GaitMesh: controller-aware navigation meshes for long-range legged locomotion planning in multi-layered environments

Martim Brandão, Omer Burak Aladag and Ioannis Havoutis

***Abstract*— Long-range locomotion planning is an important problem for the deployment of legged robots to real scenarios. Current methods used for legged locomotion planning often do not exploit the flexibility of legged robots, and do not scale well with environment size. In this paper we propose the use of navigation meshes for deployment in large-scale, potentially multi-floor sites. We leverage this representation to improve long-term locomotion plans in terms of success rates, path costs and reasoning about which gait-controller to use when. We show that NavMeshes have higher planning success rates than sampling-based planners for a fraction of the construction time (e.g. 2x success rate with 60x lower construction time), as well as finding 30% lower-cost paths, while this performance gap further increases when considering multi-floor environments. We present both a procedure for building controller-aware NavMeshes and a full navigation system that adapts to changes to the environment. We demonstrate the capabilities of the system in simulation experiments and in field trials at a real-world oil rig facility.**

I. Introduction

Legged robots are capable of locomotion using a variety of gaits. For example, quadruped robots can switch their locomotion mode to walk, trot, bound, etc. according to the terrain at hand. This diversity of ways to navigate environments renders legged robots unmatched for overcoming varied terrain. However, it comes at the cost of reasoning about terrain features and choosing the appropriate gait to use for each particular area. Even though the choice of gait-controller has been included in recent locomotion planning methods [1], it is still not clear which map representations and planning methods are most suitable to the task—especially for large-scale environments.

In this paper we present our approach to efficient and reliable map representation for long-range legged robot locomotion planning, building on navigation meshes (NavMeshes) [2], [3], [4]. Our approach was developed having in mind applications of inspection and monitoring of large industrial facilities, e.g. power plants, offshore wind and oil & gas platforms, or nuclear facilities. In this context we assume an approximate map of the environment to be known in advance or that the robot can be teleoperated to build up a map of the facility before autonomous deployment.

NavMeshes are popular map representations within computer game AI [5], where they are used to plan paths for agents over large-scale environments such as buildings or open-worlds. Such large-scale planning methods are important for autonomous long-term robot deployment. Reasoning about the choice of gait is another requirement for large-scale planning in the context of legged locomotion. For example, going through a building following a small-distance path might involve the use of slow walking gaits to climb stairs and navigate narrow corridors. In contrast, going around the same building on a longer-distance route could be a faster alternative using a speedy trotting gait.

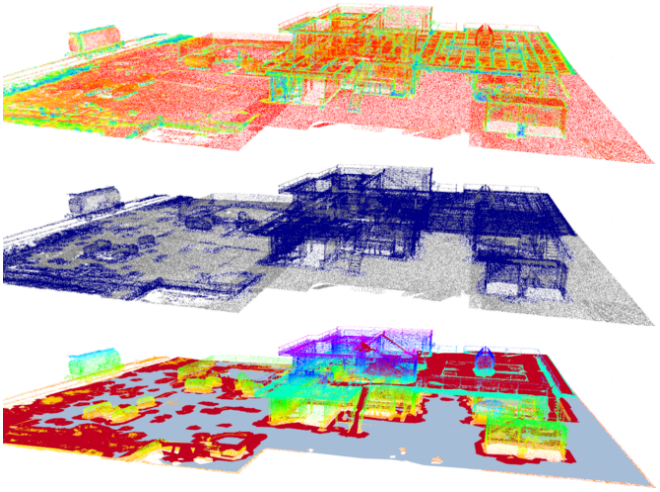

Fig. 1: Top to bottom: point cloud of an oil rig with color-coded curvature, gait-controller choice (gray for trotting and dark blue for walking), navigation mesh (light blue for trotting and red for walking).

In this paper we present methods for both building and using gait-aware navigation meshes for legged locomotion at such scales. Our contributions are:

- An automated method to generate navigation meshes of large-scale environments for legged robots with multiple gait-controllers.
- A complete system to navigate large-scale sites while reasoning (and switching between) controllers, using NavMeshes together with local planners.
- An evaluation of the advantages of navigation meshes for long-range planning.
- A detailed comparison between traditional robotics methods such as PRMs and RRTs, and NavMeshes, demonstrating how NavMeshes overall outperform these methods, especially when planning over multiple floors.

We evaluate our system both in simulation experiments and on multiple real-robot field trials at a realistic oil rig facility, used for personnel training exercises.

This work was supported by the UKRI/EPSRC ORCA Hub [EP/R026173/1], Robust Legged Locomotion [EP/S002383/1] and the EU H2020 Project THING. It was conducted as part of ANYmal Research, a community to advance legged robotics. The authors are with the Oxford Robotics Institute, University of Oxford, UK.

II. Related work

A large number of map representations have been used for robot locomotion planning. Probabilistic Roadmaps [6], occupancy grids [7], octree-encoded occupancy grids [8] and heightmaps [9], are a few of the popular map representations for this purpose. These are also popular for legged robot locomotion planning. For example, [10] uses search on occupancy grids and [11], [1] on heightmaps.

