

# Robust Sample-Based Output-Feedback Path Planning

Mahroo Bahreinian<sup>1</sup>, Marc Mitjans<sup>2</sup>, and Roberto Tron<sup>3</sup>

**Abstract**—We propose a novel approach for sampling-based and control-based motion planning that combines a representation of the environment obtained via modified version of optimal Rapidly-exploring Random Trees (RRT\*), with landmark-based output-feedback controllers obtained via Control Lyapunov Functions, Control Barrier Functions, and robust Linear Programming. Our solution inherits many benefits of RRT\*-like algorithms, such as the ability to implicitly handle arbitrarily complex obstacles, and asymptotic optimality. Additionally, it extends planning beyond the discrete nominal paths, as feedback controllers can correct deviations from such paths, and are robust to discrepancies between the map used for planning and the real environment. We test our algorithms first in simulations and then in experiments, testing the robustness of the approach to practical conditions, such as deformations of the environment, mismatches in the dynamical model of the robot, and measurements acquired with a camera with a limited field of view.

## I. INTRODUCTION

The problem of motion planning from an initial state toward a goal state has received great attention in mobile robotics. One of the currently most popular techniques for solving this problem is represented by sampling-based algorithms, where the planner is not given an explicit representation of the environment (e.g., via polygons), but instead uses a *sampling function* that can be used to query whether an arbitrary point is in free space or inside an obstacle. Together with a *steering function* that can find trajectories between samples, algorithms such as RRT\* and derivatives build a *tree* that is rooted at the goal location and that extends toward every reachable point in the free space. When such tree arrives at a given starting location, a nominal feasible path can be found by tracing it back along the tree to the root. However, in practice, following this path requires full knowledge of the position in the environment and a lower-level control to compensate for disturbances. Additionally, although the sampling process naturally reveals information about the obstacles via sample collisions, such information is typically discarded after planning; relatedly, traditional approaches do not address the fact that there might be discrepancies between the implicit map given by the sampling function and the real free configuration space.

This work was supported by ONR MURI N00014-19-1-2571 “NeuroAutonomy: Neuroscience-Inspired Perception, Navigation, and Spatial Awareness”

<sup>1</sup>Division of Systems Engineering at Boston University, Boston, MA, 02215 USA. {mahroobh@bu.edu}

<sup>2</sup>Department of Mechanical Engineering at Boston University, Boston, MA, 02215 USA. M. Mitjans is additionally supported by “la Caixa” Foundation fellowship LCF/BQ/AA18/11680117. {mmitjans@bu.edu}

<sup>3</sup>Department of Mechanical and System Engineering.{tron@bu.edu}

In this paper we take advantage of the capabilities of RRT\*-like algorithms to effectively represent the free configuration space, but augment it with linear output-feedback controllers that guide the state along the edges of the tree graph based on the observation of *landmarks*, points whose location is known in the map and that can be easily recognized (but can be generally distinct from the obstacles or generated samples). Our output-feedback controllers provide remedies to the three aforementioned shortcomings of traditional methods: 1) it enables us to simplify the tree representation (i.e., reduce the number of nodes) while also extending it to regions that were not explicitly sampled; 2) it steers clear of obstacles (within the resolution limits given by the finite sampling) by explicitly avoiding samples that were found in collision; and 3) it provides robustness to discrepancies in the map used for the planning that are reflected in the landmarks (if the actual landmark locations are somewhat different, the resulting control will change accordingly and without replanning).

### A. Review of prior work

Sampling-based planning algorithms, such as Probabilistic Road Map [1], Rapidly exploring Random Tree (RRT\*) [2], [3] and asymptotically optimal Rapidly Exploring Random Tree (RRT\*), [4], have become popular in the last few years due to their good practical performance, and their probabilistic completeness [3]–[5]. There have also been extensions considering perception uncertainty [6]. However, these algorithms only provide nominal paths, and assume that a separate low-level controller exists to generate collision-free trajectories at run time. For trajectory planning that takes into account non-trivial dynamical systems of the robot, kinodynamic RRT\* [3], [5] and closed-loop RRT\* (CL-RRT, [7]) and CL-RRT# grow the tree by sampling control inputs and then propagating forward the nonlinear dynamics (with the optional use of stabilizing controllers and tree rewiring to approach optimality). Further in this line of work, there has been a relatively smaller amount of works on algorithms that focus on producing controllers as opposed to simple reference trajectories. The safeRRT algorithm [8], [9] generates a closed-loop trajectory from an initial state to the desired goal by expanding a tree of local state-feedback controllers to maximize the volume of corresponding positive invariant sets while satisfying the input and output constraints. Based on the same idea and following the RRT\* approach, the LQR-tree algorithm [10] creates a tree by sampling over state space and stabilizes the tree with a linear quadratic regulator (LQR) feedback. With respect to the present paper, the common trait among all these works is the use of full state feedback (as opposed to output feedback).

Separately, the state-of-the-art method for synthesizing safe and stable control commands is represented by the combination of Control Barrier Functions (CBF) and Control Lyapunov Function (CLF) [11]–[13]; however, these approaches are in general not complete for complex environments (i.e., they might fail to reach the goal even when a feasible path exists).

In our algorithm, we use the min-max robust Linear Programming (LP) controller synthesis method from [14]. However, that work assumes that a polyhedral convex cell decomposition of the environment is available, which greatly reduces the applicability of that method. Moreover, that work also does not test the resulting controllers in a real experimental setting.

### B. Proposed approach and contributions

As mentioned above, at a high level, our approach first converts an implicit representation of the environment to a simplified tree graph via sampling, and then builds a sequence of linear output feedback controllers to generate piece-wise linear control laws for navigation. To build the tree, we use the RRT\* algorithm with two modifications: 1) we post-process the tree to minimize the number of nodes and decrease the overall path length of each branch, and 2) we do not discard the samples that are found to be in collision with obstacles. In addition to the sample-based tree and collision samples, we assume that the environment includes a set of landmarks that the robot can sense, such that for any location in the free space at least one landmark is available (in practice, these landmarks could correspond to visual features on the surface of obstacles, although we do not place any restriction on their location).

