Fast Footstep Planning with Aborting A*

Marcell Missura and Maren Bennewitz

***Abstract*— Footstep planning is the dominating approach when it comes to controlling the walk of a humanoid robot, even though a footstep plan is expensive to compute. The most prominent proposals typically spend up to a few seconds of computation time and output a sequence of up to 30 steps all the way to the goal. This way, footstep planning is applicable only in static environments where nothing changes after a plan has been computed. Since uncontrolled environments present challenges such as unforeseen motion of other objects and unexpected disturbances to balance, fast replanning of a footstep plan while the robot is in motion is highly desirable. We present a new way of fast footstep planning - Aborting A* - which is able to guarantee a replanning rate of 50 Hz by aborting an A* search before completion. We make aborting possible by using a novel, obstacle-aware heuristic function that lays out rotate-translate-rotate motions along the shortest path to the goal, enabling us to stop the planning progress prematurely with a target-oriented solution at any time during the search, even after only a few nodes have been expanded. We show in our experiments that despite the bounded computation time, our planner computes good results and does not get stuck in local minima.**

I. INTRODUCTION

The discrete footsteps of a humanoid robot naturally lend themselves for navigation and control when a footstep plan is computed in order to determine where the robot should place its next steps. Most research revolving around this topic has converged to regarding footstep planning as a graph-search and using the A* algorithm to solve it. Respectable results have been achieved that manage to walk humanoid robots over slanted cinder blocks and narrow beams and to step over obstacles [1]. So far, the prominent approach is to compute a long sequence of footsteps that reach all the way to the target and to follow the footsteps as closely as possible using precise motion tracking. These plans often consist of 30 steps or more, requiring up to a few seconds of computation time to produce. At this time scale, it is not practical to replan while the robot is in motion. Realistic environments, however, constantly present unforeseen events that necessitate replanning. Moving obstacles can change direction in unpredictable ways and require fast reaction to avoid a collision. Disturbances to balance need to be dealt with quickly to avoid falling. Drifting away from the plan during execution is also best solved by replanning. It is highly desirable to meet these challenges with a controller that has a short and guaranteed response time.

At the same time, finishing a plan all the way to the goal is indeed necessary in order to keep the system from getting stuck forever in local traps. The computation of a plan to completion and a controller response in bounded-time appear to be, however, mutually exclusive. Consequently, our navigation system is made of two components running at different time scales. One is a global path planner that plans all the way to the goal and has the time to do so because the second, a fast-paced but short-sighted footstep controller, takes the responsibility for control actions that have to happen quickly. The global planner informs the footstep controller of an intermediate target that the robot is to aim for in order to progress towards the global goal and it is up to the footstep controller to figure out the next steps to take towards this intermediate goal while negotiating the challenges of the environment with a high response rate. Now with our contribution, A* footstep planning can be used to realize a high-frequency footstep controller with a guaranteed response time of 20 milliseconds. The key concepts to achieve such low computation times are a bounded area for planning and a novel heuristic that lays out rotate-translate-rotate motions along the shortest path to the goal and allows aborting the search prematurely before the target was found.

All authors are with the Humanoid Robots Lab, University of Bonn, Germany. Contact: missura@cs.uni-bonn.de

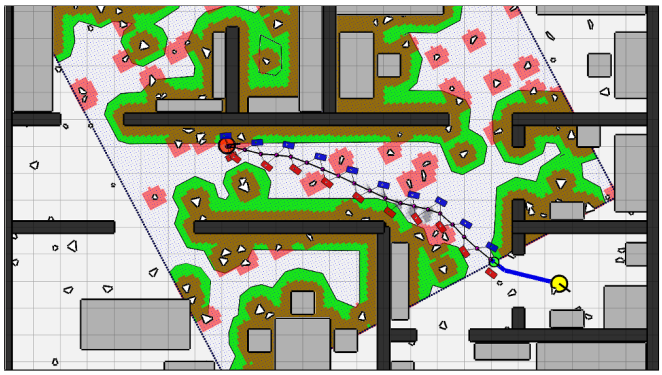


Fig. 1: Example footstep plan with cluttered objects on the floor. 20 footsteps were computed towards the intermediate target within a computation time limit of 20 milliseconds. The cyan circle marks the start state, the yellow circle marks the goal state. The smaller orange circle is an intermediate target at the boundary of the local map outside of which only a 2D path is planned (thick blue line). The areas shaded in red indicate regions where the feet must not be placed and the green colored region show polygons that are used for shortest path planning.