While these representations are suited to short-term planning, they do not scale well to large (e.g. multi-floor, city-wide) environments. For this reason, in long-term robot deployments to large office environments researchers have used manually designed topological graphs [12], [13]. Related map representations for long-term robot deployments include architectural floor plans [14], hand-drawn floor plans [15], sketches [16] and 3D vector maps [13]. Also for long-term planning, Probabilistic Roadmaps (PRMs) [6] have been used for legged robot locomotion planning [17] and in combination with Reinforcement Learning methods for long-term mobile robot navigation [18], [19]. Outside of robotics, [20] uses a database of CAD models for each floor and building of a university for navigation planning. While this representation allows for considerably larger-scale planning than the previous, it comes at the cost of intense manual labor in terms of CAD drawing, labeling of stairs and elevators, computing relative distances between floors and buildings, etc.

Our work targets long-term and large-scale robot deployments, for quadruped robots in particular. We specifically focus on building a system for fast path-finding in large-scale, potentially multi-level environments such as industrial oil rigs and other hard-to-access sites [1]. Except for PRMs, the map representations we have mentioned are either single-floor or manually designed—which could lead to imprecise maps or large manual work requirements. To account for multi-level large-scale facilities while avoiding manual labeling we use point cloud acquisition and automatic navigation-mesh construction from point clouds. The navigation mesh [2], [21], [3], [4] is a map representation that is popular in computer games due to its scalability and fast computation time for path-finding. Computation time is critical in game AI since often paths have to be found for hundreds or thousands of agents at fast rates [22]. The Recast toolkit for navigation meshes [21], for example, is used in several commercial games [23] and in the Unity engine [5]. Navigation meshes represent the world as a set of polygons and a graph representing traversability between them. Path-finding consists in a search over this compact graph. A comparison of navigation mesh methods is made in [4].

Due to the requirements of computer games, navigation meshes also usually encode per-polygon labels for the possible modes of locomotion and the costs of different areas of a map (e.g. an agent could swim over a river or travel to the closest bridge). This functionality is also close to the reality of legged robot locomotion—which is characterized by a large set of possible gaits [24] and controllers specialized for different kinds of terrain [1]. In this paper we use navigation meshes with controller-choice annotations to make long-term plans aware of the real cost of traversing different regions, and hence obtain paths of low global cost. We automatically compute these controller annotations by local 3D point cloud features similar to the heightmap-based work in [25].

III. Definitions

1) Gait controller: In this paper a "gait controller" is any method that controls the full-body motion of a robot to achieve a desired velocity, or goal position, of a robot's base. We consider a setting where multiple controllers are available for a given robot, specialized to different kinds of terrain. We represent this set of controllers by $\mathcal{M} = m_1, ..., m_M$, where M is the number of available controllers.

2) Walkable environment: We use the concept of a "walkable environment" as defined in the navigation-mesh literature [4]. A "walkable environment" W is a set of triangles that are traversable by an agent, i.e. robot. For the purpose of this paper, each triangle t is a tuple of three points and a label, $t = (p_1, p_2, p_3, m)$, where $p_1, p_2, p_3 \in \mathbb{R}^3$ and $m \in \mathcal{M}$. The label identifies the preferred choice of controller to be used when the robot's projected COM lies within the triangle. These triangles represent the surfaces traversable by the agent (i.e. robot). For typical legged robots this will consist of triangles on the floor, stairs and other traversable surfaces. Surfaces that are more inclined than what is possible by the robot's capabilities will not be part of the walkable environment.

3) Multi-layered environment: Intuitively, a "multi-layered environment" is an environment with walkable areas at multiple heights, e.g. a multi-floor building. More formally, a walkable environment W is multi-layered when the projection of its triangles to a horizontal plane leads to intersections [4].

4) Navigation mesh: A navigation mesh is a tuple $N = (W, G)$: it consists of a walkable environment W and an undirected graph G representing the possibility to navigate between adjacent triangles in W.

IV. Method

The overview of our proposed system is shown in Fig. 2. It uses high-level planning based on NavMeshes together with local-map planning to navigate large environments, while allowing for online adaptation using virtual obstacles. We will go through the process of building NavMeshes, and the content of each of the blocks in the following sections.

A. Building large controller-annotated environment meshes

Before we can build a gait-aware NavMesh we require a triangular mesh where each triangle is annotated with a gait controller choice. We will now explain the process, summarized in Fig. 3, to obtain such an annotated mesh. The method uses either a CAD model of the environment or a point-cloud acquired on-site. We will now describe the point-cloud-based method.

1) Point cloud acquisition: We obtain point clouds of whole multi-layered facilities by successive laser-scanning and registration using a portable device. In principle any mapping device can be used for this step, including the on-board robot SLAM system, as long as the cloud captures the walkable environment, i.e. floor, stairs surfaces, etc.