We then propose a way to define convex cells around each node in the tree that ensure progress from that node to its parent via a CLF constraint, while using the samples found in collision to form a local convex approximation of the free space for obstacle avoidance via CBF constraints. We apply the method of [14] to formulate a min-max robust Linear Program that synthesizes a controller for each cell which takes as inputs relative position measurements of the landmarks and outputs a control signal that respects and balances the stability constraint from the CLF, and the safety (collision avoidance) constraints from the CBF. Additionally, we show how to easily recompute the controllers (online) to handle the case where subsets of landmarks are not visible (e.g., due to the camera's limited field of view). To summarize, the main contributions of this work are:

- Integrate high-level RRT\* path planning, with low-level CLF-CBF LP control synthesis, thus allowing the method of [14] to be applied to general environments for which a cell decomposition is not available;
- Introduce a new algorithm to simplify and improve a tree generated by RRT\* algorithm with a finite number of samples (which, in general, is not optimal).
- Introduce a new method to reformulate the controller to navigate with a limited field of view.

- Obtain a method that, thanks to the use of visual features of the environment (landmarks) and output feedback controllers, automatically adapts RRT\* solutions to deformations of the environment and deviations from the nominal path without replanning;
- Implement the proposed algorithm in a real-world environment to validate the performance of our algorithm.

## II. BACKGROUND

In this section, we review the CLF and CBF constraints, and the RRT\* method in the context of our proposed work.

### A. System Dynamics

We assume that the robot has control-affine dynamics of the form

$$\dot{x} = Ax + Bu, \quad (1)$$

where  $x \in \mathcal{X} \subset \mathbb{R}^n$  denotes the state,  $u \in \mathcal{U} \subset \mathbb{R}^m$  is the system input, and  $A \in \mathbb{R}^{n \times n}$ ,  $B \in \mathbb{R}^{n \times m}$  define the linear dynamics of the system. We assume that the pair  $(A, B)$  is controllable, and that  $\mathcal{X}_i$  and  $\mathcal{U}$  are polytopic,

$$\mathcal{X} = \{x \mid A_x x \leq b_x\}, \quad \mathcal{U} = \{u \mid A_u u \leq b_u\}, \quad (2)$$

Note that, in our case,  $\mathcal{X}$  will be a convex cell centered around a sample in the tree (Section II-E).

### B. Tree graphs

A graph is a tuple  $(\mathcal{V}, \mathcal{E})$  where  $\mathcal{V}$  represents a set of nodes and  $\mathcal{E}$  represents a set of edges. If  $(i, j) \in \mathcal{E}$ , we say that  $j$  is the parent of node  $i$ . An oriented tree  $\mathcal{T}$  is a graph where each node has exactly one parent, except for the *root*, which has no parents. We refer to nodes without children as *leaves*.

### C. Optimal Rapidly-Exploring Random Tree (RRT\*)

In this section we review RRT\*, an algorithm which is typically used for single-query path planning, but that can also be used to build a representation of the free configuration space starting from a given root node (in this paper we use it for the latter purpose). The algorithm builds a tree  $\mathcal{T}$  and is summarized in Algorithm 1, and its main functions are:

- `RandomSample`: return a random sample from a uniform probability distribution in configuration space  $\mathcal{X}$ .
- `IsSampleCollision`: return *True* if the given sample is in collision with an obstacle.
- `Nearest`: return the node in  $\mathcal{V}$  closest to the random sample  $x_{\text{rand}}$ .
- `Near`: return a set of nodes in  $\mathcal{V}$  within distance  $r^*$  from the  $x_{\text{rand}}$  where  $r^*$  is defined as

$$r^* = \min(\gamma^*(\frac{\log |\mathcal{V}|}{\mathcal{V}})^{\frac{1}{d+1}}, \eta), \quad (3)$$

$d$  is the dimension of the configuration space,  $\eta$  is the constant in the definition of the Steering function,  $\gamma^* = 2((1 + \frac{1}{d})\frac{A_{\text{free}}}{\pi})^{\frac{1}{d}}$ , and  $A_{\text{free}}$  represents the area of the free space.

- `Steering`: given two states  $p$  and  $p'$ , and  $\eta$ , return a path from  $p$  toward  $p'$  with length  $\eta$  if there is no collision between the path and the obstacles.

---

**Algorithm 1** RRT\*

```

1: Input (Obstacle lists  $\mathcal{O}$ , start point,  $\text{max}_{\text{itr}}$ ,  $\eta$ )
2:  $\mathcal{V} \leftarrow$  start point,  $\mathcal{E} \leftarrow \emptyset$ ,  $\mathcal{V}_{\text{collision}} \leftarrow \emptyset$ 
3: for  $i = 1, \dots, \text{max}_{\text{itr}}$  do
4:    $x_{\text{rand}} \leftarrow \text{RandomSample}$ 
5:   if  $\text{IsSampleCollision}(x_{\text{rand}})$  then
6:     Append  $x_{\text{rand}}$  to  $\mathcal{V}_{\text{collision}}$  and break
7:   end if
8:    $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}})$ 
9:    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}}, \eta)$ 
10:  if  $\text{isEdgeCollision}(x_{\text{nearest}}, x_{\text{new}})$  then
11:     $X_{\text{near}} \leftarrow \text{Near}(G = (V, E), x_{\text{new}}, r^*)$ 
12:     $\mathcal{V} \leftarrow \mathcal{V} \cup x_{\text{new}}$ 
13:     $x_{\text{min}} \leftarrow x_{\text{nearest}}$ 
14:     $c_{\text{min}} \leftarrow \text{cost}_{\text{nearest}} + \text{norm}(x_{\text{nearest}}, x_{\text{new}})$ 
15:    for every  $x_{\text{near}} \in X_{\text{near}}$  do
16:      if  $\text{isEdgeCollision} \wedge \text{cost}_{\text{near}} + \text{norm}(x_{\text{near}}, x_{\text{new}}) < c_{\text{min}}$  then
17:         $x_{\text{min}} \leftarrow x_{\text{near}}$ 
18:         $c_{\text{min}} \leftarrow \text{cost}_{\text{near}} + \text{norm}(x_{\text{near}}, x_{\text{new}})$ 
19:      end if
20:    end for
21:     $\mathcal{E} \leftarrow \mathcal{E} \cup (x_{\text{min}}, x_{\text{new}})$ 
22:     $\mathcal{E} \leftarrow \text{Rewire}((\mathcal{V}, \mathcal{E}), X_{\text{near}}, x_{\text{new}})$ 
23:  end if
24: end for
25: return  $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ ,  $\mathcal{V}_{\text{collision}}$ .

