Autonomous Navigation with Online Replanning and Recovery Behaviors for Wheeled-Legged Robots using Behavior Trees

Alessio De Luca[1,2], Luca Muratore[1] and Nikos G. Tsagarakis[1]

***Abstract*— Performing autonomous navigation in cluttered and unstructured terrains still remains a challenging task for legged and wheeled mobile robots. To accomplish such a task, online planners shall incorporate new terrain information perceived while the robot is moving within its environment. While hybrid mobility robots offer high flexibility in traversing challenging terrains by leveraging the advantages of both wheeled and legged locomotion, the effective hybrid planning of the mobility actions that transparently combine both modes of locomotion has not been extensively explored.**

In this work, we present a hierarchical online hybrid primitive-based planner for autonomous navigation with wheeled-legged robots. The framework is handled by a Behavior Tree (BT) and it takes into account recovery methods to deal with possible failures during the execution of the navigation/mobility plan. The framework was evaluated in multiple irregular and heavily cluttered simulated environments randomly generated and in real-world trials, using the CENTAURO robot platform. With these experiments, we demonstrated autonomous capabilities without any human intervention, even in case of collision or planner failures.

***Index Terms*: Motion and Path Planning, Reactive and Sensor-Based Planning, Field Robots**

I. Introduction

Until today, legged and mobile robots, in general, are not yet extensively explored in real-world unstructured environments, which impose highly cluttered terrain challenges on them, e.g., in the field of inspection, maintenance, and search and rescue operations. An essential capability that is necessary for these robots to deal autonomously with complex terrain challenges and complete their tasks is the ability to safely navigate in cluttered/unstructured grounds, negotiating different obstacles of various shapes and dimensions. In addition, to deal with perception errors, dynamic obstacles, partially perceived entities, or other uncertainties, a robot needs an online planner that can adapt on the fly the navigation and mobility plan when these unexpected conditions occur, comprising the original plan. Online capabilities allow the continuous elaboration of the plans, taking into account the online perceived information about the terrain and environment in general during the execution of its navigation plan. This is because, with offline methods, the plan is evaluated only at the beginning of the task without considering new information coming from the sensors. For this reason, it can be reliable only in a limited area surrounding the robot while it may not be adequate and inaccurate after a few meters of navigation. This is valid, especially in irregular environments where there are obstacles that hide portions of the terrain, preventing it from being seen from the starting configuration.

To deal with online planning updates, different methods have been proposed. A possible way is to take into account the forces and contacts between the robot and the environment, like in [1]-[2]. Here, the planner has the objective of selecting the best contact points to obtain a sequence of feasible robot configurations, preventing the robot from falling or slipping while navigating towards the goal location. However, this approach may require a long computational time and there is a need for accurate contact and force estimation. Online re-planning can also be done using sampling-based methods as in [3]-[4] where whole-body motions are connected to build a tree by using Center of Mass (CoM) primitives or a more general robot state as in [5], but, with complex robots, this can be slow and memory inefficient. Another solution is to employ Model Predictive Control (MPC) like [6]-[7], continuously obtaining plans that are valid for a small horizon, requiring a fine-tuning of the model and parameters. Other methods were proposed to perform online planning, like the work presented in [8], which considers a fusion of sampling-based techniques and model-based optimization validated on a quadcopter in different scenarios. A different strategy, introduced in [9], involves the possibility of merging offline path planning and online foothold planning. The work was validated on the quadruped LittleDog in different small scenarios. Considering similar approaches, the work in [10] presented a comparison among search-based planners for legged robots, showing the advantages of ARA* compared to A* and Dijkstra. Later, in [11], the same team proposed a motion planning framework for quadrupedal locomotion embedding robot attitude planner and a terrain model to describe feasible footholds, having as a drawback a high computation time, requiring more than 10 minutes to plan a solution.

However, the majority of the works presented are carried out on quadrupeds and humanoids that have reduced kinematic complexity compared to the CENTAURO robot [12], which has four articulated 5DoF legs, ending in 360° steerable wheels, making it a hybrid mobility platform. A more similar robotic platform is the Momaro robot [13], where the team developed a framework to deal with semi-autonomous loco-manipulation tasks in [14]. Although, no online updates were demonstrated and the validation was done in simple environments only. Autonomous navigation with the CENTAURO robot was demonstrated in [15], but only driving capability was considered to reshape the support polygon based on the environment, reducing the complexity

[1]Humanoids and Human-Centered Mechatronics Research Line, Istituto Italiano di Tecnologia (IIT), Via Morego 30, Genova 16163, Italy.E-mail:(alessio.deluca@iit.it; luca.muratore@iit.it; nikos.tsagarakis@iit.it)
[2]DIBRIS, Università di Genova, Italy, 16145.

Received July 15, 2023 07:52:57 Pacific Time

and capabilities of the platform, not being able to traverse more cluttered scenarios.

In addition to the planning side, one important aspect to enhance autonomy is the possibility of reacting to failures in the execution. A potential approach to implement this is with the use of Behavior Trees (BTs) [16], which proved to be more reactive, modular, and easier to extend compared to Finite State Machines as stated in [17] and [18]. Nevertheless, to the author's knowledge, the use of behavior trees in legged and especially hybrid robot navigation is limited. The two relevant works in [19] and [20], mainly focused, respectively, on multi-robot task reallocation in case of failures and reactive push recovery, without directly addressing the challenge of autonomous navigation that also incorporates recovery methods in case of failures during the execution in cluttered environments.

