2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

A Control Barrier Function-based Motion Planning Scheme for a Quadruped Robot

Halil Utku Unlu[1], Vinicius Mariano Gonçalves[2], Dimitris Chaikalis[1], Anthony Tzes[2,3] and Farshad Khorrami[1,2]

***Abstract*— A Control Barrier Function (CBF)-based motion planning algorithm is proposed. The algorithm explores an unknown environment to reach a target point, providing velocity commands to the robot controller module. CBFs, along with a circulation inequality are used to generate safe paths toward the goal while preventing collisions with obstacles. The proposed global navigation scheme is experimentally verified on a quadruped platform to demonstrate safe, collision-free exploration over long distances.**

I. Introduction

Developments in manufacturing technologies, computation capabilities, and software toolkits enabled the realization of legged robots that can traverse uneven terrains that wheeled or skid-steering robots cannot, although with the added complexity, cost, and power consumption [1]. Safety becomes an important consideration for not only preventing damage to the environment but also to the equipment itself.

Navigation, a prerequisite for autonomous capabilities in a mobile robotic platform, depends on the position and orientation estimation in the working environment. Drift-free position information through global navigation satellite systems (GNSS) [2], [3] is not reliable in artificial, indoor structures where the radio signals cannot reach the robot. Many options are available for GNSS-denied localization [4]. In a controlled environment, a robot can extract reliable pose information via the use of motion capture systems [5], fiducial markers affixed to the environment at known locations [6] or in known shapes [7], with ultra-wideband (UWB) or similar radio frequency beacons [8], or with prior information about the environment [9].

Unfortunately, many real-life robotic missions happen in uncontrolled, unknown environments, invalidating the use of all of the aforementioned methods. In an unknown navigation scenario, the vehicle needs to simultaneously localize itself and generate a map of the environment, a problem commonly referred to as simultaneous localization and mapping (SLAM) [10]. However, false associations and dynamic objects can degrade the map quality significantly, creating a need for robust path planning and navigation algorithms with safety guarantees.

In this work, a navigation algorithm that does not assume any *a priori* knowledge of the map and that leverages the benefits of Control Barrier Functions (CBFs) [11]–[16] is proposed. With the CBF framework, it is possible to compute safe velocity commands guiding the robot to a target point, given a function that provides the distance between the robot and the environment at a given configuration and a target velocity. This is used to both plan a safe path towards the target and to avoid collisions in the proposed algorithm. For path planning , it is known that CBFs are prone to generate paths that converge to undesirable equilibrium points [17]. To mitigate this issue, the modification proposed in [18] is applied, which incorporates additional inequality constraints to circulate around obstacles. This modification is especially useful in maze-like environments with long corridors and corner points. Following the walls either clockwise or counter-clockwise is often enough to generate a safe, smooth path toward a target, which will be demonstrated in the experimental section of this paper.

Unlike many of the existing path planners ([19]–[21] to cite some recent examples), the proposed approach is deterministic, not requiring any probabilistic schemes (e.g. RRTs, PRMs) for navigation. Deterministic planners have advantages over non-deterministic ones as predictability and simplicity (it is easier to verify and validate). Finally, the proposed idea can be adapted to other modes of motion. Although the demonstrated implementation considers a robot in 2D without non-holonomic constraints (with 2D position and orientation controlled independently), with some modifications the idea can also be applied to other types of robots as drones that move in the 3D space with arbitrary 3D rotations.

The contributions of this paper are:

- A *deterministic* and *global* planner with CBF and circulation constraints that can traverse complex paths in maze-like environments,
- The formulation of a differentiable distance-like function between a robot represented as a rectangle, and a discrete point cloud representation of the environment. This differentiable formulation is necessary to implement the CBF inequalities.
- A modification of the vector field implementation proposed in [22] that can be generalized to motion in SE(2),
- Experimental validation of the proposed method on a real quadruped platform.

[1]New York University, Electrical & Computer Engineering Department, Brooklyn, NY 11201, USA.
[2]Center for Artificial Intelligence and Robotics (CAIR), New York University Abu Dhabi (NYUAD), United Arab Emirates (UAE).
[3]NYUAD, Electrical Engineering, Abu Dhabi 129188, UAE.
This work was partially supported by the NYUAD Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010. Work was partially performed in NYUAD's Kinesis Lab, Core Technology Platform.

The details of the proposed algorithm are provided in Section II. Experimental studies, along with the parameters used for the test, are given in Section III, and concluding remarks can be found in Section IV.

Mathematical notation: All the vectors are column vectors. $(\cdot)^\top$ represents the transpose. Given a function $f(q)$, $\nabla_g f(q)$ is a column vector representing the gradient on the subset of variables g of q. $\mathbb{S}$ is the set of points in a circle, representing the topology of rotations in the 2D space. The symbol $0_{a\times b}$ represents the $a \times b$ zero matrix.