```

---

**Algorithm 2** Rewire

```

1: for every  $x_{\text{near}} \in X_{\text{near}}$  do
2:   if  $\text{isEdgeCollision} \wedge \text{cost}_{\text{near}} + \text{norm}(x_{\text{near}}, x_{\text{new}}) < c_{\text{min}}$  then
3:      $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}})$ 
4:      $\mathcal{E} \leftarrow (\mathcal{E} - \{x_{\text{parent}}, x_{\text{near}}\}) \cup (x_{\text{new}}, x_{\text{near}})$ 
5:   end if
6: end for
7: return  $\mathcal{E}$ .

```

---

- **isEdgeCollision**: given two states  $p$  and  $p'$ , return *True* if there is no collision between the path that connects  $p$  to  $p'$  and the obstacles; note that in general this function is typically built by using `IsSampleCollision`.
- **Rewire**: check if the cost of the nodes in  $X_{\text{near}}$  is less through  $x_{\text{new}}$  as compared to their older costs, then its parent is changed to  $x_{\text{new}}$ .

For this paper, the only modification to the original RRT\* algorithm is represented by line 5 in Algorithm 1, which stores random samples that were found to be in collision with an obstacle in the list  $\mathcal{V}_{\text{collision}}$  instead of discarding them; this list is then returned by the algorithm and will be used to define the CBF constraints in our algorithm (see Section III-D). In general, RRT\* is guaranteed to be asymptotically complete and optimal, although these guarantees do not necessarily hold with a finite number of samples.

#### D. Control Lyapunov and Barrier Functions (CLF, CBF)

In this section we review CLF and CBF constraints, which are differential inequalities that ensure stability and safety (set invariance) of a control signal  $u$  with respect to the dynamics (1). These constraints are defined. First, it is necessary to review the following.

**Definition 1:** The Lie derivative of a differentiable function  $h$  for the dynamics (1) with respect to the vector field  $Ax$  is defined as  $\mathcal{L}_{Ax}h(x) = \frac{\partial h(x(t))}{\partial x}^\top Ax$ .

Applying this definition to (1) we obtain

$$\dot{h}(x) = \mathcal{L}_{Ax}h(x) + \mathcal{L}_{Bh}(x)u. \quad (4)$$

In this work, we assume that Lie derivatives of  $h(x)$  of the first order are sufficient [15] (i.e.,  $h(x)$  has relative degree 1 with respect to the dynamics (1)); however, the result could be extended to the higher relative degree, as discussed in [14].

We now pass to the definition of the differential constraints. Consider a continuously differentiable function  $V(x) : \mathcal{X} \rightarrow \mathbb{R}$ ,  $V(x) \geq 0$  for all  $x \in \mathcal{X}$ , with  $V(x) = 0$  for some  $x \in \mathcal{X}$ .

**Definition 2:** The function  $V(x)$  is a *Control Lyapunov Function* (CLF) with respect to (1) if there exists positive constants  $c_1, c_2, c_v$  and control inputs  $u \in \mathcal{U}$  such that

$$\mathcal{L}_AV(x) + \mathcal{L}_BV(x)u + c_vV(x) \leq 0, \forall x \in \mathcal{X}. \quad (5)$$

Furthermore, (5) implies that  $\lim_{t \rightarrow \infty} V(x(t)) = 0$ .

Consider a continuously differentiable function  $h(x) : \mathcal{X} \rightarrow \mathbb{R}$  which defines a safe set  $\mathcal{C}$  such that

$$\mathcal{C} = \{x \in \mathbb{R}^n \mid h(x) \geq 0\},$$

$$\partial\mathcal{C} = \{x \in \mathbb{R}^n \mid h(x) = 0\},$$

$$Int(\mathcal{C}) = \{x \in \mathbb{R}^n \mid h(x) > 0\}.$$

In our setting, the set  $\mathcal{C}$  will represent a convex local approximation of the free configuration space (in the sense that  $x \in \mathcal{C}$  does not contain any sample that was found to be in collision). We say that the set  $\mathcal{C}$  is *forward invariant* (also said *positive invariant* [8]) if  $x(t_0) \in \mathcal{C}$  implies  $x(t) \in \mathcal{C}$ , for all  $t \geq t_0$  [16].

**Definition 3 (CBF, [17]):** The function  $h(x)$  is a *Control Barrier Function* with respect to (1) if there exists a positive constant  $c_h$ , control inputs  $u \in \mathcal{U}$ , and a set  $\mathcal{C}$  such that

$$\mathcal{L}_{Ax}h(x) + \mathcal{L}_{Bh}(x)u + c_hh(x) \geq 0, \forall x \in \mathcal{C}. \quad (7)$$

Furthermore, (7) implies that the set  $\mathcal{C}$  is forward invariant.

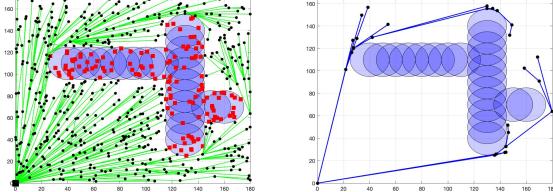
#### E. Environment

As mentioned in the previous section, the environment is implicitly represented by the `Sample` function. Additionally, we assume that the robot can measure the displacement  $\hat{l}_i - x$  between its position  $x$  and its set of *landmarks*  $\hat{l}_i$ . The location of  $\hat{l}_i$  is assumed to be known and fixed in the environment. Note that the landmarks  $\hat{l}_i$ , from the point of view of our algorithms, can be arbitrary as long as there is at least one landmark visible from any point  $x$  in the free configuration space. The landmarks do not need to be chosen from the samples of the RRT\* algorithm, or from the obstacles. Furthermore, we assume that the total extent of the environment is bounded by a convex polyhedron  $\mathcal{X}_{\text{env}}$  (e.g., simple box constraints).

### III. FEEDBACK CONTROL PLANNING VIA RRT\*

At a high level, our algorithm first divides the configuration space into cells according to a tree-graph representation of the environment, and then computes a controller for each cell that can be used to move the robot along the tree starting from any initial location. More in detail, our solution is comprised of the following steps:

- 1) Run RRT\*, and then simplify the generated tree.