In our previous work [21] we proposed a framework to perform autonomous navigation in an offline manner with the use of a set of motion primitives, employing a search-based planner. In this work, we extend the framework by proposing a hierarchical planning architecture composed of search-based global and local planners. The framework is embedded with online and reactive functionalities, being able to update the locomotion plan while the robot is moving, and autonomously respond to possible failures during the execution. The whole framework is managed by a BT demonstrating successful results in both simulation and real-world environments. This results in a much more reliable and robust execution in cluttered and unknown terrains, increasing the autonomy of the robot in dealing with such challenges while executing navigation tasks. The main contribution of the proposed approach is, therefore the following:

- Exploitation of a two-level online framework for autonomous navigation;
- Autonomous recovery behaviors in case of failures;
- A Behavior Tree implementation for managing the online navigation framework in a reactive way;

The proposed framework and the above contributions were extensively validated in extensive simulation trials considering varying complexity cluttered environments and eventually experimentally under different terrain/obstacle arrangements using the CENTAURO hybrid mobility robotic platform. The rest of the paper is organized as follows: in Section II we describe the proposed framework explaining its different components. In Section III, we present the experiments carried out and the results obtained, and finally in Section IV we draw the conclusions.

II. Proposed Framework

In our previous work [21], we designed and validated an offline method for hybrid navigation with wheeled-legged robots. The method was based on the use of parametrized motion primitives to adapt to different robots. Here, we extend our framework by combining a two-level hybrid planner together with a Plan Evaluator, recovery methods, and a two-level mapping, as can be seen in Fig. 1. In the following sections, we are going to illustrate in more detail the overall architecture of the proposed framework and the functionality and implementation of the involved components.

A. Traversability Extractor

This component is responsible for the extraction and the definition of the elevation and validity map used by the two-level hybrid planner. In particular, we employed the software of [22]-[23] to obtain a representation of the environment as elevation maps, starting from point clouds that are filtered to obtain a foothold validity map. The filtering process is carried out taking into account a safety threshold used to inflate the edges of the obstacles perceived based on the size of the end-effector (in our case, the wheel radius of the CENTAURO robot), marking those areas as non-valid. In this work, we consider two different mapping processes, the first uses the information coming from the 3D LiDAR, a Velodyne Puck 16 [1]. The result of this mapping part is called *global map* and it has a resolution of $0.04\,\mathrm{m}$ and a dimension of $12\,\mathrm{m}$ x $8\,\mathrm{m}$. In addition, we explore an Intel Realsense D435i [2] sensor, which is an RGB-D camera, to obtain a second map with a resolution of $0.02\,\mathrm{m}$ and dimension $3.5\,\mathrm{m}$ x $3.0\,\mathrm{m}$, called *d435i map*. At this point, we build the *local map* by merging the information obtained from the two sensors. In particular, the local map is the d435i map in which the unknown elements are filled up with the information obtained by the Velodyne. This allows us to obtain a map with a better resolution with respect to the global map and to deal with the smaller Field of View of the RGB-D camera compared to the Velodyne. In fact, with this approach, we always use the data coming from the D435i, which is more dense and accurate up to the limits of the local map, and we integrate them with the information coming from the LiDAR being able to extend the knowledge of the only d435i map.

Both maps are centered in the base link position of the robot and move together with it, incorporating the updates based on the new information perceived. The global and local maps are used respectively by the global and local planners. In addition, the local map is also used by the Plan Evaluator described in the next subsections.

Fig. 2 presents the global and the local maps of an environment, marking non-valid areas in red.

B. Global Planner

The global planner has the objective of guiding the movements of the robot inside the map with its resulting plan without directly commanding the execution. For the global planner, we used the same implementation we proposed in our previous work [21], based on the ARA* [24]. In more detail, the inputs of the planner are the global map obtained using only the Velodyne Puck 16 Lidar sensor, the initial robot pose, the target position and orientation we want the robot to reach, and the primitives available (whole-robot driving, stepping, and single-wheel driving) together with their parameters, based on the platform considered. This planner

[1] https://velodynelidar.com/products/puck/
[2] https://www.intelrealsense.com/depth-camera-d435i/

Received July 15, 2023 07:52:57 Pacific Time

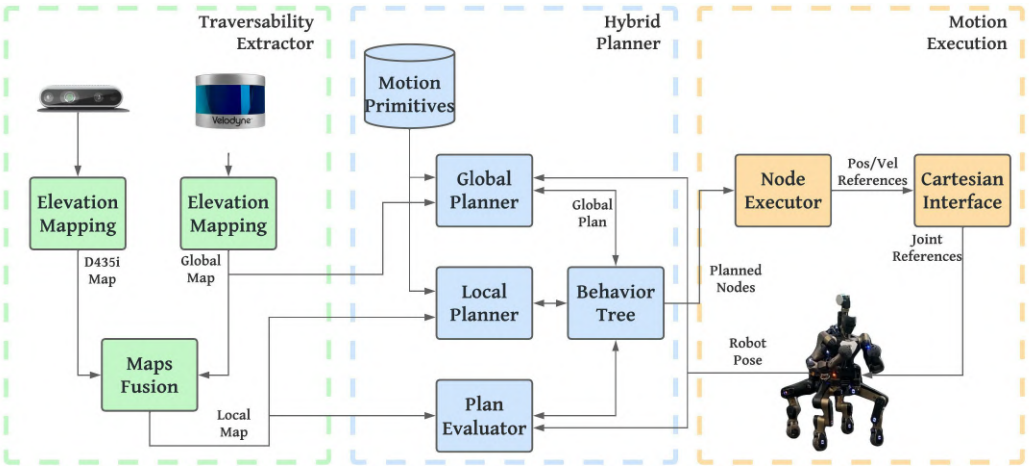

Fig. 1: Overview of the framework provided. In green, we have the perception component, in blue the two-level online planning component, and in orange the motion execution which sends the controls to the robot.