II. Methodology

A. Problem statement

This paper considers a robot in a plane with position $p \in \mathbb{R}^2$ and orientation $\theta \in \mathbb{S}$, as expressed in a fixed (world) frame. The vector $q = [p^\top,\ \theta]^\top \in \mathbb{R}^2 \times \mathbb{S}$ is the system *configuration*. It is assumed that the robot's linear velocity and angular velocity can be controlled independently, that is, $\dot{p} = v_D$ and $\dot{\theta} = \omega_D$ and thus, as far as the proposed algorithm is concerned, $v_D \in \mathbb{R}^2$ and $\omega_D \in \mathbb{R}$ are the system inputs. It is assumed that the environment is not known *a priori*. Furthermore, the robot is equipped with a LiDAR that allows it to build a (time-varying) *map* of the environment $\mathcal{M}(t)$, represented as a (time-varying) finite set of points $r_i \in \mathbb{R}^2$, also expressed in the world frame.

The robot aims to reach a target position p_{GL}, described in the world frame, though the same algorithm can be used for exploration with a slight modification.

B. Algorithm overview

Figure 1 shows an overview of the control scheme. At the top, the *Localization/Mapping* module uses the LiDAR measurements to estimate the robot's configuration q and to generate the current map $\mathcal{M}$. This information is fed to the *Global Planner* module whose purpose is to compute the linear velocity v_D and angular velocity ω_D for the robot. The computed velocities are then provided to a *Velocity Controller* module that controls the motor motions to comply with these velocities.

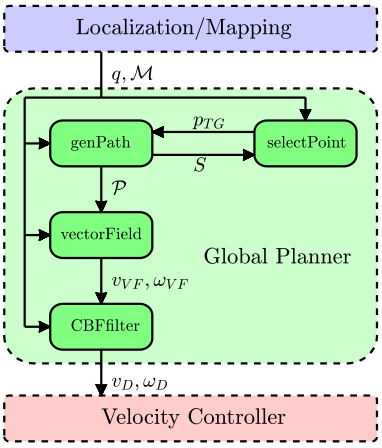

Fig. 1. Overview of the algorithm.

The localization/mapping module will be discussed in Subsection II-C. The structure of the *Global Planner* is shown in more detail in Figure 1. It is divided into four sub algorithms which will be explained in detail in the forthcoming subsections. The velocity controller module is omitted as the high-level controllers on board the quadruped platform are used.

The Global Planner, the `genPath` function, receives the current configuration, current map, and the *current* target position p_{TG} (not necessarily p_{GL}, as explained later) and generates a safe path (position and orientation) from the starting configuration to the target position. It returns a path $\mathcal{P}$ and a flag S saying if it was successful or not in planning the path towards p_{TG}. The path is not updated at every step of the control loop but at every T_{GP} step-units.

Assuming that a path is successful ($S = \mathit{TRUE}$), the path and the current configuration are passed to the `vecField` algorithm that generates linear and angular velocity commands v_{VF}, ω_{VF} to track the path $\mathcal{P}$. The path $\mathcal{P}$ is planned to be safe with the map information available at the time (however, the path may lead to collisions with the unexplored portions of the environment). To mitigate potential collisions, the `CBFfilter` function takes the current map, configuration, and velocities v_{VF}, ω_{VF} generated from the vector field and "filters" them to guarantee that the velocities generate safe motions with the most up-to-date map knowledge. These filtered results are then supplied to the velocity controller.

If the path from `genPath` is unsuccessful ($S = \mathit{FALSE}$), then the robot reaches a dead end. In this case, `selectPoint` algorithm is executed to select a new target point p_{TG} to explore the environment, aiming to reach p_{GL}. This is done by selecting a suitable *frontier point* to explore, following the seminal work [23].

C. The localization/mapping module

Leveraging the LiDAR on board, the robot uses KISS-ICP [24], an odometry algorithm that uses Point-to-Point ICP for fast and reliable odometry without any additional sensor, as the basis for localization. No inherent loop closure mechanisms are implemented within KISS-ICP, but the system is observed to provide minimal drift, thus providing an ideal condition to test the safety guarantees of the proposed system. Odometry information obtained from KISS-ICP is used to generate an octree-based 3D occupancy grid map with OctoMap [25] library. To reduce the computational load in mapping, LiDAR readings are integrated into the OctoMap up to $d_{\max}$ distance away.

Following [26], the 3D binary occupancy information is reduced to a 2D binary occupancy map via a conservative projection of the map within a height range $[z_{\min}, z_{\max}]$ onto xy-plane. The time-varying map representation $\mathcal{M}(t)$ is comprised of the occupied voxels of the projected 2D binary occupancy grid map. The odometry information projected onto this 2D occupancy map provides the system configuration vector q.

D. The robot-to-map safety function

Given a map $\mathcal{M}$ as a set of points $r_i \in \mathbb{R}^2$, it is necessary to define a *differentiable* function that measures how far the robot is from this point cloud given the current configuration. This function does not have to be exactly the Euclidean distance function but should approach 0 when close to the points and increase when far from it. The metric will be called *robot-to-map safety* function.