II. RELATED WORK

The origins of A* footstep planning lie in the pioneering work of Kuffner *et al.* [2], [3] and Chestnut *et al.* [4] who realized an online footstep planner for the ASIMO robot that could run at a rate of roughly 1 Hz. A binary occupancy map served as environment representation and a simple A* search was used with only two cost components for step costs and environment costs. The heuristic was precomputed using a wheeled robot model. Even though the heuristic

was not admissible, it helped to guide the search around non-traversable obstacles towards the target and the footstep planner computed good solution paths in reasonable time.

Ayaz *et al.* [5] presented another early approach to step over obstacles with the HRP-2 robot. The possible footstep locations were strongly constrained and several simplifying assumptions were made. As a result, the set of planning problems that could be solved was limited and the resulting plans were not optimal.

Hornung *et al.* [6] [7] investigated different flavours of the A* algorithm [8] for footstep planning in a planar environment. The authors implemented a footstep planner using R* and ARA* with the Euclidean and the Dijkstra heuristic. Anytime Repairing A* (ARA*) [9] computes multiple passes with an inflated heuristic where the inflation factor decreases with each run. This way, a suboptimal solution can be found quicker than with A* and the quality of the solution increases with time. R* [10] is a probabilistic version of A* that samples the state space and concentrates on the states that can be reached quickly. Both A* variates come with a bound on the suboptimality of the solution. Hornung *et al.* also proposed a mixed level-of-detail solution [11] where in free space only a 2D path is computed and single footsteps are planned only in the close vicinity of obstacles. With this approach, the authors achieved a planning rate of 2 to 3 Hz. Garimort *et al.* [12] proposed to use D*-Lite [13] for footstep planning due to its replanning capabilities. The authors showed that after an initial computation around the 1 second mark, the footstep plan can be updated during walking at a rate of approximately 2 Hz.

With the wind of the Darpa Humanoid Challenge in 2015, numerous instances of footstep planners have been presented [1] [14] [15] [16] [17] [18] that surpassed the earlier planar walkers with 3D walking capabilities. Driven by the tasks of the competition, human operator-guided systems emerged complete with point-cloud processing and features such as overstepping gaps and walking on surfaces made of slanted cinder blocks. Since the environment was static during the competition, fast replanning capabilities were not needed. The work of Griffin *et al.* [1] is a most refined version of such a footstep planner where based on a geometric world representation, the planning functionalities have been extended with more precise treatment of edges leading up to a humanoid robot being able to walk along a narrow beam no wider than the foot of the robot. Deits *et al.* [19] [14] presented an alternative approach to A* footstep planning using a mixed-integer quadratic program optimization. It relies on a segmentation of the environment into convex regions. The computation time needed for all of the aforementioned approaches ranges up to a few seconds.

Hildebrandt *et al.* [20] [21] proposed the most complete approach to footstep planning to our knowledge. The authors' framework includes a 3D perception module [22] where walkable surfaces are modeled as planes and all other objects as swept sphere volumes, which support real-time collision checking [23]. The included A* planner can overstep obstacles and is supported by a reactive balance controller. The system can walk in an unsupervised manner driven only by onboard sensing and replans during walking at a pace of around 5 Hz. So far, the system has been tested only in flat and open environments with small, convex objects. Since the Euclidean heuristic is used for the A* search, protruding obstacles can drastically impair performance.

Perrin *et al.* [24] used swept volume approximations for efficient 3D collision checking up to a certain height and perform 2D footstep planning. While this approach finds collision-free footstep plans in cluttered environments, it does not possess real-time capabilities. Karkowski *et al.* [25] considered the use of a limited horizon in order to accelerate the planning speed. The step action set was adapted to the terrain on the fly respecting reachability constraints. This approach has achieved remarkable runtimes around the 50 millisecond mark in small and simple environments where the target was placed in front of the robot and no obstacles block the way. Since the planner uses a forward-directed heuristic, it is likely to degrade in obstacle-rich situations.

Our contribution over the aforementioned works is the inclusion of the shortest path into the heuristic function and the formulation of the Path RTR (rotate-translate-rotate) function that greatly helps to guide the search along the shortest path and achieves true abortability. To the best of our knowledge, we are the first to breach a replanning rate of 50 Hz.

III. ABORTING A*

A. Prerequisites for Footstep Planning

We envision our footstep planner to be a part of a larger robot operating system where a global map and localization in the map already exist. We assume to be given a start pose $S = (x_S, y_S, \theta_S)$ and a goal pose $G = (x_G, y_G, \theta_G)$ in the global reference frame where x and y are the Cartesian coordinates of a pose and θ is the orientation. We also assume to be given a global 2D path $P(S, G) = \{(x_i, y_i) \mid i = 0..k\}$ where $(x_0, y_0) = (x_S, y_S)$ and $(x_k, y_k) = (x_G, y_G)$.