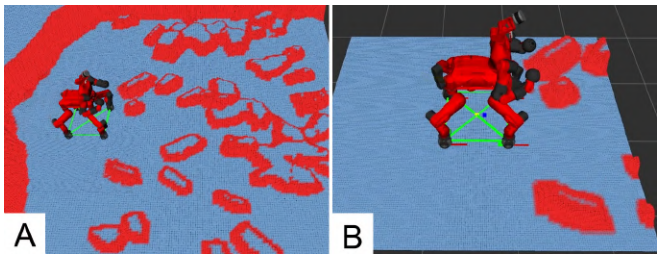

Fig. 2: (A) Global map of the environment considered. (B) Local map of the surrounding of the robot. The maps are built using different safety thresholds and resolutions. Red is used to identify non-valid areas.

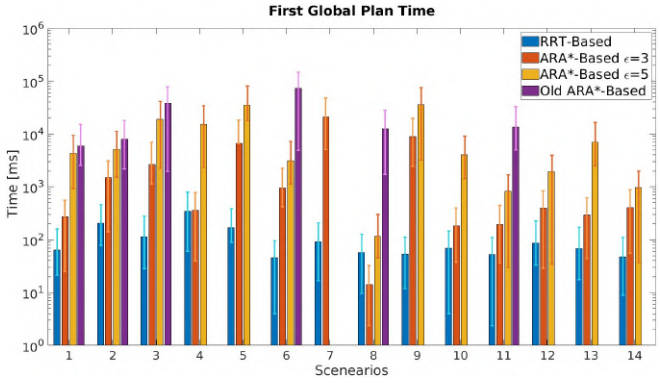

Fig. 3: Time comparison for the first solution of the global planner using RRT-based and different versions of the search-based planners with 14 randomly generated scenarios. Note that the time axis is in logarithmic scale.

was validated in our previous work performing autonomous navigation in an offline way with the CENTAURO robot.

However, for complex terrain scenarios, the computation of a long global plan can require several seconds to be found. To improve this, we have also considered a simpler global planner based on RRT [25] in which the nodes are expanded from the starting position to the goal one, taking into account a few constraints. In particular, to avoid having nodes that are too close or too far from each other, the distance between two nodes cannot be smaller than $0.05\,\mathrm{m}$ or longer than $0.12\,\mathrm{m}$ and should be guaranteed a minimum and maximum distance among the end-effectors, based on the platform considered. In addition, for each node randomly sampled, a validity check is performed to consider only the nodes for which there is at least one valid element in the traversability map for each contact point's neighborhood. We have not used directly this method because we do not check the feasibility among state transitions to speed up the execution of the planner. This will be handled by the local planner in a more exact way. Having a fast global planner allows to react faster to failures, without the need of stopping the robot's motion for a long time to wait for a new plan to follow. In addition, a global planner based on a random selection of nodes can be an appropriate choice because, in case of failures, it is difficult that it will select the same path, making the robot move and possibly find a new way to the goal.

The approach of this second method, which is proposed here, is generally coarser and it is not ensured to be feasible to track it completely, but the computational time can be significantly decreased by more than one order of magnitude, based on the complexity of the environment considered. On the other hand, to enhance feasibility, a solution is to use the search-based global planner, speeding it up by acting on the ϵ parameter of the ARA*, looking for less optimal solutions. This parameter can be modified based on the complexity of the environment with a tradeoff between the optimality of the solution and planning time.

Fig. 3 compares the two approaches in terms of planning time, considering two different values of ϵ for the search-based and also the global planner we used in our previous work (with bigger safety thresholds in the map, as explained in Section III). Note that, the tests without data are the ones in which the global planner was not able to find a solution or the solution required more than 100 s.

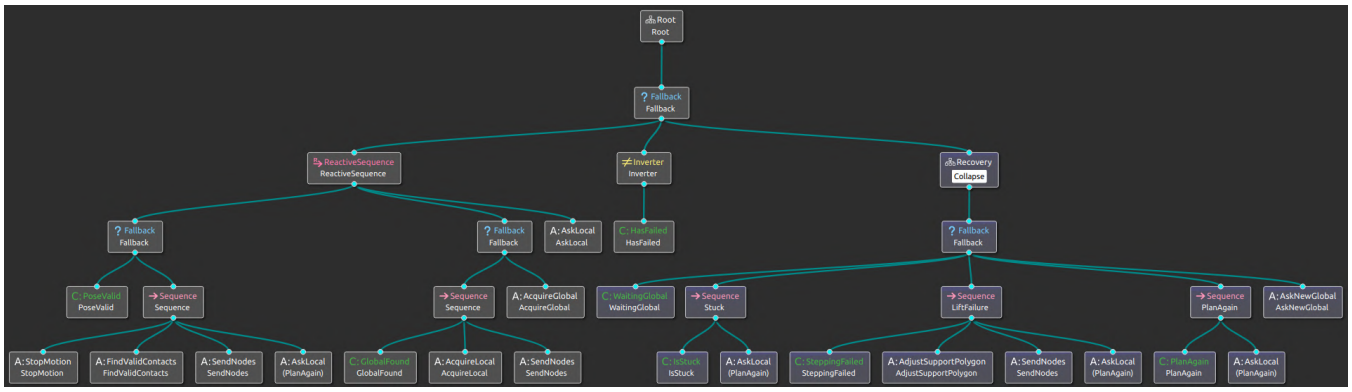

Fig. 4: Designed BT to manage online re-planning and react to failures. The left part of the tree is used to deal with the online updates of the planners and check for pose validity, while the right one is the recovery sub-tree.