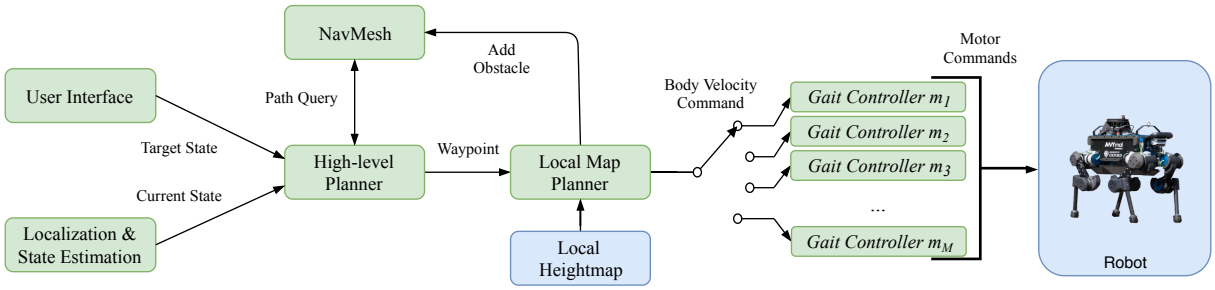

Fig. 2: System overview.

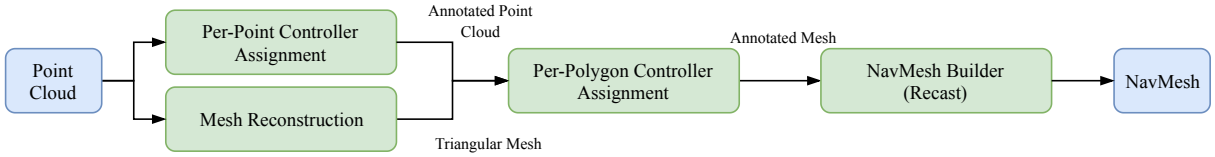

Fig. 3: Our procedure for generating gait-controller-aware navigation meshes from large point clouds.

2) Per-point controller assignment: We start by computing normals and curvature for all points in the point cloud. After, in a similar way to [25], for each point we compute the local maximum of curvature $c_{\max}$ in a spherical neighborhood around the point. We use the radius of the smallest sphere which encloses the robot. While we found that curvature information is sufficient to choose between our robot platform's controllers, other features such as roughness, slope or height-differences could be used [25] as appropriate for the robotic platform and controller choices available.

In our case we assign a trotting-gait controller, specialized for flat terrain, to all points where $c_{\max}$ is below a threshold, and otherwise assign a walking-gait controller which uses vision for foot placement. The output of this step is therefore a point-cloud annotated with a choice of controller (i.e. a unique number identifying a controller).

3) Mesh reconstruction: We reconstruct a 3D triangular mesh from the original point cloud using the Ball-Pivoting algorithm [26]. While other methods are openly available [27] we found that Ball-Pivoting produced good results for large-scale environments. We recommend the interested reader to see [28] for a comparison of this and alternative open-source mesh reconstruction methods for robotics applications.

4) Per-polygon controller assignment: The final step in the procedure is to assign a controller choice to each *triangle* of the reconstructed mesh. To achieve this we project the point-cloud controller-annotations back to mesh triangles. For each triangle t in the mesh we obtain all the points in the annotated point cloud P that fall within a distance $d_{\max}$ of t. We then run a voting procedure: we count the number of points which support each of the controllers and assign the maximally-voted controller to triangle t.

When a CAD model of the environment is available, we uniformly sample the CAD model to obtain a dense point cloud. We then run the same cloud-annotation and cloud-to-mesh annotation projection procedure, described above, to obtain a controller-annotated mesh.

B. Building multi-controller navigation meshes

We use the Recast toolkit [21] to generate navigation meshes from the annotated mesh. Recast starts from a triangular mesh as input, and produces a navigation mesh using the following procedure:

1) Voxelize the polygons. This generates a multiple-height heightfield from the triangular mesh. It consists of multiple spans for each cell on the horizontal plane, representing the occupied regions running vertically on that cell.
2) Annotate walkable space from solid voxels. This filters out voxels where a cylindrical approximation of the robot cannot stand based on surface inclination and vertical clearance. The heightfield representation obtained by the previous step is particularly suited for this stage since identifying voxels on floor surfaces and with enough ceiling-clearance is fast.
3) Compute a distance field in voxel space. This is the distance of each walkable voxel to the closest non-walkable voxel on the heightfield, or to a voxel annotated with a different controller-choice.
4) Run a "watershed partitioning" algorithm on the distance field. This basically achieves segmentation of the the walkable voxels in order to obtain simple polygons that do not overlap, and are connected in saddle points of the distance field (usually corresponding to doors, passages and other meaningful locations). This is inspired by [29]. We encourage the interested reader to refer to that publication for a detailed and intuitive explanation of the method.
5) Compute polygons representing the contours of the segmented voxel regions. This is done by tracing the contours of each region and approximating it with polygons through the Ramer-Douglas-Peucker algorithm [30].
6) Triangulate the polygons and build triangle connectivity. This is done to build the connectivity graph G used for

planning in the navigation mesh.

For a more in-depth overview of the design choices and implementation, we refer the reader to [2].

When applied to large-scale point-cloud-estimated meshes, the previous procedure generates a considerable number of unconnected (inaccessible) walkable areas—such as the top of a lamp post, the top of ceiling rails, or areas where point cloud noise is high (e.g. far away from the laser scans). This usually makes visualization of the navigation meshes more difficult and needlessly increases the size of the navigation mesh [3]. To avoid this we extend the previous procedure with an additional post-processing step: we remove all NavMesh triangles to which there exists no path to a reference point that is known to be accessible by the robot (e.g. point in a large room, or the "base station" of the robot).