B. Planning in a Local Map

We confine the operational area of our footstep planner to a local map as shown in Figure 1. The local map is a square of $8\,m \times 8\,m$ that extends four meters to the left and right, six meters to the front, and two meters to the back of the robot. Footstep planning and shortest path computations take place only within the local map. Outside of the local map, the global path P completes the plan.

The global path from S to G is intersected with the local map in order to determine an intermediate goal pose $\tilde{G}$ that serves as a target for the planning within the local map. The intersection is computed such that the global path is followed from the start towards the goal until the first intersection with the boundary of the local map is found, or the goal is reached. Thus, the intermediate goal $\tilde{G}$ is located either on the boundary of the local map, or inside the local map if the entire path lies inside in which case $\tilde{G} = G$. The start and goal poses and the intermediate goal are illustrated in Fig. 1.

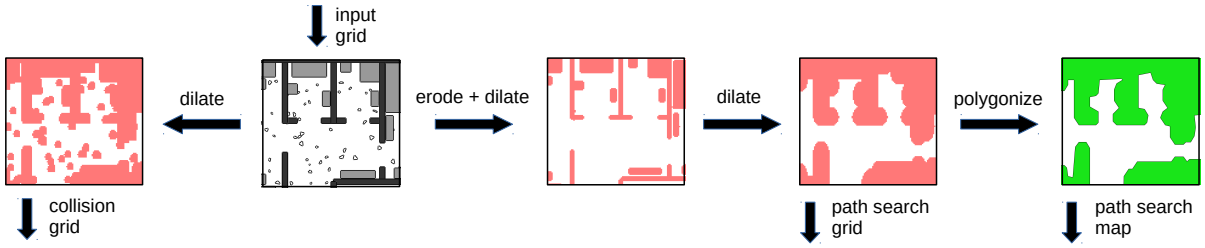


Fig. 2: Illustration of our local map processing pipeline. We process an occupancy grid in two ways. To the left: The occupancy grid is dilated by the foot radius to create a collision map for determining collision-free footsteps locations. To the right: The occupancy grid is eroded and dilated to remove small debris that can be overstepped. The result is dilated again by the robot radius. Then, the inflated grid is converted to polygons for path planning.

C. Environment Representation

The input into our planning system is an occupancy grid with a cell size of 5 cm that coincides with the local map. Each cell indicates whether it is free, occupied but oversteppable, or occupied and *not* oversteppable. Fig. 2 shows an illustration of our map processing pipeline that processes copies of the input grid using morphological operations and produces two distinct maps for the purposes of collision checking and path planning. The first map is a grid map used for collision checking of the feet. We dilate the input grid by the radius of the robot's foot and generate an inflated representation of obstacles. Using this collision map, whether a footstep can be placed at a certain location can be decided with a simple table lookup. For path planning, we first erode and then dilate a copy of the input grid with a radius of 10 cm in order to erase small objects that can be overstepped. Cells of non-oversteppable type are excluded from the erosion and remain in the map. Then, we dilate the map further with a radius of 30 cm in order to inflate the obstacles by the robot radius. Finally, we convert the dilated grid to polygons using contour detection. The obtained polygons are used for the shortest path computations in the heuristic function (cf. Sec. III-G). This processing pipeline needs only a small amount of computation time (<1 ms) and an earlier version of it has been demonstrated to work stably on a walking robot [26].

D. Action Set

The centerpiece of our A* search is a step action set

$$\mathbb{S} = \left\{ \begin{pmatrix} \Delta x(i) \\ \Delta y(j) \\ \Delta\theta(k) \end{pmatrix} \middle| \left\| \begin{pmatrix} 2\alpha_i - 1 \\ \alpha_j \\ 2\alpha_k - 1 \end{pmatrix} \right\|_p \leq 1;\ i,j,k = 0..n \right\}, \tag{1}$$

where $\alpha_i = \frac{i}{n-1}$, $\alpha_j = \frac{j}{n-1}$, and $\alpha_k = \frac{k}{n-1}$, and

$$\Delta x(i) = x_{\min} + \alpha_i \left(x_{\max} - x_{\min}\right), \tag{2}$$

$$\Delta y(j) = y_{\min} + \alpha_j \left(y_{\max} - y_{\min}\right), \tag{3}$$

$$\Delta\theta(k) = \theta_{\min} + \alpha_k \left(\theta_{\max} - \theta_{\min}\right), \tag{4}$$

respectively. We generate the step set by defining maximum offsets in the negative and positive x ($x_{\min}, x_{\max}$), y ($y_{\min}, y_{\max}$), and θ ($\theta_{\min}, \theta_{\max}$) directions and sampling each dimension evenly with $n = 9$ notches. However, we discard the candidates whose unit vector p-norm with $p = 1.7$ exceeds a length of 1 (see Eq. (1)). This pruning step removes step candidates with a too extreme combination of two or three directions. We are left with 241 step actions, which become the branching factor of the A* search. Note that the y-direction is handled differently from the other two. The reason for this is that while the robot can step in front of or behind the support foot, and can turn its foot to the left and right, it cannot cross its legs and step onto the other side of the support foot. Also note that this step set is defined for when the robot is standing on the right foot and stepping with the left. When a step with the right foot is needed, the step set is mirrored by inverting the signs of the y and θ directions.