C. Local Planner

The local planner is responsible for producing the sequence of actions that are sent to the robot in order to be executed. The implementation of the local planner is based on our previous work, using a search-based approach, considering a higher resolution for the map and for the discretization of the robot's movements compared to the search-based global planner, as explained in Section II-A.

The local planner is defined in an online fashion so that, while the robot is moving, the planner is continuously invoked to advance the state of the robot in which the starting position is extracted from the previous local plan found considering a node about $0.5\,\mathrm{m}$ ahead of the current state. The target position instead is obtained from the global plan and it is selected to be at a maximum distance of $1\,\mathrm{m}$ from the actual robot position. Since the two planners have different values in the parameters, the local planner cannot track exactly the global plan. To avoid constraining too much the local planner, it considers as a possible goal a small area around the node extracted from the global plan, without the need to track it completely. For example, a solution provided by the local planner can avoid an obstacle by moving around it, even if the global plan considers passing over the obstacle.

Once the local plan is found, it is acquired by the Planner Manager and sent to the Node Executor, which acquires the new information and continues the execution by sending the position and velocity references to the robot via our control framework CartesIO [26] and Xbot2 middleware [27].

D. Plan Evaluator

The Plan Evaluator is a safety module with the objective of checking if any of the contact points are in a non-valid area. If this is true, the PoseValid condition returns FAILURE and the motion is stopped. The evaluator, through the action FindValidContact, tries to find a valid position for the wheels, which is collected by the Planner Manager and sent to the executor, canceling the old plan received. In this way, the robot stops its execution, which can be dangerous and prone to failures, moving the wheel-feet end effectors to valid positions. Then the local planner starts again from this new, valid configuration. If the distance between the actual position of the wheel and the valid one is small enough, the movement is achieved via single-wheel driving; otherwise, a stepping maneuver can be considered for safety reasons.

E. Planner Manager

To manage the global and local planners, together with possible recovery methods, we decided to employ Behavior Trees, which were used in different applications ranging from human-robot interaction [28] to robot navigation in human search tasks [29].

The BT design can be seen in Fig. 4. The objective of this component is to manage the communication between the other components in order to guarantee the correct and safe execution of the task by the robot. The general flow, managed by the BT, in case of no failures, is the following:

- The BT acquires a target position that the robot has to reach from the user or an external module;
- The BT requests a solution to the global planner to reach the target, and acquires it;
- The local planner is invoked by the BT and searches a plan between the robot position and one of the first nodes in the global plan;
- The global plan is kept fixed during the robot movements while the local plan is updated continuously moving the starting point based on the previous local plan found and the goal point based on the global one.

The last step is iterated until the robot reaches the target point defined.

In case of failures during the execution, these are tracked by the Recovery sub-tree of the BT implemented and the PoseValid condition. Four main failure conditions are considered including: i) *Foot In Non-Valid Area*, ii) *Plan Not Found*, iii) *Robot Is Stuck*, or iv) a *Failure in Stepping*. In particular, if one of these failure conditions occurs, it is signaled by the Node Executor or the Planner Evaluator and caught by the BT, which checks the type of failure and reacts according to it. In the following, we illustrate the failures in more detail.

1) Foot in Non-Valid Area: due to errors in the localization, motion tracking, or new terrain features discovered while moving, one of the wheeled foot end-effectors of the robot may be in contact with a non-valid area. This can be dangerous because it can bring to collisions with the

environment but especially on elevated surfaces where non-valid areas are marked in the proximity of the edges of a platform. For this reason, a check is continuously run to stop the execution in case of such a condition.

If this failure is activated, the robot stops moving, and the FindValidContact action is invoked to find a valid position for the end-effectors that are in a non-valid region of the map, reshaping the support polygon. At this point, the Planner Manager directly sends the nodes to the Node Executor, and then restarts the online execution following the global path.

2) Plan Not Found: if the robot is moving and the local plan is not found in front of it (or a timer expires), a new call to the local planner is made for N times. In case of continuous failure, the Planner Manager stops the robot's motion and asks for a new global plan starting from the current robot position, restarting the execution. The same procedure is considered if the local planner fails while the robot has already stopped moving, but without iterating N times, since the robot is fixed and no new information is obtained. This is done automatically as a reactive recovery method to the failure. The new call to the global planner takes into account the new information acquired while the robot was moving and this can result in a different guiding path, especially if the sample-based global planner is used.

3) Robot Is Stuck: during whole-robot driving it may happen that errors in the localization or mapping bring the robot to hit an object. Since we do not have any force sensor on the end-effectors, we check this failure by looking at the robot's position. The actual robot's velocity is evaluated and if this is lower than a threshold, set to 0.005 m/s, the robot is considered to be stuck. Such a situation arises when the robot is constrained by the environment or when the cartesian controller cannot solve the requested velocity. In this case, the recovery behavior stops the robot and reverts the last action performed to arrive at the previous valid node in the plan. Now the local planner is invoked again, from the actual position and the robot can proceed with the execution. This failure is generally avoided by the Plan Evaluator, however, in case of obstacles not perceived by the sensors the robot can collide with the environment and stop the execution.

4) Failure In Stepping: while the robot is moving, we continuously monitor the CoM state and the support polygon of the robot to access the robot balance stability and detect potential conditions that can lead to falling incidents. In particular, before lifting the wheel to perform a stepping action, we evaluate the CoM distance from the borderline of the support triangle that will result from the lifting action. In case this distance is smaller than a minimum distance threshold, set to 0.04 m, this failure condition is triggered, suspending the execution of the stepping action. At this point, the robot moves the wheels with single-wheel actions to bring the CoM more inside the support triangle, checking the feasibility of the end-effectors on the local map. Then the CoM distance is evaluated again before executing the stepping action. If the distance of the CoM from the borderline of the support polygon is larger than the safe threshold, the stepping action is performed, otherwise, the failure is triggered again. In the latter case, the robot goes back to the previous valid state and plans again locally from that configuration.

