Proceedings of the 9th International Workshop on Robot Motion and Control, Wasowo Palace, Wasowo, Poland, July 3-5, 2013

A comparison of Search-based Planners for a Legged Robot

Muhammad Asif Arain[1], Ioannis Havoutis[2], Claudio Semini[2], Jonas Buchli[3] and Darwin G. Caldwell[2]

***Abstract*— Path planning for multi-DoF legged robots is a challenging task due to the high dimensionality and complexity of the planning space. We present our first attempt to build a path planning framework for the hydraulic quadruped - *HyQ*. Our approach adopts a similar strategy to [1], where planning is divided into a task-space and a joint-space part. The *task-space planner* finds a path for the center of gravity (COG) of the robot, while then the *footstep planner* generates the appropriate footholds under *reachability* and *stability* criteria. Next the *joint-space planner* translates the task-space COG trajectories into robot joint angles. We present a comparison of a set of *search-based* planning algorithms; Dijkstra, A* and ARA*, and evaluate these over a set of given terrains and a number of varying start and end points. All test runs support that our approach is a simple yet robust solution. We report comparisons in path length, computation time, and path cost, between the aforementioned planning algorithms.**

Keywords: ***Quadruped Locomotion, Path Planning, Search Algorithms.***

I. INTRODUCTION

Traditionally, robot locomotion has been divided into two main categories of systems: wheeled and legged. Relatively smooth and continuous surfaces are easily addressed by wheel robots. Legged locomotion on the other hand, is required when the robot needs to traverse a variety of environments, including very rough, unstructured and discontinuous surfaces.

Path-planning is a crucial part of the navigation process. It involves searching and finding a path from a start to a goal position in the environment of navigation. The high dimensionality of the state-space of quadruped robots makes path planning a complex task. Related literature consists of a number of different approaches for path planning with quadruped robots. Most of them are combinations of off-line and on-line phases, and a common theme is a moving horizon solution. Kalakrishnan et al. [1] presented a planning procedure that computes a COG path off-line and appropriate footholds on-line. In this paper we adopt a similar solution, that uses off-line planning and generates a locally optimal plan given the initial state of the robot and the environment. Furthermore, this paper presents a comparison between a set of search-based planning algorithms for a legged quadruped; the HyQ robot.

[1]M. A. Arain is with Mobile Robotics & Olfaction Lab, AASS Research Center, Örebro University, 70182 Örebro, Sweden <asif.arain@oru.se>

[2]I. Havoutis, C. Semini, and D. G. Caldwell are with Department of Advanced Robotics, Istituto Italiano di Tecnologia (IIT), 16163 Genova, Italy <ioannis.havoutis>, <claudio.semini>, <darwin.caldwell>@iit.it

[3]J. Buchli is with Agile & Dexterous Robotics Lab, ETH, 8092 Zurich, Switzerland <buchlij@ethz.ch>