E. Node Expansion

Planning happens in the coordinate system of the local grid. The start state S defines the transformation of the local grid with respect to the world and the initial footstep location $F_0 = (0, 0, 0)$ of the search lies in the origin of the local map.

The A* algorithm progresses by expanding the topmost step F_t in the priority queue and applying every step $s_i \in \mathbb{S}$ in the step set. When applying a step s_i, we compute a new support foot location $F_{t+1} = F_t + s_i$ and also a nominal center of mass location $C_{t+1} = F_t + 0.5 s_i$, which is located in the middle of the step. We designed the state space to be continuous and refrain from snapping F or C to a grid. Instead, we maintain a grid for closing states where we look up C_{t+1} and check if its location is already closed. If not, the cell is closed and all subsequent steps whose center of mass location C maps to a closed cell are ignored. The closed grid coincides with the local map and has a cell size of 5 cm or 0.1 radians, respectively.

After the closed check, the new step F_{t+1} is collision checked in the collision grid (cf. Sec. III-C). Additionally, the center of mass location C_{t+1} is checked whether it lies inside one of the polygons of the path search map. If either one is true, the step is discarded.

After the collision check, we compute $g(C_{t+1})$, the path cost so far, $h(C_{t+1})$, the estimated cost-to-go, and $f(C_{t+1}) = g(C_{t+1}) + h(C_{t+1})$, the value the steps are ordered by in the priority queue. By our design, every step has a cost of 1. Thus, g is simply the number of steps taken so far. The computation of h is more involved and will be explained in the next section. Finally, the new step F_{t+1} is pushed into the priority queue and the next step is popped for expansion.

In accordance with our Aborting A*, we check after every pop whether either a limit on the number of expansions has been reached, or the computation time has run out. If either of the abort conditions is met, we return the step with the smallest h value found so far. We chose the state with the lowest heuristic, because the last opened step can be anywhere in the graph, but the step with the lowest h value is sure to reach closest to the target. If a popped step reaches a heuristic value $h(C_t) < 0.5$ (less than half a step), the search is halted and the last opened step is returned as a solution from which parent pointers are followed to reconstruct the step sequence.

F. Heuristic Function

For the heuristic function, we use the rotate-translate-rotate (RTR) function to estimate the number of steps needed to walk along a path to the intermediate goal and to attain the goal direction as shown in Figure 3. The RTR function anticipates a motion where the robot first turns towards the direction of the target, walks to the target, and then turns into the target direction. It is commonly used and well defined. The RTR function is given by

$$\text{RTR}(C, \tilde{G}) = \frac{|\angle(C, \tilde{G}) - C_\theta|}{\theta_{\max}} + \frac{|\tilde{G} - C|}{x_{\max}} + \frac{|\tilde{G}_\theta - \angle(C, \tilde{G})|}{\theta_{\max}} \tag{5}$$

where $\angle(C, \tilde{G})$ denotes the angle of the vector from the center of mass location to the intermediate goal. Each summand is divided by the maximum rotational step size $\theta_{\max}$ and forward step size $x_{\max}$, respectively. In order to involve the shortest path in the heuristic function, we extend the RTR function to a PathRTR function by concatenating RTR motions along the path. Let $P(C, \tilde{G}) = \{(x_i, y_i, \theta_i) \mid i = 0..k\}$ be a path to the intermediate goal where $P_0 = C$ and $P_k = \tilde{G}$ with orientations $\theta_{i>0} = \angle(P_{i-1}, P_i)$. Then, the PathRTR function

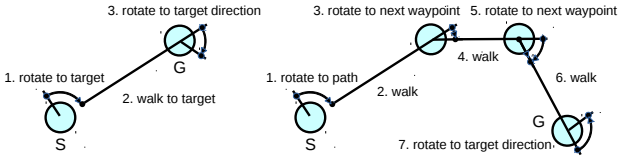
$$\text{PathRTR}(P(C, \tilde{G})) = \sum_{i=0}^{k-1} \text{RTR}(P_i, P_{i+1}) \tag{6}$$

is obtained by summing up the RTR functions of the sections of the path. The PathRTR function overestimates the costs of sideways stepping and is, thus, not admissible. However, it converges quickly to a good solution and is thus well suited for an aborted search.