In case these recovery behaviors do not allow the robot to continue with the execution, the local planner is invoked to drive the robot in the reverse direction of the current global plan, with the objective of moving away the robot from the obstacles making it easier to see the terrain area of interest. Up to now, we do not deal with any other type of failure conditions. In particular, if the robot is in the middle of a cluttered area and the planners (global and local) continually fail due to errors in the map or the complexity of the environment, the robot will be stuck and it will require human intervention to release it from such condition.

III. Experiments and Results

The presented framework was validated on the hybrid wheeled-legged robot CENTAURO, firstly in simulation environments and then on the real robotic platform.

A. Simulation

Under the Gazebo simulation environment, we perform several tests, considering different random cluttered terrain scenarios generated with a parametrized script that randomly selects objects[3] specifying the types and the desired number of the objects, the area covered, the minimum and maximum dimension and mass. Here we discarded some unfeasible scenarios with a collection of obstacles in a line that prevents the robot to traverse the scene. Some of the examples of the scenarios generated and considered can be seen in Fig. 5.

For all the simulations considered we used as a safety threshold for the mapping process, a value of $0.16\,\mathrm{m}$ for the global map and $0.14\,\mathrm{m}$ for the local map. Compared with our previous work we are able to obtain more robust navigation without the need to be far from obstacles. In fact, in the previous approach, only the global map was used. Since it has a coarser resolution, the inflation around obstacles and edges was bigger to avoid touching obstacles, having a safety threshold set to $0.24\,\mathrm{m}$. In this case, instead, we can reduce the safety distance threshold, permitting the robot to drive closer to obstacles without touching them, taking advantage of the higher resolution of the local map. In addition, the introduction of the Plan Evaluator and associated recovery methods enable us to be more robust against errors in the mapping and localization drifts.

We present the results obtained from 14 random terrain scenarios with different complexity set by the number of obstacles and their position arrangement on the map. Please note that the generated terrain scenarios, for which the global planners were not able to find a solution or the time required was too long, were discarded from the results presented in this section. This is related to the fact that the majority of the obstacles have a depth dimension close to the diameter of the wheel, therefore with the selected safety thresholds their top surfaces are considered as not valid, preventing the planner to find a solution. In addition, the existence of holes in the bricks causes differences in the elevation perceived,

[3] https://github.com/ADVRHumanoids/world_generator

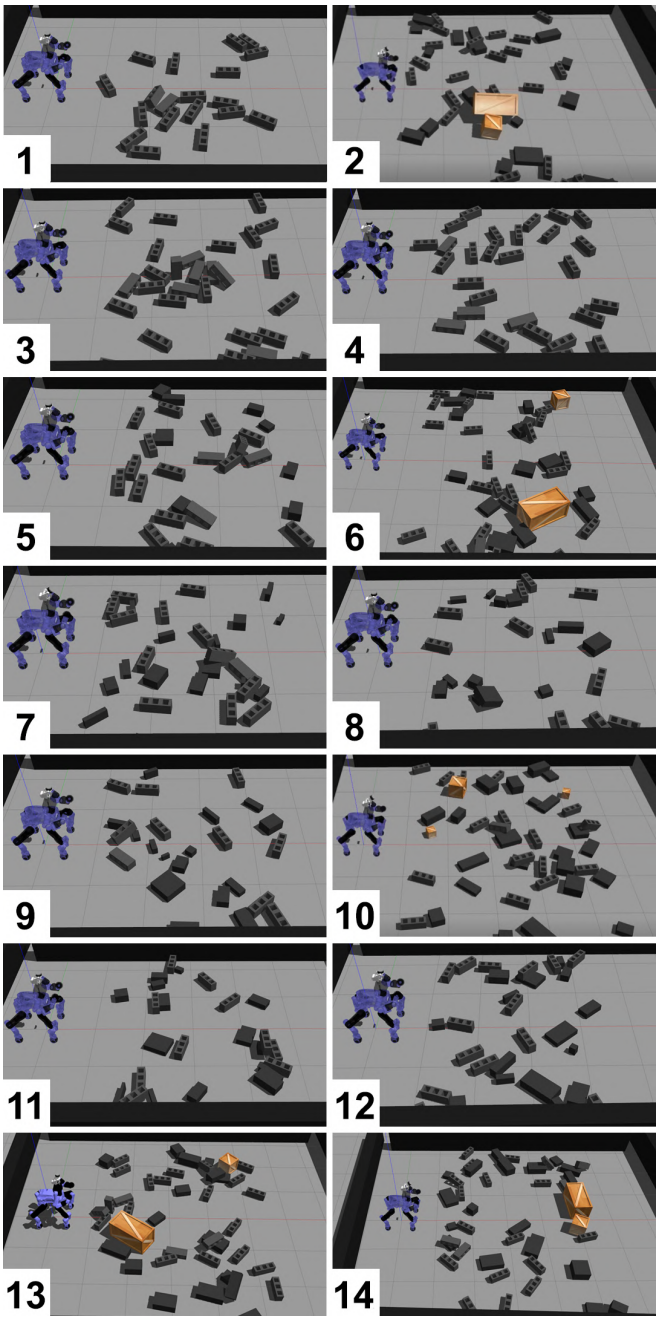

Fig. 5: Scenarios randomly generated with our terrain generator tool. The cluttered terrain arrangements contain several objects of random pose, dimension, and inertial properties.

resulting in non-valid areas. Two examples can be seen in Fig. 6.