Before this, the *robot-to-point safety function* $S : \mathbb{R}^2 \times \mathbb{S} \times \mathbb{R}^2 \mapsto \mathbb{R}$ will be defined. This function $S(q, r)$ gives a metric between the robot at the configuration $q \in \mathbb{R}^2 \times \mathbb{S}$ and a single point $r \in \mathbb{R}^2$, with $r = [r_x \ r_y]^\top$. For this, it is convenient to consider first the function $S_0(r)$ that computes this safety in the particular case when $p = 0_{2\times1}$ and $\theta = 0$, that is, $S_0(r) = S(0_{3\times1}, r)$. This function has three parameters: the *robot width* W_R, *robot length* L_R and the *smoothing parameter* h_R, that will define the geometry of the robot. The function $S_0(r)$ is defined as:

$$h_R^2 \log\left[\frac{1}{2}\left(\exp\left(\frac{r_x^2 - L_R^2/4}{h_R^2}\right) + \exp\left(\frac{r_y^2 - W_R^2/4}{h_R^2}\right)\right)\right].$$

The level set $\{r \in \mathbb{R}^2 \ | S_0(r) = 0\}$ can be seen in Figure 2, and represents the geometry for the robot. It is a smoothed rectangle with (approximate) width and length W_R and L_R respectively. The smoothing is necessary because the gradient of this function on r will be necessary. The gradient is not always continuous for a rectangular geometry. Regardless, with this definition, it is possible to define $S(q, r)$ simply as $S(q, r) \triangleq S_0(Q(\theta)^\top (r - p))$, in which $Q(\theta)$ is the 2×2 rotation matrix of the angle θ counter-clockwise. This means that the safety is computed by mapping the point r written on the (p, θ) coordinates into (p', θ') coordinates in which $p' = 0_{2\times1}$ and $\theta' = 0$, and then applying S_0.

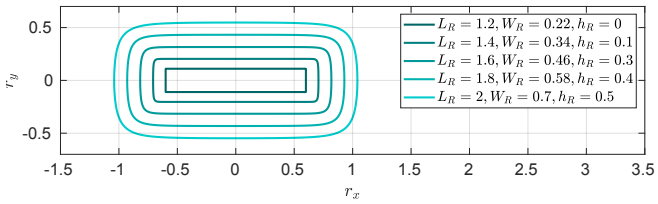

Fig. 2. Geometry of the robot (set $\{r \in \mathbb{R}^2 \ | S_0(r) = 0\}$) for different values of L_R, W_R and h_R. Note that when $h_R \to 0$, the geometry approximates a rectangle with width W_R and length L_R.

Once $S(q, r)$ is defined, it is possible to define the *robot-to-map safety* function $S_\mathcal{M}(q)$. One first idea is to define $S_\mathcal{M}(q)$ as the minimum of the values $S(q, r_i)$ among all points $r_i \in \mathcal{M}$. However, this definition implies a non-differentiable function $S_\mathcal{M}$ due to the effect of the minimum function. This issue can be solved by using a smoothed (differentiable) approximation of the minimum [27]. So, it is possible to define:

$$S_\mathcal{M}(q) \triangleq -h_S^2 \log\left(\frac{1}{|\mathcal{M}|}\sum_{i=1}^{|\mathcal{M}|} \exp\big(-S(q, r_i)/h_S^2\big)\right) \quad (1)$$

in which h_S is a parameter. When $h_S \to 0$, $S_\mathcal{M}(q)$ converges to the true minimum function.

E. The `genPath` algorithm

Given a starting pose $q_0 \in \mathbb{R}^2 \times \mathbb{S}$, the map $\mathcal{M}$ and a desired position p_{TG}, the $(\mathcal{P}, S) \leftarrow \texttt{genPath}(q_0, \mathcal{M}, p_{TG})$ function should provide a path in the pose space between the starting pose and the final position (no constraint is imposed in the final orientation) while considering the obstacles codified into $\mathcal{M}$. This is done by using a CBF+Circulation formulation, proposed in [18], with a modification to consider the orientation of the robot. This formulation will generate a sequence of linear $v = [v_x \ v_y]^\top$ and angular ω velocities that, once integrated numerically, generate a path in the pose space. This path will guide the starting pose to the desired position.

For the objective function, a *steering controller for the linear velocity* $u_v(q) \triangleq -K_v(p - p_{TG})$, for a positive scalar K_v, is considered. This would drive the position to the target one if the obstacles were disregarded. Thus, one part of the objective function is $\|v - u_v(q)\|^2$, in which v is the linear velocity. Furthermore, even though the quadruped does not have a non-holonomic constraint, it is desirable to impose it as a "soft constraint" to induce the robot to move forward (instead of sideways) as much as possible. So, a *steering controller for the angular velocity*, $u_\omega(q, v) \triangleq -K_\omega \sin(\theta) v_x + K_\omega \cos(\theta) v_y$, for a positive scalar K_ω, is also included. This induces the non-holonomic constraint $-\sin(\theta)v_x + \cos(\theta)v_y = 0$. Define $r_\omega(q) = [-K_\omega \sin(\theta) \ K_\omega \cos(\theta)]$. Thus the other part of the objective function is $\|\omega - r_\omega(q)v\|^2$. The objective function is therefore:

$$\min_{v,\omega} \quad \|v - u_v(q)\|^2 + \|\omega - r_\omega(q)v\|^2 \quad (2)$$

which is quadratic and strictly positive definite on the variables (v, ω). The first constraint of this optimization problem is the classical CBF constraint:

$$\frac{d}{dt}S_\mathcal{M} = \nabla_p S_\mathcal{M}(q)^\top v + \frac{\partial S_\mathcal{M}}{\partial \theta}(q)\omega \geq -K_{CBF} S_\mathcal{M}(q) \quad (3)$$

for a positive K_{CBF}. It is important to note that since the map is time-varying, so is the function $S_\mathcal{M}(q) = S_{\mathcal{M}(t)}(q)$. Therefore, the partial derivative term in t, $\frac{\partial S_{\mathcal{M}(t)}}{\partial t}(q)$, should be added to (3). However, this term is disregarded, especially since the function $S_{\mathcal{M}(t)}(q)$ is not continuous -and therefore not differentiable - on t, due to the fact that the map is updated "discretely" by new points being added into the map.

The other constraint is the modification proposed in [18], which helps the system avoid local equilibrium points. Given an anti-symmetric 2×2 matrix Ω and a function $\beta : \mathbb{R} \mapsto \mathbb{R}$ that is decreasing and such that $\beta(0) > 0$, the following constraint is implemented:

$$\big(\Omega \nabla_p S_\mathcal{M}(q)\big)^\top v \geq -\beta\big(S_\mathcal{M}(q)\big). \quad (4)$$

With (2) as objective function and constraints in (3) and (4) a strictly convex QP for the variables v, ω is defined. With

this QP, one can integrate it for T time units to generate a path into the pose space.

This path generation procedure is performed for three different values of matrices Ω, namely a positive rotation of 90^o, a negative rotation of 90^o, and the zero matrix (no circulation). For the "no circulation", the constraint in (4) is effectively removed. If no path is successful in reaching the target point, the `genPath` algorithm returns $S = \texttt{FALSE}$ and an empty path $\mathcal{P}$. If at least one of them is successful, the `genPath` algorithm gets the best among them (the shortest) and performs a post-processing step on it to improve its safety. So given the samples q_k of this path, iterations of the form $q_k \leftarrow q_k + \eta \nabla_q S_{\mathcal{M}}(q_k)$ for a positive η are performed. This path is then returned along with the success flag, $S = \texttt{TRUE}$.

F. The `vectorField` algorithm

For path tracking, a *vector field* approach based on [22] is used to compute $(v_{VF}, \omega_{VF}) \leftarrow \texttt{vectorField}(q, \mathcal{P})$, i.e., the velocities that should be followed to track the path $\mathcal{P}$ at the current configuration q (the map $\mathcal{M}$ is not used in this function). For the sake of mathematical exposition, the path is initially considered as a differentiable mapping $q_{\mathcal{P}} : [0,1] \mapsto \mathbb{R}^2 \times \mathbb{S}$, i.e., a parameterized position $p_{\mathcal{P}}(s)$ and orientation $\theta_{\mathcal{P}}(s)$ for the robot, $q_{\mathcal{P}}(s) = [p_{\mathcal{P}}(s)^\top \ \theta_{\mathcal{P}}(s)]^\top$. The methodology presented in [22] cannot be applied directly because it assumes that the target path is Euclidean, which is not the case here due to the presence of the rotation. [22] proposes a strategy to "Euclidianize" the target curve by considering many copies of it along the dimensions of $\mathbb{S}$. Inhere, a simpler approach is proposed. Consider the distance-to-path function $D_{\mathcal{P}}(q) \triangleq \min_{s\in[0,1]} \{0.5\|p - p_{\mathcal{P}}(s)\|^2 + 1 - \cos\big(\theta - \theta_{\mathcal{P}}(s)\big)\}$. This distance function takes into consideration the topology of the rotation space $\mathbb{S}$.

To apply the methodology in [22], it is necessary to define the *normal* and *tangent* vector to the path. Let $s^*(q)$ be the minimizer of the optimization problem for $D_{\mathcal{P}}$ in a given $q \in \mathbb{R}^2 \times \mathbb{S}$. In a similar reasoning as presented in [22], it is possible to deduce that, as long as $s^*(q)$ is unique

$$\nabla_q D_{\mathcal{P}}(q) = \begin{bmatrix} p - p_{\mathcal{P}}(s^*(q)) \\ \sin\Big(\theta - \theta_{\mathcal{P}}(s^*(q))\Big) \end{bmatrix}. \tag{5}$$

This allows the definition of the *normal vector* $N_{\mathcal{P}}(q) \triangleq \nabla_q D_{\mathcal{P}}(q) / \|\nabla_q D_{\mathcal{P}}(q)\|$. The *tangent vector* $T_{\mathcal{P}}(q)$ can be defined using the derivative of $q_{\mathcal{P}}(s)$ on s as $T_{\mathcal{P}}(q) \triangleq q'_{\mathcal{P}}(s^*(q)) / \|q'_{\mathcal{P}}(s^*(q))\|$. The Karush-Kuhn-Tucker conditions applied to the optimization problem for $D_{\mathcal{P}}$ guarantee that indeed $N_{\mathcal{P}}(q)$ and $T_{\mathcal{P}}(q)$ are orthogonal (a key feature for the vector field algorithm, as explained in [22]) whenever the two vectors are defined.