The output of the whole process is a clean NavMesh N that represents the walkable environment, and is annotated with a controller choice for each triangle. Each controller choice can be given a different weight, which scales the distance costs of the respective triangles in the graph representation of the navigation mesh.

This NavMesh can be used for fast path-finding by:

1) projecting start-goal points to the mesh,
2) running A* search on the space of NavMesh triangles, between the start and goal triangles,
3) computing the shortest line-path that connects the sequence of triangles found.

An efficient implementation of this procedure is already provided with the Recast toolkit. We implemented a ROS wrapper for both the path-finding and mesh-construction functions of Recast, as well as the additional post-processing routine described previously. As described next, this ROS wrapper is used by a high-level planner to obtain intermediate waypoints for local navigation.

We provide both the ROS wrapper for Recast and the pipeline for generating gait-controller annotations open-source in `https://github.com/ori-drs/gaitmesh` [1].

C. High-level planner: long-term multi-controller locomotion with navigation meshes

The high-level planner takes as input a goal position within the map and guides the robot to it. At 1Hz, it queries the NavMesh to obtain the global path to the goal, and sends an intermediate goal to local planner. We compute the intermediate goal by finding the point along the NavMesh's path that is either 1 meter way from the current state of the robot, or just before a sharp turn—whichever option is closest. In our experiments we defined 20 degrees as a sharp turn.

D. Local planner: short-term multi-controller locomotion

The local planner computes a trajectory to the intermediate goal given by the high-level planner, using a local heightmap obtained from the robot's onboard sensors. We use GridMap for heightmap computations [9]. We use sensor-based maps instead of the NavMesh at this stage in order to reflect the actual geometry of the environment—which might have changed since the NavMesh construction stage.

The local planner uses A* search to compute a path from the current state of the robot to the intermediate goal. The search is made in the space of 3D position and yaw rotation, thus also compensating for the cylindrical assumption of path-finding in the navigation mesh. We compute collision using a set of spheres that cover the body of the robot and computing its intersection with the local heightmap.

Furthermore, the planner constantly checks the map's controller annotations in the location over which the robot is currently walking to trigger the switch of gait controllers. For the experiments presented in this paper we used the NavMesh annotations to trigger controller switches. However, similar local features that are used for NavMesh annotation could potentially be computed on the local sensor-based maps to trigger the switches as well, as we have shown in [1].

Finally, the local planner can add obstacles to the NavMesh whenever it fails to find a path to the intermediate goal. This could happen either because the environment has changed (e.g. new furniture, closed doors), or because of errors in the NavMesh construction process (e.g. an obstacle was not properly seen by the scanning device). Such NavMesh-obstacles will immediately lead to changes in the global paths and thus in changes to the intermediate goals sent by the high-level planner to the local-planner.

We implement NavMesh-obstacles as cylinders which trigger the removal of the underlying triangles. We use cylinders of constant radius equal to the radius of the robot, although the interface is general and allows to use variable-radius cylinders in the future (e.g. estimated through perception).

V. Results

A. Building navigation meshes

We tested our NavMesh building process in two environments. Environment 1 is an oil rig at the Fire Service College (FSC) in Moreton-in-Marsh, UK, a facility to train personnel in various simulated environments. Aspects of the oil rig appear on Fig. 5 and 6. We used a portable laser-scanner to map the environment. This involved placing the device at multiple locations on a tripod to take successive 360 degree scans and register them to previous scans. We then post-processed clouds to remove people, sub-sampled the cloud, and ran the process in Fig. 3. The whole scanning-and-NavMesh-construction process took approximately 9 hours. The NavMesh generation step took 8.6 seconds.

The point cloud, curvature and annotated point clouds are shown in Fig. 1 together with the resulting navigation mesh. Trotting areas are in blue and vision-based planned-step static walking areas are in red. Floors are connected by stairs which appear connected in the navigation mesh. The number of polygons to represent the full environment is 6075. Most flat ground is annotated with trotting except around narrow passages such as doorways and staircases. The elevated floor of the facility is annotated with static walking because of its rough metal grid-like flooring.

[1]The repository will be made public upon acceptance.

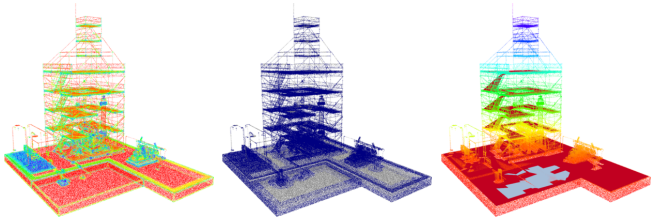

Fig. 4: Environment 2, the many-floored ARGOS Challenge facility model. Left to right, point cloud with color-coded curvature, controller choice, navigation mesh.

Environment 2 is a CAD model of a real oil & gas onshore site used for TOTAL's ARGOS Challenge. We followed the procedure for CAD models indicated in Section IV-A. NavMesh generation took 0.5 seconds. We show the clouds and NavMesh in Fig. 4. The multiple floors are connected by stairs which appear as long triangles in the navigation mesh. The number of polygons to represent the full environment is 447. Elevated floors are annotated with static-walking as the corridors are narrow.