For each of the scenarios considered, we run 3 different experiments and the results are summarized in Tab. I. In the scenarios considered, the global planner is always able to find a solution and, based on the terrain properties of each scenario, we can have a different number of calls to the local planner. Of course, the use of the ARA* based approach or the RRT-based one for the global planner can affect the result in terms of execution time, because of the sub-optimality of the sample-based implementation. In order to have a more

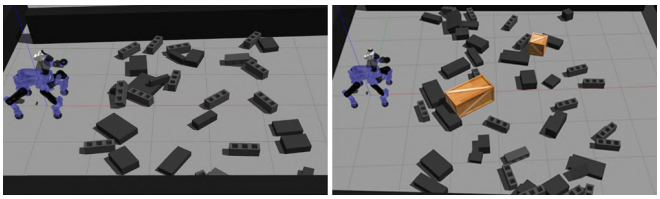

Fig. 6: Examples of simulated scenarios in which the global planner fails to find a solution.

TABLE I: Results of the simulation experiments.

Test	Succ. Rate	Exec Time [s]	Tot. Global Plans	Tot. Local Plans	Avg. Plan Time [ms]
1	3/3	142 ± 17	1 ± 0.5	244 ± 18	270 ± 612
2	3/3	165 ± 6	2 ± 0.5	333 ± 27	218 ± 748
3	3/3	148 ± 37	2 ± 1	314 ± 30	213 ± 469
4	3/3	131 ± 47	2 ± 1	277 ± 32	150 ± 321
5	1/3	249 ± 0	3 ± 0	617 ± 0	399 ± 924
6	3/3	115 ± 13	1 ± 0.5	273 ± 28	159 ± 725
7	2/3	261 ± 51	4 ± 2	482 ± 48	549 ± 814
8	3/3	140 ± 18	2 ± 1	245 ± 17	395 ± 762
9	3/3	205 ± 94	3 ± 2	314 ± 51	253 ± 617
10	3/3	199 ± 48	3 ± 2	369 ± 82	281 ± 685
11	3/3	239 ± 31	1 ± 0	315 ± 91	375 ± 545
12	3/3	173 ± 14	1 ± 0	223 ± 82	167 ± 560
13	3/3	167 ± 6	2 ± 1	348 ± 13	554 ± 958
14	2/3	184 ± 17	3.5 ± 0.5	416 ± 47	258 ± 641

accurate and feasible global solution, we used the search-based global planner for the scenarios in which the global planning time was lower than 1-2s. On the contrary, when the planning time was too high, we enabled the RRT-based global planner to obtain a fast solution.

Looking at the results, we can see that a low number of global plans acquired implies that repetitive failures in the local plan or execution are not very frequent, while a high number of local plans helps to validate the local planner and its online nature. The average local planning time is lower than 400ms, being able to adapt to the changes in the map, while its standard deviation is higher, since we set a timer for the local planner to 3.5s so, in constrained configurations, the local planner may need more time than the average one. The failure incidents experienced in the simulation are mainly related to collisions with the environment which occupy free navigation areas, and prevent the planner to find a solution.

With the goal of demonstrating the online updates, we also run an experiment in simulation in which the robot starts from an empty space and we manually add an obstacle in front of it. In this case, the new object is included in the map and the local planner is no more able to follow the global plan, so it stops the robot and requests a new global plan to deal with the new obstacle perceived. The execution is automatically adapted to the new plan for avoiding the obstacle and reaching the goal location previously assigned. A sequence of this behavior can be seen in Fig. 7. Here we placed the object a meter in front of the robot to have a clear acquisition from the sensors. If we place it much closer to the wheels, then the "Robot Is Stuck" failure would have been triggered, stopping the execution and correcting the plan as described in section II.

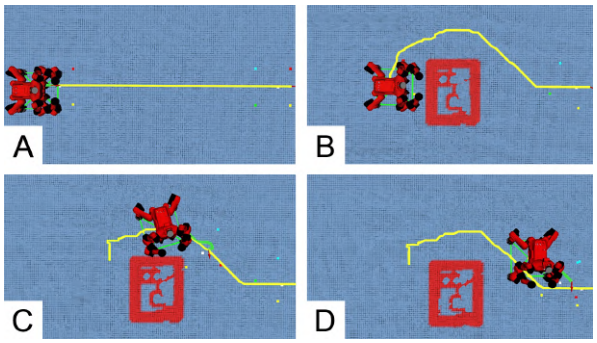

Fig. 7: Online updates of the global and local planners after perceiving a new object blocking the way in simulation. (A) Starting global plan is found, (B) an object is discovered and a new global plan is found, (C) local plan follows the new global plan, (D) the robot overcomes the obstacle.

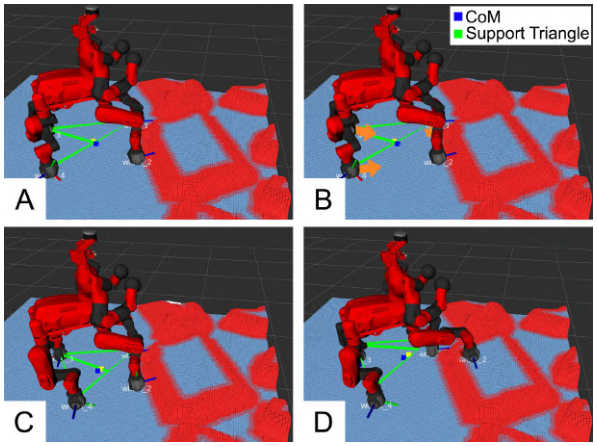

Fig. 8: Stepping recovery. (A) CoM is not in the support triangle before the lifting phase, (B) change position of the wheels, (C) support polygon changed, (D) stepping is accomplished.