(a) Tree generated by RRT\*

(b) Simplified tree

Fig. 1: Obstacles are represented by blue circles, and the start point of RRT\* is located at the origin. In Fig. 1a, yellow dots show the samples in collision with the obstacles, and the generated tree from RRT\* is plotted in green. Fig. 1b depicts the simplified tree following Section III-A.

- 2) Define a convex cell around every node of the tree while taking into account the position of its parent.
- 3) Define the CLF and CBF constraints for each cell.
- 4) Use a robust LP formulation to compute a controller for each cell that respects the CLF and CBF constraints.
- 5) Reformulate the controller in terms of visible and invisible landmarks for the limited field of view of the robot.

Below we give the details of each one of the steps.

#### A. Simplified Tree Graph

We start with a tree  $\mathcal{T} = (\mathcal{V}, \mathcal{E})$  generated by the traditional RRT\* algorithm from Section II-C. Since the number of samples is finite, the generated tree is not asymptotically optimal but it has a large number of nodes. We simplify the tree such that the tree has less number of nodes while it keeps track of all samples in collision with obstacles by following three steps:

- 1) *Post Processing Rewiring* (PPR, Algorithm 3): similarly to  $\theta^*$  [18], we examine each node starting from the root and using a breadth-first order, and use the function `isEdgeCollision` to check if it can be connected to an ancestor (testing from the parent and then moving toward the root) without collisions and while lowering the path length.
- 2) *Remove Crossing* (RC, Algorithm 4), if edge  $(i, j)$  crosses edge  $(p, q)$  with an intersection at point  $k$ , we add point  $k$  to  $\mathcal{V}$ , and edges  $(i, k)$  and  $(p, k)$  to  $\mathcal{E}$ . Then, we compare the costs of reaching the start point from  $k$  through edges  $(k, j)$  and  $(k, q)$ , and add the smallest one to  $\mathcal{E}$  as the parent of node  $k$ .
- 3) *Cutting the Leaves* (Ctl, Algorithm 5), for a node that has multiple leaves as children, we only keep a single leaf in the middle.
- 4) We repeat steps 1)-3) until there are no changes in  $\mathcal{T}$ .

Fig. 1 shows an example of the procedure, starting from the RRT\* tree (Fig. 1a), and ending with the simplified tree after the three steps (Fig. 1b).

Note that as a consequence of the simplifying steps above, it is possible to connect each sample from the original RRT\*

---

#### Algorithm 3 Post Processing Rewiring

---

```

1: Input ( $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ )
2: for  $i = 1, \dots, |\mathcal{V}|$  do
3:   while  $\exists \text{ Parent}(node_i) \wedge \exists \text{ Parent}(\text{Parent}(node_i)) \wedge$ 
      IsEdgeCollision(node_i, Parent(Parent(node_i))) do
4:      $\mathcal{E} \leftarrow \mathcal{E} - \{\text{node}_i, \text{Parent}(\text{node}_i)\} \cup \text{node}_i, \text{Parent}(\text{Parent}(\text{node}_i))$ 
5:   end while
6: end for
7: return  $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ 

```

---

#### Algorithm 4 Remove Crossing

---

```

1: Input ( $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ )
2: for for every  $(i, j) \in \mathcal{E}$  do
3:   for for every  $(p, q) \in \mathcal{E}$  do
4:     if edge  $(i, j)$  and  $(p, q)$  have an intersection then
5:        $k = \text{intersection } (i, j) \text{ and } (p, q)$ 
6:        $\text{Parent}(i) \leftarrow k$ 
7:        $\text{Parent}(p) \leftarrow k$ 
8:        $\mathcal{E} \leftarrow \mathcal{E} - \{(i, j), (p, q)\}$ 
9:        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(i, k), (p, k)\}$ 
10:       $\text{cost}_j \leftarrow \text{reach start point through node } j$ 
11:       $\text{cost}_q \leftarrow \text{reach start point through node } q$ 
12:      if  $\text{cost}_j \leq \text{cost}_q$  then
13:         $\mathcal{E} \leftarrow \mathcal{E} \cup \{(k, j)\}$ 
14:      end if
15:      if  $\text{cost}_q \leq \text{cost}_j$  then
16:         $\mathcal{E} \leftarrow \mathcal{E} \cup \{(k, q)\}$ 
17:      end if
18:    end if
19:  end for
20: end for
21: return  $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ 

```

---

to the simplified tree with a straight line, which suggests that the simplified tree will be a good roadmap representation [19] of the free configuration space reachable from the root.

#### B. Environment Constraints

For each edge  $(i, j) \in E$  in the tree, we define a cell  $\mathcal{X}_{ij}$  as

$$\mathcal{X}_{ij} = \mathcal{X}_{V_i} \cap \mathcal{X}_{P_i} \cap \mathcal{X}_{\text{env}}, \quad (8)$$

where

$$\mathcal{X}_{V_{ij}} = \{x | (x - x_i)^T \frac{x_k - x_i}{\|x_k - x_i\|} \leq \frac{\|x_k - x_i\|}{2}, k \in \mathcal{V} \setminus \{j\}\} \quad (9a)$$

$$\mathcal{X}_{P_{ij}} = \{x | (x - x_i)^T \frac{x_j - x_i}{\|x_j - x_i\|} \leq \|x_j - x_i\|, i, j \in \mathcal{V}\}, \quad (9b)$$

and  $\|x_k - x_i\|$  and  $x \in \mathbb{R}^n$  is the Euclidean distance between nodes  $i, k \in \mathcal{V}$ ; the polyhedron  $\mathcal{X}_{ij}$  is similar to a Voronoi region [20], and is defined by a set of perpendicular bisector of segments  $i, k$  for  $k \in \mathcal{V} \setminus \{j\}$ , and by the line perpendicular to  $i, j$  passing through  $j$ . The inequalities in (9a)-(9b) can be written in matrix form of (2). Note that  $\mathcal{X}_{ij}$  contains all the points that are closest to  $i$  than other vertices in  $\mathcal{T}$ , but it also includes the parent  $j$ ; we empirically noticed that with the latter modification we obtained more robust results. An example of  $\mathcal{X}_{ij}$  is shown in Fig. 2