B. Comparison to sampling-based planners

In this section we compare NavMeshes against popular sampling-based planning methods in robotics. In particular PRM* uses a similar approach to planning compared to NavMeshes—it encodes traversability between locations in the environment offline as a graph, which is then searched at query time and refined to find an optimal path. While NavMeshes rely on deterministically building a geometric representation of the environment (set of triangles), PRMs build only a graph connecting specific random locations. For this experiment, all methods minimized only distance of the path—thus ignoring gait-controller annotations.

We ran the comparison on Environment 1. To execute PRM* we used the OMPL library implementation [31] on the point cloud of the environment. We defined state validity as the intersection of a cylinder with any points on the cloud. We used a cylinder of the same dimensions as that used for the NavMesh. For a state to be feasible it also needs to intersect points in a volume below the cylinder (i.e. to make contact with the ground). States are sampled by randomly picking points from the point cloud and placing the robot at a height within a random interval. For planning trajectories in this roadmap we let PRM* refine the path. We obtained results for building times of 10, 20 and 360s, and for planning times of 0.1s and 1.0s. We also compare NavMeshes to the use of RRTs: where at planning time an RRT is run from scratch for a given time budget.

Table I shows the pre-computation time, planning time, and path length obtained with NavMeshes, PRMs and RRTs on 40 planning problems. Each planning problem is a randomly sampled start and goal state, and the average path lengths are shown. All methods were given the same planning problems. Since randomly-generated start and goal states can be on different-floors, we specifically generate 20 same-floor problems and 20 different-floor problems. The table shows that NavMeshes obtain 100% success rate at 1000x faster computation times than the best-performing sampling-based method. The best competitor was PRM* when it was left to build the roadmap for 360s (compared to 8.6s building time for NavMeshes) and at a computation time of 1 second, compared to 1ms for NavMeshes. Path lengths were around 30% higher in PRMs, and only when the roadmap is built for long enough time (360s). Of the single query methods RRTConnect did best but only solved 5 out 20 problems.

Importantly, the success rate of sampling-based planners considerably drops when the environment becomes more challenging and involves narrow passages, such as staircases to access different floors). In this setting, only PRM* manages to find paths, but again at lower speeds and at a 50% success rates compared to 100% for NavMeshes.

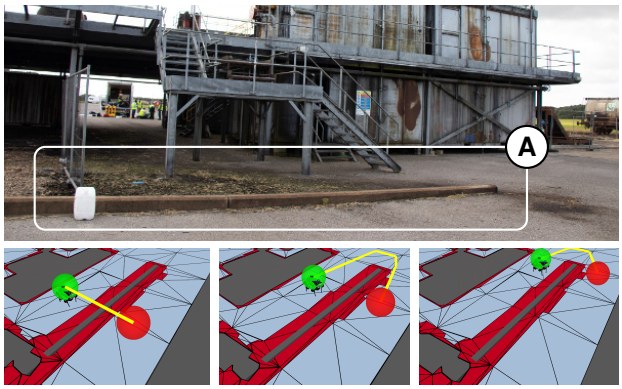

Fig. 5: The influence of controller-reasoning in the NavMesh. The robot either climbs over or trots around a barrier (A) depending on the distance to its end. The top image shows the site upon which the simulation was made.

C. Qualitative inspection of multi-controller paths

Next, we qualitatively show the behavior of the NavMesh-based planner when gait-controller annotations are considered. We have two controllers available: 1) flat-ground trotting and 2) vision-based planned-step static walking. Since the trotting controller moves at approximately 8 times faster speeds than the static walking controller, we used a cost multiplier of 8 for walking-annotated regions. This means that the cost of 1 meter on a trotting-annotated region is 1, but on a walking-annotated region it is 8.

Fig. 5 shows paths obtained around a 20cm-high barrier on the ground. It shows how the planner produces paths that go around the barrier when the robot is sufficiently close to its end, but over the barrier when far away. Fig. 6 shows a longer-range example, where the planner prefers to go around a building for a longer distance instead of going inside and outside the building in a shorter-distance straight path. Going through the building involves slow walking over the stairs to go in and out, when compared to the time required to go around the structure on a flat area while trotting. In this particular case the predicted travel time would be 3.7 higher going through the building instead of around it, despite the shorter distance.

			Same-floor problems		Different-floor problems	
Planner	Pre-computation time (s)	Planning time (s)	Success rate	Path length $\frac{l_{\text{Planner}}}{l_{\text{Recast}}}$	Success rate	Path length $\frac{l_{\text{Planner}}}{l_{\text{Recast}}}$
NavMesh	8.6	0.001	20 / 20	1.00 ± 0.00	20 / 20	1.00 ± 0.00
PRM*	10.0	0.100	00 / 20	-	00 / 20	-
PRM*	10.0	1.000	00 / 20	-	00 / 20	-
PRM*	20.0	0.100	01 / 20	1.03 ± 0.00	00 / 20	-
PRM*	20.0	1.000	01 / 20	1.03 ± 0.00	00 / 20	-
PRM*	360.0	0.100	09 / 20	1.34 ± 0.09	09 / 20	1.22 ± 0.17
PRM*	360.0	1.000	10 / 20	1.33 ± 0.09	09 / 20	1.22 ± 0.17
RRT	0.0	0.100	00 / 20	-	00 / 20	-
RRT	0.0	1.000	03 / 20	1.34 ± 0.11	00 / 20	-
RRTConnect	0.0	0.100	00 / 20	-	00 / 20	-
RRTConnect	0.0	1.000	05 / 20	1.98 ± 0.70	00 / 20	-
RRT*	0.0	0.100	00 / 20	-	00 / 20	-
RRT*	0.0	1.000	02 / 20	1.11 ± 0.11	00 / 20	-

TABLE I: Benchmark of success rates and path lengths in NavMeshes vs PRMs and RRTs.

