Received September 3, 2021, accepted November 23, 2021, date of publication December 30, 2021, date of current version January 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3139438

Reconfigurable and Agile Legged-Wheeled Robot Navigation in Cluttered Environments With Movable Obstacles

VIGNESH SUSHRUTHA RAGHAVAN[1,2], DIMITRIOS KANOULAS[3], (Member, IEEE), DARWIN G. CALDWELL[4], (Fellow, IEEE), AND NIKOS G. TSAGARAKIS[1], (Member, IEEE)

[1]Humanoids and Human-Centered Mechatronics (HHCM) Laboratory, Istituto Italiano di Tecnologia (IIT), 16163 Genova, Italy
[2]Department of Information Engineering, University of Pisa, 56122 Pisa, Italy
[3]Department of Computer Science, University College London (UCL), London WC1E 6BT, U.K.
[4]Department of Advanced Robotics, Istituto Italiano di Tecnologia (IIT), 16163 Genova, Italy

Corresponding author: Vignesh Sushrutha Raghavan (v.raghavan@studenti.unipi.it)

This work was supported in part by the CENTAURO H2020 European Union (EU) Project under Grant 644839, and in part by the Italian/Singapore MoD Project PHOLUS.

ABSTRACT Legged and wheeled locomotion are two standard methods used by robots to perform navigation. Combining them to create a hybrid legged-wheeled locomotion results in increased speed, agility, and reconfigurability for the robot, allowing it to traverse a multitude of environments. The CENTAURO robot has these advantages, but they are accompanied by a higher-dimensional search space for formulating autonomous economical motion plans, especially in cluttered environments. In this article, we first review our previously presented legged-wheeled footprint reconfiguring global planner. We describe the two incremental prototypes, where the primary goal of the algorithms is to reduce the search space of possible footprints such that plans that expand the robot over the low-lying wide obstacles or narrow into passages can be computed with speed and efficiency. The planner also considers the cost of avoiding obstacles versus negotiating them by expanding over them. The second part of this article presents our new work on local obstacle pushing, which further increases the number of tight scenarios the planner can solve. The goal of the new local push-planner is to place any movable obstacle of unknown mass and inertial properties, obstructing the previously planned trajectory from our global planner, to a location devoid of obstruction. This is done while minimising the distance traveled by the robot, the distance the object is pushed, and its rotation caused by the push. Together, the local and global planners form a major part of the agile reconfigurable navigation suite for the legged-wheeled hybrid CENTAURO robot.

INDEX TERMS Hybrid legged-wheeled navigation, legged-wheeled robotics, navigation among movable objects, path planning, quadrupedal robot, reconfigurable navigation planning.

I. INTRODUCTION

Mobile robots often need the speed of wheeled rolling motion and the agility of legged locomotion. Especially in environments cluttered by obstacles as well as in unstructured or uncertain environments, hybrid robots that comprise legs and wheels have the advantage of dealing with most of the challenges during navigation. Although, rapid sensing and planning accurately paths, are required for stable, safe, and efficient mobility in such environments.

The associate editor coordinating the review of this manuscript and approving it for publication was Shuhuan Wen.

In environments that lack free space for navigation, as is the case in cluttered spaces, planning paths for rolling robotic platforms is a very challenging task. Sometimes, the motion is insufficient or infeasible due to collisions or computationally heavy due to a large number of obstacles in the surrounding environment. The shape and size of obstacles are also major factors to be considered. This is the main limitation of robotic platforms that have only fixed wheels on the robot's main body. The difficulty for wheeled robots in navigating continuously increases considerably in such tight spaces cluttered with obstacles of different sizes and shapes. On the contrary, legged robots are gaining more attention lately given their ability to use their high

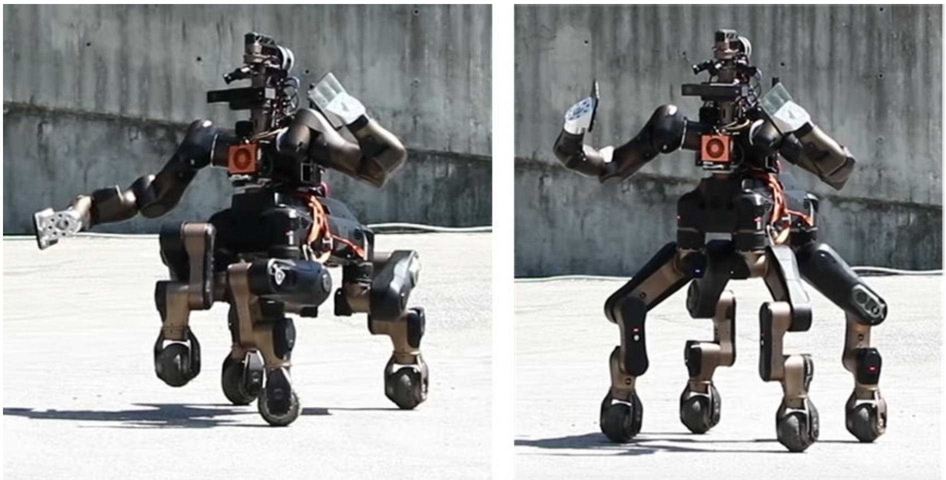

FIGURE 1. **The CENTAURO robot, with its hybrid legged-wheeled mobility modules.**

Degrees-of-Freedom (DoF) legs to negotiate uneven terrains and obstacles via locomotion. Unfortunately, planning such high-DoF trajectories might be computationally expensive, while constraints in the stability and safety of the robot need to be considered at all times. In this work, we use a hybrid quadruped robot (CENTAURO [1] in Fig. 1) that has wheels at the end of each foot. In this way, the robot can roll, but at the same time, it can alter its height and the configuration and orientation of its legs (i.e., its footprint polygon) to walk in narrow passages, over obstacles, or even push objects away in order to create free space. The control of such hybrid systems might be challenging, given that it might require switching between rolling and stepping control, with heavy path planning computations.

Path planning and navigation for wheeled robots have been extensively studied in the past [2]. The majority of those works were focused on 2D costmaps or occupancy grids, in order to allow mobile robots to transit among obstacles [3]. These maps/grids were a result of fast and robust Simultaneous Localization and Mapping (SLAM) methods, that were used to produce autonomous path planning for such robots. Even though these methods are used efficiently enough for a robot whose wheels form a fixed footprint polygon, there is still a research gap for hybrid wheeled-legged robots that can continuously modify their footprint polygon while navigating an environment with real-time computations (we review those in the next section). This is especially true, given that heuristic graph-based methods, e.g., A*-based [4], that produce the required paths by treating obstacles and objects as non-traversable areas in the map. In this way, interactions with obstacles (e.g., pushing) or transition via other actions (e.g., stepping-over) are not considered. The ideal scenario for a hybrid wheeled-legged robot is to roll quickly among obstacles (either with fixed footprint polygon or a reconfigurable by expanding for wide and narrowing for narrow pathways), in order to reduce transit time, and only use the legs for stepping when it needs to push or bypass an obstacle via stepping.

This paper is split into two parts. Firstly, we describe our previously introduced work on a novel global path planner for hybrid wheeled-legged navigation [5], [6]. The goal of the method is to reconfigure the footprint polygon of the robot, by tweaking the leg configuration and the height of the base from the ground, in order to navigate through obstacles that cannot be avoided. Those obstacles might be of height lower than the robot's pelvis (such that the robot can pass over them), wider than the robot's current footprint polygon (such that the robot can expand its feet to pass over them), or form spaces shorter than the robot's current footprint polygon (such that the robot can narrow its feet to pass via them). Allowing the robot to navigate like that in the environment allows it to find the shortest paths to the destination, without the need to take large detours around them. Our proposed methods are based on the A* [4] and Theat* [7] path planners and the robot's capability to dynamically change its footprint polygon. The introduced shortest path heuristics consider the trade-off between reconfiguring the footprint polygon and taking a detour around obstacles. In particular, two variations of the algorithm are presented. In the first one, an A* based planner using simple symmetric reconfigurable rectangle footprints is introduced. This simple method allowed the hybrid wheeled-legged robot to navigate among wide and low-height obstacles and in narrow spaces, using the rectangular footprint polygon symmetry reconfiguration. In the second one, we introduce a method to deal with spaces, where keeping the symmetry of a rectangular footprint polygon is not possible, e.g., when there is not much free space between a narrow passage and a subsequent wide obstacle. Keeping the symmetric solution would have resulted in a collision as the robot would not have had sufficient space to reconfigure. Thus, a version with improved computationally efficiency is presented, where the front and back wheel pairs may follow independent control, additionally also using the functionality of the robot that allows for omnidirectional motion of its legged-wheels. The result is that the robot is able to plan paths in tight passages and be agile enough to navigate via multiple obstacles.

In the second part of the paper, we introduce a new method to further increase the number of tight scenarios the robot can solve. A local push planner is designed to allow the robot to move any small movable obstacle outside the path planned by the global planners described above. The method uses line equations and rectangular free-space partitions, object and contact surface parameterization, as well as pixel coordinate geometry. In this way, it can be determined where in a rectangular free space partition devoid of any other obstacles, can the movable object be placed, such that there is no more obstruction. Thus after the push, the robot can either go around or negotiate the movable object so as to rejoin the original plan. The introduced local planner generates a sequence of pushes that try to achieve three objectives, namely 1) the robot travels the minimum distance to perform the push and rejoin the disturbed pre-planned trajectory, 2) the overall distance the object is pushed is minimum, and 3) the rotation of the object is kept to a minimum. The planner attempts to keep the rotation to a minimum given that it is not using the mass or inertial properties. To create the push plan, the algorithm uses the obstacle centroid, the locations of push contact points from the centroid, and standard kinematics to determine the pushes and the effects of the pushes.

The performance of our global planner is tested in simulation and on the real CENTAURO robot for different obstacle scenarios. Furthermore, various image-based simulations and Gazebo[1] simulations of the CENTAURO robot using the newly proposed local push planner to clear obstructing obstacles are demonstrated and discussed. The remainder of the paper is organized as follows. First, in Sec. I-A, we present related work on reconfigurable planning and navigation among movable objects. In Sec. I-B, we introduce the CENTAURO robot. In Sec. II, we present our obstacle negotiating A*-based path planner algorithm and its next iteration namely Trapezium Line Theta* planner. Following this, the newly developed local planner prototype algorithm is explained in detail in Sec. III. This is followed by simulation and experimental results in Sec. IV and finally, in Sec. V we conclude with a discussion and future directions.