#### C. Stability by CLF

To stabilize the navigation along an edges of a tree, we define the Lyapunov function  $V_{ij}(x)$  as

$$V_{ij}(x) = z_{ij}^T(x - x_j), \quad (10)$$

**Algorithm 5** Cutting the Leaves

```

1: Input  $(T = (V, E))$ 
2: for  $i = 1, \dots, |V|$  do
3:   for leaves  $j$  connected to  $i$  do
4:      $\text{angles}_i \leftarrow \text{angles}_i \cup \text{Angle} = (i, j)$ 
5:   end for
6:    $j_{\text{largest}} \leftarrow \text{LargestAngle}(\text{angles}_i)$ 
7:    $j_{\text{smallest}} \leftarrow \text{SmallestAngle}(\text{angles}_i)$ 
8:    $E_s \leftarrow E_s \cup \{i, j_{\text{largest}}\} \cup \{i, j_{\text{smallest}}\}$ 
9:    $V_s \leftarrow V_s \cup \{j_{\text{largest}}\} \cup \{j_{\text{smallest}}\}$ 
10: end for
11: return  $T_s = (V_s, E_s)$ 

```

where  $z_{ij} \in \mathbb{R}^n$  is the *exit direction* for edge  $(i, j)$ ,  $x_j \in \text{exit face}$  is the position of the parent of node  $i$ , and  $V_{ij}(x)$  reaches its minimum  $V(x) = 0$  at  $x_j$ . Note that the Lyapunov function represents, up to a constant, the distance  $d(x, x_j)$  between the current system position and the exit face. By Definition 2,  $V_{ij}(x)$  is a CLF.

**Definition 4:** For  $\mathcal{X}_{ij}$ , we define the exit direction as  $z_{ij} = \frac{x_j - x_i}{\|x_j - x_i\|}$  that is a unit vector from node  $j$  towards node  $i$  where  $j \in \mathcal{V}_s$  is the parent of node  $i$ .

#### D. Safety by CBF

In this section, we define barrier functions  $h_{ij}(x)$  that defines a cone representing a local convex approximation of the free space between  $i$  and  $j$ , in the sense that it excludes all samples in  $\mathcal{V}$  that are on the way from  $i$  to  $j$ . In particular, we use the following steps (we consider only the 2-D case, although similar ideas could be applied to the 3-D case):

- 1) Define set  $\mathcal{O}_{ij} \subset \mathcal{V}_{\text{collision}}$  whose projection falls on the segment  $i, j$ , i.e.,

$$\mathcal{O}_{ij} = \{o \in \mathcal{V}_{\text{collision}} \mid 0 \leq \text{proj}_{ij}(o) \leq 1\} \quad (11)$$

where

$$\text{proj}_{ij}(o) = \frac{(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_o)}{\|\mathbf{x}_i - \mathbf{x}_o\|} \quad (12)$$

is the scalar projection of the vector  $o$  onto the segment  $i, j$ .

- 2) From the set  $\mathcal{O}_{ij}$ , we choose two samples such that

$$\begin{aligned} o_u &= \min_{o \in \mathcal{O}_{ij}} \{\mathbf{x}_o - \text{proj}_{ij}^{io} \mid \theta_{io} > 0\} \\ o_d &= \min_{o \in \mathcal{O}_{ij}} \{\mathbf{x}_o - \text{proj}_{ij}^{io} \mid \theta_{io} < 0\} \end{aligned} \quad (13)$$

where  $\theta_{io} = \angle(i, j, o)$  is the oriented angle between edge  $(i, j)$  and line  $(x_o, x_i)$ .

- 3) We write the equations of two lines passing through  $\{i, o_u\}$  and  $\{i, o_d\}$  in a matrix form using  $A_{hi} \in \mathbb{R}^{2 \times n}$ ,  $b_h \in \mathbb{R}^2$  to define the invariant set

$$\mathcal{C}_{ij} = \{x \in \mathbb{R}^2 : A_{hi}x + b_{hi} > 0\}. \quad (14)$$

The corresponding CBF is then defined as

$$h_{ij}(x) = A_{hi}x + b_{hi}. \quad (15)$$

An example of the set  $\mathcal{C}_{ij}$  is shown in Fig. 2. Note that the region  $\mathcal{C}_{ij}$  might not include the entire cell  $\mathcal{X}_{ij}$ . However, the controller will be designed to satisfy the CBF and CLF constraints over the entire cell  $\mathcal{X}_{ij}$ ; in practice, this means

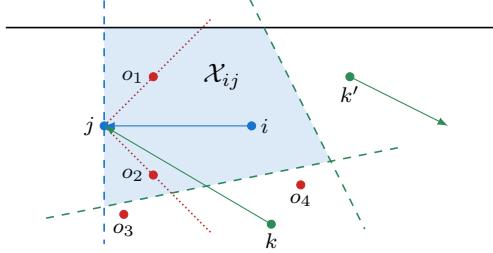


Fig. 2: The Voronoi-like region  $\mathcal{X}_{ij}$  for edge  $(i, j)$ , and the corresponding CBF constraints; green points and arrows: other vertices and edges in the tree  $\mathcal{T}$ ; red points: collision samples in  $\mathcal{V}_s$ ; black line:  $\mathcal{X}_{\text{env}}$ ; green dashed lines:  $\mathcal{X}_{V_{ij}}$  and  $\mathcal{X}_{P_{ij}}$ ; shaded region:  $\mathcal{X}_{ij}$ ; red dotted lines: CBF constraints  $h_i$ . For the CBF constraints, note that the projection of vector  $(i, o_4)$  onto edge  $(i, j)$  does not lie between node  $i$  and node  $j$ , hence  $o_4$  does not support a constraint; moreover, from (13),  $o_d = o_1$  and  $o_u = o_2$  are the closest points (in angle) to the edge  $(i, j)$  and are on two sides of the edge  $(i, j)$ .

that if the robot starts in the region  $\mathcal{X}_{ij} \setminus \mathcal{C}_{ij}$ , it will be driven toward the boundary of  $\mathcal{C}_{ij}$  (this is a consequence of the CBF constraint, and can be proved in a similar way as the original result [11]).