Conclusively, the computation of the heuristic function $h(C)$ comprises the computation of the shortest path $P(C, \tilde{G})$ and the PathRTR function over the path. Even though this makes the opening of a state more costly than just the computation of a Euclidean distance, it also drastically reduces the number of states that need to be opened so that in total, less computation time is spent.

G. Shortest Path Computation

Since the shortest path needs to be computed many times during an A* search, the computational performance of the shortest path finder plays a crucial role. We considered two different approaches to shortest path planning that are based on the different environment representations generated by our perception pipeline (cf. Sec. III-C): a geometric map with polygons and a grid map.

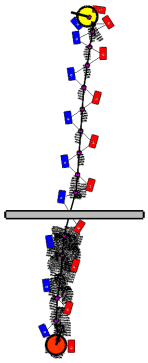


(a) RTR (b) Path RTR

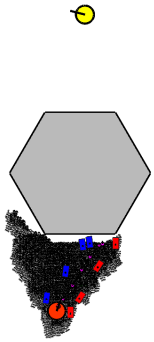
Fig. 3: The rotate-translate-rotate (RTR) heuristic is extended to a PathRTR heuristic by following a path with RTR motions.

For the geometric map, we harness the Minimal Construct algorithm [27] that excels in efficiency when it comes to computing shortest paths in polygonal scenes. The Minimal Construct algorithm constructs only a minimal portion of the Visibility Graph consisting only of the polygons that actually get in the way during the search. For an extra performance boost, we reuse the graph that Minimal Construct discovers for subsequent searches in the same graph, but we have to reset the graph in every frame when the local map changes. The output of Minimal Construct is a smooth path of a minimal number of segments, perfect to be used for PathRTR.

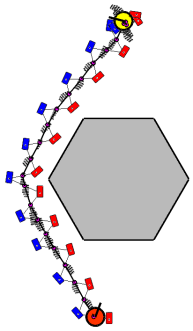
For the grid map, we consider the Dijkstra heuristic [6] as a precomputed lookup table of the shortest paths from all cells of the grid to the target. Since the local map changes in every frame, the Dijkstra table has to be recomputed for every frame and this tolls considerable overhead, but once the table is set up, it can be queried in constant time as many times as needed during a footstep search. If the fast lookup makes up for the overhead, this could be potentially faster than computing shortest paths in large numbers in a polygonal scene. We have implemented a Dijkstra grid that has been specifically tailored to be compatible with the PathRTR heuristic. One issue we had to overcome is that paths extracted from a grid structure typically consist of a hundred little pieces that are oriented in magnitudes of 45 degrees. These jagged paths are unsuited for PathRTR computation. In order to obtain paths of a smoothness comparable to the paths that come out of Minimal Construct, we use the LazyTheta* algorithm [28] to compute the Dijkstra map. LazyTheta* is, to our knowledge, the simplest and fastest any-angle A* derivate for grid structures that, when expanding a new cell, checks the line of sight from the child to the parent of the expanding parent and registers this "grandparent" as parent for the opened child, if the line of sight exists. When the line of sight is broken, the child is assigned the opening neighbour as parent. Despite the extra time for the line of sight checks, the computation of a smooth Dijkstra map is feasible for our small grid resolution of 160x160 cells and the smoothed paths work perfectly for PathRTR. By eroding oversteppable objects, we restored the admissibility of the Dijkstra heuristic with regards to humanoid robots. The paths found by the grid search are, albeit discretization errors, nearly the same as the paths found by the geometric search.



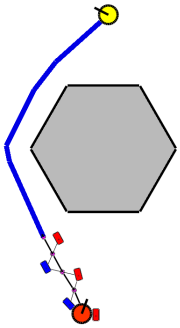
(a) Overstepping with the Euclidean heuristic.



(b) Aborted search with the Euclidean heuristic.



(c) Finished search with the PathRTR heuristic.



(d) Aborted search with the PathRTR heuristic.

	Runtime	Exps.	Steps
a)	6.3 ms	642	18
b)	212 ms	100,000	7
c)	2.49 ms	49	22
d)	1.4 ms	5	4

(e) Runtimes, expansions (Exps.), and planned footsteps in the scenarios shown on the left.

Fig. 4: Aborting A* experiment. The black dots show the states opened by A*. a) An overstepping task holds the search back only by a small amount. b) Without an obstacle-aware heuristic function, an obstructing obstacle keeps the search busy for a long time and aborting leads to a bad result. c) With our PathRTR function, the obstacle is successfully circumvented and the target is found after only a short planning time. d) The result of an aborted search with the PathRTR function consists of fewer steps that are of similar quality as the result found by the search that was allowed to finish. e) The table shows the runtimes, expansions, and planned steps for each scenario.