A. RELATED WORK

Navigation planning is one of the most well-studied areas in robotics. Shortest path planners, such as Dijkstra algorithm [8], are nowadays outperformed by more advanced ones, such as the A* [4], where optimal and efficient planning is plausible. This class of planners discretize the space into grids and use heuristics to calculate travelling cost distances between grid points towards a goal. Improvements of A* have been presented too, such as ThetaStar [7], which allows continuous angle space exploration. Another class of methods compute costs backwards from the goal grid point. Such methods are the ARA* [9], D* [10], and D*lite [11].

The Probabilistic Road Map (PRM) [12] is a popular class of planning algorithms that are typically used for manipulators with many degrees of freedom, while Random Rapidly Exploring Trees (RRT) [13] operate by incrementally building a tree and connecting a randomly sampled point to the closest point on the tree in case a collision-free path exists. Many future studies like those in [14] and [15], presented variations that improved upon the original RRT. While PRM and RRT ensure completeness of the solution, when the search space dimension is high, the number of computations is also very high. Furthermore, additional post-smoothing is needed to make the path found by RRT suitable for robot traversal. While A* finds the path quicker, in higher dimensions, the graph construction and heuristic cost calculation can be very complicated. For our current problem of legged wheeled motion in a cluttered space but flat ground, we would need a multi-dimensional search to take into consideration the ability to change the base height and footprint polygon configuration.

All the aforementioned planning methods have been used extensively to generate navigation paths for robots that are reconfigurable. For instance, in [16], [17] used reconfiguration for tracked robots but only in order to avoid or climb on objects. In [18], a tracked robot with an arm is used with an A* planner to modify its center of mass (but not the robot's footprint polygon) in order to safely deal with uneven terrains. More recently, in [19] a robot that changes its shape is presented in order to clean ground surfaces. A modified A* method produces plans in order to increase the covered cleaning surface at each instance. Compared to our work, and given the reconfiguration changes the footprint polygon, this only considers obstacles avoidance, bypassing them. Snake robots have been also used, given the natural reconfigurability of their body. In [20], a wheeled snake robot was used with an RRT* planner, to climb stairs and move around objects or narrow passages by changing its configuration. The used following-a-leader approach did not allow for radical polygon shape changes with 12 s planning times. Multi-modal PRM is used in [21] to plan simulated paths for the ATHLETE and HRP2 robots. The ATHLETE is a hybrid wheeled-legged robot, on which flat terrain and stair negotiation was studied in simulation. Planning times were around 90 s, but dealing with obstacles was not studied.

While research on path planning for footholds of humanoid robots was studied heavily in the past [22], [23], the development of hybrid robots with wheels and legs increased the interest for real-time path planners that can reconfigure footprint polygon. Closer to our work, in [24], [25], driving and stepping motions for two, hybrid wheeled-legged robots (CENTAURO and MOMARO), were studied. Using ARA* [9], the robots were alternating between rolling and stepping, without changing the width of the footprint polygon, but rather using rolling-stepping actions in parallel to

[1] http://gazebosim.org/

optimize traversability efficiency. Although, this work, compared to ours, does not address the problem of changing the footprint polygon in any direction to deal with wide obstacles or narrow spaces. In [26], an interesting approach of modelling the robot as a deformable bounding box was presented. The robots (hexapod and quadruped) were able to modify their height and width, using a CHOMP planner [27], to plan collision avoidance in narrow passages and ceilings, as well as to plan going over obstacles. Their newer work [28] improved upon this by using sampled poses, pose optimization and trajectory smoothing in a hierarchical manner, to give their legged robots like ANYMal the agility to accommodate themselves through small gaps, and traverse cluttered spaces like collapsed structures safely. The primary difference beyond the obvious difference in the robots used in the research and our work is that the above mentioned two works focus on optimising for safe obstacle collision avoidance to allow for agile and safe traversing. Whereas our algorithms focus on the distance costs involved in avoiding the obstacles versus negotiating the obstacles by changing the robot footprint configuration. Furthermore, while the algorithms in [26], [28] involve whole-body planning with a much more complex search space, in our algorithms we plan primarily for the lower body of the CENTAURO and attempt to reduce the search space to make the algorithms computationally moderate and efficient.

In this paper, we aim at novel methods that use the reconfigurable legs capability of a hybrid wheeled-legged robot, namely the quadruped CENTAURO. Our goal is to minimize planning times, through flexible cost functions (thus, not using PRM and RRT). For this reason, we use both A* and Theta* planners for low-dimensional search space path searching. In such a way, we are able to narrow or extend the footprint polygon in order to deal with various passages and obstacles in the environment. This is done by utilizing costmaps, Octomap filtering [29], and simple 2D obstacle segmentation, in order to find safe robot polygons. The search space is then reduced either using rectangular footprint polygon-based search in an A* framework [5] or a more flexible trapezium-like footprint search in the more versatile Theta* planner framework [6] to allow any-angle motion as well as independent motions for the front and back wheel pairs. We use Lidar point cloud data in order to create costmaps and segmentations in the 2D projections of the 3D point clouds. We would like to note that these methods have been previously used to plan paths over low-lying rough surfaces, using the CENTAURO robot [30].

A key aspect of the second part of this paper is that it is closely related to the popular Navigation Among Movable Objects (NAMO) problem, introduced originally in [31], [32]. The problem is NP-hard, even in its simple form, thus, graph-based optimizations were introduced. After his thesis [33], Stilman, focused on this problem through a series of interesting papers, e.g. [34]–[36], that consider unknown environments or locally optimal paths. The methods have been used on humanoid robots and wheeled ones, up to an extent (either with known environments or via open-loop controllers), as well as for robot arm manipulation [37]. Recently they have been extended for socially-aware navigation [38], [39], navigation using scene affordances [40], or even sim2real navigation [41]. In our paper, we are utilizing an object pushing action planner, based on the NAMO algorithm, introduced in [35], [38], and perform the pushes using the legs of the hybrid robot. We use similar region partitioning and setting up of the push sequences. Furthermore, we adapt it for our hybrid legged-wheeled robot by clearly defining the types of pushes the robot can execute and using these actions, each of which is a clearly defined push type, as principal components of the push sequence. Our algorithm is not as exploratory or optimal as those in [35], [38], as we use conservative region partitioning, and simple object parametrization and line geometry to merely move the object from a pre-planned trajectory, or to a position it can be avoided/negotiated. This makes the algorithm computationally moderate and fast. It is to be noted unlike most NAMO algorithms, we have no knowledge of the obstacle mass and inertial properties. These planned push sequences are integrated parallelly into the navigation suite consisting of the global Trapezium Line Theta* planner that changes the configuration of the footprint polygon, resulting in a unified re-configurable path planner.

B. THE CENTAURO ROBOT

Given that the hybrid wheeled-legged robot CENTAURO, will be used in this paper, we firstly introduce it. It is a 42 Degrees-of-Freedom (DoF) robot. Each leg has 7 motors and actuated wheels. We assume after experimental justification that the maximum footprint polygon width for which the robot is stable is 1.1m, while the minimum is 0.44m. Similarly, the maximum height of the robot's main body is 1m from the ground. The robot includes a VLP-16 Velodyne Lidar on its head, which also rotates, to provide dense point cloud data at 40Hz. It also has three realsense D435 cameras, two of which are attached close to the hind legs looking towards the front legs and one looking down in the front center of the robot on the pelvis. The robot and all its sensors are calibrated, while an odometry-based system provides the position of the robot in the world.

II. GLOBAL RECONFIGURABLE PATH PLANNER

In this section, we introduce the global path planner that allows the hybrid robot to reconfigure its legs while rolling, to facilitate navigation through challenging obstacles and passages. The incremental development of the re-configurable local planner was done in two steps. The first prototype named Obstacle Negotiating A* allowed for the robot to always assume only rectangular footprints. The length and breadths of the footprints were varied according to the navigation plan to perform collision-free navigation. The robot plans involved expanding over low-lying obstacles, narrowing into tight spaces and total obstacle avoidance. The second prototype named Trapezium Line Theta*, used trapezium-like footprints to further improve the manoeuvrability of the robot in tighter scenarios when compared to the first prototype, thereby making the robot capable of traversing more cluttered environments. For ease of presentation, the two algorithms will be named Prototype 1 and Prototype 2.

In this section, we will outline step planner algorithms developed for both prototypes. In both cases, the methods used for mapping and creation of the navigation search map were common. The prototype algorithms differed in the base planner algorithm that was used and in the footprint search method. While in the first prototype, a rectangular footprint based search was combined with an A* planner [4], in the second prototype a trapezium like footprint based search was combined with a Theta* planner [7] to plan for valid robot footprints.

A. SEGMENTED MAP CREATION

The creation of the map for performing planner search is common for both the prototype algorithms. Using the Octomap [29] and *move_bases*[2] software packages, point-clouds from the VLP velodyne laser are processed into two 2D image costmaps as follows. The first costmap consists of all obstacle points, whereas the second costmap consists of only points at a height higher than a threshold h_{obs}. For most of our experiments and simulations h_{obs} is set to be 0.4m to allow for comfortable safe clearance by the base of the robot if it is to go above a low lying obstacle. The two 3D costmaps are then converted to grayscale images – I_a and I_b. In these two images, the white pixels represent obstacles and the black pixels represent the free space. A third image I_c is derived as $I_c = I_a$ XOR I_b, which roughly consists of obstacle points of height lower than h_{obs}, which can be easily cleared by the robot pelvis. A fourth image $I_d = I_a - I_c$ is obtained followed by the final operation to obtain the segmented map as $I = 0.5 \times I_c + I_d$. Hence the segmented map classifies every point on the map as free space (black), clearable obstacle (grey) or non-clearable obstacle (white) using 3 colour codes (see Fig. 2 for an example). This image is used as the main 2D map for the introduced variable configuration path planning development and its pixels form the planner's query points.

B. SEARCH METHODOLOGY FOR PROTOTYPE 1

The search for possible paths is done pixel-wise in the created 2D segmented map. The robot is considered a single entity and with a modifiable rectangular footprint with a constant constraint on the sum of the length and width. The height of the base of the robot above the ground is determined using simple ratios of the width of the rectangle and maximum and minimum widths of the rectangular footprint. Thereby, the search is reduced to only valid collision free widths. An 8-neighborhood search is done to determine motion from one pixel to another in an attempt to determine the best path. In this prototype, the robot is always aligned in the direction of motion and since a simple A* based grid search was used, the orientations assumed by the robot were limited. The following factors are used to determine the cost:

- Deviation from the line joining the current position to the goal.
- Variation in the rectangular footprint dimensions.
- Total distance being travelled.
- Change in orientation (straighter paths preferred).