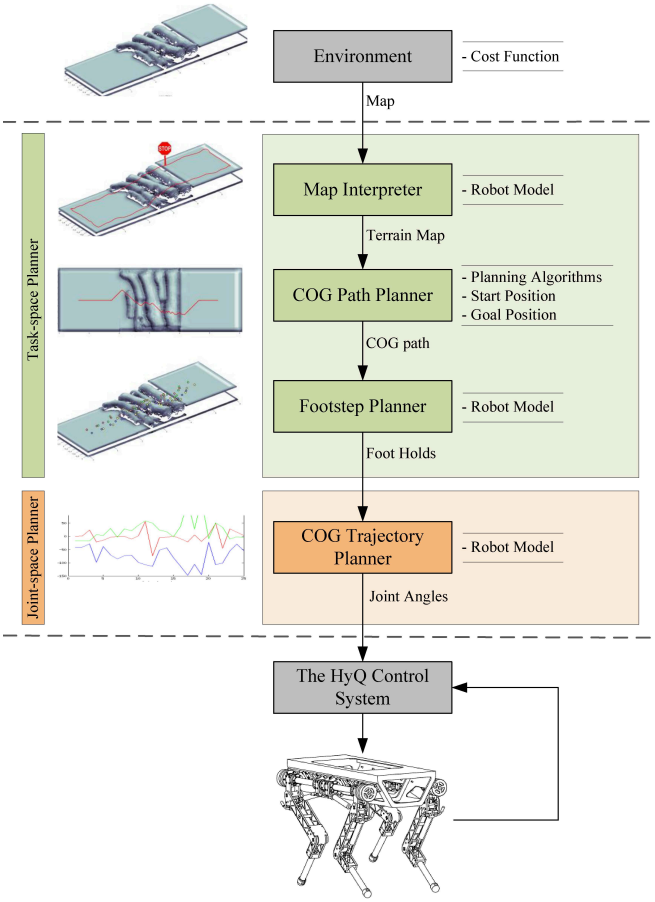


Fig. 1. An overview of the HyQ path planning framework. Details are given in the text.

Our platform, HyQ [Fig. 2], is a 12-DoF fully torque-controlled hydraulically and electrically actuated quadruped robot comparable in size to a goat and roughly 70kg, e.g. an Alpine Ibex. HyQ is capable of highly dynamic locomotion, it is 1m long and 0.5m wide and stands 1m high with the legs fully stretched [2].

Our previous work focused on dynamic locomotion, mainly trotting, using active compliance and low-level feedback for stabilization [3], [4]. In this paper we report our first steps towards a statically walking controller, capable to negotiate very rough terrain with possibly intermittent support areas.

II. RELATED WORK

Kalakrishnan et al. [1] developed a planner for the LittleDog robot. They used a template learning algorithm to learn footholds for the robot with a reward function. This

map was used to find an off-line, approximate, body path and generate footholds on-line, in a moving horizon manner. They optimized the robot pose to find a solution with maximum kinematic reachability. Kotler et al. [5] generated a high level planning solution by extracting the features from the map and finding a body path and footholds, along side a low level planning solution for robot trajectories. Zucker et al. [6] tuned a heuristic function based on a Dubins car model for overall path planning, and optimized the generated trajectories using CHOMP: a local swing trajectory optimization procedure. Hart et al. [7] presented the heuristic-based search for minimum path cost. Likhachev et al. [8] presented ARA*; an extended version of the A* search algorithm, that can be used in a number of applications including path planning.

III. THE PLANNER

Dealing with the full planning problem is too complex. This way we divide path planning into two parts; *task space* or *high level planning* and *joint space* or *low level planning*. Dividing overall task into parts helps to simplify the problem and ensures a computationally low yet optimized solution. The idea is to develop an iterative process which runs until a solution is found or report false if no solution exists. Dividing the full path planning task into state-space and joint-space subparts is a common approach. For example Kalakrishnan et al. [1], Kotler et al. [5] and Zucker et al. [6], all developed similar path planning procedures for the *LittleDog* quadruped robot.

An overview of the planner architecture that we developed is presented in Figure 1. The information or data a process requires other than outcomes from previous modules, are listed on the right. The output of each process module appears below each module. Graphs on the left of each process module give a visual example of the results. Processes filled with gray color at the top and bottom of the architecture do not fall into the scope of our planner. The planner is divided into task space planning (in three steps, in green) and joint space planning (in orange). The task space-planner consists of the map interpretation step, the COG path planning step and the footstep planning step, while the joint-space planner performs the COG trajectory planning step. The complete planning pipeline is detailed in Algorithm 1.

Given an environment map, a cost function, a start and a goal position and the robot model, Algorithm 1 will generate a path for the robot, both in the task space and the joint space. First the $Terrain$ procedure interprets the map into a meaningful form for the planner, then the $COGPathPlanner$ procedure produces a COG path for the robot, next, the $FootstepPlanner$ procedure generates optimal footholds with respect to the previously generated COG path, and finally, the $COGTrajectoryPlanner$ procedure generates the joint angle trajectories that the robot needs to follow to realize the planned path.