#### E. Controller

We assume that the robot can only measure the relative displacements between the robot's position  $x$  and the landmarks in the environment, which corresponds to the output function

$$y = (L - x\mathbf{1}^T)^\vee = L^\vee - \mathcal{I}x = \text{stack}(l_i - x), \quad (16)$$

where  $L \in \mathbb{R}^{n \times n_l}$  is a matrix of landmark locations,  $i = 1, \dots, n_l$  that  $n_l$  is the number of landmarks,  $A^\vee$  represents the vectorized version of a matrix  $A$ ,  $\mathcal{I} = \mathbf{1}_{nl} \otimes I_n$ , and  $\otimes$  is the Kronecker product. Our goal is to find a feedback controller of the form

$$u_{ij}(K) = K_{ij}y, \quad (17)$$

where  $K_{ij} \in \mathbb{R}^{m \times nn_l}$  are the feedback gains that need to be found for each cell  $\mathcal{X}_{ij}$ . Intuitively, a controller of the form (17) corresponds to a control command that is a weighted linear combination of the measured displacements  $y$ . The goal is to design  $u(y)$  such that the system is driven toward the exit direction  $z_{ij}$  while avoiding obstacles. Note that, to define a controller for edge  $(i, j)$ , the landmarks do not necessarily need to belong to  $\mathcal{X}_{ij}$ , and, in general, each cell could use a different set of landmarks (we explore this direction further in Section III-F).

Following the approach of [14], and using the CLF-CBF constraints reviewed in Section II, we encode our goal in the following feasibility problem:

find  $K_{ij}$   
subject to :  
 $\{\text{CBF: } -(\mathcal{L}_{A_i} h_{ij}(x) + \mathcal{L}_B h_{ij}(x)u + c_h^T h_{ij}(x))\} \leq 0,$   
 $\{\text{CLF: } \mathcal{L}_{A_i} V_{ij}(x) + \mathcal{L}_B V_{ij}(x)u + c_v^T V_{ij}(x)\} \leq 0,$   
 $u \in \mathcal{U}, \forall x \in \mathcal{X}_{ij}, (i, j) \in \mathcal{E}.$

(18)

In practice, we aim to find a controller that satisfies the constraints in (18) with some margin. Note that the constraints in (18) need to be satisfied for all  $x$  in the region  $\mathcal{X}_{ij}$ , i.e., the same control gains should satisfy the CLF-CBF constraints at every point in the region. We handle this type of constraint by rewriting (18) using a min-max formulation [14], arriving to the following robust optimization problem:

$$\begin{aligned} & \min_{K_{ij}, S_{V_{ij}}, S_{h_{ij}}} w_{h_{ij}}^T S_{h_{ij}} + w_{V_{ij}}^T S_{V_{ij}} \\ & \text{subject to} \\ & \max_x \text{CBF} \leq S_{h_{ij}}, \\ & \max_x \text{CLF} \leq S_{V_{ij}}, \\ & S_{V_{ij}}, S_{h_{ij}} \leq 0, \quad u \in \mathcal{U}, \quad \forall x \in \mathcal{X}_{ij}, (i, j) \in \mathcal{E}. \end{aligned} \quad (19)$$

From (4), the Lie derivatives of  $h_{ij}(x)$  and  $V_{ij}(x)$  are written as:

$$\begin{aligned} \dot{h}_{ij}(x) &= A_{hij}\dot{x} = A_{hij}(Ax + Bu), \\ \dot{V}_{ij}(x) &= z_{ij}^T \dot{x} = z_{ij}^T(Ax + Bu). \end{aligned} \quad (20)$$

Combining (16), (17), and (20) with (1), the constraints in (19) can be rewritten as:

CBF constraint:

$$\begin{aligned} & \left[ \max_x - (A_{hij}A - A_{hij}BK_{ij}\mathcal{I} + c_h A_{hij})x \right] \leq \\ & \text{subject to } A_{xij}x \leq b_{xij} \\ & \quad S_{h_{ij}} + c_b b_{hij} + A_{hij}BK_{ij}L_{ij}^\vee, \end{aligned} \quad (21)$$

CLF constraint:

$$\begin{aligned} & \left[ \max_x (z_{ij}^T A - z_{ij}^T BK_{ij}\mathcal{I} + c_v z_{ij}^T)x \right] \leq \\ & \text{subject to } A_{xij}x \leq b_{xij} \\ & \quad S_{V_{ij}} + c_v z_{ij}^T x_j - z_{ij}^T BK_{ij}L_{ij}^\vee, \end{aligned} \quad (22)$$

Constraints in (21), (22) are linear in terms of variable  $x$ , so we can write dual forms of the constraints as

CBF dual constraint:

$$\begin{aligned} & \left[ \begin{array}{l} \min_{\lambda_{bij}} \lambda_{bij}^T b_{xij} \\ \text{subject to} \\ A_{xij}^T \lambda_{bij} = (-A_{hij}A + A_{hij}BK_{ij}\mathcal{I} - c_h A_{hij})^T \\ \lambda_{bij} \geq 0, \\ S_{h_{ij}} + c_b b_{hij} + A_{hij}BK_{ij}L_{ij}^\vee, \end{array} \right] \leq \\ & \quad S_{V_{ij}} + c_v z_{ij}^T x_j - z_{ij}^T BK_{ij}L_{ij}^\vee, \end{aligned} \quad (23)$$

CLF dual constraint:

$$\begin{bmatrix} \min_{\lambda_{lij}} \lambda_{lij}^T b_{xij} \\ \text{subject to} \\ A_{xij}^T \lambda_{lij} = (z_{ij}^T A - z_{ij}^T BK_{ij}\mathcal{I} + c_v z_{ij}^T)^T \\ \lambda_{lij} \geq 0 \\ S_{V_{ij}} + c_v z_{ij}^T x_j - z_{ij}^T BK_{ij}L_{ij}^\vee, \end{bmatrix} \leq$$

Consequently, (19) with the dual constraints becomes:

$$\begin{aligned} & \min_{K, S_V, S_h} w_h^T S_h + w_V^T S_V \\ & \text{subject to :} \\ & \text{CBF dual constraint,} \\ & \text{CLF dual constraint,} \\ & S_h, S_V \leq 0, \end{aligned} \quad (25)$$

From [14, Lemma 1], the optimization problem (19) is equivalent to (25).