Standard A* cost functions for calculating the cost of moving from parent coordinates (x_p, y_p) to child coordinates (x_c, y_c) are used, with the goal coordinates being (x_g, y_g). This refers to the cost of occupying the coordinates and the heuristic costs. For Prototype 1, they are determined as follows:

$$
\begin{aligned}
g(x_c, y_c) &= g(x_p, y_p) + W_t \times |\delta\theta_o| + W_c \times |\delta w| \\
h(x_c, y_c) &= \sqrt{(x_g - x_c)^2 + (y_g - y_c)^2} + W_g \times \left|\delta\theta_g\right| \quad (1)
\end{aligned}
$$

where $\delta\theta_o$ is the normalised absolute orientation change of the robot, $\delta\theta_g$ is the normalised orientation change away from the straight line to the goal from the parent point, δw, is the change in the width of the robot footprint needed to perform obstacle negotiation if needed to obtain a collision-free path, while W represent the respective weights which are used to control the planner behaviour. In-depth descriptions of the cost equations and the algorithm can be found in [5].

C. SEARCH METHODOLOGY FOR PROTOTYPE 2

Similar to Prototype 1, the second one is also using the 8-neighborhood search, but instead of A*, Theta* is used as the base planner. This allows for paths to be more flexible and allows for any-orientation motion, unlike the first prototype where the motion orientations were limited to multiples of $\frac{\pi}{4}$. Furthermore, the robot's footprint splits into two rectangular ones, each for the front wheel pair and the back wheel pair. This is a difference that is depicted in Fig. 2. This allows for plans that modify the front and back wheel pairs independently giving rise to trapezium-like footprints when necessary.

In addition to the costs mentioned for Prototype 1, the following are the added costs used for calculating low-cost paths:

- Variation in individual wheel width pairs.
- Possibility to reach an intermediate search point without changing orientation (omnidirectional motion without orientation change for the whole robot is preferred).

The cost functions of Prototype 1 are used here too, except $|w|$ which is used to represent the absolute sum of the change of the front and back wheel widths. Furthermore, since Theta* planner is used, the parent and the child node need not be immediate neighbours. The best parent node is chosen using back-tracing, thereby allowing for all possible orientations of motion from parent to child. A further check is done to see if the motion from the parent to child node can be done omnidirectionally. Detailed explanations of the

[2] http://wiki.ros.org/navigation

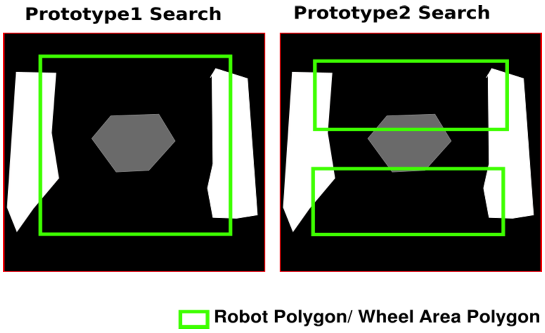

FIGURE 2. Difference in the search methodology of the two prototypes. In Prototype 1 the maximum of the entire robot polygon area around the query point is searched to obtain a collision-free configuration. Whereas in prototype two, two separate neighbourhoods in the vicinity of the front and back wheel pairs are searched to obtain collision-free widths of the wheel pairs.

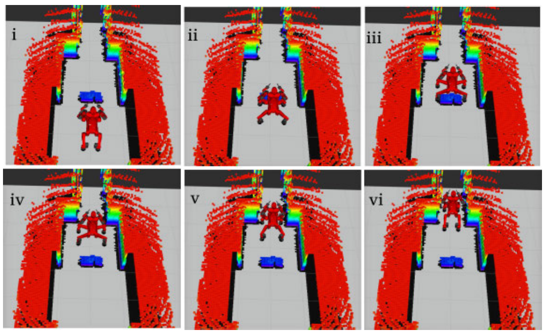

FIGURE 3. Prototype 1 planner executing a plan consisting of expanding over a low-lying obstacle and then narrowing into the corridor.

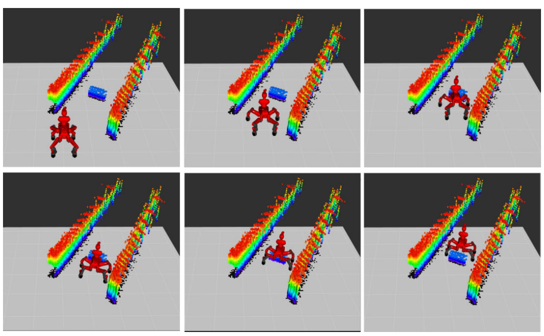

FIGURE 4. Prototype 2 planner allowing for any-angle motion alongside independent front and back wheel pair operation.

variations of the height calculations and the cost functions compared to Prototype 1 can be found in [6].

The difference in the capabilities of the two prototype can be seen in the Figs. 3, 4, and 5. Fig. 3 shows the robot expanding over a low-lying obstacle and then narrowing into the corridor. However if the distance between the end of the low-lying obstacle and the start of the corridor had been lesser than the maximum length of the robot in its narrowest polygon, the planner would have been unable to find any path. Furthermore, if the narrow corridor was angled at any orientation other than multiples of $\frac{\pi}{4}$, once again navigation would

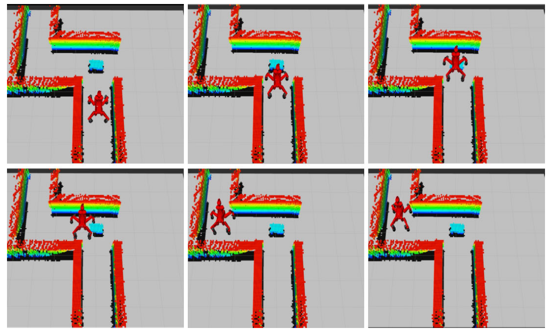

FIGURE 5. Prototype 2 planner executing omni-directional motion and independent front and back wheel pair motion with minimal cost, in a tight constrained environment.

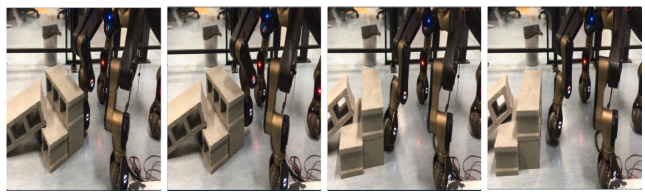

FIGURE 6. Sequence depicting the robot pushing a set of heavy bricks using a single legged push.

have been impossible. As mentioned earlier, these flaws were worked upon to improve the planner in Prototype 2. As can be seen in the simulation sequence in Fig. 4, the robot expands its front and back wheel pairs independently to negotiate the low lying obstacle while traversing a corridor angled at $\approx$60deg. Furthermore, Fig. 5 shows the robot performing obstacle negotiation in a very tight space thereby negating the flaw mentioned earlier while also navigating using omni-directional motions.

III. LOCAL PUSHING PATH PLANNER

The sequence in Fig. 6 depicts the capability of the robot to use one of its limbs to push a set of bricks. In this section, we describe our new work which makes use of this capability. In particular, we introduce a novel algorithm that plans a sequence of object pushes, to clear obstructing obstacles. The goal is to move an obstacle away from a pre-computed trajectory (provided by the global planner, see Sec. II) or push it to a location where it can be avoided or negotiated. The obstacle's mass and inertial properties might be unknown. The planner aims to perform a minimal amount of pushing, as well as travel the least distance to perform the pushes and rejoin the original path plan. The distance travelled by the wheels, the configuration changes, the overall translation of the centroid of the object, and the rotation of the object while performing the pushing sequence will be the necessary part of the weighted additive costs to be minimised.

This local push planner will run at a higher frequency than the above discussed global planner (Sec. II). The two planners in the future will be combined to form the full agile navigation system for the hybrid CENTAURO robot. The global motion planner gives initial plans to traverse cluttered spaces, ignoring pushable objects that can be viewed by the sensory system (e.g., Velodyne) during the planning phase. On the other side, the local planner will refine these plans to push obstacles out of the plans that are computed by the global planner. This is especially useful in a scenario where global plans cannot be calculated due to an object blocking the path to a goal and the object can neither be avoided nor negotiated. In cases that an object is pushable, the global planner might take that into account in order to make plans that can be further refined by the local planner. This will be done by pushing the object, such that the global path becomes clear for the robot to pass through. An example of such a scenario can be seen in Fig. 7, where the pushable object (grey) is inside a small passage. The blue, red, and green arrow lines represent the pre-planned front left wheel, robot center, and front right wheel trajectories, respectively. The lack of space on the side, as well as the width of the object being greater than the maximum expansion width of the robot, makes it impossible to plan a path inside the passage. The new proposed algorithm and solution (i.e., pushing the obstacle in order to clear the passage) can be seen in Sec. IV (image and Gazebo simulations in Figs. 16 and 18). In some scenarios, the solution provided by the new local push planner algorithm may even aid in reducing the total distance traversed by the robot as the robot could simply just push one obstacle and have its path shortened.

The computation of the local plans for our henceforth named *Local Push Planner*, in the presence of pushable obstacles, involves the following steps that will be explained in detail in the following subsections:

- Region partitioning of 2D environment into rectangular areas, where collision-free pushing of a movable obstacle can be performed (Sec. III-A).
- Parametrizing the movable obstacle to allow for repeated, easy, and efficient computations for determining push sequence. (Sec. III-B).
- Planning the push of obstacles from within a region to a suitable edge. (Sec. III-C)
- Planning the push in order to move the obstacle from one region to another. (Sec. III-D)
- Determination either of sufficient clearance of the obstacle from the path or the existence of sufficient space to negotiate/avoid it. (Sec. III-E).

A. REGION PARTITIONING

The first step of the *Local Push Planner*, obtains a 2D image map from the recognition algorithm. This map segments the image into free spaces (black pixels), immovable obstacles (white pixels), and movable obstacles (grey pixels), as can be seen in Fig. 7. We consider as free space the ground floor that the robot moves on, while movable and immovable objects can be visually determined by any object recognition algorithm, such as YOLO [42]. The recognition algorithm itself is not the main subject of this paper and thus we omit the details. Notice that we also do not study the problem of having visually movable objects that are immovable (e.g., a box that is too heavy to be pushed) and we leave this as future work.

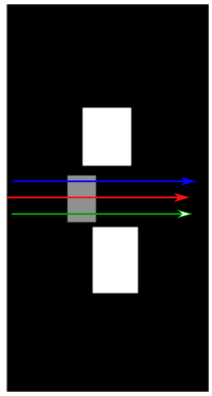

FIGURE 7. A sample segmented image map on the environment. The arrowed lines are added to show the pre-planned trajectories from the global planner. The blue line represents the front left wheel trajectory, the red represents the robot center trajectory, while the green line represents front right wheel trajectory.

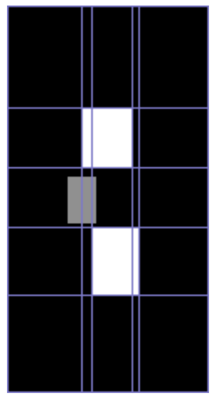

FIGURE 8. Rectangle partitioning of the environment based on bounding rectangles of the immovable obstacles.