Given a function $G : \mathbb{R}^+ \mapsto [0,1]$ that vanishes if and only if its argument is 0, $H(u) \triangleq \sqrt{1 - G(u)^2}$ and a speed multiplier A_{VF}, the vector field $f_{VF} : \mathbb{R}^2 \times \mathbb{S} \mapsto \mathbb{R}^3$ can be defined, following [22]:

$$f_{VF}(q) \triangleq A_{VF}\Big(G\big(D_{\mathcal{P}}(q)\big) N_{\mathcal{P}}(q) + H\big(D_{\mathcal{P}}(q)\big) T_{\mathcal{P}}(q)\Big). \tag{6}$$

The first two components of this 3D vector represent the linear velocity v_{VF} to track the path, whereas the last component is the angular velocity ω_{VF}.

The path provided by `genPath` is not a continuous function $q_{\mathcal{P}}$, but a sampled version $\{q_k\}$, $k = 1, 2, ..., N$. First, it may be necessary that this path is upsampled (using, for example, linear interpolation) to a set $\{\tilde{q}_k\} = \{[\tilde{p}_k^\top \ \tilde{\theta}_k]^\top\}$, $k = 1, 2, .., N'$, with $N' > N$. The optimization problem for $D_{\mathcal{P}}(q)$ can be solved numerically by checking all the points, obtaining the optimal index $k^*(q)$ so $q_{\mathcal{P}}(s^*(q)) = \tilde{q}_{k^*(q)}$. With this, (5) can be used to compute $N_{\mathcal{P}}(q)$. Numerical differentiation can also be used to compute $T_{\mathcal{P}}(q)$, however, special care must be taken when numerically differentiating the orientation θ_k, due to the topology in $\mathbb{S}$. With this consideration, $T_{\mathcal{P}}(q)$ can be calculated by computing $\Delta q = [(\tilde{p}_{k^*(q)} - \tilde{p}_{k^*(q)-1})^\top \ \sin(\tilde{\theta}_{k^*(q)} - \tilde{\theta}_{k^*(q)-1})]^\top$ and then $T_{\mathcal{P}}(q) \approx \Delta q / \|\Delta q\|$.

G. The `CBFfilter` algorithm

The function $(v_D, \omega_D) \leftarrow \texttt{CBFfilter}(q, \mathcal{M}, v_{VF}, \omega_{VF})$ is relatively simple. The optimization problem is a QP with an objective function:

$$\min_{v_D, \omega_D} \ \|v_D - v_{VF}\|^2 + \|\omega_D - \omega_{VF}\|^2 \tag{7}$$

with the same constraint as in (3), but with v and ω switched to v_D and ω_D respectively. By solving this QP, the outputs v_D, ω_D can be obtained.

H. The `selectPoint` algorithm

The `genPath` algorithm may fail to generate a path between the starting configuration and the target position p_{TG}. In that case, the $p_{TG} \leftarrow \texttt{selectPoint}(q, \mathcal{M})$ algorithm is responsible to select a new point p_{TG}, always aiming to find the "global" target p_{GL}. Once it is called, the algorithm processes the map $\mathcal{M}$ to find the set of *frontier* points, which is a set of points $\{r_i^F\}$ from the 2D binary occupancy map that is (a) in the contour of the free region, (b) is in the 8-neighborhood of an unknown region (i.e., a region without any label), and (c) does not have any occupied cells in its 8-neighborhood. Continuous segments of frontier points are clustered together. The point r_i^C with the largest safety function value $S_{\mathcal{M}}(r_i^C)$ is selected from each cluster, reducing the set of points that are candidates for exploration.

The selection is based on the following criteria: the frontier points that provide the shortest *predicted* route between the current configuration q and the global target p_{GL}, $LP(r_i^C)$. The predicted path length is composed of two terms, LP_1 and LP_2, that are computed using the *exploration graph* $\mathcal{G}$. This graph is constructed as the robot moves in the environment. Each node represents a visited position p and there is an edge between two nodes if the robot travels between the respective points. The weight of the edge is the traveled distance.

The first component, LP_1, is the length of the path from the current configuration q to the candidate frontier point r_i^C. This can be estimated by the graph $\mathcal{G}$ by finding the shortest path (on the graph) between the closest node to the current position p (so the robot orientation is ignored) and

the frontier point r_i^C. The algorithm disregards orientation to enhance efficiency. Given that the robot is not subject to non-holonomic constraints, it is expected to accurately follow any continuous path within the position space without being significantly affected by its orientation. The second component, LP_2, can be estimated by using the `genPath` function to plan a path between the frontier point r_i^C and the global target p_{TG}. If this function is not successful, LP_2 is assigned to a very large number, but this should not prevent the point r_i^C from being selected.