Starting from a point  $x \in \mathcal{X}_{ij}$ ,  $u_{ij}$  drives the robot toward  $x_j$ , the robot switches its controller to  $u_{jq}$  when  $\|x - x_j\| \leq \epsilon$  where node  $q$  is the parent of node  $j$ .

#### F. Control With the Limited Field of View

In the formulation above and in the work of [14], it is implicitly assumed that the controller has access to all the landmarks measurements at all times. However, in practice, a robot will only be able to detect a subset of the landmarks due to a limited field of view or environment occlusions. To tackle this issue, we show in this section that the controller  $u$  (17) can be designed using multiple landmarks (as in the preceding section), but then computed using a single landmark.

*Proposition 1:* Let  $K = [K_1, \dots, K_i, \dots, K_l]$  be a partition of the controller matrix conformal with  $L^\vee$ . Given an arbitrary landmark  $\hat{l}_i$  (column of  $L$ ), the controller (17) can be equivalently written as

$$u = \sum_j K_j y_i + k_{\text{bias},i} \quad (26)$$

where  $k_{\text{bias},i} \in \mathbb{R}^n$  is a constant vector given by

$$k_{\text{bias},i} = \sum_{j \neq i} K_j (\hat{l}_j - \hat{l}_i) \quad (27)$$

*Proof:* Using the conformal partition of  $K$ , we can expand (17) as

$$u = \sum_j K_j (\hat{l}_j - x) \quad (28)$$

Adding and subtracting  $\sum_j K_j y_i$  and reordering, we have

$$u = \sum_j K_j (\hat{l}_i - x) + \sum_j K_j (\hat{l}_i - \hat{l}_j), \quad (29)$$

from which the claim follows. ■

Using the fact that the global positions of the landmarks are known during planning, our new Proposition 1 shows that it is possible to implement the controller  $u$  by measuring a single displacement  $y_i$ ; moreover, since the original controller (17) is smooth, one can also switch among different landmarks without introducing discontinuities in the control. Although we stated our result for a single landmark, it is possible to prove a similar claim for any subset of landmarks.

## IV. SIMULATION AND EXPERIMENTAL RESULTS

To assess the effectiveness of the proposed algorithm, we run a set of validation using both MATLAB simulations and experiments using ROS on a Create 2 robot by iRobot [21]. While the optimization problem guarantees exponential convergence of the robot to the stabilization point, in these experiments the velocity control input  $u$  has been normalized to achieve constant velocities along the edges of the trees.

### A. MATLAB Simulation

The simulated MATLAB environment is presented in Fig. 1, where the obstacles are represented by blue circles. To generate the RRT\* we set the maximum number of iterations in RRT\* to 1000, and we choose  $\eta = 60$ . The generated tree from RRT\* and its simplified form are shown in Fig. 1a and Fig. 1b respectively. Then, we compute a controller for each edge of the simplified tree as described in Section III-E. Fig. 3 shows the resulting trajectories from four initial positions on two versions of the environment: one with the obstacles identical to the ones used during planning (Fig. 3a), and one with deformed obstacles (Fig. 3b); for the latter, also the landmarks have been modified accordingly. In all cases, the robot reaches the desired goal location by applying the sequence of controllers found during planning. Note that the deformed environment in Fig. 3b is successfully handled without replanning (i.e., by using the original controllers). This shows that our algorithm can be robust to (often very significant) deformations of the environment; however, there are also cases where, without replanning, the designed controllers might fail. Empirically, we noticed that there is a trade-off between obtaining shortest paths (that, by their nature, graze the obstacles) and the robustness of the controller; we plan to study this trade-off in future work.

### B. iRobot Create 2 Experiment

We further tested our algorithm on a Create 2 robot in lab environments that were similar to those used during the simulation. A bird's-eye view of the experimental setup is shown in Fig. 5. The robot is equipped with a calibrated onboard Arducam for Raspberry Pi camera [22], and we use OptiTrack motion capture system with 44 infra-red cameras to

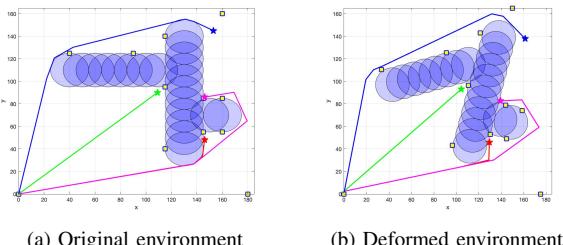


Fig. 3: Simulated trajectories for various start points for the original environment (Fig. 3a) and for a deformed version of the same environment (Fig. 3b). The start points for each trajectory are represented by star markers.

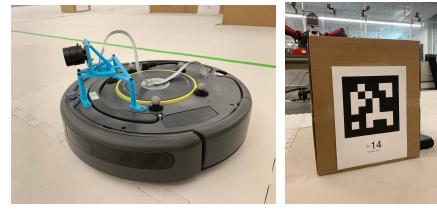


Fig. 4: The Create 2 robot used for the experiments is shown in Fig. 4a. We use AprilTags (Fig. 4b) as the landmarks for the algorithm.

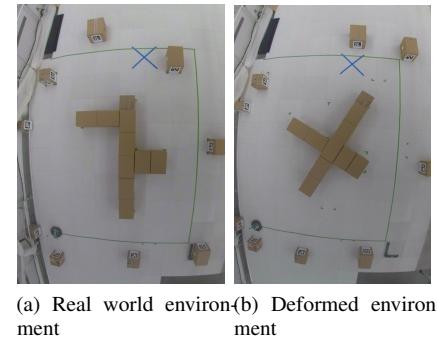


Fig. 5: On the left, Fig. 5a shows the original environment used for our experiment. On the right, Fig. 5b shows the deformed version of the environment.

collect ground truth position information (the motion capture system is not used by our controller). The landmarks are represented by fiducials (AprilTags [23]), and are placed at known positions and orientations with respect to the reference frame of the motion capture, using unique codes for data association. Our implementation is based on the Robot Operating System (ROS, [24]).