We form bounding rectangles around the white patches to determine conservative coordinate limits of the immovable obstacles. Using these bounding rectangles we partition the environment into a set of rectangles as can be seen in Fig. 8. These individual rectangular partitions are then processed and stored in a structure, where each element contains the largest possible set of contiguous rectangles forming a larger one, but devoid of any immovable obstacles. The first element of the structure consists of the largest rectangular region containing the robot in its initial position, with the subsequent elements containing rectangular regions in a directional sequence similar to that of the expected pre-planned trajectory. Therefore, during the algorithm processing, moving from one element to the next will actually mean moving from one rectangular region to the next possible, while moving in the same direction as the pre-computed robot path trajectory.

An example of the partitioning and subsequent creation of the directional structure of rectangular free spaces can be seen in Fig. 9. As can be seen, the pre-planned trajectory

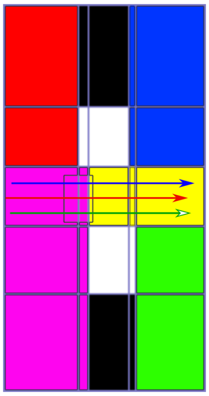

FIGURE 9. Example of computation of connected free-space rectangle regions. The various colours represent different regions or free space partitions.

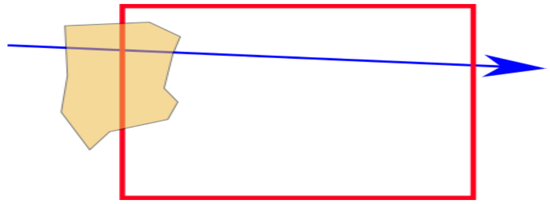

FIGURE 10. A scenario of a movable object (yellow patch) at one edge of a rectangular region (red rectangle). The wheel trajectory (blue arrowed-line) collides with the movable obstacle.

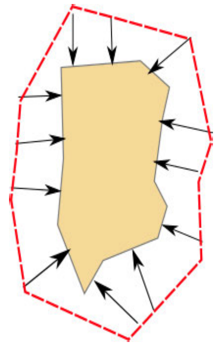

FIGURE 11. Determination of push contact points and push directions at respective contact points. The push directions represented by arrow lines are perpendicular to the surface containing point of contact. The red dotted line is the collision free path around the obstacle tracing the start points of the push direction vectors.

runs from the left to the right part of the image (blue for the front left wheel/foot, red for the robot center, and green for the front right wheel/foot), hence connected rectangular regions are also formed with similar directions. The algorithm will attempt to find the appropriate rectangular free-space partition to allow the placement of the object such that the robot can try to go around it or otherwise negotiate it. In the case of Fig. 9, the movable obstacle is present in the pink and the yellow rectangles. Thus, the processing starts from the pink towards the yellow region. This is followed by a combination of the blue rectangle and part of the yellow shaded regions aligned with the blue and green shaded rectangles. This ensures that modified plans are as close as possible to the original one, while also ensuring that unnecessary searches in unneeded directions are not performed.

In each of these rectangular regions or connected partitions, we first test if it is sufficiently large enough to accommodate the object. Then we check if there are any points in the queried rectangular partition where the movable obstacle can be placed such that the robot can navigate unhindered to the original trajectory. This can be done either by the robot footprint having enough space in the new location of the object to avoid it or the object being low-lying and of width less than the maximum width of the robot, thereby allowing it to be negotiated. If no such point is found in the queried partition then, the obstacle is pushed to the next partition as computed by the directional structure containing the sequence of rectangular partitions. This is detailed in the following subsection.

B. OBSTACLE PARAMETRIZATION AND PUSH PLANNING IN FREE SPACE PARTITION

While testing the ability of an object in a rectangular free-space partition to be cleared by the wheel trajectory, we test for clearance from three trajectory lines. These are the front left wheel, front right wheel, and the robot center trajectory lines. The clearance testing for a single trajectory line is detailed as follows. Consider Fig. 10, where an object (in yellow) needs to be pushed clear of the wheel trajectory (blue line). The object hence needs to be pushed in a manner that its centroid is moved to some point inside the red-rectangular free space region. In this case, the goal is to ensure none of the object pixels collides with the wheel trajectory line.

We begin, by first determining the push contact surfaces, using the edge pixels coordinates of the object. Since the resolution of the image is 5cm per pixel and the length of the face of the CENTAURO foot is 25cm (justification of the selected values in the global planner work [6]), we search all the edge pixels of the obstacle in the image in sets of 5 in order to find smooth surfaces. The angle between vectors from the first to the third pixel and from the fifth to the third pixel is used to determine if the surface is smooth or not. This eliminates sharp corners for the safety of the robot. We then determine the push direction to be always perpendicular to the push surface. This is done to ensure maximum contact with the surface for the duration of the push. As can be seen in Fig. 11, we furthermore find a collision-free path connecting all the start points of push actions. This collision-free path around the obstacle is used to determine if the robot has sufficient space to move around the obstacle and rejoin the original pre-planned trajectory after pushing it to a suitable location. If this is done, the robot can continue executing the plan.

From this step, we use the complete set of push surfaces and push direction vectors to completely represent the object.

The surface contact points are stored as vectors from the centroid and the push directions as unit direction vectors. Let us represent the n position vectors of the contact points with respect to the centroid as $\overrightarrow{c_{pi}}$ and the push directions as $\overrightarrow{d_{pi}}$, where $i = \{1, \ldots, n\}$. Let the centroid coordinates be represented as $\overrightarrow{O} = \{O_x, O_y\}$ and the start points for the push be written as position vectors with respect to the centroid as $\overrightarrow{s_{pi}}$. We aim at finding an $\overrightarrow{O}$, such that it satisfies either of the following conditions:

- All coordinates calculated as $\overrightarrow{O} + \overrightarrow{s_{pi}}$ are completely on one side of the wheel trajectory line.
- All the coordinates $\overrightarrow{O} + \overrightarrow{s_{pi}}$ are such that the robot in its smallest square footprint can easily traverse the curve traced by the dotted red line, as seen in the sample Fig. 11, and still be sufficiently away from the rectangular regions consisting of obstacle pixels.
- The object in the particular configuration and location can be easily negotiated by simply changing the robot footprint.

There are multiple possible push execution directions, while the hybrid CENTAURO robot has the capability to push an object using various strategies due to its high DoF and agility. Hence, to limit the search space of possible pushing strategies, as well as efficiently determine the sequence of pushes to be performed, we compute a push sequence, such that its principle components are the push actions. Each push action can be either of three types of pushes. The first type of push is the single-legged in place, where the robot merely moves one of the front limbs to move the object. The second push type is a single-legged drive through, where only one limb is in contact with the obstacle and the whole robot moves to push it. The last push type is the two-wheeled drive through, where the robot moves while both the front limbs are in contact with the object being pushed. We accumulate a set of push actions to create a push sequence. To determine the set of pushing actions, we first determine the reachable set of contacts point based on the angle of push and current robot orientation and position. If the angle of the push vector with respect to the direction of the eventual motion of the robot is within 90 degrees, then it is deemed a useful push. This, in turn, means that we discard any pushes that move the object back, towards the robot center.

Once we have a set of reachable and usable robot push contact points and push direction vectors, we create a final set of all possible valid push actions as follows. First, all valid push vectors are stored as single legged pushes that cause a translation along the line joining the contact point and object centroid, represented as a unit vector $\overrightarrow{t_{pi}}$. The corresponding component of $\overrightarrow{d_{pi}}$ perpendicular to the line joining the contact point and centroid is calculated and represented by $\overrightarrow{r_{pi}}$. The rotation caused by a single push $\overrightarrow{d_{pi}}$ for translating the object by two pixels or 10cm is calculated as follows:

$$\theta_{pi} = 0.1 \times \frac{|\overrightarrow{r_{pi}}|}{|\overrightarrow{t_{pi}}| \times |\overrightarrow{l_{pi}}|} \tag{2}$$

The 0.1 value stands for the overall tangential velocity magnitude for a motion of two pixels, which is set to 10cm/s for our robot, and $\overrightarrow{l_{pi}} = (l_{ix}, l_{iy})$ represents the vector from the push contact point to the centroid. Effectively, we calculate a simple kinematic angular velocity and assume the motion is performed in 1 s, leading to a rotation of θ_{pi}.

For two-wheeled pushes, we just select two push directions that are within 15 degrees of each other and the two contact points have a distance between the minimum and maximum widths of the possible robot footprints which in our case is either $9 - 22$ pixels or $45 - 110$cm. The above-mentioned angle restriction allows for the robot to make and maintain simultaneous double contact while pushing, thereby reducing the uncertainty of the pushes. The single-wheel push calculations are used to find the effective translation and rotation. As can be inferred, two-wheeled pushes with opposite rotations will lead to a smaller amount of object rotation and more stable pushes. Hence, this will be preferred for long range movements of the object.

Once the set of single- and double-wheeled pushes are assembled, costs are assigned to them and stored in another structure to be used for determining the best sequence of pushes. As can be observed from the very simple kinematic calculations, the algorithm does not take the mass of the object being pushed into account nor the inertial distribution. While the pushing is being carried out, the robot limbs are in position control mode, hence the force exerted on the object being pushed keeps increasing until the specified push motion is executed to completion. Without advanced learning and recognition algorithms, we cannot entirely determine the mass of the object from simple visual sensing and mapping algorithms with inputs from sensors like the Velodyne or the Realsense D435 look-down camera mounted on the hip of the robot. Hence we use simple kinematic calculations and position control to push the unknown object. This is also the primary reason that necessitates a very conservative rectangle-based partitioning, in order to determine the space in which the object can be moved. In case the push execution causes the object to not move as expected, unnecessary collision with the other environment objects can be avoided. Furthermore, the push sequence computation algorithm will attempt to use pushes with lesser rotations, as this will reduce the uncertainty in the motion of the object being pushed.

In the following subsection, we will explain the computations that are involved in order to determine required pushes that take an object to a free rectangular-partitioned space.