IV. EXPERIMENTS

A. Aborting A* Experiments

In our first simulated experiment, we show how our PathRTR heuristic function speeds up the A* footstep planner and enables abortability at the same time. Fig. 4 shows pictures of the experiment. In Fig. 4a, a narrow bar that can be overstepped is in the way. 642 expansions in 6.3 milliseconds are needed to solve the task successfully. The bar is narrow enough to be eroded from the path search map and so the shortest path is a straight line from start to target. The heuristic has only minor to no influence here. The footstep A* automatically figures out how to place the steps in order to step over the bar. While the footstep search decides whether overstepping an obstacle is possible or not based on simple size criteria, it is up to the motion execution to figure out the right motion to step over the obstacle without colliding. Sharing the responsibility between the two layers this way allows the footstep planner to make rapid decisions only about the feasibility of a step without having to know how exactly the step will be made. On the other hand, the walking motion then only needs to be computed for one set of footsteps: the final result of the search.

The obstructing obstacle in Fig. 4b is a much harder case for the footstep planner. When using the Euclidean heuristic, the search does not manage to find a solution even after 100,000 expansions in 212 milliseconds. Bear in mind that including orientation, a three dimensional space needs to be searched. The best solution found is a 7 steps long sequence leading into the obstacle. Clearly, the obstructing obstacle holds the search back way more than the oversteppable beam.

In Fig. 4c, we use our PathRTR heuristic instead of the Euclidean one. The footstep search found a solution after only 49 expansions in 2.49 milliseconds. Guided by the shortest path, it is no problem to circumvent the obstacle.

In Fig. 4d, we show the same experiment again using the PathRTR heuristic, but we abort the search after five expansions. We can see that the returned solution consisting of 4 steps is just as good as the solution found by the finished search in 4c as it leads in the same direction.

B. Quantitative Evaluation

In a statistical experiment, we exhaustively test our method in a number of maps and compare its performance when using the different methods for shortest path finding as introduced in Section III-G. It comes down to an exciting comparison between a precomputed grid representation: Grid Dijkstra, and a geometric method without precomputation: Minimal Construct.

We evaluated both versions in four different maps that are shown in Figure 5. The first map is the floor plan of an apartment with a size of 12x10 meters. The second map is an office building with a size of 30x30 meters. The third map is an outdoor scene with a size of 20x20 meters. The fourth map is a U-shaped room. With the exception of the U room, we sprinkled random clutter into all maps for the footstep planner to deal with. We generated 10,000 collision-free start-goal pairs on all maps and computed all footstep plans with both of the evaluated methods. First, we ran the experiment without a time bound. Only an expansion limit of 100,000 was active. Then, we set a time limit of 18 ms (leaving a gap for the map processing) and ran all examples again. We can draw conclusions from comparing the results of the completed and aborted runs in Fig. 6.

We will move through Fig. 6 from the top left to the bottom right. On the top left, we can see that without a time bound, both versions managed to solve nearly all cases, albeit a very few ones in the apartment map, where extreme cases of clutter made it impossible to find a solution. In the next plot, when the time bound is active, roughly half of the examples could still be finished. Here, we can see that the Grid Dijkstra method falls behind on three maps, even though it does perform well in the apartment map.

Looking at the unbounded runtimes in the next chart, we

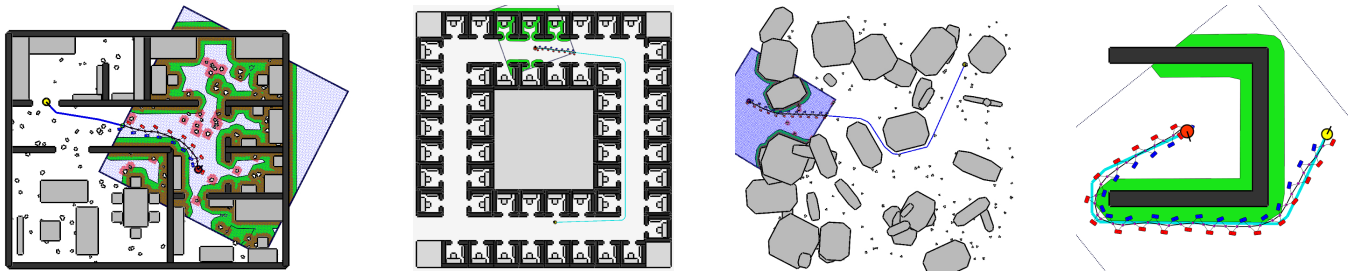


Fig. 5: Artificially generated maps that we used for statistical experiments. Left: a 12x10 meters large apartment (Apt). Left middle: a 30x30 meters large office building (Off). Right middle: a 20x20 meters large outdoor scene (Out). Right: The U room (U).