Execution time for `selectPoint` algorithm depends on the number of frontier points to be tested and the number of obstacle points to validate. With the naive implementation on a single thread, the execution time can reach 10 s. The algorithm permits parallelization (i.e. multi-core or GPU) to reduce `selectPoint` algorithm computation, but is not implemented as part of this work. Once a frontier point is selected, the `selectPoint` algorithm will generate a sequence of target points p_{TG} that travels the graph and then goes to the selected frontier point. Once it is reached, p_{TG} is set to the global target p_{GL}.

III. Experimental Studies

The algorithm was tested at the interior of the NYUAD campus using a Unitree B1 quadruped robot (see Figure 4, top left tile). The robot is tasked to reach a rough coordinate provided in the map frame, without any prior information about the environment and layout. The robot needs to navigate through narrow openings that do not admit to crossing in any arbitrary orientation for a rectangular footprint. Figure 3 provides a sketch of the environment, showing the geometry, starting point, target point, and obstacles (boxes and doors).

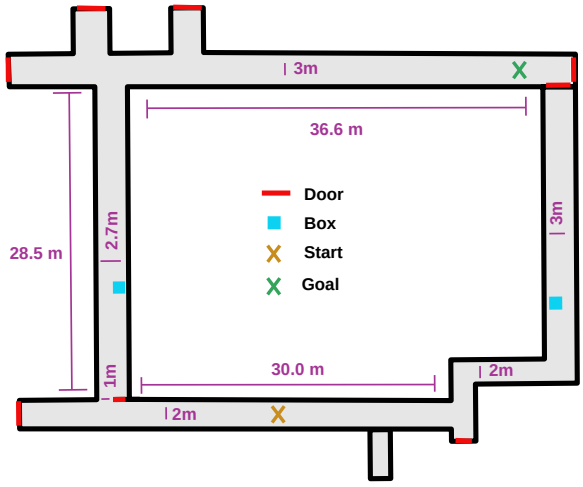

Fig. 3. Sketch of the environment used in the experiment.

The Localization/Mapping module generates pose and map estimates at 10 Hz, matching the LiDAR update rate. Map updates are performed at $d_{\max} = 10$ m with a resolution of 0.1 m. The OctoMap algorithm is observed to fall behind the target rate, but odometry updates are received consistently at around 10 Hz. As such, the functions `vectorField` and `CBFfilter` match the 10 Hz update rate.

The value selected for T_{GP} was 25 steps, which means that a path is replanned using `genPath` at every $25/10$ s = 2.5 s. For the robot geometry, $h_R = 0.15$ m, $L_R = 1.5$ m, $W_R = 0.50$ m and $h_S = 0.15$ m were used. For the `genPath` parametrization, $K_v = 0.4\ s^{-1}$, $K_\omega = 4.0\ s^{-1}$, $K_{CBF} = 1.0\ s^{-1}$ and $\beta(s) = c_\beta(1 - s/s_0)$ in which $c_\beta = 1.25$ m/s, $s_0 = 0.10$ m. For the `vectorField` algorithm, the parameters selected were $A_{VF} = 0.45$ m/s and $G(u) = (2/\pi)atan(\sqrt{u/u_G})$, in which $u_G = 0.25$ m.

Figure 4 shows photos from the experiment, while Figure 5 provides visualizations of the experiment, highlighting the characteristics of the algorithm, while also showing the timestamp and the current circulation matrix Ω. The robot was able to finish the task in 8 minutes, traveling 183 m (and thus with an average speed of 0.38 m/s). Note that the proposed circulation+CBF planner was appropriate for the geometry of the environment, composed of corridors with corners. This is seen in Figure 5-(Top Center), in which the plan was able to circulate the corners towards the target.

Fig. 4. Snapshots of the experiment at **Top Left:** $t = 0$ s, **Top Right:** at $t = 80$ s, **Bottom Left**: at $t = 305$ s and **Bottom Right:** at $t = 380$ s. To see the robot in action, see https://bit.ly/risclab-cbf-quadruped-icra24.

IV. Conclusion

In this paper, a CBF-based path planning algorithm was proposed. The algorithm leverages the properties of CBFs [11], the circulation inequalities proposed in [18], a differentiable distance-like formulation using point clouds and a modification of the vector field proposed in [22]. The utility of the approach is demonstrated on a legged robot, Unitree B1 quadruped, operating in an unknown environment. The quadruped is equipped with a LiDAR to create a partial view of the environment as the robot navigates. Although the ideas were presented and tested on a planar robot with no non-holonomic constraints, they can be generalized for other settings, like an omnidirectional drone that can accept arbitrary linear velocities and angular velocities.