C. DETERMINATION OF FEASIBLE GOAL POINTS

Considering again Fig. 7, the goal is to push the object completely below the blue wheel trajectory line. First, we determine the coordinates of the intersection between the region rectangle and the wheeled trajectory. Then, we fit a line, with the equation being $ax + by + c = 0$, between the two points. The space below the line is represented as: $ax + by + c > 0$. Let the rectangular region be represented by minimum

Algorithm 1 Push Sequence Creation for Rectangular Partition R_i

Require: Robot left, center, and right trajectory line inequalities T_l, T_c, & T_r
Require: Parametrized obstacles, initial obstacle centroid coordinates O_x, O_y, contact point vectors, push directions (t_{pi}) and push rotations.
Require: Pre-determined cost weights C_1, C_2, C_3, C_4
  for all T_i **do**
    Find all q_{ix}, q_{iy}, satisfying (3)
    Sort all q_{ix}, q_{iy} based on distance from current O_X, O_y
    for All q_{ix}, q_{iy} **do**
      Greedy search of push actions based on t_{pi}s
      Check collisions & robot escape from obstruction
      $D_r \leftarrow$ Total distance travelled by robot center
      $D_w \leftarrow$ Distance travelled by wheels
      $D_o \leftarrow$ Distance travelled by object centroid
      $D_t \leftarrow$ Rotation of the object
      $C_{ixy} \leftarrow$ Cost to move to q_{ix}, q_{iy}
      $C_{ixy} = C_1 \times D_r + C_2 \times D_w + C_3 \times D_o + C_4 \times D_t$
    end for
  end for
  if solution coordinates q_{ix}, q_{iy} found **then**
    Choose push sequence and q_{ix}, q_{iy} with min C_{ixy}
  else
    Find min. rotation push to next partition
  end if

and maximum coordinates (x_{min}, y_{min}), (x_{max}, y_{max}). Then, we find all queried coordinates $\vec{q} = (q_x, q_y)$ inside the rectangle, such that they satisfy the following conditions:

$$\begin{aligned} x_{min} &\leq q_x \leq x_{max} \\ y_{min} &\leq q_y \leq y_{max} \\ a(q_x + s_{ix}) &+ b(q_y + s_{iy}) > 0 \\ (q_x + s_{ix}, q_x &+ s_{iy}) \notin X_{Obs} \end{aligned} \tag{3}$$

If we are able to find points satisfying all the conditions in (3), then we have a goal coordinate for the object's centroid. Thus, the object can be moved to a point in the free space region, where there is no more obtrusion in the wheel trajectories. Furthermore, the last condition ensures that none of the surface contact points are in obstacle regions represented by X_{obs}. This is a check that can be done very quickly using just the bounding rectangle coordinates.

Starting from the first partition occupied by the object, the aforementioned process is applied to find out all the possible points that the centroid can be moved to in that particular partition, so as to clear the trajectory line in query. Three trajectory lines are considered namely the front left wheel trajectory, the front right wheel trajectory, and the robot center trajectory. If any of these trajectory lines intersect with the object, then the above-described process is repeated as many times as the trajectory intersections. We also check if the object can be moved to either side of the trajectory lines within the partition being examined.

Hence, for every one of the three trajectory lines intersecting with the movable obstacle, we get a set of possible coordinates where the obstacle can be moved to. This clears the lines and leaves the obstacle within a free space rectangular partition. Furthermore, the aforementioned steps are repeated for different orientations of the obstacle. In case no solution is found in the current orientation, the object is rotated by 5 degrees and similar steps are carried out. We limit the rotation of the object to a maximum of 30 degrees, as it means that the robot will have to completely turn itself in order to rotate the object. Thereby, the robot will diverge from the original plan by too much. In the case of mathematical operations, as we represent the object by a simple set of contact points with vectors to the centroid, a simple rotation matrix multiplication with the vectors is all that is needed to re-run the tests mentioned in (3).

In the case that all the conditions except the line inequality are satisfied, then it means that the object needs to be moved through the rectangular region to another neighbouring region. In this case, the object is moved in the direction of the pre-planned trajectory to the next connected free space rectangular partition. This takes place using a sequence of motions that involves the least amount of rotations and least distance travelled to reach the edge of the current rectangular partition it is situated in.

The determination of the sequence of pushes to reach either the goal coordinates in the partition or the edge of the partition shall be detailed in the next subsection.

D. DETERMINATION OF PUSH SEQUENCE

The first point to be noted is that we are merely trying to push the object in a rectangular region known to be free. Hence, considerably greedy methods can be used to find the sequence of actions from a given set of push actions.

We run an iterative local region loop to determine the push sequence. We have two scenarios. The first is that inside the given local region, it is impossible to clear the wheel trajectories. In this case, the goal would be to move the obstacle from one region to the next possible region, with minimal possible rotation.

As mentioned earlier, we either have a set of coordinates that the object can be moved to, such that the queried wheel trajectory line is cleared or it can be pushed to the next free space rectangular partition. We sort the coordinates in ascending order of distance from the current centroid coordinate. All the possible pushes are normalised, so that the application of one such push, results in the same amount of translation.

Loops are run through each of the possible goal coordinates to determine the sequence of pushes that leads to the centroid of the object at one of the goal coordinates. The determination is performed as follows:

- First, a vector from the current centroid coordinate to the goal coordinate is calculated.

- The push that has a translation vector closest to the direction of the vector computed in the previous step is chosen.
- The set of pushes with the least number of deviations, rotations, and number of steps are chosen and stored, using a simple greedy loop.
- In the case that the target is not reached in a nominal number of steps, it is checked if the predicted coordinate is in the set of goal coordinates. If so, then the push is accepted, otherwise, the push sequence is computed for the remaining goal coordinates.
- At each computation step of the sequence, every push action is checked in order to determine if the robot can move using the collision-free connected loop around the object to the right push start points without colliding or exiting the free space rectangular partition (the connected loop was computed while parametrizing the object). The free space partition boundary checks is done using simple minimum and maximum coordinate checks.

The above sets of steps are repeated for each trajectory in line, passing through the queried rectangular free space partition. All possible sequences of solutions are stored, costs are assigned based on the distance of the push, the rotation caused by the push and the distance moved by the robot to cause the push. This ensures that pushes that involve lesser rotations and changes in the push action sequence are chosen. A brief description of the flow of computations of the new local push planner for on single rectangular partition can be seen in Algorithm 1.

E. EXECUTION OF THE PUSH SEQUENCE BY THE ROBOT

As mentioned earlier, the robot has multiple ways of executing the push: 1) it can stand in place and perform a single legged push, 2) it can drive through and perform a single legged push, and finally 3) it can perform a drive through double legged push. To further simplify the process of choosing which method would be best, the following decisions are taken:

- If the single leg push is chosen and it is on the right side of the object with respect to the robot and the push results in anti-clockwise rotation, we always push with the front left so that the robot doesn't rotate into the polygon of the robot.
- Same is done for the opposite rotation and relative position of the push with respect to the robot.
- Drive-through pushes with minimal rotations are preferred if this allows for the robot to not have the need to change its configuration.

The above decisions are included in the cost as the distance travelled by the robot to reach the push contact point and the distance travelled by the wheels to execute the configuration changes needed to perform the push.

F. TERMINATING CONDITIONS FOR THE PLANNER

As mentioned earlier, we check if we can push the object completely to one side of the trajectory line or push it sufficiently far from all obstacle regions. The second of the above conditions can be easily verified by checking if all the push start points are sufficiently distanced from the obstacle rectangle lines. The sufficient distance is ensured, simply by checking if the robot is in its minimal square configuration (the footprint is a 50cm^2 square). If the robot was at the queried push start points, does not cross over into the obstacle regions, it would mean that the robot can freely move around the object and hence rejoin its original trajectory.

Another terminal condition is to see if the pushed object at any part of the plan is in such an orientation that it can be negotiated by changing the footprint configuration. Once again this is a simple calculation of orienting the points in the direction of the robot and merely finding the horizontal extremities of the obstacle pixels to determine if the robot can expand over it. This is a process similar to the computations used in the global planner of Prototype 1, as explained in Sec. II-B, but on a smaller scale to perform quick operations.

When either of these terminal conditions is met we stop any push sequence calculations and pass on the new trajectory to the trajectory execution interface.

IV. EXPERIMENTAL AND SIMULATION RESULTS

In this section, we first briefly discuss the experimental results obtained on the real CENTAURO robot using the plans from Prototype 1 and Prototype 2 of the global planner algorithms, described in Sec. II. This is followed by the simulation results of the new local planner presented in this article, as described in Sec. III.

A. EXPERIMENTAL RESULTS OF PROTOTYPE 1 AND 2

Fig. 12 shows a simple execution of the global plan, where the robot needs to narrow itself into a passage, created by the tall concrete bricks, to reach its goal. It is to be noted, that the footprint width signifies the distance between centers of the wheels. The robot initially was in an expanded configuration, with the width of the rectangular footprint being 100 cm, whereas the safe width of the narrow passage was only 75 cm. The computation time of the plan being executed in an Intel Corei7-6700 PC with 24GB RAM was approximately 0.11 s.

Fig. 13 shows a longer and slightly more complicated plan execution, where the robot has to firstly narrow into a passage and then expand over the low-lying obstacle on which the target goal coordinates were set. The width of the passage is 80 cm. The robot starts in the expanded configuration with a footprint width of 100 cm, then narrows to 60 cm in the narrow passage, and finally expands to a safe width of 76 cm over the low-lying obstacle of width 50 cm. The computation time of this plan was roughly 0.23 s.

It should be noted from Fig. 13 that the distance between the narrow passage and the low-lying obstacle in the front, is longer than the length of the robot in the narrow configuration. Had that not been the case, no plan would have been found. This flaw, as explained earlier, is solved in Prototype 2 as can be seen in Fig. 14, by having the robot operate its front and back wheel pairs independently. The distance

FIGURE 12. Sequence depicting the CENTAURO robot narrowing into the space in between the tall bricks to reach outside the narrow passage.

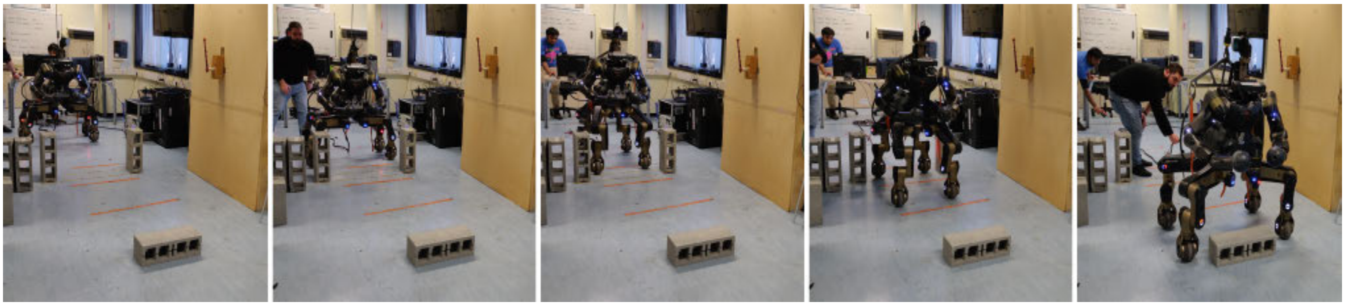

FIGURE 13. Sequence depicting the robot first narrowing in the corridor and then expanding over the low-lying wide object over which the target goal coordinate was specified.

between the narrow passage and the low-lying obstacle is much shorter than the case of Prototype 1, but as the front and back wheel pair operations are independent, the robot has sufficient agility to navigate the environment. Once again, the width of the passage is 75 cm and the width of the low-lying object is 50 cm. The distance from the end of the narrow passage to the object is 80 cm, which is 30 cm less than the length of the robot in the narrowest configuration, i.e. when both wheel pairs are inside the passage. The robot first omni-directionally aligns itself with the passage. Then, the front wheel pair narrows from a width of 70 cm to 50 cm, followed by the same operation being done by the back wheel pair. Finally, while exiting the passage the front wheel pair alone expands to a width of 80 cm. The time to compute these plans once again executed on an Intel Corei7-6700 PC with 24GB RAM was 0.0634 s on average.