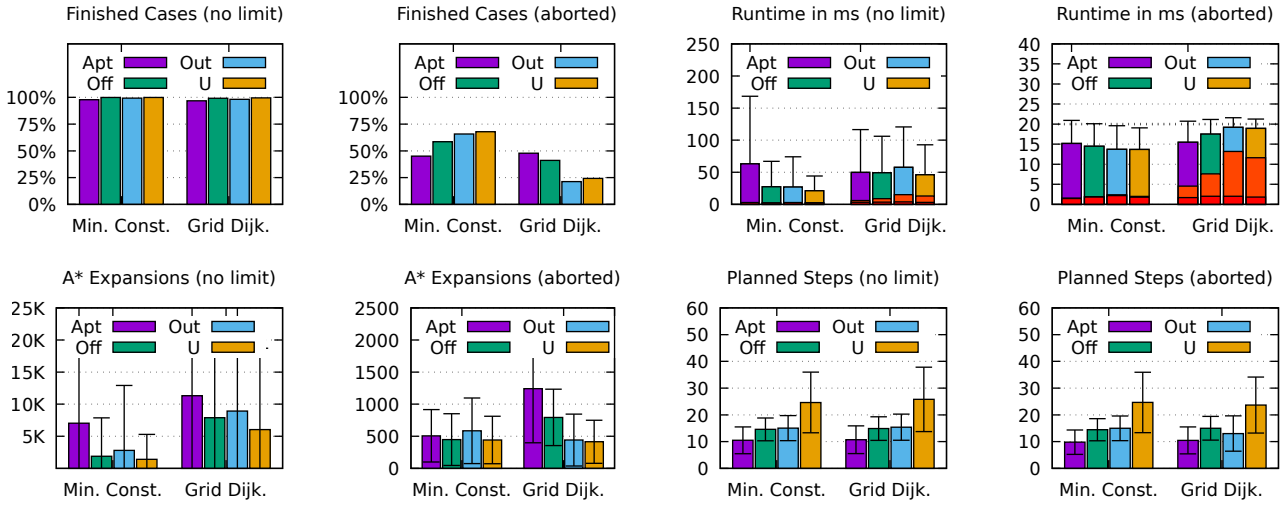


Fig. 6: Charts of the Minimal Construct and the Grid Dijkstra versions of our footstep planner performing on the Apartment map (Apt), the Office map (Off), the Outdoor map (Out), and the U room (U).

can see that the footstep plans typically finish in under 100 milliseconds on all maps. The Grid Dijkstra method needs a little more time to finish than its geometric counterpart even though the portion of time spent on the precomputation, this is marked in orange color in the bar stack, seems rather insignificant. In the next plot, the aborted times are shown. Both methods manage to stay below the 20 milliseconds mark, even though the Minimal Construct method a little more so than the Grid Dijkstra version. We can see that the map computation part shown in red at the bottom of the bar stack is only a fraction of the time needed for the search. The precomputation time for the Dijkstra table shown in orange, however, is now quite significant. Notably, this precomputation time is much smaller on the apartment map where often a large portion of the local map is blocked space or outside of the global map and, thus, less cells need to be processed. We measured the runtimes on a laptop with an Intel Core5 i5-7200U 4 x 2.50GHz CPU.

In the next plot, the number of expansions of the unaborted runs shows that when using the grid, significantly more steps need to be expanded to find the target. This is interesting since the geometric and the grid methods compute almost the same shortest paths. The only difference between them is the discretization of the Dijkstra table, which is enough to disturb the search this much. The number of expansions of the aborted searches in the next plot show that the precomputed Dijkstra grid profits from the faster lookups and manages to expand more steps in the same amount of time as the graph methods, even though this does not help solving more cases.

Finally, if we then look at the number of steps in the resulting footstep plans in the last two charts, it comes as a surprise that the average lengths of the resulting plans are nearly the same whether the search is aborted or not. Consequently, the quality of the plans found with a bounded-time search cannot be far behind the completed plans.

Additionally, we provide a demonstration video[1] of our algorithm replanning during walking at a frequency of 50 Hz.

V. CONCLUSIONS

In conclusion, we proposed a fast footstep planner that is capable of quickly computing footstep plans in a local map around the robot with a guaranteed bound on the runtime. This way, footstep planning becomes feasible even for a low-level motion controller running at 50 Hz. We were able to boost the speed of footstep planning using a novel PathRTR heuristic function combined with the shortest path to the goal, which not only speeds up the search in the presence of obstacles, but also allows us to abort an A* search prematurely with a good solution. In future work, we are planning to extend our footstep planner with Capture Step capabilities [29] that are directly included in the footstep plan. We are also planning to account for moving objects and to extend our planning capabilities to 3D.