A. Map Interpretation

The *Terrain* procedure translates a map into a meaningful form for the planner. It performs a *cost transformation* to each node in the map with respect to robot kinematic model, identifies '*no-go*' unsafe regions for the robot, and produces a *low resolution* map for COG path planner.

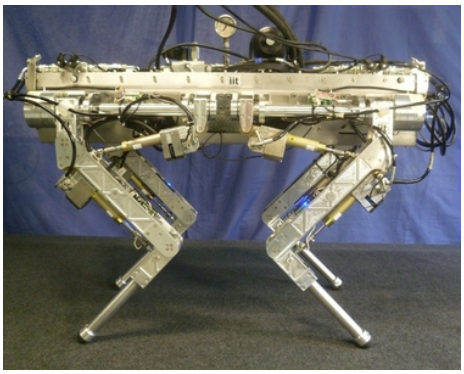
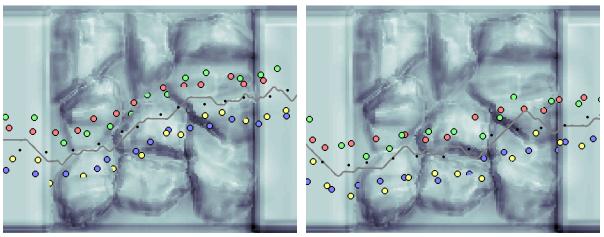


Fig. 2. The hydraulically actuated quadruped robot - HyQ. It has 12 degrees of freedom and its size is comparable to a goat. HyQ is designed for highly dynamic behavior, e.g. trotting and jumping

The cost assigned to each node in the map is given *a priori* and is based on the foothold placement conditions according to the node in consideration. For the planner to search for an optimal path of the robot COG, the cost needs to be transformed in accordance to the robot geometry. The updated cost value for the node under the robot COG projection is the mean of the cost assigned to all nodes in the workspace of each leg of the robot. Searching a $COGPath$ with updated cost values will help to find better footholds in the footstep planning phase. Figure 3 presents a comparison between these two cost scenarios. The $cost$ procedure in Algorithm 1 translates the cost associated with each node in accordance with the robot model.

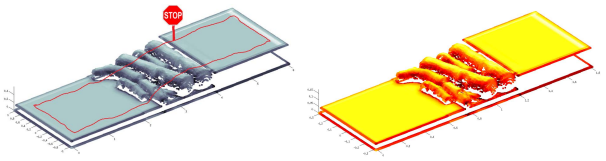
If there exists a low cost path near the boundary of the map then the footholds generated for such path may go out of the map area. To avoid such situations, boundary regions equal to the half of the robot length are termed as '*no-go areas*' [Figure 4]. The $NoGoArea$ procedure in the Algorithm 1 makes this region not available for the COG path search.

The resolution of the map for a 3×1.5 m^2 area is 366×128 cells. Considering robot dimensions and the map size, it doesn't really require such high resolution for COG path planning. The $Resolution$ procedure rebuilds the map and



(a) (b)

Fig. 3. Comparison of generated footholds with cost transformation. COG path is in gray line and footholds are in red, green, blue and yellow dots on Rock rough terrain. (a) without cost transformation, (b) with cost transformation.



(a) (b)

Fig. 4. Map interpretation (a) identified *"No-Go"* area with red color tape for the robot, (b) low resolution map.

halves the original map resolution.

B. COG Path Planner

The *COG path planner* generates a *low cost* path in *task-space*. The COG path is an approximate path that helps to find overall *better footholds*. The search algorithms we evaluated within our framework are Dijkstra, A* and Anytime Repairing A* (ARA*). Below we briefly present an overview of these search algorithms.

Dijkstra: Dijkstra can be viewed as the search algorithm ground truth as it finds the lowest cost path by definition. We used this algorithm as a reference for the rest. It is a uniform-cost search and *always* produces the lowest cost path, if one exists. For all the nodes in the map, it assigns a lowest cost from the start node to each node in the map. It uniformly explores the search space, therefore, computationally it is very costly.