B. SIMULATIONS OF THE LOCAL PUSH PLANNER

We first present simple image-based simulations, depicting just the push sequences planned for two scenarios where the path of the robot is blocked. The scenarios and the solutions, as seen in Figs. 15 and 16, depict the utility of the Local Push Planner in the specific scenario of an object obstructing the entry into a corridor/passage.

In Fig. 15, the obstructing object is before the planned entry into the passage, between the two white obstacles. The robot just uses a single-drive through-push to completely move the obstacle out of the path, allowing the robot enough space to move into the corridor without the need to change the configuration. The time taken to compute the push sequence was 29 ms on an Intel i7-6500U core, 16GB RAM laptop.

The sequence in Fig. 16, depicts the solution to the scenario in the previously presented Fig. 7, where the obstructing object is already inside the passage, through which the robot has planned to pass through. In this scenario, the planner chooses to move the object with the double legged drive-through push till it reaches a point where it detects it is in a sufficiently large free space partition and the object can be moved outside the path. Here the robot shrinks and performs a single legged push to translate and rotate the object away from the pre-planned front left legged trajectory. Thereby, it allows the robot enough space to escape the obstruction after performing the minimal required push. 33 ms was needed to compute the push sequence for this scenario.

Now we present the results of the simulated CENTAURO robot executing local push plans, through means of Gazebo simulations. The first simulated sequence can be seen in Fig. 17. In this situation, a long obstacle is obstructing the entrance to the corridor. The global planner ignores the obstructing object and plans the path through the corridor. The local planner then sees that the obstructing object can be moved to a place inside the same rectangular free-space

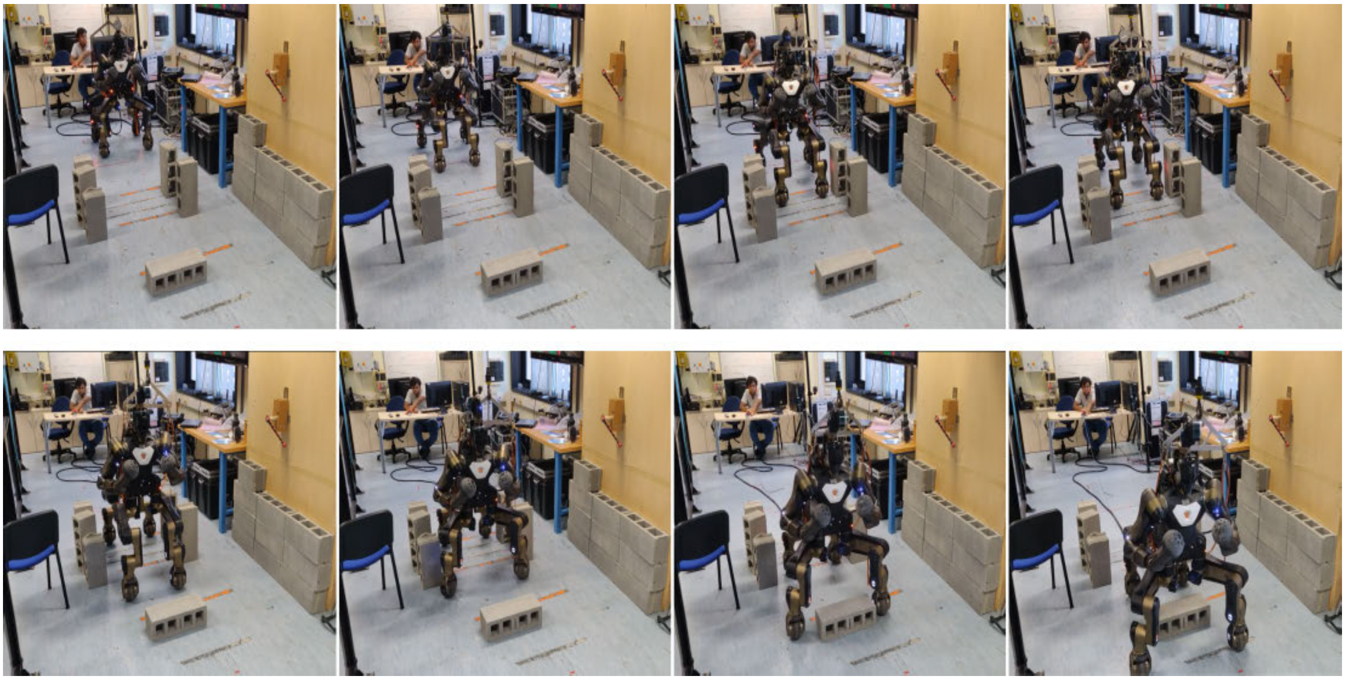

FIGURE 14. Sequence depicting the robot executing plans from Prototype 2. The robot omni-directionally aligns with the narrow passage, first narrows the front wheel pair to enter the passage, then the back wheel pair to pass through the passage. Finally it expands the front wheel pair to negotiate to go-over the low lying obstacle.

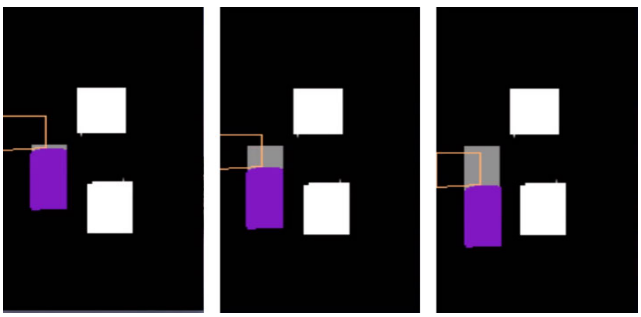

FIGURE 15. Image simulation sequence depicting push motions. The grey/purple obstacle blocks the passage between the immovable white obstacles. The purple pixels show the position of the pushed object over the execution of the planned single wheeled push sequence. The orange rectangle depicts the robot.

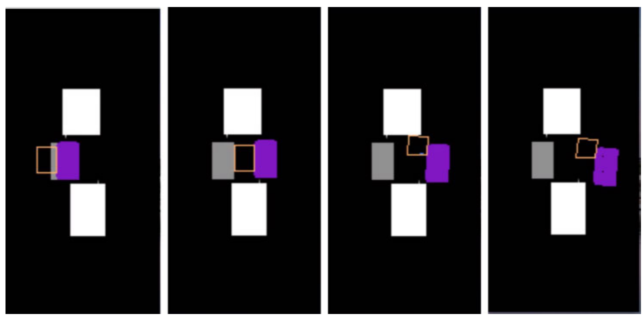

FIGURE 16. Image simulation sequence depicting a double legged drive through push sequence execution followed by a single legged rotating push to clear the robot of the initial planned path so as to allow the robot enough space to rejoin the planned again by going around the object.

partition, like that of the initial coordinates of the object using a simple single-leg push. In this case, the single-leg push could be executed using simple drive through motion and hence the object is cleared just enough to allow the robot to safely enter into the passage. It can be seen that the planner chose to keep pushing the obstacle until it is clear of the pre-planned front left wheel trajectory of the robot. If instead, it had chosen to only clear the robot center trajectory, the robot would have had to change its footprint configuration to the narrowest polygon to enter the passage. In this case, the weighted cost of the overall distance travelled by the wheels for the configuration change, outweighed the small extra distance of pushing, as the robot would have had to move in a configuration close to the minimal square configuration (square footprint of side 50cm) to safely enter the passage without colliding with the wall. Furthermore, the cost of the push would have been low as it involved minimal rotation.

Fig. 18 depicts the Gazebo simulated CENTAURO robot, performing the push sequence planned for the situation in Fig. 7. It should be noted that even the drive-through push is not entirely accurate, but the conservative nature of the plan ensures the lack of collisions with the other obstacles. Furthermore, since the object is wider than 1.1m, the robot cannot negotiate it unless the object is rotated by more than 50 degrees. It is worthwhile to note that the robot could have continued to push the object in drive-through mode for longer, and then moved around its edges. But in this case, the robot, by rotating the obstacle after pushing it through the passage, performs pushing for a smaller distance in terms of translations. Furthermore, since it changes configuration to

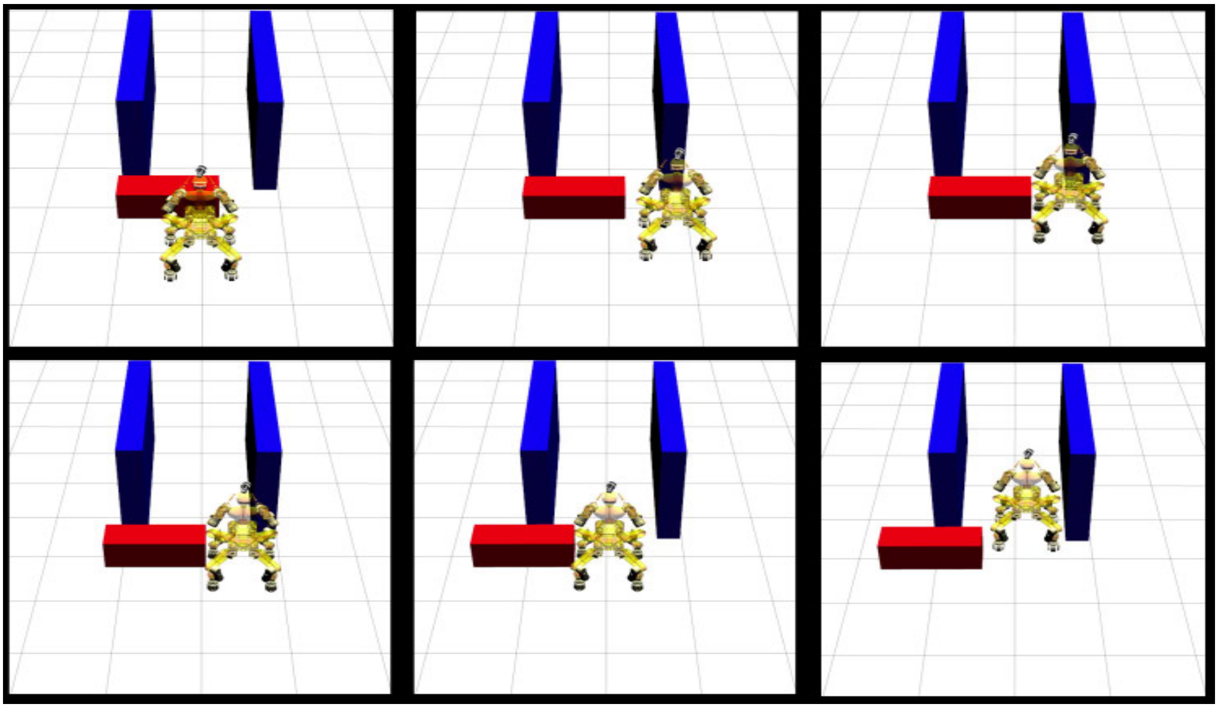

FIGURE 17. Simulation sequence depicting the robot clearing the obstructing obstacle using a single legged drive through push and then entering the corridor.

perform the push as well as move to one side of the object (thereby already in the process of going around it), it travels slightly less distance to rejoin the original plan, while also minimising the distance the object is pushed.