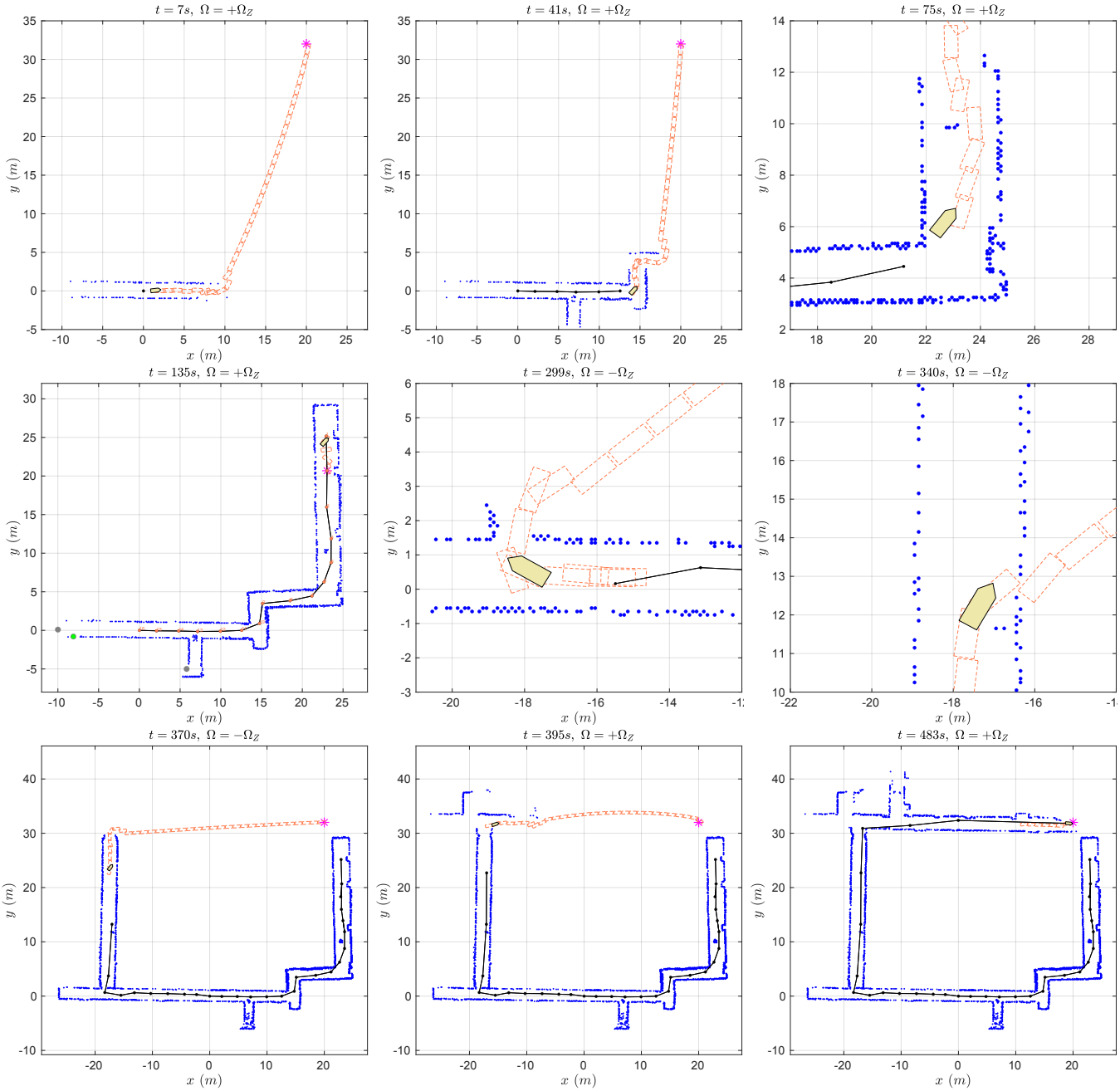

Fig. 5. **Top Left:** the robot starts planning a path $\mathcal{P}$ towards the goal (orange path), given the map so far (blue points). **Top Center:** the robot executes given velocities. Note the graph $\mathcal{G}$ (in black) mentioned in Subsection II-H. **Top Right:** zoom in on the robot planning to evade the rightmost box (see Figure 3). **Middle Left:** The robot reaches a dead end (see Figure 3), preventing it from reaching the target. The robot plans to explore a new frontier (gray dots, with the selected one in green) and then travels the graph to reach the frontier points. **Middle Center:** Upon reaching the frontier, the robot replans a path towards the target through the 1 m wide door (see Figure 3). **Middle Right:** Going through the corridor, it eventually reaches the leftmost box. The `genPath` algorithm did not replan the path yet, considering the newly discovered points. However, collision is avoided because of the layer `CBFfilter`, which acts much more often. **Bottom Left & Center**: planned paths towards the goal. **Bottom Right:** The robot completed the task.

REFERENCES

[1] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419839596, 2019.

[2] Y. Feng, J. Wang, *et al.*, "GPS RTK performance characteristics and analysis," *Positioning*, vol. 1, no. 13, 2008.