A*: A* combines a *heuristic* value with a *cost* value for each node in the map and therefore, results into a procedure that explores fewer nodes to reach a given final node. Expediting the search with a heuristic function results into less computations compared to searching with uniform-search. When the heuristic is consistent and admissible then A* always produces the lowest cost path [7].

The Heuristic Function: The addition of a heuristic function makes A* search faster than uniform-cost search. The heuristic function evaluates a *cost-to-go*; an estimated cost from any node in the graph to the goal node. A true heuristic (perfect lowest cost) is hard to estimate before reaching the goal. If the heuristic value is the same as the true cost then A* will only expand the best path nodes. A heuristic function is said to be admissible if it never overestimates the cost of reaching the goal. If the heuristic is *admissible* then A* explores less nodes than Dijkstra and still guarantees the lowest cost path. On the other hand, if the heuristic is greater than the true cost then A* runs much faster but does not guarantee a lowest cost solution. We used the Euclidean distance as a heuristic estimate as it is a popular choice for real world path planning problems.

ARA*: ARA* is an anytime version of A*. It uses the concept of adapting the heuristic function to return estimates greater than the true cost to go [8]. It starts searching with a high inflation factor (greater than unity) multiplying the heuristic value to produce a quick solution, and then as time allows reduces the inflation factor until unity, converging to an optimal solution as computed by A*. Every time ARA* reduces the inflation factor and starts searching again, it uses the previous information.

Algorithm 1 $ThePlanner$

require: $Map, f_{cost}, Start, Goal, Model_{Robot}$
return: $Path_{TaskSpace}, Path_{JointSpace}$

Main()
1: $[TerrainMap] \leftarrow Terrain()$
2: $[COGPath] \leftarrow COGPathPlanner()$
3: $[FootHolds] \leftarrow FootstepPlanner()$
4: $[JointAngles] \leftarrow COGTrajectoryPlanner()$
5: publish results

Terrain()
$cost()$
1: $cost(i) \leftarrow \left\{ f_{cost}(model_{env}, model_{robot}), \forall i \subset Map \right\}$
2: $cost(i_{COG}) \leftarrow \left\{ \sum cost(i') : i' \subset WS_{leg}, \forall Leg_{robot} \right\}$
3: update $cost(i)$ as $cost(i_{COG})$

$NoGoArea()$
1: $NoGoArea \leftarrow \left\{ \frac{1}{2} \times Length_{robot}, \forall boundry_{map} \right\}$
2: $\forall i \subset NoGoArea$ remove from Map

$Resolution()$
1: $TerrainMap \leftarrow \frac{1}{2} \times Resolution(Map)$

COGPathPlanner()
1: $COGPath \leftarrow Algorithm(TerrainMap, Start, Goal, \epsilon)$

FootstepPlanner()
1: **while** all $footHolds(LH, LF, RH, RF)$ not validated **do**
2: **for** $stepSize$ & $robotHeight$ **do**
3: $\theta_{next} \leftarrow$ Heading towards $COGnode(stepSize)$
4: $footStep_{next} \leftarrow (stepSize, \theta_{next})$
5: $Reachability()$
6: $Stability()$
7: **if** $reachability$ & $stability$ are validated **then**
8: break the loop
9: **end if**
10: **end for**
11: **end while**

$Reachability()$
1: $d \leftarrow distance(footStep_{next}, WorkSpace_{leg})$
2: $validated \leftarrow Reachability \iff d \leq k$

$Stability()$
1: $Region_{stable} \leftarrow SupportTriangle(Legs_{stance})$
2: **if** $projection(COG_{robot}) \subset Region_{stable}$ **then**
3: $validated \leftarrow Stability$
4: **end if**

COGTrajectoryPlanner()
1: $[X_{LH}, X_{LF}, X_{RH}, X_{RF}] \leftarrow FootstepPlanner$
2: $JointAngles \leftarrow IKP(X_{COG}, X_{LH}, X_{LF}, X_{RH}, X_{RF})$

This way every search iteration is computationally less expensive than searching from scratch [9]. In each search, ARA* lists the inconsistent nodes that show more than one time change in their lowest cost. ARA* uses this information and starts by looking into the inconsistent nodes in the next search and hence produces an optimal solution that is computationally less expensive.