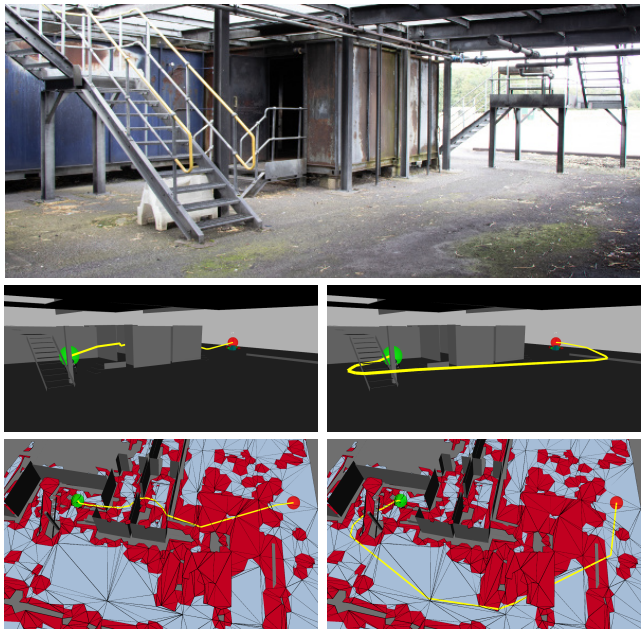

Fig. 6: The advantage of controller-reasoning in the NavMesh. Left: a least-distance path goes through buildings which require stair-climbing and few trotting. Right: the controller-aware path is lengthier but 3.7 times faster since it goes around the buildings on long trotting periods.

D. Full execution in simulation

In the next experiment we tested the execution of the full system of Fig. 2 which includes high-level planning, local-map planning and controller switching. We conducted the experiment in the Gazebo simulator in ROS. The robot started in the middle of the (approximate simulation model of the) FSC oil rig facility and we gave the high-level planner a goal inside one of the buildings. In addition, we added to the environment an object that was not present in the navigation mesh. Fig. 7 shows the robot trotting outside the building and adding virtual obstacles to the navigation mesh, indicated as red cylinders, as it fails to locally plan to follow the global path. The robot then navigates around the new object and successfully climbs the stairs to access the inside of the building. Finally it walks to reach the desired goal. A video of the experiment is included in the attachment.

E. Long-range executions on the real robot

We conducted a set of field trials at the FSC oil rig facility (Fig. 1). In the first experiment the robot was placed on flat ground and given a goal around the facility, in a location that requires going inside and then navigating a set of metal containers connected through narrow passages. A zoom-up of the location is shown in Fig. 8 together with the trajectory executed by the robot, as given by the output of the localization block. The figure shows the robot continuously trotting to the entrance, walking over the steps, navigating the containers, and then exiting the structure through another set of steps on a distant side of the facility. The total traveled distance was 29 meters. Path-planning within the NavMesh took approximately 1ms consistently throughout the execution.

For our second experiment the robot was placed outside on flat ground and given a goal straight ahead on the other side of the facility, after the 20cm barrier previously described in Sec. V-C. We set a goal close to the end of the barrier on purpose, so that a plan is produced that walks around the barrier instead of the shorter-distance option of climbing over—as in the experiment of Sec. V-C. Fig. 9 shows the path the robot takes, straight on flat ground, around the pillars and staircases and then around the barrier to reach the goal. The total traveled distance was 33 meters. Path-planning within the NavMesh took less than 1ms throughout.

Finally, we placed the robot in front of the same barrier but further away from its end, as in Fig. 5. The robot was given a goal straight ahead. Fig. 10 shows the robot trotting up to the barrier, walking over it using the vision-based walking controller and then trotting to the goal. The total traveled distance was 6 meters.

VI. Conclusion and Discussion

We proposed the use of navigation meshes as a high-level planning tool for long-range legged locomotion. We proposed a way to automatically annotate and build these structures in a way that is relevant to the multi-controller nature of legged robots. We integrated NavMeshes with high- and low-level planners that deal with long- and short-term reasoning, as