[3] X. Li, X. Li, Y. Yuan, K. Zhang, X. Zhang, and J. Wickert, "Multi-GNSS phase delay estimation and PPP ambiguity resolution: GPS, BDS, GLONASS, Galileo," *Journal of geodesy*, vol. 92, pp. 579–608, 2018.

[4] N. Gyagenda, J. V. Hatilima, H. Roth, and V. Zhmud, "A review of GNSS-independent UAV navigation techniques," *Robotics and Autonomous Systems*, p. 104069, 2022.

[5] G. Guerra-Filho, "Optical Motion Capture: Theory and Implementation," *RITA*, vol. 12, no. 2, pp. 61–90, 2005.

[6] B. Mantha and B. Garcia de Soto, "Designing a reliable fiducial marker network for autonomous indoor robot navigation," in *Proceedings of the 36th International Symposium on Automation and Robotics in Construction (ISARC)*, pp. 74–81, 2019.

[7] N. Evangeliou, D. Chaikalis, A. Tsoukalas, and A. Tzes, "Visual Collaboration Leader-Follower UAV-Formation for Indoor Exploration," *Frontiers in Robotics and AI*, vol. 8, p. 777535, 2022.

[8] B. Yang and E. Yang, "A survey on radio frequency based precise localisation technology for UAV in GPS-denied environment," *Journal of Intelligent & Robotic Systems*, vol. 103, pp. 1–30, 2021.

[9] F. Wang, K. Wang, S. Lai, S. K. Phang, B. M. Chen, and T. H. Lee, "An efficient UAV navigation solution for confined but partially known indoor environments," in *11th IEEE International Conference on Control & Automation (ICCA)*, pp. 1351–1356, IEEE, 2014.

[10] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[11] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control Barrier Functions: Theory and Applications," in *2019 18th European Control Conference*, pp. 3420–3431, 2019.

[12] V. M. Gonçalves, D. Chaikalis, A. Tzes, and F. Khorrami, "Safe multi-agent drone control using control barrier functions and acceleration fields," *Robotics and Autonomous Systems*, vol. 172, p. 104601, 2024.

[13] B. Dai, P. Krishnamurthy, and F. Khorrami, "Learning a Better Control Barrier Function," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 945–950, IEEE, 2022.

[14] B. Dai, H. Huang, P. Krishnamurthy, and F. Khorrami, "Data-Efficient Control Barrier Function Refinement," in *2023 American Control Conference (ACC)*, pp. 3675–3680, IEEE, 2023.

[15] B. Dai, R. Khorrambakht, P. Krishnamurthy, V. Gonçalves, A. Tzes, and F. Khorrami, "Safe Navigation and Obstacle Avoidance Using Differentiable Optimization Based Control Barrier Functions," *IEEE Robotics and Automation Letters*, vol. 8, no. 9, p. 5376–5383, 2023.

[16] H. U. Unlu, D. Chaikalis, V. Gonçalves, and A. Tzes, "Control Barrier Functions and LiDAR-Inertial Odometry for Safe Drone Navigation in GNSS-Denied Environments," in *Motion Planning for Dynamic Agents*, IntechOpen, 2023.

[17] M. F. Reis, A. P. Aguiar, and P. Tabuada, "Control Barrier Function-Based Quadratic Programs Introduce Undesirable Asymptotically Stable Equilibria," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 731–736, 2021.

[18] V. M. Gonçalves, P. Krishnamurthy, A. Tzes, and F. Khorrami, "Using Circulation to Mitigate Spurious Equilibria in Control Barrier Function," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 1770–1775, IEEE, 2023.

[19] A. Bircher, M. S. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding Horizon "Next-Best-View" Planner for 3D Exploration," in *2016 IEEE International Conference on Robotics and Automation*, pp. 1462–1468, 05 2016.

[20] A. Manjunath and Q. Nguyen, "Safe and Robust Motion Planning for Dynamic Robotics via Control Barrier Functions," 2021.

[21] M. Naazare, F. G. Rosas, and D. Schulz, "Online Next-Best-View Planner for 3D-Exploration and Inspection With a Mobile Manipulator Robot," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3779–3786, 2022.

[22] A. M. C. Rezende, V. M. Goncalves, and L. C. A. Pimenta, "Constructive Time-Varying Vector Fields for Robot Navigation," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 852–867, 2022.

[23] B. Yamauchi, "A Frontier-Based Approach for Autonomous Exploration," in *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, CIRA '97, p. 146, IEEE Computer Society, 1997.

[24] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 2, pp. 1029–1036, 2023.

[25] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on Octrees," *Autonomous Robots*, vol. 34, pp. 189–206, 2013.

[26] H. U. Unlu, D. Chaikalis, A. Tsoukalas, and A. Tzes, "UAV Indoor Exploration for Fire-Target Detection and Extinguishing," *Journal of Intelligent & Robotic Systems*, vol. 108, no. 3, p. 54, 2023.

[27] B. Gao and L. Pavel, "On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning," *ArXiv*, vol. abs/1704.00805, 2017.