C. Footstep Planner

To realize the computed COG path, the planner needs to generate appropriate footholds for each leg of the robot. The *FootstepPlanner* procedure adopts an iterative algorithm for searching for the largest footstep that falls under the reachable area of each leg. For each iteration, the next four footholds are generated; one for each leg. There are two criteria that needs to be satisfied before a foothold is selected, *reachability* and *stability*.

The procedure starts validating both parameters with a maximum footstep size of 0.375m and a height of 0.6m for the robot. The footstep size and the robot height are changed by 10% with each iteration until all footholds are validated for reachability and stability. The planner uses equal sized footsteps for all the legs. Although, this approach may limit the capabilities of the planner for selecting relatively better footholds that might exist, on the other hand, this reduces the complexity.

Reachability: A foothold is reachable if given the COG position and orientation of the robot, the foothold is within the workspace of the leg in question.

Stability: We used a predefined crawl gait that uses static stability with the traditional definition: the robot is statically stable if its COG projection is within the support triangle of stance legs. The sequence of leg transfer that follows a predefined crawl gait, cycles through the left hind leg (LH), the left front leg (LF), the right hind leg (RH), and the right front leg (RF).

Figure 5 presents an example of a produced path planning solution in task-space (top-down view).

D. COG Trajectory Planner

From the task-space solution, the planner knows the position of footholds [Figure 5] and the intermediate translations of footsteps. Along with the robot body position (x,y,z) and orientation (pitch, yaw, roll), we used an inverse kinematic procedure to calculate joint angles that correspond to feet position at each time step. This results to a complete plan from start to goal position.

IV. TEST CYCLE

To evaluate the set of search-based planning algorithms, we developed a test cycle with four different terrains: *logs, rocks, round rocks* and *slopes*, shown in Figure 6. A map is a combination of three terrains, two flat terrains side-by-side and one rough terrain in the middle. The start positions for the path planning query are at one of the flat terrains and goal positions are on the other. Cost assigned to each node in the map is feature-based (not a distance).

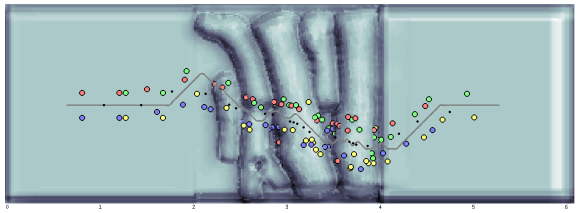
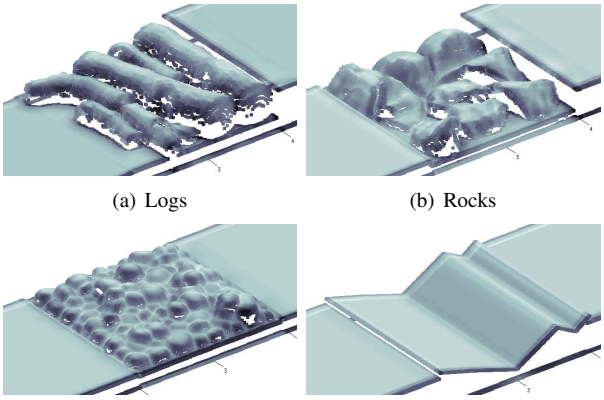


Fig. 5. One *Task-Space path planning* solution. COG path is in gray line, footholds for left hind, left front, right hind and right front legs are in red, green, blue and yellow circles respectively, and robot COG projection after each cycle is in black dot.



(a) Logs (b) Rocks

(c) Round Rocks (d) Slopes

Fig. 6. Types of rough terrain.

The robot plans a path for a total of 25 combinations of start and end points for each rough terrain scenario. A cycle of four rough terrain models result into 100 planning queries for each algorithm. We evaluated the search algorithms in terms of *path cost, computation time*, and *path length* for the total of 300 trials.