The final simulation, as can be seen in Fig. 19, shows a sequence of pushes where the robot executes a non-drive through single legged push. In this situation, a two-wheeled push would have resulted in more distance pushed for the object. Moreover, the object would have been rotated more than in the case where a single horizontal push would be performed from the side. Since the object is already onto one side of the exit of the passage and not in the center, simply pushing it more to the side which it was on the exit of the passage, meant less pushing distance, and less probable rotation. Hence, the robot expands fully and moves to the opposite side, so that the front right leg could move from its widest position and then push towards the inward direction, in turn pushing the object to the side. As can be seen, the robot also moves horizontally after the front right leg reaches its inner width limit. Then the robot returns, reconfigures to the smallest and narrowest configuration, and then goes around the object. As can be seen, the push is not exact, as the object is rotated to a little extent, but that is fine due to the conservative nature of our planner.

Through the experimental results of the Prototype 1 and 2 algorithms, we have managed to increase the number of cluttered environment scenarios the robot can solve and traverse when compared to simple wheeled navigation. This is done while still allowing us to take maximum advantage of the fast wheeled-motion mode of the CENTAURO. Furthermore, through our image-based and Gazebo simulations, it can be seen that the robot can easily clear obstacles from the path. This is especially true in the particular scenarios of blocked entry to or exit from passages. The presented algorithm though is not without its downsides. All our simulations consisted of moving only a single obstacle, hence it is only expected that the computation time will increase with the increase in the obstacles to be pushed. Furthermore, complex scenarios where the robot might need to perform very accurate and precise pushes to avoid damaging the environment may be beyond the capabilities of the current algorithm. While the conservative and simplistic geometry approach may limit us in more complex scenarios, the fact that we are able to plan these pushes in very short times allows for potential repeated planning and multi-object pushing in the future versions of the algorithm. While multiple movable obstacles in close proximity might hinder the search for solutions, a simple algorithm that plans for the separation of objects from one another might be beneficial. In such a case, instead of pushing the obstacle to clear a trajectory line, it would be needed to push one obstacle from another. A similar framework that was presented in this work can be used with appropriate modifications to clear one obstacle from a region than a trajectory line. This can be followed by subsequent planning, using the framework presented in this work, to clear the obstacle from wheel trajectory lines,

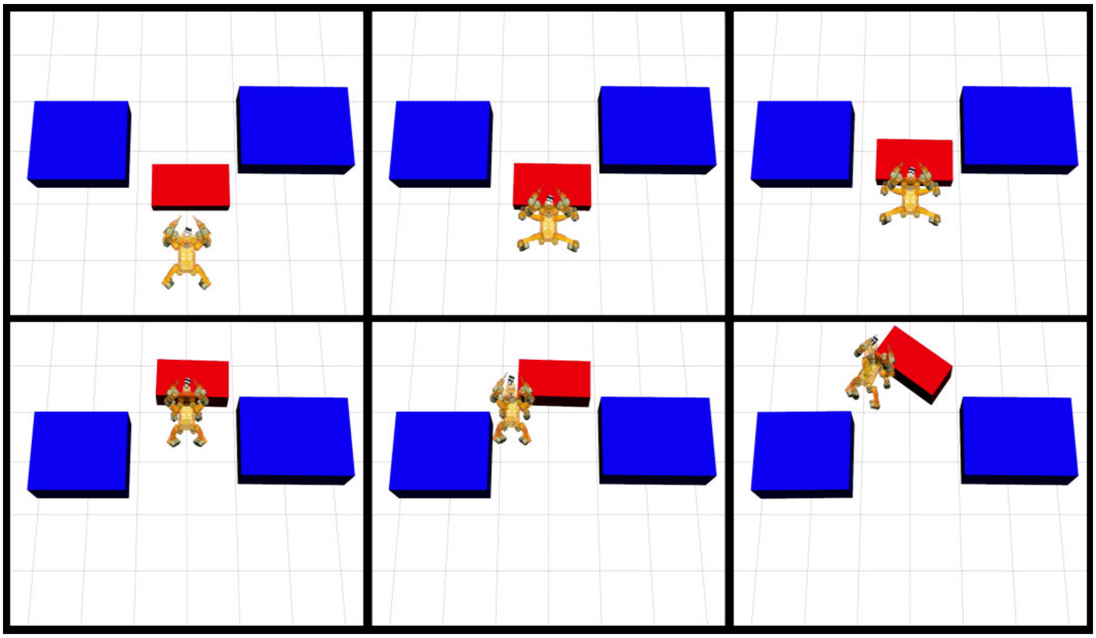

FIGURE 18. Gazebo simulation sequence of the scenarios in Fig. 7. The robot performs a two-legged drive through push to move the obstacle in the small passage before using a single wheeled push sequence to rotate the object, to give sufficient space to go around the obstacle and rejoin the original plan.

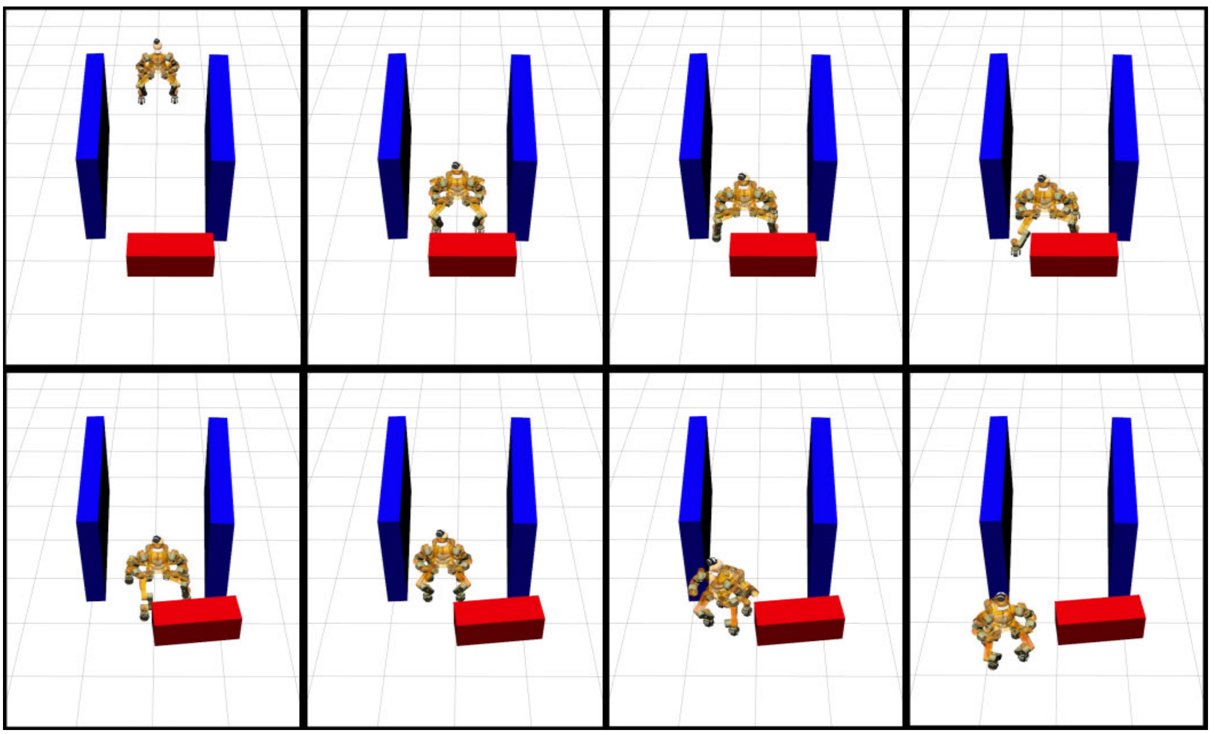

FIGURE 19. Simulation sequence depicting the robot moving through a corridor, clearing the obstructing object and moving forward. The robot first reaches the obstructing movable obstacle. It then expands and moves to the side to allow the front right wheel to use its full range of horizontal motion inwards to perform the push and then push the object from the side.

as there is sufficient separation between the objects. The agility of the robot and the usage of single legged operations would be sufficient to perform such subtle pushes. Thereby this improves the ability of the robot to traverse heavily cluttered spaces.

V. CONCLUSION AND FUTURE WORK

In this article, the development of a reconfigurable and agile navigation planner suite for hybrid legged-wheeled robots was outlined. First, the previously developed prototypes for the global planner were described, comparisons between the prototypes were made, and the results were reviewed and presented. The plan from the global planner Prototype 2 allowed for very agile navigation in tight cluttered spaces giving the robot the ability to solve and traverse more environments than simple non-reconfigurable wheeled navigation planning, by using the full capabilities of reconfiguration. At the same time, the planning search space was designed to be low-dimensional, allowing for high-speed computations. This, in the future, would allow for repeated quick long-distance global planning for the robot.

Following the previously developed global planner, a new local planner, to clear movable objects obstructing the plans from the global planner, was introduced. The planner used line equations, rectangular partitions, object parametrization, and simple collision checks to determine a sequence of push actions that lead to minimum rotation, minimum pushed distance, and minimal configuration changes of the robot in order to achieve the push and, thus, escape the obstruction. Image-based and Gazebo simulations, with the CENTAURO robot, were performed to demonstrate the capability of the new local planner. This new prototype runs in parallel with the global planner in the final version of the navigation.

