabarti *et al.*, 1986] PP Chakrabarti, Sujoy Ghose, and SC Desarkar. Heuristic search through islands. *J. AI*, 29(3):339–347, 1986.

[Cohen *et al.*, 2014] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Single- and dual-arm motion planning with heuristic search. *IJRR*, 33(2):305–320, 2014.

[Denny *et al.*, 2014] Jory Denny, Read Sandström, Nicole Julian, and Nancy M. Amato. A region-based strategy for collaborative roadmap construction. In *WAFR*, pages 125–141, 2014.

[Dionne *et al.*, 2011] Austin J. Dionne, Jordan Tyler Thayer, and Wheeler Ruml. Deadline-aware search using on-line measures of behavior. In *SoCS*, pages 39–46, 2011.

[Dornbush *et al.*, 2018] Andrew Dornbush, Karthik Vijayakumar, Sameer Bardapurkar, Fahad Islam, and Maxim Likhachev. A single-planner approach to multi-modal humanoid mobility. *CoRR*, abs/1801.10225, 2018.

[Gochev *et al.*, 2011] Kalin Gochev, Benjamin J. Cohen, Jonathan Butzke, Alla Safonova, and Maxim Likhachev. Path planning with adaptive dimensionality. In *SoCS*, pages 52–59, 2011.

[Gray *et al.*, 2017] Steven Gray, Robert Chevalier, David Kotfis, Benjamin Caimano, Kenneth Chaney, Aron Rubin, and Todd Danko. An architecture for human-guided autonomy: Team TROOPER at the DARPA robotics challenge finals. *J. Field Robotics*, 34(5):852–873, 2017.

[Hauser *et al.*, 2008] Kris K. Hauser, Timothy Bretl, Jean-Claude Latombe, Kensuke Harada, and Brian Wilcox. Motion planning for legged robots on varied terrain. *IJRR*, 27(11-12):1325–1349, 2008.

[Hernández and Baier, 2012] Carlos Hernández and Jorge A Baier. Avoiding and escaping depressions in real-time heuristic search. *J. Artif. Intell. Res*, 43:523–570, 2012.

[Ishida, 1992] Toru Ishida. Moving target search with intelligence. In *AAAI*, pages 525–532, 1992.

[Islam *et al.*, 2015] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. Dynamic multi-heuristic A*. In *ICRA*, pages 2376–2382, 2015.

[Kaneko *et al.*, 2004] Kenji Kaneko, Fumio Kanehiro, Shuuji Kajita, Hirohisa Hirukawa, T. Kawasaki, Masaru Hirata, Kazuhiko Akachi, and Takakatsu Isozumi. Humanoid robot HRP-2. In *ICRA*, pages 1083–1090, 2004.

[Koenig and Sun, 2009] Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *AAMS*, 18(3):313–341, 2009.

[Korf, 1990] Richard E Korf. Real-time heuristic search. *J. AI*, 42(2-3):189–211, 1990.

[LaValle, 2006] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[Matsuzawa *et al.*, 2015] Takashi Matsuzawa, Kenji Hashimoto, Xiao Sun, Kazuhiro Uryu, Ayanori Koizumi, Shinya Hamamoto, Tomotaka Teramachi, and Atsuo Takanishi. Development of disaster response robot with commonly structured limbs and experiment in climbing vertical ladder. In *ICAM*, pages 142–143, 2015.

[McGill *et al.*, 2017] Stephen G McGill, Seung-Joon Yi, Hak Yi, Min Sung Ahn, Sanghyun Cho, Kevin Liu, Daniel Sun, Bhoram Lee, Heejin Jeong, Jinwook Huh, et al. Team THOR's entry in the DARPA Robotics Challenge Finals 2015. *J. Field Robotics*, 4(34):775–801, 2017.

[Narayanan *et al.*, 2015] Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Improved multi-heuristic A* for searching with uncalibrated heuristics. In *SoCS*, pages 78–86, 2015.

[Perrin *et al.*, 2012] Nicolas Perrin, Olivier Stasse, Leo Baudouin, Florent Lamiraux, and Eiichi Yoshida. Fast humanoid robot collision-free footstep planning using swept volume approximations. *Trans. Robotics*, 28(2):427–439, 2012.

[Phillips *et al.*, 2015] Mike Phillips, Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Efficient search with an ensemble of heuristics. In *IJCAI*, 2015.

[Pohl, 1970] Ira Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[Spenko *et al.*, 2018] Matthew Spenko, Stephen Buerger, and Karl Iagnemma. *The Darpa Robotics Challenge Finals: Humanoid Robots To The Rescue*, volume 121 of *Springer Tracts in Advanced Robotics*. 2018.

[Taïx *et al.*, 2012] Michel Taïx, David Flavigné, and Etienne Ferré. Human interaction with motion planning algorithm. *J. Intell. Robot. Syst.*, 67(3):285–306, 2012.

[Vijayakumar, 2017] Karthik Vijayakumar. A multi-heuristic framework for humanoid planning. Master's thesis, Robotics Institute, Carnegie Mellon University, 2017.

[Xia *et al.*, 2009] Zeyang Xia, Guodong Chen, Jing Xiong, Qunfei Zhao, and Ken Chen. A random sampling-based approach to goal-directed footstep planning for humanoid robots. In *AIM*, pages 168–173, 2009.

[Yan *et al.*, 2015] Yu Yan, Emilie Poirson, and Fouad Bennis. An interactive motion planning framework that can learn from experience. *CAD*, 59:23–38, 2015.

[Ye and Alterovitz, 2017] Gu Ye and Ron Alterovitz. Demonstration-guided motion planning. In *Robotics Research*, pages 291–307. 2017.