A. Path Cost

The path cost is the total cost for the COG path of the robot in terms of the cost function. The accumulated path cost of each algorithm is averaged over 25 trials and shown in Figure 7. The bar shows mean outcome for each algorithm while the error bars show the dispersion between trials.

By definition, the average path cost for each algorithm must be the same only if the heuristic function is admissible. The heuristic cost function that we used is not guaranteed to be admissible and consistent. This way the heuristic-based algorithms are expected to produce paths that are not globally optimal. For that reason we used Dijkstra as a ground truth algorithm since it does not use a heuristic function.

For the *logs* rough terrain, Dijkstra has found an average cost value of 964.99. Averaged cost for the A* is 1008.82; 4.54% higher than Dijkstra. ARA* has found 0.53% higher

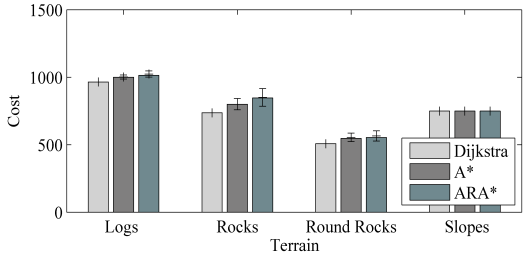


Fig. 7. Average path cost of test cycle.

cost then A*. Ideally, ARA* should have improved its cost value to the A*, but due to inadmissible heuristic, it did not reach the exact solution as A*. For the *rocks* rough terrain, the average cost of Dijkstra is 737.2. A* has found an average path cost of 800.6; 8.60% higher cost than Dijkstra. Average cost for ARA* is 846.97 that is 5.79% higher than A*. For the *round rocks* rough terrain trials show an average cost path of 508.1 for Dijkstra, 546.38 for A*, and 554.06 for ARA*. Trials for the *Slopes* rough terrain have produced a solution of an average value of 749.37, 749.42 and 749.43 for Dijkstra, A* and ARA* respectively. All three algorithms show negligible dispersion. It is because the cost value over all the nodes in the *slopes* terrain is uniformly distributed along with slope.

B. Computation Time

Computation time is the total time taken by the algorithm to compute a path. Our evaluations were performed on commodity hardware, using a simple Matlab implementation. Note that a C/C++ implementation is more suitable for search based algorithms as more efficient data structures can be used, providing a significant benefit in terms of computation time.

For the *logs* rough terrain, the average computation time of Dijkstra over 25 trials is 705.52 seconds. A* takes only 81.67 seconds; more than 8 times less than Dijkstra. ARA* by default is faster than A*. It takes 67.46 seconds, which is about 1.21 times less than searching with A*.

Trials for the *rocks* rough terrain have taken an average computation time for Dijkstra, A*, and ARA* of 705.40, 54.11 and 62.75 seconds respectively. Computation time taken by A* is more than 13 times less than Dijkstra. ARA* should have taken less time than A* but it took more than 1.15 times more than A*. Dispersion with A* is little higher than ARA*, this means that for some of the trials the computation time of A* is almost same as ARA*. Results for the *round rocks* are almost similar as of the *rocks* terrain. A* takes more than 15.49 times less computation time than Dijkstra. Dijkstra computed a COG path in 656.20 seconds and A* in 42.34 seconds. ARA* still takes a little higher average computation time than A*, i.e. 58.39 seconds. Tests for *slopes* terrain has produced expected results. Dijkstra, A* and ARA* take 707.30, 84.30 and 63.17 seconds respectively.

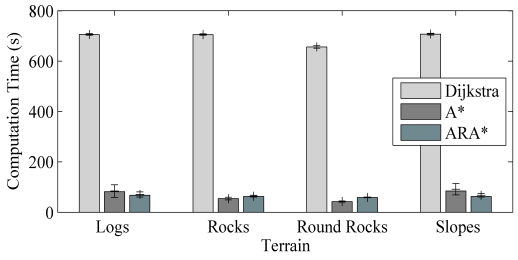


Fig. 8. Computation time of test cycle.