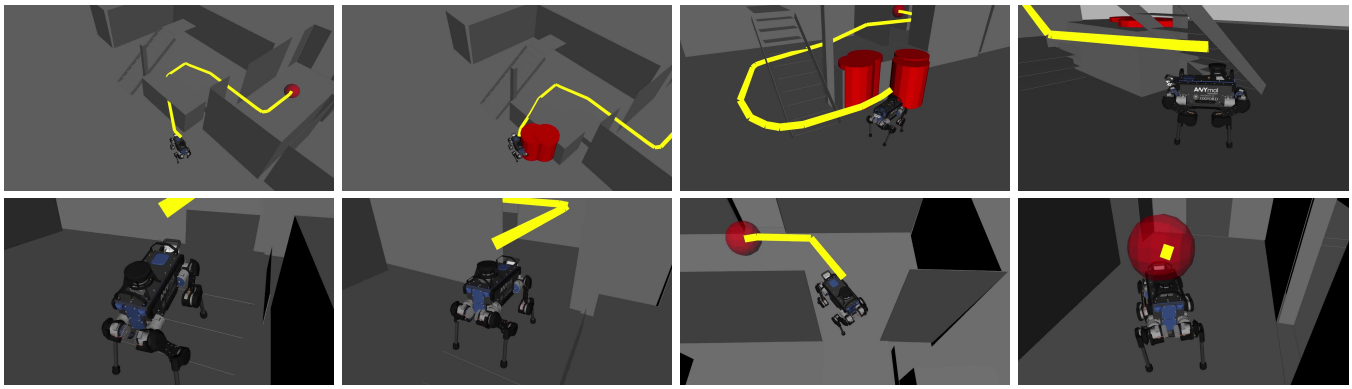

Fig. 7: Full execution in a simulated version of the FSC oil rig environment. Yellow lines indicate the shortest-path to the goal as computed in the NavMesh. The large box that the first paths go through was not modeled in the original NavMesh. Red cylinders indicate virtual obstacles added to the NavMesh on-line. The red sphere indicates the goal.

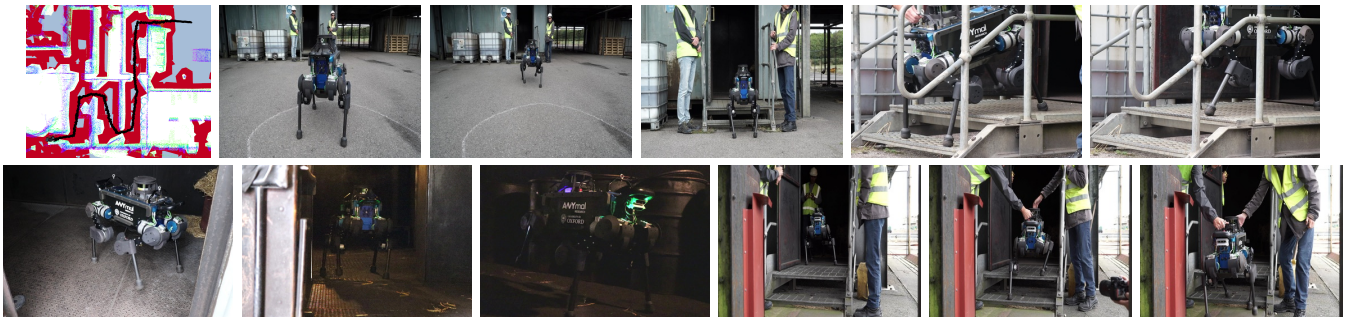

Fig. 8: Experiment 1 at the FSC oil rig site. The first image shows the original point cloud and navigation mesh (cropped for visibility) overlapped with the executed path given by localization.

Fig. 9: Experiment 2 at the FSC oil rig site. The first image shows the original point cloud and navigation mesh (cropped for visibility) overlapped with the executed path given by localization. The robot trots through the facility and around a 20cm barrier to reach a goal close its end.

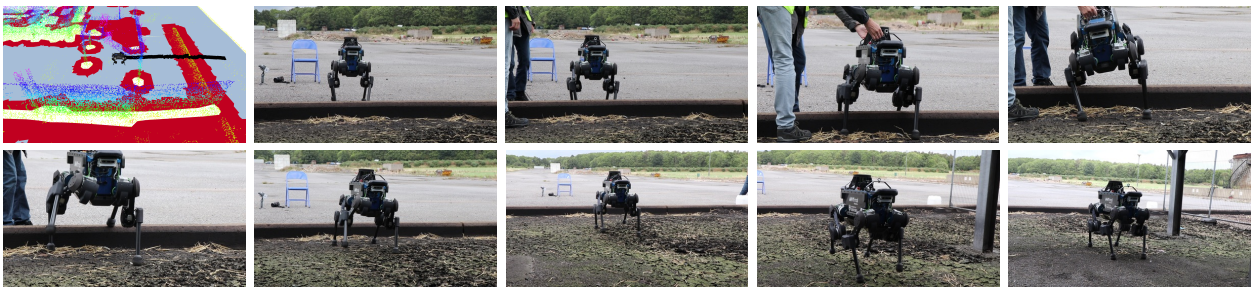

Fig. 10: Experiment 3 at the FSC oil rig site. The first image shows the original point cloud and navigation mesh (cropped for visibility) overlapped with the executed path given by localization. The robot climbs over a 20cm barrier when far away from its end.

well as a way to switch between controllers and deal with unmodeled or new obstacles. We compared the performance of path-finding in NavMeshes against traditional sampling-based planners and quantitatively showed the superiority of NavMeshes—they are both faster to build and query than PRMs, as well as finding considerably more paths and of 30% shorter lengths. We ended by demonstrating the usefulness of such a system in real-world large-scale locomotion examples in an industrial facility.