In Fig. 8 instead, we can see the results of the "Failure in stepping" in which, after the failure is signaled, the wheels are moved to regulate the support polygon to increase the CoM distance from the borderline of the support polygon.

The demonstrated results would not have been possible to achieve with our previous work. Using only the offline global planner, the map considered would have needed higher thresholds, preventing possible solutions to be found in such irregular and cluttered environments. In addition, with offline methods, there is no chance to deal with new information perceived. The use of offline methods, as well as the absence of the BT, implies no updates of the plan, nor reactive recovery behaviors, failing the execution of the task in case of collisions or other failures.

B. Real Robot

After successfully evaluating the proposed methods in simulation, we perform validation trials on real environments using our CENTAURO robot. We carried out similar tests, dealing with scenarios of increasing complexity, starting from objects in a configuration to force the reshaping of the support polygon, to configurations in which stepping on and down obstacles was required, with environments in which the solution can not be trivially retrieved. In these experimental tests, we considered the search-based global planner to enhance feasibility when the planning time was small enough, and we used the same parameters of the simulations for the mapping, motion primitives, and planners. The only differences are related to:

- the update frequency of the map, which is half of the simulated one;
- the reduction of the motion execution speed on the robot for safety reasons as well as for favoring localization precision. Setting single wheel velocity to half of the simulated one that is 0.20 m/s.

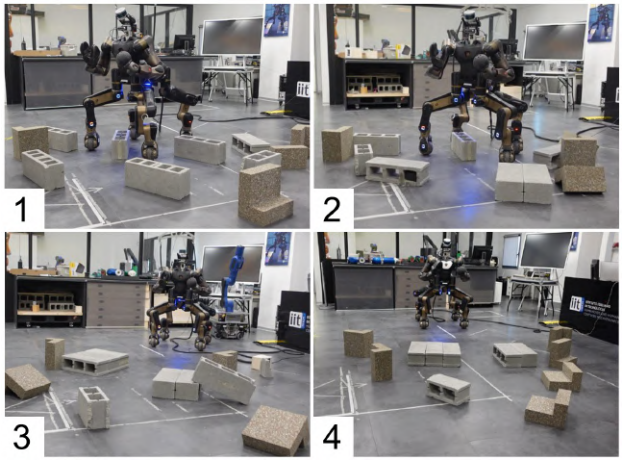

Fig. 9: Scenarios considered for real robot execution.

TABLE II: Results of the real robot experiments.

Test	Succ. Rate	Exec Time [s]	Tot. Global Plans	Tot. Local Plans	Avg. Plan Time [ms]
1	4/4	242 ± 94	2 ± 1	103 ± 31	450 ± 932
2	3/4	225 ± 55	2 ± 1	79 ± 21	399 ± 777
3	1/4	335 ± 0	2 ± 0	115 ± 0	415 ± 879
4	2/4	275 ± 92	4 ± 2	98 ± 5	467 ± 889

Fig. 9 introduces some of the scenarios we have considered. All the environments were crossed without falling down or other major collision incidents thanks to the Plan Evaluator that ensured to stop the robot when close to the edges of the obstacles and moved the wheels far away if very close or in contact with them. In particular, in scenarios 2, 3, and 4, the robot is required to step on the bricks to complete the task. The small area of the brick's top surface makes the task more constrained since the robot needs to correctly negotiate the shape of the support polygon to keep the equilibrium while crossing the obstacles without colliding with them. A video of the results achieved, showing the real robot's execution, together with Rviz view and BT monitoring, can be seen at the following link [4].

In Tab. II are summarized the results obtained on the real robot experiments. Also for the real robot, we can see that the

[4] https://youtu.be/tr5gCxs78vg

number of global plans requested is close to 1 demonstrating that the online updates of the planner and the autonomous recovery are able to follow the global candidate and unstuck the robot in case of the failure conditions considered. Since on the real robot, the update frequency of the local map is half of the simulation, we had to increase the timer's duration for the local planner to 7.5 s, resulting in a lower number of local plans found within the execution time.

The failures we experienced on the real platforms, in particular in scenarios 3 and 4, are mainly related to the errors in the localization (visual odometry), especially in the roll, pitch, and elevation (z) of the pelvis. In fact, the local planner was able to find a solution to move on and arrive at the goal, however, due to the z error, a wheel was sometimes lifted of few centimeters after stepping actions. The error in the pelvis roll and pitch angles instead, causes the pelvis to be wrongly oriented, making the ankles to be not perpendicular to the ground forcing to stop the execution to avoid overloading the robot's ankle joints.

IV. Conclusions

In this paper, we presented the concept and implementation of a hybrid online planning framework, which enables autonomous navigation for wheeled-legged robotic platforms. The framework was validated both in simulation and on the real CENTAURO robot, demonstrating the capabilities acquired with respect to our previous work in terms of planning performance, accuracy, and recovery during failures. Thanks to the introduction of the BT the recovery methods can be addressed in a reactive way, improving safety and drastically reducing the incidents in which the robot is blocked and cannot continue its navigation, improving the robot autonomy and reducing the need for human intervention. Future works will address the introduction of manipulation actions to rearrange obstacles when navigation is not possible. In addition, we will consider the introduction of moving/dynamic obstacles, and the execution of outdoor experiments, validating the framework in more realistic outdoor terrain environments.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101016007 CONCERT and the Italian Fondo per la Crescita Sostenibile - Sportello "Fabbrica intelligente", PON I&C 2014 - 2020, project number F/190042/01-03/X44 RELAX.

References

[1] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Trans. Robot.*, vol. 34, no. 3, pp. 586–601, 2018.