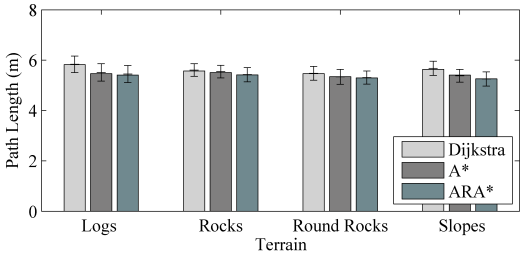


Fig. 9. Average path length of test cycle.

C. Path Length

The path length evaluation criterion for each algorithm compares the total path length, in meters, from the start to the end position. However, the shortest length path does not mean a lowest cost path or otherwise. Heuristic-based search should have a path of equal or lesser length than uniform-cost search. We evaluated how close the path length of the heuristic-cost based search is by comparing against the solution that Dijkstra computes.

Results in Figure 9 show that there is no significant difference in length for the paths computed by each algorithm. Note that the lowest cost does not imply shortest length, in fact for most of our trials the lower cost paths are the ones that avoid areas of the terrain that are evaluated as rough and dangerous for the stability of the robot. For the *logs* type of rough terrain, an average path length of each algorithm is 5.82, 5.46 and 5.40 meters for Dijkstra, A* and ARA* respectively. Dijkstra has produced paths that are slightly longer that the rest of the algorithms. Average path length for the *rocks* is 5.57, 5.51 and 5.41 meters for Dijkstra, A* and ARA* respectively. On the *round rocks* rough terrain, Dijkstra and A* have a difference of 0.12 meters. The path that ARA* produces have almost same length as A*. Results of the *slopes* rough terrain have introduced a little more difference between Dijkstra and A* than previous search with rough terrain of *round rocks*. Dijkstra path is 0.23 meters more lengthy than A*. ARA* is less lengthy by 0.15 meters as of A*.

D. Weighting factor for wA*

ARA* uses wA* in each search. It starts with an weighting/inflation factor greater than unity. If the weighting factor is too high then wA* will turn into *Best-First* search [10] and will produce a solution that is highly sub-optimal. On the other hand, if this weighting factor is too small then wA* will result an expensive solution in terms of computation. Therefore, the weighting factor of wA* is a trade off between a highly sub-optimal and a computationally high cost solution.

To find the best weighting factor for the first search of ARA*, we experimented with a set of weights, $w = [5, 4, 3, 2, 1]$. The planner runs wA* for the 9 trials on each terrain, this is a total of 180 test runs that were computed off-line in order to choose a suitable weighting factor.

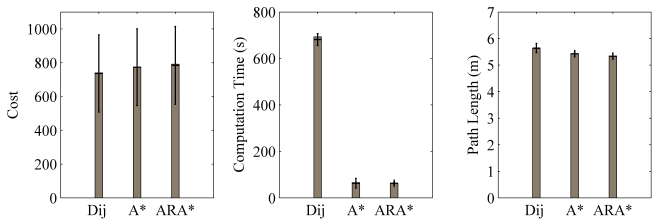


Fig. 10. Net results for the evaluation criteria of, (left) path cost, (middle) computation time, and (right) path length.

Averaged results of the test runs for each weighting factor are summarized bellow. Setting $w = 1$ (turning wA* into A*) results into the lowest path cost i.e 777.34. For $w = 2$, the path cost increased by 1.69% compared to A*. For $w = 3$, the increase in cost was around 2.6%. For a weighting value of $w = 4$ increased the cost by 25.5 or 3.28% of A*. There is no significant increment in the path cost from $w = 4$ to onwards.

The evaluation of wA* in terms of *computation time* is the the factor that we weighted more. For $w = 1$ (hence wA* turns into A*), is the most expensive computation solution. A weighting factor of $w = 2$ is 38.26% less expensive than A*. For $w = [3, 4, 5]$ there was no significant reduction in computation time. Path length comparison does not shown a big difference among weighting factors of greater than unity.