There are three practical considerations that need to be taken into account in the implementation. First, due to the limited field of view of the camera, we use Proposition 1 to compute the controller based on one of the fiducials detected

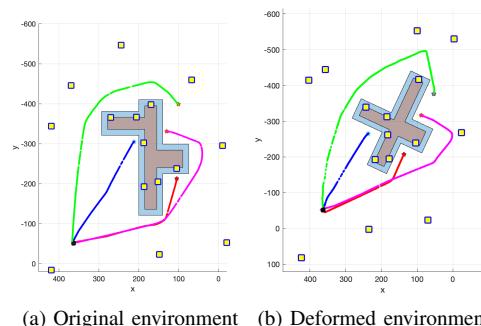


Fig. 6: Real trajectories followed by the Create for both the original and deformed environments. For all the tested start points, the robot converged to the expected goal position.

by the camera at each time instant. Second, the approach presented in the previous sections implicitly assumes that the robot has access to the measurements  $Y$  in a frame which is rotationally aligned with the world reference frame. To satisfy this assumption, the measured displacement of an AprilTag with respect to the Create in world coordinates ( ${}^W t_{AT-C}$ ) is computed as:

$${}^W \mathbf{t}_{AT-C} = {}^W \mathbf{R}_{AT} \cdot ({}^C \mathbf{R}_{AT})^\top \cdot {}^C \mathbf{t}_{AT-C}, \quad (30)$$

where  ${}^C \mathbf{t}_{AT-C}$  is the measured displacement in Create coordinates,  ${}^C \mathbf{R}_{AT}$  is the measured orientation of the AprilTag with respect to the Create, and  ${}^W \mathbf{R}_{AT}$  is the *a priori* known orientation of the AprilTag with respect to the world reference frame. Finally,  $y_i \doteq {}^W \mathbf{t}_{AT-C}$  is used to compute the next control input  $u$  following equation (29).

Finally, previous sections assumed a linear dynamical model for the robot, while the Create 2 has a unicycle dynamics. We map the original 2D input  $u$  to a linear velocity  $u_x$  along the  $x$  axis of the robot and an angular velocity  $\omega_z$  around the  $z$  axis of the robot using a rather standard low-level controller:

$$u_x = \frac{\alpha}{\|u\|} \begin{bmatrix} \cos \varphi \\ \sin \varphi \end{bmatrix}^\top u, \quad \omega_z = \frac{\beta}{\|u\|} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^\top \left( \begin{bmatrix} \cos \varphi \\ \sin \varphi \\ 0 \end{bmatrix} \times \begin{bmatrix} u \\ 0 \\ 0 \end{bmatrix} \right) \quad (31)$$

where  $\varphi$  is the instantaneous yaw rotation of the robot with respect to the world reference frame, which is extracted from  ${}^C \mathbf{R}_{AT}$  and  ${}^W \mathbf{R}_{AT}$ , and  $\times$  represent the 3-D cross product;  $\alpha$  and  $\beta$  are user-defined scalar gains, 0.1 and 0.5 respectively.

Fig. 6 depicts the real robot trajectories. Both in the original and deformed environments, the robot followed the edges of the RRT\* tree and reached the expected goal for all starting positions and with the same control gains, despite the fact that the measurements were obtained with vision alone, and despite the different dynamics of the robot.

## V. CONCLUSIONS AND FUTURE WORKS

In this work, we introduced a new approach to integrate the high-level RRT\* path planning with a low-level controller. We represented the environment via a simplified tree graph by implementing a modified sampling-based RRT\* algorithm. We defined convex cells around the nodes of the tree and formulated a min-max robust Linear Program with CLF and CBF constraints to guarantee the stability and safety of the system. We built a robust output feedback controller for each cell which takes relative displacement measurements between a set landmarks positions and position of the robot as an input. We addressed the limited field of view of the robot issue by reformulating a controller based on the visible landmarks. We validated our approach on both simulation environment and real-world environment and represented the robustness of our algorithm by applying the controller to a significantly deformed environment without replanning. We plan to prove the robustness of our algorithm theoretically and define the conditions of robustness of the controller in our future work.

Furthermore, we plan to study the trade-off between optimal navigation and the robustness of the controller.

## REFERENCES

- [1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.
- [3] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.
- [4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
- [5] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
- [6] V. Renganathan, I. Shames, and T. H. Summers, "Towards integrated perception and motion planning with distributionally robust risk constraints," *arXiv preprint arXiv:2002.02928*, 2020.
- [7] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How, "Motion planning in complex environments using closed-loop prediction," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008, p. 7166.
- [8] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
- [9] A. Weiss, C. Danielson, K. Berntorp, I. Kolmanovsky, and S. Di Cairano, "Motion planning with invariant set trees," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2017, pp. 1625–1630.
- [10] R. Tedrake, "Lqr-trees: Feedback motion planning on sparse randomized trees," *MIT Press*, 2009.
- [11] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 6271–6278.
- [12] S.-C. Hsu, X. Xu, and A. D. Ames, "Control barrier function based quadratic programs with application to bipedal robotic walking," in *2015 American Control Conference (ACC)*. IEEE, 2015, pp. 4542–4548.
- [13] U. Borrman, L. Wang, A. D. Ames, and M. Egerstedt, "Control barrier certificates for safe swarm behavior," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 68–73, 2015.
- [14] M. Bahreinian, E. Aasi, and R. Tron, "Robust planning and control for polygonal environments via linear programming," *2020 IEEE American Control Conference (ACC)*, 2020.
- [15] A. Isidori, *Nonlinear control systems*. Springer Science & Business Media, 1995.
- [16] X. Xu, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Robustness of control barrier functions for safety critical control," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 54–61, 2015.
- [17] Q. Nguyen and K. Sreenath, "Exponential control barrier functions for enforcing high relative-degree safety-critical constraints," in *2016 American Control Conference (ACC)*. IEEE, 2016, pp. 322–328.
- [18] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Theta^\*: Any-angle path planning on grids," in *AAAI*, vol. 7, 2007, pp. 1177–1183.
- [19] H. M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.
- [20] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.
- [21] iRobot, "Create 2 Programmable Robot," <https://edu.irobot.com/what-we-offer/create-robot>.
- [22] Arducam, "Camera for Raspberry Pi," <https://www.arducam.com/docs/cameras-for-raspberry-pi/>.
- [23] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, oct 2016, pp. 4193–4198.
- [24] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: <https://www.ros.org>