In future work, we aim to tackle two of the limitations of the current system. One is the mesh reconstruction step, which because of point cloud noise can lead to narrow passages becoming even narrower on the reconstructed mesh and NavMesh. Currently this means that either robot-cylinder radii have to be made smaller than the actual robot for planning to be possible, or post-processing of the point cloud must be done to clean narrow passages before mesh reconstruction (we used the former in this paper). In the future we will investigate better mesh reconstruction methods for this purpose. Another limitation of our approach is the cylindrical approximation of NavMeshes itself—which implies that NavMesh paths are not guaranteed to be executable in general but especially for long robots such as quadrupeds on narrow turns. To alleviate this issue we are considering a path-verification step of path planning or NavMesh-construction to identify such cases and obtain alternative paths. Other interesting research directions include online NavMesh building, and the use of NavMeshes for sampling-bias or cost-heuristics in sampling and search-based multi-gait planning methods such as [1].

REFERENCES

[1] M. Brandao, M. Fallon, and I. Havoutis, "Multi-controller multi-objective locomotion planning for legged robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2019.

[2] M. Mononen, "Navigation mesh generation via voxelization and watershed partitioning," *AiGameDev.com*, 2009.

[3] R. Oliva and N. Pelechano, "Neogen: Near optimal generator of navigation meshes for 3d multi-layered environments," *Computers & Graphics*, vol. 37, no. 5, pp. 403–412, 2013.

[4] W. Van Toll, R. Triesscheijn, M. Kallmann, R. Oliva, N. Pelechano, J. Pettré, and R. Geraerts, "A comparative study of navigation meshes," in *9th International Conference on Motion in Games*. ACM, 2016, pp. 91–100.

[5] Unity. (2019) Unity user manual - navigation and pathfinding. [Online]. Available: https://docs.unity3d.com/Manual/Navigation.html

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

[7] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[8] K. M. Wurm *et al.*, "Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems," in *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2, 2010.

[9] P. Fankhauser and M. Hutter, "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation," in *Robot Operating System (ROS) The Complete Reference (Volume 1)*, A. Koubaa, Ed. Springer, 2016, ch. 5.

[10] C. Mastalli, I. Havoutis, A. W. Winkler, D. G. Caldwell, and C. Semini, "On-line and on-board planning and perception for quadrupedal locomotion," in *2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, May 2015, pp. 1–7.

[11] D. Maier, C. Lutz, and M. Bennewitz, "Integrated perception, mapping, and footstep planning for humanoid navigation among 3d obstacles," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2658–2664.

[12] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova, J. Young, J. Wyatt, D. Hebesberger, T. Kortner, *et al.*, "The strands project: Long-term autonomy in everyday environments," *IEEE Robotics & Automation Magazine*, vol. 24, no. 3, pp. 146–156, 2017.

[13] J. Biswas and M. M. Veloso, "Localization and navigation of the cobots over long-term deployments," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1679–1694, 2013.

[14] F. Boniardi, T. Caselitz, R. Kümmerle, and W. Burgard, "Robust lidar-based localization in architectural floor plans," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3318–3324.

[15] F. Boniardi, A. Valada, W. Burgard, and G. D. Tipaldi, "Autonomous indoor robot navigation using a sketch interface for drawing maps and routes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2896–2901.

[16] V. Setalaphruk, A. Ueno, I. Kume, Y. Kono, and M. Kidode, "Robot navigation in corridor environments using a sketch floor map," in *2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 2. IEEE, 2003, pp. 552–557.

[17] K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1325–1349, 2008.

[18] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5113–5120.

[19] J.-J. Park, J.-H. Kim, and J.-B. Song, "Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning," *International Journal of Control, Automation, and Systems*, vol. 5, no. 6, pp. 674–680, 2007.

[20] E. J. Whiting, "Geometric, topological & semantic analysis of multi-building floor plan data," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[21] M. Mononen, "Recast navigation," *https://github.com/recastnavigation/recastnavigation*, 2014.

[22] I. Millington and J. Funge, *Artificial intelligence for games*. CRC Press, 2016.

[23] MobyGames. (2019) Games using recast. [Online]. Available: https://www.mobygames.com/game-group/middleware-recast

[24] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2261–2268, 2018.

[25] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter, "Navigation planning for legged robots in challenging terrain," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 1184–1189.

[26] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE transactions on visualization and computer graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[27] T. Wiemann, A. Nüchter, and J. Hertzberg, "A toolkit for automatic generation of polygonal maps-las vegas reconstruction," in *ROBOTIK 2012; 7th German Conference on Robotics*. VDE, 2012, pp. 1–6.

[28] T. Wiemann, H. Annuth, K. Lingemann, and J. Hertzberg, "An evaluation of open source surface reconstruction software for robotic applications," in *2013 16th International Conference on Advanced Robotics (ICAR)*. IEEE, 2013, pp. 1–7.

[29] D. Haumont, O. Debeir, and F. Sillion, "Volumetric cell-and-portal generation," in *Computer Graphics Forum*, vol. 22, no. 3. Wiley Online Library, 2003, pp. 303–312.

[30] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[31] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.