The most efficient weighting factor in terms of computation time (a decrease by 38.26% to A*) and path cost (an increase of only 1.69% to A*) was $w = 2$. On the other side $w = 3$ resulted in similar improvement in terms of computation time and did not increase in the path cost significantly, compared to $w = 2$ (an increase of 2.6% to A*). Average computation time for $w = 2$ has high dispersion and results into some inconsistency. Considering the above we have selected to start ARA* by setting the weighting factor to $w = 3$.

V. EVALUATION

An overview of the results from the test cycle are available in Table I. Dijkstra with uniform-cost search achieved the lowest path cost. A* and ARA* are slightly more expensive than Dijkstra in terms of path cost. Searching with the heuristic-based algorithms was not overall much more expensive in terms of path cost. ARA* and A* did not converge to the same path cost solutions. This is due to the lack of a guarantee of admissibility and consistency of the heuristic cost function used.

The comparison in terms of computation time what sets apart Dijkstra and the heuristic-based algorithms. Dijkstra takes approximately ten times more time to compute a path, since this would be the globally optimal with respect to the cost function. A* and ARA* are very efficient compared to Dijkstra, though the paths returned, in our case, are deemed to be suboptimal due to the inadmissibility of the heuristic function. ARA* in general is faster than A* even after exhausting the weighting factor to unity.

TABLE I
COMPARISON

	Dijkstra	A*	ARA*
Path Cost	739.92	774.31	797.55
Computation Time (seconds)	693.61	65.61	62.94
Path Length (meters)	5.62	5.43	5.34

All algorithms share almost the same path length. Dijkstra produced an average more lengthy paths as the cost function depends more on the local characteristics of the rough terrain in question than the length of the path. This difference had little impact after the planner translates the path into the actual robot footholds.

VI. CONCLUSION

We have presented an off-line path planning framework for the quadruped robot HyQ. Our framework divides the path planning problem into high-level planning in the state-space of the robot, and into low level-planning for the joint-space of the robot. We compared a set of search algorithms for the computation of the high-level path that the robot needs to follow, in order to reach a given goal. Overall we presented a simple yet robust approach to the path planning problem in the context of a quadrupedal robotic platform.

In future work we aim to experiment with other anytime variants of systematic search (e.g D*, AD*) and probabilistic search approaches (e.g R*, RRT). We are currently also working on the real-time execution aspect of our planning framework and aim to utilize a receding horizon based approach.

REFERENCES

[1] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, Learning, Planning, and Control for Quadruped Locomotion over Challenging Terrain, International Journal of Robotics Research 2010 (2):236-285.
[2] C. Semini, HyQ - Design and Development of a Hydraulically Actuated Quadruped Robot, PhD Thesis, Istituto Italiano di Tecnologia, Genova, Italy, April 2010.
[3] I. Havoutis, C. Semini, J. Buchli and D. G. Caldwell, Quadrupedal trotting with active compliance, IEEE International Conference on Mechatronics (ICM), 2013.
[4] V. Barasuol, J. Buchli, C. Semini, M. Frigerio, E. R. De Pieri, and D. G. Caldwell, A Reactive Controller Framework for Quadrupedal Locomotion on Challenging Terrain, IEEE International Conference on Robotics and Automation (ICRA), 2013.
[5] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng, A Control Architecture for Quadruped Locomotion Over Rough Terrain, International Conference on Robotics and Automation (ICRA), 2008.
[6] M. Zucker and J. A. (Drew) Bagnell, C. Atkeson, and J. Kuffner, An Optimization Approach to Rough Terrain Locomotion, IEEE Conference on Robotics and Automation, May 2010.
[7] P.E. Hart, N.J. Nilsson, and B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics, July 1968, Vol. 4, No. 2, pg:100-107, ISSN:0536-1567.
[8] M. Likhachev, G. Gordon, and S. Thrun, ARA*: Anytime A* with Provable Bounds on Sub-Optimality, Advances in Neural Information Processing Systems 16 (NIPS), MIT Press, Cambridge, MA, 2004.
[9] M. Likhachev, Search-based Planning for Large Dynamic Environments, PhD Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA, September 2005.
[10] S. M. LaValle, Planning Algorithms, Cambridge University Press, Cambridge, U.K. 2006.