[2] K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *I. J. Robotic Res.*, vol. 27, pp. 1325–1349, 11 2008.

[3] M. Cognetti, P. Mohammadi, and G. Oriolo, "Whole-body motion planning for humanoids based on com movement primitives," in *IEEE Int. Conf. on Humanoid Robots*, 2015, pp. 1090–1095.

[4] P. Ferrari, M. Cognetti, and G. Oriolo, "Anytime whole-body planning/replanning for humanoid robots," in *IEEE Int. Conf. on Humanoid Robots*, 2018, pp. 1–9.

[5] W. Reid, R. Fitch, A. Göktoğan, and S. Sukkarieh, "Sampling-based hierarchical motion planning for a reconfigurable wheel-on-leg planetary analogue exploration rover," *Journal of Field Robotics*, vol. 37, 10 2019.

[6] S. Xin and S. Vijayakumar, "Online dynamic motion planning and control for wheeled biped robots," in *IEEE Int. Conf. Intell. Robots Syst.*, 2020, pp. 3892–3899.

[7] C. Rosmann, A. Makarow, and T. Bertram, "Online motion planning based on nonlinear model predictive control with non-euclidean rotation groups," in *Eur. Control Conf.*, 2021, pp. 1583–1590.

[8] L. Campos, D. Gomez-Gutierrez, R. Aldana, R. Guardia, and J. Vilchis, "A hybrid method for online trajectory planning of mobile robots in cluttered environments," *IEEE Robot. Automat. Lett.*, vol. 2, pp. 935–942, 04 2017.

[9] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," *The Int. Journal of Robotics Research*, vol. 30, no. 2, pp. 236–258, 2011.

[10] M. A. Arain, I. Havoutis, C. Semini, J. Buchli, and D. G. Caldwell, "A comparison of search-based planners for a legged robot," in *Int. Workshop on Robot Motion and Control*, 2013, pp. 104–109.

[11] C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Motion planning for quadrupedal locomotion: Coupled planning, terrain mapping, and whole-body control," *IEEE Trans. Robot.*, vol. 36, no. 6, pp. 1635–1648, 2020.

[12] N. Kashiri, L. Baccelliere, L. Muratore, A. Laurenzi, Z. Ren, E. M. Hoffman, M. Kamedula, G. F. Rigano, J. Malzahn, S. Cordasco, P. Guria, A. Margan, and N. G. Tsagarakis, "Centauro: A hybrid locomotion and high power resilient manipulation platform," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1595–1602, 2019.

[13] M. Schwarz, T. Rodehutskors, M. Schreiber, and S. Behnke, "Hybrid driving-stepping locomotion with the wheeled-legged robot momaro," in *IEEE Int. Conf. Robot. Autom.*, 2016, pp. 5589–5595.

[14] T. Klamt and S. Behnke, "Anytime hybrid driving-stepping locomotion planning," in *IEEE Int. Conf. Intell. Robots Syst.*, 2017, pp. 4444–4451.

[15] V. S. Raghavan, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Agile legged-wheeled reconfigurable navigation planner applied on the centauro robot," in *Int. Conf. Robot. Autom.*, 2020, pp. 1424–1430.

[16] M. Colledanchise and P. Ogren, *Behavior Trees in Robotics and AI: An Introduction*, 07 2018.

[17] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[18] O. Biggar, M. Zamani, and I. Shames, "An expressiveness hierarchy of behavior trees and related architectures," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5397–5404, 2021.

[19] Z. Zhou, D. J. Lee, Y. Yoshinaga, S. Balakirsky, D. Guo, and Y. Zhao, "Reactive task allocation and planning for quadrupedal and wheeled robot teaming," in *IEEE Int. Conf. on Automation Science and Engineering*, 2022, pp. 2110–2117.

[20] Z. Gu, N. Boyd, and Y. Zhao, "Reactive locomotion decision-making and robust motion planning for real-time perturbation recovery," in *Int. Conf. Robot. Autom.*, 2022, pp. 1896–1902.

[21] A. De Luca, L. Muratore, and N. G. Tsagarakis, "A hybrid primitive-based navigation planner for the wheeled-legged robot centauro," in *IEEE Int. Conf. Intell. Robots Syst.*, 2022, pp. 7904–7911.

[22] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *IEEE Robot. Automat. Lett.*, vol. 3, no. 4, pp. 3019–3026, 2018.

[23] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," 2014.

[24] M. Likhachev, G. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Proc. Int. Conf. on Neural Information Processing Systems*, ser. NIPS'03. Cambridge, MA, USA: MIT Press, 2003, p. 767–774.

[25] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[26] A. Laurenzi, E. M. Hoffman, L. Muratore, and N. G. Tsagarakis, "Cartesi/o: A ros based real-time capable cartesian control framework," in *Int. Conf. Robot. Autom.*, 2019, pp. 591–596.

[27] A. Laurenzi, D. Antonucci, N. G. Tsagarakis, and L. Muratore, "The xbot2 real-time middleware for robotics," *Robotics and Autonomous Systems*, vol. 163, p. 104379, 2023.

[28] P. Tulathum, B. Usawalertkamol, G. A. G. Ricardez, J. Takamatsu, T. Ogasawara, and K. Matsumoto, "Human-robot interaction system for non-expert users in convenience stores using behavior trees," in *IEEE Int. Symposium on System Integration*, 2022, pp. 1072–1077.

[29] M. Stuede, T. Lerche, M. A. Petersen, and S. Spindeldreier, "Behavior-tree-based person search for symbiotic autonomous mobile robot tasks," in *IEEE Int. Conf. Robot. Autom.*, 2021, pp. 2414–2420.