Through the new local planner, the robot is now capable of traversing more complex environments by performing a small sequence of pushes to clear the path and reach target goals. This will be extremely useful when the robot is commissioned to perform tasks in cluttered spaces. Furthermore, there exists one more method of locomotion that the CENTAURO can use to navigate cluttered spaces, which is stepping over obstacles of appropriate height. The final version of the suite will contain a step planner, which in the presence of low-height obstacles, will determine safe footsteps and determine if stepping is worth the safety cost to perform, rather than avoiding or negotiating the obstacle. Adding a computationally moderate stepping planner will give the robot another option to navigate cluttered paths. Together these three computationally moderate components, namely, the global obstacle negotiating planner, local push planner, and the immediate step planner could allow for repeated, reliable and quick planning in cluttered spaces.

REFERENCES

[1] N. Kashiri, L. Baccelliere, L. Muratore, A. Laurenzi, Z. Ren, and E. M. Hoffman, "CENTAURO: A hybrid locomotion and high power resilient manipulation platform," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1595–1602, Apr. 2019.

[2] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2006.

[3] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 300–307.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[5] V. S. Raghavan, D. Kanoulas, A. Laurenzi, D. G. Caldwell, and N. G. Tsagarakis, "Variable configuration planner for legged-rolling obstacle negotiation locomotion: Application on the CENTAURO robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Nov. 2019, pp. 4738–4745.

[6] V. S. Raghavan, D. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Agile legged-wheeled reconfigurable navigation planner applied on the CENTAURO robot," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 1424–1430.

[7] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Theta: Any-angle path planning on grids," in *Proc. AAAI*, vol. 7, 2007, pp. 1177–1183.

[8] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[9] E. A. Hansen and R. Zhou, "Anytime heuristic search," *J. Artif. Intell. Res.*, vol. 28, pp. 267–297, Mar. 2007.

[10] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, Oct. 1994, pp. 3310–3317.

[11] S. Koenig and M. Likhachev, "$\hat{D^*}$lite," in *Proc. IAAI*, vol. 15, 2002, pp. 476–483.

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

[13] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Dept. Comput. Sci., Iowa State Univ., Ames, IA, USA, Tech. Rep., 1998.

[14] S. M. LaValle and J. J. Kuffner, Jr., "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2000, vol. 2, pp. 995–1001.

[15] A. Atramentov and S. M. LaValle, "Efficient nearest neighbor searching for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2002, pp. 632–637.

[16] M. Brunner, B. Bruggemann, and D. Schulz, "Motion planning for actively reconfigurable mobile robots in search and rescue scenarios," in *Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot. (SSRR)*, Nov. 2012, pp. 1–6.

[17] M. Menna, M. Gianni, F. Ferri, and F. Pirri, "Real-time autonomous 3D navigation for tracked vehicles in rescue environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Sep. 2014, pp. 696–702.

[18] M. Norouzi, J. V. Miro, and G. Dissanayake, "Planning high-visibility stable paths for reconfigurable robots on uneven terrain," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Oct. 2012, pp. 2844–2849.

[19] A. Le, V. Prabakaran, V. Sivanantham, and R. Mohan, "Modified A-star algorithm for efficient coverage path planning in tetris inspired self-reconfigurable robot with integrated laser sensor," *Sensors*, vol. 18, no. 8, p. 2585, 2018.

[20] L. Pfotzer, S. Klemm, A. Roennau, J. M. Zöllner, and R. Dillmann, "Autonomous navigation for reconfigurable snake-like robots in challenging, unknown environments," *Robot. Auto. Syst.*, vol. 89, pp. 123–135, Mar. 2017.

[21] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *Int. J. Robot. Res.*, vol. 29, no. 7, pp. 897–915, Jun. 2010.

[22] D. Kanoulas, A. Stumpf, V. S. Raghavan, C. Zhou, A. Toumpa, O. Von Stryk, D. G. Caldwell, and N. G. Tsagarakis, "Footstep planning in rough terrain for bipedal robots using curved contact patches," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 1–9.

[23] D. Kanoulas, C. Zhou, A. Nguyen, G. Kanoulas, D. G. Caldwell, and N. G. Tsagarakis, "Vision-based foothold contact reasoning using curved surface patches," in *Proc. IEEE-RAS 17th Int. Conf. Hum. Robot.*, Nov. 2017, pp. 121–128.

[24] T. Klamt and S. Behnke, "Planning hybrid driving-stepping locomotion on multiple levels of abstraction," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 1695–1702.

[25] T. Klamt, D. Rodriguez, M. Schwarz, C. Lenz, D. Pavlichenko, D. Droeschel, and S. Behnke, "Supervised autonomous locomotion and manipulation for disaster response with a centaur-like robot," 2018, *arXiv:1809.06802*.

[26] R. Buchanan, T. Bandyopadhyay, M. Bjelonic, L. Wellhausen, M. Hutter, and N. Kottege, "Walking posture adaptation for legged robot navigation in confined spaces," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2148–2155, Apr. 2019.

[27] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 489–494.

[28] R. Buchanan, L. Wellhausen, M. Bjelonic, T. Bandyopadhyay, N. Kottege, and M. Hutter, "Perceptive whole-body planning for multilegged robots in confined spaces," *J. Field Robot.*, vol. 38, no. 1, pp. 68–84, 2021.

[29] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auto. Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[30] V. Suryamurthy, V. S. Raghavan, A. Laurenzi, N. G. Tsagarakis, and D. Kanoulas, "Terrain segmentation and roughness estimation using RGB data: Path planning application on the CENTAURO robot," in *Proc. IEEE-RAS 19th Int. Conf. Hum. Robot.*, Oct. 2019, pp. 1–8.
[31] M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Proc. 4th IEEE/RAS Int. Conf. Hum. Robot.*, vol. 1, Nov. 2004, pp. 322–341.
[32] M. Stilman, K. Nishiwaki, S. Kagami, and J. J. Kuffner, "Planning and executing navigation among movable obstacles," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, May 2006, pp. 820–826.
[33] M. Stilman, "Navigation among movable obstacles," Ph.D. dissertation, Robot. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2007.
[34] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *Int. J. Robot. Res.*, vol. 27, nos. 11–12, pp. 1295–1307, Nov. 2008.
[35] H.-N. Wu, M. Levihn, and M. Stilman, "Navigation among movable obstacles in unknown environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Oct. 2010, pp. 1433–1438.
[36] M. Levihn, M. Stilman, and H. Christensen, "Locally optimal navigation among movable obstacles in unknown environments," in *Proc. IEEE-RAS Int. Conf. Hum. Robot.*, Nov. 2014, pp. 86–91.
[37] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *IEEE Int. Conf. Robot. Autom.*, Oct. 2007, pp. 3327–3332.
[38] B. Renault, J. Saraydaryan, and O. Simonin, "Towards S-NAMO: Socially-aware navigation among movable obstacles," *CoRR*, vol. abs/1909.10809, pp. 1–13, Sep. 2019.
[39] B. Renault, J. Saraydaryan, and O. Simonin, "Modeling a social placement cost to extend navigation among movable obstacles (NAMO) algorithms," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Oct. 2020, pp. 11345–11351.
[40] M. Wang, R. Luo, A. O. Onol, and T. Padir, "Affordance-based mobile robot navigation among movable obstacles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Oct. 2020, pp. 2734–2740.
[41] K. Ellis, H. Zhang, D. Stoyanov, and D. Kanoulas, "Navigation among movable obstacles with object localization using photorealistic simulation," Dept. Comput. Sci., Univ. College London, London, U.K., Tech. Rep., 2021. [Online]. Available: https://sites.google.com/view/isaac-namo/home
[42] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

VIGNESH SUSHRUTHA RAGHAVAN received the Bachelor of Technology degree in instrumentation and control from the National Institute of Technology, Tiruchirappalli, India, in 2014, and the master's degree in systems and control from Technische Universiteit Delft, The Netherlands, in 2016. He is currently pursuing the Ph.D. degree with the Humanoids and Human-Centered Robotics (HHCM) Laboratory, Istituto Italiano di Tecnologia, Genova, and the Department of Information Engineering, University of Pisa. His main research interests include reconfigurable autonomous navigation and sensor data fusion for mapping and localization.

DIMITRIOS KANOULAS (Member, IEEE) received the Ph.D. degree from Northeastern University, Boston. He is currently a Lecturer (an Assistant Professor) in robotics and computation at the Department of Computer Science, University College London (UCL). Before joining UCL, he was a Postdoctoral Researcher at the Italian Institute of Technology for five years. His research interests include robot perception, planning, and learning, while he has worked on more than six humanoid/animaloid robots. He has published more than 40 research articles in high-impact robotic journals and conferences, while he has won the Best Interactive Paper Award in IEEE Humanoids 2017 and the Best Student Paper Award Finalist in IEEE ICARCV 2018.

DARWIN G. CALDWELL (Fellow, IEEE) received the B.Sc. and Ph.D. degrees in robotics from the University of Hull, in 1986 and 1990, respectively, and the M.Sc. degree in management from the University of Salford, in 1994. He is currently the Founding Director of the Italian Institute of Technology (IIT), where he is also the Director of the Department of Advanced Robotics. He is or has been an Honorary Professor at the Universities of Manchester, Sheffield, King's College, the University of Bangor, U.K., and Tianjin University, China. He is a pioneer in the development of compliant and variable impedance actuation, soft and human friendly robotics and the creation of 'softer', safer robots that have been fundamental to advances in humanoids, quadrupeds, exoskeletons, and medical robots. Key robots developed by him and his teams include: iCub, a child-sized humanoid robot; COMAN, a controllably compliant humanoid; WALK-MAN, that competed in the DARPA Robotics Challenge; the HyQ series of high performance quadrupeds; and Centauro, a centaur based robot for harsh, unstructured environments. He has published over 700 papers and won/been nominated for over 50 conference/journal awards. His research interests include medical/surgical robotics, exoskeletons and haptics. He is a fellow of the Royal Academy of Engineering.

NIKOS G. TSAGARAKIS (Member, IEEE) received the D.Eng. degree in electrical and computer science engineering from the Polytechnic School, Aristotle University of Thessaloniki, Thessaloniki, Greece, in 1995, and the M.Sc. degree in control engineering and the Ph.D. degree in robotics from the University of Salford, Salford, U.K., in 1997 and 2000, respectively. Since 2013, he has been working as a Visiting Professor with the Center for Robotics Research, Department of Informatics, King's College University, London, U.K. He is currently a Tenured Senior Scientist with the Istituto Italiano di Tecnologia, Genoa, Italy, with overall responsibility for humanoid design and human centered mechatronics development. He was a Technical Editor of the IEEE/ASME TRANSACTIONS ON MECHATRONICS. He is on the Editorial Board of the IEEE ROBOTICS AND AUTOMATION LETTERS.