[1] Video: https://youtu.be/EI9mHx0uxLU

REFERENCES

[1] Robert J. Griffin, Georg Wiedebach, Stephen McCrory, Sylvain Bertrand, Inho Lee, and Jerry Pratt. Footstep planning for autonomous walking over rough terrain. In *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2019.

[2] J. Kuffner, K. Nishiwaki, S. Kagami, and M. Inaba. Footstep planning among obstacles for biped robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2001.

[3] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Online footstep planning for humanoid robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2003.

[4] J. Chestnutt, M. Lau, K.M. Cheung, J. Kuffner, J.K. Hodgins, and T. Kanade. Footstep planning for the Honda ASIMO humanoid. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2005.

[5] Y. Ayaz, K. Munawar, M.B. Malik, A. Konno, and M. Uchiyama. Human-like approach to footstep planning among obstacles for humanoid robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2006.

[6] Armin Hornung, Andrew Dornbush, Maxim Likhachev, and Maren Bennewitz. Anytime search-based footstep planning with suboptimality bounds. In *Proc. of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2012.

[7] Armin Hornung, Daniel Maier, and Maren Bennewitz. Search-based footstep planning. In *Proc. of the ICRA Workshop on Progress and Open Problems in Motion Planning and Navigation for Humanoids*, 2013.

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

[9] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613 – 1643, 2008.

[10] Maxim Likhachev and Anthony Stentz. R* search. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 344–350, 2008.

[11] Armin Hornung and Maren Bennewitz. Adaptive level-of-detail planning for efficient humanoid navigation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[12] Johannes Garimort, Armin Hornung, and Maren Bennewitz. Humanoid navigation with dynamic footstep plans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[13] S. Koenig and M. Likhachev. D^* Lite. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2002.

[14] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the Atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

[15] Dimitrios Kanoulas, Alexander Stumpf, Vignesh Sushrutha Raghavan, Chengxu Zhou, Alexia Toumpa, Oskar von Stryk, Darwin G. Caldwell, and Nikos G. Tsagarakis. Footstep planning in rough terrain for bipedal robots using curved contact patches. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2018.

[16] Alexander Stumpf, Stefan Kohlbrecher, David C. Conner, and Oskar von Stryk. Supervised footstep planning for humanoid robots in rough terrain tasks using a black box walking controller. In *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2014.

[17] S. Feng, X. Xinjilefu, C. G. Atkeson, and J. Kim. Optimization based controller design and implementation for the Atlas robot in the DARPA robotics challenge finals. In *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2015.

[18] S. Feng, X. Xinjilefu, C. G. Atkeson, and J. Kim. Robust dynamic walking using online foot step optimization. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2016.

[19] R. Deits and R. Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *Proc. of the IEEE/RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2014.

[20] Arne-Christoph Hildebrandt, Robert Wittmann, Felix Sygulla, Daniel Wahrmann, Daniel Rixen, and Thomas Buschmann. Versatile and robust bipedal walking in unknown environments: real-time collision avoidance and disturbance rejection. *Autonomous Robots*, 43, 2019.

[21] A. C. Hildebrandt, D. Wahrmann, R. Wittmann, D. Rixen, and T. Buschmann. Real-time pattern generation among obstacles for biped robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2015.

[22] Daniel Wahrmann, Arne-Christoph Hildebrandt, Tamas Bates, Robert Wittmann, Felix Sygulla, Philipp Seiwald, and Daniel Rixen. Vision-based 3d modeling of unknown dynamic environments for real-time humanoid navigation. *Int. J. Humanoid Robotics*, 16(1), 2019.

[23] Arne-Christoph Hildebrandt, Robert Wittmann, Daniel Wahrmann, Alexander Ewald, and Thomas Buschmann. Real-time 3d collision avoidance for biped robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2014.

[24] Nicolas Perrin, Olivier Stasse, Léo Baudouin, Florent Lamiraux, and Eiichi Yoshida. Fast humanoid robot collision-free footstep planning using swept volume approximations. *IEEE Transactions on Robotics*, 28(2), 2012.

[25] P. Karkowski, S. Oßwald, and M. Bennewitz. Real-time footstep planning in 3d environments. In *Proc. of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2016.

[26] Marcell Missura, Arindam Roychoudhury, and Maren Bennewitz. Polygonal perception for mobile robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2020.

[27] Marcell Missura, Daniel D. Lee, and Maren Bennewitz. Minimal construct: Efficient shortest path finding for mobile robots in polygonal maps. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 2018.

[28] A. Nash, S. Koenig, and Craig A. Tovey. Lazy Theta*: Any-angle path planning and path length analysis in 3D. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2010.

[29] Marcell Missura, Maren Bennewitz, and Sven Behnke. Capture steps: Robust walking for humanoid robots. *Int. J. Humanoid Robotics*, 16(6):1950032:1–1950032:28, 2019.